

**STOCK MARKET PREDICTION USING NATURAL LANGUAGE PROCESSING**

**BY**

**LOOI WEI HUNG**

**A REPORT**

**SUBMITTED TO**

**Universiti Tunku Abdul Rahman**

**in partial fulfillment of the requirements**

**for the degree of**

**BACHELOR OF INFORMATION SYSTEMS (HONOURS) BUSINESS INFORMATION**

**SYSTEMS**

**Faculty of Information and Communication Technology**

**(Kampar Campus)**

**JUNE 2024**

## REPORT STATUS DECLARATION FORM

**Title:** STOCK MARKET PREDICTION USING NATURAL LANGUAGE PROCESSING(NLP)

**Academic Session:** \_\_JUNE/2024\_\_

I \_\_\_\_\_  
LOOI WEI HUNG  
(CAPITAL LETTER)

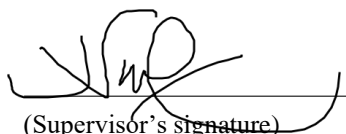
declare that I allow this Final Year Project Report to be kept in  
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,



\_\_\_\_\_  
(Author's signature)

  
(Supervisor's signature)

**Address:**

8,LALUAN MENGLEMBU 9  
TAMAN MENGLEMBU BERLIAN  
31450 IPOH,PERAK

**Date:** 13/9/2024

**Supervisor Name:**

Cik Nurul Syafidah Binti Jamil

**Date:** 13/09/2024

<b>Universiti Tunku Abdul Rahman</b>			
Form Title : <b>Sample of Submission Sheet for FYP/Dissertation/Thesis</b>			
Form Number: <b>FM-IAD-004</b>	Rev No.: <b>0</b>	Effective Date: <b>21 JUNE 2011</b>	Page No.: <b>1 of 1</b>

**FACULTY/INSTITUTE\* OF \_ INFORMATION AND COMMUNICATION  
TECHNOLOGY**

**UNIVERSITI TUNKU ABDUL RAHMAN**

Date: 13/9/2024

**SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS**

It is hereby certified that LOOI WEI HUNG (ID No: 20ACB04565) has completed this final year project/ dissertation/ thesis\* entitled “STOCK MARKET PREDICTION USING NATURAL LANGUAGE PROCESSING(NLP)” under the supervision of Cik Nurul Syafidah Binti Jamil (Supervisor) from the Department of Digital Economy Technology, Faculty/Institute\* of \_ INFORMATION AND COMMUNICATION TECHNOLOGY\_.

I understand that University will upload softcopy of my final year project / dissertation/ thesis\* in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

LOOI WEI HUNG

(Student Name)

\*Delete whichever not applicable

## DECLARATION OF ORIGINALITY

I declare that this report entitled “STOCK MARKET PREDICTION USING NATURAL LANGUAGE PROCESSING(NLP)” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.



Signature : \_\_\_\_\_

Name : LOOI WEI HUNG

Date : 13/9/2024



## **ACKNOWLEDGEMENTS**

I am immensely grateful to my project supervisor, Cik Nurul Syafidah Binti Jamil, for her unconditional support and assistance throughout this time, patiently answering my questions and humorously comforting me when I felt stressed about the progress of my project. Your guidance has taught me to maintain a positive attitude and face challenges with a smile.

Also, I deeply appreciate the care from my academic supervisor, Mr. Tey Chee Chieh, has shown towards my academic journey. Your youthful enthusiasm for academia is incredibly infectious, and your continuous encouragement in facing academic challenges has been invaluable. I must say, you are a very responsible and excellent advisor.

Besides that, I want to express my gratitude to my family for providing me with emotional support at every stage of my life's growth. Thank you to my parents for their unwavering efforts to nurture me with love, allowing me to enjoy a carefree learning journey. Thank you for believing in me and allowing me to embark on the path of my own choice and always willing to be my faithful listeners. Family is the strongest support for my soul and an indispensable part of my heart. You have given me so much; this time, let me ease your worries and shoulder your burdens.

Finally, I want to sincerely thank myself. Thank you for facing every aspect of yourself and being honest with your inner self from beginning to end. Thank you for never giving up and thank you for accepting your ordinariness. You don't have to be perfect; you are you, and I will love you, love every aspect of you, every stage of you, and every state of you. Again, I wish you to be strong and clear-headed, independent, and joyful. May every step you take be driven by your passion.

## **ABSTRACT**

This project proposes a stock market prediction framework based on Natural Language Processing (NLP) to improve investment decision-making, deal with the information. overload and empower real-time decision-making. The proposed system aims to significantly enhance prediction accuracy by leveraging NLP tools to analyse unstructured textual data and extract hidden signals that might influence stock prices. Additionally, the project contributes to the evolution of Financial Technology (FinTech) and provides innovative methods and techniques to market participants to remain competitive. The report outlines the project's scope and objectives, methods and technologies involved, and makes significant contributions to the evolution of NLP research. The project's success offers significant benefits to a wide range of financial stakeholders, such as investors, financial institutions, and academicians.

# TABLE OF CONTENTS

<b>CHAPTER 1: INTRODUCTION</b>	<b>1</b>
1.1 PROBLEM STATEMENT	2
1.2 MOTIVATION	3
1.3 RESEARCH OBJECTIVES	4
1.4 PROJECT SCOPE AND DIRECTION	5
1.5 CONTRIBUTIONS	6
1.6 REPORT ORGANIZATION	7
<b>CHAPTER 2: LITERATURE REVIEW</b>	<b>8</b>
2.1 PREVIOUS WORKS ON STOCK MARKET PREDICTION USING NATURAL LANGUAGE PROCESSING (NLP)	8
2.2 STRENGTHS AND WEAKNESS	18
2.3 PROPOSED SOLUTION	20
<b>CHAPTER 3: SYSTEM MODEL</b>	<b>22</b>
3.1 STOCK PRICE PREDICTION FRAMEWORK	22
3.2 MODEL DESCRIPTIONS, EQUATIONS, AND THEIR ROLE IN THE FRAMEWORK	23
3.2.1 Support Vector Machine (SVM)	23
3.2.2 Long Short-Term Memory (LSTM)	25
3.2.3 Convolutional Neural Networks (CNN)	27
3.2.4 Recurrent Neural Networks (RNN) with Attention	28
3.2.5 FinBERT: Sentiment Analysis for Stock Price Prediction	32
3.2.6 FinBERT-SVM Hybrid Model	34
<b>CHAPTER 4: EXPERIMENTAL SETUP</b>	<b>37</b>
4.1 SYSTEM REQUIREMENT	37
4.1.1 Hardware	37
4.1.2 Software Involved	38
4.2 WORKFLOW	39
4.2.1 System Design Diagram	42
4.3 TIMELINE	54
<b>CHAPTER 5: SYSTEM SIMULATION</b>	<b>56</b>
5.1 DATA PREPARATION	56
5.1.1 Data Collection	57
5.1.2 Data Cleaning	58
5.1.3 Data Transformation	61
5.2 EXPLORATORY DATA ANALYSIS	70
5.3 SENTIMENT ANALYSIS	78
5.4 ADVANCED SENTIMENT ANALYSIS AND FURTHER DATA PREPROCESSING	83

5.5	TIME-SERIES STOCK PRICE DATA PREPROCESSING AND DATA VISUALISATION	89
5.6	MODELLING PHASE	108
5.6.1	<i>Model Data Preparation and Transformation</i>	108
5.6.2	<i>Model Defining, Building And Compiling</i>	117
5.6.3	<i>Model Training and Training Performance Evaluation</i>	123
5.6.4	<i>Model Cross Validation Evaluation</i>	130
5.6.4	<i>Model Hyperparameters and Fine-Tuning</i>	137
5.6.4	<i>Model Test-Set Evaluation</i>	144
5.6.4	<i>Model Results Visualisation</i>	148
5.6.5	<i>Hybrid Modelling</i>	167
5.7	SUMMARY	176
5.8	IMPLEMENTATION ISSUES AND CHALLENGES	177
<b>CHAPTER 6:</b>	<b>SYSTEM EVALUATION</b>	<b>179</b>
6.1	MODEL PERFORMANCE	179
6.1.1	<i>Perfomance Matrices</i>	179
6.1.2	<i>Model Train Result Evaluation</i>	183
6.1.3	<i>Model Cross Validation Evaluation</i>	187
6.1.4	<i>Model Test Set Evaluation</i>	189
6.1.5	<i>Final Hybrid Model Performance Evaluation</i>	191
6.2	FINAL HYBRID MODEL VISUALIZATION	192
<b>CHAPTER 7:</b>	<b>CONCLUSION</b>	<b>194</b>
7.1	CONCLUSION	194
7.2	FUTURE WORK	195
<b>REFERENCES</b>		<b>196</b>
<b>APPENDIX</b>		<b>A</b>

## LIST OF FIGURES

Figure 2.1	FinBERT model architecture	10
Figure 2.2	Performance Results on RMSE in testing data	10
Figure 2.3	LSTM Model Layers	11
Figure 2.4	Accuracy metrics for IBM dataset	12
Figure 2.5	Performance Results on NASDAQ-100 Index	14
Figure 2.6	SVM's Model Architecture	14
Figure 2.7	Classifier performance.	15
Figure 2.8	results between Original SVM and Ensemble SVM	17
Figure 2.9	Accuracy scoring between ANN and RNN	17
Figure 2.10	Average Error Value between the models with and without sentiment	18
Figure 2.11	Prototype of proposed solution (FinBERT-LSTM)	22
Figure 3.1	Unlabelled Stock News Headline Data and its data preparation process	26
Figure 4.2	Overall Project Framework of Stock Market Prediction	27
Figure 4.3	The type of data preparation	28
Figure 4.4	Implementation of FinBERT in sentiment Analysis and output the Sentiment Data	31
Figure 4.5	Historical Stock Price data and its data preparation	32
Figure 4.6	Models need to train in FYP 2	32
Figure 4.7	Performance Matrices of Machine Learning models	33
Figure 4.8	Gantt Chart for FYP 1	36
Figure 4.9	Gantt Chart for FYP 2.	36
Figure 5.1	The Google Colab Platform with Colab Pro Subscription	37
Figure 5.2	The Version of Python that Used for Project.	37

Figure 5.3	Read “news” Csv file and its first 10 rows of data	38
Figure 5.4	Read the last 5 rows of data	39
Figure 5.5	Remove Punctuations Function	40
Figure 5.6	Remove Unknown Symbol Function	40
Figure 5.7	Drop Duplicates Rows Function	41
Figure 5.8	Output on the remaining rows after dropped duplicates	41
Figure 5.9	“Time” data type changed from object to datetime	42
Figure 5.10	“Date” data type changed from object to datetime	43
Figure 5.11	Expand Contractions Function	43
Figure 5.12	Data type of headline changed to string function	44
Figure 5.13	Normalization process on headline column.	44
Figure 5.14	remove stopwords process on headline column	45
Figure 5.15	conversion of all numbers into words process on headline column.	46
Figure 5.16	lemmatization and tokenization process on headline column	46
Figure 5.17	Removal of the data that consists of 3 or less than 3 tokens in tokens column	47
Figure 4.18	remaining rows of data after removal data of 3 or <3 tokens	47
Figure 5.19	Maximum and minimum tokens in tokens column	47
Figure 5.20	Do truncation and padding in tokens column	48
Figure 5.21	Vectorization results in Vec column	48
Figure 5.22	Implementation of MinMax Scaler in Vec column	48
Figure 5.23	Output of scaled data in Vec column	49
Figure 5.24	Histogram and KDE plot Diagram related distribution of Scaled Data By Sentiment	51
Figure 5.25	Sentiment Score Distribution’s Pie Chart	52
Figure 5.26	Bar Chart related to the most frequent words in headlines	53
Figure 5.27	Bar Chart related to the most frequent words in headlines in term frequency	54

## LIST OF TABLES

<b>Table Number</b>	<b>Title</b>	<b>Page</b>
Table 4.1	Specifications of laptop	23
Table 4.2	Software Involved	24
Table 4.3	Data Description on provided Dataset “news.CSV”.	25

## LIST OF ABBREVIATIONS

<i>NLP</i>	Natural language Processing
<i>FinTech</i>	Financial technology
<i>LSTM</i>	Long Short-Term Memory
<i>RNN</i>	Recurrent Neural Network
<i>SVM</i>	Support Vector Machine
<i>FYP</i>	Final Year Project
<i>BERT</i>	Bidirectional Encoder Representations from Transformers



## CHAPTER 1: INTRODUCTION

The endeavour of predicting stock market movements has been a very ongoing challenge in the ever-evolving world of finance. Stock prices are volatile everyday and are strongly affected by many variables, including economic trends, news events, and investor sentiment. Individual traders and institutional investors are likely looking for the ways to acquire the competitive advantage in understanding the dynamics of the stock market. The digital era, beside that, has brought about a revolutionary change. An abundance of textual data has identified as key aspect of the financial sector, including financial news, social media opinions, viewpoints of experts, and public sentiment. This textual valuable data provides priceless insights that might mean difference between profit and loss, propelling the effort to extract actionable information from textual sources to the forefront of financial strategy.

During this shifting terrain, Natural Language Processing (NLP), a powerful aspect of artificial intelligence, takes the spotlight. NLP allows us to bridge the semantic complexity of textual data with the huge number of stock market analytics algorithms can detect small fluctuations in market sentiment, follow developing trends, and uncover crucial information that would be hidden in massive of textual data through sentiment analysis, keyword extraction, and linguistics pattern recognition.

Besides, the sentiment of investors is one of the most important factors that influence stock prices movement. When investors are positive about the market, they are more inclined to purchase stocks, and when they are pessimistic, they are likely to sell stocks. Textual data, such as news articles, posts on social media, and financial reports, may be used to extract out sentiment.

Consider the strength of NLP to identify an unforeseen shift in a CEO's tone during an earnings call, identifying patterns in social media discussions about a specific stock, or quickly summarizing the sentiment of a plethora of news articles about a specific industry sector. These are only a few examples of NLP's transformational potential in the financial sector. However, Natural Language Processing (NLP) face various obstacles when used to stock market prediction in current volatile financial scenario, despite its enormous potential. Financial markets move at breakneck pace, creating massive volumes of data in real time, and

NLP systems must struggle with the sheer volume and velocity of this data to give timely insights. Textual data sources can be loud and unstructured, making it difficult to ensure the correctness and dependability of data inputs, as errors might lead to inaccurate predictions. Furthermore, financial language is frequently sophisticated and filled with slang, sarcasm, and metaphors, requiring NLP models to cope with the complexities of financial language in order to effectively capture sentiment and trends. Because these models are subject to biases in the data on which they are trained, it is important to address ethical problems and prevent prejudice in financial sentiment research. Successful NLP projects in finance need a thorough grasp of both language processing and financial markets, emphasizing the necessity of cooperation between linguists and financial professionals in solving these complex issues.

## **1.1 Problem Statement**

### *Inaccurate Predictions*

The problem of inaccurate stock market predictions derives from traditional forecasting models' excessive reliance on numerical data and quantitative research. These models rely heavily on past market data, such as price movements and trade volumes, while mainly disregarding the importance of textual data and mood. Financial news, social media posts, analyst reports, and public sentiments are all rich sources of information that can have an immediate impact on market sentiment and stock prices. Traditional models struggle to account for the market's dynamic nature, resulting in projections that frequently overlook key subtleties caused by mood and news events. The intrinsic complexity of human behaviour in financial markets, fast swings in market sentiment, and stock market volatility all pose further challenges to the accuracy of these models.

### *Information Overload*

The difficulty of information overload in financial markets stems from the ever-growing number of unstructured textual data sources such as articles on the news, social media discussions, and expert opinions. In the modern era of technology, the vast amount of information created on a daily basis can be intimidating for investors and traders looking to make informed decisions. This deluge of data presents a big challenge since investors must not only consume it but also extract significant insights from it. The process is exacerbated further by the unstructured nature of textual data, which lacks the nicely organized arrangement of

numerical data. Investors confront the onerous task of spotting crucial signals among the noise, looking at market sentiment, and evaluating growing trends in real time.

### *Complicated Financial Language*

Traditional stock market prediction models have significant problems due to the complexities of complicated financial terminology. Financial discourse is characterized by its extensive use of industry-specific terminologies, sophisticated terminology, and nuanced subtleties that can evade simple understanding. This language complexity extends to diverse financial instruments, market dynamics, and economic indicators, making traditional models difficult to fully understand and analyze. Misinterpretations of sentiment and market trends occur frequently because these algorithms struggle to distinguish tiny differences in language, tone, and context. An apparently favourable remark in the financial industry, for example, may include underlying risks or uncertainties that need a more sophisticated analysis.

## **1.2 Motivation**

The motivation for this project stems from an essential need to improve the accuracy of stock market predictions, an important part of the financial sector. The stock market is a gauge of economic health, with a significant impact on global economies and individual financial well-being. Accurate predictions of stock price changes have enormous significance that goes beyond intellectual interest. They have a direct influence on investors, firms, and economies as a whole. In the first, this project seeks to improve investor decision-making. Individual traders managing their money or institutional investors managing large portfolios all have one aim in mind: to make informed decisions that result in favourable outcomes. The capability to accurately predict stock price changes gives investors a competitive advantage, allowing them to optimize their investment strategies, minimize risks, and maximize returns.

Furthermore, it addresses the difficulty of dealing with the digital age's information deluge. The sheer volume of textual data produced daily, which includes financial news, social media discussions, analyst reports, and company communications, provides both an opportunity and an obstacle. While this data contains key details, manually collecting and evaluating it is a daunting task. Natural Language Processing (NLP) emerges as an essential strategy for quickly navigating this data landscape and extracting hidden signals that might influence stock prices. Furthermore, this project makes significant contributions to the evolution of financial

technology (FinTech). FinTech is revolutionising established financial practices, and the financial industry has seen amazing technological innovation. We embrace the forefront of financial technology by incorporating NLP into stock market prediction, pushing the limits of what is possible. Besides, this project also makes a significant contribution to the continuous evolution of FinTech by providing innovative methods and techniques to market participants for them to remain competitive in a continually changing marketplace. Furthermore, it aims to bridge the theoretical and practical divides. Academic research is frequently restricted to theoretical domains, isolated from real-world applications. This project strives to fill that need by turning cutting-edge NLP research into useful tools and methods. This promotes a closer relationship between academia and industry, ensuring that our results have practical, real-world systems that benefit both researchers and practitioners.

### **1.3 Research Objectives**

#### *To Enhance Prediction Accuracy with NLP*

The objective is to build a stock market prediction system that leverages Natural Language Processing (NLP) methods. The major goal is to drastically enhance the accuracy of predictions by tackling a long-standing issue: the limits of traditional models that rely largely on numerical data. Traditional prediction models are reliant mostly on numerical data, such as historical price movements and trade volumes, and frequently overlook the ever-changing nature of financial markets, which are impacted by real-time textual data sources such as financial news, social media trends, analyst reports, and public sentiment. This research intends to overcome this gap by using NLP approaches and employing NLP algorithms to extract significant insights from textual material. This comprises sentiment analysis, keyword extraction, and linguistic recognition of patterns, resulting in more exact predictions for investors and financial experts.

#### *To Achieve Efficient Textual Data Processing*

The objective focuses on developing streamlined mechanisms inside the stock market prediction system to successfully handle and analyses massive amounts of textual data provided from a variety of sources, such as financial news, social media discussions, analyst reports, and public opinion. The crucial requirement here is to address the persistent issue of information overload that afflicts financial markets in the digital age. The vast amount of textual data generated daily is mind-boggling, posing both a chance and a major issue. It includes a wide range of sources, each of which has significant information, but the unstructured nature

of this data can overwhelm investors and financial experts. Sifting through this deluge manually is a tremendous task, and typical numerical data analysis is insufficient in this case. To make informed decisions, it is critical to analyse textual data rapidly, extract relevant signals, and detect growing trends in real-time. This project intends to allow investors and financial institutions to traverse this data landscape successfully by delivering efficient textual data processing with NLP, guaranteeing they can stay well-informed, optimize their investment strategies, and respond quickly to market dynamics.

### *To Handle Complex Financial Language*

The objective is to include powerful NLP algorithms into the stock market prediction system to comprehend complicated financial terminology. Financial conversations are loaded with industry-specific jargon, complicated jargon, and nuanced nuances that defy simple comprehension. These language complexities extend over a wide range of financial instruments, market dynamics, and economic indicators, leaving traditional models insufficient for complete analysis. Because of their incapacity to distinguish small linguistic differences in tone, context, and meaning, they frequently misinterpret sentiment and market trends. By using cutting-edge NLP algorithms, the stock market prediction system can successfully navigate this complex financial language terrain, allowing for a deeper and more precise comprehension.

## **1.4 Project Scope and Direction**

The primary scope of the "Stock Market Prediction Using NLP" project is to pioneer a novel and revolutionary system that improves the accuracy and efficacy of stock market predictions. At its core, this project's another crucial scope is to leverage the potential of Natural Language Processing (NLP) tools, combining the complexities of language research with the mathematical precision of financial forecasting. The project's scope encompasses several key areas, including:

- (1) Build a comprehensive stock market prediction framework.
- (2) Implement Natural Language Processing (NLP) techniques for enhanced prediction accuracy.
- (3) Process and analyse large volumes of unstructured textual data from various sources, including financial news, social media, analyst reports, and public sentiment.

- (4) Incorporate advanced NLP algorithms to decipher complex financial language, industry-specific terminologies, and nuanced subtleties.
- (5) Provide timely insights to investors and financial professionals.
- (6) Address the challenge of information overload in the digital age.
- (7) Bridge the gap between traditional numerical data analysis and the linguistic richness of textual data for stock market predictions

## **1.5 Contributions**

The project contributes significant benefits to a variety of financial stakeholders. First and foremost, investors, both individual traders and institutional corporations, stand to benefit greatly from this system. This project's improved prediction accuracy enables investors to make better informed decisions. Investors may significantly enhance their financial well-being by optimising their investing strategy, minimising risks, and maximising rewards.

Furthermore, the project provides tremendous advantages to a wide range of financial stakeholders. At its core, this approach will enormously benefit investors, both individual traders and bigger organizations. The enhanced prediction accuracy of this project allows investors to make more informed decisions.

By maximising their investment approach, reducing risks, and optimising returns, investors may dramatically improve their financial well-being. In addition, incorporating powerful NLP algorithms improves comprehension of complicated financial jargon and industry-specific terminology. This benefits not just investors by boosting the accuracy of sentiment and market trend interpretation, but it also helps financial institutions and analysts provide more accurate insights to their clients. A better understanding of market dynamics improves the financial sector, enabling improved decision-making and risk management practises.

Also, the project's emphasis on real-time decision-making is extremely important in the fast-paced world of financial markets. Investors and financial professionals may adapt quickly to shifting market mood, seize opportunities, and reduce risks, all of which contribute to their success.

## **1.6 Report Organization**

The report is divided into five sections, each of which focuses on various aspects of the "Stock Market Prediction Using Natural Language Processing" project. In chapter 1, it introduced the project's problem statement, motivation, objectives, project scope and contributions, which ended with an overview of the report organization. Then, in chapter 2, it will focus on reviews the natural language processing and their applications in financial analysis, which is focusing on the sentiment analysis with FinBERT, as well as LSTM,CNN,RNN with attention mechanism, and SVM algorithms for the sequential data analysis in stock market prediction. For the chapter 3,it will detail out project's system model, which is the models used for the whole project, that started from data preparation until Performance Evalaution, and also include the Gantt Chart for Final Year Project (FYP) 1 and Gantt Chart for FYP 2.Furthermore,chapter 4 will present the project's experimental work on the model's development of financial news sentiment analysis integration with the financial fine-tuned machine learning algorithm, which is ML model.In chapter 6,,models evaluation and results showed and produced final model, which was FinBERT-SVM Lastly, chapter 7 will conclude the whole FYP , providing a comprehensive understanding of the project's contents and outcome.

## CHAPTER 2: Literature Review

### 2.1 Previous works on Stock Market Prediction using Natural Language Processing (NLP)

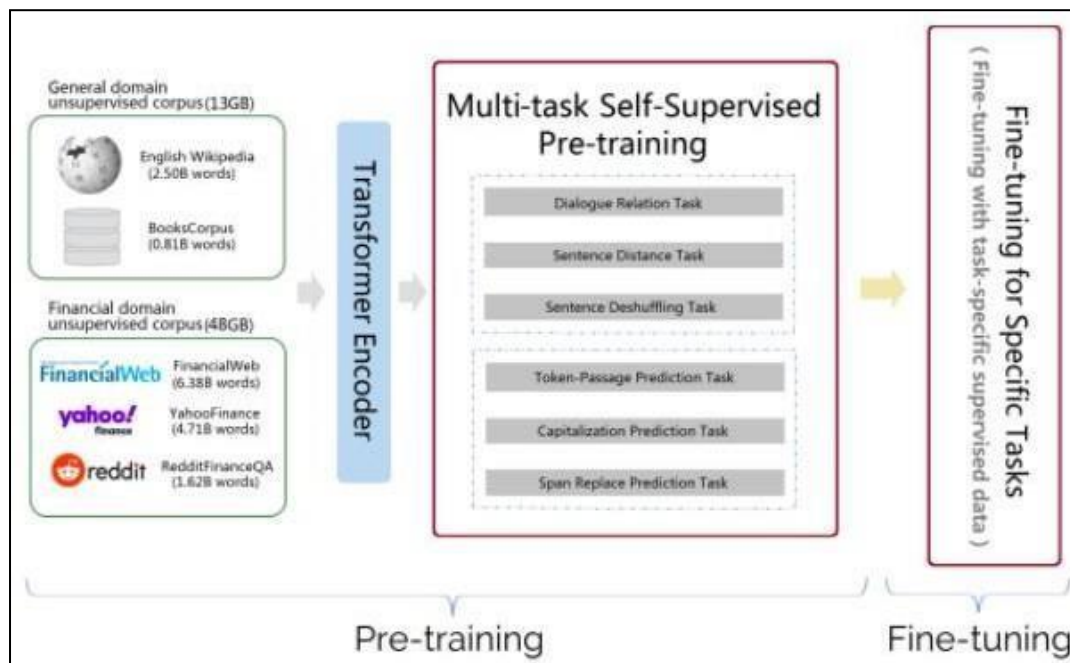
#### *Natural Language Processing (NLP) and Stock Market Prediction*

As stock market price prediction is not based on a predetermined mathematical formula, it will be a very challenging task. It is sensitive to change at any time depending on a few factors, such as changes in politics and inflation. Still, we may make reasonably accurate short-term predictions by finding patterns in the data. Machine learning techniques have been applied to overcome the short-term prediction problem, allowing humans to estimate figures that are nearly right.[1] In addition, a large number of news articles are produced every day, offering an extensive amount of helpful data for stock market analysis. These articles link positive news to exceptional performance and negative news to an organization's subpar performance. This implies that news reports can be studied to learn more about the trend of a stock. In order to extract the important insights that financial institutions and investors need to make educated decisions about stock market predictions, all textual data that contains valuable stock market information has been used as indicators to increase stock prediction rates using natural language processing (NLP). NLP can also be used to conduct sentiment analysis to understand people's sentiment or reaction, such as analyst opinions, public perceptions, and investors' emotions on news articles. As a result, the field of prediction analytics, which had previously relied heavily on traditional quantitative models as well as technical analysis of the stock's graph candle indications, has gained a fresh perspective with the introduction of NLP.



*FinBERT and Stock News Sentiment Analysis*

One excellent method of anticipating the short-term movement of equities is textual stock news information. Stock news headline sentiment analysis is a useful method for evaluating a stock's short-term performance.[1] A pre-trained NLP model called FinBERT, a BERT-based variant, is particularly good at analysing the sentiment of financial textual data from stock news stories. FinBERT is built by refining the BERT language model for financial sentiment categorization with additional training in the finance domain utilizing a large financial text collection, as J. Wang[4] has noted [2]. They achieved excellent results on FiQA's emotion scoring and Financial PhraseBank. They also achieved a 15% improvement in the categorization task's state-of-the-art accuracy. In addition, FinBERT uses textual news data as inputs for string analysis, producing three sentiment labels—positive, negative, or neutral—as well as a score between 0 and 1. Furthermore, a higher score The FinBERT analyser evaluates strings from 0 to 1 and outputs a sentiment label (positive, negative, or neutral) after analysis. Higher levels of reliability in the corresponding label are indicated by higher scores. The FinBERT algorithm's model architecture is shown in the figure below.



**Figure 2.1** FinBERT model architecture

Based on [3], K.Puh and M.B.Babac also stated that FinBERT achieved the best Root Mean Square Error (RMSE), 370.155, which is the lowest errors on average in term of calculate the square root of the average of squared differences between predicted values and true values with their data compared to Gated recurrent unit (GRU) with the pairs of price and sentiment score, GRU with only prices, Autoregressive Integrated Moving Average (ARIMA), LSTM and also Convolutional Neural Network (CNN). So that, it proved that the NLP model outperformed other algorithms, which is some of them were pure time-series models in term of doing this type of prediction task. So that, the FinBERT concluded as the best model to predict the stock price movement by analyse the news sentiment.

Model	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

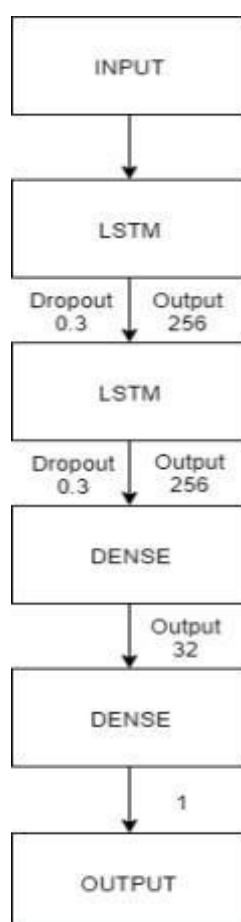
**Figure 2.2** Performance Results on RMSE in testing data[3]

### *Time Series models and Stock Market Prediction*

The environment surrounding the stock market has changed dramatically in recent decades due to technological advancements. Predicting stock prices and returns and evaluating stock market activity are only two of the many fields in which machine learning and deep learning have become indispensable. This development has greatly helped industry professionals, who now have access to more precise stock market predictions to aid in their investment decisions [4]. The field of stock market prediction has fundamentally transformed because of machine learning and deep learning, which make it easier to predict stock prices and market behavior. Additionally, it has introduced new models for market analysis and managing time-series data.

*Long Short-Term Memory (LSTM)*

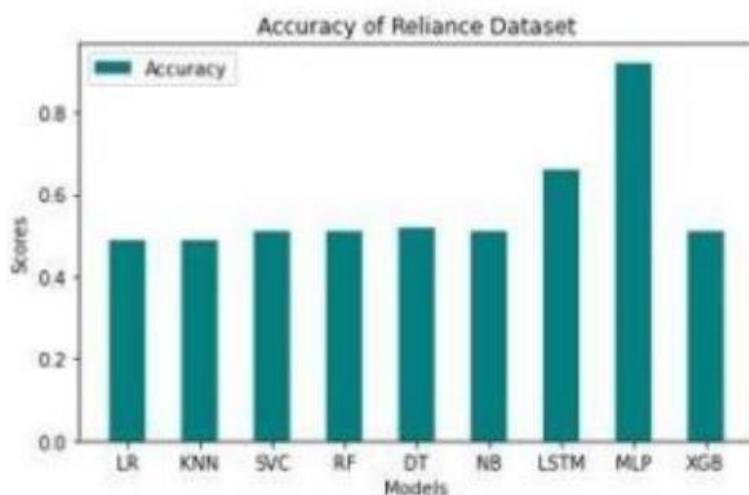
According to [5], LSTM networks are an improvement over conventional Recurrent Neural Networks (RNNs), which were created to deal with the difficulty of remembering information across longer sequences. LSTM networks are superior at managing long-term dependencies, whereas RNNs are mainly concerned with capturing links between recent and present input. The capacity of LSTM to retain knowledge from earlier phases is extremely helpful in the context of stock market prediction, where projections mostly depend on large amounts of past data. LSTM guarantees constant learning rates and improves prediction accuracy by addressing problems such as the vanishing gradient problem, which is frequently encountered in models handling huge datasets. Additionally, the model efficiently identifies pertinent features for prediction and analyses sequential data.



**Figure 2.3** LSTM Model Layers [5]

Based on [6], L.Kotapati et.al showed that the LSTM model achieved an quite better accuracy of 66% and still outperformed most of the models, which is SVM(51%), Decision Tree(51%), also Naïve Bayes(51%) and others models mentioned inside although it still lose to the highest accuracy of 92 % from MLP. This shows that the LSTM still a better model to handle time-series data in term of stock market predictions and achieved a quite good stock patterns result. The accuracy performance results of various models are shown in Figure 2.4.

It is specifically made for time series data categorization when integrating with FinBERT for hybrid model, with FinBERT handling the sentiment analysis role.[7] In addition, the Long Short-Term Memory (LSTM) model will extract features from sequential input data, such as historical stock prices and sentiment scores produced by FinBert, which are required to forecast changes in stock prices. Consequently, the sentiment results from FinBERT as well as previous stock price data will have an impact on the final predictions.



**Figure 2.4** Accuracy metrics for IBM dataset [6]

For the performance of FinBERT-LSTM hybrid model, S.Halder in [1] has examined few models with NASDAQ-100 Index, which were Multilayer Perceptron(MLP), Long Short Term Memory(LSTM) and also FinBERT-LSTM and found out that the FinBERT-LSTM has the best accuracy in stock price prediction, which was 98.59% compare to the MLP(98.23%) and LSTM(98.54%). In term of Mean Absolute Error(MAE), it was performed as the best, which is 174.94, the smallest errors on average compare to LSTM(180.58) and MLP(218.33). Also, In results of Mean Absolute Percentage Error (MAPE), FinBERT-LSTM also achieved the best

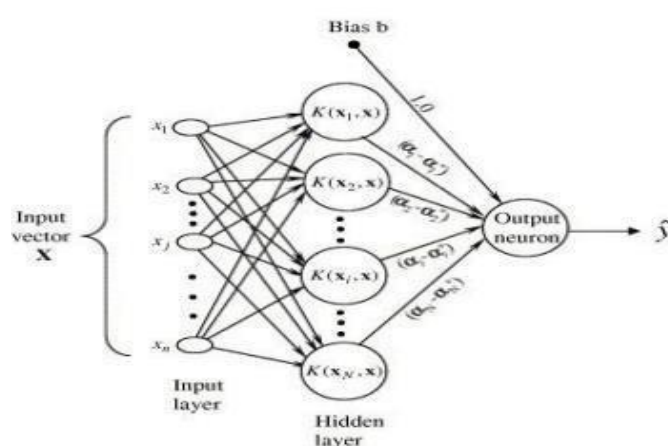
performance in result of 0.01410, smallest error percentage errors on average while compared to LSTM(0.01457) and MLP(0.01767). That's mean that integrate the news sentiment and time series classification can improve the model to gain the best stock patterns in term of prediction in stock market.

