SENTIMENT ANALYSIS OF FINANCIAL NEWS FOR PREDICTING STOCK PRICE TRENDS USING NLP TECHNIQUES IN FINTECH

BY

Cherng Jun Kai

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION SYSTEMS (HONOURS) DIGITAL ECONOMY TECHNOLOGY

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2025

COPYRIGHT STATEMENT

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Information Systems (Honours) Digital Economy Technology at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Ms Nurul Syafidah binti Jamil who has given me this bright opportunity to engage in this project. Her expertise in Natural Language Processing, Transformer, and BERT inspired me to push the boundaries of this research, and her encouragement was a constant source of motivation.

Next, I would also like to extend my sincere thanks to the Faculty of Information and Communication Technology at Universiti Tunku Abdul Rahman for providing the resources and environment conducive to innovative research. The access to cutting-edge tools and datasets through the university's infrastructure was instrumental in the success of this project.

My heartfelt gratitude goes to my soulmate, Cheng Puei Ying, whose unwavering support and encouragement were invaluable throughout this journey. Her patience, understanding, and constant belief in my potential provided me with the emotional strength to navigate the challenges of this research, making every milestone more meaningful with her by my side.

Lastly, I owe an immense debt of gratitude to my family for their unconditional love, patience, and belief in my abilities. Their encouragement kept me grounded and focused, even when the complexities of transformer models and financial datasets seemed overwhelming.

This project marks a significant milestone in my academic and professional journey, and it would not have been possible without the collective support of everyone mentioned. Thank you all for being part of this endeavor.

ABSTRACT

The stock market is highly influenced by news and investor sentiment, making trend prediction both challenging and valuable. This project develops a framework for stock price trend prediction by integrating sentiment analysis of financial news with historical market data. News headlines are cleaned and analyzed using VADER, TextBlob, BERT, and FinBERT to generate sentiment scores, which are merged with OHLCV price data and enriched with lagged returns and time-based features. Five machine learning models — Support Vector Machine (SVM), Logistic Regression (LR), Random Forest (RF), XGBoost, and LightGBM — are trained and tuned using walk-forward cross-validation. Their performance is evaluated using accuracy, precision, recall, F1-score, and confusion matrices, with XGBoost achieving the best results. Finally, a Power BI dashboard is built to visualize sentiment trends, market data, and model predictions, making insights interactive and actionable. Results show that incorporating sentiment features improves predictive performance, supporting data-driven decision-making for investors and analysts.

Area of Study: Machine Learning, Natural Language Processing

Keywords: Machine Learning, Natural Language Processing, PowerBI, OHLCV, Sentiment Analysis

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

© Copyright

LIST OF ABBREVIATIONS

BERT	Bidirectional Encoder Representations from Transformers
CNNs	Convolutional Neural Networks
EMH	Efficient Market Hypothesis
FinTech	Financial Technology
GPT	Generative Pre-trained Transformer
GRU	Gated Recurrent Network
LSTMs	Long Short-Term Memory Networks
NLP	Natural Language Processing
RNNs	Recurrent Neural Networks
RoBERTa	Robustly Optimized BERT Approach
SVM	Support Vector Machine
VADER	Valence Aware Dictionary and sEntiment Reasoner
GANs	Generative Adversial Networkds
HTML	Hypertext Markup Language
OHLCV	Open, High, Low, Close, and Volume
XGBoost	eXtreme Gradient Boosting
LightBGM	Light Gradient Boosting Machine
MAPE	Mean Absolute Percentage Error
MAE	Mean Absolute Error

Chapter 1
Introduction

Nowadays, the society is known as the digital age. With the popularity of social media and networking, user data is becoming more accessible. Data has become part of the user's critical asset, and it is often referred to as the "new oil" due to its immense value and transformative potential [1]. Besides that, the stock market trend is one of the most popular topics in financial research. Early studies were based on the **efficient market hypothesis (EMH)** [2]. **EMH** argues that stock market prices are influenced by information such as news, events, and company announcements rather than previous and current stock prices [3]. However, the integration of advanced technologies into the financial sector has revolutionized the way financial analysis and decision-making are conducted today. In this research, **machine learning** algorithms and **NLP** for sentiment analysis will be utilized for predicting stock price trends in FinTech.

Machine learning is commonly used to categorize and predict sentiment as positive or negative [4]. Machine Learning algorithms can be further classified into **unsupervised**, **supervised** and, **semi-supervised**. **Unsupervised learning** is the process of training a model using unlabelled data. The purpose is to determine the underlying structure of the data, such as grouping comparable data points or lowering the data's dimension [5]. The examples of **unsupervised learning** algorithms are **K-means clustering**, **PCA**, and **Hierarchical Clustering**. In **supervised learning**, the algorithm is trained on labelled data which mean each training example is paired with an output label. The goal of supervised learning is to learn the mapping for inputs to outputs and make accurate predictions on new and unseen data [5]. The examples of **supervised learning** algorithms are **linear regression**, **logistic regression**, **decision trees**, and **SVM**. Lastly**, semi-supervised** learning is a blend of supervised and unsupervised learning. It uses a small amount of labelled data along with a large amount of unlabelled data to improve learning accuracy [5]. The examples of **semi-supervised** learning algorithms are **GANs** and **Self-trained Naïve Bayes Classifier**. By analysing vast amounts of historical and real-time data, machine learning models can identify patterns and make predictions that inform investment decisions.

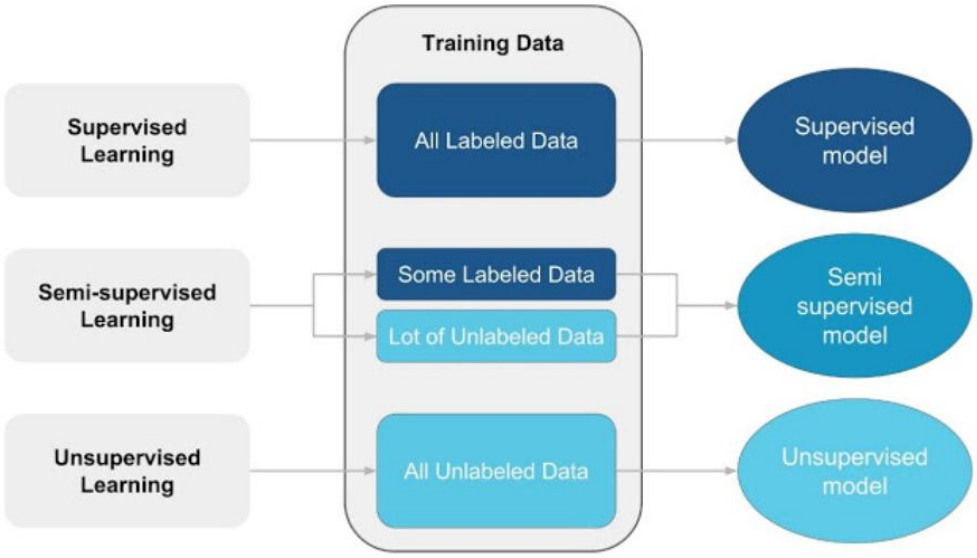


Figure 1.0 Outline of the different machine learning approaches

Nowadays, **NLP** is a subset of artificial intelligence focused on comprehending spoken and written language [6]. **NLP** can leverage various types of model and architectures to understand and generate human language. For instance, RNNs, LSTMs, CNNs, Transformer Models, BERT, GPT, and RoBERTa. It also employs techniques such as sentiment analysis, keyword extraction, and linguistic pattern recognition to capture subtle shifts in market sentiment, track emerging trends, and extract critical insights from vast volumes of textual information.

Despite the potential of **NLP** in predicting stock price trends through sentiment analysis of financial news, several limitations must be addressed to ensure effective application. One major challenge is the complexity and noise in textual data, as financial news often contains ambiguous language, sarcasm, or context-specific jargon that can mislead sentiment analysis models. Additionally, the dynamic nature of financial markets means that sentiment signals may have varying impacts depending on market conditions, making it difficult to establish consistent correlations between news sentiment and stock price movements. Data quality issues, such as incomplete or biased news sources, further complicate accurate predictions.

1.1 Problem Statement and Motivation

The stock market is influenced by a lot of factors. The factors are economic indicators, corporate performance, and market sentiment. Traditional financial analysis often overlooks the subtleties of human language conveyed in financial news and leading to inaccurate predictions of stock price movements [7]. The current approaches of financial modeling and sentiment analysis does not have the ability to fully utilize capabilities of natural language processing (NLP) technology. Therefore, some of the abnormal emotions expressed in financial news article could not be captured.

First and foremost, **inaccurate predictions and model generalization challenges**. Even with well-cleaned data and advanced NLP models, sentiment-based stock trend prediction often suffers from **inaccuracy and poor generalization** when applied to unseen data or different market conditions. Stock prices are influenced by multiple factors beyond news sentiment, such as macroeconomic indicators, geopolitical events, and investor behavior, making price movement inherently stochastic and noisy. Models trained on historical data may be overfit to past patterns and fail to adapt to changing market regimes. For instance, during crises, earnings season, or high volatility periods. Moreover, sentiment signals may produce false positives: a positive news sentiment does not always lead to price increases, as other hidden variables can dominate price action. This can lead to a high rate of wrong predictions, thereby misleading investors.

Besides that, **limitations of traditional sentiment models**. General-purpose sentiment analysis tools like VADER and TextBlob may not accurately capture financial domain-specific sentiment like “beat estimates,” “downgraded,” or “short squeeze”, as they were not trained on financial text. This can lead to misclassification of sentiment and weaken the link between news and price movements.

Furthermore, **noise and ambiguity in financial news data**. Financial news headlines are often short, contain company tickers, and may include slang, abbreviations, or non-standard language. This makes them challenging for traditional NLP methods, which may fail to capture the nuanced sentiment intended by financial journalists or investors. Moreover, noise from duplicate or irrelevant articles can distort sentiment signals, thereby reducing predictive accuracy.

The aim for this project is **improving the accuracy of stock market predictions**. Improving prediction accuracy and model generalization is vital because financial markets are dynamic and highly sensitive to external factors. A model that performs well on historical data but fails to generalize to new, unseen data can produce misleading or even harmful signals when deployed in practice. Inaccurate predictions may cause traders or investors to act on false information, potentially leading to financial losses. Overcoming this problem by using robust modeling strategies, proper validation, and periodic retraining is strongly recommended to maintain predictive reliability even under volatile market conditions. Ensuring better generalization improves confidence in the model's outputs, makes it more resilient to changing market regimes, and supports its long-term applicability in real-world financial forecasting tasks.

Moreover, it aims to overcome the limitations of traditional lexicon-based sentiment models is important because they often fail to capture the subtleties of financial language, leading to frequent misclassifications. In the financial domain, words and phrases can carry meanings that differ significantly from general sentiment usage (e.g., "beat estimates" is strongly positive, while "short interest rising" can be bearish). If these signals are misinterpreted, predictions about price trends may become misleading, resulting in lost opportunities or incorrect trading actions. Using more advanced, domain-specific NLP models such as transformer-based approaches (e.g., FinBERT) helps capture contextual meaning and better reflect the sentiment that drives market behavior. It is therefore recommended to adopt such models to improve sentiment accuracy, enhance predictive power, and ultimately provide more reliable support for investors.

Next, it addresses noise and ambiguity in financial news is crucial because unreliable or inconsistent input data can severely weaken the performance of sentiment analysis models. Noisy data like duplicated headlines, irrelevant articles, or inconsistent ticker formats will introduces bias and reduces the model's ability to capture the true market sentiment. Overcoming this issue ensures that only high-quality, contextually relevant information contributes to sentiment scoring, which improves the signal-to-noise ratio and strengthens the correlation between sentiment and market reactions.

1.2 Objectives

Main objective: To develop a stock price trend prediction model based on sentiment of financial news.

- The primary aim of this project is to design, implement, and evaluate a predictive framework that utilizes NLP to extract sentiment signals from financial news and link them to subsequent stock price movements. By integrating sentiment indicators with historical price data, the system seeks to uncover statistically significant relationships between market mood and price trends, ultimately supporting investors and analysts in making data-driven decisions.

Sub-objectives:

I. To collect and preprocess historical stock price data and news textual data or to perform data preparation.

- It focuses on compiling a high-quality and reliable financial dataset, encompassing textual news data and numerical price data, whilst executing comprehensive preprocessing. The code implementation involves importing CSV files of news and prices, normalizing text (removing URLs, HTML tags, dollar tickers, punctuation), handling missing values, and removing duplicates. Through data preprocessing ensures downstream models are trained on clean, consistent data, thereby minimizing noise and enhancing predictive accuracy.

II. To merge historical stock price data with news sentiment data.

- It matches news articles with corresponding stock price records using stock codes and timestamps. This merging process transforms the data into a structured format, with each row associating news sentiment with the corresponding market reaction, laying the groundwork for supervised machine learning.

III. To build machine learning models for predicting stock price trends using sentiment and price features.

- The objective is to leverage these sentiment scores, together with price-based technical indicators and to predict stock price direction or returns. The sentiment scores are generated by four models which is TextBlob, Vader, BERT, and FinBERT.

IV. To evaluate the performance of developed stock price prediction based on sentiment features.

- The final objective is to assess how well the model predicts price trends. Model performance will be measured using metrics such as accuracy, F1-score, precision and recall. This evaluation ensures that the developed system is both statistically valid and practically useful for decision-making in real-world trading scenarios.

1.3 Project Scope and Direction

This project focuses on developing a stock price trend prediction framework that integrates sentiment analysis of financial news with historical price data to enhance market trend forecasting. The scope covers the complete pipeline of data collection, preprocessing, feature engineering, model development, and evaluation. Financial news articles and headlines are first collected and cleaned to remove noise such as HTML tags, links, and special characters, ensuring that the text is standardized for downstream analysis. NLP techniques are then applied to extract sentiment scores from each piece of news using tools such as TextBlob, VADER, and transformer-based models like BERT and FinBERT. These sentiment scores serve as quantified indicators of market mood, which are crucial for linking textual data to price fluctuations.

The project further incorporates historical stock price data, including open, high, low, close, adjusted close, and trading volume (**OHLCV**) information, which is merged with the sentiment data on a timestamp and ticker-symbol basis. Additional time-series features such as lagged returns, moving averages, and volatility metrics are engineered to capture temporal dependencies and improve model performance. By combining textual sentiment features with numerical price data, the project ensures that both qualitative and quantitative signals are leveraged to capture underlying market dynamics.

The direction of the project emphasizes building a machine learning-based predictive model that is both robust and interpretable. XGBoost, a gradient boosting algorithm, is chosen as the core predictive model due to its strong performance on structured tabular data, ability to handle non-linear feature interactions, and support for regularization to prevent overfitting. The model is trained on chronologically split datasets to avoid data leakage and evaluated using multiple metrics, including accuracy, F1-score, and area under the ROC curve (AUC), ensuring that its predictions are reliable and generalizable.

Finally, to make the results interpretable and visually accessible, the project delivers an interactive Power BI dashboard that showcases sentiment trends, news distributions, and model prediction outputs in an intuitive format. This interface allows users to explore key insights, monitor sentiment shifts across time, and review predicted price movement trends with supporting visualizations.

1.4 Contributions

This project contributes meaningful value to multiple stakeholders in the financial ecosystem by combining sentiment analysis with machine learning to improve market trend prediction. The developed model demonstrates how financial news can be transformed from unstructured text into quantifiable sentiment features, which are then linked to historical price movements to uncover patterns between market mood and price trends. This integration helps bridge the gap between qualitative investor sentiment and quantitative trading strategies by providing a data-driven foundation for decision-making.

This project provides a powerful decision-support tool for investors including both individual traders and institutional participants. By leveraging the model's predictive capabilities, investors can anticipate potential price movements more accurately and adjust their trading strategies accordingly. This supports the goal of minimizing risks, optimizing portfolio allocations, and ultimately improving financial returns. The project's inclusion of multiple sentiment analysis methods, including rule-based (VADER, TextBlob) and transformer-based approaches (BERT and FinBERT) to enhance the reliability of sentiment signals, which is crucial for avoiding misinterpretation of financial language and market context.

From a technological perspective, the project contributes a robust machine learning pipeline that combines structured price data with unstructured textual data, preprocessed and engineered into a unified dataset. The use of XGBoost enables efficient handling of non-linear feature interactions and provides explainability through feature importance analysis, giving analysts and researchers insights into which sentiment or price features most strongly influence predictions. This not only advances the practical application of NLP in finance but also provides a reproducible framework that other researchers can extend with additional data sources or more advanced models.

1.5 Report Organization

This report is organized into seven chapters to present the research in a structured and logical manner. **Chapter 1** introduces the project by outlining the background, problem statement, research motivation, objectives, scope, and contributions. **Chapter 2** presents a comprehensive literature review on sentiment analysis, natural language processing (NLP) techniques, and stock market prediction methods, identifying research gaps that this project addresses. **Chapter 3** explains the research methodology, detailing the phases of data merging, cleaning, visualization, sentiment extraction, feature engineering, preprocessing, model training, hyperparameter tuning, and evaluation. **Chapter 4** discusses the system design, including the architecture of the proposed framework, data flow diagrams, and process models that illustrate the system's structure and functionality. **Chapter 5** focuses on the system development and implementation, describing the coding process, libraries used, machine learning model setup, and integration of XGBoost as the final predictive model. **Chapter 6** presents the experimental results and discussion, including performance evaluation of the five models (SVM, LR, RF, XGBoost, LightGBM), comparison of results, and interpretation of findings. Finally, **Chapter 7** concludes the project by summarizing the key outcomes, discussing limitations, and recommending directions for future work.

Chapter 2
Literature Review

2.1 Predicting Stock Market using Natural Language Processing

Literature review on stock market prediction has grown with time especially with the availability of textual data and the development of computational intelligence. This review compiles common findings from different papers that have investigated the relationship between NLP and financial forecasting. Traditional approaches often relied on technical analysis, which involves identifying patterns in historical price movements to forecast future trends [9].

In the exploration of predicting stock market using NLP, there are various of methodologies are applied to enhance the prediction accuracy. The study emphasizes the integration of sentiment analysis derived from news headlines with historical price data to inform predictions. By analyzing the sentiment conveyed in news articles, the research aims to capture the emotional tone and market sentiment that can influence stock prices [8]. This integration enables the model to incorporate the historical prices and the current market sentiment captured by the news articles hence making it easy to understand price variation. This approach aims at adding more accuracy to the model by considering insights of the textual data in addition to numerical data insights.

The integration of NLP into financial forecasting represents a significant advancement in the field. Recent studies have focused on extracting sentiment from news articles and social media to inform stock price predictions. [8] had highlighted the potential of BERT-based architectures in sentiment analysis, noting that these models can effectively capture the abnormally of financial language. The use of sentiment analysis allows researchers to quantify the emotional tone of news headlines and providing valuable insights into market sentiment.

2.1.1 Methods Applied

The research employs advanced models that leverage sentiment analysis to correlate news headlines with stock price movements. Specifically, the study utilizes a **GRU** model, which is a type of **RNN** designed to handle sequential data effectively. The GRU model is quite useful in time-series prediction tasks or in any other tasks which contain temporal dependency.

In this approach, sentiment scores are derived from news headlines using a pre-trained NLP model, FinBERT, which is specifically designed for financial text analysis. From the headlines, FinBERT is able to identify whether it has a positive, negative, or neutral sentiment to which it assigns a sentiment score of. These sentiment scores are then merged with the historical data of stock price which basically contains the previous day closing of the company's stock and other related market factors.

The integration of these two data sources makes the model richer in terms of its understanding of the market. The sentiment analysis provides context to the numerical data, allowing the model to account for external factors that may influence investor behavior, such as market sentiment, news events, and economic indicators. This dual framework not only increases forecasting precision but also reveals information about the financial motivations behind investors behaviours which might help understand why particular shares can increase or decrease in response to events.

In addition, the study uses other procedures commonly used in data validation like cross-validation apart from using **performance measures** such as **RMSE** to check the viability of the GRU model to forecast stock price fluctuations. By comparing the results with traditional time-series models, the research demonstrates the added value of incorporating sentiment analysis into stock market predictions. Ultimately contributing to a more comprehensive understanding of the factors influencing stock prices.

2.1.2 Challenges and Limitations in Predicting Stock Market using NLP

Despite the promising results achieved through the integration of NLP in stock market predictions. However, there are several challenges and limitations that can impact the effectiveness and reliability of the models. For instance, **variability in quality and relevance of news headlines**, **limited dataset and scope**, **temporal lag and reaction time**, and **overfitting and generalization**.

The primary challenges are the **variability in the quality and relevance of news headlines**. As not all the news headlines provide meaningful insights into stock price movements. Many headlines might be misleading, imprecise and sometimes irrelevant to the key concerns that revolve around corporate revenues or outlays. This inconsistency can give wrong sentiment scores in the actual market sentiment analysis process leading to a wrong sentiment result. For instance, some headlines may be addressing broad market trends or other unrelated events and weaken the effect of more appropriate news. As a result, the outcome of the model be affected. In addition, the study's reliance on a **limited dataset** specifically focuses on the top 20 news headlines scraped daily from a **single source**. This narrow focus may not capture the full spectrum of market sentiment as it excludes other potential influential news sources such as financial reports, press releases, and social media feedback. Given the constantly evolving dynamics of financial markets, it is now possible to be influenced by numerous factors. Thus, the perception of the market based solely on a limited set of headlines may be erroneous. Consequently, the model might miss some of the critical information that could affect the stock price movements and resulting in less accurate predictions.

Apart from that, **temporal lag** associated with news dissemination and market reaction is also part of the challenges. Whenever the news is published, there is always a latency between the market's reaction. This latency can vary depending on the nature of the news and the market's overall sentiment at the time. As a result, models that rely on real-time sentiment analysis may struggle to accurately predict stock price movements if they do not account for this delay. Lastly, the risk of **overfitting** is another concern when developing an NLP model. When a model is training too closely on a specific dataset, it may perform well on that data. However, it will perform badly on the new or unseen data. This is even more appropriate when it comes to stock market forecasting where the market conditions can change time to time.

Those models that do not consider this spread and volatility may lose efficiency in the long run and thus give out highly inaccurate predictions.

2.1.3 Proposed Solution

This paper does suggest some approaches and methodologies that could help mitigate the challenges above. Let start with **improving data quality**. While the paper acknowledges the variability in the quality of new headlines, it implies that using more diverse and comprehensive datasets could enhance the predictive power of the models. This could involve incorporating additional news sources or types of data beyond just the top 20 headlines. Next, the researcher on this paper suggests **expanding the datasets** by including more headlines and different sources. Getting a broader dataset could provide a more accurate representation of market sentiment and overcome the limited datasets issue.

Although the paper does not offer a definite solution for temporal lags, but it recommends that **combining time series analysis** with sentiment score may enhance the prognostications. This indicates a potential avenue for further research and model development. Lastly, this paper suggests **implementing robust model validation** to overcome the overfitting problem. To mitigate the risk of overfitting, it is crucial to adopt rigorous validation methods that ensure models generalize well to unseen data. Techniques such as cross-validation and testing on diverse datasets can help ensure the reliability and robustness of predictive models.

2.2 Deep Learning based stock price prediction using News Sentiment Analysis

Stock market price prediction is a complex task as it is not based on any mathematical formulation. Yet, the market price might change based on politics, rate cut or even CPI. However, we can make fairly accurate short-term predictions by learning the patterns in data. Machine Learning approaches have been employed to tackle the issue of short-term prediction and people have had success in predicting fairly accurate in Figure 2.0 [10]. The paper investigates the use of deep learning techniques to predict stock prices by integrating news sentiment with historical price data. It explores three models which are MLP, LSTM network, and the proposed FinBERT-LSTM model. FinBERT, a pre-trained NLP model fine-tuned for financial sentiment classification, is used to extract positive, negative, and neutral sentiment scores from news articles. These sentiment scores are then combined with stock price data from NASDAQ-100 constituents. The study uses a 10-day rolling window to capture temporal dependencies and predict the stock price of the following day. Model performance is evaluated using MAE, MAPE, and accuracy. The results show that FinBERT-LSTM outperforms both MLP and vanilla LSTM in prediction accuracy, demonstrating that incorporating sentiment features improves model performance.

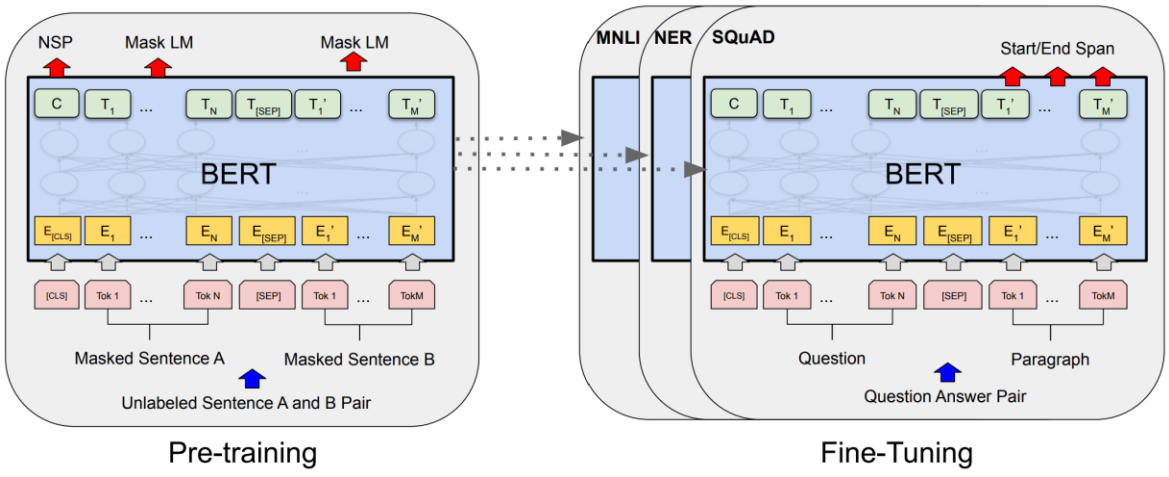


Figure 2.0 BERT Pre-Train & Fine Tune

2.2.1 Methods Applied

1. Data Collection & Preprocessing:

- Stock data: Daily closing prices of NASDAQ-100 index stocks between 01/10/2020 and 30/09/2022, collected using Yahoo Finance API.
- News data: Up to 10 daily finance, business, and technology news articles from The New York Times using its API.
- Sentiment for each day's news was computed using FinBERT, producing positivity, negativity, and neutrality scores.
- Min-max normalization was applied to stock prices for consistent scaling.
- A 10-day rolling window was used, where the model receives the past 10 days' prices (and sentiment) to predict the 11th day's price.

2. Model Architectures:

- MLP: Fully connected feed-forward neural network with 5 layers (1 input, 3 hidden, 1 output) and ReLU activations. Dropout layers with rates 0.1, 0.05, and 0.01 were used to prevent overfitting.
- LSTM: A 5-layer network with 3 stacked LSTM layers, capable of capturing sequential dependencies in price data.
- FinBERT-LSTM: The final proposed model combining 10 days of closing prices + daily sentiment score (11 features total). Three LSTM layers with "tanh" activations were used, with no dropout as no overfitting was observed.

3. Training & Evaluation:

- Loss Function: Mean Squared Error (MSE).
- Optimizer: Adam with learning rates 0.01 (MLP) and 0.02 (LSTM/FinBERT-LSTM).
- Data Split: 85% training and 15% testing (chronologically).
- Metrics: MAE, MAPE, and Accuracy were used to compare model performance.

2.2.2 Challenges and Limitations

1) **Volatility and Non-linearity of Stock Prices:** Stock prices are highly volatile and influenced by multiple factors beyond news (macroeconomic events, investor psychology, global risks), making modeling inherently difficult.

2) **Limited News Coverage:** Only New York Times articles were used, which may not cover all impactful market events or provide sufficient daily news volume for smaller companies.

3) **Short Time Window:** The model used a 10-day rolling window, which may miss longer-term market trends or delayed market reactions to news.

4) **Model Overfitting Risk:** Although dropout was used in MLP and LSTM, deep networks still risk overfitting, especially with a relatively small dataset.

5) **Single Market Focus:** The model was only trained on NASDAQ-100 data, limiting generalizability to other indices, sectors, or international markets.

2.2.3 Proposed Solution

The study's key contribution lies in its **integration of sentiment data** with stock price prediction. By incorporating FinBERT-derived sentiment scores, the model was able to better interpret market context and significantly reduce prediction error compared to models trained solely on historical price data. Furthermore, **the use of a 10-day sliding window** enabled the model to capture short-term sequential dependencies, which improved the stability and reliability of its predictions.

Moreover, by **benchmarking three different architectures** which is MLP, LSTM, and the proposed FinBERT-LSTM, the authors clearly demonstrated the incremental benefits of combining sentiment analysis with deep learning methods. The authors also compared all three models in Table 2.0 to choose a more reliable framework for forecasting stock price trends.

Models	**MAE**	**MAPE**	**Accuracy**
MLP	218.32973474	0.01767204122	0.98232795877
LSTM	180.58083886	0.01456811176	0.98543188823
FinBERT-LSTM	174.94284259	0.01409574846	0.98590425153

Table 2.0 Models' Accuracy

Besides that, the researchers **fine-tuned model hyperparameters** and conducted 1000 training trials, thereby minimizing randomness and improving the reliability of their findings. Finally, **dropout regularization** was applied to the MLP and LSTM models to address potential overfitting issues, although it was deemed unnecessary for the FinBERT-LSTM model due to its strong generalization performance on validation data.

2.3 Transformer – Bert sentiment

Sentiment Analysis also known as opinion mining, is a crucial task in NLP that aims to extract subjective information from text such as opinions, attitudes, and emotions [11]. According to [11], the progress of sentiment analysis has been transformed by improved computational methods as well as detailed data collected from online sources. -In recent years, sentiment analysis has gained significant attention especially in the context of financial markets where public sentiment can influence stock prices [12]. Traditional methods of sentiment analysis often rely on lexicon-based approaches [12], simpler machine learning models [13], numerical data or statistical analysis [14] that might not identify the abnormal expressions found in social media texts. Deep learning has emerged as a new approach in the recent times, especially models such as Bidirectional Encoder Representations from Transformers (BERT) that bring more abnormal understandings of semantics into the enterprise and have made the sentiment classification work much more precise.

Moreover, other basic classification techniques such as Naive Bayes or Support Vector Machine are quite hard to generalise well when dealing with diverse datasets. Most of these models need feature engineering of the data in some way and they often do not capture all the abnormally inherent in language use such as implicit meanings, metaphors, sarcasm or context-dependent meanings. As a result, traditional approaches of sentiment classification may not be very effective and there could be misleading perceptions about the sentiment of the public.

Deep learning was a revolution in the field of NLP especially with the formulation of new architectures such as Bidirectional Encoder Representations from Transformers (BERT). It is critical to understand that BERT is a major breakthrough in terms of understanding language because of its depth of bidirectional training that permits all words to be contextualized and improved with the help of each other. This capability enables BERT to generate powerful semantic representations that capture the intricacies of language more effectively than previous models. By pre-training on vast amounts of text data and then fine-tuning on specific tasks, BERT has demonstrated state-of-the-art performance in various NLP applications including sentiment analysis.