PERFORMANCE ON NASDAQ-100 INDEX			
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

**Figure 2.5** Performance Results on NASDAQ-100 Index [1]

*Support Vector Machine (SVM)*

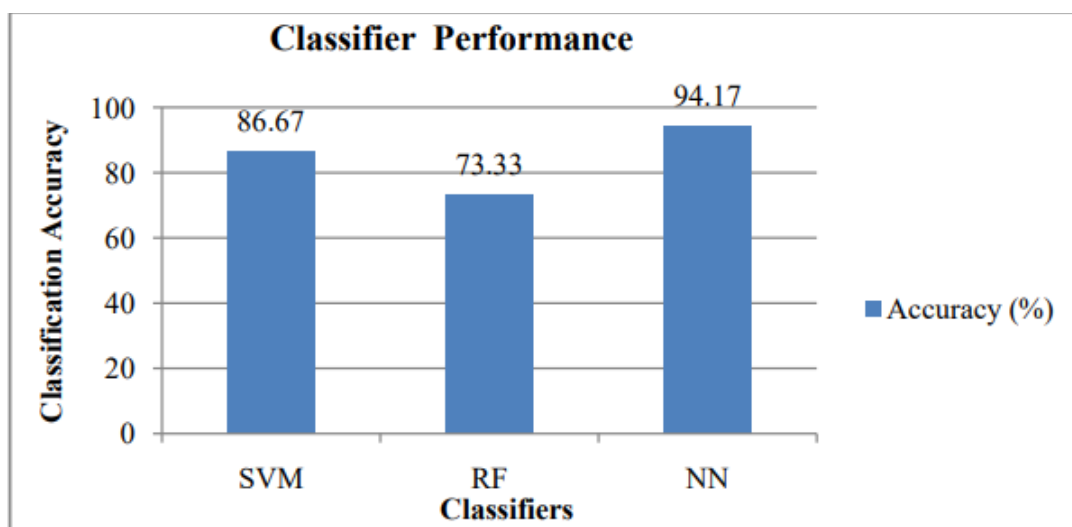
Machine learning techniques that utilize Support Vector Machine (SVM) are used to create regression analyses and classifications. An SVM operates by being given a training dataset that consists of samples that are divided into two different groups. SVM takes centre stage in this predictive modelling methodology. It takes data points and transforms them into a high-dimensional feature space. Next, it finds the best hyperplane to distinguish between positive and negative stock market performance.[6]



**Figure 2.6** SVM's Model Architecture [8]

In term of the performance of SVM , the researchers in [9] has showed its accuracy of performance matrix, which is 86.67% , better than Random Forest (RF),73.33% in this case by using the open-source available dataset using MATLAB software for training. It is considerable as the model in classification task and suitable for handling time series sequential stock data although it beat out by the Neural Network (NN) with the best score of 94.17%

Adding SVM to the hybrid model with FinBERT will involve using it first for feature extraction from the sentiment analysis output of FinBERT, which is their sentiment score, which is produced from an analysis of financial text data. After that, it will be trained to spot patterns and connections between the input features and their matching labels, enabling it to predict the stock market's most likely future movement.



**Figure 2.7** Classifier performance [9]

Based on the [10], J.X.Liu et.al stated the performance of ensemble SVM after integrated with the FinBERT were higher than the original SVM in overall results as shown in Figure 2.8. The best F1 score that achieved by ensemble SVM is 65.30% ,while original SVM achieved the best one of 62.19%. This proved that Integrated with FinBERT is absolutely an improved way on performance matrices for ensemble SVM and outperformed the original-based SVM in order to increase the prediction rates in stock market price.

**Table 7**

**The results of forecasting price movement.**

	Original SVM		Ensemble SVM	
	F1-score	Training size in window	F1-score	Training size in window
Exp. 1	53.57%	236, 238	55.02%	240
Exp. 2	59.74%	246	61.41%	234
Exp. 3	58.72%	234	60.16%	234
Exp. 4	62.19%	236	65.30%	238
			(Proposed Method)	
Exp. 5	61.04%	238	61.66%	232

**Figure 2.8** results between Original SVM and Ensemble SVM [10]

*Recurrent Neural Networks (RNN)*

Recurrent Neural Networks (RNNs) are widely used in many different applications due to its built-in "memory" mechanism, which allows them to absorb information from previous elements in a sequence during output creation. A directed graph is formed along the sequence by the interconnected units that set RNNs apart and allow them to perform jobs where the context provided by previous elements is necessary for output production. Specifically, RNNs are very useful for analysing sequential data, such as past stock prices, or market indicators, in the context of stock prediction analysis. RNNs can identify underlying patterns and trends that could impact future market behaviour by identifying temporal dependencies in the data. This is a useful predicting tool for traders and investors.[12].

Based on [11], the researchers found out that RNN is outperformed the Artificial Neural Network (ANN), which were 87.32 % compared to only 58% in term of accuracy. This means that it will be a good option to choose for training as models for sequential data in form of time series for stock market data. And can achieved good results for the stock price movement and patterns.

<b>Comparison</b>			
<b>Research Work</b>	<b>Data Set</b>	<b>Algorithm</b>	<b>Accuracy Score</b>
Yumo Xu et al [10] 2018	Google Stock Price From GitHub	Artificial Neural Networks (ANN)	58%
Our Research Work	Google Stock Price From Kaggle	Recurrent Neural Network (RNN) [3]	87.32%

**Figure 2.9** Accuracy scoring between ANN and RNN [11]

Since RNNs are specifically made to handle sequential data, they are a good choice for applications like analysing historical stock prices or market indicators. This is because they can be integrated with the FinBERT. They generate output by processing data sequentially, considering the pieces that came before it. Additionally, it functions in combination with RNNs and FinBERT's sentiment analysis features. RNNs process sequential data, such as historical



stock prices or market indicators, and add further insights to FinBERT's analysis by processing it. FinBERT does sentiment analysis on financial text data. Then, as it goes through the training process, it may adjust its parameters to maximize performance and boost accuracy in order to produce the best prediction. It will then discover stock patterns and relationships within the data.[2]

Based on [13], the researchers just mentioned that they are using FinBERT- RNN model to do the stock price prediction task. However, there has no results related to its performance matrices like accuracy showing in this paper due to the paper not accessible to public for free of charge. So that, LSTM-GRU is used to integrate with FinBERT , which is similar to RNN model as to shows the results of the model , that might closely to same performance as RNN model. Therefore, in [14], the LSTM-GRU model with sentiment is smaller than the ones without sentiment in term of average error value. For example, in term of RMSE, the LSTM-GRU model with sentiment achieved the lowest of 10.02 compared the one without sentiment, which was scored 11.31 in its lowest record. This means that integrated with the FinBERT can improved significantly as hybrid model to increase the prediction rated in the area of stock market to gain better insights in term of investing.

[Solana] Average Error Value				
Sentiment	Model	MAE	MAPE	RMSE
Not Included	LSTM	16.65	7.97%	19.36
	GRU	13.38	6.51%	15.28
	LSTM-GRU	9.58	4.64%	11.31
Included	LSTM	16.20	7.76%	18.91
	GRU	11.14	5.47%	13.20
	LSTM-GRU	8.41	4.13%	10.02

**Figure 2.10** Average Error Value between the models with and without sentiment [14]

## 2.2 Strengths and Weakness

Model/ Algorithm	Strength	Weakness
<b>FinBERT</b>	<ul style="list-style-type: none"> <li>• Better sentiment understanding, which assists in identifying the sentiment of news articles, which has an impact on stock prices, by offering positivity, negativity, and neutrality scores.[1]</li> <li>• <i>Domain-driven Analysis</i>: designed specifically for financial text analysis, it can comprehend and evaluate sentiments specific to finance industry.[1]</li> </ul>	<ul style="list-style-type: none"> <li>• Accuracy decreased when the word orderings in sentences randomized [13].</li> </ul>
<b>LSTM</b>	<ul style="list-style-type: none"> <li>• <i>Long-Term Dependency Handling</i>: Able to capture long-term dependencies in sequential data by solving the vanishing gradient issue[1]</li> </ul>	<ul style="list-style-type: none"> <li>• Costly as well as difficult to understand and train: Needs an extensive amount of computational power, textual data, and training time.[15]</li> </ul>

<p><b>SVM</b></p>	<ul style="list-style-type: none"> <li>• <i>Robust to overfitting:</i> Their regularization parameters that prevent overfitting issue even in the case of limited training data.</li> <li>• <i>Better handling with sequential data:</i> Helpful in time series analysis and suited for processing the sequential data like stock price changes over time, where the order of the data points is important.[1]</li> </ul>	<ul style="list-style-type: none"> <li>• <i>Lack of interpretability:</i> less interpretable than some other models because give limited insight into the underlying relationships in the data.[7].</li> </ul>
<p><b>RNN with Attention Mechanism</b></p>	<ul style="list-style-type: none"> <li>• <i>Variable-length sequences:</i> Adaptable for scenario where the length of input data might vary since able to handle input sequences of different length.[7]</li> </ul>	<ul style="list-style-type: none"> <li>• <i>Training instability:</i> RNN's training can unstable, particularly when deal with lengthy sequences, as errors may vanish in process of back-propagation.[7]</li> <li>• <i>Difficult to capture the complex patterns:</i> Struggling in capturing complex patterns for sequential data While dealing with tasks that require precise timing.</li> </ul>

### 2.3 Proposed Solution

The proposed solution for the "Stock Market Forecasting Using NLP" project introduces a hybrid model that fuses FinBERT (Financial Bidirectional Encoder Representations from Transformers) and SVM (Support Vector Machine) (FinBERT-SVM) architectures. This hybrid model aims to combine the unique strengths of each model to enhance the accuracy and robustness of stock market predictions.

FinBERT, a model based on transformer architecture, specializes in grasping the Financial contextual intricacies of language. It gains proficiency by undergoing pre-training on extensive financial textual data, allowing it to comprehend complex relationships between words and phrases in textual information from financial news headlines. In the realm of stock market prediction, FinBERT's proficiency lies in its ability to interpret the nuances of financial news and analyst reports. This capability enables the model not only to gauge sentiments as positive or negative or neutral but also to grasp the subtleties in language that can sway stock prices. Conversely, SVM represents a recurrent neural network design tailored for the analysis of sequential data. Its effectiveness is evident in capturing dependencies and patterns within time-series data, which is critical for predicting stock market trends. SVM's sequential learning capabilities enable it to identify trends, cycles, and recurring patterns within historical stock price data. This sequential analysis complements FinBERT's contextual understanding as it zeroes in on the temporal facets of market behaviour.

The integration of FinBERT and LSTM results in a symbiotic relationship. FinBERT excels at preprocessing financial textual data, extracting significant features and embeddings that LSTM can subsequently utilize for sequential analysis. The amalgamation of contextual comprehension and sequential learning in the hybrid model leads to more accurate and insightful stock market predictions. This approach considers a more extensive array of financial textual data sources, acknowledges their historical context, and adapts to evolving market dynamics. An essential benefit of this hybrid model lies in its refined sentiment analysis and time-series classification. While FinBERT enhances sentiment polarity detection and intensity assessment, LSTM further hones this analysis by incorporating historical trends. This fusion results in a more nuanced grasp of market sentiment, equipping the model to detect subtle shifts that can potentially influence stock prices.

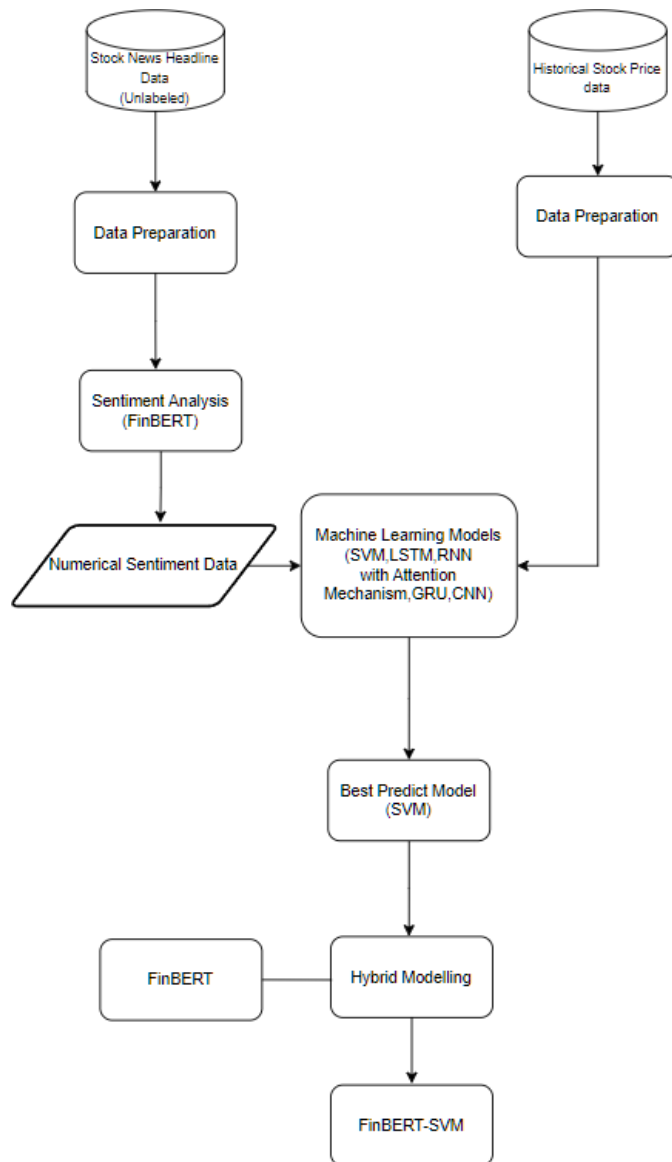


Figure 2.11 Prototype of proposed solution (FinBert-SVM)

## Chapter 3: System Model

### 3.1 Stock Price Prediction Framework

The Stock Price Prediction Framework integrates both machine learning and deep learning models to forecast the next day's Close price of a stock. This framework leverages a variety of quantitative features, such as historical stock prices, trading volume, and technical indicators, alongside qualitative insights derived from market sentiment analysis, which is processed using FinBERT. The dataset used in the framework includes several key features: '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. 'Headline' is a textual feature that contains financial news, which is processed using FinBERT to generate sentiment insights. 'sentiment\_score' is a numerical value generated by FinBERT that captures the sentiment of the 'Headline' data, categorizing it as positive, neutral, or negative. 'Close\_lag1' and 'Close\_lag2' represent the stock's closing price from the previous days, enabling the model to capture the temporal dependencies in stock price behaviour. 'Close\_rolling\_mean' and 'Close\_rolling\_std' provide rolling statistics, such as the average price and volatility over a certain time window, helping the model understand the stock's trend and variability.

The framework utilizes six distinct models to maximize prediction accuracy. The Support Vector Machine (SVM) is designed to capture non-linear relationships between features like Volume and *sentiment\_score*, which makes it ideal for modelling the complex interactions within the dataset. Long Short-Term Memory (LSTM) networks are incorporated to handle long-term dependencies in sequential stock price data, utilizing memory cells to retain crucial historical patterns. Convolutional Neural Networks (CNN) are applied to detect local patterns and short-term trends in stock price data, such as sudden spikes in price or volume.

In addition to these, Recurrent Neural Networks (RNN) with Attention mechanisms prioritize critical time steps by assigning weights to important events or trends, ensuring that key moments in the stock price history have a larger influence on predictions. The framework also includes a hybrid model called FinBERT-SVM, which combines FinBERT's ability to analyze sentiment from textual financial data with the SVM model's capacity to model non-linear relationships between market sentiment and stock prices. Lastly, FinBERT itself is used

as an independent model to generate `sentiment_score` from `Headline` data, integrating market sentiment directly into the stock price prediction process.

### 3.2 Model Descriptions, Equations, and Their Role in the Framework

#### 3.2.1 Support Vector Machine (SVM)

##### *Framework Structure and How It Works*

The Support Vector Machine (SVM) is a supervised learning model primarily used to handle non-linear relationships between stock price features and the target Close price. In stock markets, the relationship between features like ‘Volume’, ‘`sentiment_score`’, and stock price movement is often non-linear. The SVM model employs a Radial Basis Function (RBF) kernel to map the input features into a higher-dimensional space where these non-linear relationships become easier to model.

The SVM works by identifying support vectors, which are key data points that define the decision boundary in the transformed feature space. In this case, the support vectors are the most influential stock price data points that help in determining the stock’s future price.

[ SVM Equation ]

The SVM model is based on the following mathematical equation:

$$f(x) = \sum_{i=1}^n \alpha_i K(x_i, x) + b$$

Where:

- $x$  is the input feature vector, which includes ‘Volume’, ‘`sentiment_score`’, ‘`Close_lag1`’, ‘`Close_rolling_mean`’, and ‘`Close_rolling_std`’.
- $K(x_i, x)$  is the Radial Basis Function (RBF) kernel

$$K(x_i, x) = \exp(-\gamma ||x_i - x||^2)$$

- where  $\gamma$  is a hyperparameter that controls the spread of the RBF kernel, determining how flexible or complex the decision boundary will be.

- $\alpha_i$  are the learned coefficients associated with each support vector  $x_i$ .
- $b$  is the bias term that adjusts the model's prediction

*Explanation*

- **RBF Kernel:** The RBF kernel is essential for mapping the input features into a higher-dimensional space, where non-linear relationships become linear and easier to model. For example, the relationship between 'Close' and 'sentiment\_score' may not be straightforward, but the kernel transformation allows the SVM to model these interactions more effectively.
- **Support Vectors:** The SVM model does not rely on all data points; instead, it focuses on a subset of key data points called support vectors. These vectors are close to the decision boundary and define the model's predictive power, allowing it to generalize well even with complex, non-linear data.

*Role in the Framework*

The SVM model is critical for modeling non-linear relationships between features like Volume and sentiment\_score and stock price movements. By transforming the input data using the RBF kernel, SVM enables the framework to handle cases where stock price movements are not directly proportional to changes in features. This is especially useful in predicting stock prices influenced by market sentiment, where changes in sentiment\_score may have a significant but non-linear impact on stock price.



## 3.2.2 Long Short-Term Memory (LSTM)

*Framework Structure and How It Works*

The LSTM model is a type of Recurrent Neural Network (RNN) that is designed to handle sequential data by maintaining a memory of past information over long periods. Stock prices are inherently sequential, and the price movements on one day are often influenced by prior trends and patterns. The LSTM model is particularly useful in this context because it can learn and remember long-term dependencies in stock prices.

An LSTM unit consists of three gates—the forget gate, input gate, and output gate—which together control the flow of information. The forget gate decides how much of the past information to retain, the input gate determines how much of the new information to store, and the output gate decides what part of the information to pass on to the next step.

[ LSTM Equations ]

The LSTM model works through the following key equations:

**1. Forget Gate:**

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

The forget gate controls how much of the past information  $C_{t-1}$  should be retained.

**2. Input Gate:**

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\hat{A}_t = \sigma(W_A \cdot [h_{t-1}, x_t] + b_A)$$

The input gate determines how much new information from the current input should be added to the cell state.

### 3. Cell State Update:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \hat{A}_t$$

The cell state  $C_t$  is updated based on both the forget gate and input gate.

### 4. Output Gate:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] b_o)$$

$$h_t = o_t \cdot \tanh(C_t)$$

The output gate decides what information from the cell state will be passed to the next hidden state  $h_t$ .

#### *Explanation*

- **Forget Gate:** The forget gate  $f_t$  is responsible for deciding what portion of the previous information should be discarded. This is critical in stock price prediction, where not all historical data is equally important. For example, the model may choose to forget information about price fluctuations that occurred months ago if they are no longer relevant.
- **Cell State:** The cell state  $C_t$  retains long-term memory of the stock price trends. This is particularly useful in capturing recurring patterns in stock prices, such as weekly or monthly trends.
- **Output Gate:** The output gate decides which parts of the information stored in the cell state should influence the next prediction.

#### *Role in the Framework*

The LSTM model is vital for capturing long-term dependencies in stock prices. It allows the framework to learn from extended sequences of stock price data, ensuring that important trends and patterns are preserved over time. For example, the LSTM model might learn that a

consistent rise in ‘Close\_lag1’ over several days often leads to an upward trend in stock prices. By incorporating ‘sentiment\_score’ along with traditional stock price features, the LSTM model can better predict future price movements based on both quantitative and qualitative data.

### 3.2.3 Convolutional Neural Networks (CNN)

#### *Framework Structure and How It Works*

The Convolutional Neural Network (CNN) is designed to identify local patterns and short-term trends in stock price data. While CNNs are traditionally applied in image recognition tasks, they have also proven effective in time-series data analysis, such as stock prices. In the context of stock price prediction, CNNs capture small, localized patterns within sequences of input features, such as sudden spikes in trading volume or sharp price movements over a short period.

The CNN model operates by applying convolutional filters over input sequences, such as stock prices or sentiment scores, to extract relevant features. These features are then passed through pooling layers to reduce the dimensionality and retain only the most important patterns, followed by fully connected layers to combine the learned features and make a prediction. In this framework, CNN focuses on detecting short-term variations in the stock prices, such as rapid changes in the ‘Volume’, ‘Close’, and ‘sentiment\_score’, that might indicate significant price movements.

[ CNN Equation ]

The convolution operation in a 1D CNN for stock price data can be represented as:

$$h_i = \sum_m X_{i+m} w_m + b$$

Where:

- $X_{i+m}$  represents the window of input data, such as stock prices, ‘Volume’, or ‘sentiment\_score’, over a specific time period.
- $w_m$  are the filter weights learned by the model to detect important patterns in the input sequence.

- $h_i$  is the output feature map, which represents the presence of certain patterns in the stock price data.
- $b$  is the bias term added to the convolution output.

After the convolution operation, pooling layers are applied to reduce the size of the feature maps, focusing on the most relevant information while discarding unnecessary details.

#### *Explanation*

- Convolutional Filters: CNNs use multiple filters to scan through input sequences (such as ‘Volume’ or ‘Close’ prices) and detect important short-term trends or anomalies. These filters are tuned to capture local relationships in the data, such as the impact of a sudden increase in ‘Volume’ or changes in the ‘sentiment\_score’ on the stock price over a few days.
- Feature Maps: Each filter produces a feature map, which highlights areas in the data where certain patterns are detected. For example, a filter might detect a sharp price spike, which could indicate a significant short-term trend that affects the stock price.
- Pooling: After the convolution, pooling reduces the dimensionality of the feature maps by retaining only the most important information. This ensures that the model focuses on key trends without overfitting to minor fluctuations.

#### *Role in the Framework*

The CNN plays a crucial role in capturing short-term trends and local patterns in the stock price data. By focusing on localized features, such as sudden changes in Volume or price movements influenced by market sentiment, CNN enhances the framework’s ability to detect and react to rapid fluctuations in stock prices. For instance, if a stock experiences a sudden surge in ‘Volume’ following a positive news headline, the CNN can detect this short-term pattern and adjust the stock price prediction accordingly. This makes CNN particularly useful for identifying trends that might not be visible when analyzing longer time frames or more complex relationships captured by other models like LSTM or SVM.

#### 3.2.4 Recurrent Neural Networks (RNN) with Attention

##### *Framework Structure and How It Works*

Recurrent Neural Networks (RNN) are designed to model sequential data by maintaining a hidden state that updates with each time step. While standard RNNs are powerful in capturing dependencies over time, they can struggle with long sequences where distant past information is important for making current predictions. The Attention mechanism was introduced to address this limitation by allowing the model to focus more heavily on the most relevant parts of the sequence.

In stock price prediction, the RNN with Attention mechanism enables the model to weigh certain historical events or trends more heavily than others. For example, the model might assign more importance to time steps where there was significant price volatility or where sentiment scores spiked, compared to more routine price movements. By focusing on these critical time steps, the RNN with Attention is better equipped to make accurate predictions based on both the sequence of stock prices and other features like volume and sentiment.

[ RNN with Attention Equations ]

The RNN model processes sequential data using the following key equations:

### 1. Hidden State Update:

$$\mathbf{h}_t = \sigma(W_h \cdot [h_{t-1}, x_t] b_h)$$

Where:

- $\mathbf{h}_t$  is the hidden state at time  $t$ , which is influenced by both the previous hidden state  $h_{t-1}$  and the current input  $x_t$  (example, stock prices, volume, sentiment).
- $W_h$  are the learned weights, and  $b_h$  is the bias term.

### 2. Attention Weights Calculation: The attention mechanism calculates a score for each time step to determine its importance:

$$\mathbf{e}_t = v_a \cdot \tanh(W_a \cdot h_t b_a)$$

where:

- $\mathbf{e}_t$  represents the alignment score for time step  $t$ , measuring how relevant the hidden state  $h_t$  is to the prediction.

- $W_a$  and  $v_a$  are learned weight matrices, and  $b_a$  is the bias term.
- 3. Attention Score (Softmax):** The attention weights  $\alpha_t$  are computed using a softmax function to normalize the alignment scores:

$$\alpha_t = \frac{\exp(e_t)}{\sum_{k=1}^n \exp(e_k)}$$

The attention weight  $\alpha_t$  represents the importance of the hidden state at time step  $t$  relative to the entire sequence.

- 4. Context Vector:** The context vector  $c$  is computed as a weighted sum of the hidden states:

$$c = \sum_{t=1}^T \alpha_t \cdot h_t$$

where  $T$  is the total number of time steps. The context vector  $c$  summarizes the relevant parts of the sequence, incorporating the most important time steps according to the attention weights.

- 5. Final Prediction:** The context vector  $c$ , along with the hidden state, is used to make the final prediction:

$$\hat{A} = W_{\hat{A}} \cdot c + b_{\hat{A}}$$

where  $W_{\hat{A}}$  are learned weights and  $b_{\hat{A}}$  is the bias.

*Explanation*

- **Hidden State:** The hidden state  $h_t$  summarizes the information from previous time steps and is updated as new data (example, stock prices or volume) is processed. This allows the RNN to maintain a memory of past events.
- **Attention Mechanism:** The attention mechanism assigns different weights  $\alpha_t$  to different time steps based on their relevance to the prediction. This allows the model to focus on critical time steps, such as price movements during a news announcement or a period of high volume.
- **Context Vector:** The context vector  $c$  captures the most important information from the entire sequence, using the attention weights to prioritize relevant time steps over less important ones.

*Role in the Framework*

The RNN with Attention is crucial for handling sequences of stock price data where some time steps are more important than others. In stock markets, not all events are equally influential—major announcements, significant shifts in sentiment, or high trading volumes might have a greater impact on future stock prices than other, more routine fluctuations. The attention mechanism allows the model to focus on these critical events, ensuring that they have a larger influence on the final stock price prediction. For example, if a significant price spike occurred due to a positive earnings report, the attention mechanism ensures that the model gives this event more weight, improving the prediction accuracy.

### 3.2.5 FinBERT: Sentiment Analysis for Stock Price Prediction

#### *Framework Structure and How It Works*

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 or market reports, to extract sentiment information that can be used to predict stock prices. In the context of stock price prediction, market sentiment often plays a key role in influencing stock prices, especially when driven by major news events. FinBERT converts qualitative data (news headlines) into a ‘sentiment\_score’ that reflects the market’s emotional or behavioral reaction to the news.

The sentiment score can be categorized as positive, neutral, or negative and is used as an additional input feature in the stock price prediction models. By incorporating sentiment analysis, the framework becomes more dynamic, as it can react to real-time news and market events.

[ FinBERT Sentiment Score Equation ]

The process of generating a sentiment score using FinBERT is as follows:

$$sentiment\_score = f_{FinBERT}(Headline)$$

Where:

- $f_{FinBERT}$  is the FinBERT model function that processes the input text (news headlines or financial reports) and outputs a sentiment score.
- ‘Headline’ refers to the textual input data, which contains news articles or financial information.
- The output is the ‘sentiment\_score’, a numerical value that represents the sentiment (positive, neutral, or negative).

#### *Explanation*

- Pre-processing: The input text (‘Headline’) is first tokenized and converted into word embeddings using BERT’s transformer architecture. These embeddings capture the contextual meaning of each word in the sentence.
- Contextual Understanding: FinBERT, having been pre-trained on large-scale financial data, is able to understand the specific nuances of financial language. For instance, it



recognizes terms like “earnings growth” or “regulatory scrutiny” within a financial context and assigns appropriate sentiment to them.

- Output: The model outputs a ‘sentiment\_score’, which is typically represented as a numerical value. For example, positive sentiment might be represented by a score of 1, neutral sentiment as 0, and negative sentiment as -1.

#### *Role in the Framework*

- FinBERT plays a crucial role in providing the framework with qualitative insights from news and financial reports, which can heavily influence stock prices. By converting textual information into a ‘sentiment\_score’, FinBERT allows the prediction models to factor in market sentiment, something that purely quantitative models may overlook. For instance, a series of negative news articles about a company could signal a potential drop in its stock price. FinBERT captures this sentiment and feeds it into the prediction models, allowing the framework to react to changes in market perception and improve the accuracy of its predictions.
- FinBERT’s integration into the stock price prediction framework ensures that the model is not solely reliant on historical price data but also takes into account the real-time qualitative sentiment driving market behavior. This makes the framework more responsive to external factors, such as news or regulatory changes, which can have an immediate impact on stock prices.

### 3.2.6 FinBERT-SVM Hybrid Model

#### *Framework Structure and How It Works*

The FinBERT-SVM model is a hybrid approach that combines FinBERT's sentiment analysis capabilities with the Support Vector Machine (SVM)'s ability to model non-linear relationships between stock features. This hybrid model aims to capture both qualitative data (example , sentiment from financial news headlines) and quantitative data (example., stock price, volume, and technical indicators) to predict future stock prices.

FinBERT processes textual data (such as news headlines and financial reports) and converts it into a `sentiment_score` (positive, neutral, or negative). This sentiment score, along with traditional stock market features like 'Volume', 'Close\_lag1', and 'Close\_rolling\_mean', is used as an input to the SVM. The SVM then models the complex, non-linear relationships between these inputs and the predicted Close price of the stock.

The strength of the FinBERT-SVM model lies in its ability to combine market sentiment with traditional financial features, providing a more holistic prediction of stock prices. Market sentiment often drives short-term price movements, while traditional features capture long-term trends and price behavior. By integrating both into a single model, FinBERT-SVM ensures that the influence of real-time news and external events is factored into the stock price predictions.

[ FinBERT-SVM Equation ]

#### 1. FinBERT Sentiment Score:

$$sentiment\_score = f_{FinBERT}(Headline)$$

Where:

- $f_{FinBERT}$  represents the FinBERT model, which processes 'Headline' data (news, reports, etc.) and generates a sentiment score.
- 'Headline' refers to the input textual data (example., news headlines or financial articles).
- The output is a 'sentiment\_score' that reflects the sentiment of the market (positive, neutral, or negative).

## 2. SVM Prediction

$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i K(x_i, \mathbf{x}) + b$$

Where:

- $\mathbf{x}$  is the input feature vector, which includes traditional financial features like ‘Volume’, ‘sentiment\_score’ from FinBERT, ‘Close\_lag1’, ‘Close\_rolling\_mean’, and ‘Close\_rolling\_std’.
- $K(x_i, \mathbf{x})$  is the Radial Basis Function (RBF) kernel

$$K(x_i, \mathbf{x}) = \exp(-\gamma \|x_i - \mathbf{x}\|^2)$$

- where  $\gamma$  is a hyperparameter that controls the flexibility of the RBF kernel, determining how it transforms the input space
- $\alpha_i$  are the learned coefficients associated with each support vector  $x_i$ , which is represent critical points in input space that define the decision boundary
- $b$  is the bias term that adjusts the model’s prediction based on input data

### *Explanation*

- **FinBERT Sentiment Score:** FinBERT processes the input text (example, news headlines) and generates a ‘sentiment\_score.’ This score is a key indicator of how the market views the stock, with positive sentiment typically leading to increased demand and negative sentiment potentially signalling a price drop. The sentiment score is then fed into the SVM model alongside other financial features to improve prediction accuracy.
- **SVM Prediction:** The SVM uses the ‘sentiment\_score’ from FinBERT along with traditional stock features to predict the next day’s Close price. The RBF kernel maps the input features into a higher-dimensional space, enabling the SVM to capture complex, non-linear relationships between sentiment, volume, lagged stock prices, and the target stock price. The support vectors define the decision boundary in this space, allowing the model to generalize well on unseen data.

*Role in the Framework*

The FinBERT-SVM model serves a dual purpose by combining sentiment analysis with financial data modeling. In the stock market, price movements are often influenced not only by historical trends but also by external factors, such as news headlines, earnings reports, and public sentiment. Traditional stock price prediction models that rely solely on quantitative data may fail to capture the impact of real-time news and market sentiment, which can drive sudden price changes. FinBERT-SVM addresses this gap by integrating sentiment as a core feature.

1. **Market Sentiment Integration:** FinBERT converts textual data into a ‘sentiment\_score’ that reflects the market’s perception of the stock. For example, positive news about a company’s earnings can drive up the stock price, even if traditional financial indicators suggest otherwise. FinBERT-SVM captures this behavior by factoring in market sentiment alongside traditional stock features like ‘Volume’ and ‘Close\_lag1’.
2. **Non-linear Modeling with SVM:** SVM with the RBF kernel is particularly effective at capturing non-linear relationships between features. For instance, the impact of a negative ‘sentiment\_score’ may not be linear with respect to stock price—minor negative news might have a small impact, but major scandals or economic events could trigger large price drops. The SVM model allows the framework to handle these complex relationships more effectively.
3. **Holistic Stock Prediction:** By combining qualitative insights from FinBERT and the quantitative modeling power of SVM, the FinBERT-SVM model provides a more comprehensive approach to stock price prediction. It ensures that both real-time market sentiment and long-term price patterns are considered, improving the accuracy of the overall stock price prediction framework.

For example, a series of negative news reports about a company’s leadership or earnings could lead to a negative sentiment score from FinBERT. When combined with rising Volume and past price movements (example., ‘Close\_lag1’), the FinBERT-SVM model might predict a significant price drop, even if traditional indicators alone do not show such a trend.

## Chapter 4: Experimental Setup

The development of the proposed stock market prediction framework has different phases included, Stock News Headline Data Preparation, Historical Stock Price Data Preparation, Sentiment analysis, Final Processed Data Preparation, Modelling, Initial Model Performance Evaluation, Best Model Selection, Hybrid Modelling and finally the Hybrid Model Performance Evaluation.

### 4.1 System Requirement

#### 4.1.1 Hardware

The hardware involved in this project is primarily a computer. A computer is used for various essential tasks within the stock prediction system, including data collection, preprocessing, and model training. It plays an important role in running the Natural Language Processing (NLP) algorithm, which is FinBERT for sentiment analysis. Additionally, the computer is essential for integrating NLP model, which is FinBERT with time series models, like SVM, LSTM, CNN and RNN to enhance prediction accuracy for the stock price. Furthermore, it supports the real-time prediction component, enabling the trained models to generate stock predictions.

**Table 4.1** Specifications of laptop

Description	Specifications
Model	Dell Precision 5480 Workstation
Processor	Intel Core i9-13900H
Operating System	Windows 11 Pro
Graphic	Integrated: Intel® Iris Xe Graphics Discrete: NVIDIA® RTX 2000 Ada Generation laptop GPU, 8 GB GDDR6
Memory	64GB, LPDDR5, 6000MHz, integrated, dual-channel RAM
Storage	M.2 2280, 512 GB, PCIe NVMe Gen4 x4, Class 40 SSD

## 4.1.2 Software Involved

**Table 4.2** Software Involved

1	Python
2	Google Colab
3	Microsoft Excel

**Python:** Python serves as the core programming language, providing versatility for implementing NLP algorithm, like FinBERT and machine learning models like SVM, RNN and LSTM. Key libraries such as NLTK, NumPy, Seaborn, Pandas, Matplotlib/Seaborn will play pivotal roles in Exploratory Data Analysis, NLP (sentiment analysis), and predictive modelling on time series stock data in term of machine learning. It's is important during all phases of development for the whole FYP 1 and 2 to build out the final model for the project as it has lots of package and library that are essential for project development.

**Web-based IDE Platform:** Google Collaboratory, a robust Web Integrated Development Environment platform providing a collaborative and efficient environment for the project's NLP and machine learning models development and training. Besides that ,it is using the Python programming language and the coding project will automatically stored on the Google Cloud Server, which is more convenience for users to minimalize the risk of file losing. The machine learning modelling part will be on the planning in FYP2, while the NLP(sentiment analysis)part will be on existing planning , which is FYP 1.

**Microsoft Excel:** Excel serves as the application to use for the data collection and also the data storing. The dataset “news.csv ” are stored in Excel and will search the historical stock price data for FYP 2 in csv file format, which is also need to stored in the Microsoft Excel.

## 4.2 Workflow

### *Data Preparation*

The Stock Market Prediction Project will be using the news headlines data provided by Github [16]. The dataset consists of 3690 stock news headlines content textual data that related to Company (Apple Inc), which was collected from December 2018 until February 2024 in related to the Apple Incorporation that listed on NASDAQ Index with symbol (NASDAQ:AAPL). Each headline is associated to the corresponding date and time provided in the dataset. Also, there are sources of news agencies that linked with each headline in dataset. Since this dataset is unlabelled, there was no any sentiment scores and class label found inside the dataset and will be used for sentiment analysis by implement the NLP algorithm, which is the FinBERT model.

Beside using the news headlines data, it also using the 6 historical stock price data which consisted of 253,19,251,252,253 and 254 rows respectively. These numerical historical stock data that related to Apple Inc, which were collected from December 2018 until February 2024. Each stock price data is associated to the corresponding date and time provided in the their datasets. Also, there were 6 columns of the Apple stock prices data, which were Date, Open, High, Low, Close and Volume. The Table 4.3 below shown about both stock news headlines and historical stock price data data description .

**Table 4.3** Data Description on provided Dataset “news.CSV”. Source from [15]

Data Field	Description
Date	Date of the Apple’s stock news headlines published which is the dates are between December 2018 and February 2024.
Time	Time of the Apple’s stock news headlines published
Source	Sources of Apple’s stock news headlines
Headline	The contents of the Apple’s stock news headlines generated
Symbol	Symbol of the Company. It is used to be “NASDAQ: AAPL” for all the column data.
Company	Company that related to the Stock news headlines, which is Apple Incorporation.

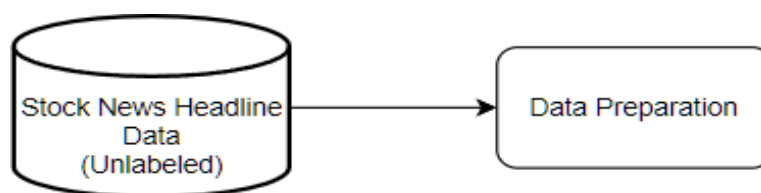
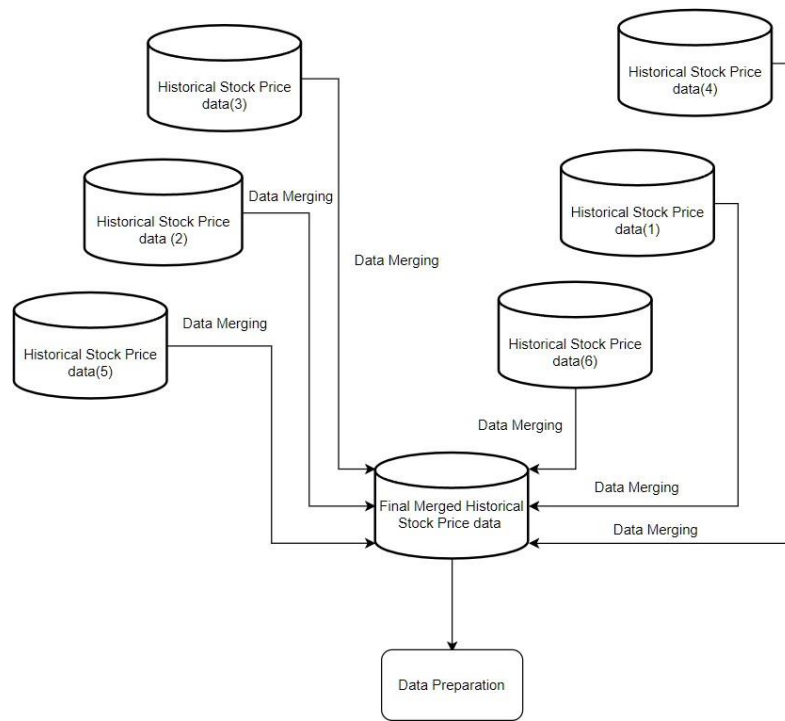


Figure 4.1 Unlabelled Stock News Headline Data and its data preparation process

**Table 4.4** Data Description on provided by Dataset ‘Download Data STOCK\_US\_XNAS\_AAPL (1)-(6)’.Source from [15]

Data Field	Description
Date	Date of the Apple’s stock price movements recorded which is the dates are between December 2018 and February 2024.
Open	The opening price of Apple's stock on a given trading day. This field reflects the price at which the stock first traded when the market opened for that specific day.
High	The highest price Apple's stock reached during the trading day. It shows the peak value of the stock price for that day, providing insight into intraday volatility.
Low	The lowest price Apple's stock fell to during the trading day. This field captures the minimum value of the stock price for that particular day, indicating potential bearish pressure during the session.
Close	The closing price of Apple's stock at the end of the trading day. This is the primary target variable for prediction and represents the final price at which the stock traded before the market closed.
Volume	The total number of shares of Apple stock traded throughout the trading day. This field serves as an indicator of market activity, showing how much interest there was in trading Apple stock on that particular day.





**Figure 4.2** Historical Stock Price Data and its data preparation process

4.2.1 System Design Diagram

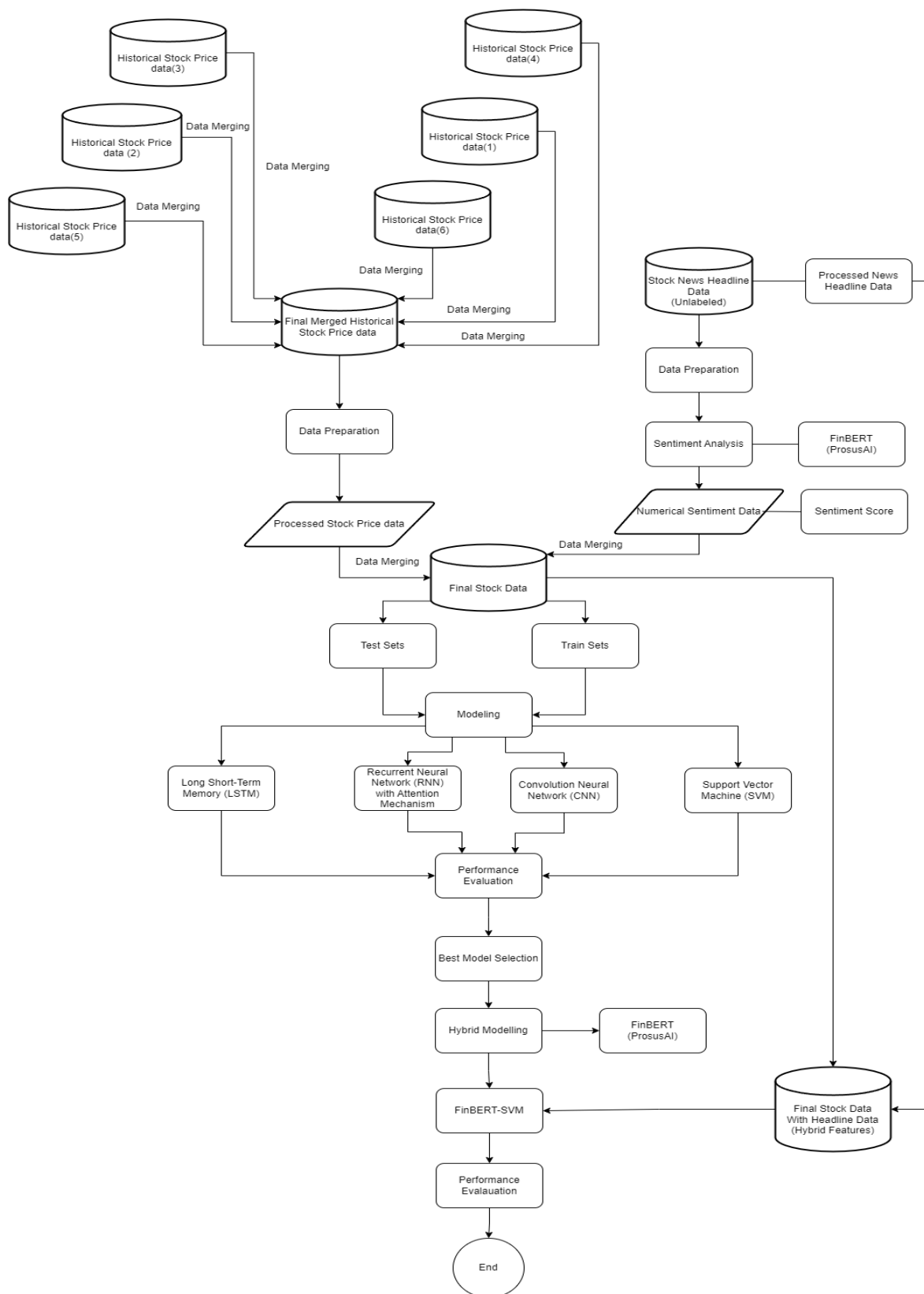
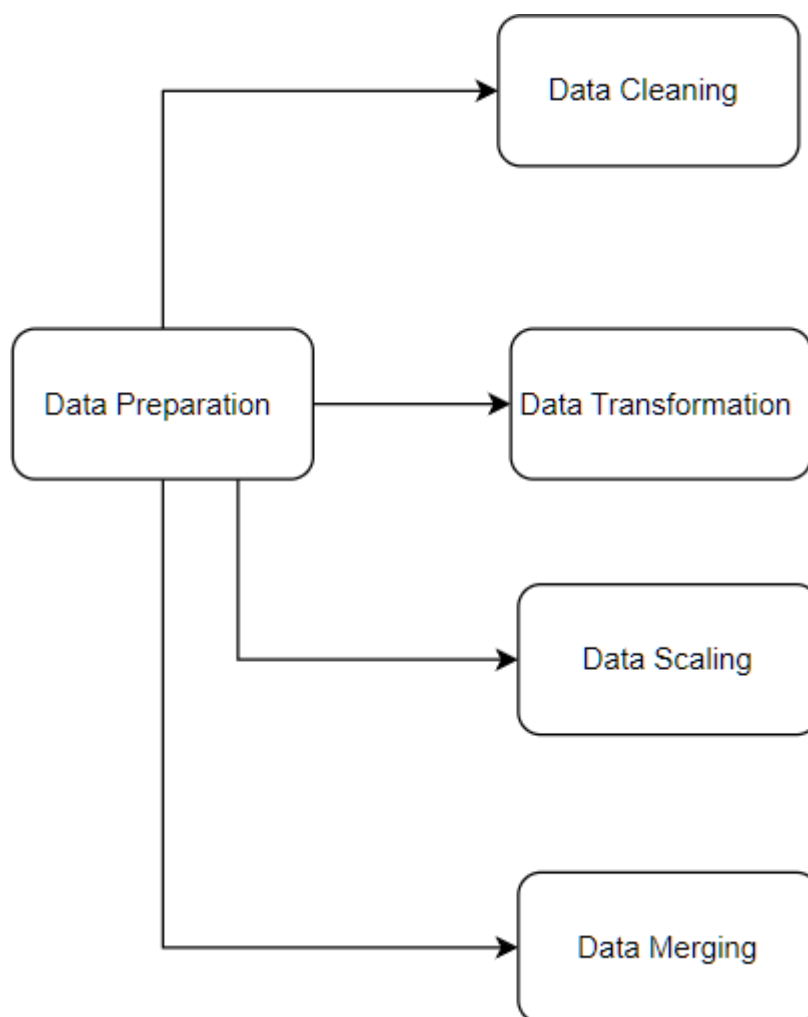


Figure 4.3 Overall Project Framework of Stock Market Prediction

**Data Preprocessing****Figure 4.4** The type of data preparation**Data Cleaning:**

For data preprocessing, it is a very important step for preparing both the textual news data and time-series stock data. For the textual data, it begins with the data cleaning process to remove noise and inconsistencies in the “news.csv” dataset. This includes removing punctuation marks to ensure consistency and readability, and dropping duplicate rows to eliminate redundancy and maintain data integrity. For the time-series stock data, missing values are a common issue. Techniques like forward fill and backward fill are applied to handle missing values while

preserving the chronological order of the data. By starting with these cleaning steps, we ensure that clean, uniform data makes later sentiment analysis and stock price modeling more accurate and effective.

#### *Data Transformation:*

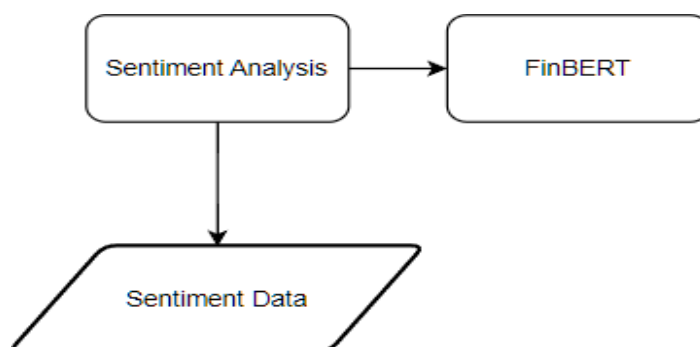
For data transformation, this step involves converting both textual and time-series data into structured formats for analysis and modeling. For the textual data, the “Time” column is changed from object to a time-based format for consistency, and the “Date” column is converted to a date type to facilitate time-series analysis. Additionally, contractions in the "Headline" column are expanded, and the column is normalized by converting text to lowercase, removing extra whitespace, and eliminating stopwords. The textual data is then lemmatized, tokenized into individual words, and vectorized to transform it into numerical representations. For the time-series data, transformations include extracting features like year, month, day, and hour from the "Date" column. Resampling is done to maintain a consistent frequency (e.g., daily or weekly). Lag features are created by shifting stock price values to previous time steps (e.g., 2-day lag), and rolling window features such as Cross Rolling Mean are added to capture trends and volatility. These steps ensure that both textual and time-series data are transformed into usable formats for further analysis and modeling.

#### *Data Scaling:*

Data scaling is an essential step to standardize the range of numerical features for both the textual and time-series data, improving consistency and model performance. For the textual data, the vectorized representations (in the "Vec" column) are scaled using the Min-Max Scaler, which normalizes the feature values between 0 and 1. This step preserves the relative importance of each vector while preventing large magnitudes from dominating the model. For the time-series data, stock prices and volumes are also scaled using techniques like Min-Max Scaling to ensure that all numerical values are on a comparable scale. This helps reduce the impact of outliers and improves the robustness of machine learning models, leading to more accurate predictions when relying on both stock price data and sentiment vectors.

*Data Merging:*

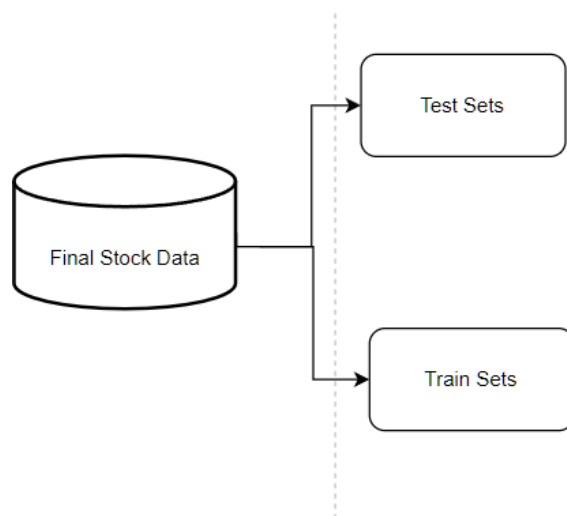
The final data preparation step involves merging both the processed textual and time-series data into one comprehensive dataset. For the textual data, after performing sentiment analysis and generating vectorized representations of the headlines (e.g., from FinBERT), this sentiment data is merged with the time-series stock data. For the time-series data, after creating lag features, rolling statistics, and scaling the numerical values, the cleaned and transformed stock price data is combined with the sentiment features. This integration ensures that both qualitative (sentiment) and quantitative (stock price) data are used together, enriching the final dataset for modeling tasks like stock price prediction or sentiment-based trading decisions.

***Sentiment Analysis Using FinBERT:***

**Figure 4.5** Implementation of FinBERT in sentiment Analysis and output the Sentiment Data

By using the FinBERT from Hugging face in term of NLP algorithm, it is now running the sentiment analysis using processed news headlines textual data. The purpose of running this algorithm may identify the tone or emotions expressed in textual data, like news headlines, which might offer insightful information for predicting the stock market. We can extract complex sentiment information from financial writings, such as sentiment polarity or scoring (positive, negative, or neutral), intensity, and class label by utilizing the FinBERT model. We are able to evaluate investor and market sentiment toward individual equities through this study, which can help guide trading tactics and financial choices.

Moreover, the implementation of Hugging Face's FinBERT enables comprehensive sentiment analysis customized for the financial industry. FinBERT can precisely analyze financial texts and capture market sentiment thanks to its pre-trained knowledge of financial language and sentiment nuances. We can improve our stock prediction model's predictive accuracy and resilience by including FinBERT into our workflow. Sentiment research provides us with insights that help us better understand market dynamics, spot new trends, and predict future market movements. This helps investors make more educated decisions in the financial markets by providing them with actionable information. Then, A new dataframe will added which contain "Headline", "Positive", "Negative", "Neutral" with their own scoring respectively. Furthermore, based on the scoring, it will identify the sentiment and also identify the class label, which 1 represents for stock increase and 0 represents for stock decrease. Also, insert Vec column from previous dataframe into this new dataframe and output it as new csv file.

**Data Splitting**

**Figure 4.6** Splitting Final Stock Data into Test sets and Train sets

In the data splitting phase, the final processed stock data, which contains both the time-series stock features and the sentiment scores data derived from textual analysis, is divided into two key datasets: train sets and test sets.

For the train sets, a large portion of the data is used to train the machine learning models. This ensures that the models can learn from historical stock prices, sentiment analysis features, and any additional engineered features like lagged data or rolling averages. The models are trained to detect patterns in the stock price movement and how it correlates with sentiment.

For the test sets, a separate portion of the data, which the models have not seen during training, is used to evaluate their performance. This helps ensure that the model's predictions are generalizable and perform well on unseen data. Splitting the data in this manner helps in measuring how well the model can forecast future stock prices based on both the historical stock data and sentiment signals from news headlines.

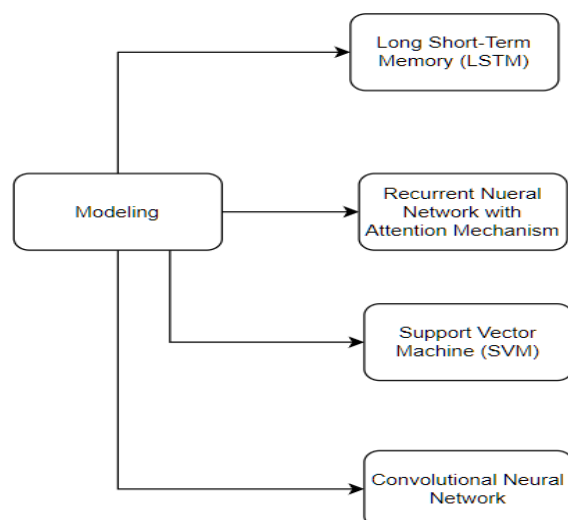
**Modelling**

Figure 4.7 Modelling Phase for LSTM ,RNN with attention Mechanism,SVM and CNN

The modeling phase involves applying various machine learning algorithms to the comprehensive and cleaned stock market dataset, which contained sentiment score and the cleaned, transformed time-series stock data after the data splitting phase. For sequential prediction tasks, Long Short-Term Memory (LSTM) is employed to capture long-term dependencies in the stock price data, making it ideal for time-series forecasting. Additionally, the Recurrent Neural Network (RNN) with Attention Mechanism enhances the model's ability to focus on the most relevant time periods or critical news events, allowing it to prioritize important data points in both stock prices and sentiment scores.

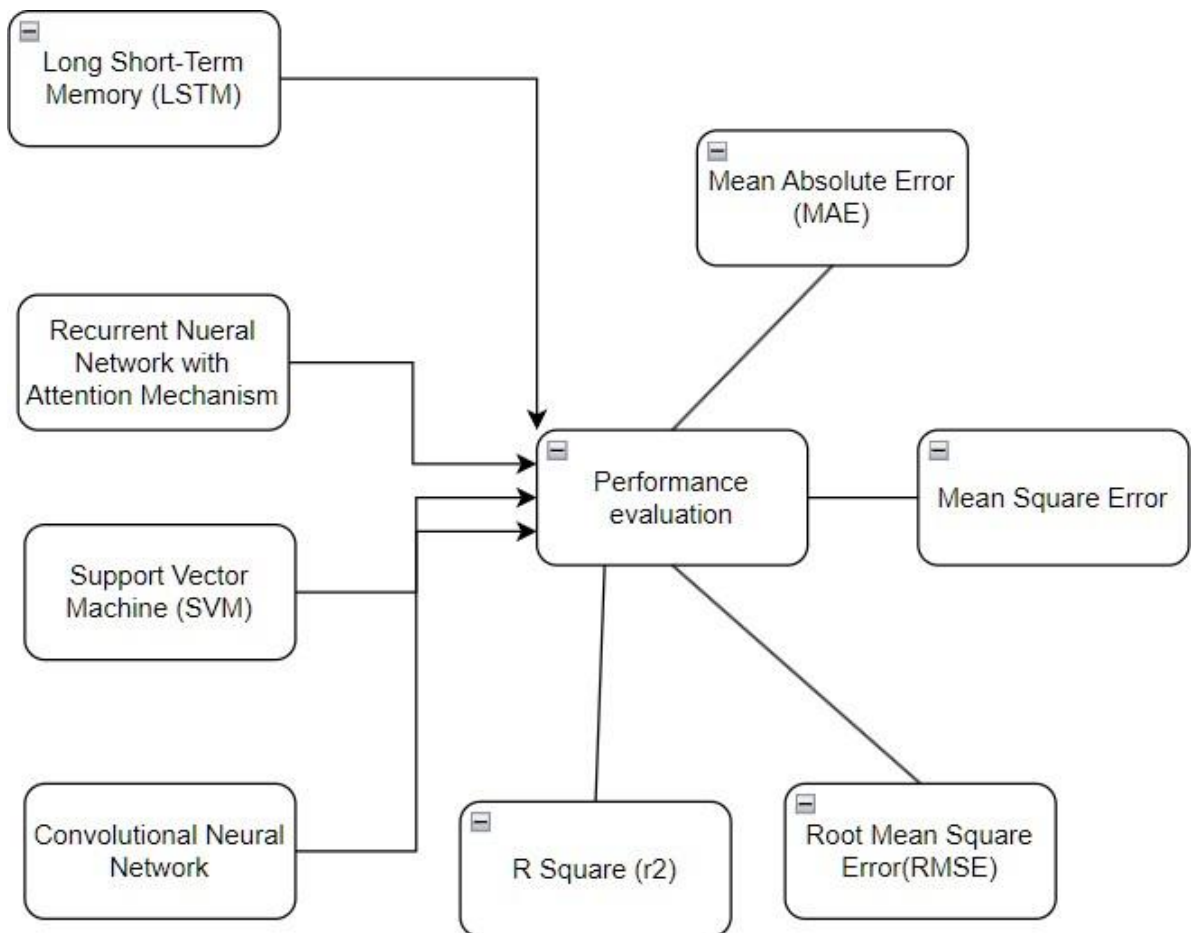
For regression tasks, Support Vector Machine (SVM) is used to predict continuous outcomes, such as stock prices. In this context, the SVM regression model fits a hyperplane that best represents the relationship between the input features—historical stock data and sentiment score data—and the target variable, which is the stock's close price. This ensures that the model can accurately forecast future stock prices based on both past trends and sentiment features.

Finally, Convolutional Neural Network (CNN) is applied to the time-series data to capture local patterns and short-term trends, providing additional insights into stock price movements.



By implementing these diverse models, the system evaluates multiple approaches to identify the best fit for predicting stock prices, ensuring both numerical and textual data are leveraged effectively for accurate forecasting.

**Model Performance Evaluation**



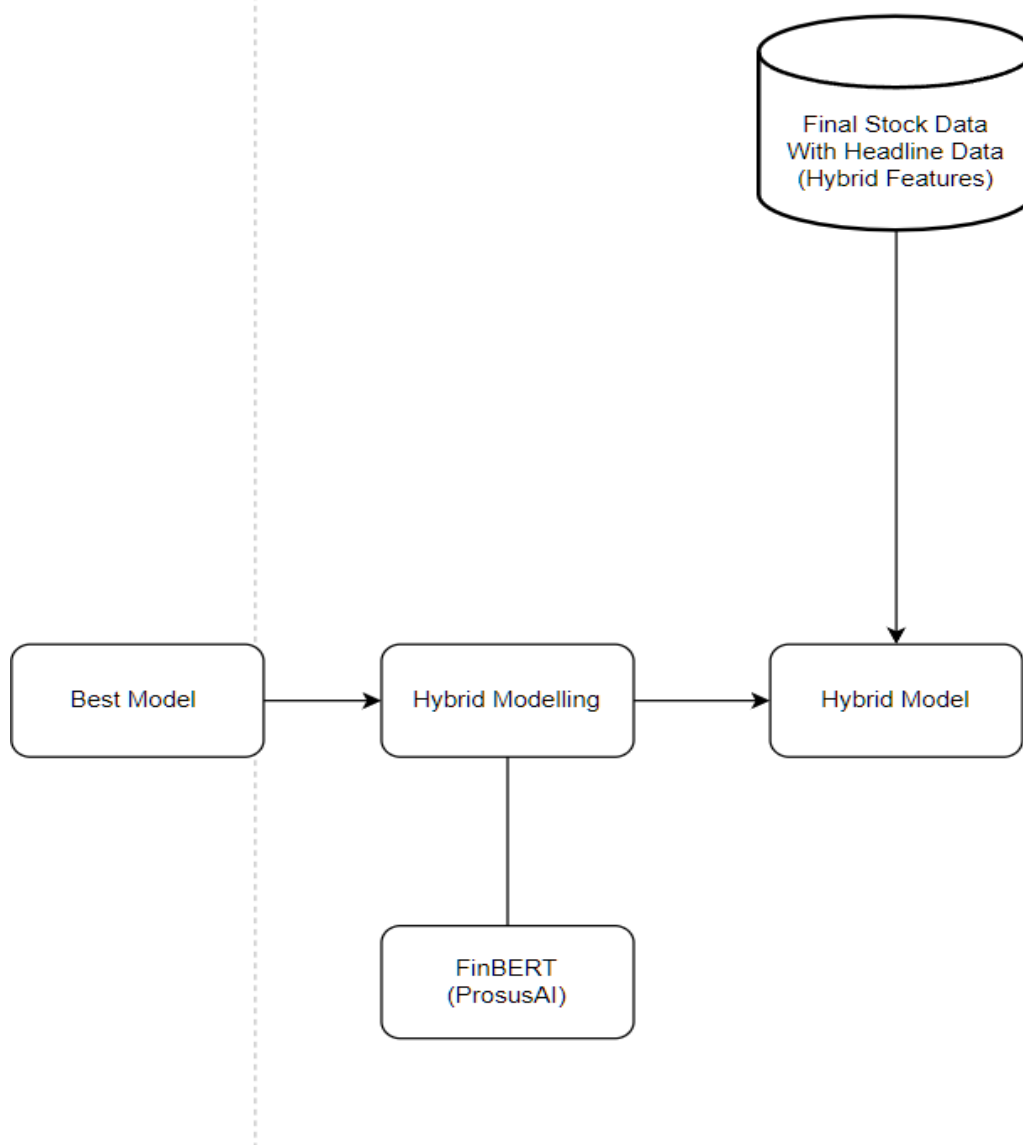
**Figure 4.8** Modelling Performance Evaluation Phase for LSTM ,RNN with attention Mechanism, SVM and CNN

In the performance evaluation phase, the predictive capabilities of various models—Long Short-Term Memory (LSTM), Recurrent Neural Network (RNN) with Attention Mechanism, Support Vector Machine (SVM), and Convolutional Neural Network (CNN)—are assessed using key regression metrics. These metrics provide insight into how well each model predicts stock prices based on the merged dataset of time-series stock data and sentiment features.

One of the key metrics used for evaluation is Mean Absolute Error (MAE), which measures the average magnitude of the errors in the model's predictions. MAE gives a clear indication of how much, on average, the predicted stock prices deviate from the actual prices. Another important metric is Mean Square Error (MSE), which squares the prediction errors before averaging them. This ensures that larger errors are given more weight, helping to identify significant discrepancies in the predictions.

Additionally, Root Mean Square Error (RMSE), the square root of MSE, is used to provide a more interpretable error metric in the same units as the target variable, 'Close' which in this case is stock prices. RMSE is particularly useful for understanding the magnitude of the prediction errors and comparing models based on their overall performance.

To evaluate how well the models explain the variance in stock prices, R-squared ( $R^2$ ) is applied.  $R^2$  measures the proportion of the variance in the target variable that is captured by the model's predictions, with values closer to 1 indicating a better fit. By combining these metrics, the performance evaluation phase ensures that the model with the overall smallest error values (MAE, MSE, RMSE) and the highest  $R^2$  score is chosen, providing the most reliable predictions based on both stock and sentiment data. This comprehensive assessment ensures that the best selected model is well-suited for forecasting stock price trends and making accurate predictions and integrate with FinBERT for further hybrid modelling phase.

*Hybrid Modelling*

**Figure 4.9** Hybrid Modelling Phase that Best model selected to integrate with ProsusAI’s FinBERT

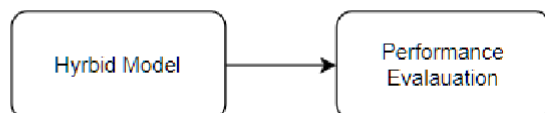
The Hybrid Modeling phase represents a crucial step in combining multiple types of data—numerical stock data and textual sentiment data from news headlines—into a unified model. This process aims to improve predictive accuracy by leveraging both structured (numerical) and unstructured (textual) data.

The Best Model chosen from the performance evaluation phase (such as LSTM, RNN with Attention, SVM, or CNN) is integrated with the FinBERT model, a pre-trained language model specifically fine-tuned for financial sentiment analysis. FinBERT processes the headlines to

extract sentiment scores, which are highly relevant in the financial domain, where market sentiment can strongly influence stock prices. These sentiment scores are treated as additional features to enrich the final dataset.

The Final Stock Data with Headline Data (Hybrid Features) is created by combining traditional stock market data, such as historical stock prices, volumes, and technical indicators, with sentiment features generated from FinBERT. This hybrid feature set allows the model to incorporate both the numerical patterns observed in stock prices and the sentiment-driven fluctuations influenced by news headlines.

In this phase, the hybrid model is constructed by training on both stock data and sentiment data, creating a model that can predict stock prices or other financial outcomes more accurately than using either data type alone. By merging the strengths of the Best Model (which captures the numerical trends in stock price data) with FinBERT's textual sentiment analysis, the hybrid model can provide more informed predictions. This combination enables the model to account for both the historical data patterns and real-time market sentiment shifts.

***Hybrid Model Performance Evaluation***

---

**Figure 4.10** Hybrid Model Performance Evaluation Phase

The final phase of the Stock Price Prediction process is the Performance Evaluation of the Hybrid Model. After integrating both historical stock data and sentiment analysis derived from FinBERT, it becomes crucial to assess the model's performance on unseen data. This evaluation is conducted by comparing the model's predictions with the actual stock prices in the test set, using key metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared ( $R^2$ ).