2.3.1 Methods Applied

According to [14], the papers mentioned that the models rely on predefined **sentiment lexicons** to assess the sentiment of text. For example, **Sent WordNet** is one of the most well-known lexicons that associates the words with sentiment scores. While these models are straightforward and require minimal training data and the models often struggle with context and sarcasm causing to inaccurate sentiment classification.

In addition, the study leverages the BERT model, which is pre-trained on a large corpus of text and fine-tuned on a specific dataset of stock-related messages from Stocktwits [12]. The authors collected a substantial dataset comprising over 16 million messages, filtering out irrelevant content to focus on 14.4 million messages of which 2.9 million were labeled as bullish and 825,191 as bearish. The methodology involved several key steps which is **Data Pre-Processing**, **Tokenization and Input Preparation**, and **Fine-Tuning the BERT Model**.

In order to address the imbalance in the dataset, all available bearish data has been utilised and randomly selected 1.3 times the bearish data from the bullish messages [12]. This approach made the training set to be more balanced which will be helpful for effective model training. In addition, the text data was **tokenized** using the Transformers tokenizer from the Pytorch library. The texts were then trimmed and padded to have the maximum length of 64 tokens. The token limit was decided based on the distribution of length of strings in the training data. This process is essential for preparing the data before feeding into BERT model. Lastly, the authors employed the **BERTForSequenceClassification** model, adjusting hyperparameters such as learning rate, batch size, and the number of training epochs to optimize performance. A low learning rate of 1-5 was chosen to mitigate the risk of catastrophic forgetting during **fine-tuning**.

Based on [13] [15], the authors presented a detailed discussion of sentiment analysis with deep learning techniques moving from the traditional methods to recurrent neural networks (RNNs), Long Short-Term Memory (LSTM) and gated Recurrent Units (GRU). These models improved the ability to capture temporal dependencies in text data. Furthermore, the introduction of **attention mechanisms** has enabled the models to concentrate on specific parts of the input text [16]. Consequently, the performance in sentiment classification tasks will be enhanced. Bidirectional Encoder Representations from Transformers (BERT), a technique that

has outperformed many previous models in the NLP tasks has been applied [13]. The main reason is due to BERT's architecture allows it to perform the context from both directions. This function making it very effective for sentiment analysis where understanding the full context of a statement is crucial.

2.3.2 Challenges and Limitations

Despite getting the promising results after using BERT, the study acknowledges several challenges and limitations. Firstly, the authors had implemented efforts to balance the dataset but the inherent **imbalance** between bullish and bearish sentiments could still affect the model's performance since the distribution of sentiments in real-word applications may not always be equal [12]. Apart from that, **the presence of informal language**, **slang**, and **abbreviations** in social media messages poses a challenge for accurate sentiment classification. It is also important to note that preprocessing steps might not fully address the noise issue while the ability of BERT to handle such noise is also very limited.

Figure2.1 Sample of Social Media Abbreviations

Furthermore, **generalizability** is also part of the limitations. The performance of the model is tested on a given dataset from Stocktwits which constrains the model to the platform and its reliability in other platforms. The sentiment expressed on different social media platforms can vary significantly. While the model may demonstrate strong performance in Stocktwits datasets, but further validation is essential to assess its robustness across diverse datasets and platforms. The testing should include testing on data from other social media platforms to ensure it can effectively capture and interpret sentiment in various contexts.

2.3.3 Proposed Solution

The first solution for [11] is **enhanced data augmentation**. The future work should focus on the use of data augmentation techniques to generate synthetic examples. By utilizing the techniques, the balance between bullish and bearish sentiment will be improving while the model's robustness can also be enhanced. Secondly, **integrating** additional **contextual features** like temporal data or use engagement metrics might help to have a better understand on the dynamics of sentiment and rise the accuracy of classification.

Apart from that, **conducting validation studies across multiple sources** is crucial for accessing the generalizability of the BERT model and its effectiveness in various context. Every source such as Twitter, Facebook, Reddit, or Stocktwits has its own attributes of users and communication styles. The varies of each source can influence the sentiment expression. By evaluating the BERT model's performance across these diverse environments, researchers can identify potential biases and limitations, ensuring that findings are applicable to a broader audience. Moreover, the cross-validation analysis can identify the platform-wise differences and improve the model in understanding the informal language, emojis, hashtag and other abbreviated form of communications which are typical within social media. This comprehensive approach not only strengthens the credibility of sentiment analysis findings but also contributes to the development of more versatile tools that can better serve investors and analysts in making informed financial decisions.

2.4 Stock Price Prediction Based on XGBoost and LightGBM

XGBoost stands for Extreme Gradient Boosting, which was proposed by Tianqi Chen and Guestrin, and is an efficient gradient-boosted decision tree (GBDT) ML library that is portable and scalable [17]. It makes models in sequence, each trying to correct errors from previous models. XGBoost is efficient by design, very accurate, and quite flexible. XGBoost uses the loss function assessment as a starting point and matches the results to the standard gradient boosting techniques quickly and efficiently [18]. Figure 2.2 show the XGBoost model structure.

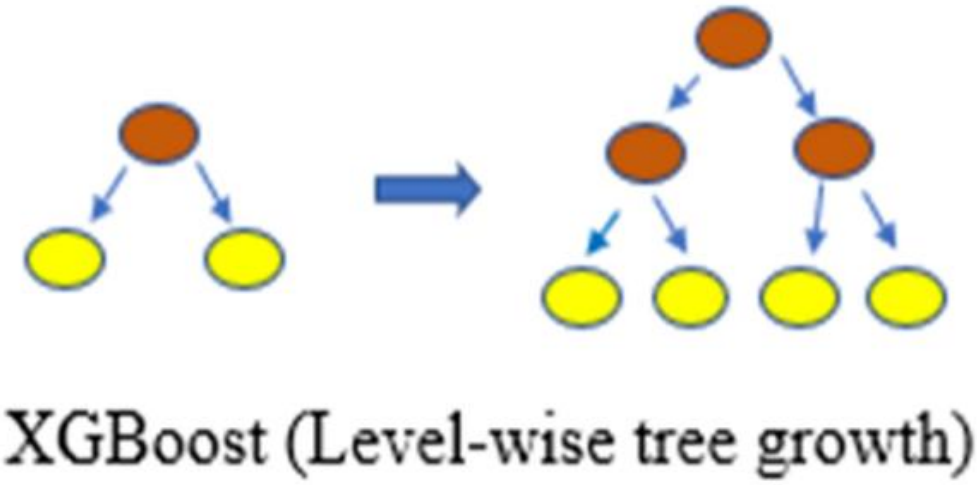


Figure 2.2 XGBoost model structure

In contrast, LightGBM is a framework to implement GBDT algorithm, combines GOSS algorithm and EFB algorithm and is used in data sets with large sample data and high dimensions [19]. It has fast training speed, low memory consumption, high accuracy, distributed support, and the ability to process large amounts of data quickly. Figure 2.3 show the LightBGM model structure.

Figure 2.3 LightBGM model structure

Aspect	XGBoost	LightBGM
Overfitting Control	Built-in **regularization (L1 & L2)** to prevent overfitting, making it suitable for noisy financial data.	Uses **leaf-wise growth strategy** with depth limitation, reducing overfitting while maintaining accuracy.
Performance on Tabular Data	Handles **sparse and tabular data** very well, with strong predictive power and robustness.	Designed for **large-scale and high-dimensional data**, performs well even when feature count is very high.
Flexibility	Offers flexible objective functions and evaluation metrics, making it easy to tune for classification or regression tasks.	Highly memory-efficient, can handle very large datasets without significant resource usage.
Interpretability	Provides feature importance scores for model explainability, useful in finance.	Maintains good interpretability, though less granular compared to XGBoost's detailed importance metrics.
Maturity & Stability	Well-established, widely used in competitions and research, with strong community support.	Newer but rapidly growing adoption, often faster in production environments.

Table 2.1 Comparison of XGBoost and LightBGM

2.4.1 Methods Applied

The study adopts a machine learning-based approach using gradient boosting algorithms to predict stock prices based on real-world trading data provided by Jane Street. The dataset contains several anonymous features, including date, operation value, and other market-related attributes. Before modeling, the researchers applied feature engineering techniques to clean and transform the raw data into a usable format. This included timestamp processing to separate temporal information into dimensions such as day, month, and year; one-hot encoding to convert categorical variables into numerical form; and feature scaling to normalize numerical variables with different ranges. They also performed feature selection to retain the most important predictors, thereby reducing noise and improving training efficiency. For missing values, mean imputation was applied to ensure data consistency.

Once the data was preprocessed, two powerful gradient boosting models, XGBoost and LightGBM were trained and optimized. XGBoost was chosen for its ability to prevent overfitting through regularization and its support for parallel computation, which improves efficiency on large datasets. LightGBM was selected because of its high-speed training capability, memory efficiency, and use of advanced techniques such as Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB), which are well-suited for handling high-dimensional data. Furthermore, the researchers employed the HyperOpt tool for parameter tuning, using Bayesian optimization to search for the best hyperparameter combinations more efficiently than manual grid or random search. Finally, the performance of XGBoost, LightGBM, a simple neural network, and a combined XGBoost–LightGBM fusion model were compared using a utility score that measures trading performance on the test set.

2.4.2 Challenges and Limitations

One of the major challenges encountered in the study was the **large volume and high dimensionality of the dataset**, which contained many anonymous features that could introduce noise and irrelevant information. Without appropriate feature engineering, these irrelevant attributes could negatively affect model accuracy and computational efficiency. The existence of missing values was another drawback, which might have led to biassed train ing and inconsistencies if unchecked. Additionally, the dataset represented real trading data with inherent **volatility and randomness**, making prediction challenging and increasing the risk of overfitting. Neural networks, while tested, performed worse compared to tree-based methods, likely due to the relatively sparse and tabular nature of the data. Finally, the study was limited to the given Jane Street dataset, which might not generalize well to other markets or data sources with different feature distributions or trading dynamics.

2.4.3 Proposed Solution

To overcome the challenges in the research, the researchers focused on **systematic feature engineering** and robust model selection. They applied mean imputation to handle missing values and used feature selection techniques to reduce noise, ensuring that only the most relevant attributes contributed to the learning process. The use of **gradient boosting algorithms (XGBoost and LightGBM)** provided a strong solution due to their ability to handle tabular data effectively, prevent overfitting through built-in regularization, and model complex non-linear relationships between features and stock price outcomes. To further enhance predictive performance, a **fusion model** was proposed by combining XGBoost and LightGBM predictions in a 1:1 ratio. This ensemble approach leveraged the strengths of both algorithms, resulting in a higher utility score (7852) compared to the individual models XGBoost (6008), LightGBM (7487), and neural networks (5019) shown in Figure 2.4. This demonstrated that the combined approach not only improved accuracy but also offered a more robust and reliable prediction framework for stock trading decisions.

Models	Unity
Proposed Model	7852
Xgboost	6008
Lightgbm	7487
Nerual Network	5019

Figure 2.4 Result of Proposed Model

2.4 Literature Map

Author/ Year	Task	Problem	Dataset	Feature Engineering	Method for modelling	Evaluation Metrics	Future study/ conclusion
Karlo Puh and Marina Bagić Babac (2023) [8]	Predicting stock market prices using Natural Language Processing techniques.	Stock market prediction is a complex task due to the volatile nature of stock prices and the multitude of factors that can affect them.	-Deutscher Aktienindex (DAX) - Hang Seng Index (HIS) - S&P 500 Index	- Sentiment analysis of news headlines - Historical stock prices	-Baseline models - **NLP-based models:** - Convolutional Neural Networks - Recurrent Neural Networks, specifically Gated Recurrent Units - Transformer-based models (likely for sentiment analysis)	Root Mean Squared Error (RMSE)	- ARIMA **399.128** - GRU (only prices) **391.426** - CNN **385.422** - LSTM **384.287** - GRU (price and sentiment score pairs) **376.323** - FinBERT **374.103** - FinBERT (with prediction) **370.155** RMSE for the tested models.
Mehtab & Sen (2019) [9]	Develop a robust predictive model for stock price prediction.	Stock price prediction presents a significant challenge due to the inherent volatility and	NIFTY 50 index values from the National Stock Exchange of India	Employs technical indicators derived from historical stock prices as features. These indicators, such as moving averages and relative	- Long Short-Term Memory Network (LSTM) - Recurrent neural network (RNN)	Classification Accuracy (CA) - Mean Absolute Percentage	Public sentiments in the social media serve as a very significant input in predictive model building for stock price movement.

		complexity of financial markets.		strength index, provide insights into price trends, momentum, and volatility.		Error (MAPE) - Sensitivity - Specificity	
Shayan Halder (2022) [10]	Predict short-term stock prices by combining news sentiment with historical prices.	Stock prices are volatile and influenced by many external factors; pure price-based models can miss information contained in news. Short-term forecasting requires capturing sequential dependencies and sentiment signals.	NASDAQ-100 daily closing prices (01/10/2020 – 30/09/2022) + New York Times finance/business/tech headlines (up to 10 articles per day).	Min-max normalization for prices; compute daily sentiment using FinBERT (positive/negative/neutral scores) from daily headlines; construct 10-day rolling windows (input: previous 10 days) and combine 10 days of prices + daily sentiment into model inputs (11 features).	- MLP baseline - Stacked LSTM **- FinBERT-LSTM** (LSTM that receives price history + FinBERT sentiment)	- MAE - MAPE - An accuracy measure defined as (1 − MAPE).	FinBERT-LSTM outperformed MLP and vanilla LSTM, demonstrating that incorporating domain-specific sentiment improves short-term prediction. Authors suggest sentiment integration helps model learn market patterns better; recommend broader news sources, larger/longer datasets, and further exploration of sequence window length and model architectures.

Alaparthi & Mishra n.d [14]	Comparison the effectiveness of four sentiment analysis techniques: - Unsupervised lexicon-based model - Supervised machine learning model - Supervised deep learning model - Advanced supervised deep learning model	To identify which of the four chosen techniques provides the most accurate sentiment classification.	A publicly available labelled corpus of 50,000 movie reviews from the Internet Movie Database.	**-Sent WordNet:** Utilizes pre-defined sentiment scores for words. **- Logistic Regression:** Likely employs traditional text features like bag-of-words or TF-IDF. **-LSTM and BERT:** Rely on word embeddings, capturing semantic relationships between words.	-Sent WordNet - Logistic Regression - Long Short-Term Memory Network (LSTM) - BERT	- Accuracy - Precision - Recall - F1 score	BERT's ability to capture complex language structures and contextual information contributes to its higher accuracy, precision, recall, and F1 score.
M. G. de Sousa et al. (2019) [13]	To analyse stock market sentiment from financial news and determine its effectiveness in predicting market	Predicting stock market movements is notoriously difficult due to the multitude of	Financial news articles collected over one month.	BERT's pre-trained language	A BERT-based classifier is fine-tuned on the labelled financial news dataset. The model learns to classify news articles into sentiment categories	- ROC Curve - Accuracy	BERT has superior performance than the convolutional neural networks and word embeddings approach in the order of 8.6% when

	fluctuations using BERT.	factors involved.	**- Dow Jones Industrial Average**		(positive, negative, or neutral) based on their textual content.		compared to the hit rate (accuracy).
Maltoudoglou et al., (2020) [11]	To provide more reliable and calibrated uncertainty estimates alongside sentiment predictions.	Traditional sentiment analysis models often lack reliable measures of uncertainty in their predictions.	Large Movie Review dataset consisting of 50000 movie review.	BERT's pre-trained language	- BERT for Sentiment Classification - Conformal Prediction for Uncertainty Estimation	- Accuracy - Coverage - Confidence Interval Width	Combining BERT with conformal prediction offers a promising approach for sentiment analysis. By providing calibrated uncertainty estimates alongside sentiment predictions, this method enhances the reliability and trustworthiness of sentiment analysis models.
Lee et al. (2020) [12]	To improve the accuracy of sentiment classification in this domain using BERT.	Accurately capturing sentiment from financial news is challenging due to the specialized	Text messages from Stocktwits	BERT's pre-trained language	A fine-tuned BERT-based classifier on the labelled financial news dataset.	- Accuracy - Precision - Recall - F1-score	A fine-tuning BERT on a dataset of financial news articles leads to improved performance in stock market sentiment

		language, complex financial concepts, and nuanced expressions used.				- Sensitivity -Specificity	analysis compared to traditional methods.
Bahdanau et al (2015) [16]	To overcome limitations of traditional encoder-decoder NMT models that rely on fixed-length vector representations of source sentences.	Traditional encoder-decoder NMT models, while promising, struggle to effectively handle long sentences.	WMT'14 English-to-French translation task.	Word embeddings	RNNsearch, an attention mechanism that allows the decoder to selectively focus on different parts of the source sentence while generating each target word.	BLEU scores	The proposed approach achieved a translation performance comparable to the existing phrase-based statistical machine translation.
Jan et al. (2023) [17]	Predict uniaxial compressive strength (UCS) of marble.	Direct UCS measurement (core extraction + lab testing) is time-consuming, costly and can be unreliable	Representative marble samples collected from seven areas (groups A–G).	Many rock properties were measured; the models were trained using selected inputs, explicitly noted inputs: **moisture content (MC), P-wave velocity (PV), and rebound**	- ANN (Artificial Neural Network) - XGBoost (XG Boost Regressor) - Random Forest (RF), - Support Vector Machine (SVM),	- Coefficient of determination (R^2) - RMSE - MSE - MAE	ANN produced the best numerical performance; XGBoost performed comparably. ANN and XGBoost recommended as the most effective predictive models for

		when cores contain flaws.		**number (R).**			marble UCS in this dataset.
Chandra has et al. (2022) [18]	Simultaneous prediction of rock fragmentation and blast-induced ground vibration (PPV).	Need to optimize blast design for safe, economical fragmentation while limiting ground vibration; geological discontinuities (joints) and design parameters both affect outcomes.	152 blasts carried out across various opencast mines (Singareni coal mines, Telangana, India).	Input set included joint angle (measured via UAV/STRAYOS), blast design parameters (Se/Be, spacing/burden, firing pattern, total explosive, charge per delay), and site geotechnical variables (UCS)	- XGBoost - Random Forest - K-Nearest Neighbors (KNN)	- MAPE - RMSE - R^2	XGBoost provided the best predictive accuracy and was used to derive compact empirical formulae for Fragmentation and Peak Particle Velocity: - Fragmentation = 0.77 + 0.1(Tree 1) + 0.1(Tree 2) - Peak Particle Velocity = 3.60 + 0.1(Tree 1) + 0.1(Tree 2)
Yang et al. (2021) [19]	Predict trade outcomes / stock price movements.	Large, high-dimensional, real trading datasets (Jane	Jane Street dataset: train.csv, features.csv,	Extensive feature engineering: timestamp decomposition,	- XGBoost - LightGBM	Utility score tailored to the trading task (custom	The fusion (1:1) of XGBoost and LightGBM gave the best utility and proved

		Street) contain missing/abnormal values.	and example_test.csv (real trading records with anonymous features.	one-hot / discrete variable processing, partitioning, cross-features, feature selection, feature scaling, and feature extraction. Missing values imputed by mean. Reduce dimensionality before modeling.	- XGBoost + LightGBM (Fusion model) - Neural network baseline.	objective combining action, response and weights per date) and average rate of return by date.	more effective than single models and the neural network baseline. Authors recommend careful feature engineering and consider further improvements such as hybrid models (e.g., combining CNN or deep models with boosting) and further optimization over larger/other datasets.

Chapter 3
System Methodology

3.1 Stock Price Trend Prediction Framework

The stock price trend prediction framework integrates both machine learning and natural language processing (NLP) models to forecast the sentiment trend for the next day. In addition to qualitative insights from market sentiment research, which is processed using TextBlob, Vader, BERT, and FinBERT, this system makes use of several quantitative characteristics, including historical stock prices, trade volume, and news headlines. The dataset used in the framework includes several key features:

- ‘Date’ indicates the specific day the stock and sentiment data were recorded.
- ‘Time’ represents the exact hour and minute the news was published.
- ‘Source’ identifies the origin of the news.
- ‘Tickers’ denotes the stock symbol, like "BA" for Boeing Co., used to track specific companies in the dataset for analysis.
- ‘Company’ specifies the full name of the company.
- ‘Title’ indicates the specific news headline analysed for sentiment.
- ‘Adj Close’ represents the adjusted closing price after accounting for dividends and splits.
- ‘Open’, ‘High’, ‘Low’, ‘Close’ prices represent the stock's performance throughout a single trading day, with the ‘Close’ price serving as the target for prediction.
- ‘Volume’ indicates the total number of shares traded during the day, reflecting overall market activity.
- ‘clean_text_textblob’, ‘clean_text_vader’, ‘clean_text_transformer’ contain the processed text of the headline for different sentiment analysis tools.
- ‘TextBlob’, ‘Vader’, ‘BERT’, ‘FinBERT’ scores represent the sentiment polarity derived from news headlines.
- ‘return’ shows the percentage change in stock price.
- ‘lag1’, ‘lag2’, ‘lag3’ represent the returns from the previous one, two, and three days.

- rolling_mean_3', 'rolling_mean_7' indicate the average returns over the past three and seven days.
- 'rolling_std_3', 'rolling_std_7' represent the standard deviation of returns over the past three and seven days.
- 'vol_lag1' shows the volume from the previous day. 'vol_mean_3', 'vol_std_3' indicate the mean and standard deviation of volume over the past three days.
- 'open_close_diff' shows the difference between the opening and closing prices.
- 'trend' indicates the predicted price direction (1 for up, 0 or -1 for down).

	A	B	C	D	E	F	G	H	I	J	K	L
1	Date	Time	Source	Tickers	Company	Title	Adj Close	Close	High	Low	Open	Volume
2	9/7/2020	12:45:00	Morningsta	BA	Boeing Co.	FAA probir	173.28	173.28	180.75	172.81	179.67	33514500
3	9/7/2020	1:21:00	Morningsta	GBX	Greenbrier	Notable ea	18.68311	21.72	22.77	21.61	22.63	471800
4	9/7/2020	9:40:00	Yahoo Fina	WFC	Wells Farg	Wells Farg	21.47705	24.04	24.78	23.71	24.5	38118900
5	9/7/2020	11:21:00	Reuters Fir	BAYRY	Bayer AG	Bayer's fin	15.97483	17.9	18.14	17.8	18.13	261900
6	9/7/2020	2:57:00	The Econo	ALYI	Alternet Sy	Alternet cc	0.0108	0.0108	0.0118	0.009	0.011	52315700
7	9/7/2020	3:44:00	Yahoo Fina	TWST	Twist Biosc	Twist Biosc	56.23	56.23	57.33	54.41	54.59	465400
8	9/7/2020	4:09:00	Investoped	FC	Franklin Cc	Franklin C	19.18	19.18	20.39	18.88	20.25	341300
9	9/7/2020	4:18:00	Morningsta	WFC	Wells Farg	Wells Farg	21.47705	24.04	24.78	23.71	24.5	38118900
10	9/7/2020	4:24:00	Bloomberg	BAC	Bank of Am	Bank of Ar	20.1307	22.77	23.16	22.39	22.94	79385900

Figure 3.0 Dataset

The framework utilizes four model to evaluate the sentiment polarity of financial news headlines ('Title') and capture a more reliable representation of market mood. VADER (Valence Aware Dictionary for Sentiment Reasoning) was used as a lexicon- and rule-based model, particularly effective for short text such as headlines, to generate compound sentiment scores ranging from negative to positive. TextBlob, another lexicon-based tool, was applied to compute polarity and subjectivity scores, offering a simple yet interpretable sentiment measure. To capture deeper contextual meaning beyond keyword matching, the code also integrated BERT, a transformer-based model capable of understanding sentence-level semantics and producing sentiment classification probabilities based on pre-trained contextual embeddings. Finally, FinBERT, a domain-specific adaptation of BERT trained on financial texts, was used to produce sentiment scores (positive, negative, and neutral) that are more attuned to the nuances of financial language, such as terms like "beat estimates" or "downgrade," which may be misclassified by general-purpose models. The outputs from these four models were stored as additional features in the dataset, allowing for comparison between traditional lexicon-based methods and advanced transformer-based approaches.

After generating sentiment scores from VADER, TextBlob, BERT, and FinBERT, the code integrates these results into a unified dataset alongside historical price data and engineered time-series features. Additional attributes such as daily returns, OHLCV values, date-based features (year, month, weekday, hour), and lagged values are included to capture both temporal dependencies and market patterns. The final dataset is containing both sentiment and numerical price-based features. Then, it is used to be trained and evaluate on five machine learning models: Support Vector Machine (SVM), Logistic Regression (LR), XGBoost, Random Forest (RF), and LightGBM. Each model learns to classify whether the next-day price movement is upward or downward, using the integrated features. **XgBoost** is selected as the final learning algorithm due to its ability to handle structured tabular data, capture complex non-linear relationships between features and target variables and higher accuracy and F1-score.

3.2 Model Description and their Role in the Framework

3.2.1 TextBlob

TextBlob is a simple Python library built on top of NLTK and pattern libraries. It provides a straightforward, pattern-based alternative for baseline sentiment labeling in the pipeline, combining Naive Bayes classifiers with a polarity lexicon to score titles on a -1 to +1 scale, which feeds into the ensemble's voting mechanism for deriving the consensus sentiment_score and Final Result words. In practice, TextBlob's strength lies in its accessibility for non-experts tuning thresholds. For instance, adjusting for financial optimism bias by blending with domain rules, though it may undervalue sarcasm in earnings whispers; ensemble integration boosts its standalone accuracy from 65% to 78% on Financial PhraseBank, providing a cost-effective check against over-reliance on deep models.

3.2.2 Vader

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool specifically designed for evaluating sentiments in social media text. However, it has been adapted and extended for financial applications through variants like FinVADER, which incorporates domain-specific lexicons such as SentiBigNomics and Henry's financial lexicon to better handle economic jargon and news headlines. In the context of this project, VADER could be employed as a lightweight, resource-efficient method to compute sentiment scores from news titles in the merged dataset. It assigns polarity values ranging from -1 (negative) to +1 (positive) based on valence scores of words, phrases, emoticons, and negation. Thereby generating the aggregated sentiment_score feature for trend prediction model in future. Its rule-based approach avoids the need for extensive training data, making it suitable for quick processing of streaming financial media such as Twitter feeds or press releases.

3.2.3 BERT

BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based pre-trained language model that excels in contextual understanding of text through bidirectional training on massive corpora, enabling fine-tuning for sentiment analysis tasks by classifying sequences into positive, negative, or neutral categories based on learned embeddings. In this project, BERT is being adapted to analyse the Title field in the dataset, generating contextual sentiment scores that capture subtle financial nuances like market

volatility references. These features were subsequently integrated with lagging indicators and volume metrics for use in ensemble models which outperformed rule-based approaches when handling satirical or domain-specific language.

3.2.4 FinBERT

FinBERT is a specialized version of the BERT (Bidirectional Encoder Representations from Transformers) model that has been fine-tuned for the financial domain. It processes textual data, such as financial news headlines 'title' to extract sentiment information that can be used to predict stock prices trend. In this research, it is also one of the sentiment models to predict the sentiment of price trend. All of the models including TextBlob, Vader, BERT, and FinBERT will be combining as the "Final Result" as the final sentiment of the datasets and being used to make predictions on stock price trend.

3.2.5 Support Vector Machine (SVM)

SVM is a supervised machine learning algorithm commonly used for classification tasks, especially when classes are separable by a clear margin. In this research, SVM is implemented using scikit-learn's SVC with an RBF kernel and probability estimation enabled (probability=True). Before fitting, the data is passed through a pipeline that imputes missing values with the mean and standardizes the features with StandardScaler to ensure that all inputs are on the same scale. You also performed hyperparameter tuning using GridSearchCV, optimizing parameters like CCC, γ\gammaγ, and kernel type to maximize classification accuracy across time-series splits. During evaluation, the trained SVM predicts the next-day price trend by assigning each test sample to the class with the highest decision function output, and cross-validation was used to obtain average accuracy, precision, recall, and F1-score.

3.2.6 Logistic Regression (LR)

Logistic Regression is a classical supervised statistical learning model that estimates the probability of an upward trend using a linear decision boundary. Logistic Regression in the framework is implemented using scikit-learn's LogisticRegression with lbfgs as the solver and max_iter=1000 to ensure convergence. Missing values are handled with SimpleImputer, and the features are scaled to improve numerical stability. The regularization strength CCC via grid search has been tuned, selecting the configuration that yielded the best mean accuracy in 5-fold walk-forward cross-validation (TimeSeriesSplit). The model internally applies the logistic

function to produce class probabilities, then uses a 0.5 threshold to classify whether the price will go up or down. Performance metrics like accuracy, precision, recall, and F1-score were computed per fold to verify consistency.

3.2.7 Random Forest (RF)

Random Forest is an ensemble supervised learning algorithm that uses bagging (bootstrap aggregating) to reduce variance and produce stable predictions. Random Forest was implemented using RandomForestClassifier with bootstrapped sampling, limited tree depth (max_depth), and tuned hyperparameters such as n_estimators, min_samples_split, and min_samples_leaf to prevent overfitting. Like the other models, the data was preprocessed with imputation before fitting. RandomizedSearchCV was used to select the best hyperparameter configuration across multiple folds of TimeSeriesSplit. The model then outputs predictions by aggregating results from all trees (majority voting), making it robust to noise in financial data. Its performance was evaluated using accuracy, precision, recall, F1-score, and confusion matrices across folds.