MAE provides an average measure of how much the predicted values deviate from the actual stock prices, offering an easy-to-understand metric of the model's overall error. On the other hand, MSE and RMSE emphasize larger errors, which helps to detect any significant deviations that the model might be making. R-squared ( $R^2$ ) evaluates the model's explanatory power, showing how much of the variance in stock prices is captured by the hybrid model, with values closer to 1 indicating a strong fit.

This comprehensive performance evaluation is critical in determining whether the hybrid model, which leverages both numerical stock data and market sentiment, outperforms models that rely on either data type alone. By analyzing the results across multiple metrics, this phase ensures that the hybrid model is not only accurate but also robust and reliable for real-world stock price forecasting.

### 4.3 Timeline

In Figure 3.8, The Gantt chart for Final Year Project 1 shows all of the completed tasks along with their related timelines in Figure 3.3 Tasks include, but are not restricted to, planning and initiating projects, understanding and visualizing data, preprocessing data, NLP modeling, reports writing, and preparing FYP presentations.

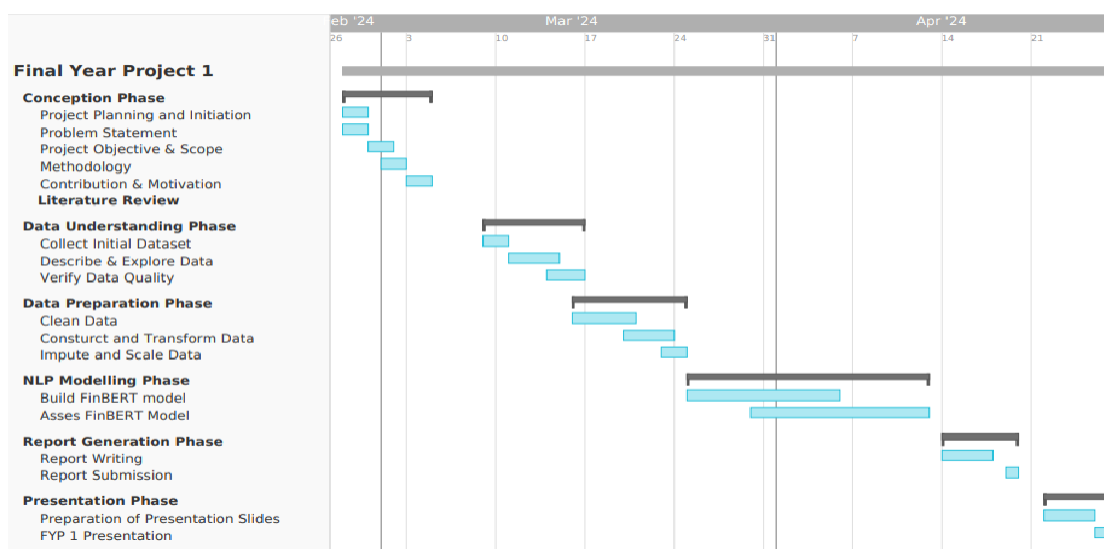
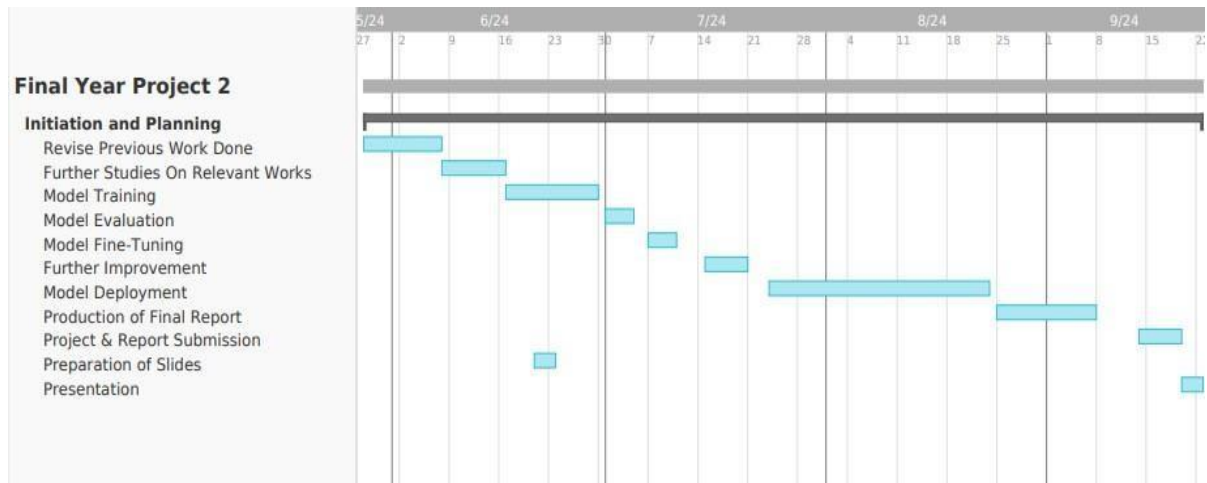


Figure 4.11 Gantt Chart for FYP 1

In Figure 3.9, Gantt chart for Final Year Project 2 is roughly drafted and outlined, with works to be done with corresponding timeline during next trimester. Works including, but not restricted to, revise previous work done, further studies on relevant works, model training and model evaluation, model tuning, report writing and FYP presentation preparation.

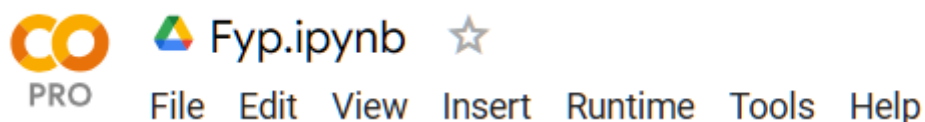


**Figure 4.12** Gantt Chart for FYP 2

## Chapter 5: System Simulation

### 5.1 Data Preparation

In this phase, we have been chosen to use Google Colab instead of Jupyter Notebook. This is because with the Web-based IDE environment, we don't required to install any software and to set up a local environment like JupyterLab environment. So that, everything in the project can be ran and saved on the cloud server and does not face any configuration hassle when need to start do coding project. Plus, We have upgrade to the Colab Pro because We need to access more to computational resources, which is Tensor Processing Units( TPU) for more Virtual Machine Memory RAM to speed up our training on the NLP and Machine learning alorighms for the whole FYP. The python version that I used in Google Colab is 3.10.12 to do our coding project.



**Figure 5.1** The Google Colab Platform with Colab Pro Subscription.

```
✓ [1] !python --version
0s
Python 3.10.12
```

**Figure 5.2** The Version of Python that Used for Project.



5.1.1 Data Collection

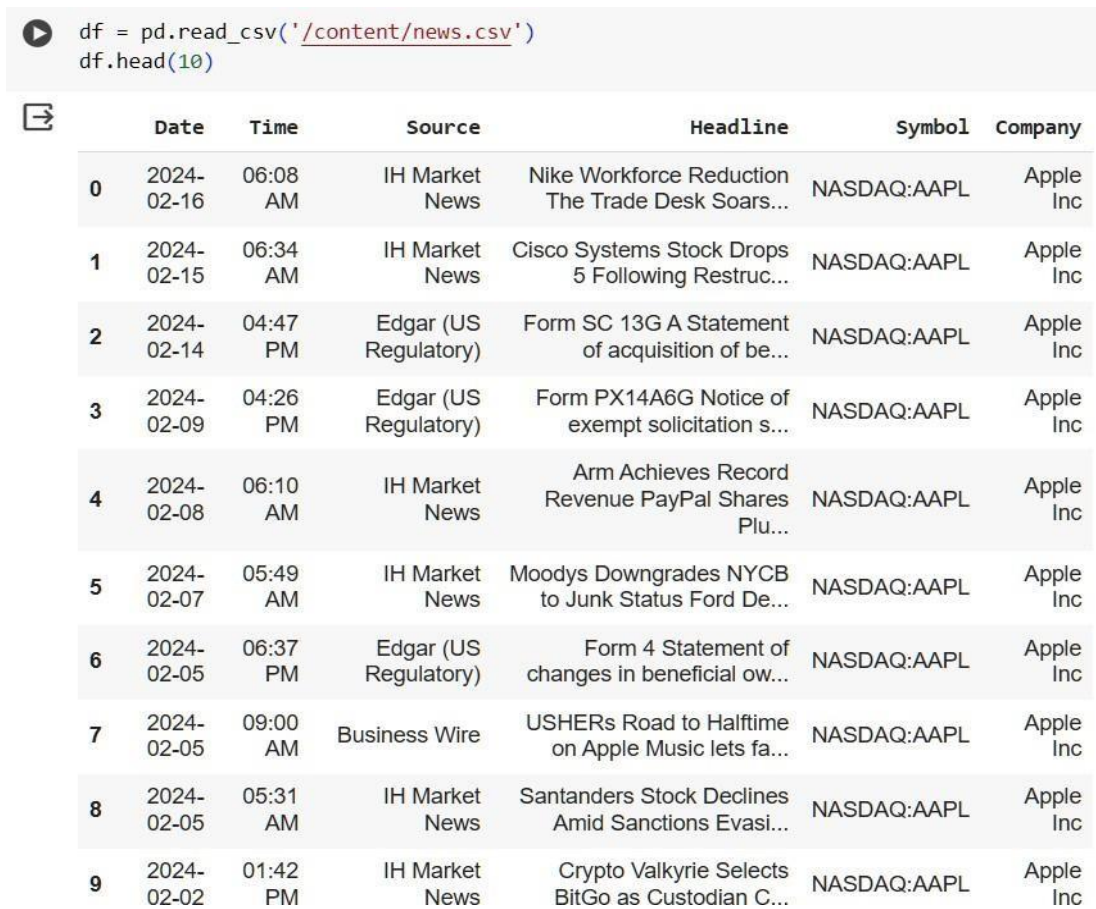


Figure 5.3 Read “news” Csv file and its first 10 rows of data

We used this stock news headlines data that is unlabelled in terms of sentiment polarity and scoring, which is found in Github.[15]. This stocks news headlines data is related to the Apple Incorporation. It consisted of six columns, which are “Date”, “Time”, “Source”, “Headline”, “Symbol” and “Company” in term of attributes. The data for all the rows in this “Symbol” column is NASDAQ: AAPL and also the data for all the rows in “Company” columns is Apple Inc.



	Date	Time	Source	Headline	Symbol	Company
3685	1/2/2019	5:03PM	Dow Jones News	Apple Revises Guidance	NASDAQ:AAPL	Apple Inc
3686	1/2/2019	4:32PM	Edgar (US Regulatory)	Current Report Filing (8-k)	NASDAQ:AAPL	Apple Inc
3687	1/2/2019	4:32PM	Business Wire	Letter from Tim Cook to Apple Investors	NASDAQ:AAPL	Apple Inc
3688	12/28/2018	3:02AM	Dow Jones News	Buybacks Come Back to Bite Firms -- WSJ	NASDAQ:AAPL	Apple Inc
3689	12/27/2018	2:09PM	Dow Jones News	Foxconn to Produce Some iPhones in India -- Re...	NASDAQ:AAPL	Apple Inc

Figure 5.4 Read the last 5 rows of data

This data consists of 3689 rows of news headlines data, which was using `df.tail()` to retrieve the information of rows numbers as shown in Figure 5.4 above. We chose this data because it was more suitable for our project that has the simple yet enough information for us to do the analysis on this set of dataset. This dataset previously also used by the github users, which called tyaan to do similar project title with us, which was “Stock Prediction with sentiment analysis”. This dataset although is not quite widely used by public, but this dataset collected until February 2024 as shown in figure 5.3, which was very latest and so far quite less people use this dataset to do research or develop their own project.

### 5.1.2 Data Cleaning

```
[ ] # remove all the punctuations include this " ' " in headline column
import string
def remove_punctuation(text):
    for punctuation in string.punctuation:
        text = text.replace(punctuation, ' ')
    return text
df['Headline'] = df['Headline'].apply(remove_punctuation)
df.head()
```

Figure 5.5 Remove Punctuations Function

For the first data cleaning step in figure 5.5 is to import the string module, which consist of set of punctuation marks and remove punctuations , which also include “ ‘ ”, which is single quote character for all the rows in “Headline” column in the dataset “news” , which later defined as “df”. This is to ensure the data consistency in term of textual data. For example, “ Zed play ball, and game” will become “Zed play ball and game”.

```

▶ # remove all of the unknown symbol in the headline columns, include this " ' " symb
import re

def remove_unknown_symbols(text):
    # Remove the ' ' symbol
    text = text.replace(' ', '')
    # Remove any other unknown symbols
    text = re.sub(r'^\x00-\x7F', '', text)
    return text

df['Headline'] = df['Headline'].apply(remove_unknown_symbols)
df.head()

```

Figure 5.6 Remove Unknown Symbol Function

Secondly, figure 5.6 shown that it imported re module, which consist of set of unknown symbol and remove all of them, which also include “ ‘ ” for all the rows in “Headline” column in df. The purpose of adding the “ ‘ ” symbol again is because to ensure that the symbol can removed completed to avoid any leftover form the data. Removal of those unknown symbol was a crucial step for data cleaning to ensure the good data quality. For example, “Zed’ phone rings @#” will become “Zed phone rings”.

```

[17] #drop duplicates rows in headline column
df.drop_duplicates(subset='Headline', inplace=True)

```

Figure 5.7 Drop Duplicates Rows Function

Third, figure 5.7 shown that dropped all duplicates rows in the headline column permanently, which was aimed to improve the accuracy of the analysis.

df.tail()

	Date	Time	Source	Headline	Symbol	Company
2442	1/2/2019	5:06PM	Dow Jones News	Apple Sees 1Q Revenue Below Views Update	NASDAQ:AAPL	Apple Inc
2443	1/2/2019	5:03PM	Dow Jones News	Apple Revises Guidance	NASDAQ:AAPL	Apple Inc
2444	1/2/2019	4:32PM	Business Wire	Letter from Tim Cook to Apple Investors	NASDAQ:AAPL	Apple Inc
2445	12/28/2018	3:02AM	Dow Jones News	Buybacks Come Back to Bite Firms WSJ	NASDAQ:AAPL	Apple Inc
2446	12/27/2018	2:09PM	Dow Jones News	Foxconn to Produce Some iPhones in India Re...	NASDAQ:AAPL	Apple Inc

Figure 5.8 Output on the remaining rows after dropped duplicates

After that, the remaining dataset had become 2446 rows after performed the dropped duplicates function, that's mean had about 1243 duplicated rows has been removed.

## 5.1.3 Data Transformation

```

# change the data column "Time" data in data type, from object to time and change the
df['Time'] = pd.to_datetime(df['Time'])
df['Time'] = df['Time'].dt.strftime('%I:%M %p')
df.head()

```

<ipython-input-21-8e2ae9583f46>:3: UserWarning: Could not infer format, so each element  
df['Time'] = pd.to\_datetime(df['Time'])

	Date	Time	Source	Headline	Symbol	Company
0	2024-02-16	06:08 AM	IH Market News	Nike Workforce Reduction The Trade Desk Soars...	NASDAQ:AAPL	Apple Inc
1	2024-02-15	06:34 AM	IH Market News	Cisco Systems Stock Drops 5 Following Restruc...	NASDAQ:AAPL	Apple Inc
2	2024-02-14	04:47 PM	Edgar (US Regulatory)	Form SC 13G A Statement of acquisition of be...	NASDAQ:AAPL	Apple Inc
3	2024-02-09	04:26 PM	Edgar (US Regulatory)	Form PX14A6G Notice of exempt solicitation s...	NASDAQ:AAPL	Apple Inc
4	2024-02-08	06:10 AM	IH Market News	Arm Achieves Record Revenue PayPal Shares Plu...	NASDAQ:AAPL	Apple Inc

Figure 5.9 “Time” data type changed from object to datetime

First of all, by changing the data type to datetime, the datetime module need to be imported as dt and then changed the “Time” column data from object into time with format of 12-hours clock. For example,2248 become 10:48 PM, which was aimed for data standardization in data transformation.

```
[ ] # change the data column "Date" data in data type, from object to date and change the

# Convert the 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Make the changes permanent
df.to_csv('/content/news.csv', index=False)

# Print the updated DataFrame
df.head()
```

	Date	Time	Source	Headline	Symbol	Company
0	2024-02-16	06:08 AM	IH Market News	Nike Workforce Reduction The Trade Desk Soars...	NASDAQ:AAPL	Apple Inc
1	2024-02-15	06:34 AM	IH Market News	Cisco Systems Stock Drops 5 Following Restruc...	NASDAQ:AAPL	Apple Inc
2	2024-02-14	04:47 PM	Edgar (US Regulatory)	Form SC 13G A Statement of acquisition of be...	NASDAQ:AAPL	Apple Inc
3	2024-02-09	04:26 PM	Edgar (US Regulatory)	Form PX14A6G Notice of exempt solicitation s...	NASDAQ:AAPL	Apple Inc
4	2024-02-08	06:10 AM	IH Market News	Arm Achieves Record Revenue PayPal Shares Plu...	NASDAQ:AAPL	Apple Inc

Figure 5.10 “Date” data type changed from object to datetime

As shown in figure 5.10, the step to change the datatype from object to datetime in “Date” column with format “Year-Month-Day”. For example, 12/2/2024 become 2024-02-12, which was ensured the good data quality of the dataset df.

```
# Handle Contractions for the headline column

def expand_contractions(text):
    return contractions.fix(text)

# Apply the function to the 'Headline' column
df['Headline'] = df['Headline'].apply(expand_contractions)
```

Figure 5.11 Expand Contractions Function

In figure 5.11, it showed that the data transformation’s step to expand the contraction in all the rows in data column of “headline”, which was to normalize the data for ensuring all the consistency in textual data .Before that, the contractions package needed to install and import to execute the function. For example, ‘don’ t’ become ‘do not’.



```

✓ [25] #change the headline column data type to String
df['Headline'] = df['Headline'].astype('str')

```

Figure 5.12 Data type of headline changed to string function

As shown in figure 5.12, all of the rows in “Headline” column was changed to string as previous data type was object in this case to allow the further text processing applied on it.

```

▶ # Do normalization on the whole headline column

# Define the normalization function
def normalize_headline(text):
    # Convert to lowercase
    text = text.lower()

    # Remove extra whitespace
    text = ' '.join(text.split())

    return text

# Apply the normalization function to the 'Headline' column
df['Headline'] = df['Headline'].apply(normalize_headline)

# Print the updated DataFrame
df['Headline'].head(10)

```

```

0  nike workforce reduction the trade desk soars ...
1  cisco systems stock drops 5 following restruct...
2  form sc 13g a statement of acquisition of bene...
3  form px14a6g notice of exempt solicitation sub...
4  arm achieves record revenue paypal shares plum...
5  moodys downgrades nycb to junk status ford dec...
6  form 4 statement of changes in beneficial owne...
7  ushers road to halftime on apple music let us ...
8  santanders stock declines amid sanctions evasi...
9  crypto valkyrie selects bitgo as custodian cha...

```

Figure 5.13 Normalization process on headline column

In figure 5.13, the normalization process, included converting to lowercase, remove extra whitespace applied to the “Headline” column data and aimed to reduce redundancy in data and standardized the data. For example, ‘ZED sleep’ become ‘zed sleep’.

```

▶ # remove stopwords for then headline column

# Import the stopwords library from NLTK
from nltk.corpus import stopwords

# Define a function to remove stopwords
def remove_stopwords(text):
    # Split the text into words
    words = text.split()

    # Get the set of stopwords
    stop_words = set(stopwords.words('english'))

    # Remove the stopwords from the list of words
    filtered_words = [word for word in words if word not in stop_words]

    # Join the filtered words back into a string
    return ' '.join(filtered_words)

# Apply the function to the 'Headline' column
df['Headline'] = df['Headline'].apply(remove_stopwords)

```

Figure 5.14 remove stopwords process on headline column

As figure 5.14, it showed that stopwords inside the Headline column data needed to be removed and module “stopwords” of NLTK needed to be downloaded before execute this action. This was because it was to reduce the noise inside the data. For example, ‘zed is sleep’ become ‘zed sleep’.

```

# convert all of numbers inside into words that contains in contents of headline columns, regardless anything,

def convert_numbers_to_words(text):
    # Split the text into words
    words = text.split()

    # Convert each word to a number if possible
    for i, word in enumerate(words):
        try:
            number = int(word)
            words[i] = num2words.num2words(number)
        except ValueError:
            pass

    # Join the words back into a string
    return ' '.join(words)

def convert_data_regex(text):
    pattern = r"(\D+)(\d+([a-zA-Z]+))" # Matches text followed by digits and optional letters
    match = re.search(pattern, text)

    if match:
        text_part, numeric_part, suffix = match.groups()
        converted_numeric = num2words.convert(int(numeric_part)) # Assuming numeric_part is an integer
        return f"{text_part} {converted_numeric} {suffix}"
    else:
        return text

# Apply the function to the 'Headline' column
df['Headline'] = df['Headline'].apply(convert_numbers_to_words)

```



Figure 5.15 conversion of all numbers into words process on headline column

In figure 5.15, it was required to install num2words package before done this conversion. After that, data transformation steps required to run, which was to enhance interpretability of the data for NLP algorithm. For example, 2 become two.

```
# do lemmatization and tokenization in headline columns

# Import the necessary libraries

from nltk.stem import WordNetLemmatizer
from nltk import word_tokenize
# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()

# Define a function to perform lemmatization
def lemmatize_text(text):
    # Tokenize the text
    words = nltk.word_tokenize(text)

    # Lemmatize each word
    lemmatized_words = [lemmatizer.lemmatize(word) for word in words]

    # Join the lemmatized words back into a string
    return ' '.join(lemmatized_words)

# Apply the lemmatization function to the 'Headline' column
df['Headline'] = df['Headline'].apply(lemmatize_text)

# After lemmatization, create a new column named 'tokens'
df['tokens'] = df['Headline'].apply(word_tokenize)

# Print the updated DataFrame
df.head()
```

Figure 5.16 lemmatization and tokenization process on headline column

**tokens**

```
[nike, workforce, reduction, trade, desk, soar...
[cisco, system, stock, drop, five, following, ...
[form, sc, 13g, statement, acquisition, benefi...
[form, px14a6g, notice, exempt, solicitation, ...
[arm, achieves, record, revenue, paypal, share...
```

Figure 5.17Output of the tokenization process

Based on figure 5.16, it is required to download the punkt and wordnet of NLTK library. Then, lemmatization and tokenization process will be done to the “Headline” column during data

transformation stage. For example, running become run. For the tokenization, the sentences in the headline column will be separated into individual tokens (words) and put into the arrays, and the results are added into the new column called "tokens" in figure 5.17 in df. This was to reduce the dimensionality of the whole dataset.

```
# remove the data that has only 3 or less than tokens  
df = df[df['tokens'].map(lambda x: len(x)) > 3]  
print(df)
```

Figure 5.18 Removal of the data that consists of 3 or less than 3 tokens in tokens column

[2412 rows x 7 columns]

Figure 5.19 remaining rows of data after removal data of 3 or <3 tokens

Based on figure 5.18, the step to remove the rows that only consist of 3 or less than 3 tokens (words) in array in 'tokens' column is to reduce the noise in terms of the data. After that, only 2411 rows, which 2412 need to minus 1 for the title row, remained and 35 rows removed which consisted of 3 or less than 3 tokens in figure 5.17.

```

▶ #print maximum and minimum tokens in tokens data

print(df['tokens'].map(lambda x: len(x)).max())
print(df['tokens'].map(lambda x: len(x)).min())

31
4

```

Figure 5.20 Maximum and minimum tokens in tokens column

```

▶ # do Truncation to tokens column, if the tokens less than ten, do padding to create uniform length

def truncate_and_pad(tokens, max_length):
    if len(tokens) > max_length:
        return tokens[:max_length]
    else:
        padding = [0] * (max_length - len(tokens))
        return tokens + padding

# Set the maximum length of the tokens
max_length = 31

# Apply the truncate_and_pad function to the 'tokens' column
df['tokens'] = df['tokens'].apply(lambda x: truncate_and_pad(x, max_length))

# Print the updated DataFrame
df.head()

```

Figure 5.21 Do truncation and padding in tokens column

As in figure 5.21, the truncation and padding process was applied to the tokens column, which aimed to create uniform length for the input sequences in tokens column. The max length of the tokens set to maximum length of 31 as the maximum tokens found in figure 5.18 was 31. Therefore, The “0” was added inside and the numbers of “0” added into the tokens data were depending on the length of the tokens. For example, if the length of the tokens is 10, the numbers of “0” added as tokens will be 21 to standardize the input length.

```

[34] #Do vectorization
from gensim.models import Word2Vec
model = Word2Vec(df['tokens'], min_count=1, vector_size=31)
df['Vec'] = df['tokens'].apply(lambda x: np.mean([model.wv[word] for word in x if word in model.wv], axis=0))
df.head()

```

Figure 5.22 Do vectorization to tokens column

	Vec
.	[0.21134265, 0.8664383, -0.41621563, -0.427607...
.	[0.1739054, 0.9676703, -0.42511603, -0.4611979...
.	[0.18414985, 1.1166161, -0.47038597, -0.522360...
.	[0.18713246, 1.1598363, -0.4884018, -0.5433145...
.	[0.19271828, 1.1134086, -0.4744971, -0.5198172...

Figure 5.23 Vectorization results in Vec column

Based on figure 5.22, the tokens column had been applied into the vectorization process that need to install and import genism package for Word2Vec. Then, it changed the tokens(words) into the numeric representation. So than the results outputted into a new column call Vec column, which was compatible to the machine learning models for training as shown in figure 5.23.

5.1.4 Data Scaling

```

# do minmaxscaler
from sklearn.preprocessing import MinMaxScaler

# Create a MinMaxScaler object
scaler = MinMaxScaler()

# Fit the scaler to the 'Vec' column
scaler.fit(df['Vec'].values.tolist())

# Transform the 'Vec' column using the fitted scaler
df['Vec'] = scaler.transform(df['Vec'].values.tolist())

# Print the updated DataFrame
df.head()
    
```

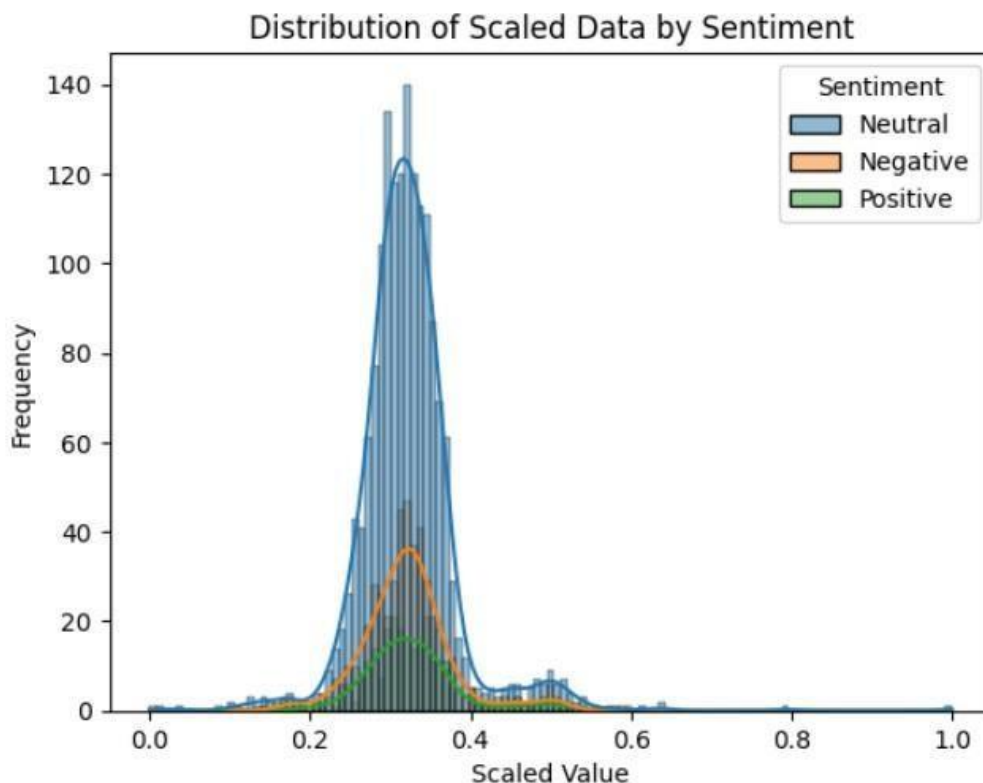
Figure 5.24 Implementation of MinMax Scaler in Vec column

Vec
0.321145
0.229111
0.254296
0.261628
0.275360

Figure 5.25 Output of scaled data in Vec column

In figure 5.24, Min-Maxscaler function was required to import from sklearn to normalize the values in Vec column within the range between 0 and 1 and outputted the scaled data as shown in figure 5.25. Let said if not performed data scaling, it can cause the poor model performance during model training and increase the sensitivity to the outliers, so that it was better result to do data scaling, which can improve convergence of the algorithm in term of quicker and stable. Besides that, the scaled data can be fitted in SVM,RNN with Attention Mechanism, CNN and LSTM in this FYP .The reason for chosen these algorithms, because firstly , SVM is effective in handling the linear and nonlinear decision boundaries in high-dimensional boundaries. Also, the RNN is able to capture the temporal dependencies , which is fit for the sequential inputs. Lastly, in term of LSTM can capture the long-term dependencies, also excel in solving the vanish gradient problem.

### 5.2 Exploratory Data Analysis



Mean scaled value by sentiment:

```
Sentiment
Negative    0.321906
Neutral     0.322437
Positive    0.326091
Name: Vec, dtype: float64
```

Standard deviation of scaled value by sentiment:

```
Sentiment
Negative    0.061545
Neutral     0.061570
Positive    0.061622
```

Figure 5.26 Histogram and KDE plot Diagram related distribution of Scaled Data By Sentiment

Based on the graph which related to the distribution of Scaled Data By Sentiment, the positive sentiment category had the highest mean scaled value of 0.326, followed by the neutral category with a mean of 0.322 and the negative category with a mean of 0.321. Also, the positive sentiment category also had the widest distribution, indicating that there is more variability in

the scaled values for this category. Lastly, The neutral and negative sentiment categories had similar distributions, with the positive category having a slightly wider distribution.

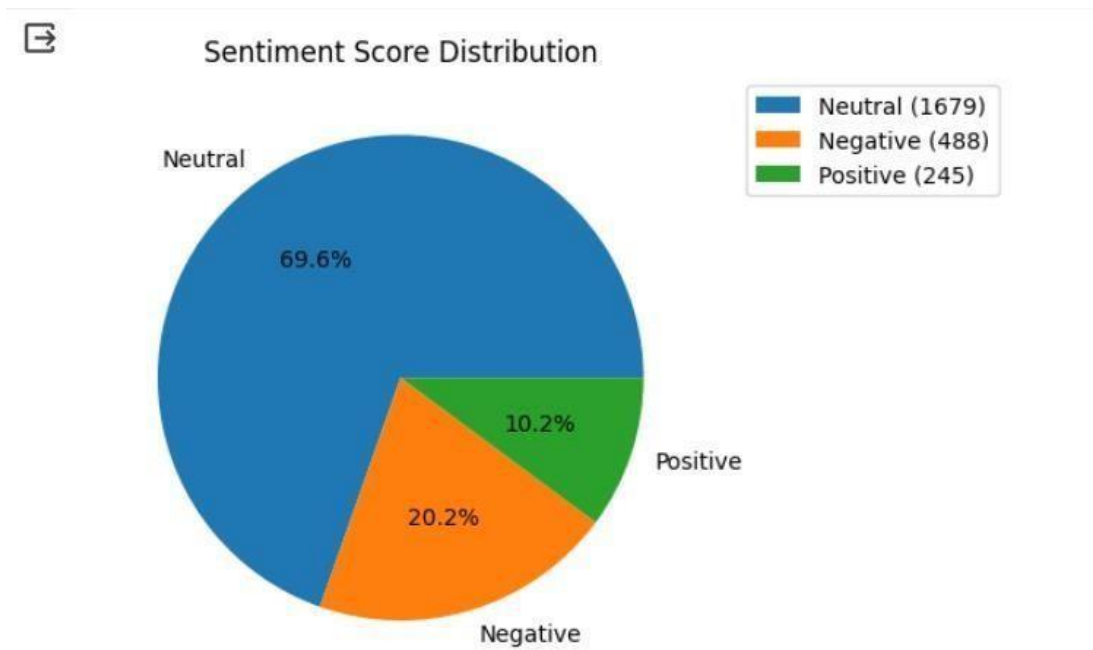
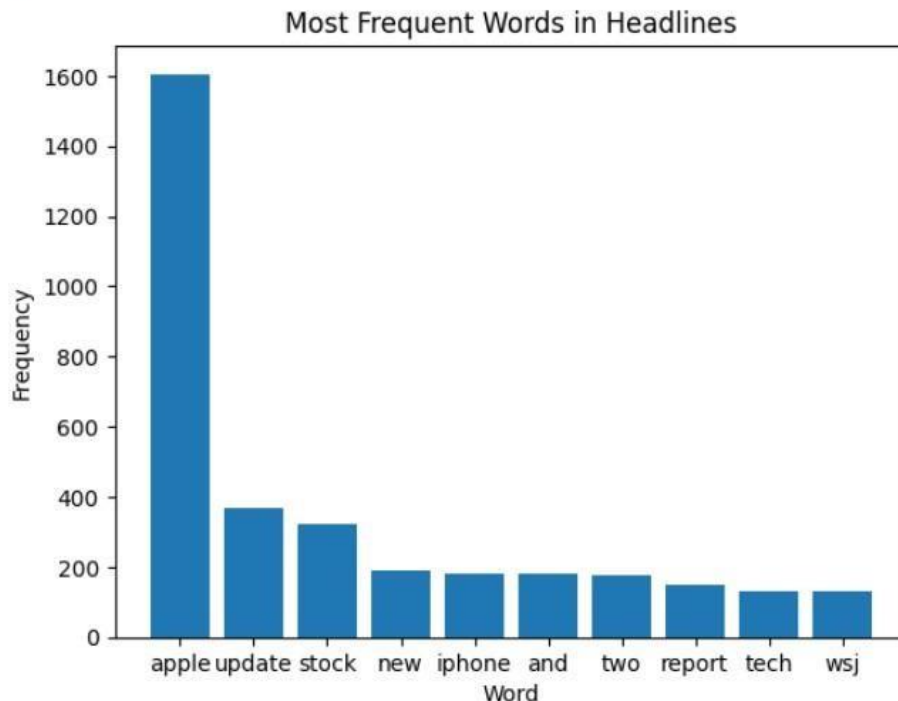


Figure 5.27 Sentiment Score Distribution’s Pie Chart

Based on Pie Chart, we can conclude that there is around 69.6% neutral in term of sentiment scoring, followed by the negative sentiment score with 20.2 and lastly is the positive sentiment score with only 10.2%. This predicted that the Apple company’s stock prices did not affect much based on the sentiment from news headline data and stay neutral, which is not much volatile in term of its stock prices movement.

```

➡ apple: 1605
  update: 366
  stock: 324
  new: 191
  iphone: 181
  and: 179
  two: 178
  report: 148
  tech: 132
  wsj: 129
    
```



The most frequent word in the headlines is 'apple'.

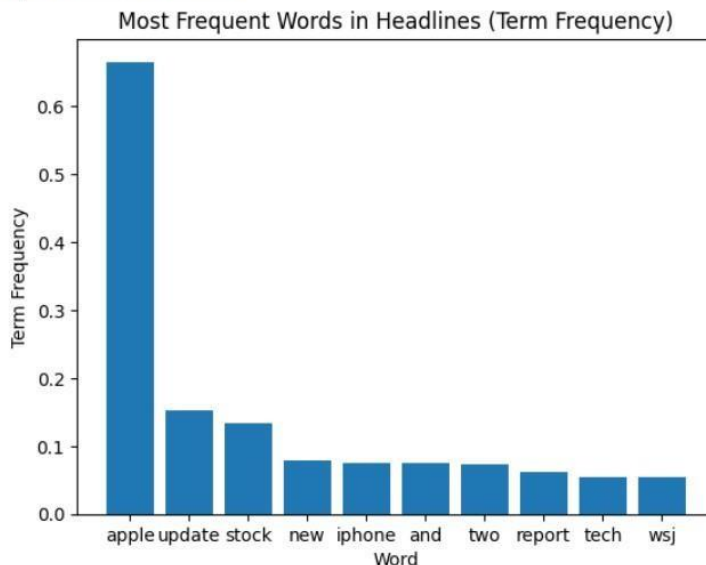
Figure 5.28 Bar Chart related to the most frequent words in headlines

Based on figure 5.28, it showed that the most frequent word in the headlines is 'apple'. This proved that 'apple' words said that can be affected the most in terms of its stock price movements but in other hand, also common since the news headlines is related to the Apple Incorporation. It was followed by the word of ' update', which might be meant about it have the products update ,software update and others. Third, 'stock' words will be the third , which is discuss about its stock by the news.



```

apple: 0.6654228855721394
update: 0.1517412935323383
stock: 0.13432835820895522
new: 0.07918739635157546
iphone: 0.07504145936981758
and: 0.074212271973466
two: 0.07379767827529021
report: 0.06135986733001658
tech: 0.05472636815920398
wsj: 0.053482587064676616
    
```



The most frequent word in the headlines is 'apple', with a term frequency of 0.6654228855721394.

Figure 5.29 Bar Chart related to the most frequent words in headlines in term frequency

Based on figure 5.29, it showed that the most frequent word in the headlines was 'apple'. This proved that 'apple' words, with 0.6654229 in term of term frequency said that can be appeared the most in terms of its stock news headlines but in other hand, also common since the news headlines is related to the Apple Incorporation. It was followed by the word of 'update'(0.1517413) and stock(0.1343283).

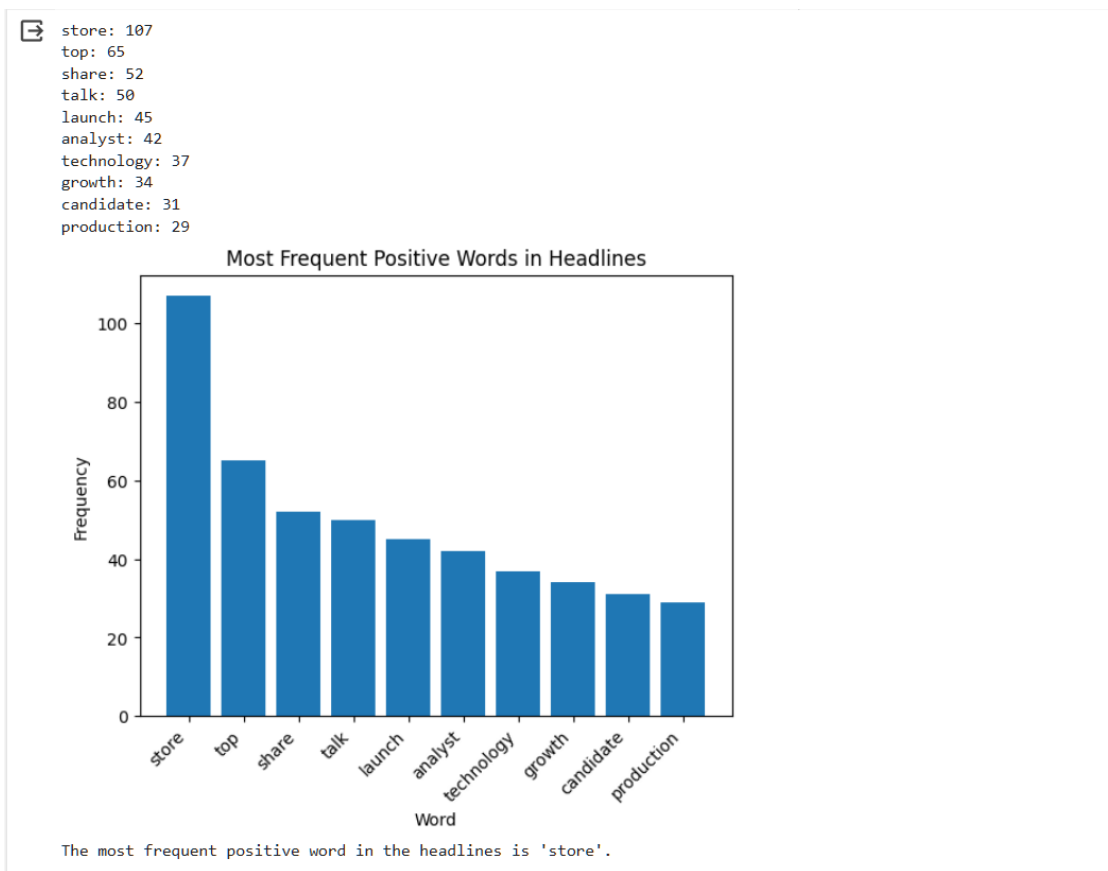
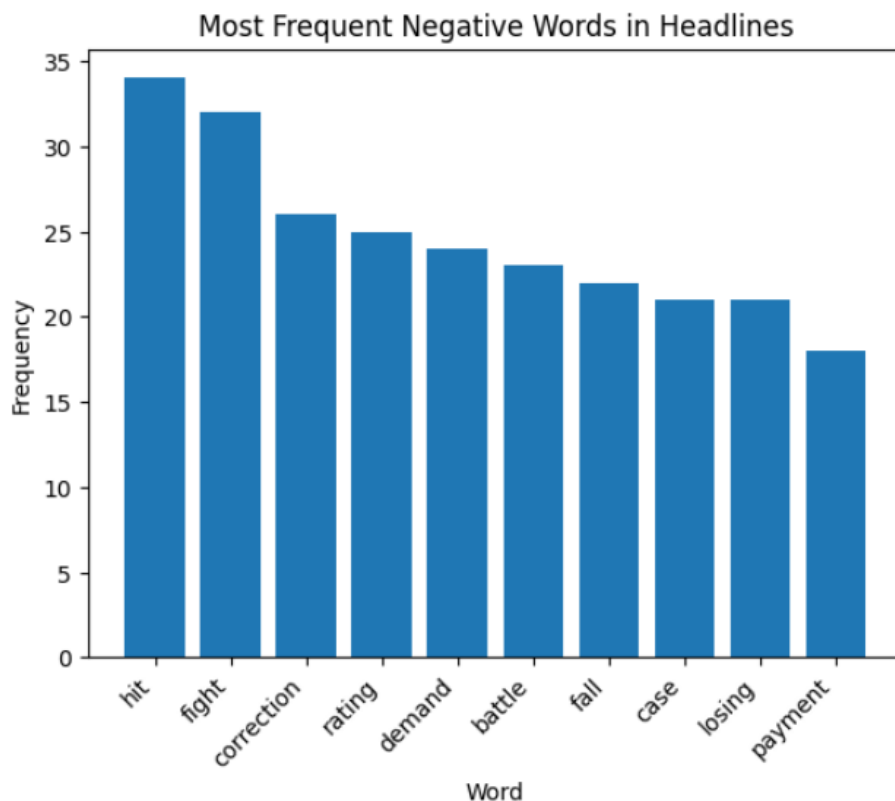


Figure 5.30 Bar Chart related to the most frequent positive words in headlines

Based on figure 5.30, the most frequent positive word in the headlines was 'store'(107). This proved that ‘store’ word will meant as Apple was open more stores in worldwide, or will store its stock for waiting the opportunity for instant growth in stock prices market.It followed by the word “top”(65), which meant it has the industry -leading technology and products at the top level based on the prediction. The ‘share’ will be the third as meant to be related to its share news that may brought positive news to its investors.

```

hit: 34
fight: 32
correction: 26
rating: 25
demand: 24
battle: 23
fall: 22
case: 21
losing: 21
payment: 18
    
```



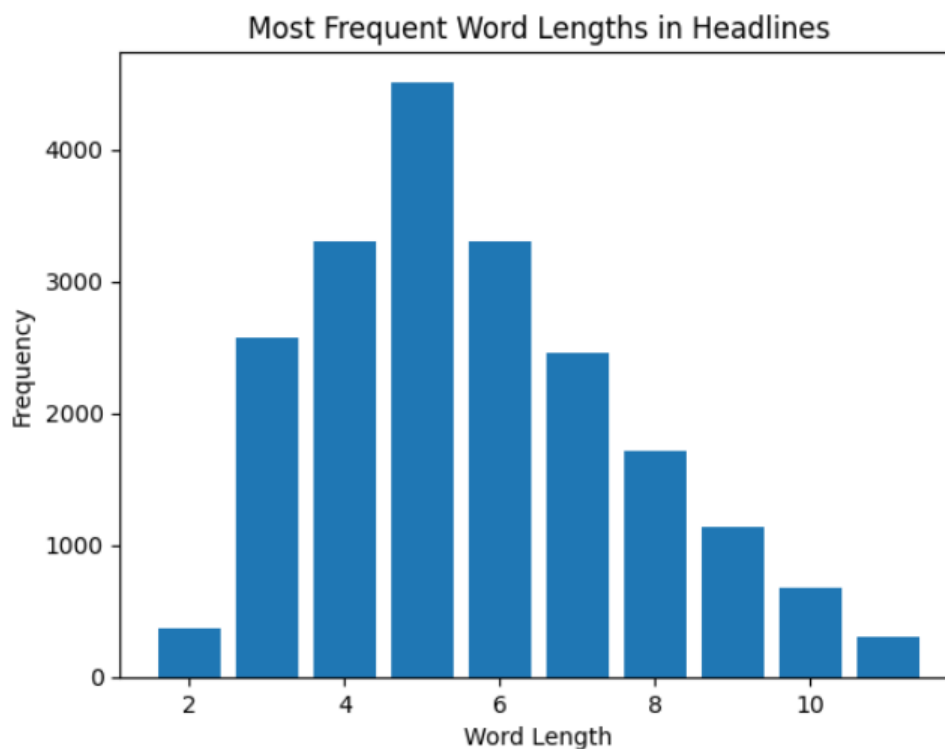
The most frequent negative word in the headlines is 'hit'.

Figure 5.31 Bar Chart related to the most frequent negative words in headlines

Based on 5.31, the most frequent negative word in the headlines was 'hit'(34).It might meant that it's hit by its competitor, which might be Samsung, especially on total sale on products since long time ago had become rival. Secondly, it followed by the word 'fight'(32), which was meant to fight by its competitors in tough condition, especially against Samsung in Smartphone Industry. Third, it would be the 'correction'(26) , which meant Apple needed to

do some correction in their mistakes, which related to their phone architecture or software bug that make users more dissatisfied .

↪ 5 Characters: 4513  
 4 Characters: 3303  
 6 Characters: 3300  
 3 Characters: 2571  
 7 Characters: 2461  
 8 Characters: 1712  
 9 Characters: 1133  
 10 Characters: 670  
 2 Characters: 372  
 11 Characters: 307

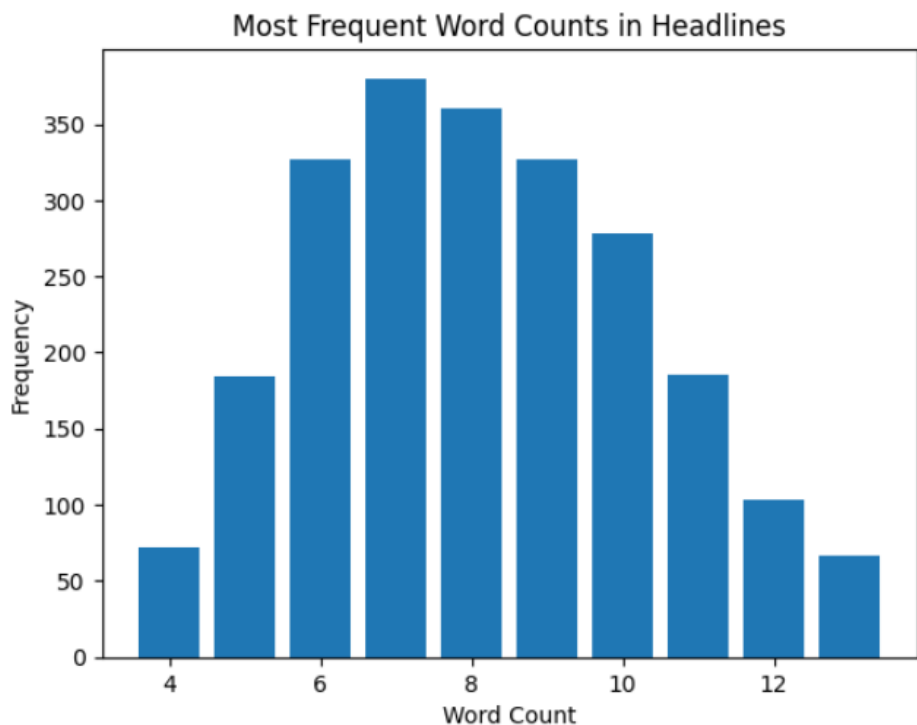


The most frequent headline length is 5 characters.

Figure 5.32 Bar Chart related to the most frequent word lengths in headlines

Based on figure 5.32, the most frequent word lengths was 5 characters (4513). This was meant might be the word “apple”, “store” and also “stock”. Beside that, it followed by the 4 word characters (3303), which might be the words ‘rise’, ‘drop’ and others. Then, the word characters with 6 will be the third with 3300, which might be ‘Supply’, ‘Streak’ and others.

↗ Word count 7: 380  
 Word count 8: 360  
 Word count 9: 327  
 Word count 6: 327  
 Word count 10: 278  
 Word count 11: 185  
 Word count 5: 184  
 Word count 12: 103  
 Word count 4: 72  
 Word count 13: 67



The most frequent word count in each sentences in Headline is 7 Words.

Figure 5.33 Bar Chart related to the most frequent word count in headlines

Based on figure 5.33, the most frequent word count in each sentence in Headline is 7 words in each sentences, which was 380. For example, ‘stocks slide as growth fears spread wsj’ and others. Besides that, it would be word count 8 with (360) and also followed by word count 9 with (329).

### 5.3 Sentiment Analysis

```
[36] from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch
```

Figure 5.34 Library and Packages downloaded and installed for FinBERT

```
[ ] df_array = np.array(df)
df_list = list(df_array[:,3])
```

Figure 5.35 Convert df into Numpy array

```
#Finbert
tokenizer = AutoTokenizer.from_pretrained("ProsusAI/finbert")
model = AutoModelForSequenceClassification.from_pretrained("ProsusAI/finbert")

inputs = tokenizer(df_list, padding = True, truncation = True, return_tensors='pt')
outputs = model(**inputs)
```

/usr/local/lib/python3.10/dist-packages/huggingface\_hub/utils/\_token.py:88: UserWarning: The secret `HF\_TOKEN` does not exist in your Colab secrets. To authenticate with the Hugging Face Hub, create a token in your settings tab (<https://huggingface.co/settings/tokens>). You will be able to reuse this secret in all of your notebooks. Please note that authentication is recommended but still optional to access public models or datasets. warnings.warn()

tokenizer_config.json:	100%		252/252	[00:00<00:00, 24.1kB/s]
config.json:	100%		758/758	[00:00<00:00, 74.2kB/s]
vocab.txt:	100%		232k/232k	[00:00<00:00, 6.43MB/s]
special_tokens_map.json:	100%		112/112	[00:00<00:00, 12.1kB/s]
pytorch_model.bin:	100%		438M/438M	[00:02<00:00, 218MB/s]

Figure 5.36 FinBERT model By ProsusAI

```
predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
```

Figure 5.37 Conversion from raw scores to probability scores

```

model.config.id2label

#Tweet #Positive #Negative #Neutral
positive = predictions[:, 0].tolist()
negative = predictions[:, 1].tolist()
neutral = predictions[:, 2].tolist()

table = {'Headline':df_list,
         "Positive":positive,
         "Negative":negative,
         "Neutral":neutral}

df2 = pd.DataFrame(table, columns = ["Headline", "Positive", "Negative", "Neutral"])

```

Figure 5.38 Extract the results of sentiment scores and put into a new dataframe

	Headline	Positive	Negative	Neutral
0	nike workforce reduction trade desk soar ninet...	0.064691	0.452897	0.482412
1	cisco system stock drop five following restruc...	0.022322	0.783024	0.194654
2	form sc 13g statement acquisition beneficial o...	0.030191	0.019631	0.950178
3	form px14a6g notice exempt solicitation submit...	0.020032	0.048665	0.931304
4	arm achieves record revenue paypal share plumm...	0.081968	0.853507	0.064525
...	...	...	...	...
2407	apple see 1q revenue view weakness emerging ma...	0.081989	0.029276	0.888735
2408	apple see 1q revenue view update	0.045928	0.028964	0.925109
2409	letter tim cook apple investor	0.048799	0.037007	0.914194
2410	buyback come back bite firm wsj	0.705303	0.118918	0.175779
2411	foxconn produce iphones india reuters	0.049095	0.020053	0.930852

Figure 5.39 Extract the results of sentiment scores and put into a new dataframe,df2

```

#Add the sentiment column before the headline to determine whether they are positive, negative or neutral

df2.insert(loc=0, column='Sentiment', value='Neutral')

for i, row in df2.iterrows():
    if row['Positive'] > row['Negative'] and row['Positive'] > row['Neutral']:
        df2.loc[i, 'Sentiment'] = 'Positive'
    elif row['Negative'] > row['Positive'] and row['Negative'] > row['Neutral']:
        df2.loc[i, 'Sentiment'] = 'Negative'
    else:
        df2.loc[i, 'Sentiment'] = 'Neutral'

df2

```

Figure 5.40 Add sentiment label into the df2

	Sentiment	Headline	Positive	Negative	Neutral
0	Neutral	nike workforce reduction trade desk soar ninet...	0.064691	0.452897	0.482412
1	Negative	cisco system stock drop five following restruc...	0.022322	0.783024	0.194654
2	Neutral	form sc 13g statement acquisition beneficial o...	0.030191	0.019631	0.950178
3	Neutral	form px14a6g notice exempt solicitation submit...	0.020032	0.048665	0.931304
4	Negative	arm achieves record revenue paypal share plumm...	0.081968	0.853507	0.064525
...	...	...	...	...	...
2407	Neutral	apple see 1q revenue view weakness emerging ma...	0.081989	0.029276	0.888735
2408	Neutral	apple see 1q revenue view update	0.045928	0.028964	0.925109
2409	Neutral	letter tim cook apple investor	0.048799	0.037007	0.914194
2410	Positive	buyback come back bite firm wsj	0.705303	0.118918	0.175779
2411	Neutral	foxconn produce iphones india reuters	0.049095	0.020053	0.930852

Figure 5.41 output of the sentiment label

```
df2['Class'] = np.where(df2['Positive'] > df2['Negative'], 1, 0)
df2.head()
```

	Sentiment	Headline	Positive	Negative	Neutral	Class
0	Neutral	nike workforce reduction trade desk soar ninet...	0.064691	0.452897	0.482412	0
1	Negative	cisco system stock drop five following restruc...	0.022322	0.783024	0.194654	0
2	Neutral	form sc 13g statement acquisition beneficial o...	0.030191	0.019631	0.950178	1
3	Neutral	form px14a6g notice exempt solicitation submit...	0.020032	0.048665	0.931304	0
4	Negative	arm achieves record revenue paypal share plumm...	0.081968	0.853507	0.064525	0



```
# add Vec of df into df2 column which at position 6
df2.insert(6, 'Vec', df['Vec'])

df2.head()
```

	Sentiment	Headline	Positive	Negative	Neutral	Class	Vec
0	Neutral	nike workforce reduction trade desk soar ninet...	0.064691	0.452897	0.482412	0	0.549927
1	Negative	cisco system stock drop five following restruc...	0.022322	0.783024	0.194654	0	0.480751
2	Neutral	form sc 13g statement acquisition beneficial o...	0.030191	0.019631	0.950178	1	0.571132
3	Neutral	form px14a6g notice exempt solicitation submit...	0.020032	0.048665	0.931304	0	0.594192
4	Negative	arm achieves record revenue paypal share plumm...	0.081968	0.853507	0.064525	0	0.584870

Figure 5.42 add class label into the df2 Figure 5.43 insert Vec column of df into df2

```
# Output the df2 to csv file called dataset
df2.to_csv('/content/dataset.csv', index=False)
```

Figure 5.44 Output the df2 as dataset.csv

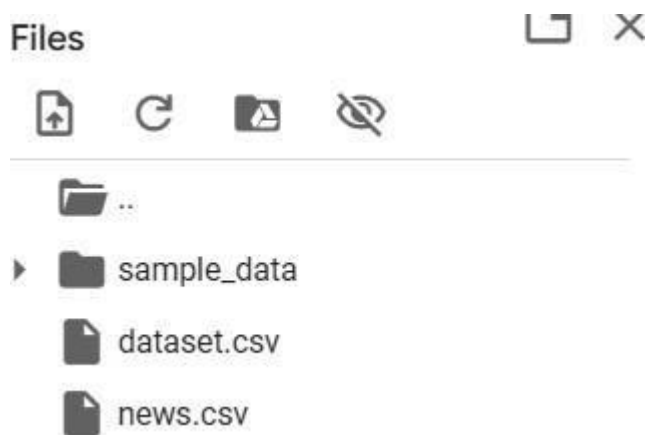


Figure 5.45 Dataset.csv file output to files section

FinBERT, a financial fine-tuning Bert-Based model, specialize on analyse sentiment on the financial text, which is use large financial corpus and trained with financial sentiment classification task. The reason we chose this FinBERT because it excelled in a financial text analysis and specialty for the financial sector, which suitable for our project. By started this section,we firstly install transformers and import the AutoTokenizer and AutoModelForSequenceClassification .Then , it converted the df data into numpy array as the input to the FinBERT model. Then, we imported the FinBERT model from ProsusAI from hugging face and ran the input from df, which outputted the sentiment scores with raw scores in Logit form. Then, next would be converted into the probability scores for the respective sentiment polarity (positive, negative and neutral).Then it extracted the results of sentiment scores and put into a new dataframe,df2 will columns, like Headline, Positive , Negative and Neutral with sentiment scores respectively.Additionally,it added sentiment label in new column based on the sentiment scores for each headline news. Furthermore, it also added the class label in new column based the sentiment label, which class 0 identified as stock might decrease and class 1 might identified as stock might increase. After that, inserted the Vec column from df into df2.Finally, it outputted the df2 as “dataset” csv.file and downloa into filesection.

Based on the previous EDA result, the neutral sentiment had 69.6% of overall distribution, which cannot proved that the news headline data does provide the result whether was positive impacted or negative impacted to their stock prices. Therefore, the prediction will go further to the next phase, modelling, which is for much more accurate prediction on the stock price movement on Apple Incorporation.

## 5.4 Advanced Sentiment Analysis and Further Data Preprocessing

```
[ ] df = pd.read_csv('/content/news_processed.csv')
```

Figure 5.46 Read “news\_processed” Csv file

	Date	Time	Source	\
0	2024-02-16	06:08 AM	IH Market News	
1	2024-02-15	06:34 AM	IH Market News	
2	2024-02-14	04:47 PM	Edgar (US Regulatory)	
3	2024-02-09	04:26 PM	Edgar (US Regulatory)	
4	2024-02-08	06:10 AM	IH Market News	
...	...	...	...	...
2407	2019-01-02	05:07 PM	Dow Jones News	
2408	2019-01-02	05:06 PM	Dow Jones News	
2409	2019-01-02	04:32 PM	Business Wire	
2410	2018-12-28	03:02 AM	Dow Jones News	
2411	2018-12-27	02:09 PM	Dow Jones News	

	Headline	Symbol	\
0	nike workforce reduction trade desk soar ninet...	NASDAQ:AAPL	
1	cisco system stock drop five following restruc...	NASDAQ:AAPL	
2	form sc 13g statement acquisition beneficial o...	NASDAQ:AAPL	
3	form px14a6g notice exempt solicitation submit...	NASDAQ:AAPL	
4	arm achieves record revenue paypal share plumm...	NASDAQ:AAPL	
...	...	...	...
2407	apple see 1q revenue view weakness emerging ma...	NASDAQ:AAPL	
2408	apple see 1q revenue view update	NASDAQ:AAPL	
2409	letter tim cook apple investor	NASDAQ:AAPL	
2410	buyback come back bite firm wsj	NASDAQ:AAPL	
2411	foxconn produce iphones india reuters	NASDAQ:AAPL	

	Company	tokens	Vec
0	Apple Inc	['nike', 'workforce', 'reduction', 'trade', 'd...	0.507322
1	Apple Inc	['cisco', 'system', 'stock', 'drop', 'five', '...	0.437770
2	Apple Inc	['form', 'sc', '13g', 'statement', 'acquisitio...	0.509907
3	Apple Inc	['form', 'px14a6g', 'notice', 'exempt', 'solic...	0.532671
4	Apple Inc	['arm', 'achieves', 'record', 'revenue', 'payp...	0.531288
...	...	...	...
2407	Apple Inc	['apple', 'see', '1q', 'revenue', 'view', 'wea...	0.568491
2408	Apple Inc	['apple', 'see', '1q', 'revenue', 'view', 'und...	0.674522

✓ 0s completed at 1:07 AM

Figure 5.47 Output the “news\_processed” columns data

In this time, processed stock news headlines data is used and that is now labeled in terms of vectorized numerical data in “Vec” column as was derived from the data from “tokens” column data, which was represented as the overall numerical meaning of those tokens for each rows.

This Csv file consisted of the 8 columns of data, which were consisted of 8 columns, “Date”, ”Time”, ”Source”, “Headline”, “Symbol”, “tokens” and “Vec”. This was because vectorized data can be processed by machine learning models.

```

] ] from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch
from transformers import pipeline

#Finbert
tokenizer = AutoTokenizer.from_pretrained("ProsusAI/finbert")
model = AutoModelForSequenceClassification.from_pretrained("ProsusAI/finbert")

/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggi)
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or
warnings.warn(
tokenizer_config.json: 100% ██████████ 252/252 [00:00<00:00, 3.31kB/s]
config.json: 100% ██████████ 758/758 [00:00<00:00, 15.3kB/s]
vocab.txt: 100% ██████████ 232k/232k [00:00<00:00, 4.04MB/s]
special_tokens_map.json: 100% ██████████ 112/112 [00:00<00:00, 1.85kB/s]
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureRek
warnings.warn(
pytorch_model.bin: 100% ██████████ 438M/438M [00:02<00:00, 152MB/s]

] ] nlp= pipeline("sentiment-analysis", model="ProsusAI/finbert", tokenizer="ProsusAI/finbert")

```

Figure 5.48 Load the FinBERT model for sentiment analysis by using the Hugging Face Transformers library.

In this process, it imported the necessary components: AutoTokenizer, AutoModel For Sequence Classification, and pipeline from transformers package, as well as the torch to enable PyTorch functionality. Then, FinBERT model was loaded by initializing a tokenizer and model, both sourced from Hugging Face's "ProsusAI/finbert" pre-trained model repository. The tokenizer was responsible for converting raw text, which was the headlines data into tokenized inputs that the model can understand, while the model itself was a sequence classification model trained for financial sentiment analysis. After that, it initialized a sentiment analysis pipeline using the pre-trained FinBERT model and tokenizer, which was allowing the user to perform sentiment analysis on the “Headlines” data that contain financial information.

```

# generate sentiment score

def get_sentiment(text):
    try:
        result = nlp(text)[0]
        label = result['label']
        score = result['score']
        if label == 'positive':
            return score
        elif label == 'negative':
            return -score
        else:
            return 0
    except Exception as e:
        print(f"Error processing text: {e}")
        return 0

df['sentiment_score'] = df['Headline'].apply(lambda x: get_sentiment(x))

```

Figure 5.50 Call the generalization of sentiment score function

To get the "Headline" data's sentiment score, the `get_sentiment` function is called to process text of Headlines to generate the sentiment score using the FinBERT model. This function first applies the sentiment analysis pipeline (`nlp`) to the input text, which was extracting the result's label (either 'positive', 'negative', or 'neutral') and score (a confidence measure). If the label is 'positive', the function returns the score as is; if the label is 'negative', it returns the negative of the score to represent negative sentiment. In case of a neutral or unrecognized label, the function returned 0 to signify neutral sentiment. A try-except block was used for error handling, ensuring that if an exception occurs during text processing, it will print an error message and return a neutral score of 0. Finally, the function was applied to each entry in the 'Headline' column of "news\_processed" dataframe using a lambda function, and the sentiment scores are stored in a new column called 'sentiment\_score'. This allows for automated sentiment scoring of financial headlines in the dataset.

```
print(df['sentiment_score'])
```

0	0.000000
1	-0.783024
2	0.000000
3	0.000000
4	-0.853508
	...
2407	0.000000
2408	0.000000
2409	0.000000
2410	0.705303
2411	0.000000

Name: sentiment\_score, Length: 2412, dtype: float64

Figure 5.51 Print the result of the sentiment score of the “Headline” data

After that, the column “sentiment score” being printed, which contains the sentiment scored for the 2412 rows data of “news\_processed” csv file. The scores range between positive and negative values, with some values being neutral (0). Negative scores represented negative sentiment in the text of news headlines, while positive scores represent positive sentiment, and a score of 0 represents neutral sentiment. This output showed the application of the `get_sentiment` function across the dataset, where each headline has been assigned a corresponding sentiment score.

```
[ ] # output latest dataframe to csv
```

```
df.to_csv('news_processed_with_sentiment.csv', index=False)
```

Figure 5.52 Output the updated dataframe with additional of “sentiment\_score” column data.

Therefore, the processed DataFrame, which now included the calculated sentiment scores, to a CSV file named `news_processed_with_sentiment.csv`. The `to_csv` function was used to write the DataFrame to a CSV file, and the parameter `index=False` ensures that the row indices are not written into the file. This allows for a clean output of the DataFrame content, with only the 9 columns headers, “Date”, ”Time”, ”Source”, “Headline”, “Symbol”, “tokens”, “Vec” and “sentiment\_score” and their corresponding data saved in the CSV file.

```
[ ] df2 = pd.read_csv('/content/news_processed_with_sentiment.csv')  
  
# remove the line with value 0 in sentiment score  
df2 = df2[df2['sentiment_score'] != 0]
```

Figure 5.53 Read “news\_processed\_with\_sentiment” Csv file and then Filtering rows with Zero Sentiment Scores.

For the first part, loaded a CSV file, “news\_processed\_with\_sentiment.csv” into a new DataFrame `df2` using the pandas library. The path to the file is specified as `/content/news_processed_with_sentiment.csv`, that is stored in the Colab environment.

Next, the next part of the code filters the DataFrame, `df2` to remove rows where the `sentiment_score` column has a value of 0. The line `df2 = df2[df2['sentiment_score'] != 0]` creates a new DataFrame that only contains rows where the sentiment score is non-zero. This removes any neutral entries, ensuring that the final dataset consists only of positive or negative sentiments.

```
# prompt: print row
print(df2)
```

	Date	Time	Source	Headline	Symbol
1	2024-02-15	06:34 AM	IH Market News	cisco system stock drop five following restruc...	NASDAQ:AAPL
4	2024-02-08	06:10 AM	IH Market News	arm achieves record revenue paypal share plumm...	NASDAQ:AAPL
5	2024-02-07	05:49 AM	IH Market News	moody downgrade nycb junk status ford declares...	NASDAQ:AAPL
8	2024-02-05	05:31 AM	IH Market News	santanders stock decline amid sanction evasion...	NASDAQ:AAPL
10	2024-02-02	08:47 AM	IH Market News	apple beat q1 revenue earnings estimate amid c...	NASDAQ:AAPL
...	...	...	...	...	...
2402	2019-01-02	05:36 PM	Dow Jones News	apple revise guidance see lower revenue first ...	NASDAQ:AAPL
2403	2019-01-02	05:31 PM	Dow Jones News	apple revise guidance see lower revenue first ...	NASDAQ:AAPL
2404	2019-01-02	05:22 PM	Dow Jones News	apple revise guidance see lower revenue first ...	NASDAQ:AAPL
2405	2019-01-02	05:15 PM	Dow Jones News	apple see 1q revenue view china weakness 4th u...	NASDAQ:AAPL
2410	2018-12-28	03:02 AM	Dow Jones News	buyback come back bite firm wsj	NASDAQ:AAPL

	Company	tokens	Vec
1	Apple Inc	['cisco', 'system', 'stock', 'drop', 'five', '...', '...']	0.437770
4	Apple Inc	['arm', 'achieves', 'record', 'revenue', 'payp...', '...']	0.531288

	sentiment_score
1	-0.783024
4	-0.853508
5	-0.618619
8	-0.972149
10	-0.683396
...	...
2402	-0.933342
2403	-0.962072
2404	-0.958088
2405	-0.522876
2410	0.705303

[733 rows x 9 columns]

Figures 5.54 and 5.55 The latest dataframe df2 results after filtering out neutral sentiment scores.