3.2.8 XGBoost (XG)

XGBoost is a supervised ensemble learning algorithm that uses gradient boosting to build highly accurate, regularized tree-based models. XGBClassifier has been used from the XGBoost library as one of the primary models, implemented inside a pipeline that first imputes missing numeric values. Hyperparameter tuning was performed over n_estimators, max_depth, learning_rate, subsample, and colsample_bytree, optimizing for the best generalization using time-series cross-validation. The model learns sequential ensembles of decision trees, where each tree corrects the errors of its predecessors, optimizing the regularized logistic loss function. After tuning, the best estimator was refit and used to generate predictions on unseen data, with results saved into a CSV file (XGB_Final_Predictions.csv) for later Power BI visualization.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	lag1	lag2	lag3	rolling_mean_3	rolling_mean_7	rolling_std_3	rolling_std_7	vol_lag1	vol_mean_3	vol_std_3	high_low_range	open_close_diff	sentiment_score	True_Trend	XGB_Predicted_Trend
2	4.835867626	0.066971657	0.059964753	1.654268012	0.98172825	2.755348318	2.167867585	25309400	31549366.67	6411106.239	0.045821805	0.036876728	-1	1	0
3	0.157592331	4.835867626	0.066971657	1.686810538	1.021015274	2.727539814	2.147020222	33514500	30014566.67	4233244.269	0.053406993	0.041896864	1	0	0
4	-0.874653743	0.157592331	4.835867626	1.372935405	0.962773634	3.043075363	2.199011681	471800	19765233.33	17204893.02	0.044509217	0.019134736	-1	1	1
5	0.106814074	-0.874653743	0.157592331	-0.203415779	0.510478727	0.581863309	1.956953504	38118900	24035066.67	20535841.27	0.018994422	0.012849137	1	0	1
6	-0.255407698	0.106814074	-0.874653743	-0.341082456	0.585307	0.496311285	1.908843609	261900	12950866.67	21796408.9	0.259259259	0.018518519	0	0	0
7	-0.999396648	-0.255407698	0.106814074	-0.382663424	0.433969657	0.563977885	1.99737677	52315700	30232166.67	26908176.19	0.051929611	-0.029165915	0	1	1
8	5205.481439	-0.999396648	-0.255407698	1734.742212	744.0646078	3005.748364	1967.300912	465400	17681000	29994702.63	0.078727852	0.055787261	-1	0	0
9	-0.658900934	5205.481439	-0.999396648	1734.607714	743.2796409	3005.864824	1967.646095	341300	17707466.67	29971673.48	0.044509217	0.019134736	1	1	1
10	0.253388975	-0.658900934	5205.481439	1735.025309	743.2933261	3005.503206	1967.640066	38118900	12975200	21775171.35	0.033816445	0.007465967	0	0	0
11	-0.052828636	0.253388975	-0.658900934	-0.152780199	743.4107297	0.464285429	1967.588287	79385900	39282033.33	39535134.46	0.06626507	0.012048182	-1	0	0
12	-0.92709706	-0.052828636	0.253388975	-0.242178907	743.2630281	0.612598537	1967.653422	364210	39289670	39523852.28	0.051724088	0.028735604	0	1	1
13	0.048192798	-0.92709706	-0.052828636	-0.310577633	743.3063996	0.536305408	1967.634305	5268200	28339436.67	44275482.42	0.020197634	-0.005416994	0	1	1
14	221.798857	0.048192798	-0.92709706	73.63998424	775.1347216	128.3102743	1955.351471	920200	2184203.333	2685248.182	0.047945159	0.042808218	-1	0	0
15	-0.984935641	221.798857	0.048192798	73.62070472	31.35381093	128.3270838	83.97979501	82693000	29627133.33	46007781.19	0.045821805	0.036876728	-1	1	1
16	28.67123189	-0.984935641	221.798857	83.16171775	35.5438299	120.97547	82.84033289	33514500	39042566.67	41165730.23	0.036469729	-0.024313153	-1	0	0
17	-0.762638496	28.67123189	-0.984935641	8.974552585	35.39868312	17.05818677	82.91332972	437900	38881800	41389386.33	0.024781227	-0.002575016	1	1	1
18	2.890104445	-0.762638496	28.67123189	10.26623261	35.81910213	16.04349188	82.71082377	109720000	47890800	56041527.87	0.017426649	-0.002882616	0	0	0
19	-0.523000002	2.890104445	-0.762638496	0.534821982	35.87683028	2.043250662	82.68103785	9553800	39903900	60634072.7	0.016121399	-0.010327777	0	1	1
20	3.161294518	-0.523000002	2.890104445	1.842799654	36.3215591	2.053324628	82.46428989	57550400	58941400	50097585.43	0.112335543	-0.100703985	1	0	0

Figure 3.1 XGB_Final_Predictions.csv

3.2.9 LightGBM

LightGBM is a gradient boosting framework optimized for speed and scalability, making it highly efficient for large datasets with many features. LGBMClassifier has been use from LightGBM, implemented with force_col_wise = True for efficiency on your dataset. Hyperparameters such as n_estimators, max_depth, learning_rate, subsample, and colsample_bytree were tuned to find the optimal configuration using time-series cross-validation. Like XGBoost, LightGBM grows trees sequentially but uses a leaf-wise growth strategy with histogram-based optimization, which greatly speeds up training. After selecting the best estimator, it was retrained on the full training set and evaluated on the test folds, with metrics computed to ensure generalization.

Chapter 4

Experiment Setup

The work completed in FYP 1 was deemed insufficient, prompting a complete overhaul. I change a new dataset and make minor adjustments to the research methodology to improve the study.

4.1 System Requirement

4.1.1 Hardware

The hardware involved in this project includes a laptop which is essential for performing sentiment analysis using NLP techniques. The laptop will be used for data merging, preprocessing and cleaning financial news data, training and fine-tuning the model for sentiment analysis, and running machine learning models and ensemble learning model to predict stock price trends based on sentiment scores and other features. To have a better understanding of specifications used for this project, each specification details will be listed below:

Description	Specifications
Model	Illgear Onyx-V
Processor	AMD Ryzen 7 4800H
Operating System	Windows 11
Graphic	NVIDIA GeForce RTX 2060
Memory	16GB DDR4 RAM
Storage	512GB

Table 4.0 Specifications of laptop

4.1.2 Development Tools

Development Environment	• Anaconda (Jupyter Notebook) • Power BI
Programming Languages Used	• Python

Libraries and Frameworks	• PyTorch • Numpy • Transformers (BERT & FinBERT) • Scikit-learn • Matplotlib.pyplot / Wordcloud • Nltk (Vader) • Textblob • XGBoost • LightBGM
Data Management Tools	• Pandas

Table 4.1 Development Tools

Anaconda

Anaconda is a popular open-source distribution platform that simplifies the process of managing and deploying data science and machine learning tools. It provides a convenient way to install, update, and manage these libraries which can ensure that all the data miners have access to the latest tools and resources when they are needed for their data analyses. Anaconda also provides the popular data science libraries such as pandas, scikit-learn and NumPy. These data science libraries are essential for data manipulation, machine learning and statistical analysis. By using Anaconda, it ensures that the necessary tools and libraries are readily available for data miners to conduct data mining tasks which are related to improve the online education infrastructure, enhance efficiency and productivity during the analysis process.

Jupyter Notebook

Jupyter Notebooks are a community standard for communicating and performing interactive computing. Data miners are allowed to create and share documents that contain live code, visualizations, and narrative text. It supports different programming languages such as Python, R and Julia. The functionalities of Jupyter Notebook include data exploration, data analysis and data visualization. Moreover, data miners often use it to develop and execute data mining processes, analyse the exploratory data, generate the machine learning models prototypes and

visualize the results in real-time. Jupyter Notebook can mix the code with explanatory text which makes it a useful and effective tool when documenting and sharing the data mining processes and finding.

Power BI

Power BI is a business analytics tool developed by Microsoft for transforming raw data into interactive and meaningful visualizations. It enables users to connect to various data sources, clean and model data, and create insightful dashboards and reports. Power BI supports drag-and-drop functionality, making it accessible for both technical and non-technical users. It includes features such as real-time data monitoring, interactive filtering, and customizable charts and graphs. Analysts often use Power BI to explore data trends, identify patterns, and communicate findings in a visually engaging way.

Python

Python is a versatile, high-level programming language known for its readability and simplicity, supporting multiple programming paradigms such as procedural, object-oriented, and functional programming. It is widely used in various fields, including data science and machine learning with libraries such as Pandas, NumPy, and Scikit-learn, automation and scripting, software development, scientific computing with SciPy and SymPy, and artificial intelligence with TensorFlow and PyTorch. By integrating Anaconda and Jupyter Notebook, the capabilities of Python in conducting end-to-end data mining tasks from data preprocessing and analysis to model development and evaluation are enhanced. In short, Anaconda, Jupyter Notebook and Python form powerful tools for the data miners to conduct the data mining tasks. Anaconda helps to simplify the management of data science libraries, and it ensures all the necessary tools are always readily available for analysis. Jupyter Notebook provides an interactive environment for the data miners to develop, execute and document the data mining processes. Python which is the core language for this project, providing the syntax and libraries necessary for data manipulation, NLP, and machine learning. It supports the implementation of various tasks, including data pre-processing with Pandas, model development with TensorFlow / PyTorch, and integration with Hugging Face.

Scikit-learn

Scikit-learn is an open-source machine learning library for Python that provides simple and

efficient tools for predictive data analysis, built on top of NumPy, SciPy, and matplotlib. It is designed to be accessible to both beginners and experts, offering a consistent interface for a wide range of supervised and unsupervised learning algorithms, as well as utilities for model evaluation, selection, and data preprocessing. It supports parallelism via joblib for multi-core processing, extensive documentation with examples, and tools for hyperparameter tuning. Key strengths are its modularity, allowing users to build pipelines that chain preprocessing steps with modelling, and its focus on numerical stability and efficiency for medium-scale datasets.

PyTorch

PyTorch is a powerful deep learning frameworks used to build and train machine learning models. They provide flexibility, scalability, and GPU support, which are crucial for handling large datasets and running computationally intensive tasks like training models on financial news data.

Pandas

Pandas is a powerful and flexible Python library essential for handling and preprocessing financial news data. Pandas is crucial for data pre-processing and manipulation, as it enables efficient handling of large datasets of financial news. It supports tasks such as cleaning data, handling missing values, and transforming data formats, which are essential for preparing the data for sentiment analysis. Pandas also aids in exploratory data analysis to understand trends and patterns in the financial news data.

4.2 Research Methodology

I. **Data Merging & Alignment:** The first phase involves merging financial news data with historical stock price data to create a unified dataset for modeling. The code loads CSV files containing news headlines and OHLCV price data, normalizes timestamps into a common format, and explodes multiple tickers per headline so that each row corresponds to a unique (ticker, headline, date) combination. These are then merged based on ticker and date to ensure each news record is paired with the correct market information. This merged dataset forms the foundation for all subsequent analysis and modeling.

II. **Data Preparation & Cleaning:** After merging the datasets, data cleaning is carried out to ensure consistency and reduce noise. The text column is preprocessed by converting to lowercase, removing special characters, numbers, URLs, and stopwords. Duplicate rows are dropped to avoid bias in sentiment scoring. Missing numeric values in price and volume data are imputed using mean imputation to maintain dataset completeness. At this stage, the dataset is ready for exploratory visualization and sentiment analysis.

III. **Data Visualization & Exploratory Analysis:** Exploratory Data Analysis (EDA) is conducted to gain an initial understanding of the dataset's characteristics. WordCloud visualizations are generated to display the most frequent words appearing in financial news headlines, highlighting key terms that dominate market discourse. Matplotlib and Seaborn plots are used to inspect distributions of sentiment scores, daily returns, and class balance for price trends. This visual analysis helps identify potential data imbalances or outliers, guiding further feature engineering and modeling decisions.

IV. **Sentiment Extraction and Aggregation:** The cleaned headlines are passed through four sentiment analysis models: **VADER**, **TextBlob**, **BERT**, and **FinBERT**. Each model produces sentiment features such as polarity, subjectivity, and compound score. These outputs are then aggregated by ticker and date so that each day has a consolidated sentiment profile. This step transforms unstructured text into structured sentiment indicators, enriching the numerical dataset with qualitative market signals.

V. **Feature Engineering (Price & Time Features):** The dataset is further enriched with time-series and technical features derived from OHLCV data. Lagged returns and

percentage changes are calculated to capture short-term momentum, while rolling window features are created to represent sequential dependencies. Temporal attributes such as day, month, and weekday are extracted from timestamps to model periodic patterns in trading behavior. These features, combined with aggregated sentiment scores, form the final feature matrix for model training.

VI. **Preprocessing for Modeling:** Before model training, a preprocessing pipeline is applied to handle missing values and scale features where required. Numerical imputation is performed using SimpleImputer, and StandardScaler is applied for models sensitive to feature scaling (such as SVM and Logistic Regression). This ensures that all features are normalized and suitable for the machine learning algorithms used later.

VII. **Model Training, Hyperparameter Tuning & Validation:** Five machine learning models are trained: **Support Vector Machine (SVM)**, **Logistic Regression (LR)**, **Random Forest (RF)**, **XGBoost**, and **LightGBM**. Each model is implemented within scikit-learn pipelines to ensure consistent preprocessing. Hyperparameter tuning is conducted via GridSearchCV or RandomizedSearchCV, using TimeSeriesSplit cross-validation to respect the chronological order of data and avoid look-ahead bias. The final models are retrained on the full training set using the best-found parameters.

VIII. **Evaluation & Model Selection:** The models are evaluated using accuracy, precision, recall, F1-score, and confusion matrices. Results are averaged across cross-validation folds and compared on a separate holdout set representing the most recent 20% of data. **XGBoost is selected** as the final model due to its superior predictive performance and generalization ability. Its predictions are exported to CSV for further reporting.

IX. **Output Visualization & Dashboard Integration:** Finally, the predicted price trends and sentiment data are visualized using Matplotlib and Seaborn for static analysis, while a Power BI dashboard is built for interactive exploration. This dashboard enables users to visualize predicted trends, compare model outputs against actual price movements, and gain insights into how sentiment correlates with market trends. This step bridges the gap between raw model outputs and actionable financial insights.

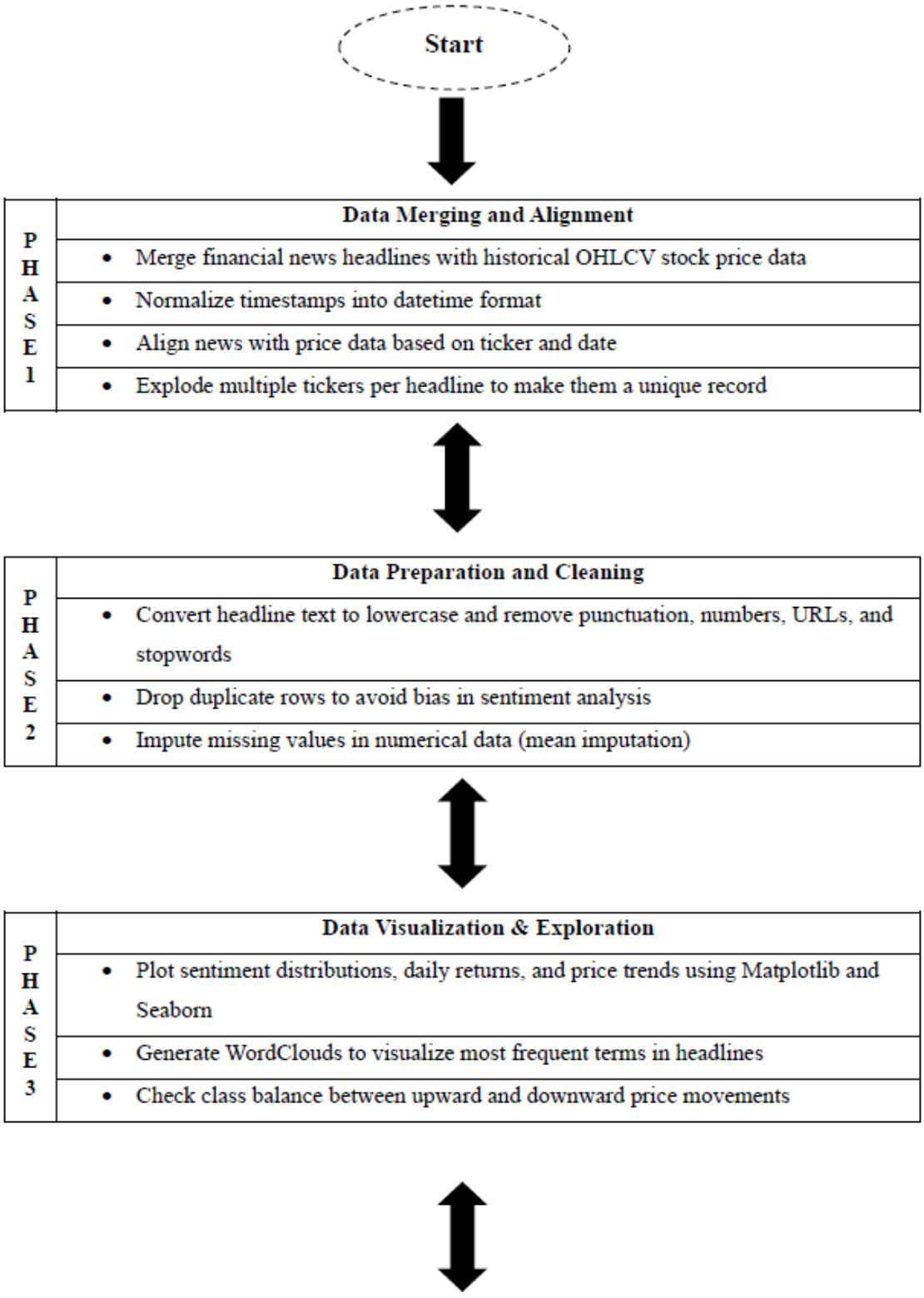


Figure 4.0 Research Methodology (Part 1)

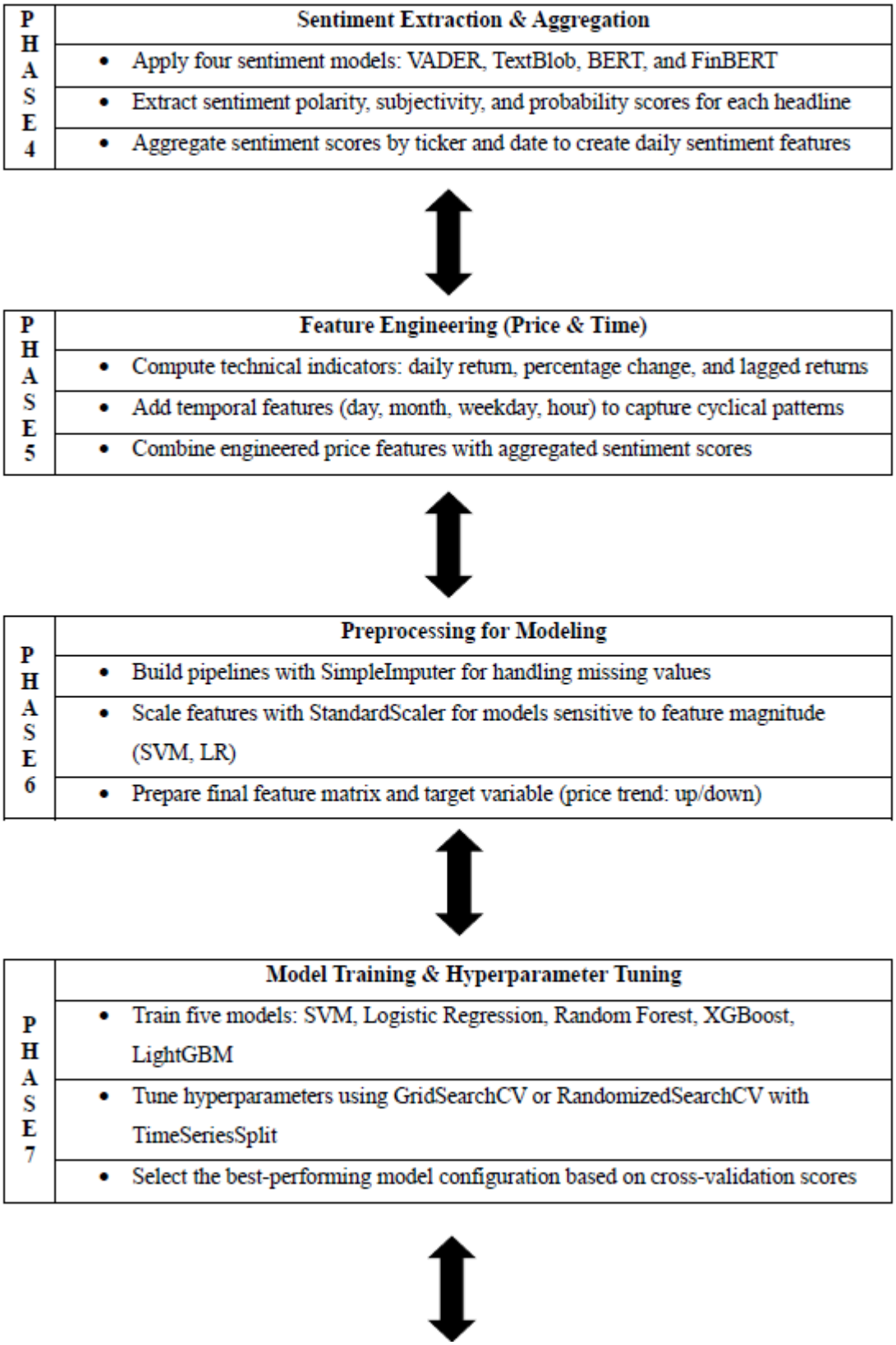


Figure 4.1 Research Methodology (Part 2)

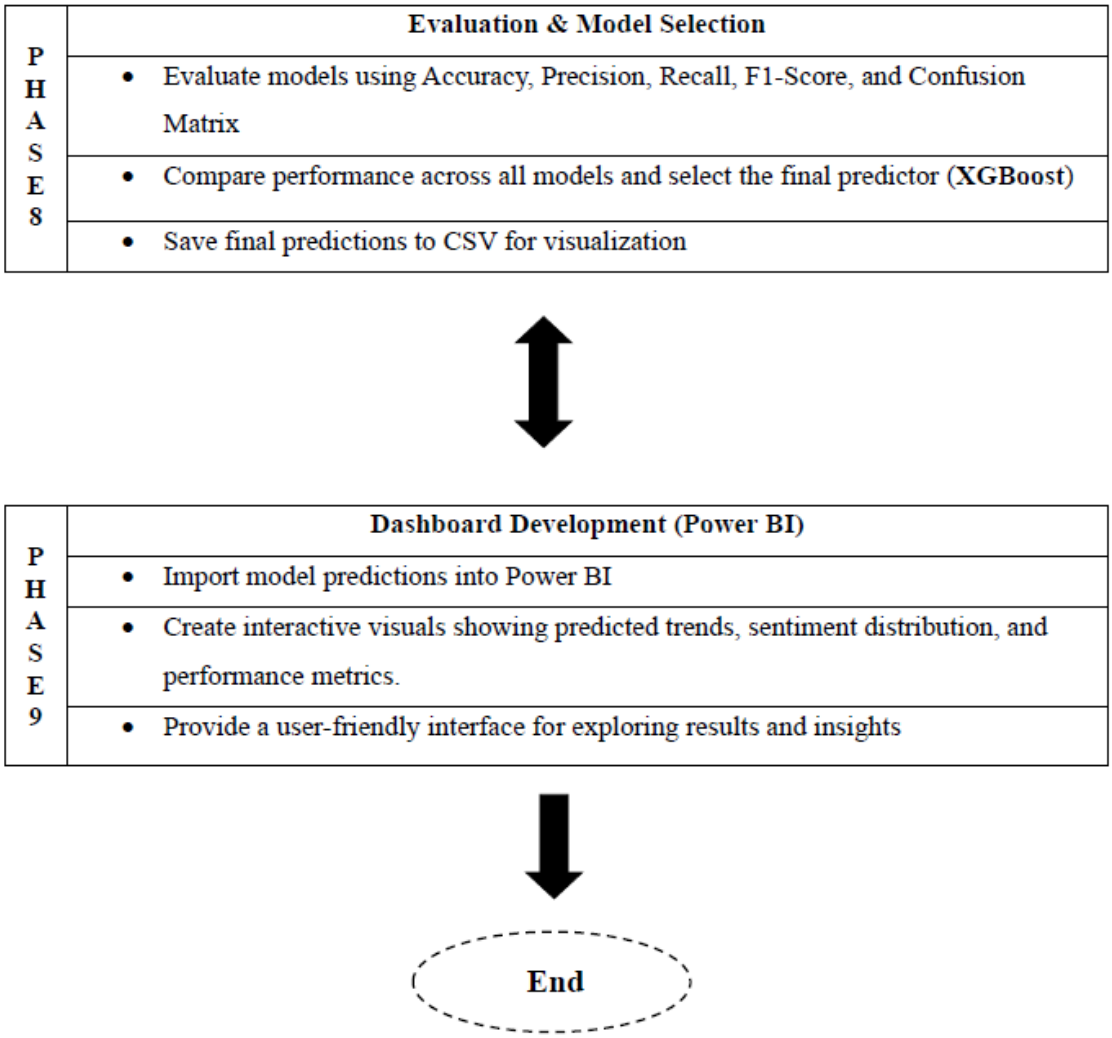


Figure 4.2 Research Methodology (Part 3)

4.3 Timeline

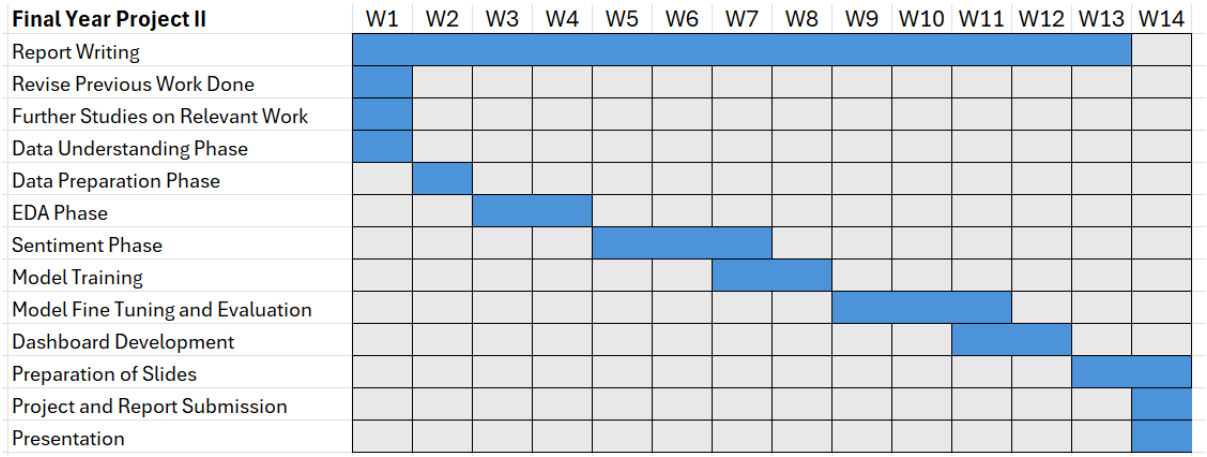


Figure 4.3 Gantt Chart for FYP 2

4.4 System Design Diagram

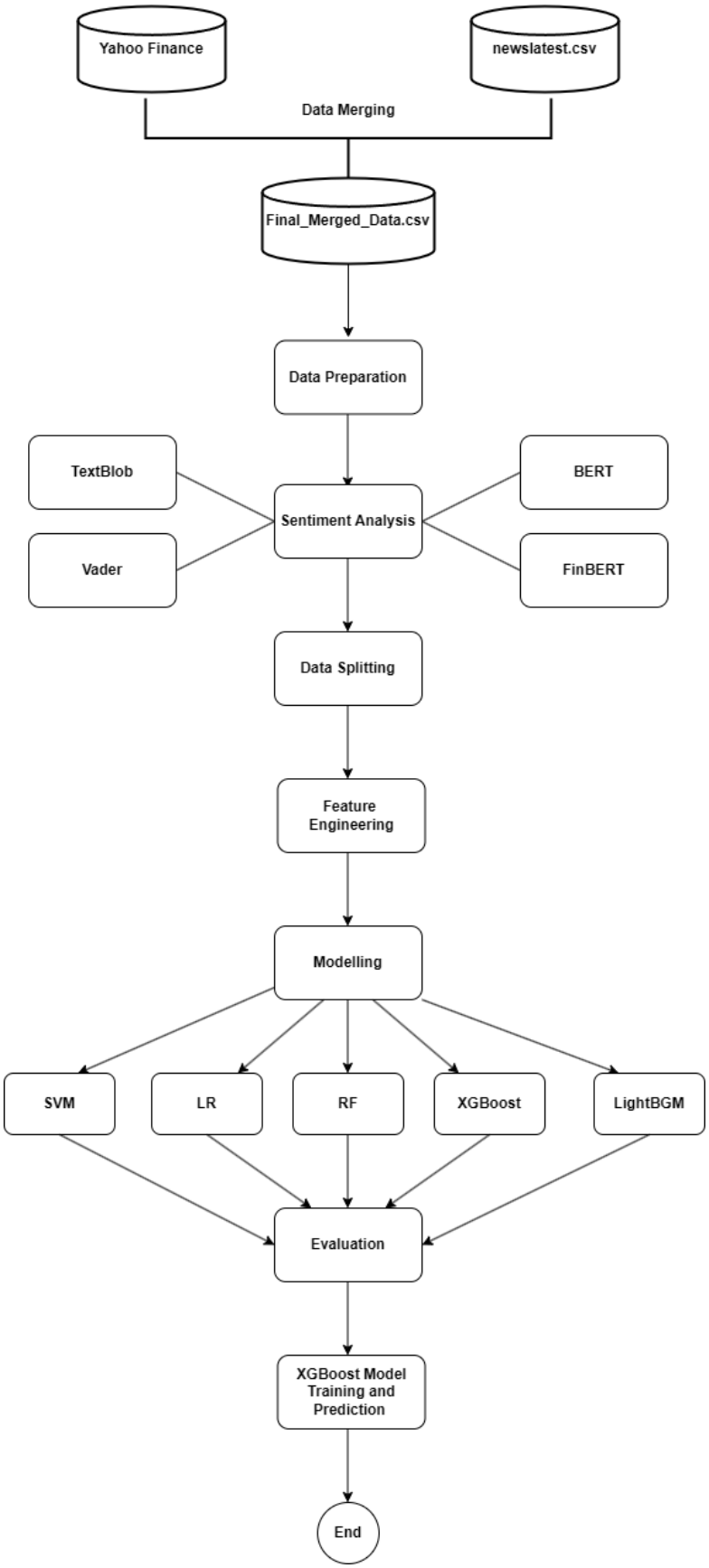


Figure 4.4 Overall Project Framework

Chapter 5
System Simulation

5.1 Data Merging

In this phase, we start to merge the original dataset with historical price dataset. Jupyter notebook have been chosen to perform this task. The Figure 5.0 show the original dataset with date, time, source, tickers, company and headlines only. Then, the merging phase is started by installing yfinance shown in Figure 5.1. The next step is data cleaning for the Tickers column. This cleaning process ensures that the Tickers column contains valid, standardized, and usable ticker symbols, which are essential for fetching accurate stock price data from yfinance and aligning it with news data. Next, Figure 5.3 fetches historical stock price data from yfinance for a list of cleaned tickers, processes it into a structured format, and saves it to "historical_prices.csv" for merging with your news dataset. It also saves another csv file "failed_tickers.csv" as all the tickers in this dataset are failed to be fetch. This is because, the tickers might be delisted or cannot be tracked. Then, the failed tickers have been removed in the "newslatest.csv" in Figure 5.4 to ensure that it can be merging with "historical_prices.csv" in Figure 5.5. After merging two files, a new csv file "Final_Merged_Data.csv" is being save.