In this latest dataframe of df2, Key columns include the “Date” and “Time” of the financial news, the “Source” of the news (such as IH Market News or Dow Jones News), the “Headline” summarizing the financial event, and the associated stock “Symbol” (like NASDAQ for Apple Inc.). Each headline has been tokenized into a list of Tokens, and there's a Vec column, likely



representing some headlines data's vectorized meanings in numerical form. The most important column is the sentiment score, where negative values indicate negative sentiment and positive values reflect positive sentiment. For instance, some of the headlines in figures 4.53 display negative sentiment with values like -0.783024 or -0.972149, while a few, such as one in row 2410, show positive sentiment with a score of 0.705303. This dataset is now ready for further analysis, such as examining the relationship between news sentiment and stock price performance.

## 5.5 Time-Series Stock Price Data Preprocessing and Data Visualisation

```
# merge the 6 stock data files into 1

# Assuming you have 6 dataframes named df_stock1, df_stock2, ..., df_stock6
# Replace these with your actual dataframe names
import pandas as pd

# Concatenate the dataframes
merged_df = pd.concat([pd.read_csv('/content/Download Data - STOCK_US_XNAS_AAPL (1).csv'),
                       pd.read_csv('/content/Download Data - STOCK_US_XNAS_AAPL (2).csv'),
                       pd.read_csv('/content/Download Data - STOCK_US_XNAS_AAPL (3).csv'),
                       pd.read_csv('/content/Download Data - STOCK_US_XNAS_AAPL (5).csv'),
                       pd.read_csv('/content/Download Data - STOCK_US_XNAS_AAPL (4).csv'),
                       pd.read_csv('/content/Download Data - STOCK_US_XNAS_AAPL.csv')], ignore_index=True)

# Print the merged dataframe
print(merged_df)
```

	Date	Open	High	Low	Close	Volume
0	12/31/2020	134.08	134.74	131.72	132.69	99,116,594
1	12/30/2020	135.58	135.99	133.40	133.72	96,452,117
2	12/29/2020	138.05	138.79	134.34	134.87	121,047,297
3	12/28/2020	133.99	137.34	133.51	136.69	124,486,203
4	12/24/2020	131.32	133.46	131.10	131.97	54,930,059
...	...	...	...	...	...	...
1271	01/08/2019	37.39	37.96	37.13	37.69	164,101,248
1272	01/07/2019	37.18	37.21	36.48	36.98	219,111,048
1273	01/04/2019	36.13	37.14	35.95	37.07	234,428,280
1274	01/03/2019	36.00	36.43	35.50	35.55	365,248,812
1275	01/02/2019	38.72	39.71	38.56	39.48	148,158,952

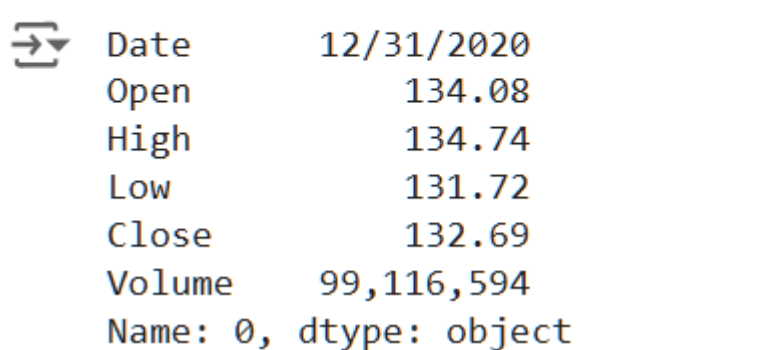
[1276 rows x 6 columns]

Figure 5.56 Merged 6 CSV files containing Apple Inc. (AAPL) stock data into a single DataFrame.

It begun by reading each CSV file using `pd.read_csv()`, which loads the stock data including columns for Date, Open, High, Low, Close, and Volume. The six DataFrames were then combined using `pd.concat()`, which merged them into one large DataFrame (`merged_df`) with

1,276 rows and 6 columns. The `ignore_index=True` parameter ensured that the rows are re-indexed sequentially. Finally, the merged DataFrame was printed, showing the stock data in chronological order with values for each day's opening, closing, high, and low prices, as well as trading volume.

```
▶ print(merged_df.iloc[0])
```



```
⇒ Date      12/31/2020
   Open      134.08
   High      134.74
   Low       131.72
   Close     132.69
   Volume    99,116,594
   Name: 0, dtype: object
```

Figure 5.57 Displayed the first row of the merged\_df dataset

The output showed the stock data for Apple Inc. (AAPL) on December 31, 2020, by accessing the first row of the merged DataFrame using the `.iloc[0]` method. This row included key financial indicators for the day: the stock opened at a price of 134.08, reached a high of 134.74, a low of 131.72, and closed at 132.69. Additionally, the trading volume for that day was 99,116,594, indicating the total number of shares traded. This row represented a typical snapshot of Apple's daily stock market data.

```

▶ # compare the new_processed_with_sentiment_score with the merge_df

# Convert 'Date' column in merged_df to datetime objects
merged_df['Date'] = pd.to_datetime(merged_df['Date'])
merged_df.drop_duplicates(inplace=True)

# Convert 'Date' column in df2 to datetime objects
df2['Date'] = pd.to_datetime(df2['Date'])

# Merge the dataframes based on the 'Date' column
merged_df = merged_df.merge(df2[['Date', 'sentiment_score']], on='Date', how='left')

# Drop rows with NaN values
merged_df.dropna(inplace=True)

# Remove rows with empty strings in any column
merged_df = merged_df[~merged_df.applymap(lambda x: x == '').any(axis=1)]

# Print the merged dataframe
print(merged_df)

```

```

↩

```

	Date	Open	High	Low	Close	Volume	sentiment_score
0	2020-12-31	134.08	134.74	131.72	132.69	99,116,594	-0.919542
5	2020-12-23	132.16	132.43	130.78	130.96	88,223,688	-0.916875
9	2020-12-17	128.90	129.58	128.05	128.70	94,359,805	-0.614334
10	2020-12-16	127.41	128.37	126.56	127.81	98,208,594	-0.924740
22	2020-11-30	116.97	120.97	116.81	119.05	169,410,203	-0.797998
...	...	...	...	...	...	...	...
1542	2019-01-03	36.00	36.43	35.50	35.55	365,248,812	-0.899975
1543	2019-01-02	38.72	39.71	38.56	39.48	148,158,952	-0.933342
1544	2019-01-02	38.72	39.71	38.56	39.48	148,158,952	-0.962072
1545	2019-01-02	38.72	39.71	38.56	39.48	148,158,952	-0.958088
1546	2019-01-02	38.72	39.71	38.56	39.48	148,158,952	-0.522876

```

[677 rows x 7 columns]
<ipython-input-15-f1f4bac79ff5>:21: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.map instead.
merged_df = merged_df[~merged_df.applymap(lambda x: x == '').any(axis=1)]

```

Figure 5.58 and Figure 5.59 Process and Output of merging two DataFrames: one containing stock data (merged\_df) and the other containing sentiment scores (df2).

Firstly, it converted the Date column in both DataFrames to datetime objects to ensure proper alignment for merging. Duplicate rows were then removed from the stock data (merged\_df). The two DataFrames were merged on the Date column using a left join, meaning all rows from the stock data are retained, and corresponding sentiment scores from df2 were added where available. After merging, rows with missing values (NaNs) were dropped, ensuring that only complete rows remained. Additionally, rows with empty strings in any column were removed.

Finally, the merged DataFrame was printed, showing 677 rows and 7 columns, including the sentiment\_score alongside stock data such as open, high, low, close prices, and volume. This merged dataset now combined both stock and sentiment information.

```
#Handle outlier

# Calculate the IQR
Q1 = merged_df['Open'].quantile(0.25)
Q3 = merged_df['Open'].quantile(0.75)
IQR = Q3 - Q1

# Define the lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers = merged_df[(merged_df['Open'] < lower_bound) | (merged_df['Open'] > upper_bound)]

# Remove outliers
merged_df = merged_df[(merged_df['Open'] >= lower_bound) & (merged_df['Open'] <= upper_bound)]
```

Figure 5.60 Outlier detection and removal based on the Open column in the merged\_df DataFrame using the Interquartile Range (IQR) method.

This code used the Interquartile Range (IQR) method to detect and remove outliers in the Open column of the merged\_df DataFrame. First, the first quartile (Q1) and third quartile (Q3) of the Open values were calculated to determine the IQR, which represents the range between these quartiles. Using the IQR, lower and upper bounds are defined to identify outliers—values below Q1 minus 1.5 times the IQR or above Q3 plus 1.5 times the IQR are considered outliers. It also identified the rows that contain these outlier values and then removes them from the dataset, ensuring that the cleaned DataFrame only contained values within the typical range for the Open column. This process helped improve the reliability of subsequent analyses by eliminating extreme or anomalous data points.

```
# generate boxplot after the handle outlier

import matplotlib.pyplot as plt

# Create a boxplot of the 'sentiment_score' column
plt.boxplot(merged_df['Open'])
plt.title('Stock Prices Open Boxplot')
plt.ylabel('Open')
plt.show()
```

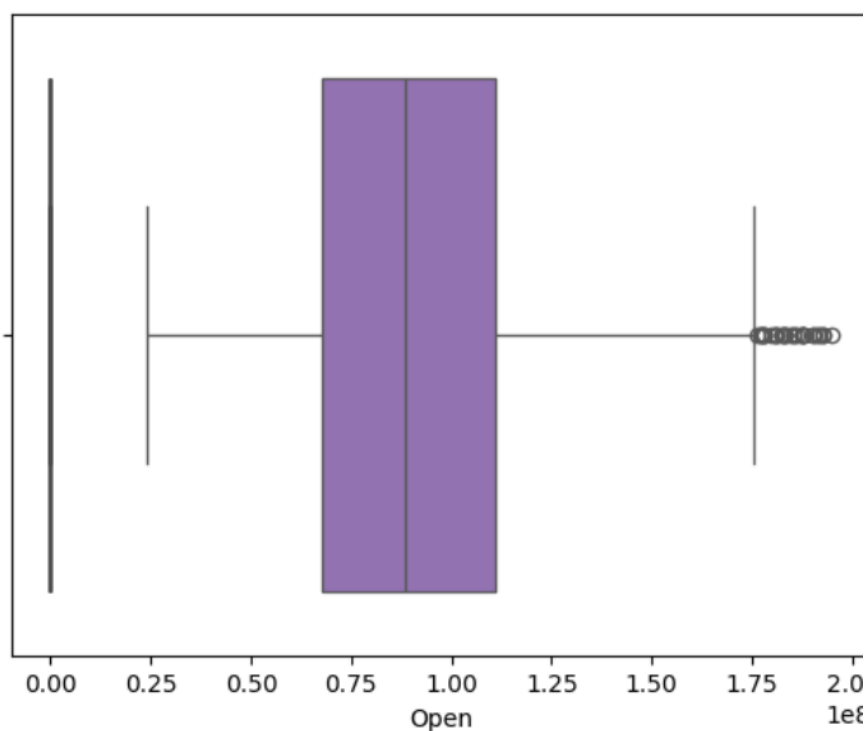


Figure 5.61 Boxplot generated to visualize distribution of the Open column of merged\_df after handling outliers.

In this code, it generated a boxplot to visually represent the distribution of the Open stock prices in the merged\_df DataFrame after handling outliers. Using the matplotlib.pyplot library, the boxplot displayed the interquartile range (IQR) as a box, with the median price indicated by a line inside the box. In this case, it appeared that the median lies somewhere close to the middle of the range of values. Also, it represented the middle 50% of the data, showing the interquartile range between the first (Q1) and third quartiles (Q3). A relatively large box indicated variability in the data, suggesting that while the majority of data is centered in the middle range, there are some significant fluctuations in opening stock prices. The whiskers extended from the box show the range of the data, excluding any outliers. If the box were smaller, it would indicate that most of the Open prices cluster around a narrow range, meaning the stock opens at similar

prices over time. Any remaining outliers are shown as individual dots outside the whiskers. This visualization provides a clear summary of the central tendency (median), spread (IQR), and any extreme values that still exist in the Open stock prices after outlier removal. The plot helps highlight any significant deviations or trends in the stock prices, facilitating further analysis of the data.

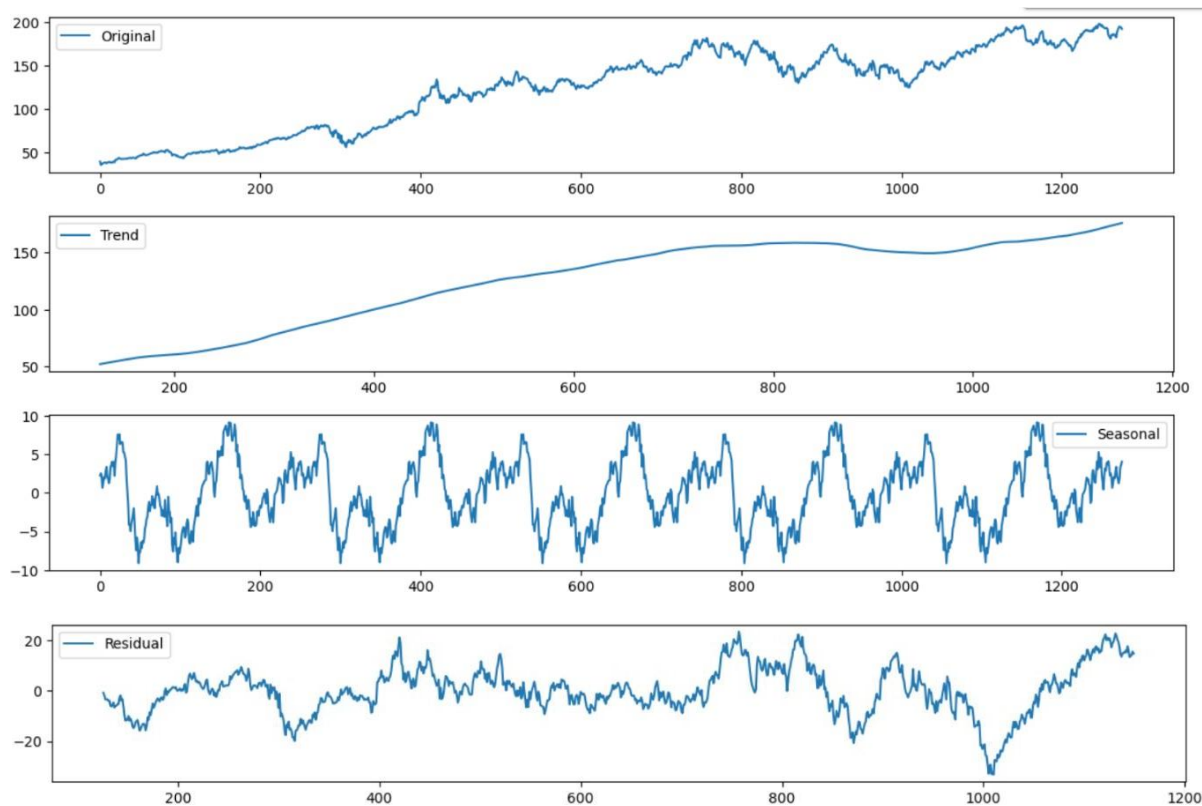
```
#do seasonal composition
import matplotlib.pyplot as plt # import the module for plotting
from statsmodels.tsa.seasonal import seasonal_decompose

# Perform seasonal decomposition on the 'Close' price
result = seasonal_decompose(merged_df['Close'], model='additive', period=252)

# Extract the components
trend = result.trend
seasonal = result.seasonal
residual = result.resid

# Plot the components
plt.figure(figsize=(12, 8))
plt.subplot(411)
plt.plot(merged_df['Close'], label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal, label='Seasonal')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residual')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

Figure 5.62 Process of Seasonal Composition of the Close Stock prices in merged\_df



Figures 5.63 and 5.64 showed the Output of Seasonal Decomposition's visualisations using the `seasonal_decompose` function from `statsmodels` library.

In this process, it performed a seasonal decomposition of the stock's Close prices using an additive model to separate the data into three components: trend, seasonality, and residuals. The decomposition process isolated the long-term movement of the stock price (trend), repetitive patterns that occur over a yearly cycle (seasonality), and the random fluctuations or noise (residuals) not captured by the trend or seasonality. The trend component showed the overall upward growth of the stock, while the seasonal component highlighted regular fluctuations, potentially linked to market events or trading patterns. The residuals represented unpredictable variations, such as responses to unexpected market news. Visualizing these components helped better understand the behavior of stock prices, allowing for more informed analyses by distinguishing systematic patterns from random noise.



```
[ ] # Ensure 'Date' column exists in df2
    print(df2.columns)

    # If 'Date' column is missing, you might need to reload or recreate df2

    # Perform the merge operation with 'Date' column included in df2
    merged_df = merged_df.merge(df2[['Date', 'Vec']], on='Date', how='left')
```

```
↳ Index(['Date', 'Time', 'Source', 'Headline', 'Symbol', 'Company', 'tokens',
        'Vec', 'sentiment_score'],
        dtype='object')
```

Figures 5.65 showed the Process for ensuring the Date column existed in df2 and Left join performed between two DataFrames, merged\_df and df2, based on the Date column.

First, it checked the columns of df2 to verify the presence of the Date column, which was essential for merging the two DataFrames. If the Date column was missing, it may need to reload or recreate df2 to include it. The merge operation was performed by aligning rows from both DataFrames where the Date values match. The merge specifically added the Vec column from df2 to merged\_df, ensuring that all rows from merged\_df were retained, even if no matching dates were found in df2. The resulting DataFrame included columns such as “Date”, “Time”, “Source”, “Headline”, “Symbol”, “Company”, “tokens”, “Vec”, and “sentiment\_score”, combining information from both DataFrames for further analysis.



```

▶ print(merged_df)
↕
      Date    Open    High    Low    Close    Volume  sentiment_score
0  2020-12-31  134.08  134.74  131.72  132.69  99,116,594      -0.919542
1  2020-12-23  132.16  132.43  130.78  130.96  88,223,688      -0.916875
2  2020-12-17  128.90  129.58  128.05  128.70  94,359,805      -0.614334
3  2020-12-16  127.41  128.37  126.56  127.81  98,208,594      -0.924740
4  2020-11-30  116.97  120.97  116.81  119.05  169,410,203      -0.797998
...
1804 2019-01-02  38.72  39.71  38.56  39.48  148,158,952      -0.958088
1805 2019-01-02  38.72  39.71  38.56  39.48  148,158,952      -0.522876
1806 2019-01-02  38.72  39.71  38.56  39.48  148,158,952      -0.522876
1807 2019-01-02  38.72  39.71  38.56  39.48  148,158,952      -0.522876
1808 2019-01-02  38.72  39.71  38.56  39.48  148,158,952      -0.522876

      Vec
0    0.577343
1    0.650329
2    0.677827
3    0.629622
4    0.649070
...
1804 0.593876
1805 0.567607
1806 0.590094
1807 0.607391
1808 0.593876

[1809 rows x 8 columns]

```

Figures 5.66 Displayed the merged DataFrame, which now consisted of 1,809 rows and 8 columns after previous merging process.

The columns include stock market data such as “Date”, “Open”, “High”, “Low”, “Close”, “Volume”, and two additional columns: `sentiment_score` and `Vec` after merged from previous code snippet. The stock market data represented daily information about Apple’s stock, including its opening price, highest and lowest prices during the day, closing price, and the trading volume. The `sentiment_score` column shows the sentiment analysis results, with negative values representing negative sentiment and positive values reflecting positive sentiment. The `Vec` column contains vectorized data from the sentiment analysis process, which likely represents additional features derived from the headlines. Each row corresponds to a specific trading day, showing how stock prices and sentiment evolve over time.

```

# Replace commas with empty strings and convert to float
merged_df['Volume'] = merged_df['Volume'].str.replace(',', '').astype(float)

```

Figures 5.67 Cleaning process of the Volume column in the merged\_df DataFrame by removing commas, and converting into float data type

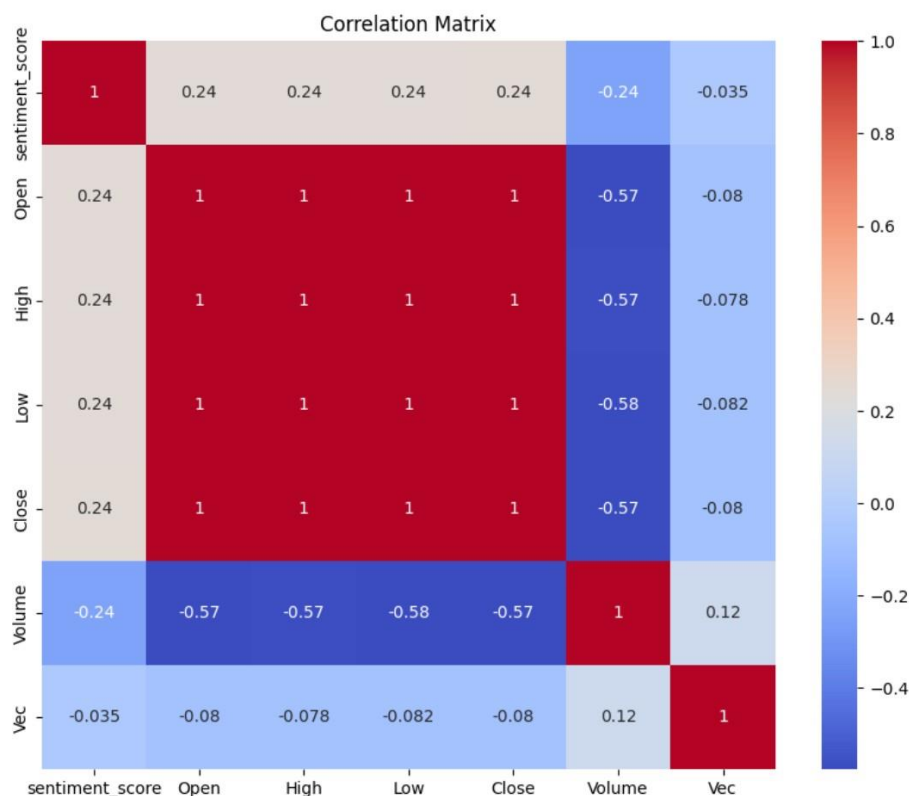
In this code snippet, the str.replace(', ') method removed all commas, which are used as thousand separators, making the data consistent for numerical operations. Once the commas are removed, the .astype(float) function is applied to convert the cleaned Volume values from strings to floating-point numbers. This conversion allowed the Volume data to be used in calculations and analysis, such as aggregating total volume or exploring correlations with other stock metrics.

```
Suggested code may be subject to a license | medium.com/@ditechdiop/unlock-the-power-of-python-a-comprehensive-guide-to-unlock-its-full-potential-66acdf716b70
# Correlation map

import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the correlation matrix
correlation_matrix = merged_df[['sentiment_score', 'Open', 'High', 'Low', 'Close', 'Volume', 'Vec']].corr()

# Create the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



Figures 5.68 and 5.69 Calculation and visualization of a correlation matrix for key stock market features, including “sentiment\_score”, “Open”, “High”, “Low”, “Close”, “Volume”, and “Vec” numerical columns.

The correlation matrix, shown as a heatmap, highlighted the relationships between these variables. The stock price variables (Open, High, Low, Close) were highly correlated with each other (correlation of 1.0), indicating that they move closely together throughout the trading day. The sentiment\_score had a moderate positive correlation (0.24) with stock prices, suggesting that positive sentiment may have a weak-to-moderate influence on higher stock prices. The Volume showed a moderate negative correlation (-0.57) with stock prices, implying that higher trading volumes might be associated with lower stock prices, potentially reflecting selling pressure. The Vec column had low correlations with all other variables, indicating it captures different features not directly tied to stock price movements or volume. This analysis helped in understanding the relationships between sentiment, stock prices, and trading volume.

```
[ ] merged_df.merge(df2[['Date', 'Headline']], on='Date', how='left')
```

	Date	Open	High	Low	Close	Volume	sentiment_score	Vec	Headline
0	2020-12-31	134.08	134.74	131.72	132.69	99116594.0	-0.919542	0.577343	apple remove forty-six zero unlicensed apps ch...
1	2020-12-23	132.16	132.43	130.78	130.96	88223688.0	-0.916875	0.650329	apple china app store shed game pressure
2	2020-12-17	128.90	129.58	128.05	128.70	94359805.0	-0.614334	0.677827	antitrust case facing big tech update
3	2020-12-16	127.41	128.37	126.56	127.81	98208594.0	-0.924740	0.629622	facebook accuses apple hurting small business ...
4	2020-11-30	116.97	120.97	116.81	119.05	169410203.0	-0.797998	0.649070	italian antitrust authority fine apple twelve ...
...	...	...	...	...	...	...	...	...	...
8274	2019-01-02	38.72	39.71	38.56	39.48	148158952.0	-0.522876	0.607391	apple see 1q revenue view china weakness 4th u...
8275	2019-01-02	38.72	39.71	38.56	39.48	148158952.0	-0.522876	0.593876	apple revise guidance see lower revenue first ...
8276	2019-01-02	38.72	39.71	38.56	39.48	148158952.0	-0.522876	0.593876	apple revise guidance see lower revenue first ...
8277	2019-01-02	38.72	39.71	38.56	39.48	148158952.0	-0.522876	0.593876	apple revise guidance see lower revenue first ...
8278	2019-01-02	38.72	39.71	38.56	39.48	148158952.0	-0.522876	0.593876	apple see 1q revenue view china weakness 4th u...

8279 rows × 9 columns

Figure 5.70 Displayed a merging process of merged\_df DataFrame with df2 ‘s ‘Date’ and “Headline” while aligning with the ‘Date’ columns from both dataframe with Left join.

The 9 columns include Date, Open, High, Low, Close, Volume, sentiment\_score, Vec, and Headline., which now consisting of 8,279 rows, were combining stock price data with associated financial news headlines. The stock price data (Open, High, Low, Close, and Volume) represented Apple's daily stock performance, while the sentiment\_score indicated the sentiment analysis of the news headlines, with negative values represented negative sentiment and positive values represented positive sentiment. The Vec column contained vectorized data

from the sentiment analysis process, likely represented features extracted from the news articles. The `Headline` column contained news headlines associated with the stock's performance on the corresponding dates, offering a textual context for the sentiment score. This merged dataset can be used for further analysis, such as examining how specific news events and their sentiment relate to changes in stock prices and trading volume.

```
merged_df = merged_df.drop(columns=['Vec'])
```

Figure 5.71 Removed the `Vec` column from the `merged_df` DataFrame

In this code snippet, the `.drop()` method is used with the `columns` parameter to specify that the column labeled `Vec` should be deleted. This operation resulted in a new version of `merged_df` without the `Vec` column, which likely contained vectorized features or data that are no longer needed for the analysis. The `Vec` column may have been useful during previous steps of the analysis, but it might now be redundant or irrelevant for the final stages of the analysis, such as correlating sentiment scores or news headlines with stock price movements. By removing this column, the DataFrame is simplified into 8 columns, retaining only the essential data for further analysis, such as stock prices, sentiment scores, and headlines, which were more directly related to the prediction task at hand.

```
[ ] # remove na
merged_df.dropna(inplace=True)
```

```
[ ] # drop the duplicate rows
merged_df.drop_duplicates(inplace=True)
```

Figure 5.72 Functions to remove Missing Values and Duplicate Rows in `merged_df` Dataframe

```

▶ print(merged_df)
↕
   Date      Open      High      Low      Close      Volume  sentiment_score \
0  2020-12-31  134.08  134.74  131.72  132.69  99116594.0      -0.919542
1  2020-12-23  132.16  132.43  130.78  130.96  88223688.0      -0.916875
2  2020-12-17  128.90  129.58  128.05  128.70  94359805.0      -0.614334
3  2020-12-16  127.41  128.37  126.56  127.81  98208594.0      -0.924740
4  2020-11-30  116.97  120.97  116.81  119.05  169410203.0     -0.797998
...
8250 2019-01-02  38.72  39.71  38.56  39.48  148158952.0     -0.958088
8263 2019-01-02  38.72  39.71  38.56  39.48  148158952.0     -0.522876
8264 2019-01-02  38.72  39.71  38.56  39.48  148158952.0     -0.522876
8265 2019-01-02  38.72  39.71  38.56  39.48  148158952.0     -0.522876
8266 2019-01-02  38.72  39.71  38.56  39.48  148158952.0     -0.522876

      Headline
0  apple remove forty-six zero unlicensed apps ch...
1           apple china app store shed game pressure
2           antitrust case facing big tech update
3  facebook accuses apple hurting small business ...
4  italian antitrust authority fine apple twelve ...
...
8250 apple see 1q revenue view china weakness 4th u...
8263 apple revise guidance see lower revenue first ...
8264 apple revise guidance see lower revenue first ...
8265 apple revise guidance see lower revenue first ...
8266 apple see 1q revenue view china weakness 4th u...

[1809 rows x 8 columns]

```

Figure 5.73 Output of the latest dataframe after perform data cleaning steps

This cleaned `merged_df` DataFrame consisted of 1,809 rows and 8 columns, combining stock price data, sentiment scores, and associated financial news headlines. The columns include `Date`, representing each trading day; `Open`, `High`, `Low`, and `Close` prices for Apple's stock; `Volume`, indicating the number of shares traded; `sentiment_score`, which reflected the sentiment of news headlines (positive or negative); and the `Headline` column contained the relevant financial news articles. After performing data cleaning operations to remove missing values and duplicate rows, the DataFrame was now one step away for ready to feed in to the models .

```

# Function to get sentiment score using FinBERT
def get_sentiment(text):
    try:
        result = nlp(text)[0]
        label = result['label']
        score = result['score']
        if label == 'positive':
            return score
        elif label == 'negative':
            return -score
        else:
            return 0
    except Exception as e:
        print(f"Error processing text: {e}")
        return 0

# Apply the sentiment analysis function to the 'Headline' column
merged_df['new_sentiment_score'] = merged_df['Headline'].apply(lambda x: get_sentiment(x))

# Filter out rows where the new sentiment score doesn't match the existing score
merged_df = merged_df[merged_df['sentiment_score'] == merged_df['new_sentiment_score']]

# Drop the 'new_sentiment_score' column as it's no longer needed
merged_df = merged_df.drop(columns=['new_sentiment_score'])

```

Figure 5.74 Functions to get sentiment score of Headline using FinBERT , filtered out rows that not matched between two sets of sentiment scores and drop the new sentiment score column after done filtering.

This code snippet is designed to validate and cross-check sentiment scores using FinBERT. It begins by defining the `get_sentiment(text)` function, which processed each news headline to calculate its sentiment. The function used a pre-trained FinBERT model to analyze the headline text and returns a sentiment score based on whether the sentiment is positive, negative, or neutral. If the sentiment was positive, the function returned the score, while a negative sentiment returned a negative score. For neutral or unclassified sentiments, the function returned 0. Additionally, if an error occurred during the sentiment analysis, the function caught the exception and logs an error message.

The function was then applied to the Headline column of the `merged_df` DataFrame to generate a new sentiment score, which is stored in a column called `new_sentiment_score`. The code filtered out rows where the original `sentiment_score` did not match the newly computed score, ensuring only consistent sentiment scores remain. Once this validation was completed,



the `new_sentiment_score` column is removed, as it is no longer needed. This process resulted in a clean and validated dataset, where the sentiment analysis from FinBERT aligned with the original data, improving the reliability of the sentiment scores.

```
# Create lagged features
merged_df['Close_lag1'] = merged_df['Close'].shift(1)
merged_df['Close_lag2'] = merged_df['Close'].shift(2)

#Calculate rolling statistics
merged_df['Close_rolling_mean'] = merged_df['Close'].rolling(window=5).mean()
merged_df['Close_rolling_std'] = merged_df['Close'].rolling(window=5).std()

# Fill missing values (if any)
merged_df.fillna(method='ffill', inplace=True) # Forward fill

# Print the updated dataframe
print(merged_df)
```

Figure 5.75 Created Lagged Features and Rolling Statistics, filled missing values and printed the final updated dataframe

This code created additional features from the existing Close price column in the `merged_df` DataFrame to enhance the dataset for further analysis or modeling. It first generated two lagged features: `Close_lag1` and `Close_lag2`. These are created by shifting the Close column by 1- and 2-time steps, respectively, to capture the closing prices of the previous day and two days prior. Lagged features were useful for incorporating historical price data into predictive models, as they provide context about the stock's recent performance. Next, the code calculated rolling statistics for the Close column. It computed the 5-day rolling mean (`Close_rolling_mean`) and the 5-day rolling standard deviation (`Close_rolling_std`) to capture short-term trends and volatility in the stock prices. These rolling statistics helped smooth out daily fluctuations and provided a clearer picture of price movements over a defined period.