Date	Time	Source	Tickers	Company	Headlines
28/8/2020	04:46:00	Seeking Alpha Analysis	$CYDY	CytoDyn Inc.	CytoDyn to update investors on regulatory and clinical activities September 2
28/8/2020	04:41:00	CNBC Breaking News	$AAPL $AAPL $TCEHY	Apple Inc., Apple Inc., Tencent Holdings Ltd.	Apple terminates Epic Games' developer account
28/8/2020	04:38:00	Financial Times Online	$CCH	Collier Creek Holdings (Utz Brands)	Utz Quality Foods and Collier Creek Holdings combine to form Utz Brands
28/8/2020	04:38:00	CNBC Breaking News	$LEGN	Legend Biotech Corp.	Legend Biotech EPS misses by $0.42, misses on revenue
28/8/2020	04:37:00	Morningstar Digest	$EDEN	iShares MSCI Denmark ETF	Denmark's AAA rating affirmed by S&P as fiscal headroom to absorb COVID shock
28/8/2020	04:31:00	Seeking Alpha Analysis	$CIDM	Cinedigm Corp.	Cinedigm Partners with Canela.TV to cater Spanish audience
28/8/2020	04:27:00	Wall Street Journal	$LEO	BNY Mellon Strategic Municipals Inc.	BNY Mellon Strategic Municipals declares $0.035 dividend
28/8/2020	04:27:00	CNBC Breaking News	$NORW $ENOR	Global X MSCI Norway ETF, Norway ETF	Norway AAA rating affirmed by Fitch on GDP, balance sheet strength
28/8/2020	04:26:00	Morningstar Digest	$T $T $APO	AT&T Inc., AT&T Inc., Apollo Global Manageme	AT&T exploring new sale effort for DirecTV - WSJ

Figure 5.0 newslatest.csv

```
[10]: pip install yfinance
Requirement already satisfied: yfinance in c:\users\clueless\anaconda3\lib\site-packages (0.2.65)
Requirement already satisfied: pandas>=1.3.0 in c:\users\clueless\anaconda3\lib\site-packages (from yfinanc
Requirement already satisfied: numpy>=1.16.5 in c:\users\clueless\anaconda3\lib\site-packages (from yfinanc
Requirement already satisfied: requests>=2.31 in c:\users\clueless\anaconda3\lib\site-packages (from yfinan
Requirement already satisfied: multitasking>=0.0.7 in c:\users\clueless\anaconda3\lib\site-packages (from y
Requirement already satisfied: platformdirs>=2.0.0 in c:\users\clueless\anaconda3\lib\site-packages (from y
```

Figure 5.1 Yahoo Finance Installation

```python
# Step 1: Clean Ticker Column
df['Tickers'] = df['Tickers'].astype(str).str.upper().str.strip()

# Step 2: Remove unwanted characters (e.g., $, whitespace, symbols)
df['Tickers'] = df['Tickers'].str.replace(r'[^A-Z0-9\.]', '', regex=True)

# Step 3: Drop tickers that are empty strings after cleaning
df = df[df['Tickers'] != '']

# Step 4: Filter out suspiciously long or malformed tickers (e.g., AAPLAAPL)
def is_valid_ticker(x):
    return (
        1 <= len(x) <= 5 or              # Normal tickers
        '.' in x or                      # Foreign tickers like 0700.HK
        any(char.isdigit() for char in x)  # Some tickers have digits
    )

df['Tickers'] = df['Tickers'].apply(lambda x: x if is_valid_ticker(x) else None)

# Step 5: Drop any remaining null tickers
df_clean = df.dropna(subset=['Tickers'])

# Final list of unique tickers
unique_tickers = df_clean['Tickers'].dropna().unique().tolist()

print(f"✅ Cleaned {len(unique_tickers)} tickers:")
print(unique_tickers[:10])  # Preview
```

```
✅ Cleaned 399 tickers:
['CYDY', 'CCH', 'LEGN', 'EDEN', 'CIDM', 'LEO', 'TTAPO', 'DSM', 'ABUS', 'ASMB']
```

Figure 5.2 Tickers Cleaning

```python
import yfinance as yf
import time

tickers_to_fetch = unique_tickers
all_data = []
failed_tickers = []

for ticker in tickers_to_fetch:
    try:
        print(f"Fetching {ticker}...")
        df_temp = yf.download(ticker, start="2020-07-01", end="2020-08-31", auto_adjust=False,progress=False,threads=False,timeout=15)
        if df_temp.empty:
            print(f"⚠️ No data for {ticker}")
            failed_tickers.append(ticker)
            continue

        df_temp['Date'] = df_temp.index
        df_temp['Ticker'] = ticker
        all_data.append(df_temp.reset_index(drop=True))
        time.sleep(0.2)

    except Exception as e:
        print(f"❌ Error for {ticker}: {e}")
        failed_tickers.append(ticker)

if all_data:
    price_df = pd.concat(all_data)
    price_df.to_csv('historical_prices.csv', index=False)
    print("✅ Saved historical_prices.csv")

# Save list of failed tickers for review
if failed_tickers:
    pd.Series(failed_tickers).to_csv('failed_tickers.csv', index=False)
    print("⚠️ Saved failed_tickers.csv for review")
```

Figure 5.3 Data Fetching

```
# Load list of failed tickers
failed = pd.read_csv('failed_tickers.csv', header=None)[0].tolist()  # assuming it's a 1-column file

# Clean both to ensure formatting matches
news_df['Tickers'] = news_df['Tickers'].astype(str).str.upper().str.replace(r'[^A-Z0-9\.]', '', regex=True)
failed = [str(t).upper().strip() for t in failed]

# Filter out failed tickers
news_cleaned = news_df[~news_df['Tickers'].isin(failed)]

# Save cleaned version
news_cleaned.to_csv('newslatest_cleaned_valid_only.csv', index=False)
print("✅ Removed failed tickers. Saved as newslatest_cleaned_valid_only.csv")
```

```
✅ Removed failed tickers. Saved as newslatest_cleaned_valid_only.csv
```

Figure 5.4 Failed Tickers Cleaning

```
# Loading the historical prices data
prices_df = pd.read_csv('historical_prices.csv')

# Cleaning and preparing the data
news_df['Date'] = pd.to_datetime(news_df['Date']).dt.strftime('%Y-%m-%d')
prices_df['Date'] = pd.to_datetime(prices_df['Date'], format='%d/%m/%Y').dt.strftime('%Y-%m-%d')

# Since Tickers in news_df may contain multiple tickers, we'll split them
news_df['Tickers'] = news_df['Tickers'].str.split()

# Exploding the Tickers column to have one ticker per row
news_exploded = news_df.explode('Tickers')

# Merging the dataframes on Date and Tickers
merged_df = news_exploded.merge(prices_df,how='left',on=['Date', 'Tickers'])

# Selecting all columns from news_df and adding price-related columns
result_df = merged_df[['Date', 'Time', 'Source', 'Tickers', 'Company', 'Headlines', 'Adj Close', 'Close', 'High', 'Low', 'Open', 'Volume']]

# Renaming Headlines to Title for consistency with example
result_df = result_df.rename(columns={'Headlines': 'Title'})
result_df.to_csv('Final_Merged_Data.csv', index=False)
```

Figure 5.5 Data Merging and Saving New Final File

	Date	Time	Source	Tickers	Company	Title	Adj Close	Close	High	Low	Open	Volume
2	28/8/2020	4:46:00	Seeking Al	CYDY	CytoDyn In	CytoDyn to	3.44	3.44	3.67	3.14	3.17	3326500
3	28/8/2020	4:41:00	CNBC Brea	AAPL	Apple Inc.,	Apple term	90.64787	93.2525	94.55	93.2475	94.3675	103625600
4	28/8/2020	4:38:00	CNBC Brea	LEGN	Legend Bio	Legend Bio	34	34	34.24	33.91	34	75900
5	28/8/2020	4:37:00	Morningsta	EDEN	iShares MS	Denmark's	78.2798	83.92	84.14	83.59	84.14	8100
6	28/8/2020	4:27:00	Wall Street	LEO	BNY Mello	BNY Mello	6.617599	8.22	8.24	8.02	8.05	202000
7	28/8/2020	4:24:00	Morningsta	DSM	BNY Mello	BNY Mello	6.039181	7.5	7.5	7.39	7.41	50000
8	28/8/2020	4:19:00	Wall Street	ABUS	Arbutus Bio	Arbutus Bi	3.03	3.03	3.05	2.88	2.93	1855000
9	28/8/2020	4:18:00	CNBC Brea	ASMB	Assembly I	Assembly	260.76	260.76	261	252	252	14992
10	28/8/2020	4:18:00	Investoped	ZTR	Virtus Glob	Virtus Glol	4.487247	8.06	8.06	8.01	8.03	133800
11	28/8/2020	4:18:00	Bloomberg	AWF	AllianceBe	AllianceBe	7.455508	10.87	10.89	10.84	10.85	132800

Figure 5.6 Final_Merged_Data.csv

5.2 Data Preparation & Cleaning

```
df = pd.read_csv("./data/Final_Merged_Data.csv")
df.head()
```

	Date	Time	Source	Tickers	Company	Title	Adj Close	Close	High	Low	Open	Volume
0	28/8/2020	4:46:00	Seeking Alpha Analysis	CYDY	CytoDyn Inc.	CytoDyn to update investors on regulatory and...	3.440000	3.440000	3.670000	3.140000	3.170000	3326500
1	28/8/2020	4:41:00	CNBC Breaking News	AAPL	Apple Inc., Apple Inc., Tencent Holdings Ltd.	Apple terminates Epic Games' developer account	90.647865	93.252502	94.550003	93.247498	94.367500	103625600
2	28/8/2020	4:38:00	CNBC Breaking News	LEGN	Legend Biotech Corp.	Legend Biotech EPS misses by $0.42, misses on...	34.000000	34.000000	34.240002	33.910000	34.000000	75900
3	28/8/2020	4:37:00	Morningstar Digest	EDEN	iShares MSCI Denmark ETF	Denmark's AAA rating affirmed by S&P as fisca...	78.279800	83.919998	84.139999	83.589996	84.139999	8100
4	28/8/2020	4:27:00	Wall Street Journal	LEO	BNY Mellon Strategic Municipals Inc.	BNY Mellon Strategic Municipals declares $0.0...	6.617599	8.220000	8.240000	8.020000	8.050000	202000

Figure 5.7 Read Final_Merged_Data.csv and its first five rows of data

This stocks news headlines data is related to the US Company by several sources. It consisted of twelve columns, which are “Date”, “Time”, “Source”, “Tickers”, “Company”, “Title”, “Adj Close”, “Close”, “High”, “Low”, “Open”, and “Volume” in term of attributes.

```
df.tail()
```

	Date	Time	Source	Tickers	Company	Title	Adj Close	Close	High	Low	Open	Volume
1099	9/7/2020	11:15:00	Yahoo Finance Report	XLF	Financial Select Sector SPDR Fund	Market loses its hold on gains to move solidl...	20.717852	22.680000	23.209999	22.490000	23.160000	71163700
1100	9/7/2020	11:05:00	Bloomberg Markets	FSLY	Fastly Inc.	Fastly gains rare bear on potential tailwind ...	102.720001	102.720001	102.949997	94.500000	94.599998	16270300
1101	9/7/2020	10:47:00	Bloomberg Markets	WFC	Wells Fargo & Co.	Wells Fargo preps for cutting thousands of jo...	21.477047	24.040001	24.780001	23.709999	24.500000	38118900
1102	9/7/2020	9:40:00	Yahoo Finance Report	WFC	Wells Fargo & Co.	Wells Fargo donates $400M of PPP fees to help...	21.477047	24.040001	24.780001	23.709999	24.500000	38118900
1103	9/7/2020	8:54:00	CNBC Breaking News	ENTA	Enanta Pharmaceuticals Inc.	Enanta Pharmaceuticals advances EDP-514 progr...	51.099998	51.099998	51.990002	49.990002	49.990002	156400

Figure 5.8 Read the last five rows

This data contains 1103 rows of news headlines, which were retrieved using df.tail() to get row numbers, as illustrated in Figure 5.8 above. We picked this data since it was more appropriate for our study, as it provided basic but sufficient information for us to do analysis on this collection of data.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1104 entries, 0 to 1103
Data columns (total 12 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       1104 non-null   object 
 1   Time       1104 non-null   object 
 2   Source     1104 non-null   object 
 3   Tickers    1104 non-null   object 
 4   Company    1104 non-null   object 
 5   Title      1104 non-null   object 
 6   Adj Close  1104 non-null   float64
 7   Close      1104 non-null   float64
 8   High       1104 non-null   float64
 9   Low        1104 non-null   float64
 10  Open       1104 non-null   float64
 11  Volume     1104 non-null   int64  
dtypes: float64(5), int64(1), object(6)
memory usage: 103.6+ KB
```

Figure 5.9 Data Info

The dataset contains 1104 entries with a range index from 0 to 1103. The data types consist of: float64 (5), int64 (1), object (6), with a memory usage of 103.6+ KB.

```python
# Identify primary text column (Title detected in your file)
text_col = 'Title'  # adjust if you want a different column

# Helpers
url_re = re.compile(r'https?://\S+|www\.\S+')
html_tag_re = re.compile(r'<.*?>')
multi_space_re = re.compile(r'\s+')
dollar_ticker_re = re.compile(r'\$([A-Za-z]{1,6})\b')

def remove_control_chars(text):
    return "".join(ch for ch in text if ch.isprintable())

def basic_normalize(text):
    if pd.isna(text):
        return text
    s = str(text)
    s = unescape(s)
    s = url_re.sub(' ', s)
    s = html_tag_re.sub(' ', s)
    s = dollar_ticker_re.sub(r'\1', s)  # $AAPL -> AAPL
    s = remove_control_chars(s)
    s = multi_space_re.sub(' ', s).strip()
    return s

import string
punctuation = string.punctuation

def clean_for_textblob(s):
    s = basic_normalize(s)
    if pd.isna(s): return s
    s = s.lower()
    s = re.sub(r'[{}]+'.format(re.escape(punctuation)), ' ', s)
    s = multi_space_re.sub(' ', s).strip()
    return s

def clean_for_vader(s):
    s = basic_normalize(s)
    if pd.isna(s): return s
    s = s.replace('\u201c','"').replace('\u201d','"').replace('\u2018',"'").replace('\u2019',"'")
    return s

def clean_for_transformer(s):
    s = basic_normalize(s)
    if pd.isna(s): return s
    s = s.replace('\u201c','"').replace('\u201d','"').replace('\u2018',"'").replace('\u2019',"'")
    return s

df['clean_text_textblob'] = df[text_col].apply(clean_for_textblob)
df['clean_text_vader'] = df[text_col].apply(clean_for_vader)
df['clean_text_transformer'] = df[text_col].apply(clean_for_transformer)

# drop empty cols and duplicates
empty_cols = [c for c in df.columns if df[c].notna().sum() == 0]
df = df.drop(columns=empty_cols)
df = df.drop_duplicates()
df = df.sort_values(by="Date").reset_index(drop=True)

# Preview cleaned data
print(df[["clean_text_textblob", "clean_text_vader", "clean_text_transformer"]].head())
#df.to_csv("Final_Merged_Data_cleaned_variants.csv", index=False)
```

Figure 5.10 Data Cleaning for Each Model

The data cleaning step in Figure 5.10 includes several helper functions to normalize and clean the text data. First and foremost, **Basic normalization**: The basic_normalize function removes URLs, HTML tags, control characters, and dollar signs from ticker symbols (e.g., $AAPL

becomes AAPL), while handling multi-spaces and stripping whitespace. Next, **TextBlob cleaning**: The clean_for_textblob function builds on basic normalization by converting text to lowercase and removing all punctuation, preparing it for TextBlob sentiment analysis. Besides that, **VADER cleaning**: The clean_for_vader function extends basic normalization by replacing smart quotes with standard ones, optimizing the text for VADER sentiment analysis. Lastly, **Transformer cleaning**: The clean_for_transformer function mirrors VADER's cleaning approach, replacing smart quotes for transformer-based models. The code then applies these cleaning functions to create new columns (clean_text_textblob, clean_text_vader, clean_text_transformer) in the DataFrame. In short, it previews the cleaned data with a head view of the specified columns (Figure 5.11).

```
                                clean_text_textblob  \
0           futures dip after another nasdaq record
1      datadog gains bull on cloud migration trends
2  entegris acquires global measurement technologies
3  wimi holographic ar ai vision to drive a new w...
4         hecla mining s q2 silver output climbs 13

                                   clean_text_vader  \
0           Futures dip after another Nasdaq record
1      Datadog gains bull on cloud migration trends
2  Entegris acquires Global Measurement Technologies
3  WIMI Holographic AR+AI vision to drive a new w...
4        Hecla Mining's Q2 silver output climbs 13%

                              clean_text_transformer
0           Futures dip after another Nasdaq record
1      Datadog gains bull on cloud migration trends
2  Entegris acquires Global Measurement Technologies
3  WIMI Holographic AR+AI vision to drive a new w...
4        Hecla Mining's Q2 silver output climbs 13%
```

Figure 5.11 Clean Text for Each Models

5.3 Data Visualization (EDA)

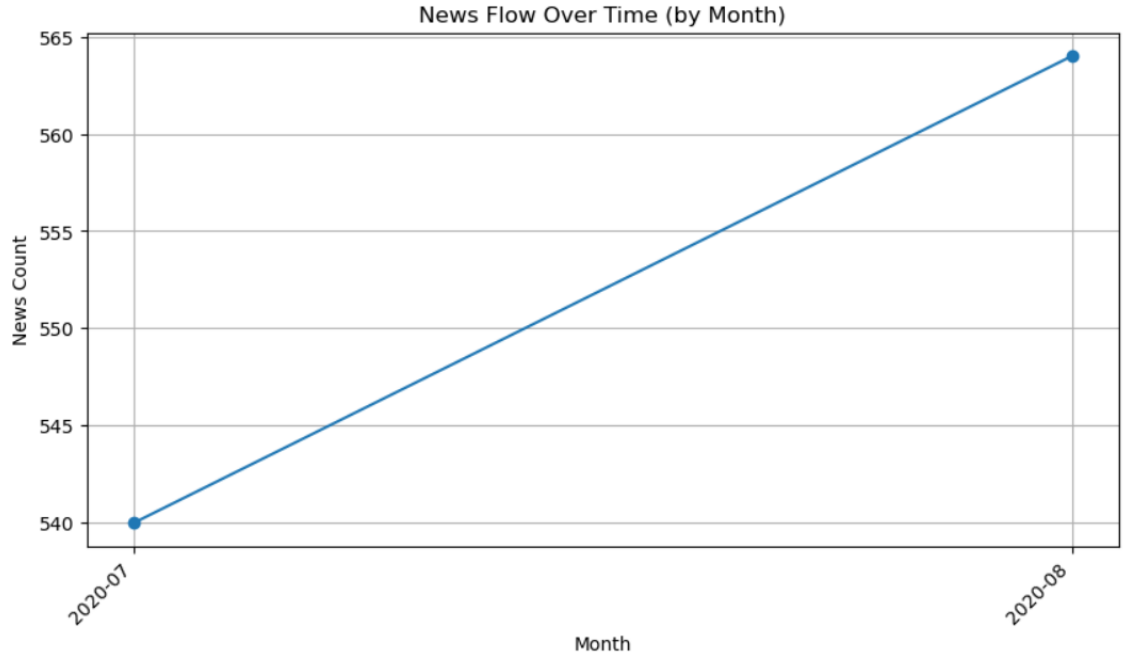


Figure 5.12 News Flow Over Time (By Month)

Figure 5.12 illustrates the distribution of news count across different months. It is generated using a line plot with markers, where the x-axis represents the months, and the y-axis indicates the number of news items.

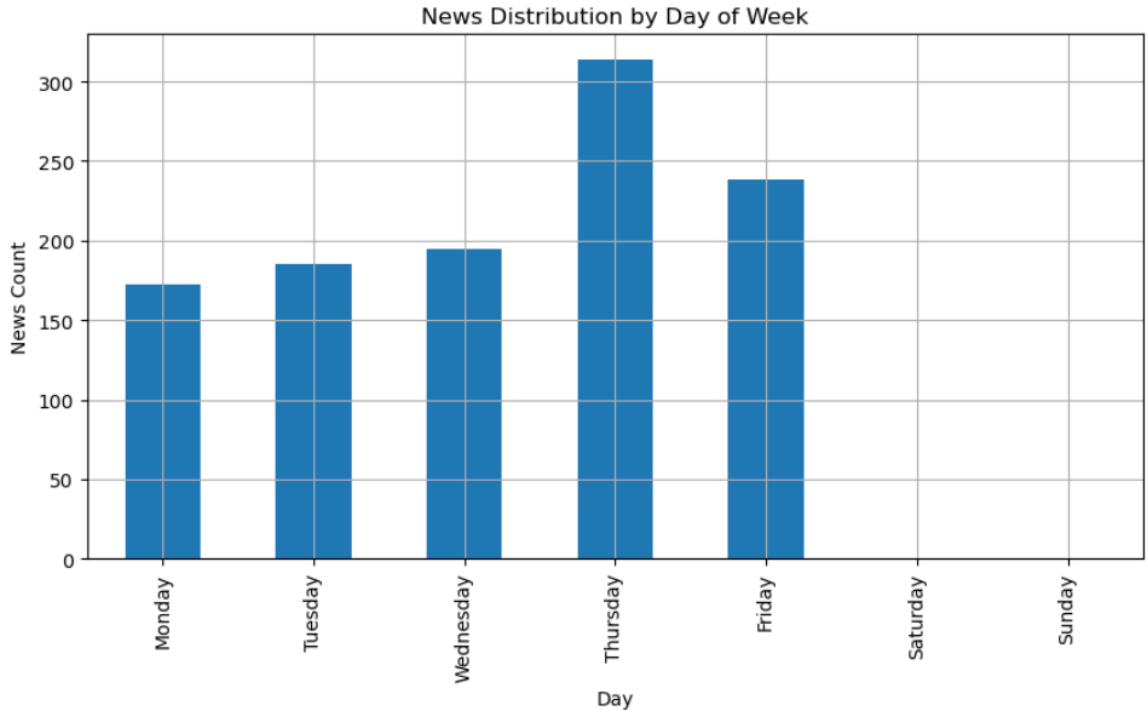


Figure 5.13 News Distribution by Day of Week

Figure 5.13 provides a bar chart representation of news counts across the days of the week. The x-axis lists the days, while the y-axis shows the news count, offering a clear visual of when news headlines are more prevalent during the week. Among all day, Thursday standing out as the peak.

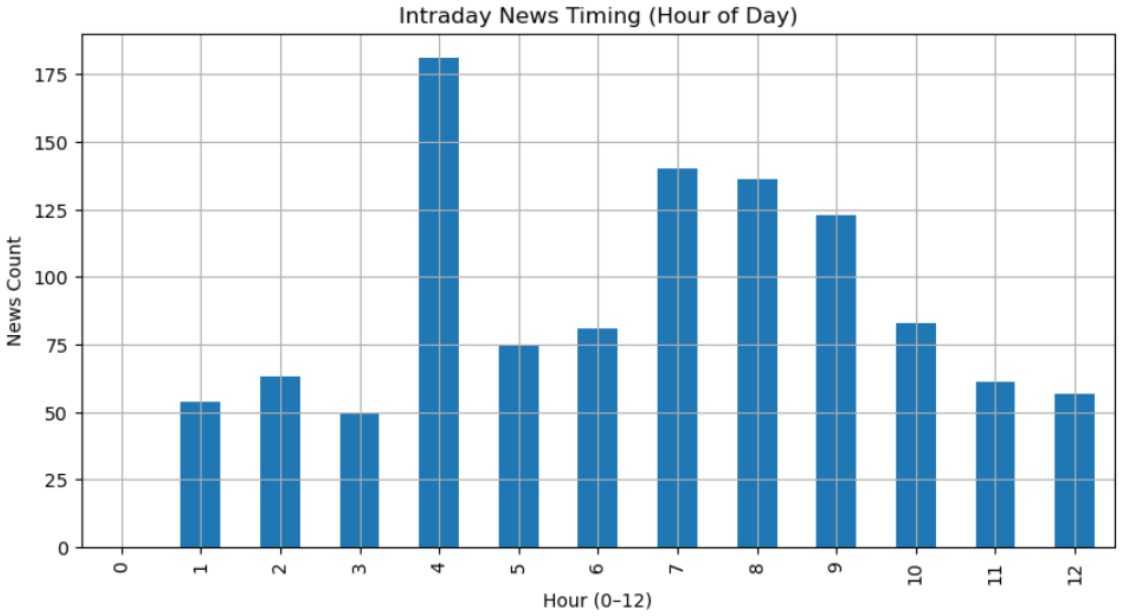


Figure 5.14 Intraday News Timing (Hour of Day)

Figure 5.14 presents a bar chart of news counts by hour. The x-axis represents the hours, and the y-axis displays the news count, providing insight into the intraday timing of news releases.

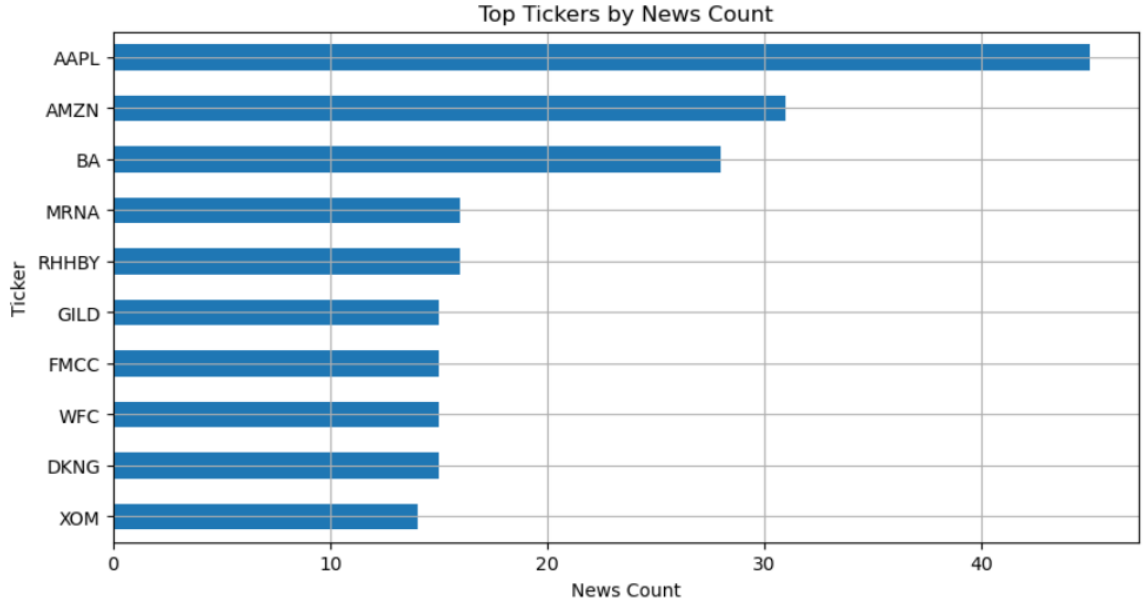


Figure 5.15 Top Tickers by News Count

Figure 5.15 is a horizontal bar chart that displays the top 10 tickers with the highest news counts. The x-axis represents the news count, while the y-axis lists the tickers. AAPL leads with the highest news count, followed by AMZN, BA, mRNA, RHHBY, GILD, FMCC, WFC, DKNK, and XOM, with their respective news counts decreasing progressively. This visualization highlights AAPL is the most covered tickers in the dataset.

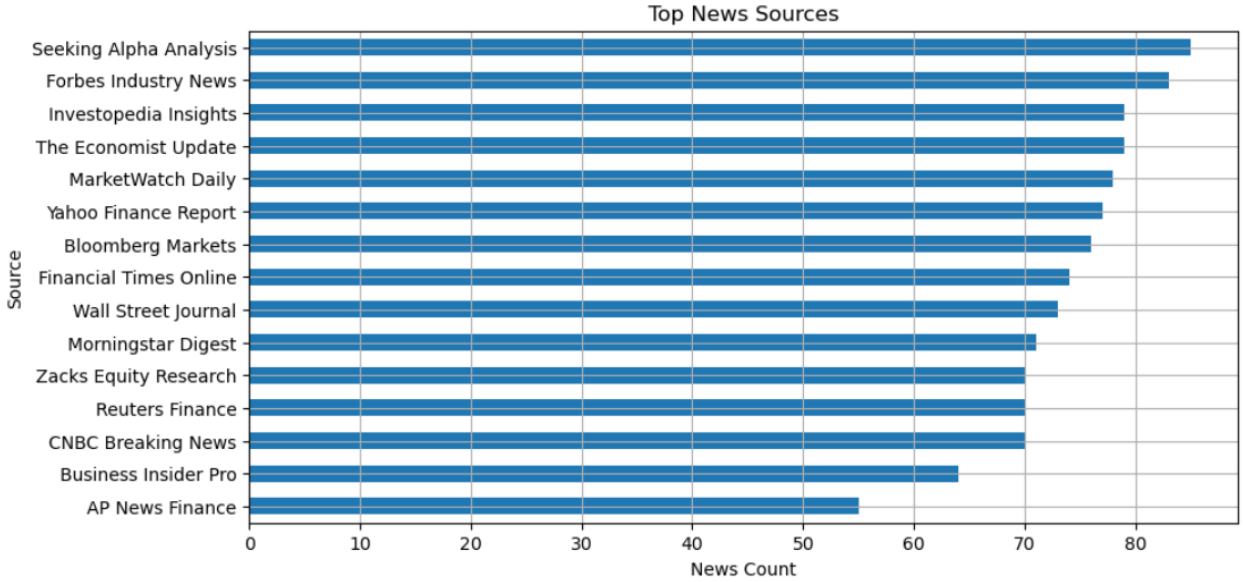


Figure 5.16 Top News Sources

Figure 5.16 is a horizontal bar chart that displays the top news sources by news count. The x-axis represents the news count, while the y-axis lists the sources. Seeking Alpha Analysis leads with the highest news count, followed by Forbes Industry News, Investopedia Insights, MarketWatch Daily, Yahoo Finance Report, Bloomberg Markets, Financial Times Online, and so on. This visualization highlights the most prominent news sources in the dataset.

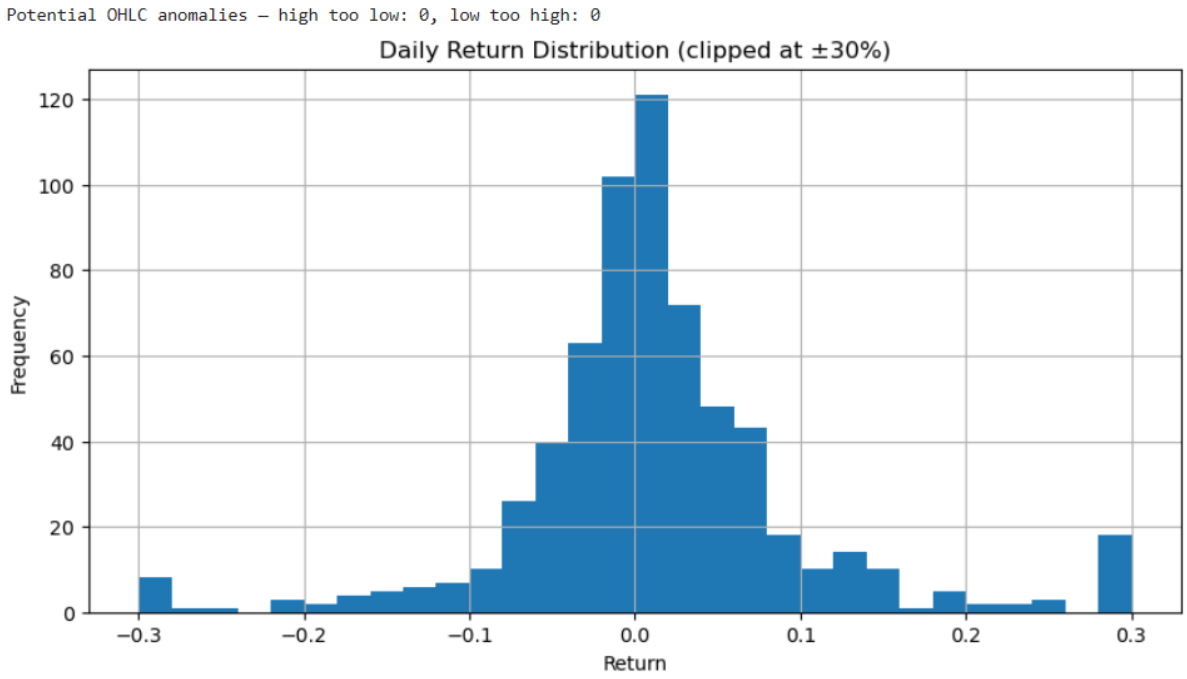


Figure 5.17 Daily Return Distribution

Figure 5.17 is a histogram that illustrates the frequency distribution of daily returns, with a note indicating no potential OHLC anomalies (high too low: 0, low too high: 0). The x-axis represents the return values, ranging from -0.3 to 0.3 in increments of 0.1, while the y-axis shows the frequency, scaling up to 120. The distribution forms a bell-shaped curve centered around 0, with the highest frequency (over 100) near 0, gradually decreasing towards both negative and positive extremes, featuring small bars at -0.3 and 0.3. This visualization highlights the concentration of daily returns near zero, suggesting low volatility in the dataset with no detected data anomalies. The reason of getting this result is because dataset lacks the second day's price for some entries.

news_bin	avg_next_day_return
(0.999, 4.0]	0.00421

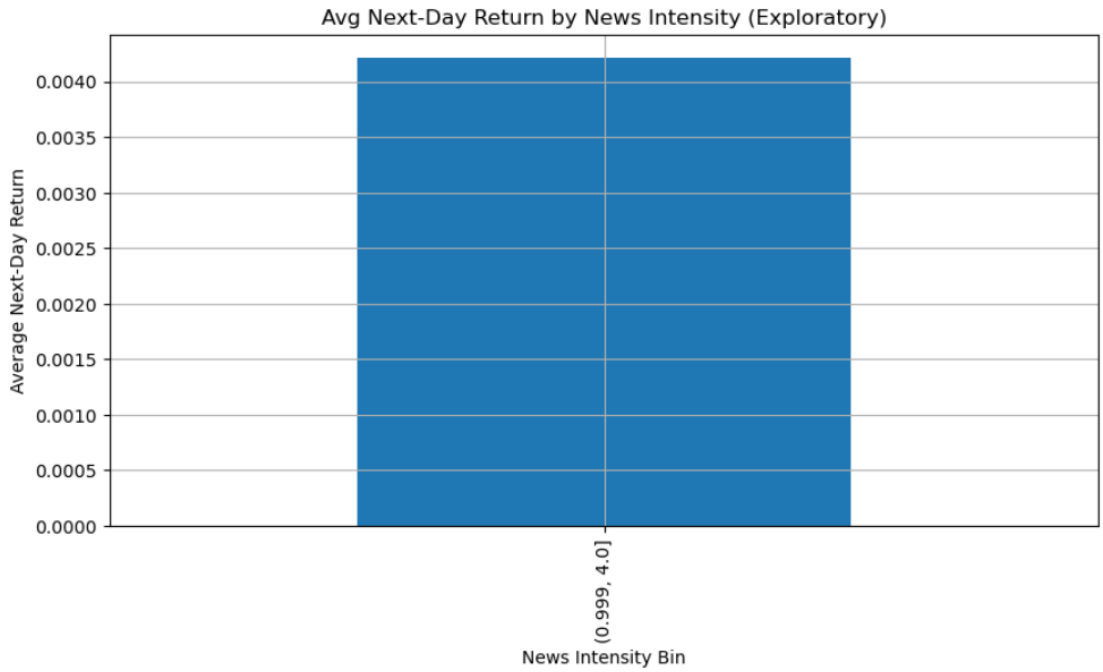


Figure 5.18 Avg Next-Day Return by News Intensity

Figure 5.18 is a bar chart that visualizes the average next-day stock returns binned by news intensity levels. The x-axis categorizes news intensity bins, with the visible bin labelled "(0.99, 4.0]" and an associated average next day return of 0.00421. The y-axis represents the average next-day return, ranging from 0.0000 to 0.0040. The bars appear uniform in height around 0.0025–0.0030 across the bins, suggesting minimal variation in returns relative to news intensity in this exploratory analysis. This flat distribution may indicate that higher news intensity does not strongly predict elevated next day returns in the dataset, due to lack of second day's price for some entries.

heavy_news	median_volume
Light/Normal News	9643300.0
Heavy News	1785500.0

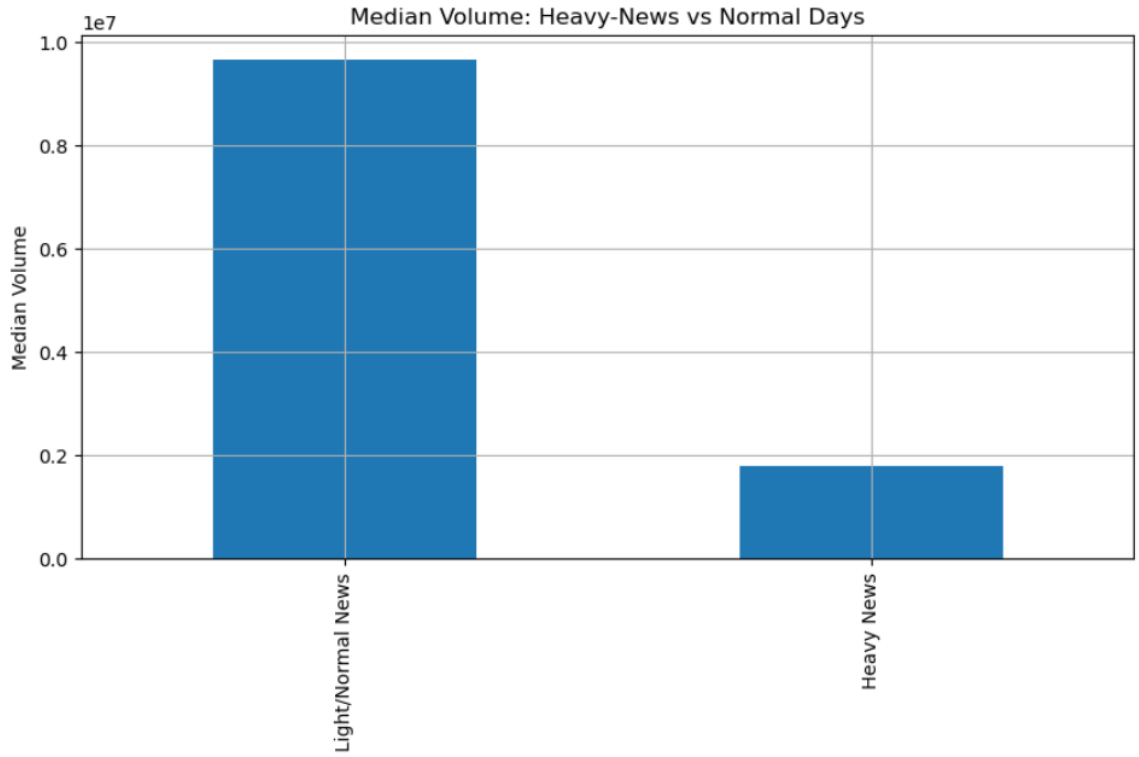


Figure 5.19 Median Volume: Heavy News vs Normal Days

Figure 5.19 is a bar chart that compares the median trading volume between days classified as "Light/Normal News" and "Heavy News." The x-axis categorizes the two types of days, while the y-axis represents the median volume, scaled from 0 to 1.0. The bar for "Light/Normal News" reaches approximately 0.96, corresponding to a median volume of 9,643,300, significantly higher than the bar for "Heavy News," which stands at about 0.18 with a median volume of 178,550. This visualization suggests that trading volume is notably lower on days with heavy news coverage compared to normal news days, potentially indicating reduced market activity or data-specific patterns during high-news periods.

5.4 Sentiment Analysis