After generating the lagged features and rolling statistics, the code checked for any missing values that may have been introduced (especially in the first few rows where lagging and rolling operations would create NaN values). It used the `fillna()` method with forward fill (`ffill`) to propagate the last valid observation forward, ensuring the dataset remains complete and free of gaps. Finally, the updated DataFrame, including the newly created features, is printed.

```

0      2/1/2019  38.72  39.71  38.56  39.48  148,158,952  -0.522876
1      2/1/2019  38.72  39.71  38.56  39.48  148,158,952  -0.933342
2      2/1/2019  38.72  39.71  38.56  39.48  148,158,952  -0.962072
3      2/1/2019  38.72  39.71  38.56  39.48  148,158,952  -0.958088
4      3/1/2019  36.00  36.43  35.50  35.55  365,248,812  -0.899975
..      ...      ...      ...      ...      ...      ...
672    23/1/2024  195.02  195.75  193.83  195.18  42,355,594  -0.960746
673    24/1/2024  195.42  196.38  194.34  194.50  53,631,320  -0.954821
674    25/1/2024  195.22  196.27  193.11  194.17  54,822,129  -0.912749
675    25/1/2024  195.22  196.27  193.11  194.17  54,822,129  -0.968698
676    26/1/2024  194.27  194.76  191.94  192.42  44,594,008  -0.958609

      Close_lag1  Close_lag2  Close_rolling_mean  Close_rolling_std
0              0.00         0.00              0.000             0.000000
1             39.48         0.00              0.000             0.000000
2             39.48         39.48              0.000             0.000000
3             39.48         39.48              0.000             0.000000
4             39.48         39.48              38.694             1.757549
..            ...         ...              ...              ...
672          191.56        188.63             188.336            5.278677
673          195.18        191.56             190.510            5.091041
674          194.50        195.18             192.808            2.709127
675          194.17        194.50             193.916            1.380083
676          194.17        194.17             194.088            1.019544

      Headline
0  apple see 1q revenue view china weakness 4th u...
1  apple revise guidance see lower revenue first ...
2  apple revise guidance see lower revenue first ...
3  apple revise guidance see lower revenue first ...
4  apple warning sale stuns market wsj
..  ...
672 wall street highlight vroom shuts online car s...
673 wall street highlight ebay slash nine workforc...
674 wall street highlight nokias margin surprise f...
675 apple smartphone shipment china fell fourth qu...
676 wall street highlight intels disappointing qua...

[677 rows x 12 columns]

```

Figure 5.76 Output of final merged\_df DataFrame

The output showed the merged\_df DataFrame, which now includes 12 columns and 677 rows. The added columns were Close\_lag1, Close\_lag2, Close\_rolling\_mean, and Close\_rolling\_std, which have been calculated based on the stock's Close prices. The Close\_lag1 column contained the closing price from the previous day, while Close\_lag2 included the closing price from two days prior. The Close\_rolling\_mean represented the 5-day moving average of the closing prices, and the Close\_rolling\_std captured the 5-day rolling standard deviation, indicating price volatility over that period.

Rows with earlier dates (e.g., the first few rows) had 0.00 values for the lag and rolling columns due to insufficient prior data, especially for calculating the rolling statistics. The DataFrame also retained its previous columns: stock prices (Open, High, Low, Close), trading volume, sentiment scores, and related news headlines. The addition of the lagged and rolling features enriched the dataset by providing historical price context and trends, which can be useful for predictive analysis or modelling.



```

▶ # do correlation matrix exclude date with heatmap

import matplotlib.pyplot as plt
import seaborn as sns

# Select only numerical features
numerical_features = merged_df.select_dtypes(include=np.number)

# Calculate the correlation matrix
correlation_matrix = numerical_features.corr()

# Create a heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()

```

Figure 5.77 Process of Generalized a correlation matrix of numerical features from the merged\_df DataFrame

First, the numerical features were selected using `select_dtypes()` to exclude non-numerical columns like dates and headlines. The correlation matrix was then calculated using `.corr()`, which computed the pairwise correlation between all the numerical features, such as Open, High, Low, Close, Volume, sentiment\_score, and the newly created features like Close\_lag1, Close\_lag2, Close\_rolling\_mean, and Close\_rolling\_std.

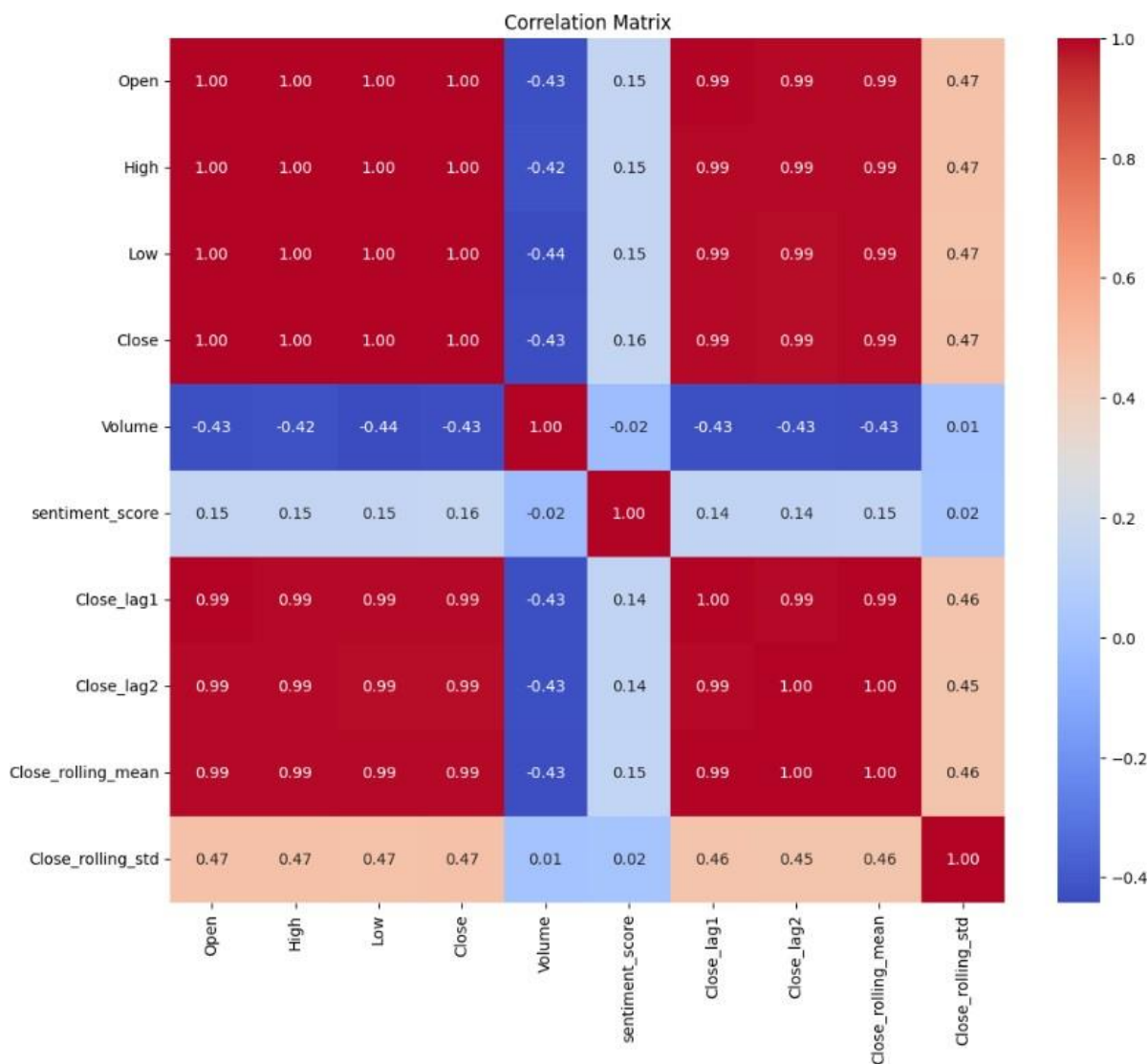


Figure 5.79 Visualization as a heatmap for numerical column in final merged\_df dataframe

The resulting heatmap visualized the strength of these relationships, with darker colors indicating stronger correlations. For example, the stock price variables (Open, High, Low, Close) had a near-perfect correlation of 1.0 with each other, indicating that they move almost identically. Meanwhile, Volume showed a moderate negative correlation with stock prices (-0.43), suggesting that higher trading volumes may correspond with lower stock prices. The sentiment\_score had a weak positive correlation (around 0.15) with stock prices, indicated that positive sentiment might slightly influence stock movements. The Close\_lag and Close\_rolling features also showed strong correlations with the closing price, indicated that past prices and rolling

statistics were good predictors of current prices. This heatmap helped in understanding the interrelationships between the variables in the dataset. This dataset offered a comprehensive view by merging stock performance metrics with sentiment analysis of news headlines, enabling the exploration of relationships between sentiment and stock price movements or trading volumes.

```
[ ] merged_df.to_csv('final_processed_data.csv', index=False)
```

Figure 5.80 Output the final processed dataframe for further use of modelling phase

This code exported the merged\_df DataFrame to a CSV file named final\_processed\_data.csv. The .to\_csv() function is used to write the DataFrame to a file, and the index=False parameter ensures that the index column (row numbers) is not included in the CSV file. Then, the csv file was ready to feed into the models for further modelling steps.

## 5.6 Modelling Phase

### 5.6.1 Model Data Preparation and Transformation

#### *Support Vector Machine*

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import numpy as np
from sklearn.model_selection import TimeSeriesSplit, cross_val_score
from sklearn.model_selection import RandomizedSearchCV
from transformers import BertTokenizer, BertModel
import torch
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import joblib

# Load the dataset
file_path = '/content/final_processed_data .csv'
data = pd.read_csv(file_path)

# Preprocess the numerical features
numerical_columns = [
    'Open', 'High', 'Low', 'Volume', 'Close_lag1', 'Close_lag2',
    'Close_rolling_mean', 'Close_rolling_std', 'sentiment_score']

data['Volume'] = data['Volume'].str.replace(',', '').astype(float)

# Drop rows with missing target values
data = data.dropna(subset=['Close'])

# Define target variable
target = 'Close'
y = data[target].values
X_numerical = data[numerical_columns].values

```

Figure 5.81 SVM Model Data Preparation I (Imported Libraries, Loaded dataset, and Preprocessed the numerical features, Dropped rows with missing target values and Defined Target variable.

This code imported a diverse set of libraries to handle data loading, preprocessing, model training, and model evaluation for performance on predicting stock market prices using Support Vector Regression (SVR). First, pandas was used for data manipulation, allowing the dataset to be loaded from a CSV file and processed efficiently. The sklearn.model\_selection module provided tools like train\_test\_split to split the dataset into training and testing subsets and TimeSeriesSplit for cross-validation, ensuring time series data is handled sequentially. RandomizedSearchCV from the same module was used for hyperparameter tuning to find the optimal configuration for the SVR model. The core algorithm, SVR, was imported from sklearn.svm, and StandardScaler from sklearn.preprocessing ensures that all numerical features are scaled, improving the performance of the SVR model. Additionally, make\_pipeline from sklearn.pipeline simplified the sequential steps, combining preprocessing and model training into a single process. Performance evaluation was handled with metrics like mean\_squared\_error, r2\_score, and mean\_absolute\_error from sklearn.metrics, which quantified the model's prediction accuracy. High, Low, Close), lagged features (Close\_lag1, Close\_lag2), rolling statistics (Close\_rolling\_mean, Close\_rolling\_std), and sentiment scores. Also, numpy provided efficient numerical operations, such as converting data into arrays, while matplotlib.pyplot and matplotlib.dates are used for visualizing actual vs predicted prices trend and formatting date-based plots on those visualisation for visualize them within date ranges. Finally, joblib allowed the trained model to be saved for future use, ensuring that the SVR model can be reloaded without retraining. Overall, this comprehensive set of libraries enables the entire workflow, from data preparation and model training to evaluation and visualization, facilitating the prediction of stock prices based on historical data and sentiment scores

After loading the dataset, a list of numerical columns was defined to extract the necessary features. The Volume column is cleaned again by removing commas and converting it to a float data type to ensure consistency in numerical representation.

Next, the code handled any missing data in the Close column by dropping rows where the target value is missing, ensuring the model has complete data for training. The Close column is then defined as the target variable, which was also known as the variable that the model will predict. This target data was extracted into the y variable. The input features, defined earlier in the numerical\_columns list, were extracted into the X\_numerical variable, which will be used as

inputs to the model. This setup ensured to prepare the data ready for training the SVR model, which will attempt to predict future closing stock prices based on historical price data, lagged features, rolling statistics, and sentiment analysis.

```
[ ] n = len(X_numerical) # Get the total number of data points in X_numerical
    train_size = int(n * 0.8) # Calculate 80% of the data as the training set size
    X_train, X_test = X_numerical[:train_size], X_numerical[train_size:] # Split the features into training and testing
    y_train, y_test = y[:train_size], y[train_size:] # Split the target variable into training and testing sets
```

Figure 5.82 SVM Model Data Preparation II (split the data into training and testing sets)

In this step, the numerical features data in `final_proceesed_data` was being split into training and testing sets, which was essential for building and evaluating the machine learning model. The length of the dataset was determined using  $n = \text{len}(X_{\text{numerical}})$ , where `X_numerical` represents the feature set of the data. The training set size is then calculated as 80% of the total number of samples, ensuring that the model is trained on the majority of the data while the remaining 20% is reserved for testing the model's performance. This is done with `train_size = int(n * 0.8)`. The features and target values were split accordingly: `X_train` and `y_train` contain the first 80% of the samples for training, while `X_test` and `y_test` contain the remaining 20% for testing. The training set will be used to fit the model, and the test set will be used to evaluate how well the model generalizes to unseen data, ensuring it was not overfitted to the training data.

This step is crucial as it ensures that the model is evaluated fairly, using a portion of the data that the model has not been exposed to during training. This split helps in assessing the model's ability to generalize well to future, unseen data, making it more reliable for real-world applications.

*Long Short-Term Memory (LSTM)*

```

Suggested code may be subject to a license | schwarzer79/stash | JuanBayon/Juan_Bayon
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import TimeSeriesSplit
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam, RMSprop, Nadam, SGD
from tensorflow.keras.regularizers import l2
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import numpy as np
import matplotlib.pyplot as plt
import kerastuner as kt

# Load the dataset
data = pd.read_csv('/content/final_processed_data.csv')

# Preprocess the numerical features
numerical_columns = ['Open', 'High', 'Low', 'Volume', 'Close_lag1', 'Close_lag2', 'Close_rolling_mean', 'Close_rolling_std', 'sentiment_score']
data['Volume'] = data['Volume'].replace({',': ''}, regex=True).astype(float)

# Drop rows with missing target values
data = data.dropna(subset=['Close'])

# Split data into features and target
X_numerical = data[numerical_columns].values
y = data['Close'].values
dates = data['Date']

# Scale the numerical features
scaler_X = MinMaxScaler(feature_range=(0, 1))
scaler_y = MinMaxScaler(feature_range=(0, 1))

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

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

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

```

Figure 5.83 LSTM Model Data Preparation (Imported Libraries, Loaded dataset, and Preprocessed the numerical features, Dropped rows with missing target values ,Splitted data into Feature and Target variable, Scaled the numerical features, Reshaped X to fit LSTM input shape and also split data into train and test data.)

The code prepares data for an LSTM (Long Short-Term Memory) model, following a similar process as the SVR (Support Vector Regression) model but also incorporating differences to handle time series data and the specific requirements of LSTM models. Like in the SVR model, the dataset was loaded using pandas, and key numerical features, such as Open, High, Low, Volume, sentiment\_score and other lagged and rolling features, are extracted for use in modeling. The Volume column was cleaned by removing commas and converting it to float. Missing values in the Close column are also handled by dropping rows that contain



missing target values. The feature set ( $X_{\text{numerical}}$ ) and target variable ( $y$ , representing Close values) are extracted similarly, preparing the data for model training.

In terms of libraries, there are several differences between the models. The LSTM model requires TensorFlow for deep learning functionalities, including the construction of the LSTM network. TensorFlow's submodules, such as keras, are imported to build and train the model using layers like LSTM, Dense, and Dropout. Additionally, various optimizers such as Adam and RMSprop were imported for model training, which are specific to deep learning models like LSTM. Furthermore, keras-tuner is imported to assist with hyperparameter tuning, which is commonly used to find the best architecture for LSTM networks. Besides, LSTM leveraged deep learning frameworks, highlighting a key distinction between traditional machine learning models and deep learning models designed to handle time series data.

Also, there were notable differences in how the data was processed for the LSTM model compared to SVR. LSTM used MinMaxScaler instead of StandardScaler for scaling the data. This scaled the features and target values to a range of  $[0,1]$ , which was essential for LSTM as it performed better when features are normalized between a small range. LSTMs also required input data to be in a 3D shape. This is achieved by reshaping the scaled feature data ( $X_{\text{scaled}}$ ) into this format ( $X_{\text{scaled}} = X_{\text{scaled}}.\text{reshape}(X_{\text{scaled}}.\text{shape}[0], X_{\text{scaled}}.\text{shape}[1], 1)$ ), reflecting time steps, so the feature data was reshaped to (samples, timesteps, features) format, allowing the model to learn sequential patterns in the data. Additionally, the dates column was extracted and splitted alongside the features and target, which is crucial for time series forecasting. The `train_test_split` function was used to divide the dataset into training and test sets, but the LSTM model splits the dates for easier temporal tracking of predictions.



*Convolution Neural Network (CNN)*

```

pip install optuna

Collecting optuna
  Downloading optuna-4.0.0-py3-none-any.whl.metadata (16 kB)
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.13.2-py3-none-any.whl.metadata (7.4 kB)
Collecting colorlog (from optuna)
  Downloading colorlog-6.8.2-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from optuna) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from optuna) (24.1)
Requirement already satisfied: sqlalchemy>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from optuna) (2.0.32)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from optuna) (4.66.5)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from optuna) (6.0.2)
Collecting Mako (from alembic>=1.5.0->optuna)
  Downloading Mako-1.3.5-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: typing-extensions>=4 in /usr/local/lib/python3.10/dist-packages (from alembic>=1.5.0->optuna) (4.6.0)
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-packages (from sqlalchemy>=1.3.0->optuna) (3.0.3)
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.10/dist-packages (from Mako->alembic>=1.5.0->optuna) (2.1.5)
Downloading optuna-4.0.0-py3-none-any.whl (362 kB)
 362.8/362.8 kB 8.7 MB/s eta 0:00:00
Downloading alembic-1.13.2-py3-none-any.whl (232 kB)
 233.0/233.0 kB 7.0 MB/s eta 0:00:00
Downloading colorlog-6.8.2-py3-none-any.whl (11 kB)
Downloading Mako-1.3.5-py3-none-any.whl (78 kB)
 78.6/78.6 kB 4.3 MB/s eta 0:00:00
Installing collected packages: Mako, colorlog, alembic, optuna
Successfully installed Mako-1.3.5 alembic-1.13.2 colorlog-6.8.2 optuna-4.0.0

```

Figure 5.84 CNN Model Data Preparation I (Install Optuna Library)

```

[ ] import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, KFold
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv1D, Dense, Input, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import matplotlib.pyplot as plt

# Step 1: Load and preprocess the data
file_path = '/content/final_processed_data.csv'
data = pd.read_csv(file_path)

# Replace commas in numerical columns and convert to float
numerical_columns = ['Open', 'High', 'Low', 'Volume', 'Close_lag1', 'Close_lag2', 'Close_rolling_mean', 'Close_rolling_std', 'sentiment_score']

# Drop rows with missing target values
data = data.dropna(subset=['Close'])

# Feature scaling
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X = data[numerical_columns].values
y = data['Close'].values.reshape(-1, 1)

X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y)

# Reshape X to fit CNN input shape (samples, timesteps, features)
X_scaled = X_scaled.reshape(X_scaled.shape[0], X_scaled.shape[1], 1)

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

```

Figure 5.85 CNN Model Data Preparation II (Imported Libraries, Loaded dataset, and Preprocessed the numerical features, Dropped rows with missing target values, Splitted data)

into Feature and Target variable, Scaled the x and y numerical features, Reshaped X to fit CNN input shape and also split data into train and test data.)

The provided codes demonstrated the data preparation process for a Convolutional Neural Network (CNN) model for time-series forecasting. Similar to the previous models, the data is loaded using pandas, where the dataset was read from a CSV file and numerical features such as Open, High, Low, Volume, and engineered features (Close\_lag1, Close\_rolling\_mean, etc.) are extracted. Like in LSTM, MinMaxScaler was used to scale these numerical features to a range of [0,1], ensuring that the input data was normalized for training. Missing values in the target column (Close) were handled by dropping incomplete rows to maintain data integrity during training. A key difference between CNN and the other models was the input data reshaping process. CNN required the data to be reshaped into a 3D format (samples, timesteps, features), which is done to allow convolutional layers to capture patterns in the time-series data. This is similar to the reshaping required in LSTM but differs significantly from SVR, which does not require reshaping and treats each data point independently. Also, the installation process of optuna, a popular hyperparameter optimization library often used for fine-tuning deep learning models like CNNs (Convolutional Neural Networks). Optuna helped automate the process of finding the best hyperparameters for a CNN by evaluating different configurations efficiently, allowing for improved performance without manually adjusting the parameters.

Additionally, the CNN model involved specific libraries like TensorFlow and Keras, which are used to construct and train the CNN architecture. These libraries provided layers such as Conv1D (for performing convolutional operations over time-series data), Dense, Dropout, and optimizers like Adam. This contrasts with SVR, which relies on scikit-learn and traditional machine learning techniques. While both CNN and LSTM models used deep learning frameworks, CNN's architecture focused on convolutional operations to extract spatial features, whereas LSTM focused on capturing temporal dependencies through memory cells. Overall, while some preprocessing steps were shared across models, CNN's data preparation was tailored for its specialized architecture, involving data reshaping and deep learning-specific libraries to leverage convolution for time-series forecasting.

## Recurrent neural network (RNN) with Attention Mechanism

```
[ ] pip install optuna
Collecting optuna
  Downloading optuna-4.0.0-py3-none-any.whl.metadata (16 kB)
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.13.2-py3-none-any.whl.metadata (7.4 kB)
Collecting colorlog (from optuna)
  Downloading colorlog-6.8.2-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from optuna) (1.26.4)
Requirement already satisfied: packaging>20.0 in /usr/local/lib/python3.10/dist-packages (from optuna) (24.1)
Requirement already satisfied: sqlalchemy>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from optuna) (2.0.32)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from optuna) (4.66.5)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from optuna) (6.0.2)
Collecting Mako (from alembic>=1.5.0->optuna)
  Downloading Mako-1.3.5-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: typing-extensions>=4 in /usr/local/lib/python3.10/dist-packages (from alembic>=1.5.0->optuna) (4.12.2)
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-packages (from sqlalchemy>=1.3.0->optuna) (3.0.3)
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.10/dist-packages (from Mako->alembic>=1.5.0->optuna) (2.1.5)
Downloading optuna-4.0.0-py3-none-any.whl (362 kB)
----- 362.0/362.8 kB 8.7 MB/s eta 0:00:00
Downloading alembic-1.13.2-py3-none-any.whl (232 kB)
----- 233.0/233.0 kB 7.0 MB/s eta 0:00:00
Downloading colorlog-6.8.2-py3-none-any.whl (11 kB)
Downloading Mako-1.3.5-py3-none-any.whl (78 kB)
----- 78.6/78.6 kB 4.3 MB/s eta 0:00:00
Installing collected packages: Mako, colorlog, alembic, optuna
Successfully installed Mako-1.3.5 alembic-1.13.2 colorlog-6.8.2 optuna-4.0.0
```

Figure 5.86 RNN with Attention Mechanism Model Data Preparation I (Install Optuna Library)

```
import pandas as pd
import numpy as np
from sklearn.model_selection import TimeSeriesSplit
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout, Input, Multiply, Permute, Lambda
from tensorflow.keras.optimizers import Adam # Import Adam
import tensorflow.keras.backend as K
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import optuna
from tensorflow.keras.optimizers import Adam, RMSprop, Nadam, SGD

# Load your data
data = pd.read_csv('/content/final_processed_data.csv')

data['Volume'] = data['Volume'].replace({' ': ''}, regex=True).astype(float)

# Define features and target
features = ['Open', 'High', 'Low', 'Volume', 'sentiment_score', 'Close_lag1', 'Close_lag2', 'Close_rolling_mean', 'Close_rolling_std']
target = 'Close'

X = data[features]
y = data[target]

# Feature scaling
scaler_X = StandardScaler()
scaler_y = StandardScaler()

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

# Reshape X for RNN
X_scaled = X_scaled.reshape(X_scaled.shape[0], X_scaled.shape[1], 1)

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

Figure 5.87 RNN with Attention Mechanism Model Data Preparation II (Imported Libraries, Loaded dataset, and Preprocessed the numerical features, Defined Features (X) and Y(target) , Scaled and transform the x and y numerical features, Reshaped X to fit RNN with Attention Mechanism input shape and also split data into train and test data.)

In these code snippets, the data preparation for an RNN model with an attention mechanism builded on the same foundational steps used for other deep learning models such as CNNs or LSTMs. The feature scaling process applied the StandardScaler from sklearn.preprocessing, ensuring that numerical features like Open, High, Low, Volume, and lagged features like Close\_lag1 and Close\_lag2 are standardized. This scaling were crucial for improving model performance and convergence during training. The target variable (Close) was also scaled to maintain consistency with the input features. After scaling, the input data was reshaped into a 3D structure, necessary for RNNs to capture sequential dependencies over time steps, formatted as (samples, timesteps, features). This was a common practice for RNNs and LSTMs but differs from CNNs, which process data in spatial dimensions.

A key difference in this model preparation is the addition of the attention mechanism, which helps the model weigh certain time steps more than others based on their relevance. For this purpose, libraries such as tensorflow.keras.layers.Input, SimpleRNN, Dense, Dropout, Permute, and Lambda are utilized. These layers allowed for greater flexibility in building custom models that can incorporate attention, unlike CNNs, which rely on Conv1D layers to capture spatial patterns, or standard RNNs that lack the capability to focus on specific time steps. The optimizer Adam was imported to fine-tune the model's learning process, along with additional optimizers like RMSProp and Nadam for flexibility during experimentation. Finally, Optuna was introduced to handle hyperparameter optimization, making the RNN with attention a more sophisticated model for time series prediction by selectively focusing on important input data.

## 5.6.2 Model Defining, Building And Compiling

### *Support Vector Machine (SVM)*

```
# Create an SVM model with a pipeline
svm_pipeline = make_pipeline(StandardScaler(), SVR(kernel='rbf',gamma='auto',C=10,epsilon=0.2))
```

Figure 5.88 SVM Model defining and building

This code snippet established an SVM (Support Vector Machine) model pipeline with integrated scaling, streamlining the machine learning process by ensuring the features are appropriately scaled before model training. The pipeline was built using the `make_pipeline()` function, which combined two essential steps: feature normalization using `StandardScaler()` and model creation with `SVR()` (Support Vector Regression). The `StandardScaler()` ensured that all features had a mean of 0 and a standard deviation of 1, effectively normalizing the data. This step is critical for SVM models, which are sensitive to the scale of input features and perform better when the data is standardized.

The SVR model was configured with several specific hyperparameters. It employed the RBF (Radial Basis Function) kernel, which is widely used in SVM due to its ability to handle nonlinear relationships by transforming the data into a higher-dimensional space. The `gamma='auto'` parameter defined the influence of individual training examples on the decision boundary, controlling the flexibility of the model to adjust to data points. A higher gamma leads to a more flexible model that fits the data closely, while a lower gamma makes the model more rigid. The `C=10` parameter regulated the trade-off between the model's complexity and the margin of error. A higher C allowed the model to pay more attention to minimizing the error on the training data, but it could lead to overfitting if not tuned properly. Lastly, `epsilon=0.2` specified the epsilon-tube, a margin within which predictions were not penalized for being off-target. This helped the model tolerate small errors while focusing on capturing significant trends.

*Long Short-Term Memory (LSTM)*

```

# Define the LSTM model
def create_model(units=50, dropout_rate=0.2, optimizer='adam'):
    model = Sequential()

    model.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], 1), kernel_regularizer=l2(0.001)))
    model.add(Dropout(0.2))

    model.add(LSTM(50, return_sequences=False, kernel_regularizer=l2(0.001)))
    model.add(Dropout(0.2))

    model.add(Dense(1))

    model.compile(optimizer=optimizer, loss='mean_squared_error')
    return model

```

Figure 5.89 LSTM Model defining and building

This code defined an LSTM model using TensorFlow and Keras, specifically designed for time series prediction. The function `create_model` allowed for flexible configuration of units, dropout rate, and optimizer. The model began with the Keras `Sequential()` API, which enabled easy stacking of layers. The first LSTM layer consisted of 50 units, with `return_sequences=True` to output the entire sequence for further processing by the subsequent LSTM layer. This layer was fed input with a shape defined by the training data, and L2 regularization (with a penalty of 0.001) was applied to avoid overfitting. Following the first LSTM, a dropout layer was included to randomly drop 20% of the neurons during training, further regularizing the model and enhancing its generalization.

The second LSTM layer, also with 50 units, had `return_sequences=False` since it was the final recurrent layer and only the last output was required. Like the first LSTM, it was regularized with L2. Another dropout layer with a 20% rate was applied after this LSTM to prevent overfitting. Finally, the model ended with a dense layer containing a single neuron, which predicted the target variable, in this case, the stock's "Close" price. The model was compiled using the `adam` optimizer, which is known for its adaptive learning rate and efficiency. The loss function was set to `mean_squared_error`, commonly used for regression tasks like stock market prediction to minimize the squared difference between predicted and actual values.



*Convolution Neural Network (CNN)*

```
[ ] # Build the CNN model
def build_cnn_model(input_shape):
    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=3, padding='same', activation='relu', input_shape=input_shape))
    model.add(Conv1D(filters=64, kernel_size=3, padding='same', activation='relu'))
    model.add(Flatten())
    model.add(Dense(50, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1))
    return model
```

Figure 5.90 CNN Model building

```
▶ # Compile the model
cnn_model = build_cnn_model((X_train.shape[1], X_train.shape[2]))
cnn_model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')

# Display the model summary
cnn_model.summary()
```

Figure 5.91 CNN model Compiling and function to display model summary

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarn:
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 9, 64)	256
conv1d_1 (Conv1D)	(None, 9, 64)	12,352
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 50)	28,850
dropout (Dropout)	(None, 50)	0
dense_1 (Dense)	(None, 1)	51

```
Total params: 41,509 (162.14 KB)
Trainable params: 41,509 (162.14 KB)
Non-trainable params: 0 (0.00 B)
```

Figure 5.92 Output of CNN model's model summary

These code snippets above showed CNN model was built to handle time series forecasting tasks, specifically designed to capture sequential patterns in the data. The function `build_cnn_model` constructed the model using the Keras Sequential API, allowing for the layer-

by-layer addition of neural network components. The architecture started with two Conv1D layers, both using 64 filters, which were small sliding windows that scanned through the time series data to detect local patterns. These layers used a kernel size of 3, meaning each filter looked at three consecutive time steps at a time to extract meaningful patterns. The padding was set to 'same', ensuring that the output dimensions remained the same as the input dimensions after convolution, preserving the size of the data for deeper layers.

The activation function in both convolutional layers was relu (Rectified Linear Unit), which introduced non-linearity and helped the model learn complex representations by allowing only positive values to pass through. The Flatten layer was crucial because it converted the 3D output from the convolutional layers into a 2D array, making it suitable for the fully connected layers that followed.

After flattening, a Dense layer with 50 neurons was added, which served as a fully connected layer where each neuron was connected to all the outputs from the previous layer. The relu activation continued to ensure non-linearity in the model. A Dropout layer with a rate of 0.2 was applied to mitigate overfitting by randomly setting 20% of the neurons to zero during each training iteration. This regularization technique helped prevent the model from memorizing the training data and encouraged generalization.

The final layer was a Dense layer with a single output neuron because the task at hand was regression, and the model was expected to predict a single continuous value (the stock price or another target). The optimizer used for compiling the model was Adam, a widely adopted optimizer known for its efficient and adaptive learning rate. The learning rate was set at 0.001, which allowed the model to make gradual updates to weights, ensuring stability in learning. The loss function, `mean_squared_error`, was chosen to measure the squared difference between the predicted and actual values, penalizing large errors.

Finally, the model's summary was displayed, providing a detailed overview of each layer, its output shape, and the number of parameters (trainable weights). The two Conv1D layers contributed the most parameters because they contained several filters. The overall model had 41,509 trainable parameters, indicating the complexity of the neural network. This model was particularly useful for sequential data because the Conv1D layers captured temporal



patterns effectively by using sliding filters across time steps, making it ideal for time series forecasting tasks.

### *Recurrent neural network (RNN) with Attention Mechanism*

```

# Custom attention layer
def attention_3d_block(inputs):
    input_dim = int(inputs.shape[2])
    a = Permute((2, 1))(inputs)
    a = Dense(X_train.shape[1], activation='softmax')(a)
    a = Permute((2, 1), name='attention_vec')(a)
    output_attention_mul = Multiply()(inputs, a)
    return output_attention_mul

# Function to create RNN with Attention model
def create_rnn_attention_model(units=64, dropout_rate=0.2, optimizer=Adam()): # Instantiate Adam optimizer
    input_layer = Input(shape=(X_train.shape[1], X_train.shape[2]))
    rnn_layer = SimpleRNN(units, return_sequences=True)(input_layer)
    attention_layer = attention_3d_block(rnn_layer)
    attention_output = Dense(1)(attention_layer)
    model = Model(inputs=[input_layer], outputs=[attention_output])

    model.compile(optimizer=optimizer, loss='mean_squared_error')
    return model

# Create and train the initial model
initial_model = create_rnn_attention_model()

```

Figure 5.92 Building the attention Layer, function to create the RNN with attention model and create the initial model.

The code defines an RNN model with an attention mechanism, introducing a more advanced structure that enables the network to focus on specific time steps within the sequence data. The Attention Layer (`attention_3d_block`) was designed to enhance the RNN's ability to focus on specific time steps in the sequence. The attention mechanism was applied across the time dimension, with the input dimension determined by the shape of the input sequence, specifically the number of features or units in the input data, denoted as `X_train.shape[2]`. In this case, `X_train.shape[2]` refers to the number of features in each time step of the sequence. For example, if the dataset contains stock data with features such as 'Open', 'Close', 'Volume', etc., then `X_train.shape[2]` would represent the number of such features.

The `Permute` layer was used to change the input shape to (2, 1). Here, the numbers (2, 1) refer to the axes that were swapped—axis 2 (features) and axis 1 (time steps). This permutation allows the attention mechanism to operate across the time steps rather than across the features. After calculating the attention scores, the input sequence was permuted back to its original shape using another `Permute` layer.

A Dense layer was then applied with an output size equal to the number of time steps in the input sequence, which is represented by `X_train.shape[1]`. The value `X_train.shape[1]` refers to the length of the sequence (i.e., the number of time steps in each sequence). For instance, if analyzing stock prices over 30 days, `X_train.shape[1]` would be 30. The Dense layer's output size ensures that the attention mechanism can assign different importance to each time step. The softmax activation function was used to ensure that the attention scores across all time steps summed to 1, making it possible for the model to focus on the most relevant parts of the sequence.

Once the attention scores were computed, the input sequence was multiplied by these scores using the Multiply layer. This multiplication applied the attention mechanism, effectively weighting each time step by its importance, as determined by the attention scores. This allowed the model to concentrate on the most relevant time steps when making predictions.