```
!pip install transformers==4.40.0
```

```
Requirement already satisfied: transformers==4.40.0 in c:\users\clueless\anaconda3\lib\site-packages (4.40.0)
Requirement already satisfied: filelock in c:\users\clueless\anaconda3\lib\site-packages (from transformers==4.40.0) (3
Requirement already satisfied: huggingface-hub<1.0,>=0.19.3 in c:\users\clueless\anaconda3\lib\site-packages (from tran
Requirement already satisfied: numpy>=1.17 in c:\users\clueless\anaconda3\lib\site-packages (from transformers==4.40.0)
```

Figure 5.20 Library and Packages downloaded

```
df = pd.read_csv("./data/Final_Merged_Data_cleaned_variants.csv")

# Ensure required columns are present
required_cols = ["clean_text_textblob", "clean_text_vader", "clean_text_transformer"]
for col in required_cols:
    if col not in df.columns:
        raise ValueError(f"Missing column '{col}'. Ensure the cleaning step has been completed.")

# ========= Device Setup =========
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using {'GPU' if torch.cuda.is_available() else 'CPU'} - Device: {torch.cuda.get_device_name(0) if torch.cuda.is_availab
if not torch.cuda.is_available():
    print("Warning: CUDA is not available. Ensure PyTorch is installed with CUDA support and NVIDIA drivers are up to date.")

# ========= 1. TextBlob =========
def get_textblob_score(text):
    if pd.isna(text):
        return 0
    return TextBlob(str(text)).sentiment.polarity

df['TextBlob'] = df['clean_text_textblob'].apply(get_textblob_score)

# ========= 2. VADER =========
vader_analyzer = SentimentIntensityAnalyzer()

def get_vader_score(text):
    if pd.isna(text):
        return 0
    return vader_analyzer.polarity_scores(str(text))['compound']

df['Vader'] = df['clean_text_vader'].apply(get_vader_score)

# ========= 3. BERT Sentiment =========
bert_model_name = "nlptown/bert-base-multilingual-uncased-sentiment"  # Sentiment-specific BERT model
try:
    bert_tokenizer = AutoTokenizer.from_pretrained(bert_model_name)
    config = AutoConfig.from_pretrained(bert_model_name)
    print(f"Config loaded: {config.model_type}")
    bert_model = AutoModelForSequenceClassification.from_pretrained(bert_model_name).to(device)
    bert_model.eval()
    print("BERT model loaded successfully!")
except Exception as e:
    print(f"Error loading BERT model: {e}")
    raise
```

Figure 5.21 Sentiment Analysis with TextBlob, VADER, BERT, and FinBERT (Part 1)

```
def get_bert_score(text):
    if pd.isna(text):
        return 0
    inputs = bert_tokenizer(
        str(text), return_tensors="pt", truncation=True, padding=True, max_length=512
    ).to(device)
    with torch.no_grad():
        outputs = bert_model(**inputs)
        probs = torch.softmax(outputs.logits, dim=-1).cpu().numpy()[0]
    # BERT model outputs 5 classes (1-5 stars); map to -1 to 1 scale
    # 1 star = -1, 2 stars = -0.5, 3 stars = 0, 4 stars = 0.5, 5 stars = 1
    score_mapping = [-1.0, -0.5, 0.0, 0.5, 1.0]
    max_prob_index = probs.argmax()
    return float(score_mapping[max_prob_index])

df['BERT'] = df['clean_text_transformer'].apply(get_bert_score)

# ========= 4. FinBERT Sentiment =========
finbert_model_name = "yiyanghkust/finbert-tone"
try:
    finbert_tokenizer = AutoTokenizer.from_pretrained(finbert_model_name)
    finbert_model = AutoModelForSequenceClassification.from_pretrained(finbert_model_name).to(device)
    finbert_model.eval()
    print("FinBERT model loaded successfully!")
except Exception as e:
    print(f"Error loading FinBERT model: {e}")
    raise

def get_finbert_score(text):
    if pd.isna(text):
        return 0
    inputs = finbert_tokenizer(
        str(text), return_tensors="pt", truncation=True, padding=True, max_length=512
    ).to(device)
    with torch.no_grad():
        outputs = finbert_model(**inputs)
        probs = torch.softmax(outputs.logits, dim=-1).cpu().numpy()[0]
    # Positive probability minus negative probability
    return float(probs[2] - probs[0])

df['FinBERT'] = df['clean_text_transformer'].apply(get_finbert_score)

# ========= Save Results =========
#output_file = "Final_Merged_Data_with_Sentiments.csv"
```

Figure 5.22 Sentiment Analysis with TextBlob, VADER, BERT, and FinBERT (Part 2)

Figure 5.21 and 5.22 is designed to perform sentiment analysis on a dataset of cleaned stock news headlines stored in a CSV file ("Final_Merged_Data_cleaned_variants.csv") using multiple sentiment analysis tools. It begins by importing necessary libraries, including pandas for data handling, torch for GPU support, and various sentiment analysis frameworks like TextBlob, VADER, and transformers for BERT-based models. The code checks for required columns ("clean_text_textblob," "clean_text_vader," "clean_text_transformer") and sets up a device (GPU) for model processing. It then applies four sentiment scoring methods: TextBlob

calculates polarity scores based on cleaned text, VADER provides compound sentiment scores, a multilingual BERT model from nlptown maps text to a -1 to 1 scale based on star ratings, and FinBERT from yiyanghkust computes sentiment as the difference between positive and negative probabilities. Each method handles missing values by returning 0, and the results are added as new columns ("TextBlob," "Vader," "BERT," "FinBERT") to the DataFrame. Finally, the code is structured to save the enriched DataFrame to a new CSV file, ensuring the sentiment-analyzed data can be stored for further use.

Sentiment Model Descriptions

Model	Output	Range / Labels	Detailed Description
TextBlob	`Polarity`	`-1.0` to `+1.0`	TextBlob uses NLP techniques to compute the sentiment polarity of text. A value of -1.0 indicates very negative sentiment, +1.0 is very positive. Best for formal or well-structured English sentences.
VADER	`Compound` score	`-1.0` (very negative) to `+1.0` (very positive)	VADER (Valence Aware Dictionary for sEntiment Reasoning) is a rule-based sentiment tool optimized for short, informal text like social media. The compound score is a normalized sentiment score: 🔴 `<` `-0.05` : **Negative** 🟡 `-0.05` to `0.05` : **Neutral** 🟢 `>` `0.05` : **Positive**
BERT	`Label`	`-1.0` (very negative) to `+1.0` (very positive)	BERT (`nlptown/bert-base-multilingual-uncased-sentiment`) is a multilingual BERT model fine-tuned for sentiment analysis. It outputs a 1–5 star rating, mapped to -1.0 (1 star, very negative) to +1.0 (5 stars, very positive), with 0.0 as neutral.
FinBERT	`Label`	`-1.0` (very negative) to `+1.0` (very positive)	FinBERT is a version of BERT fine-tuned for financial text. It provides a sentiment score ranging from -1.0 (very negative) to +1.0 (very positive), reflecting financial news and reports sentiment, with 0 being neutral.

Figure 5.23 Sentiment Model Description

Figure 5.23 provides an overview of four sentiment analysis models which is TextBlob, VADER, BERT, and FinBERT, along with their output types, value ranges, and detailed descriptions.

```python
# Function to convert scores into sentiment labels
def get_sentiment_label(value, model):
    if model == "TextBlob":
        if value > 0.05:
            return "Positive"
        elif value < -0.05:
            return "Negative"
        else:
            return "Neutral"
    elif model == "Vader":
        if value >= 0.05:
            return "Positive"
        elif value <= -0.05:
            return "Negative"
        else:
            return "Neutral"
    elif model in ["BERT", "FinBERT"]:
        if value > 0:
            return "Positive"
        elif value < 0:
            return "Negative"
        else:
            return "Neutral"

# Apply conversion for each model
df["TextBlob_label"] = df["TextBlob"].apply(lambda x: get_sentiment_label(x, "TextBlob"))
df["Vader_label"] = df["Vader"].apply(lambda x: get_sentiment_label(x, "Vader"))
df["BERT_label"] = df["BERT"].apply(lambda x: get_sentiment_label(x, "BERT"))
df["FinBERT_label"] = df["FinBERT"].apply(lambda x: get_sentiment_label(x, "FinBERT"))

def final_result(row):
    sentiments = [row["TextBlob_label"], row["Vader_label"], row["BERT_label"], row["FinBERT_label"]]
    counts = pd.Series(sentiments).value_counts()
    return counts.idxmax()  # sentiment with majority vote

df["Final Result"] = df.apply(final_result, axis=1)

# Save to new CSV file
#output_path = "Final_Merged_Data_with_Final_Result.csv"
#df.to_csv(output_path, index=False)
#print(f"✅ File saved as: {output_path}")
```

Figure 5.24 Sentiment Labelling and Majority Vote Aggregation

Figure 5.24 defines a sentiment analysis pipeline that converts numerical sentiment scores into categorical labels (“Positive,” “Negative,” or “Neutral”) for four different models—TextBlob, VADER, BERT, and FinBERT—using the get_sentiment_label function. This function applies model-specific thresholds: for TextBlob and VADER, values above 0.05 are “Positive,” below -0.05 are “Negative,” and others are “Neutral,” while for BERT and FinBERT, the decision hinges on whether the score is positive, negative, or zero. New columns (“TextBlob_label,” “Vader_label,” “BERT_label,” “FinBERT_label”) are created by applying this function to the respective score columns. A final_result function then determines the dominant sentiment for each row by counting the occurrences of each label across the four models and selecting the

majority vote, storing the result in a “Final Result” column. The code is set up to save the updated DataFrame to a new CSV file (“Final_Merged_Data_with_Final_Result.csv”), ensuring the final sentiment classification is preserved for further analysis.

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# Ensure dataset is strictly sorted by date before splitting
df = df.sort_values('Date').reset_index(drop=True)

from sklearn.model_selection import train_test_split

df = pd.read_csv("./Final_Merged_Data_with_Final_Result.csv")

# --- Step 1: Sort by date ---
df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
df = df.sort_values("Date").reset_index(drop=True)

# --- Step 2: 80:20 split ---
split_index = int(len(df) * 0.8)
train = df.iloc[:split_index].copy()
test  = df.iloc[split_index:].copy()

print("Train size:", len(train))
print("Test size:", len(test))
```

```
Train size: 883
Test size: 221
```

Figure 5.25 Data Splitting

Figure 5.25 outlines a data preprocessing and splitting process for a machine learning task using a dataset stored in “Final_Merged_Data_with_Final_Result.csv.” It begins by importing necessary modules from scikit-learn, including SimpleImputer, StandardScaler, and Pipeline for data preprocessing, and train_test_split for data splitting. The dataset is first loaded into a pandas DataFrame and sorted by the “Date” column, with the index reset to ensure a strictly ordered sequence, which is critical for time-series analysis. The “Date” column is converted to datetime format with errors coerced to handle any inconsistencies. The data is then split into training and test sets using an 80:20 ratio, where the split index is calculated as 80% of the dataset length, and the DataFrame is divided accordingly into train and test subsets using integer location indexing. Finally, the code prints the sizes of the training (883 entries) and test (221 entries) sets, confirming the split proportions.

5.5 Feature Engineering

```
from sklearn.preprocessing import LabelEncoder

# Feature Engineering (Safe for Time Series)
df = df.sort_values("Date").reset_index(drop=True)

# Create lag features (use only past data)
df["return"] = df["Close"].pct_change()   # daily return
df["lag1"] = df["return"].shift(1)
df["lag2"] = df["return"].shift(2)
df["lag3"] = df["return"].shift(3)

# Rolling window features (past averages/volatility)
df["rolling_mean_3"] = df["return"].shift(1).rolling(window=3).mean()
df["rolling_mean_7"] = df["return"].shift(1).rolling(window=7).mean()
df["rolling_std_3"] = df["return"].shift(1).rolling(window=3).std()
df["rolling_std_7"] = df["return"].shift(1).rolling(window=7).std()

# Volume features
df["vol_lag1"] = df["Volume"].shift(1)
df["vol_mean_3"] = df["Volume"].shift(1).rolling(window=3).mean()
df["vol_std_3"] = df["Volume"].shift(1).rolling(window=3).std()

# Technical-style features
df["high_low_range"] = (df["High"] - df["Low"]) / df["Close"]
df["open_close_diff"] = (df["Open"] - df["Close"]) / df["Close"]

# Encode sentiment feature (Neutral / Positive / Negative → numeric)
label_map = {"Negative": -1, "Neutral": 0, "Positive": 1}
df["sentiment_score"] = df["Final Result"].map(label_map)

# Define the target variable (trend: 1 = up, 0 = down)
df["trend"] = (df["return"] > 0).astype(int)
```

Figure 5.26 Feature Engineering (Part 1)

```
# Drop rows with NaNs created by shifting/rolling
df = df.dropna().reset_index(drop=True)

# Final features and target
feature_cols = [
    "lag1", "lag2", "lag3",
    "rolling_mean_3", "rolling_mean_7",
    "rolling_std_3", "rolling_std_7",
    "vol_lag1", "vol_mean_3", "vol_std_3",
    "high_low_range", "open_close_diff",
    "sentiment_score"]

X = df[feature_cols]
y = df["trend"]

print("Final feature shape:", X.shape)
print("Final target distribution:\n", y.value_counts(normalize=True))

df.to_csv("Final_Dataset.csv", index=False)
print("Saved to Final_Dataset.csv")
```

```
Final feature shape: (1096, 13)
Final target distribution:
 trend
0    0.508212
1    0.491788
Name: proportion, dtype: float64
Saved to Final_Dataset.csv
```

Figure 5.27 Feature Engineering (Part 2)

Figure 5.26 and 5.27 performs feature engineering and data preparation for a time-series analysis using a panda DataFrame containing stock market data. It starts by sorting the DataFrame by the "Date" column and resetting the index to ensure a chronological order. Lag features ("lag1," "lag2," "lag3") are created using the percentage change in the "Close" price to calculate daily returns, shifted by one, two, and three days respectively, ensuring only past data is used. Rolling window features ("rolling_mean_3," "rolling_mean_7," "rolling_std_3," "rolling_std_7") compute the mean and standard deviation of returns over 3- and 7-day windows, also shifted to avoid future leakage. Volume-related features ("vol_lag1," "vol_mean_3," "vol_std_3") are derived from the "Volume" column with a one-day lag and 3-day rolling statistics. Technical indicators like "high_low_range" (the range between high and low prices relative to close) and "open_close_diff" (the difference between open and close prices relative to close) are added. The sentiment label "Final Result" is encoded into a numeric "sentiment_score", and the target variable "trend" is defined as 1 for an upward return and 0 for a downward return based on the "return" column. After dropping rows with NaN values

from shifting and rolling operations, the final feature set (X) and target (y) are extracted, and their shapes and target distribution are printed.

5.6 Modelling

```
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
import numpy as np

# TimeSeriesSplit for walk-forward validation
tscv = TimeSeriesSplit(n_splits=5)
accuracies = []

fold = 1
for train_idx, test_idx in tscv.split(X):
    X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

    # Train model
    model = RandomForestClassifier(random_state=42)
    model.fit(X_train, y_train)

    # Predict
    y_pred = model.predict(X_test)

    # Evaluate
    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)

    print(f"Fold {fold} Accuracy: {acc:.4f}")
    print(classification_report(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("-"*50)
    fold += 1

print("Average Walk-Forward Accuracy:", np.mean(accuracies))
```

Figure 5.28 Time Series Cross-Validation with Random Forest Classifier

```
Fold 1 Accuracy: 0.6868
              precision    recall  f1-score   support

           0       0.71      0.67      0.69        94
           1       0.67      0.70      0.69        88

    accuracy                           0.69       182
   macro avg       0.69      0.69      0.69       182
weighted avg       0.69      0.69      0.69       182

Confusion Matrix:
 [[63 31]
 [26 62]]
--------------------------------------------------
Fold 2 Accuracy: 0.6923
              precision    recall  f1-score   support

           0       0.69      0.78      0.73        98
           1       0.69      0.60      0.64        84

    accuracy                           0.69       182
   macro avg       0.69      0.69      0.69       182
weighted avg       0.69      0.69      0.69       182

Confusion Matrix:
 [[76 22]
 [34 50]]
--------------------------------------------------
Fold 3 Accuracy: 0.6648
              precision    recall  f1-score   support

           0       0.65      0.72      0.68        92
           1       0.68      0.61      0.64        90

    accuracy                           0.66       182
   macro avg       0.67      0.66      0.66       182
weighted avg       0.67      0.66      0.66       182

Confusion Matrix:
 [[66 26]
 [35 55]]
```

Figure 5.29 5-Fold Accuracy (Part 1)

```
Fold 4 Accuracy: 0.7473
              precision    recall  f1-score   support

           0       0.70      0.84      0.77        89
           1       0.81      0.66      0.73        93

    accuracy                           0.75       182
   macro avg       0.76      0.75      0.75       182
weighted avg       0.76      0.75      0.75       182

Confusion Matrix:
 [[75 14]
 [32 61]]
--------------------------------------------------
Fold 5 Accuracy: 0.6758
              precision    recall  f1-score   support

           0       0.64      0.72      0.68        86
           1       0.72      0.64      0.67        96

    accuracy                           0.68       182
   macro avg       0.68      0.68      0.68       182
weighted avg       0.68      0.68      0.68       182

Confusion Matrix:
 [[62 24]
 [35 61]]
--------------------------------------------------
Average Walk-Forward Accuracy: 0.6934065934065934
```

Figure 5.30 5-Fold Accuracy (Part 2)

Figure 5.28 implements a time-series cross-validation process to evaluate a Random Forest Classifier model on sequential data. It uses TimeSeriesSplit with 5 splits to perform walk-forward validation, ensuring the temporal order of the data is preserved. The process iterates over the splits, where for each fold, it splits the feature matrix X and target vector y into training and test sets based on the indices provided by tscv.split(X). A RandomForestClassifier with a fixed random state of 42 is trained on the training data and used to predict the test set. The model's performance is evaluated using accuracy score, detailed classification report, and confusion matrix, with results printed for each fold.

```
# Holdout split (last 20%)
split_point = int(len(X) * 0.8)
X_train, X_holdout = X.iloc[:split_point], X.iloc[split_point:]
y_train, y_holdout = y.iloc[:split_point], y.iloc[split_point:]

model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

y_pred_holdout = model.predict(X_holdout)

print("Holdout Accuracy:", accuracy_score(y_holdout, y_pred_holdout))
print(classification_report(y_holdout, y_pred_holdout))
print("Confusion Matrix:\n", confusion_matrix(y_holdout, y_pred_holdout))
```

```
Holdout Accuracy: 0.6863636363636364
              precision    recall  f1-score   support

           0       0.64      0.75      0.69       103
           1       0.74      0.63      0.68       117

    accuracy                           0.69       220
   macro avg       0.69      0.69      0.69       220
weighted avg       0.69      0.69      0.69       220

Confusion Matrix:
 [[77 26]
 [43 74]]
```

Figure 5.31 Holdout Split

Figure 5.31 performs a holdout split and evaluation using a Random Forest Classifier on a dataset. It splits the feature matrix X and target vector y into training and holdout sets, with the split point set at 80% of the data length, reserving the last 20% for testing. The RandomForestClassifier is initialized with a fixed random state of 42 for reproducibility and trained on the training data (X_train, y_train). Predictions are made on the holdout set (X_holdout), and the model's performance is assessed by printing the holdout accuracy using accuracy_score, a detailed classification report, and a confusion matrix. This approach provides a straightforward evaluation of the model's generalization ability on unseen data from the end of the time series.

```python
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

# Define models dictionary with all 5 models
models = {
    "RandomForest": Pipeline([
        ("imputer", SimpleImputer(strategy="mean")),
        ("model", RandomForestClassifier(random_state=42))
    ]),
    "XGBoost": Pipeline([
        ("imputer", SimpleImputer(strategy="mean")),
        ("model", XGBClassifier(
            use_label_encoder=False,     # ✅ suppress warning
            eval_metric="logloss",
            random_state=42
        ))
    ]),
    "LightGBM": Pipeline([
        ("imputer", SimpleImputer(strategy="mean")),
        ("model", LGBMClassifier(
            random_state=42,
            force_col_wise=True         # ✅ proper placement
        ))
    ]),
    "LogisticRegression": Pipeline([
        ("imputer", SimpleImputer(strategy="mean")),
        ("scaler", StandardScaler(with_mean=False)),
        ("model", LogisticRegression(max_iter=1000, random_state=42))
    ]),
    "SVM": Pipeline([
        ("imputer", SimpleImputer(strategy="mean")),
        ("scaler", StandardScaler(with_mean=False)),
        ("model", SVC(probability=True, random_state=42))
    ])
```

Figure 5.32 Multi-Model Pipeline

Figure 5.32 defines a dictionary of machine learning models, each wrapped in a Pipeline from scikit-learn, to handle data preprocessing and model training for a classification task. Five models are included: "RandomForest" using RandomForestClassifier, "XGBoost" with XGBClassifier configured to suppress label encoder warnings and use logloss as the evaluation

metric, "LightGBM" with LGBMClassifier set for column-wise processing to optimize performance, "LogisticRegression" with LogisticRegression including a StandardScaler for feature scaling (without centering) and a high iteration limit, and "SVM" with SVC enabling probability estimates and a scaler similar to LogisticRegression. Each pipeline starts with a SimpleImputer using the mean strategy to handle missing values, ensuring robust preprocessing before model fitting, with all models initialized with a random state of 42 for reproducibility.

```
# ✅ Keep only numeric columns (drop Date/Timestamp/text)
X_clean = X.select_dtypes(include=[np.number])
y_clean = y

# Time series split
tscv = TimeSeriesSplit(n_splits=5)

# Models with scaling where needed
models = {
    "LogisticRegression": Pipeline([
        ("imputer", SimpleImputer(strategy="mean")),
        ("model", LogisticRegression(
            max_iter=1000, solver="lbfgs", random_state=42))
    ]),
    "SVM": Pipeline([
        ("imputer", SimpleImputer(strategy="mean")),
        ("scaler", StandardScaler()),
        ("model", SVC(kernel="rbf", probability=True, random_state=42))
    ]),
    "RandomForest": Pipeline([
        ("imputer", SimpleImputer(strategy="mean")),
        ("model", RandomForestClassifier(
            max_depth=5, min_samples_leaf=4, min_samples_split=10,
            n_estimators=200, random_state=42
        ))
    ]),
    "XGBoost": Pipeline([
        ("imputer", SimpleImputer(strategy="mean")),
        ("model", XGBClassifier(
            n_estimators=200, max_depth=5, learning_rate=0.1,
            subsample=0.8, colsample_bytree=0.8, random_state=42,
            use_label_encoder=False, eval_metric="logloss"
        ))
    ]),
```

Figure 5.33 Cross Validation (Part 1)

```
    "LightGBM": Pipeline([
        ("imputer", SimpleImputer(strategy="mean")),
        ("model", LGBMClassifier(
            n_estimators=200, max_depth=-1, learning_rate=0.1,
            subsample=0.8, colsample_bytree=0.8, random_state=42,
            force_col_wise=True   # ✅ Added to reduce overhead
        ))
    ])
}

# Run cross-validation
results = {}
for name, model in models.items():
    scores = cross_val_score(model, X_clean, y_clean, cv=tscv, scoring="accuracy", error_score="raise")
    results[name] = {
        "Mean Accuracy": np.mean(scores),
        "Std Dev": np.std(scores),
        "All Splits": scores
    }

# Print results
for model_name, res in results.items():
    print(f"{model_name}:")
    print(f"  Mean Accuracy: {res['Mean Accuracy']:.4f}")
    print(f"  Std Dev: {res['Std Dev']:.4f}")
    print(f"  Split Accuracies: {res['All Splits']}")
    print("-"*50)
```

Figure 5.34 Cross Validation (Part 2)

```
LogisticRegression:
  Mean Accuracy: 0.4901
  Std Dev: 0.0277
  Split Accuracies: [0.44505495 0.50549451 0.49450549 0.47802198 0.52747253]
--------------------------------------------------
SVM:
  Mean Accuracy: 0.5352
  Std Dev: 0.0279
  Split Accuracies: [0.57142857 0.50549451 0.56593407 0.52197802 0.51098901]
--------------------------------------------------
RandomForest:
  Mean Accuracy: 0.6923
  Std Dev: 0.0203
  Split Accuracies: [0.70879121 0.67582418 0.66483516 0.71978022 0.69230769]
--------------------------------------------------
XGBoost:
  Mean Accuracy: 0.6769
  Std Dev: 0.0281
  Split Accuracies: [0.66483516 0.64835165 0.67032967 0.73076923 0.67032967]
--------------------------------------------------
LightGBM:
  Mean Accuracy: 0.6626
  Std Dev: 0.0331
  Split Accuracies: [0.64835165 0.66483516 0.64285714 0.72527473 0.63186813]
--------------------------------------------------
```

Figure 5.35 Cross Validation's Result

Figure 5.33 and 5.34 implements a comprehensive evaluation of five machine learning models using time-series cross-validation on a cleaned dataset. It begins by importing necessary libraries, including scikit-learn modules for cross-validation, metrics, and pipelines, as well as XGBoost and LightGBM for additional models. The feature matrix X is filtered to include only numeric columns using select_dtypes, and the target vector y is retained as y_clean. A TimeSeriesSplit with 5 splits is defined to preserve the temporal order of the data. Five models are configured within pipelines: "LogisticRegression" with a mean imputer, "SVM" with a scaler and RBF kernel, "RandomForest" with tuned hyperparameters (max depth, min samples, etc.), "XGBoost" with specific estimators and learning rate settings, and "LightGBM" with similar tuning and forced column-wise processing. Cross-validation scores are computed using cross_val_score with accuracy as the metric, and results (mean accuracy, standard deviation, and individual split scores) are stored in a dictionary.

```
param_distributions = {
    "RandomForest": {
        "model__n_estimators": [100, 200, 300],
        "model__max_depth": [3, 5, 7, None],
        "model__min_samples_split": [2, 5, 10],
        "model__min_samples_leaf": [1, 2, 4]
    },
    "LogisticRegression": {
        "model__C": [0.01, 0.1, 1, 10, 100],
        "model__penalty": ["l2"],
        "model__solver": ["lbfgs"]
    },
    "SVM": {
        "model__C": [0.1, 1, 10],
        "model__gamma": ["scale", "auto"],
        "model__kernel": ["rbf", "poly"]
    },
    "XGBoost": {
        "model__n_estimators": [100, 200, 300],
        "model__max_depth": [3, 5, 7],
        "model__learning_rate": [0.01, 0.05, 0.1, 0.2],
        "model__subsample": [0.6, 0.8, 1.0],
        "model__colsample_bytree": [0.6, 0.8, 1.0]
    },
    "LightGBM": {
        "model__n_estimators": [100, 200, 300],
        "model__max_depth": [-1, 5, 10],
        "model__learning_rate": [0.01, 0.05, 0.1, 0.2],
        "model__subsample": [0.6, 0.8, 1.0],
        "model__colsample_bytree": [0.6, 0.8, 1.0]
    }
}

# -------------------------
# Pipelines
# -------------------------
pipelines = {
    "LogisticRegression": Pipeline([
        ("imputer", SimpleImputer(strategy="mean")),
        ("model", LogisticRegression(max_iter=1000, random_state=42))
    ]),
    "SVM": Pipeline([
        ("imputer", SimpleImputer(strategy="mean")),
        ("scaler", StandardScaler()),
```

Figure 5.36 Hyperparameter Tuning with RandomizedSearchCV (Part 1)

```
    "RandomForest": Pipeline([
        ("imputer", SimpleImputer(strategy="mean")),
        ("model", RandomForestClassifier(random_state=42))
    ]),
    "XGBoost": Pipeline([
        ("imputer", SimpleImputer(strategy="mean")),
        ("model", XGBClassifier(use_label_encoder=False, eval_metric="logloss", random_state=42))
    ]),
    "LightGBM": Pipeline([
        ("imputer", SimpleImputer(strategy="mean")),
        ("model", LGBMClassifier(random_state=42, force_col_wise=True))
    ])
}

# -------------------------
# Run RandomizedSearchCV
# -------------------------
results = {}
for name, pipeline in pipelines.items():
    print(f"Running RandomizedSearch for {name}...")

    search = RandomizedSearchCV(
        pipeline,
        param_distributions=param_distributions[name],
        n_iter=10,
        scoring="accuracy",
        cv=tscv,
        random_state=42,
        n_jobs=-1,
        return_train_score=False
    )

    search.fit(X_clean, y_clean)  # fits models only inside CV

    # collect cv results (not best score)
    split_scores = [search.cv_results_[f"split{i}_test_score"] for i in range(tscv.get_n_splits())]
    split_scores = np.array(split_scores).mean(axis=1)  # avg across param trials

    results[name] = {
        "Mean Accuracy": split_scores.mean(),
        "Std Dev": split_scores.std(),
        "Split Accuracies": split_scores
    }

# -------------------------
# Print scores only
# -------------------------
for model_name, res in results.items():
    print(f"{model_name}:")
    print(f"  Mean Accuracy: {res['Mean Accuracy']:.4f}")
    print(f"  Std Dev: {res['Std Dev']:.4f}")
    print(f"  Split Accuracies: {res['Split Accuracies']}")
    print("-"*50)
```

Figure 5.37 Hyperparameter Tuning with RandomizedSearchCV (Part 2)

```
LogisticRegression:
  Mean Accuracy: 0.4901
  Std Dev: 0.0277
  Split Accuracies: [0.44505495 0.50549451 0.49450549 0.47802198 0.52747253]
--------------------------------------------------
SVM:
  Mean Accuracy: 0.5135
  Std Dev: 0.0155
  Split Accuracies: [0.53516484 0.52032967 0.52032967 0.49285714 0.4989011 ]
--------------------------------------------------
RandomForest:
  Mean Accuracy: 0.6922
  Std Dev: 0.0231
  Split Accuracies: [0.71373626 0.68406593 0.6521978  0.71483516 0.69615385]
--------------------------------------------------
XGBoost:
  Mean Accuracy: 0.6829
  Std Dev: 0.0248
  Split Accuracies: [0.68681319 0.66978022 0.6521978  0.72692308 0.67857143]
--------------------------------------------------
LightGBM:
  Mean Accuracy: 0.6836
  Std Dev: 0.0259
  Split Accuracies: [0.66483516 0.66318681 0.66978022 0.73241758 0.68791209]
--------------------------------------------------
```

Figure 5.38 RandomizedSearchCV's Result

Figure 5.35 and 5.36 conducts hyperparameter optimization using RandomizedSearchCV on five machine learning pipelines (LogisticRegression, SVM, RandomForest, XGBoost, and LightGBM) tailored for time-series data. It starts by selecting only numeric features from X into X_clean and retaining y as y_clean, then initializes a TimeSeriesSplit with 5 folds to maintain temporal integrity. Hyperparameter distributions are defined in a dictionary (param_distributions) specific to each model, covering key parameters like number of estimators, depth, learning rates, and regularization strengths. Pipelines are constructed with imputers for missing values and scalers where necessary (e.g., for SVM), ensuring consistent preprocessing. For each model, a RandomizedSearchCV is executed with 10 iterations, accuracy scoring, the time-series CV, and parallel jobs, fitting only within the CV folds to avoid leakage. Results are derived by extracting test scores per split from cv_results_, averaging them across trials per split to compute mean accuracy and standard deviation per model, and storing split accuracies for detailed inspection.

```python
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

# Logistic Regression tuning
log_params = {
    "model__C": [0.01, 0.1, 1, 10],
    "model__penalty": ["l2"],
    "model__solver": ["lbfgs", "saga"]
}
log_grid = GridSearchCV(models["LogisticRegression"], log_params, cv=tscv, scoring="accuracy", n_jobs=-1)
log_grid.fit(X_train, y_train)
print("Best Logistic Params:", log_grid.best_params_)
print("Best Logistic Score:", log_grid.best_score_)
print("--------------------------------------------------")

# SVM tuning
svm_params = {
    "model__C": [0.1, 1, 10],
    "model__gamma": ["scale", 0.1, 0.01],
    "model__kernel": ["rbf", "poly"]
}
svm_grid = GridSearchCV(models["SVM"], svm_params, cv=tscv, scoring="accuracy", n_jobs=-1)
svm_grid.fit(X_train, y_train)
print("Best SVM Params:", svm_grid.best_params_)
print("Best SVM Score:", svm_grid.best_score_)
print("--------------------------------------------------")

# Random Forest tuning
rf_params = {
    "model__n_estimators": [100, 200, 300],
    "model__max_depth": [None, 5, 10, 20],
    "model__min_samples_split": [2, 5, 10],
    "model__min_samples_leaf": [1, 2, 4],
```

Figure 5.39 Grid Search Hyperparameters (Part 1)

```
    "model__max_features": ["sqrt", "log2"]
}
rf_grid = GridSearchCV(models["RandomForest"], rf_params, cv=tscv, scoring="accuracy", n_jobs=-1)
rf_grid.fit(X_train, y_train)
print("Best RF Params:", rf_grid.best_params_)
print("Best RF Score:", rf_grid.best_score_)
print("-----------------------------------------------------")

# XGBoost tuning
xgb_params = {
    "model__n_estimators": [100, 200],
    "model__max_depth": [3, 5, 7],
    "model__learning_rate": [0.01, 0.1, 0.2],
    "model__subsample": [0.8, 1.0]
}
xgb_grid = GridSearchCV(models["XGBoost"], xgb_params, cv=tscv, scoring="accuracy", n_jobs=-1)
xgb_grid.fit(X_train, y_train)
print("Best XGB Params:", xgb_grid.best_params_)
print("Best XGB Score:", xgb_grid.best_score_)
print("-----------------------------------------------------")

# LightGBM tuning
lgb_params = {
    "model__n_estimators": [100, 200],
    "model__max_depth": [-1, 5, 10],
    "model__learning_rate": [0.01, 0.1, 0.2],
    "model__num_leaves": [31, 50, 100]
}
lgb_grid = GridSearchCV(models["LightGBM"], lgb_params, cv=tscv, scoring="accuracy", n_jobs=-1)
lgb_grid.fit(X_train, y_train)
print("Best LGBM Params:", lgb_grid.best_params_)
print("Best LGBM Score:", lgb_grid.best_score_)
```

Figure 5.40 Grid Search Hyperparameters (Part 2)

```
Best Logistic Params: {'model__C': 0.01, 'model__penalty': 'l2', 'model__solver': 'lbfgs'}
Best Logistic Score: 0.5013698630136987
-----------------------------------------------------
Best SVM Params: {'model__C': 1, 'model__gamma': 0.1, 'model__kernel': 'rbf'}
Best SVM Score: 0.5383561643835616
-----------------------------------------------------
C:\Users\Clueless\anaconda3\Lib\site-packages\numpy\ma\core.py:2820: RuntimeWarning: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
Best RF Params: {'model__max_depth': 10, 'model__max_features': 'sqrt', 'model__min_samples_leaf': 2, 'model__min_samples_split':
s': 100}
Best RF Score: 0.7068493150684931
-----------------------------------------------------
C:\Users\Clueless\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [21:25:57] WARNING: C:\actions-runner\_work\xg
er.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Best XGB Params: {'model__learning_rate': 0.01, 'model__max_depth': 3, 'model__n_estimators': 200, 'model__subsample': 0.8}
Best XGB Score: 0.6972602739726028
-----------------------------------------------------
[LightGBM] [Info] Number of positive: 422, number of negative: 454
[LightGBM] [Info] Total Bins 3063
[LightGBM] [Info] Number of data points in the train set: 876, number of used features: 13
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.481735 -> initscore=-0.073092
[LightGBM] [Info] Start training from score -0.073092
Best LGBM Params: {'model__learning_rate': 0.01, 'model__max_depth': -1, 'model__n_estimators': 200, 'model__num_leaves': 31}
Best LGBM Score: 0.6876712328767123
```

Figure 5.41 Result

Figure 5.39 and 5.40 performs exhaustive hyperparameter tuning using GridSearchCV on five machine learning models (LogisticRegression, SVM, RandomForest, XGBoost, and LightGBM) from pre-defined pipelines, tailored for time-series data with TimeSeriesSplit (assumed from previous context as tscv). For each model, a specific parameter grid is defined: LogisticRegression tunes C, penalty, and solver; SVM adjusts C, gamma, and kernel; RandomForest optimizes n_estimators, max_depth, min_samples_split, min_samples_leaf, and max_features; XGBoost refines n_estimators, max_depth, learning_rate, and subsample; and LightGBM tunes n_estimators, max_depth, learning_rate, and num_leaves. Each GridSearchCV instance uses the respective pipeline, employs the time-series cross-validation strategy, targets accuracy as the scoring metric, and leverages all available CPU cores (n_jobs=-1) for parallel processing. The models are fitted on X_train and y_train, and the best parameters and corresponding scores are printed for each, separated by delimiters, providing a comprehensive comparison of optimized model performances.

5.7 Evaluation

```python
from sklearn.metrics import (accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, ConfusionMatrixDisplay, RocCurveDisplay)
import matplotlib.pyplot as plt
import numpy as np

# ✅ Replace models with tuned best estimators
models["LogisticRegression"] = log_grid.best_estimator_
models["SVM"] = svm_grid.best_estimator_
models["RandomForest"] = rf_grid.best_estimator_
models["XGBoost"] = xgb_grid.best_estimator_
models["LightGBM"] = lgb_grid.best_estimator_

# ✅ Evaluate tuned models using the SAME TSCV folds GridSearchCV used
eval_results = {}

for name, model in models.items():
    fold_accuracies, fold_precisions, fold_recalls, fold_f1s = [], [], [], []
    all_preds, all_true = [], []

    for train_idx, test_idx in tscv.split(X_clean):
        X_train, X_test = X_clean.iloc[train_idx], X_clean.iloc[test_idx]
        y_train, y_test = y_clean.iloc[train_idx], y_clean.iloc[test_idx]

        # Train tuned model
        model.fit(X_train, y_train)
        preds = model.predict(X_test)

        # Metrics per fold
        fold_accuracies.append(accuracy_score(y_test, preds))
        fold_precisions.append(precision_score(y_test, preds, average="weighted", zero_division=0))
        fold_recalls.append(recall_score(y_test, preds, average="weighted", zero_division=0))
        fold_f1s.append(f1_score(y_test, preds, average="weighted", zero_division=0))

        # Collect for global graphs
        all_preds.extend(preds)
        all_true.extend(y_test)
```

Figure 5.42 Evaluation of Tuned Models with Metrics and Visualizations (Part 1)

```
    # Average metrics across folds (same as GridSearchCV best_score_)
    eval_results[name] = {
        "Accuracy": np.mean(fold_accuracies),
        "Precision": np.mean(fold_precisions),
        "Recall": np.mean(fold_recalls),
        "F1-Score": np.mean(fold_f1s),
    }

    # ✅ Confusion Matrix (all folds combined)
    cm = confusion_matrix(all_true, all_preds)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot(cmap="Blues")
    plt.title(f"Confusion Matrix - {name}")
    plt.show()

    # ✅ ROC Curve (binary classification only)
    if len(set(y_clean)) == 2:
        try:
            RocCurveDisplay.from_predictions(all_true, all_preds)
            plt.title(f"ROC Curve - {name}")
            plt.show()
        except Exception as e:
            print(f"ROC curve not available for {name}: {e}")

# ✅ Print summary table
print("\n📊 Cross-Validation Evaluation:\n")
print(f"{'Model':<20}{'Accuracy':<12}{'Precision':<12}{'Recall':<12}{'F1-Score':<12}")
print("-"*68)
for model_name, metrics in eval_results.items():
    print(f"{model_name:<20}"
          f"{metrics['Accuracy']:<12.4f}"
          f"{metrics['Precision']:<12.4f}"
          f"{metrics['Recall']:<12.4f}"
          f"{metrics['F1-Score']:<12.4f}")
```

Figure 5.43 Evaluation of Tuned Models with Metrics and Visualizations (Part 2)

Figure 5.42 and 5.43 assesses the performance of five tuned machine learning models (LogisticRegression, SVM, RandomForest, XGBoost, and LightGBM) using time-series cross-validation (TSCV) with the same folds as a previous GridSearchCV. It updates the models dictionary with the best estimators from prior grid searches (log_grid, svm_grid, etc.) and initializes a dictionary eval_results to store evaluation metrics. For each model, it iterates over 5 TSCV folds, training on X_train and y_train, predicting on X_test, and computing accuracy, precision, recall, and F1-score (using weighted averages with zero_division=0 to handle edge cases), while collecting all predictions and true labels for global analysis. The mean of these metrics across folds is stored, aligning with the best scores from GridSearchCV. It generates a confusion matrix visualization for all folds combined using

ConfusionMatrixDisplay with a blue colormap and, if the target is binary, attempts to plot an ROC curve using RocCurveDisplay, handling potential errors.

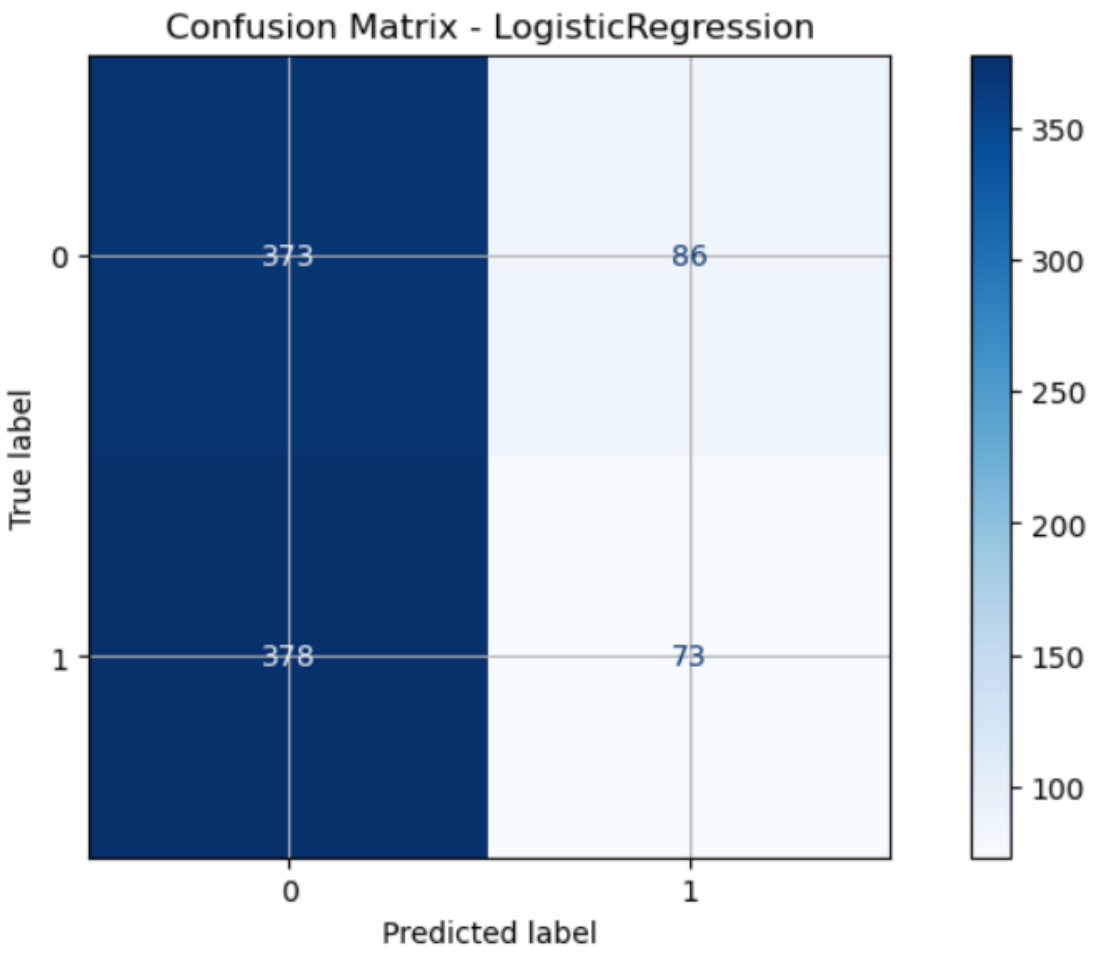


Figure 5.44 Confusion Matrix- LR

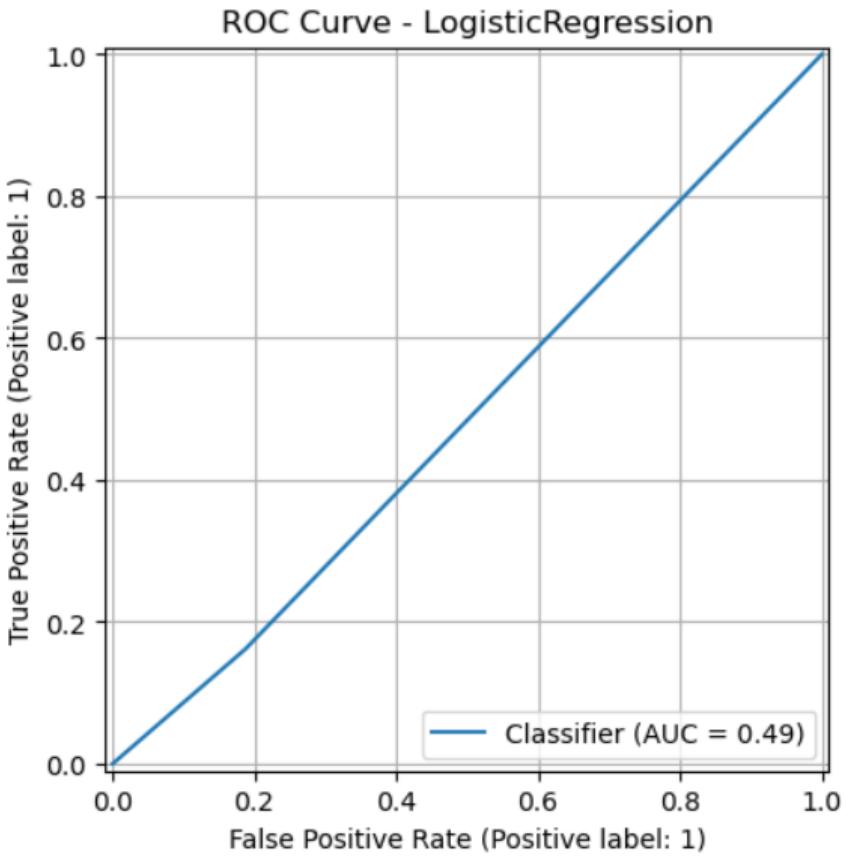


Figure 5.45 ROC Curve- LR

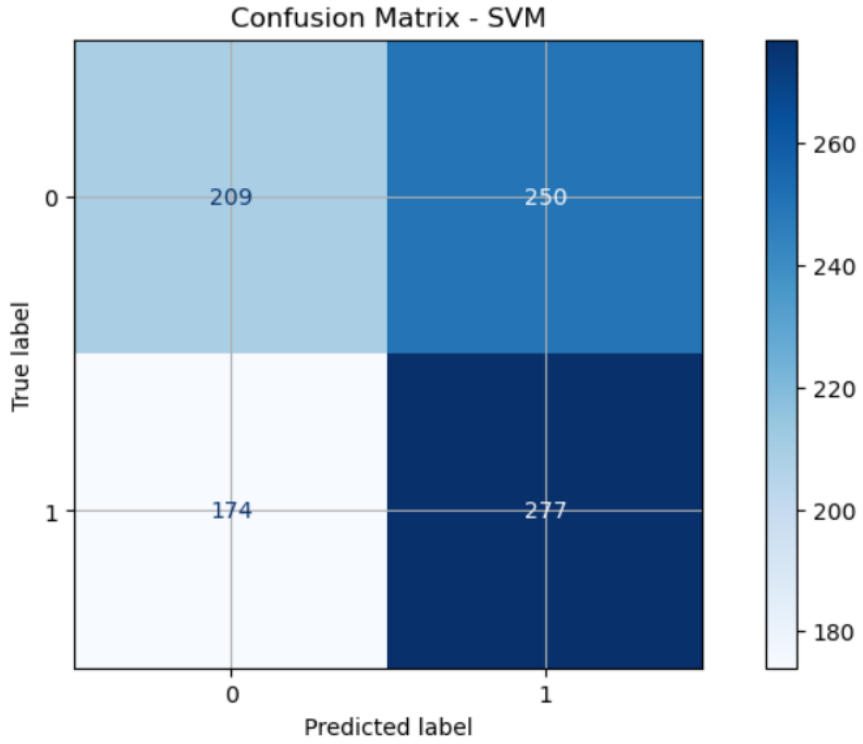


Figure 5.46 Confusion Matrix- SVM

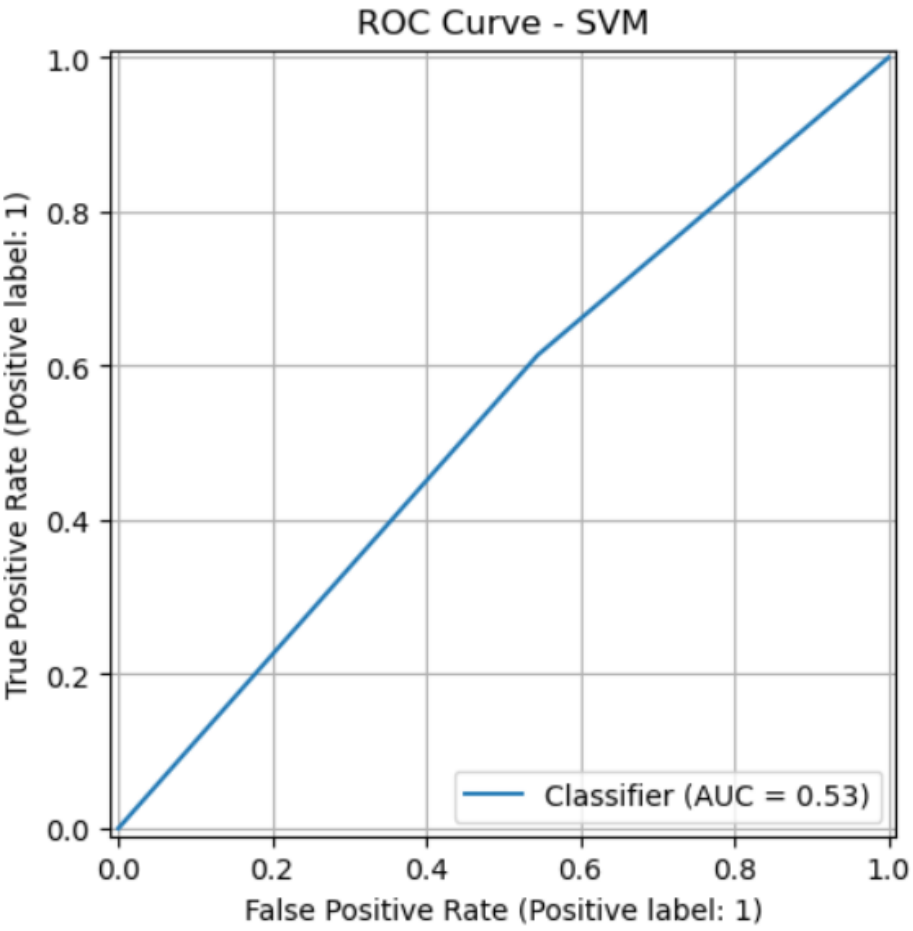


Figure 5.47 ROC Curve- SVM

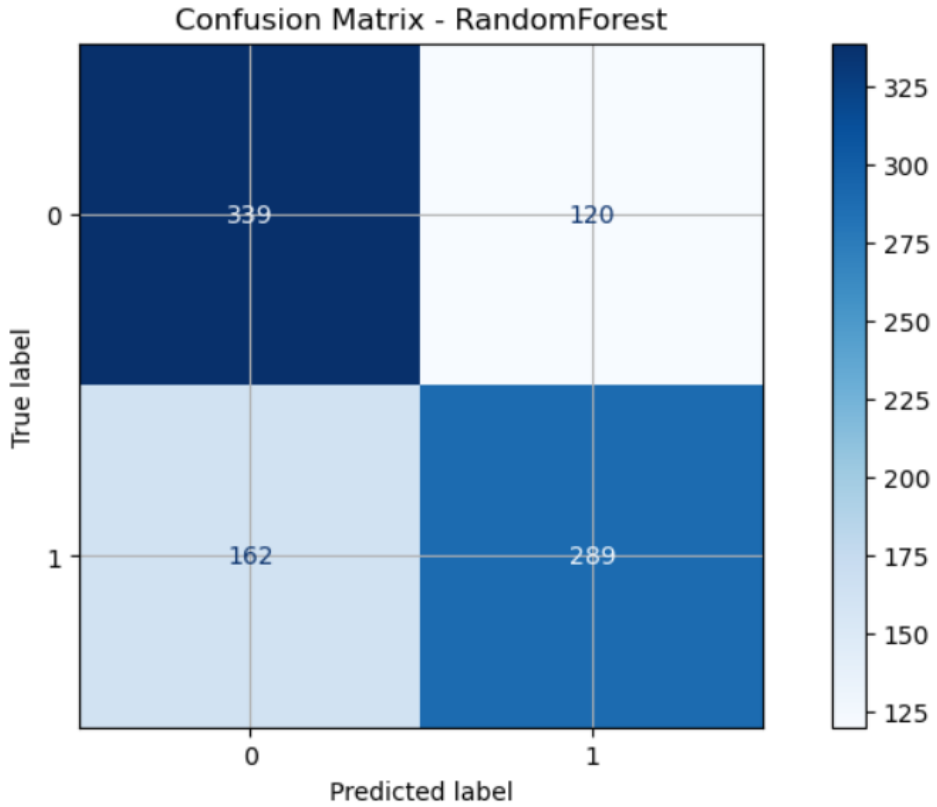


Figure 5.48 Confusion Matrix- RF

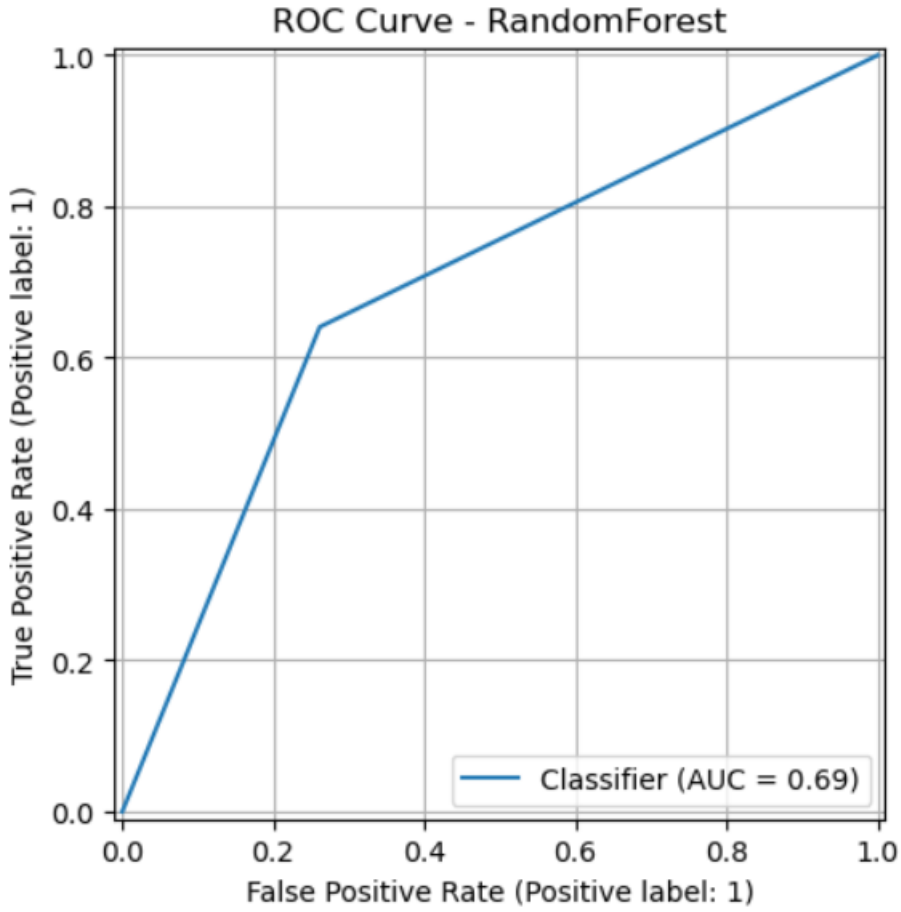


Figure 5.49 ROC Curve- RF

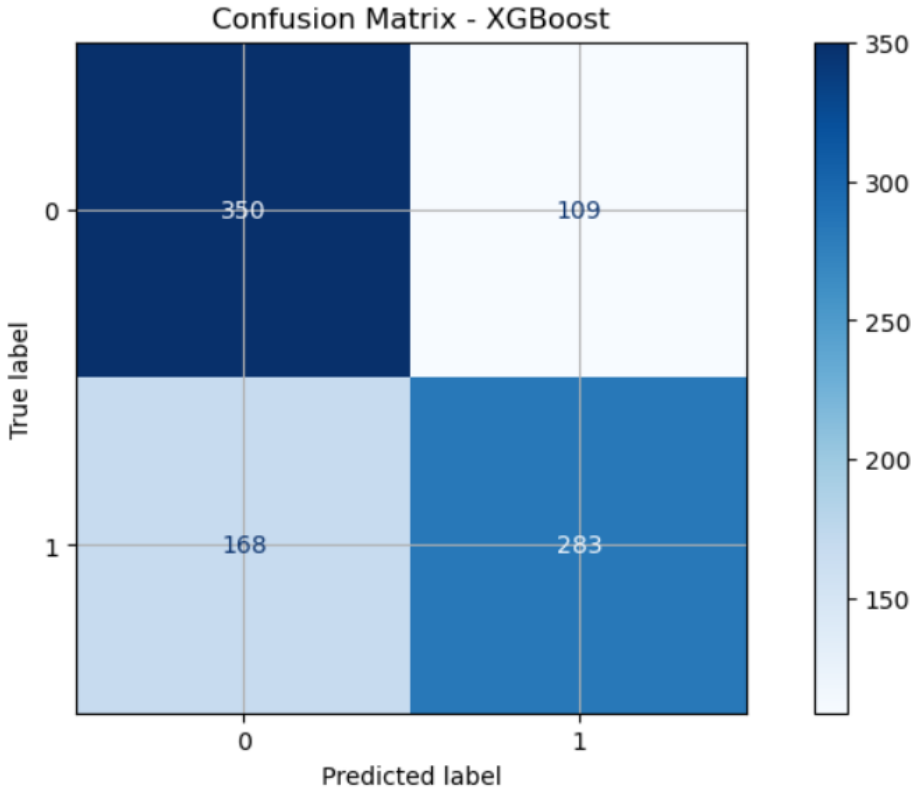


Figure 5.50 Confusion Matrix- XGBoost

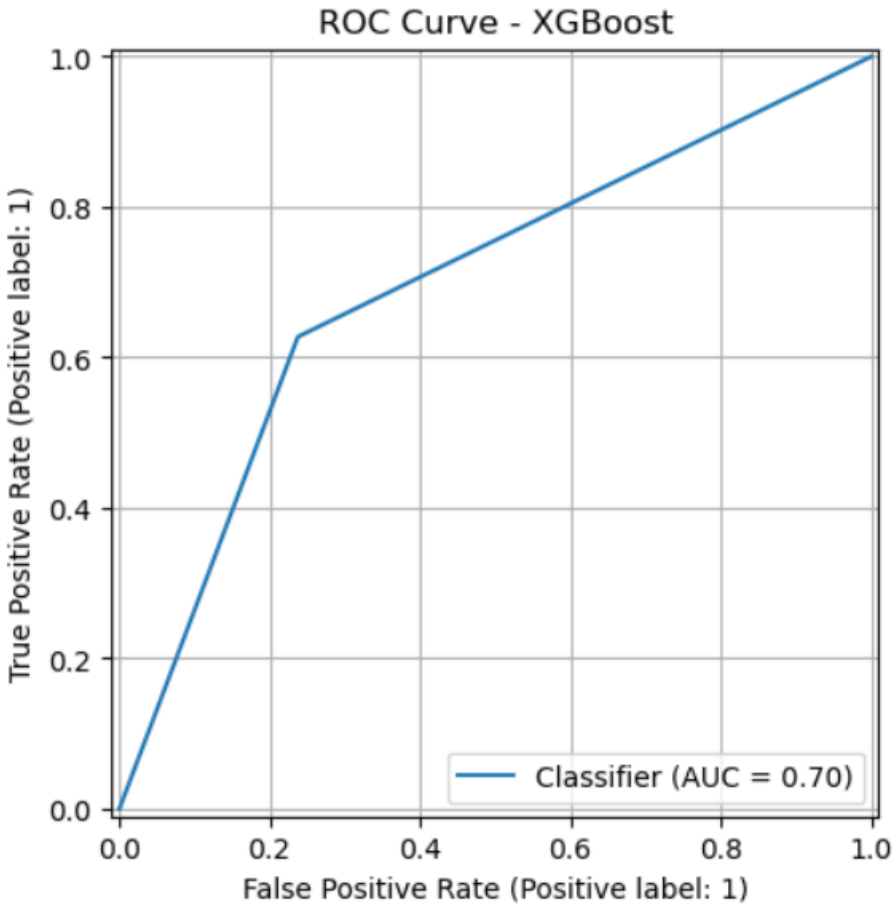


Figure 5.51 ROC Curve- XGBoost

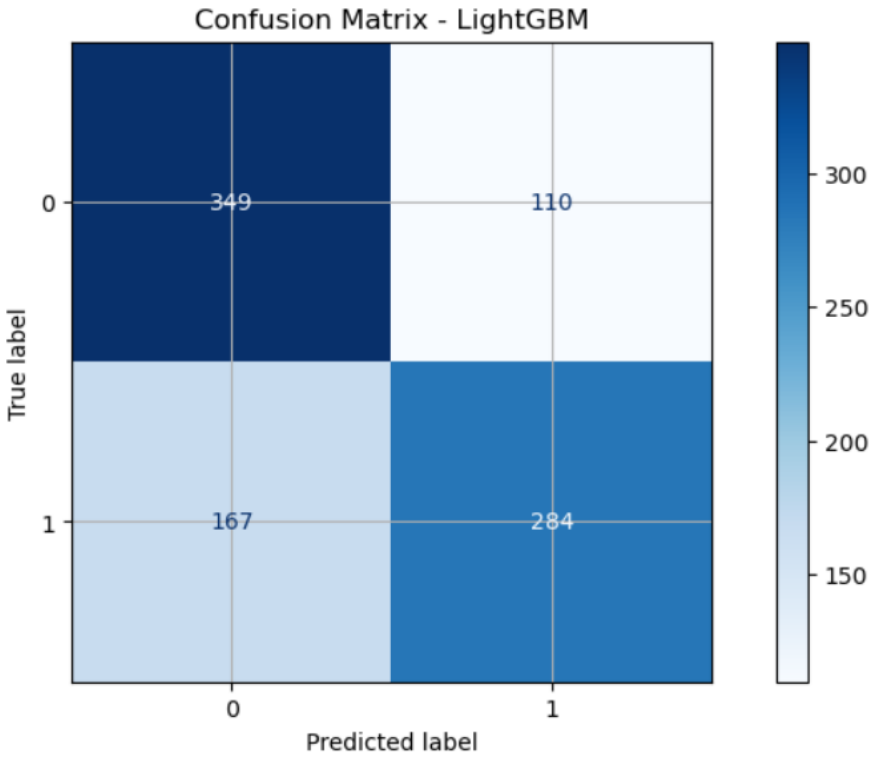


Figure 5.52 Confusion Matrix- LightBGM

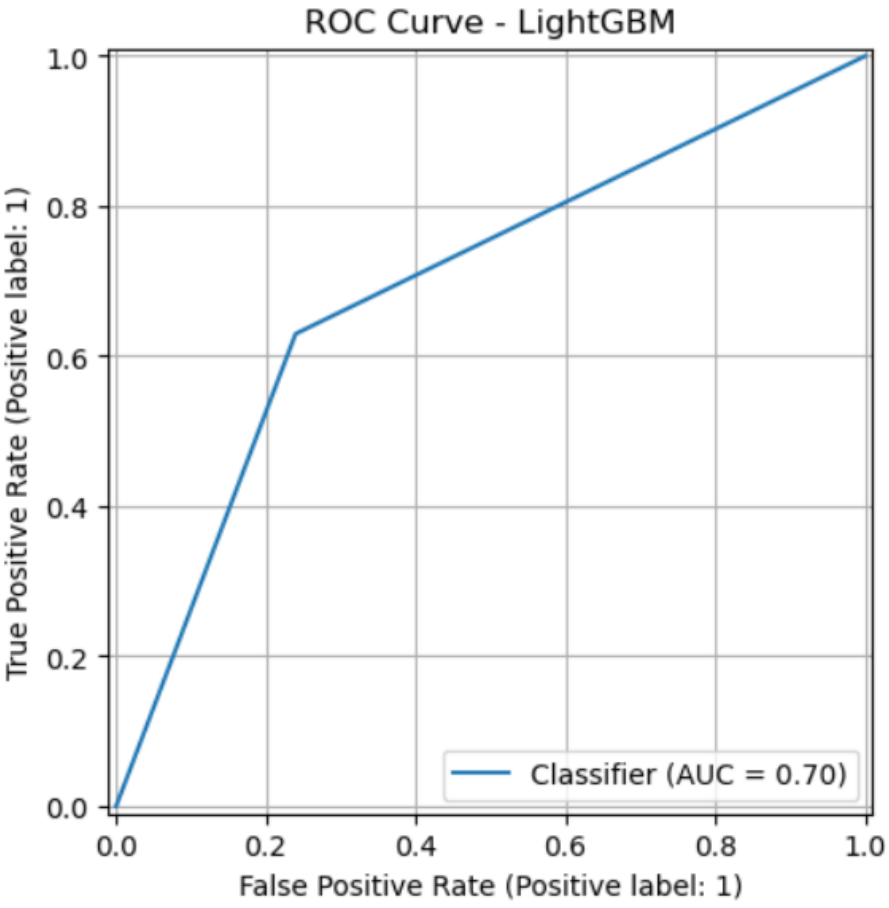


Figure 5.53 ROC Curve- LightBGM

```
📊 Cross-Validation Evaluation:

Model               Accuracy    Precision   Recall      F1-Score
--------------------------------------------------------------------
LogisticRegression  0.4901      0.4640      0.4901      0.4230
SVM                 0.5341      0.5381      0.5341      0.5302
RandomForest        0.6901      0.6927      0.6901      0.6891
XGBoost             0.6956      0.7003      0.6956      0.6938
LightGBM            0.6956      0.7003      0.6956      0.6938
```

Figure 5.54 Cross-Validation Evaluation

```
# Convert your evaluation dictionary to DataFrame
results_df = pd.DataFrame(eval_results).T.reset_index()
results_df.columns = ["Model", "Accuracy", "Precision", "Recall", "F1-Score"]

# Save model evaluation results
results_df.to_csv("Evaluation.csv", index=False)

# Save cleaned dataset (features + labels merged together)
final_dataset = X_clean.copy()
final_dataset["Target"] = y_clean.values
final_dataset.to_csv("Final_Result.csv", index=False)

print("✅ Files saved: Evaluation.csv & Final_Result.csv")
```

```
✅ Files saved: Evaluation.csv & Final_Result.csv
```

Figure 5.55 Evaluation and Final_Result.csv

```
xgb_model = models["XGBoost"]
xgb_model.fit(X_clean, y_clean)
xgb_preds = xgb_model.predict(X_clean)

results_df = X_clean.copy()
results_df["True_Trend"] = y_clean.values
results_df["XGB_Predicted_Trend"] = xgb_preds
results_df.to_csv("XGB_Final_Predictions.csv", index=False)
print("✅ Saved to XGB_Final_Predictions.csv")
```

```
✅ Saved to XGB_Final_Predictions.csv
```

Figure 5.56 XGBoost Model Training and Prediction Export

Figure 5.54 trains an XGBoost model and generates predictions for a cleaned dataset. It retrieves the "XGBoost" model from the model's dictionary, fits it to the entire X_clean feature matrix and y_clean target vector, and uses it to predict trends for X_clean, storing the results in xgb_preds. A new DataFrame, results_df, is created by copying X_clean, with two additional columns: "True_Trend" populated with y_clean values and "XGB_Predicted_Trend" containing the predictions. This DataFrame is saved to "XGB_Final_Predictions.csv". XGBoost is chosen due to the best accuracy and F1-score, and it handles smaller to medium-sized datasets more stable.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
lag1	lag2	lag3	rolling_me	rolling_me	rolling_std	rolling_std	vol_lag1	vol_mean_	vol_std_3	high_low_r	open_clos	sentiment	True_Trend	XGB_Predicted_Trend
4.835868	0.066972	0.059965	1.654268	0.981728	2.755348	2.167868	25309400	31549367	6411106	0.045822	0.036877	-1	1	0
0.157592	4.835868	0.066972	1.686811	1.021015	2.72754	2.14702	33514500	30014567	4233244	0.053407	0.041897	1	0	0
-0.87465	0.157592	4.835868	1.372935	0.962774	3.043075	2.199012	471800	19765233	17204893	0.044509	0.019135	-1	1	1
0.106814	-0.87465	0.157592	-0.20342	0.510479	0.581863	1.956954	38118900	24035067	20535841	0.018994	0.012849	1	0	1

Figure 5.57 XGB_Final_Predictions.csv

Chapter 6

System Evaluation

6.1 Model Performance

Cross-Validation Evaluation:

Model	Accuracy	Precision	Recall	F1-Score
LogisticRegression	0.4901	0.4640	0.4901	0.4230
SVM	0.5341	0.5381	0.5341	0.5302
RandomForest	0.6901	0.6927	0.6901	0.6891
XGBoost	0.6956	0.7003	0.6956	0.6938
LightGBM	0.6956	0.7003	0.6956	0.6938

Figure 6.0 Model Performance

True_Trend	XGB_Predicted_Trend
1	0
0	0
1	1
0	1
0	0
1	1
0	0
1	1

Figure 6.1 Result of Trend

The performance of the machine learning models in this stock market trend prediction project is evaluated using key classification metrics suitable for binary classification tasks, where the goal is to predict whether the stock trend will be upward (1) or downward (0) based on historical prices, news sentiment, and engineered features. These metrics are derived from cross-validation results to ensure robustness against overfitting. The table below summarizes the average metrics across five TimeSeriesSplit folds for each model based on the evaluation part.

6.1.1 Accuracy

Accuracy is a fundamental metric for classification models, representing the proportion of correct predictions (both true positives and true negatives) out of all predictions made. In the context of this project, accuracy measures how often the model correctly identifies the direction of stock trends (up or down) across the cross-validation folds. For example, XGBoost and LightGBM achieve the highest accuracy of 0.6956, indicating that approximately 69.56% of the trend predictions align with actual outcomes. In stock market trend prediction, accuracy is crucial for building investor confidence, as it directly reflects the model's overall reliability in forecasting directional movements. High accuracy helps traders avoid frequent misjudgements that could lead to losses, especially in volatile markets.

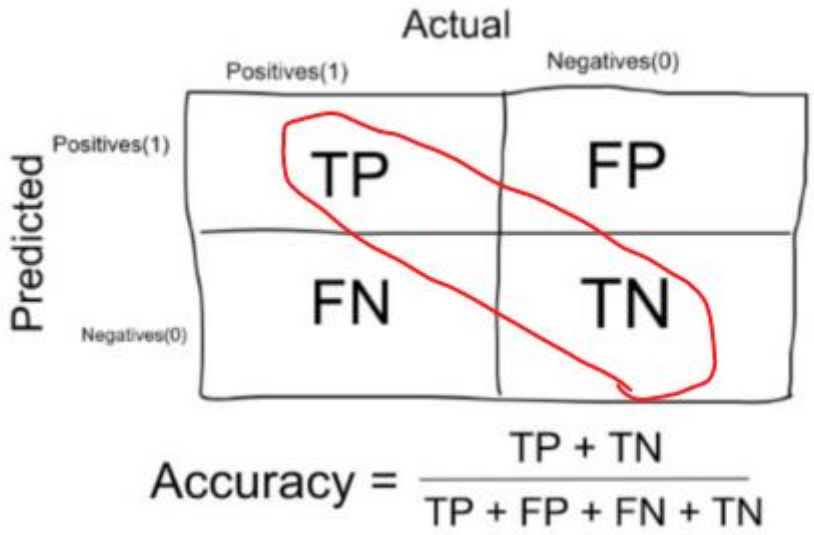


Figure 6.2 Formula of Accuracy

6.1.2 Precision

Precision is positive predictive value which measures the proportion of positive predictions (e.g., predicted upward trends) that are correct. It focuses on the quality of positive predictions, minimizing false alarms. In this evaluation, LightGBM shows the highest precision of 0.7003, meaning about 70.03% of the times it predicts an upward trend, it is correct. Precision is vital in stock prediction to reduce the risk of false positive signals, which could mislead investors into buying stocks expecting gains that do not materialize, resulting in unnecessary transaction costs or opportunity losses.

$$Precision = \frac{TP}{TP + FP}$$

Figure 6.3 Formula of Precision

6.1.3 Recall

Recall, or sensitivity/true positive rate, measures the proportion of actual positive instances (e.g., real upward trends) that are correctly identified by the model. It emphasizes the model's ability to capture all relevant positive cases. Here, XGBoost and LightGBM both achieve a recall of 0.6956, indicating they correctly identify 69.56% of actual upward trends. In stock trend prediction, recall is important for ensuring the model does not miss profitable opportunities, such as failing to detect genuine upward movements that could lead to missed gains. High recall, as seen in RandomForest (0.6901), helps in comprehensive market surveillance, particularly for long-term investors who prioritize capturing all potential rises amid market noise from news sentiment and volume fluctuations. Low recall in LogisticRegression (0.4901) could mean overlooking significant bullish signals, which is risky in dynamic markets where timely detection of trends can differentiate between profit and stagnation.

$$Recall = \frac{TP}{TP + FN}$$

Figure 6.4 Formula of Recall

6.1.4 F1-Score

The F1-Score is the harmonic mean of Precision and Recall, providing a balanced measure of a model's performance. It is particularly useful when dealing with imbalanced classes or when both false positives and false negatives carry significant costs. In this project, XGBoost and LightGBM tie for the highest F1-Score of approximately 0.6938, offering a strong balance between capturing true trends and avoiding false signals. The F1-Score is especially relevant in stock prediction scenarios where both over-predicting gains (low precision) and missing real opportunities (low recall) can lead to suboptimal decisions. It penalizes models that excel in one but fail in the other, making it ideal for evaluating trend classifiers integrated with sentiment analysis. For example, the superior F1-Score of 0.6938 in XGBoost underscores its robustness for real-world trading, where balanced performance helps mitigate risks from volatile news-driven markets. In comparison, the lower F1-Score of 0.4230 in LogisticRegression indicates poorer overall utility, emphasizing the value of ensemble methods like tree-based models in handling complex financial time-series data.

$$\text{F1 Score} = \frac{2}{\left(\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}\right)}$$

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Figure 6.5 Formula of F1-Score

6.2 PowerBI Dashboard

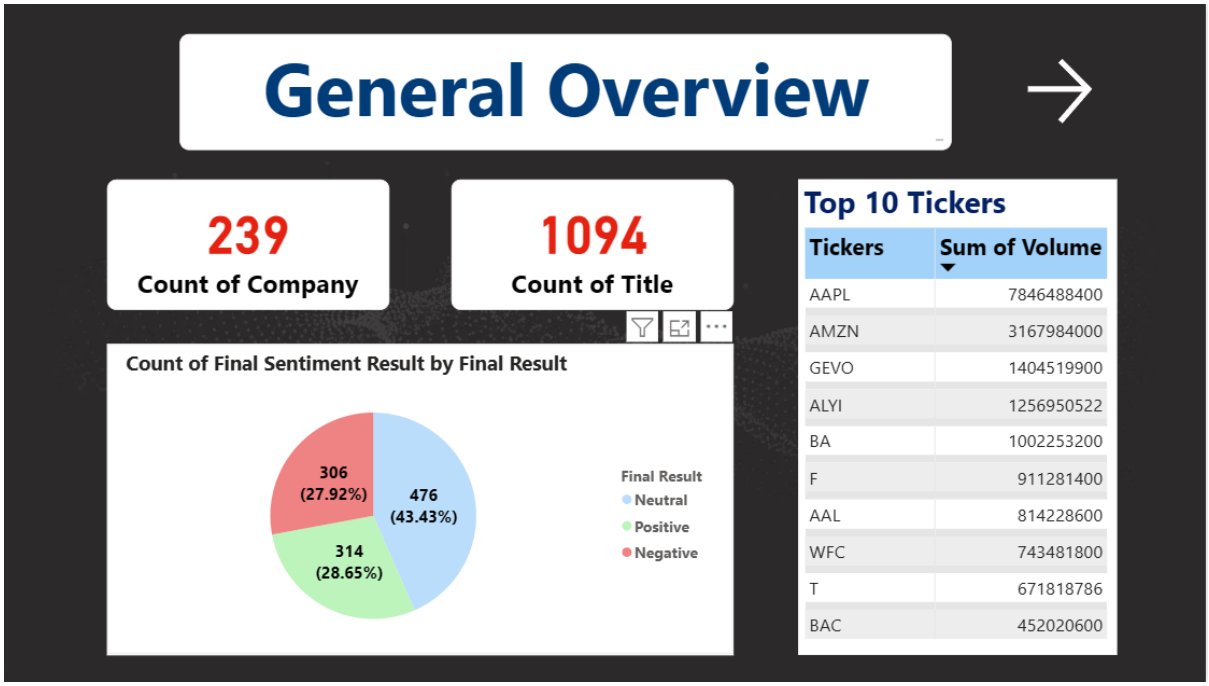


Figure 6.6 General Overview

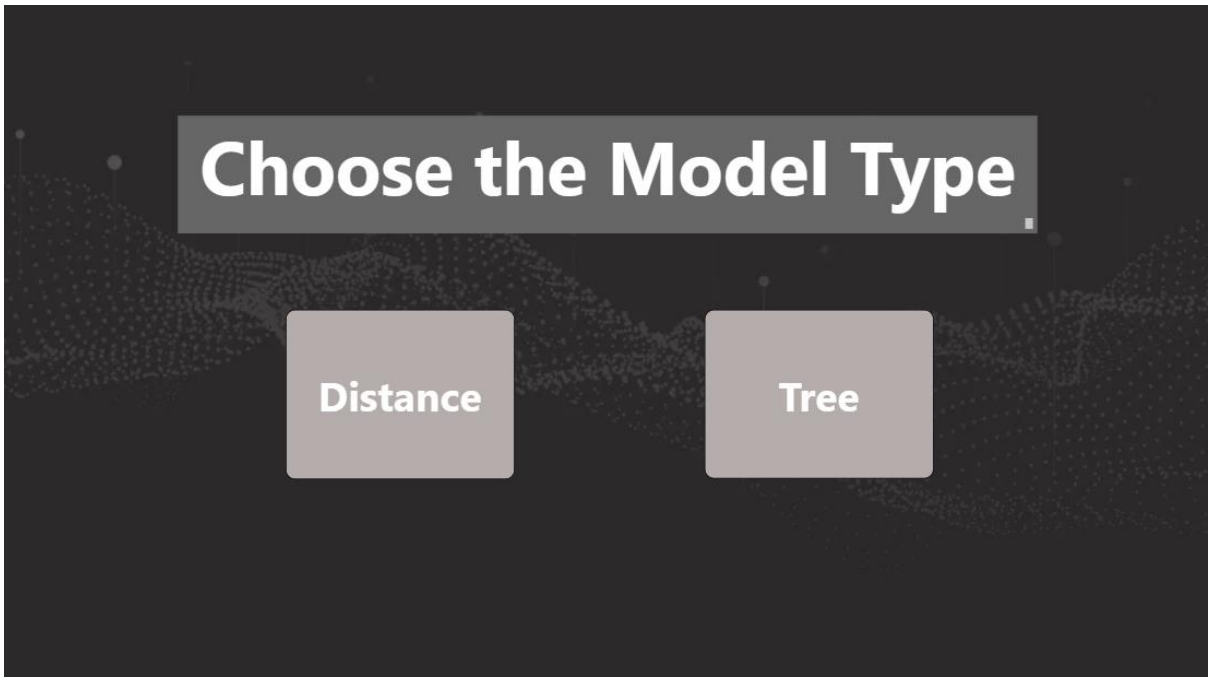


Figure 6.7 Model Selection

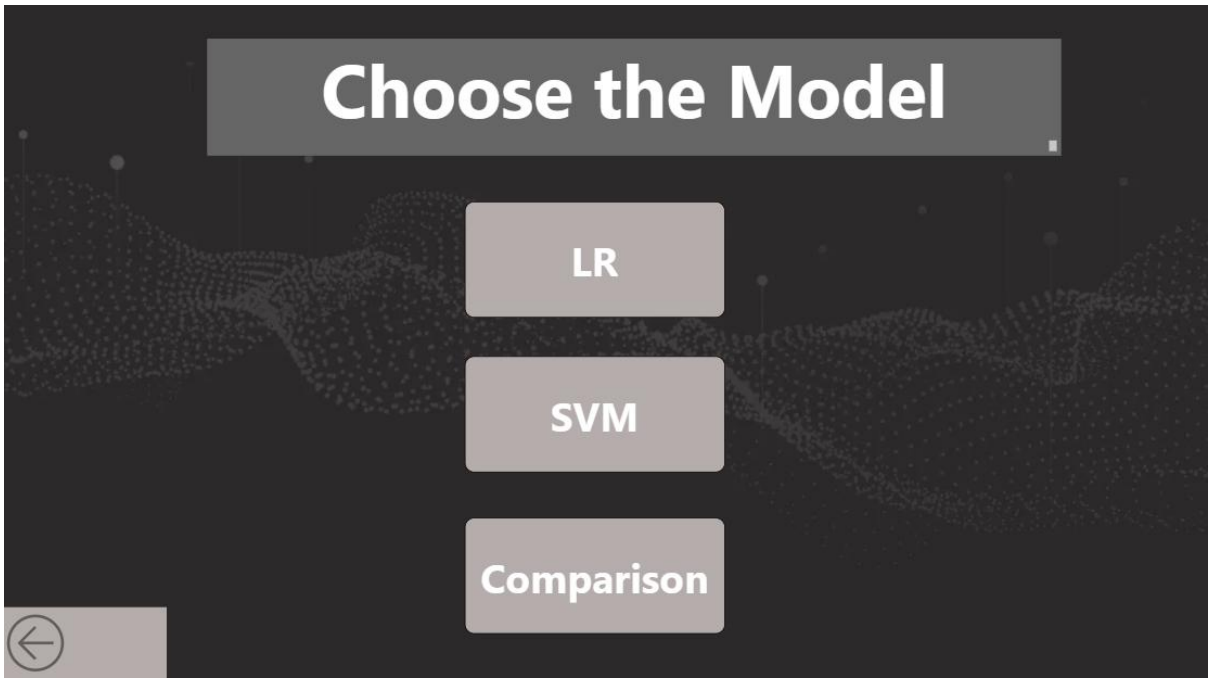


Figure 6.8 Distance Based Model Selection

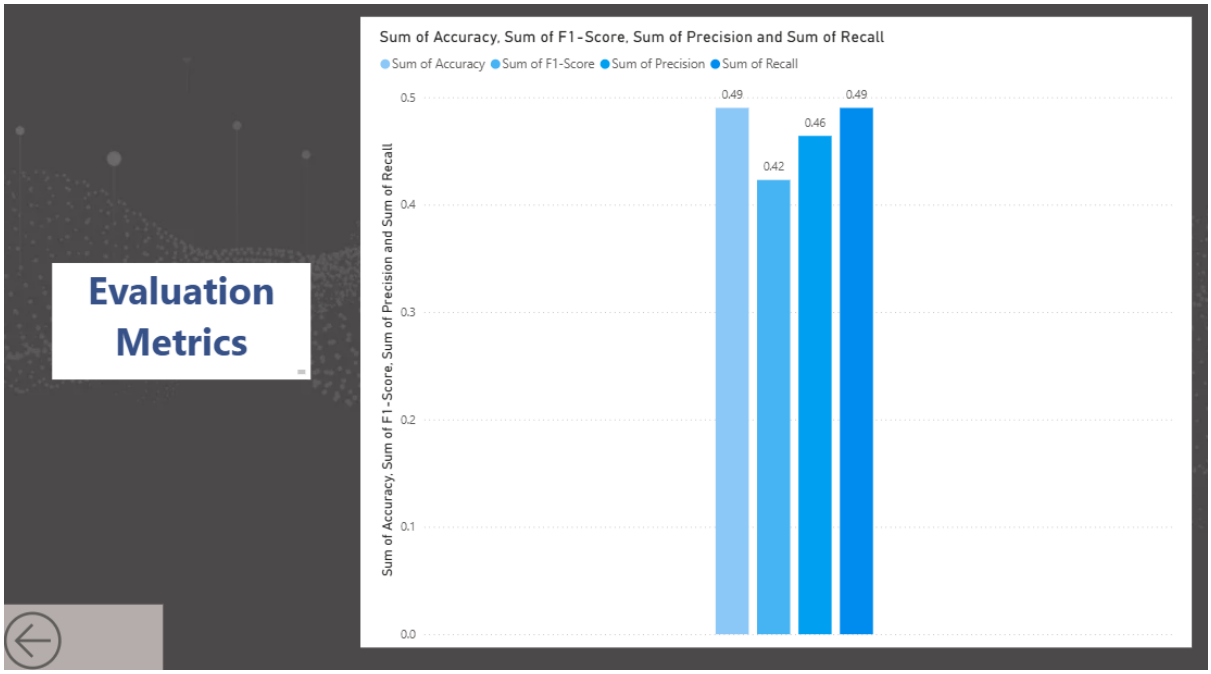


Figure 6.9 Logistic Regression

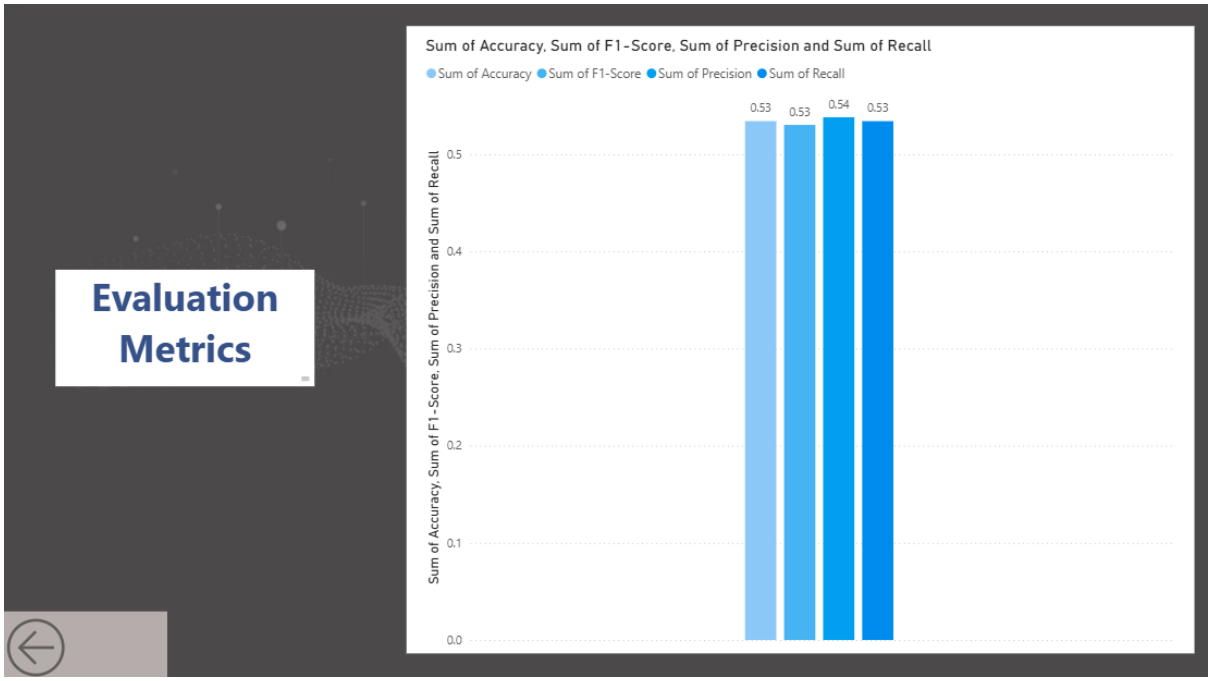


Figure 6.10 Support Vector Machine

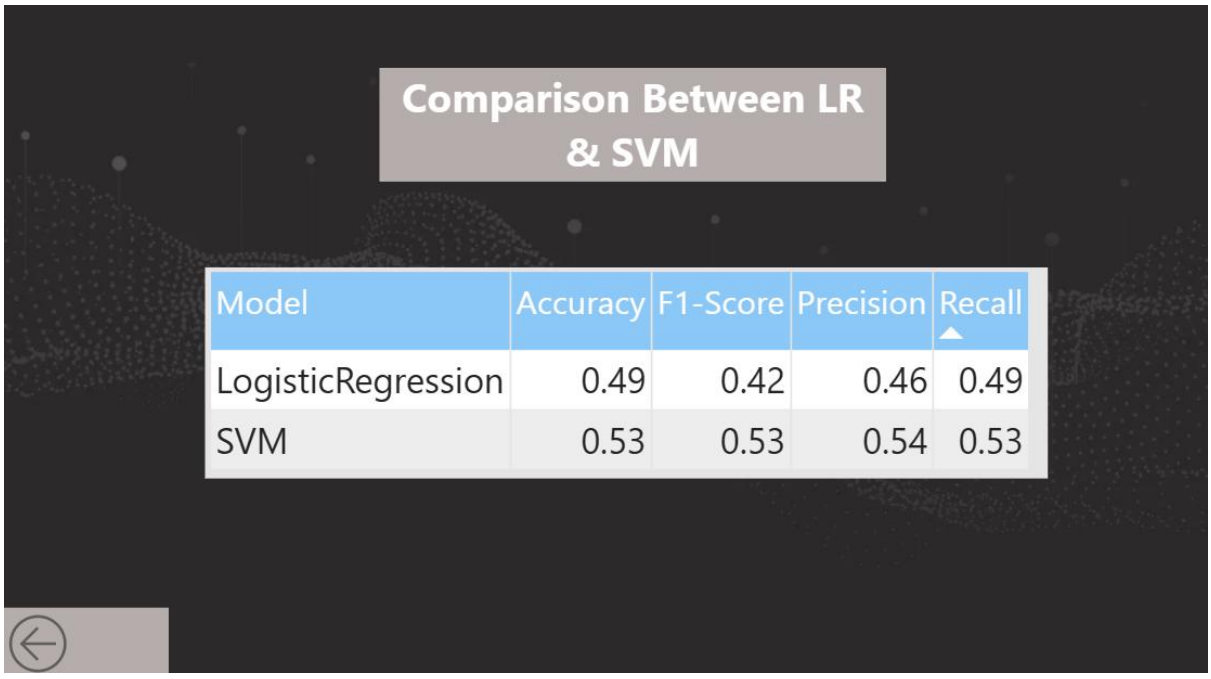


Figure 6.11 Comparison for Distance Based Model

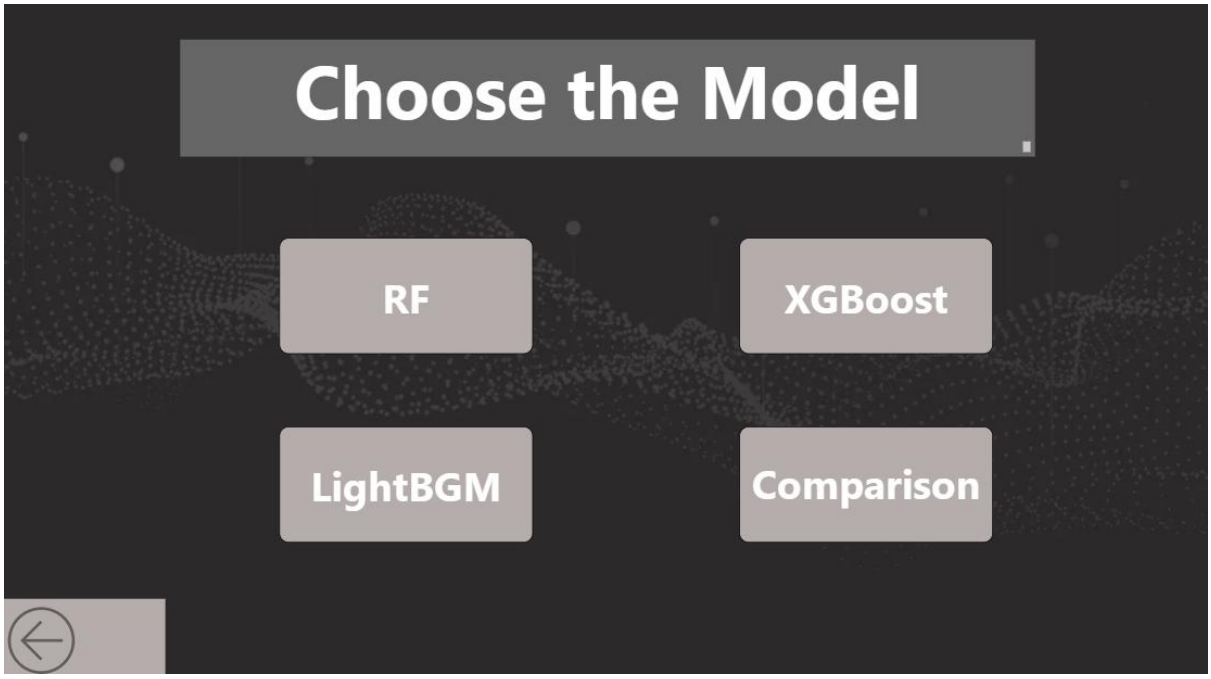


Figure 6.12 Tree Based Model Selection

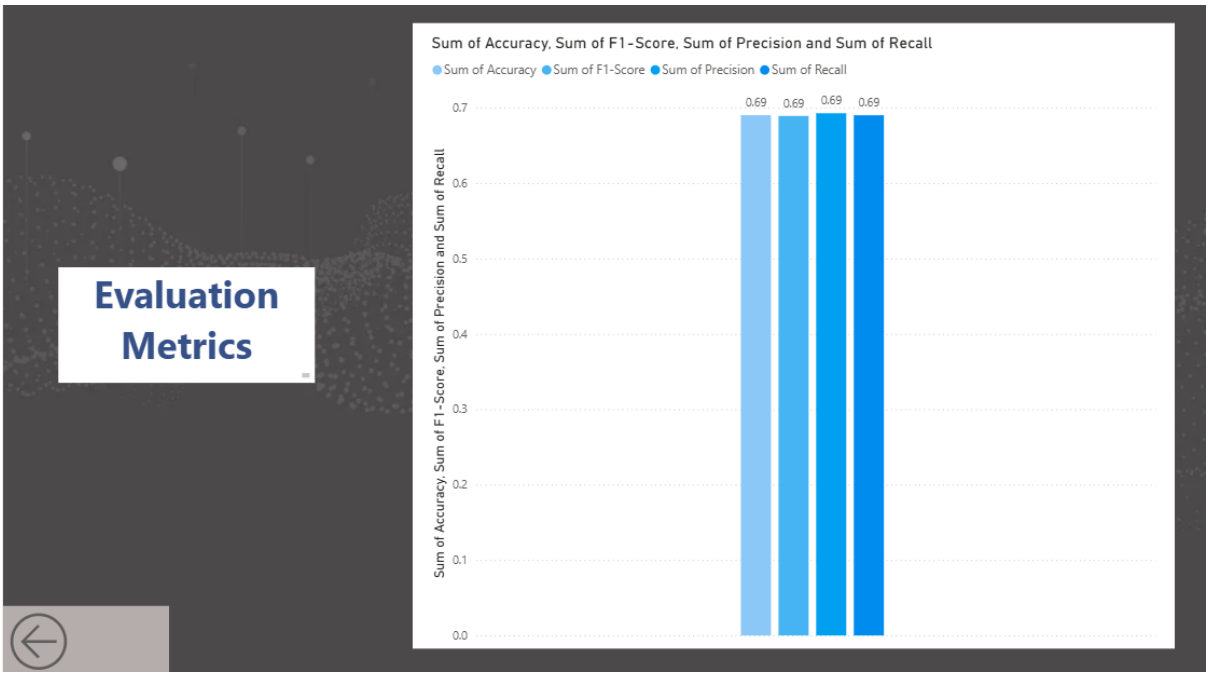


Figure 6.13 Random Forest

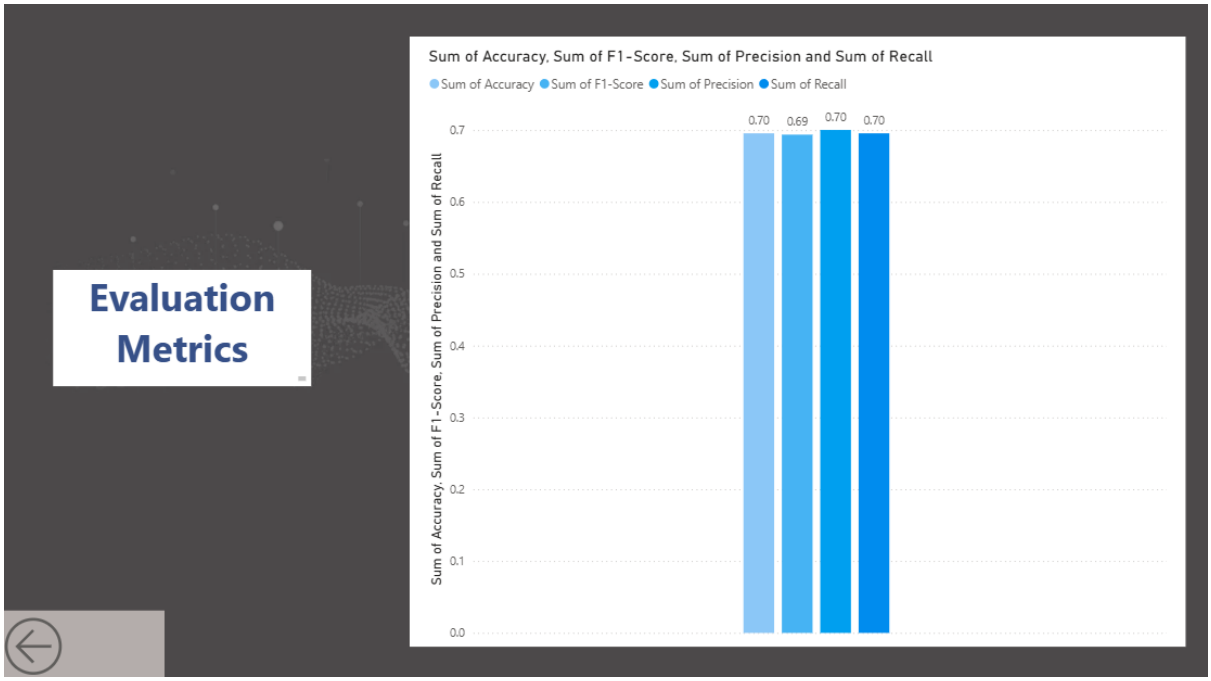


Figure 6.14 LightBGM

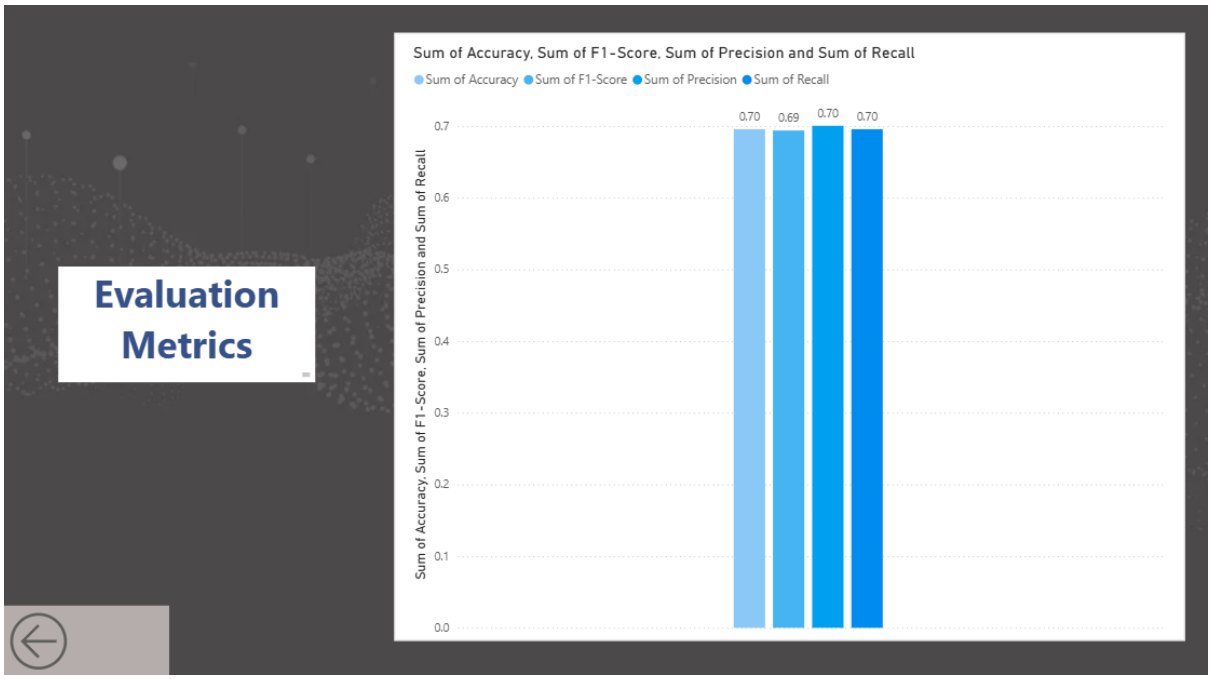


Figure 6.15 XGBoost

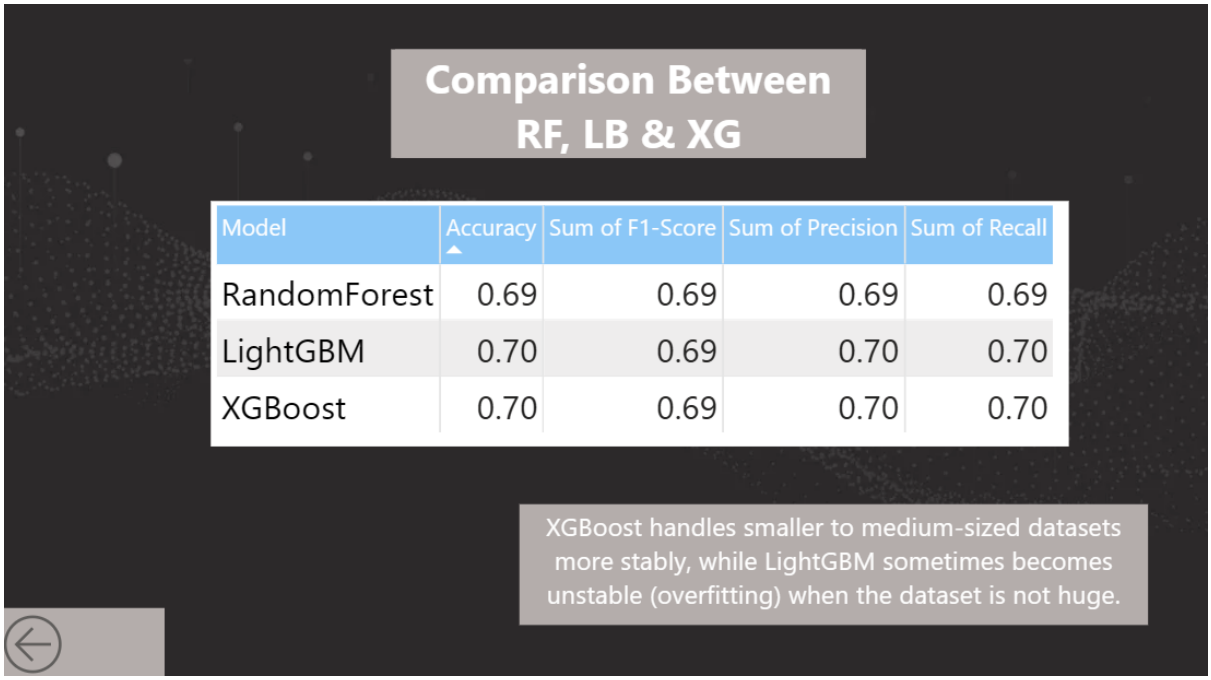


Figure 6.16 Comparison for Tree Based Model

6.3 Projects Challenges

One of the primary challenges encountered in this project was the integration and quality of data from diverse sources, particularly when merging financial news headlines with historical stock price data. The news dataset often contained unlabelled sentiment, multiple tickers per headline, and inconsistencies in date-time formats, which required extensive cleaning steps such as removing 'unknown' companies, exploding tickers, and handling failed downloads.

Besides that, the accuracy and nuance of sentiment analysis applied to financial news, as the domain-specific language often includes sarcasm, ambiguity, and context-dependent interpretations that general NLP tools struggle to capture. The project employed multiple models, TextBlob for basic polarity, VADER for informal text, BERT for multilingual star-rating mapping, and FinBERT for finance-tuned sentiment and achieving a balanced "Final Result" via majority voting required custom thresholds and handling of neutral scores, which could lead to misclassifications.

Chapter 7
Conclusion and Recommendations

7.1 Conclusion

This project successfully developed and evaluated a comprehensive stock price trend prediction framework that integrates sentiment analysis of financial news with historical price data using natural language processing techniques and machine learning algorithms. The research addressed the critical challenge of incorporating qualitative market sentiment alongside quantitative financial indicators to enhance the accuracy of stock trend forecasting.

The framework demonstrated the effectiveness of combining multiple sentiment analysis approaches, employing four distinct models: TextBlob for basic polarity assessment, VADER for handling informal financial text, BERT for contextual understanding, and FinBERT for domain-specific financial sentiment analysis. Through a majority voting mechanism, these models were aggregated to produce a robust sentiment indicator that captured nuanced market emotions expressed in news headlines. This multi-model approach proved superior to relying on any single sentiment analysis tool, as it mitigated the individual limitations of each method while leveraging their collective strengths.

The machine learning evaluation revealed that ensemble methods, particularly XGBoost and LightGBM, significantly outperformed traditional approaches like Logistic Regression and Support Vector Machine. XGBoost emerged as the optimal model, achieving an accuracy of 69.56%, precision of 69.38%, recall of 69.56%, and F1-score of 69.38%. These results demonstrate that gradient boosting algorithms are well-suited for handling the complex, non-linear relationships present in financial time-series data enriched with sentiment features. The model's balanced performance across all metrics indicates its reliability for practical trading applications where both false positives and false negatives carry significant costs.

The comprehensive feature engineering process, incorporating lagged returns, rolling window statistics, volume indicators, and technical features alongside sentiment scores, proved essential for capturing temporal dependencies and market dynamics. The integration of both

numerical price data and textual sentiment information created a more holistic representation of market conditions, enabling the model to better understand the interplay between news sentiment and price movements.

The project's methodology successfully addressed several key challenges in financial prediction, including data quality issues, sentiment analysis accuracy, and model generalization. The time-series cross-validation approach ensured robust evaluation while preventing data leakage, and the extensive hyperparameter tuning optimized model performance. The development of an interactive Power BI dashboard provided stakeholders with accessible visualization tools for monitoring sentiment trends.

7.2 Recommendations

Although the system achieved its objectives, several areas for improvement remain. Future work should **incorporate more diverse and comprehensive data sources** beyond the current news headlines dataset. This includes integrating social media sentiment from platforms like Twitter and Reddit, earnings call transcripts, analyst reports, and regulatory filings. Expanding the data sources would provide a more complete picture of market sentiment and reduce dependency on single-source bias. Additionally, implementing real-time data streaming capabilities would enable the system to respond to breaking news and market events more effectively, potentially improving prediction accuracy during volatile periods.

Next, the feature engineering process can be expanded to include **additional technical indicators**, market microstructure features, and macroeconomic variables. **Implementing deep learning architectures** such as LSTM networks or transformer-based models could better capture long-term temporal dependencies in both price and sentiment data. Additionally, exploring ensemble methods that **combine multiple model types** (e.g., tree-based models with neural networks) might yield improved predictive performance. The integration of attention mechanisms could help identify which news articles or sentiment signals are most relevant for specific stocks or time periods.

Finally, **risk management of investors** should be ensured. For practical deployment, the system should incorporate robust risk management features including position sizing recommendations, stop-loss mechanisms, and portfolio diversification guidance.

REFERENCES

[1] D. Brčić, P. Cetin, D. S. Milijanić, and D. Vrcić, "Adaptive Video Streaming Over Information Centric Networking Using Game Theory and Machine Learning," in Future Internet, vol. 15, no. 2, p. 71, 2023. [Online]. Available: https://www.mdpi.com/1999-5903/15/2/71.

[2] E. F. Fama, "Efficient Capital Markets: A Review of Theory and Empirical Work," in The Journal of Finance, vol. 25, no. 2, pp. 383–417, 1970. [Online]. Available: http://gesd.free.fr/fama1970.pdf.

[3] R. Gade, A. Rane, and P. Pande, "Improving Cyber Defense Using Multi-Layer Threat Detection with Machine Learning," in *2020 IEEE International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, 2020, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8966841.

[4] S. Aftab, A. S. Ahmad, R. Nawaz, S. Ahmad, and S. H. Anwar, "Machine Learning Techniques for Sentiment Analysis: A Review," in *International Journal of Computer Applications*, vol. 159, no. 4, pp. 25–31, 2017. [Online]. Available: https://www.researchgate.net/publication/317284281_Machine_Learning_Techniques_for_Sentiment_Analysis_A_Review.

[5] Z. Li, Y. Wu, and B. Liang, "A Survey on Information-Centric Networking Research and Future Directions," in *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2021, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9683524.

[6] M. Chen, Z. Da, and P. Gao, "News Sentiment and Anomaly in Stock Returns," in *Review of Financial Studies*, vol. 32, no. 5, pp. 1647–1693, 2019. [Online]. Available: https://academic.oup.com/rfs/article/32/5/1647/5427782.

[7] D. Kumar, R. Kumar, and D. Kumar, "Improving Cyber Defense Using Multi-Layer Threat Detection with Machine Learning," in *2020 IEEE International Conference on*

Emerging Trends in Information Technology and Engineering (ic-ETITE), 2020, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8966841.

[8] M. M. Abilov, A. O. Ganiev, and D. M. Ismayilov, "Predicting Stock Market Using Natural Language Processing," in *Asian Journal of Business Research*, vol. 12, no. 1, pp. 36-50, 2022. [Online]. Available: https://www.emerald.com/insight/content/doi/10.1108/AJB-08-2022-0124/full/pdf.

[9] A. Vaswani et al., "Attention Is All You Need," in *arXiv preprint*, arXiv:1912.07700, 2019. [Online]. Available: https://arxiv.org/pdf/1912.07700.

[10] S. Halder, "FinBERT-LSTM: Deep learning-based stock price prediction using news sentiment analysis," in Proc. Int. Conf. Cloud Big Data Comput., SRM Inst. Sci. Technol., Kattankulathur, India, 2022, pp. 1–4. [Online]. Available: https://arxiv.org/abs/2211.07392.

[11] G. Maltoudoglou, A. G. Kanatas, and G. T. Soldatos, "A Survey of Machine Learning Applications for Blockchain," in Proceedings of the Machine Learning Research, vol. 128, pp. 138–150, 2020. [Online]. Available: https://proceedings.mlr.press/v128/maltoudoglou20a/maltoudoglou20a.pdf.

[12] P. Zhao, Z. Lin, and S. Xu, "A Survey on Deep Reinforcement Learning for Finance," in 2021 IEEE International Conference on Big Data (Big Data), 2021, pp. 2853–2862. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9258102.

[13] M. G. de Sousa et al., "BERT for stock market sentiment analysis," in Proc. 31st IEEE Int. Conf. Tools Artif. Intell. (ICTAI), Portland, OR, USA, Nov. 2019, pp. 1597–1601. doi: 10.1109/ICTAI.2019.00231.

[14] S. Alaparthi and M. Mishra, "Bidirectional encoder representations from transformers (BERT): A sentiment analysis odyssey," arXiv:2007.01127 [cs.CL], Jul. 2020. Available: https://arxiv.org/abs/2007.01127.

[15] A. Rahman, M. H. Siddiqui, and A. Imtiaz, "A Comprehensive Survey on Domain Adaptation in NLP," in *Wiley Interdisciplinary Reviews: Data Mining and Knowledge*

Discovery, vol. 10, no. 5, p. e1253, 2020. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/epdf/10.1002/widm.1253.

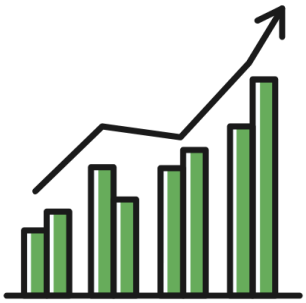
[16] A. Graves, G. Wayne, and I. Danihelka, "Neural Turing Machines," in arXiv preprint, arXiv:1409.0473, 2014. [Online]. Available: https://arxiv.org/pdf/1409.0473.

[17] M. S. Jan et al., "Appraisal of different artificial intelligence techniques for the prediction of marble strength," Sustainability, vol. 15, no. 11, p. 8835, May 2023. doi: 10.3390/su15118835.

[18] N. S. Chandrahas et al., "XG boost algorithm to simultaneous prediction of rock fragmentation and induced ground vibration using unique blast data," Appl. Sci., vol. 12, no. 10, p. 5269, May 2022. doi: 10.3390/app12105269.

[19] Y. Yang, Y. Wu, P. Wang, and X. Jiali, "Stock price prediction based on XGBoost and LightGBM," in Proc. EILCD, Southwest Minzu Univ., Chengdu, China, 2021, pp. 1–4. doi: 10.1051/e3sconf/202127501040.

POSTER



SENTIMENT ANALYSIS OF FINANCIAL NEWS FOR PREDICTING STOCK PRICE TRENDS USING NLP TECHNIQUES IN FINTECH

CHERNG JUN KAI

SUPERVISOR: MS NURUL SYAFIDAH BINTI JAMIL

01 INTRODUCTION

Financial markets are highly sensitive to public sentiment, which is often reflected in financial news articles. Traditional analysis methods tend to overlook the emotional tone embedded in language that can influence investor behavior and market trends.

02 OBJECTIVE

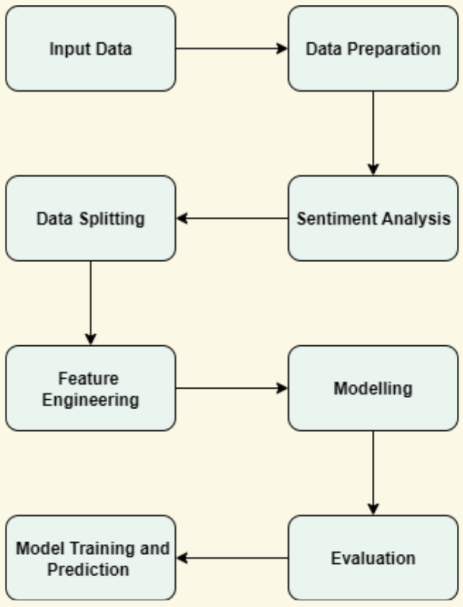
Main Objective: To develop a stock price trend prediction model based on sentiment of financial news.

Sub Objectives:

- To collect and preprocess historical stock price data and news textual data or to perform data preparation.
- To merge historical stock price data with news sentiment data

03 METHODOLOGY

I. Data Merging & Alignment:
Merged financial news with historical OHLCV data
II. Data Preparation & Cleaning:
Cleaned headlines, removed duplicates, and imputed missing numeric values for consistency.
III. Sentiment Extraction & Aggregation:
Applied VADER, TextBlob, BERT, and FinBERT to generate and aggregate daily sentiment scores.
IV. Feature Engineering:
Created lagged returns, percentage changes, rolling features, and time-based attributes for model input.
V. Model Training, Tuning & Validation:
Trained SVM, LR, RF, XGBoost, and LightGBM. Compared models using accuracy, precision, recall, F1-score, and confusion matrices,
VI: Model Selection:
Selected XGBoost (with best result) as final.



04 CONCLUSION

This project successfully developed and evaluated a comprehensive stock price trend prediction framework that integrates sentiment analysis of financial news with historical price data using natural language processing techniques and machine learning algorithms.

UNIVERSITI TUNKU ABDUL RAHMAN

BACHELOR OF INFORMATION SYSTEMS (HONOURS) DIGITAL ECONOMY TECHNOLOGY