In the Function to Create the RNN with Attention (`create_rnn_attention_model`), the core layer was a SimpleRNN with 64 units (`units=64`). This means that the RNN output for each time step had a dimensionality of 64, meaning it output 64 values at each time step. The `return_sequences=True` parameter was essential because it made the RNN return the full sequence of outputs across all time steps (as opposed to just the output from the final time step), which was necessary for the attention mechanism to operate on the entire sequence. Without this, the attention layer wouldn't have the full sequence to work with.

A Dropout layer with a rate of 0.2 (20%) was added after the RNN layer to reduce overfitting. This layer randomly set 20% of the input units to zero during each training iteration, helping the model generalize better to unseen data. The output from the RNN layer was then passed to the custom `attention_3d_block` function, where the attention mechanism was applied to the sequence, focusing the model on the most important time steps. After applying attention, a Dense layer with a single unit was added to generate the final prediction. This single output is typical for regression tasks, where the goal is to predict a single continuous value (such as a stock price).

Finally, in the Create and Train the Initial Model section, the model was initialized by calling the `create_rnn_attention_model` function. The model was compiled using the Adam

optimizer, which is known for its efficient handling of large datasets and sparse gradients. The `mean_squared_error` loss function was used, which is appropriate for regression tasks, where the model is trained to minimize the difference between predicted and actual values.

### 5.6.3 Model Training and Training Performance Evaluation

#### *Support Vector Machine (SVM)*

```
# Train the model
svm_pipeline.fit(X_train, y_train)

# Evaluate on the training set
y_train_pred = svm_pipeline.predict(X_train)

# Calculate performance metrics on the training set
mse_train = mean_squared_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)
rmse_train = np.sqrt(mse_train)
mae_train = mean_absolute_error(y_train, y_train_pred)

# Print training performance metrics
print(f"Training Set Performance:")
print(f"Mean Squared Error (MSE): {mse_train}")
print(f"R-squared (R2): {r2_train}")
print(f"Root Mean Squared Error (RMSE): {rmse_train}")
print(f"Mean Absolute Error (MAE): {mae_train}")
```

Figure 5.93 SVM Model training and Training Performance Evaluation

The provided code above implemented the training and evaluation process of an SVM model using various performance metrics to assess the accuracy and fit on the training dataset. Initially, the `svm_pipeline.fit(X_train, y_train)` command fit the SVM model to the training data, where the model learned patterns in the features (`X_train`) to predict the target values (`y_train`). After training, the model was used to make predictions on the training data with the `svm_pipeline.predict(X_train)` function, which stored the predicted target values (`y_train_pred`).

To evaluate the model's performance, several metrics were calculated. The Mean Squared Error (MSE) was computed using `mean_squared_error(y_train, y_train_pred)`, which measured the average squared difference between the predicted values and the actual target values, giving an idea of how close the predictions were. Next, R-squared ( $R^2$ ) was calculated using `r2_score(y_train, y_train_pred)`, indicating how well the predictions captured the variance in the actual target values, with values closer to 1 showing a better fit. Additionally, Root Mean Squared Error (RMSE), calculated as the square root of MSE, offered a more interpretable error magnitude in the same unit as the target. Lastly, Mean Absolute Error (MAE) was derived using `mean_absolute_error(y_train, y_train_pred)`, providing the average of the absolute differences between predicted and actual values. These metrics were printed out to help assess the model's performance, giving insights into the error magnitude and how well the model fit the training data.

### Long Short-Term Memory (LSTM)

```
# Train the model
model = create_model()
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), verbose=1)

/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an "input_shape"/"input_dim" argument to a l
super().__init__(**kwargs)
Epoch 1/50
17/17 ----- 7s 114ms/step - loss: 0.2733 - val_loss: 0.1064
Epoch 2/50
17/17 ----- 0s 8ms/step - loss: 0.0934 - val_loss: 0.0703
Epoch 3/50
17/17 ----- 0s 8ms/step - loss: 0.0680 - val_loss: 0.0520
Epoch 4/50
17/17 ----- 0s 7ms/step - loss: 0.0507 - val_loss: 0.0380
Epoch 5/50
17/17 ----- 0s 8ms/step - loss: 0.0409 - val_loss: 0.0318
Epoch 6/50
17/17 ----- 0s 8ms/step - loss: 0.0330 - val_loss: 0.0269
Epoch 7/50
17/17 ----- 0s 7ms/step - loss: 0.0292 - val_loss: 0.0232
Epoch 8/50
17/17 ----- 0s 8ms/step - loss: 0.0255 - val_loss: 0.0205
Epoch 9/50
17/17 ----- 0s 7ms/step - loss: 0.0228 - val_loss: 0.0183
Epoch 10/50
17/17 ----- 0s 8ms/step - loss: 0.0201 - val_loss: 0.0166
Epoch 11/50
17/17 ----- 0s 9ms/step - loss: 0.0200 - val_loss: 0.0152
Epoch 12/50
17/17 ----- 0s 7ms/step - loss: 0.0178 - val_loss: 0.0150
Epoch 13/50
17/17 ----- 0s 7ms/step - loss: 0.0167 - val_loss: 0.0133
Epoch 14/50
17/17 ----- 0s 8ms/step - loss: 0.0153 - val_loss: 0.0123
Epoch 15/50
17/17 ----- 0s 7ms/step - loss: 0.0144 - val_loss: 0.0115
Epoch 16/50
17/17 ----- 0s 8ms/step - loss: 0.0139 - val_loss: 0.0110
Epoch 17/50
17/17 ----- 0s 8ms/step - loss: 0.0130 - val_loss: 0.0102
Epoch 18/50
17/17 ----- 0s 8ms/step - loss: 0.0124 - val_loss: 0.0096
Epoch 19/50
17/17 ----- 0s 8ms/step - loss: 0.0125 - val_loss: 0.0092
Epoch 20/50
17/17 ----- 0s 8ms/step - loss: 0.0119 - val_loss: 0.0086
Epoch 21/50
17/17 ----- 0s 9ms/step - loss: 0.0110 - val_loss: 0.0083
Epoch 22/50
17/17 ----- 0s 7ms/step - loss: 0.0104 - val_loss: 0.0078
Epoch 23/50
17/17 ----- 0s 8ms/step - loss: 0.0093 - val_loss: 0.0079
```

Figure 5.94 LSTM Model Training

```

▶ # Evaluate the initial model on the training set
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import numpy as np

# Predict on the training set
y_train_pred = model.predict(X_train)
y_train_pred = scaler_y.inverse_transform(y_train_pred)
y_train_original = scaler_y.inverse_transform(y_train)

# Calculate metrics for the training set
mse_train = mean_squared_error(y_train_original, y_train_pred)
rmse_train = np.sqrt(mse_train)
mae_train = mean_absolute_error(y_train_original, y_train_pred)
r2_train = r2_score(y_train_original, y_train_pred)

# Print evaluation metrics for the training set
print(f"Training Set Performance with LSTM:")
print(f"Mean Squared Error (MSE): {mse_train}")
print(f"Root Mean Squared Error (RMSE): {rmse_train}")
print(f"Mean Absolute Error (MAE): {mae_train}")
print(f"R-squared (R2): {r2_train}")

```

Figure 5.95 LSTM Model Training Performance Evaluation

The model training process for the LSTM began with the `create_model()` function, which defined the LSTM architecture. The model was trained using the `fit()` method with 50 epochs and a batch size of 32. The training process included validation data (`X_test` and `y_test`), allowing the model to be monitored for both training and validation performance during each epoch. As the model iterated through the epochs, the loss and validation loss were printed, reflecting how the model was adjusting its weights to reduce errors and improve its predictions over time. The `verbose=1` parameter ensured that detailed progress was displayed for each epoch.

After training the LSTM model, predictions on the training dataset were generated using the `predict()` method. To revert the scaled predictions and target values (`y_train`) back to their original values, the inverse transform of the scaler was applied. The model's performance on the training data was then evaluated using regression metrics: mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), and R-squared ( $R^2$ ). These metrics helped assess how well the model captured the underlying patterns in the training data, providing insight into the error magnitude and the proportion of variance explained by the model. This process effectively closed the loop on training by determining how well the LSTM had learned from the data.

## Convolution Neural Network (CNN)

```

▶ # Train the CNN model
history = cnn_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), verbose=1)

Epoch 1/50
17/17 ━━━━━━━━━━━ 6s 44ms/step - loss: 0.1232 - val_loss: 0.0034
Epoch 2/50
17/17 ━━━━━━━━━━━ 0s 9ms/step - loss: 0.0121 - val_loss: 7.9512e-04
Epoch 3/50
17/17 ━━━━━━━━━━━ 0s 15ms/step - loss: 0.0066 - val_loss: 6.2112e-04
Epoch 4/50
17/17 ━━━━━━━━━━━ 0s 10ms/step - loss: 0.0069 - val_loss: 7.8256e-04
Epoch 5/50
17/17 ━━━━━━━━━━━ 0s 10ms/step - loss: 0.0072 - val_loss: 4.1685e-04
Epoch 6/50
17/17 ━━━━━━━━━━━ 0s 8ms/step - loss: 0.0073 - val_loss: 0.0012
Epoch 7/50
17/17 ━━━━━━━━━━━ 0s 16ms/step - loss: 0.0066 - val_loss: 7.1130e-04
Epoch 8/50
17/17 ━━━━━━━━━━━ 0s 12ms/step - loss: 0.0078 - val_loss: 0.0011
Epoch 9/50
17/17 ━━━━━━━━━━━ 0s 18ms/step - loss: 0.0067 - val_loss: 2.8314e-04
Epoch 10/50
17/17 ━━━━━━━━━━━ 0s 4ms/step - loss: 0.0064 - val_loss: 2.4065e-04
Epoch 11/50
17/17 ━━━━━━━━━━━ 0s 4ms/step - loss: 0.0052 - val_loss: 3.8172e-04
Epoch 12/50
17/17 ━━━━━━━━━━━ 0s 5ms/step - loss: 0.0057 - val_loss: 8.0095e-04
Epoch 13/50
17/17 ━━━━━━━━━━━ 0s 5ms/step - loss: 0.0058 - val_loss: 2.8413e-04
Epoch 14/50
17/17 ━━━━━━━━━━━ 0s 4ms/step - loss: 0.0047 - val_loss: 0.0010
Epoch 15/50
17/17 ━━━━━━━━━━━ 0s 5ms/step - loss: 0.0063 - val_loss: 2.9273e-04
Epoch 16/50
17/17 ━━━━━━━━━━━ 0s 5ms/step - loss: 0.0042 - val_loss: 6.3125e-04
Epoch 17/50
17/17 ━━━━━━━━━━━ 0s 4ms/step - loss: 0.0059 - val_loss: 4.2165e-04
Epoch 18/50

```

Figure 5.96 CNN Model Training

```

▶ # Evaluate on training set
y_train_pred = cnn_model.predict(X_train)
y_train_pred = scaler_y.inverse_transform(y_train_pred)
y_train_original = scaler_y.inverse_transform(y_train)

# Calculate performance metrics on the training set
mse_train = mean_squared_error(y_train_original, y_train_pred)
r2_train = r2_score(y_train_original, y_train_pred)
rmse_train = np.sqrt(mse_train)
mae_train = mean_absolute_error(y_train_original, y_train_pred)

# Print training performance metrics
print(f"Training Set Performance:")
print(f"Mean Squared Error (MSE): {mse_train}")
print(f"R-squared (R²): {r2_train}")
print(f"Root Mean Squared Error (RMSE): {rmse_train}")
print(f"Mean Absolute Error (MAE): {mae_train}")

```

Figure 5.97 CNN Model Training Performance Evaluation

In this code, the CNN model training and evaluation process was executed. The `cnn_model.fit()` function was used to train the model with the training dataset (`X_train`, `y_train`). The model was trained for 50 epochs with a batch size of 32, while validation was performed using the test dataset (`X_test`, `y_test`). Each epoch's progress was displayed with the training loss and validation loss, indicating how well the model learned during training.

After training, the model was evaluated on the training set. Predictions were made using `cnn_model.predict(X_train)`, and these predictions were inverse transformed using the scaler to return them to their original scale. The code then calculated various performance metrics on the training set, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared ( $R^2$ ). These metrics provided insight into the model's performance, specifically how closely the predictions matched the actual values. The final step printed these metrics, allowing for a detailed evaluation of how well the CNN model performed on the training data.



*Recurrent neural network (RNN) with Attention Mechanism*

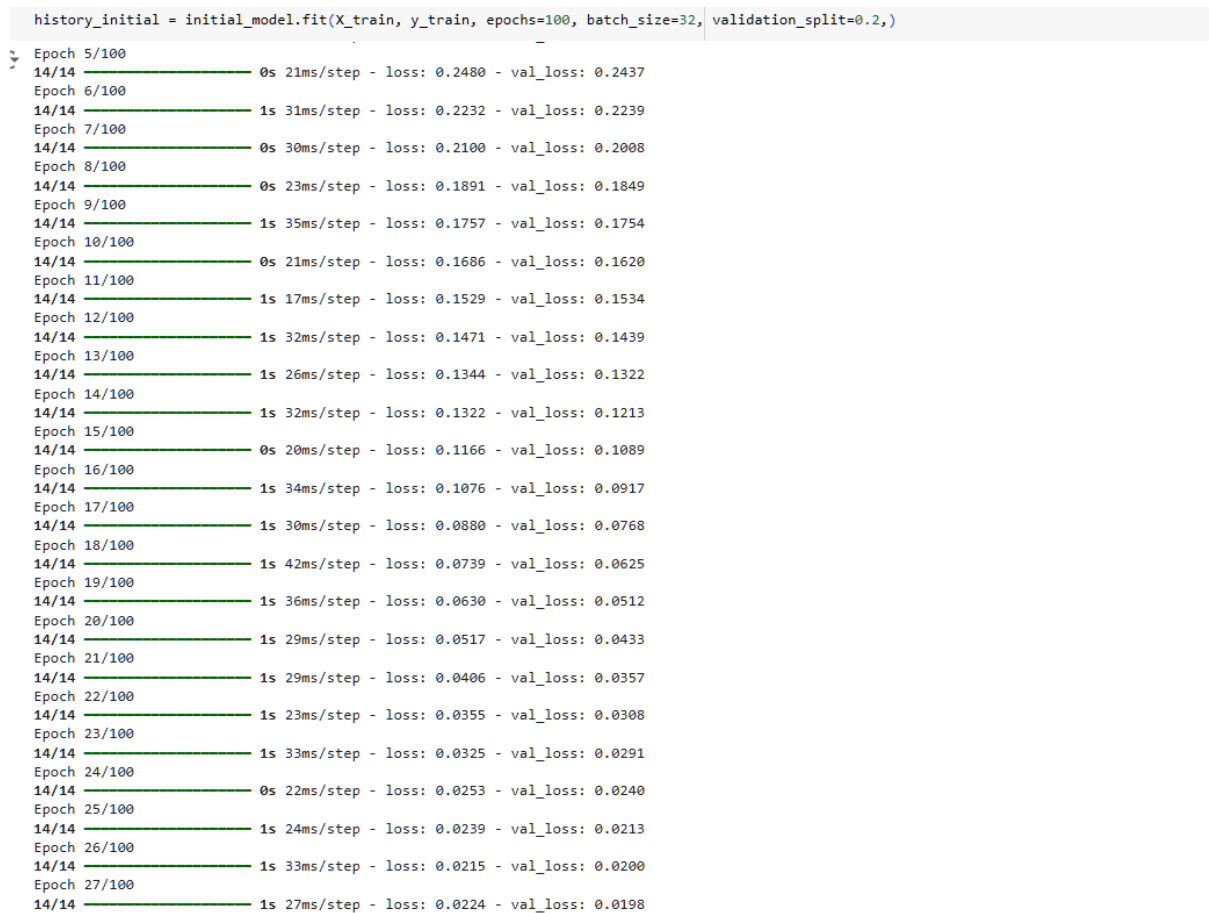


Figure 5.98 RNN with Attention Mechanism Model Training



```

▶ # Predict on the training set
y_train_pred = initial_model.predict(X_train)

# Extract the last time step of the sequence, as we're predicting the next step
y_train_pred = y_train_pred[:, -1] # Extract the last time step prediction

# Inverse transform the predictions and original values
y_train_pred = scaler_y.inverse_transform(y_train_pred.reshape(-1, 1))
y_train_original = scaler_y.inverse_transform(y_train)

# Ensure both arrays have matching shapes
print(f"Shape of y_train_original: {y_train_original.shape}")
print(f"Shape of y_train_pred: {y_train_pred.shape}")

# Calculate performance metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import math

mae = mean_absolute_error(y_train_original, y_train_pred)
mse = mean_squared_error(y_train_original, y_train_pred)
rmse = math.sqrt(mse)
r2 = r2_score(y_train_original, y_train_pred)

print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
print(f"R2: {r2}")

```

 17/17 ————— 0s 13ms/step  
Shape of y\_train\_original: (541, 1)  
Shape of y\_train\_pred: (541, 1)

Figure 5.99 RNN with Attention Mechanism Model Training Performance Evaluation

The model training and evaluation process for the RNN with attention mechanism were carried out using 100 epochs, with a batch size of 32 and a validation split of 0.2, meaning 80% of the data was used for training and 20% for validation. During training, the fit() function was used to optimize the model by minimizing the mean squared error (MSE) loss for both the training and validation datasets. The output for each epoch showed a progressive reduction in both training and validation loss, indicating that the model was learning to predict the target values effectively without overfitting. This decreasing trend in the loss functions suggested that the model was improving its accuracy as training progressed.

After the training phase, the model's performance on the training data was evaluated. Predictions were made on the training dataset using the predict() function, specifically targeting

the last time step of the sequence since the task was to predict the next value in the sequence. The predicted values were then inverse transformed, along with the original training data, using the scaler that had been applied earlier during preprocessing to normalize the data. This step restored the predictions and actual values to their original scale, enabling proper evaluation. It was important to ensure that both the predicted and actual arrays had matching shapes for evaluation, with the output shape of the predictions and original data being (541, 1), meaning 541 data points for each variable.

The model's performance was assessed using several key metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared ( $R^2$ ). MAE provided an average measure of prediction error, while MSE emphasized larger errors by squaring the differences. RMSE, being the square root of MSE, offered an interpretable error in the same scale as the data, and  $R^2$  indicated the correlation between predicted and actual values, with values closer to 1 showing strong predictive accuracy. Overall, these metrics, along with the shape of the outputs, highlighted the RNN with attention mechanism's ability to effectively capture temporal dependencies in the sequence, providing a solid performance on the training data.

#### 5.6.4 Model Cross Validation Evaluation

##### *Support Vector Machine (SVM)*

```
# Create a TimeSeriesSplit object
tscv = TimeSeriesSplit(n_splits=5)

# Perform cross-validation
cv_scores = cross_val_score(svm_pipeline, X_numerical, y, cv=tscv, scoring='neg_mean_squared_error')

# Calculate the average MSE
avg_mse = -np.mean(cv_scores)
avg_rmse = np.sqrt(avg_mse)

# Print the average MSE
print(f"Average MSE with TimeSeriesSplit: {avg_mse}")
print(f"Average RMSE with TimeSeriesSplit: {avg_rmse}")
```

Figure 5.100 SVM Model TimeSeriesSplit Cross Validation

This code snippet implemented the cross-validation process for evaluating the performance of the SVM model using a time series split. First, a `TimeSeriesSplit` object was created with 5 splits (`n_splits=5`), which ensured that the data was split in a way that respected the temporal order of the dataset, as is necessary in time series forecasting tasks. The cross-validation was then performed using the `cross_val_score()` function, applying the SVM pipeline to the features (`X_numerical`) and target variable (`y`). The scoring parameter was set to `neg_mean_squared_error`, which means the function calculated the negative of the Mean Squared Error (MSE) for each fold of the cross-validation. This is done because cross-validation functions in `scikit-learn` are set up to maximize scores, so the negative sign allows for minimization of MSE.

Once the cross-validation was completed, the average MSE across all the cross-validation folds was calculated using `np.mean()` on the `cv_scores`. Since the MSE was negative, it was multiplied by `-1` to restore the correct MSE values. The Root Mean Squared Error (RMSE) was also computed using `np.sqrt()` on the average MSE to provide a more interpretable metric. Both the average MSE and RMSE were printed out for further analysis. These metrics provided insight into the model's performance across different training and test splits, offering a more robust evaluation than a single training-validation split would provide. The MSE and RMSE values reflect the model's average prediction error during cross-validation.

### Long Short-Term Memory (LSTM)

```

# Time Series Cross-Validation
tscv = TimeSeriesSplit(n_splits=5)
cv_mse_scores = []

for train_index, val_index in tscv.split(X_train):
    X_train_cv, X_val_cv = X_train[train_index], X_train[val_index]
    y_train_cv, y_val_cv = y_train[train_index], y_train[val_index]

    # Define and compile the LSTM model
    model_cv = Sequential()
    model_cv.add(LSTM(50, return_sequences=True, input_shape=(X_train_cv.shape[1], 1)))
    model_cv.add(Dropout(0.2))
    model_cv.add(LSTM(50, return_sequences=False))
    model_cv.add(Dropout(0.2))
    model_cv.add(Dense(1))

    model_cv.compile(optimizer='adam', loss='mean_squared_error')

    # Train the model
    model_cv.fit(X_train_cv, y_train_cv, epochs=50, batch_size=32, verbose=0)

    # Validate the model
    y_val_pred_cv = model_cv.predict(X_val_cv)
    mse_cv = mean_squared_error(y_val_cv, y_val_pred_cv)
    cv_mse_scores.append(mse_cv)

average_cv_mse = np.mean(cv_mse_scores)
print(f'Average MSE across 5-fold TimeSeriesSplit cross-validation: {average_cv_mse}')

```

## Figure 5.101 LSTM Model TimeSeriesSplit Cross Validation

This code performed a time series cross-validation (CV) for an LSTM model using a TimeSeriesSplit object with 5 splits (`n_splits=5`). The TimeSeriesSplit ensures that each fold maintains the temporal order of the data, which is crucial for time series modeling.

Within the for loop, the training and validation indices (`train_index` and `val_index`) were generated, and corresponding training and validation sets (`X_train_cv`, `X_val_cv`, `y_train_cv`, `y_val_cv`) were defined for each fold. For each fold, the LSTM model (`model_cv`) was defined and compiled. The model had two LSTM layers, both with 50 units. The first LSTM layer was configured with `return_sequences=True` because it was not the final recurrent layer and needed to pass the full sequence to the next LSTM layer. The model also had two dropout layers with a dropout rate of 0.2 to prevent overfitting. Finally, a Dense layer with a single unit was added to output the final prediction.

The model was then trained on the current fold's training data (`X_train_cv`, `y_train_cv`) for 50 epochs with a batch size of 32. Once training was complete, the model was validated on the current fold's validation set (`X_val_cv`), and the predictions (`y_val_pred_cv`) were evaluated using the Mean Squared Error (MSE). The MSE for each fold was appended to the list `cv_mse_scores`.

After the loop completed all folds, the average cross-validation MSE was computed using `np.mean()` on the `cv_mse_scores`. This value gave a robust estimate of the model's generalization performance across multiple time series splits.

*Convolution Neural Network (CNN)*

```

Suggested code may be subject to a license |
from sklearn.model_selection import TimeSeriesSplit

# Time series split
tscv = TimeSeriesSplit(n_splits=6)

cv_mse_scores = []
cv_rmse_scores = []
cv_mae_scores = []
cv_r2_scores = []

for train_index, val_index in tscv.split(X_train):
    X_train_cv, X_val_cv = X_train[train_index], X_train[val_index]
    y_train_cv, y_val_cv = y_train[train_index], y_train[val_index]

    # Build and compile the model
    model_cv = build_cnn_model((X_train_cv.shape[1], X_train_cv.shape[2]))
    model_cv.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')

    # Train the model
    model_cv.fit(X_train_cv, y_train_cv, epochs=50, batch_size=32, verbose=0)

    # Predict on the validation set
    y_val_pred = model_cv.predict(X_val_cv)
    y_val_pred = scaler_y.inverse_transform(y_val_pred)
    y_val_original = scaler_y.inverse_transform(y_val_cv)

    # Calculate performance metrics on the validation set
    mse_val = mean_squared_error(y_val_original, y_val_pred)
    rmse_val = np.sqrt(mse_val)
    mae_val = mean_absolute_error(y_val_original, y_val_pred)
    r2_val = r2_score(y_val_original, y_val_pred)

    # Store the cross-validation metrics
    cv_mse_scores.append(mse_val)
    cv_rmse_scores.append(rmse_val)
    cv_mae_scores.append(mae_val)
    cv_r2_scores.append(r2_val)

# Print cross-validation results
print(f"Time Series Cross-Validation Results:")
print(f"Mean MSE: {np.mean(cv_mse_scores)}")
print(f"Mean RMSE: {np.mean(cv_rmse_scores)}")
print(f"Mean MAE: {np.mean(cv_mae_scores)}")
print(f"Mean R-squared (R²): {np.mean(cv_r2_scores)}")

```

Figure 5.102 CNN Model TimeSeriesSplit Cross Validation

This code implemented a time series cross-validation (CV) for a CNN model using the TimeSeriesSplit object with 6 splits (`n_splits=6`). The purpose was to evaluate the performance of the CNN model on multiple time series folds, ensuring the temporal order of the data was maintained.

The for loop iterated over the training and validation indices (`train_index`, `val_index`) generated by `tscv.split()`. The training and validation sets (`X_train_cv`, `X_val_cv`, `y_train_cv`,

`y_val_cv`) were extracted for each fold, and for each fold, the CNN model (`model_cv`) was built and compiled. The CNN model included two Conv1D layers with 64 filters each, followed by a Dense layer with 50 units, Dropout layers for regularization, and a final Dense layer for regression. The model was compiled with the Adam optimizer and the loss function set to mean squared error (MSE), which is commonly used for regression tasks.

The CNN model was trained on the current fold's training data (`X_train_cv`, `y_train_cv`) for 50 epochs with a batch size of 32. After training, the model was validated on the current fold's validation set (`X_val_cv`), and the predictions (`y_val_pred`) were made. The predictions were inverse transformed using the scaler, to match the scale of the original data. Performance metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R2) were calculated on the validation set, and the results were stored in their respective lists (`cv_mse_scores`, `cv_rmse_scores`, `cv_mae_scores`, `cv_r2_scores`).

After completing all folds, the mean cross-validation scores for MSE, RMSE, MAE, and R2 were computed using `np.mean()` and printed, providing an overall assessment of the model's performance across all time series splits. This approach ensured that the CNN model's generalization ability was thoroughly evaluated across multiple folds.



*Recurrent neural network (RNN) with Attention Mechanism*

```

# Rolling window cross-validation
fold_scores = []
tscv = TimeSeriesSplit(n_splits=5)

for train_index, test_index in tscv.split(X_scaled):
    X_train_fold, X_test_fold = X_scaled[train_index], X_scaled[test_index]
    y_train_fold, y_test_fold = y_scaled[train_index], y_scaled[test_index]

    # Ensure that y_train_fold and y_test_fold have consistent lengths with X_train_fold and X_test_fold
    assert X_train_fold.shape[0] == y_train_fold.shape[0], "Mismatch in train set sizes"
    assert X_test_fold.shape[0] == y_test_fold.shape[0], "Mismatch in test set sizes"

    # Reshape the input for the RNN model (3D shape for RNNs)
    X_train_fold = np.reshape(X_train_fold, (X_train_fold.shape[0], X_train_fold.shape[1], 1))
    X_test_fold = np.reshape(X_test_fold, (X_test_fold.shape[0], X_test_fold.shape[1], 1))

    # Train the model on the current fold
    model.fit(X_train_fold, y_train_fold, epochs=100, batch_size=32, verbose=0,
              callbacks=[early_stopping, reduce_lr], validation_data=(X_test_fold, y_test_fold))

    # Predict on the test fold
    y_pred_fold = model.predict(X_test_fold)

    # Only extract the last time step prediction for each sample
    y_pred_fold = y_pred_fold[:, -1]

    # Reshape y_pred_fold to 2D before inverse transforming
    y_pred_fold = y_pred_fold.reshape(-1, 1)

    # Inverse transform the predicted and actual values to the original scale
    y_pred_fold = scaler_y.inverse_transform(y_pred_fold) # Inverse transform to original scale
    y_test_fold_original = scaler_y.inverse_transform(y_test_fold.reshape(-1, 1))

    # Ensure that predictions and true values are flattened (1D) for metric calculation
    y_pred_fold = y_pred_fold.flatten()

    # Calculate RMSE for the fold
    rmse = math.sqrt(mean_squared_error(y_test_fold_original, y_pred_fold))
    fold_scores.append(rmse)

# Return the average score across all folds
return np.mean(fold_scores)

```

Figure 5.103 RNN with attention Model Rolling Window Cross Validation with the TimeSeriesSplit object-based

This code implemented a rolling window cross-validation for an RNN with attention mechanism. The TimeSeriesSplit object was used to create 5 folds ( $n\_splits=5$ ), which ensured that the temporal order of the data was preserved throughout the process.

In each iteration of the cross-validation loop, training and test indices were generated (`train_index` and `test_index`), and the corresponding data folds (`X_train_fold`, `X_test_fold`, `y_train_fold`, `y_test_fold`) were extracted from the scaled dataset (`X_scaled` and `y_scaled`). Consistency checks were applied to ensure that the lengths of `y_train_fold` and `y_test_fold` matched with the respective `X_train_fold` and `X_test_fold`.

Before feeding the data into the RNN model, the input was reshaped into a 3D format, (samples, timesteps, features), required for RNNs. Both the training and test data were reshaped accordingly. The model was then trained on the current fold's training data for 100 epochs with

a batch size of 32. Callbacks, such as `early_stopping` and `reduce_lr`, were used to prevent overfitting and dynamically adjust the learning rate based on the validation performance.

After training, the model was tested on the fold's test data (`X_test_fold`), and predictions were made (`y_pred_fold`). Since the model was predicting the next time step, only the last time step prediction for each sample was extracted. This prediction was reshaped to 2D and inverse-transformed back to the original scale using the scaler. The same inverse transformation was applied to the true values (`y_test_fold`).

Both the predicted and true values were flattened into 1D arrays for metric calculation. The Root Mean Squared Error (RMSE) was calculated for each fold using `mean_squared_error`, and the square root of this value was stored in `fold_scores`. Finally, after all folds were evaluated, the average RMSE across all folds was returned using `np.mean(fold_scores)`.



## 5.6.4 Model Hyperparameters and Fine-Tuning

*Support Vector Machine(SVM)*

```
[ ] # Define the parameter grid
param_dist = {
    'svr__C': [0.1, 1, 10, 100],
    'svr__gamma': ['scale', 'auto'],
    'svr__epsilon': [0.1, 0.2, 0.5],
    'svr__kernel': ['rbf', 'linear']
}

# Set up RandomizedSearchCV to find the best hyperparameters
random_search = RandomizedSearchCV(svm_pipeline, param_distributions=param_dist, n_iter=20, cv=tscv, scoring='neg_mean_squared_error', verbose=1, n_jobs=-1, random_state=42)
random_search.fit(X_train, y_train)

# Get the best model
best_svm_model = random_search.best_estimator_

# Print the best parameters
print("Best hyperparameters:", random_search.best_params_)
```

Figure 5.104 SVM model Hyperparameters and Fine-tuning for get best model

The code implemented hyperparameter tuning for an SVM model using `RandomizedSearchCV` to find the optimal combination of hyperparameters. A parameter grid, `param_dist`, was defined, which included a range of values for `svr_C`, `svr_gamma`, `svr_epsilon`, and `svr_kernel`. These hyperparameters were essential for controlling various aspects of the Support Vector Regressor (SVR). Specifically, `svr_C`, with values ranging from 0.1 to 100, determined the regularization strength, controlling the trade-off between model complexity and training error. The `svr_gamma` parameter, tested with values 'scale' and 'auto', controlled the influence range of a single data point, while `svr_epsilon`, tested with values of 0.1, 0.2, and 0.5, defined the epsilon-tube where errors were not penalized. The `svr_kernel`, set to 'rbf' and 'linear', determined the kernel type used in the SVR.

The `RandomizedSearchCV` was configured with 20 iterations (`n_iter=20`) to randomly sample combinations from the hyperparameter grid. The scoring metric used was the negative mean squared error (`neg_mean_squared_error`), which is commonly used for regression tasks. `TimeSeriesSplit` (`cv=tscv`) was applied to ensure that the temporal structure of the data was preserved during cross-validation.

After completing the hyperparameter search, the best model (`best_svm_model`) was selected, and the optimal hyperparameters were retrieved and printed using `random_search.best_params_`. The `n_jobs=-1` parameter was used to parallelize the search

across all available processors, while the `random_state=42` ensured reproducibility. This process optimized the SVM model by exploring various combinations of hyperparameters, ultimately identifying the configuration that minimized the model's mean squared error.

*Long Short-Term Memory(LSTM)*

```
[ ] # Define the model building function for Keras Tuner
def build_model(hp):
    model = Sequential()

    # Tune LSTM units (50 to 300)
    hp_units = hp.Int('units', min_value=50, max_value=300, step=50)
    model.add(LSTM(units=hp_units, return_sequences=True, input_shape=(X_train.shape[1], 1)))

    # Tune dropout rate (0.1 to 0.5)
    hp_dropout = hp.Float('dropout_rate', min_value=0.1, max_value=0.5, step=0.1)
    model.add(Dropout(hp_dropout))

    # Add another LSTM layer
    model.add(LSTM(units=hp_units, return_sequences=False))
    model.add(Dropout(hp_dropout))

    # Output layer
    model.add(Dense(1))

    # Tune optimizer and learning rate
    hp_optimizer = hp.Choice('optimizer', values=['adam', 'rmsprop', 'nadam', 'sgd'])
    hp_learning_rate = hp.Float('learning_rate', min_value=1e-5, max_value=1e-3, sampling='log')

    if hp_optimizer == 'adam':
        optimizer = Adam(learning_rate=hp_learning_rate)
    elif hp_optimizer == 'rmsprop':
        optimizer = RMSprop(learning_rate=hp_learning_rate)
    elif hp_optimizer == 'nadam':
        optimizer = Nadam(learning_rate=hp_learning_rate)
    else:
        optimizer = SGD(learning_rate=hp_learning_rate, momentum=0.9)

    model.compile(optimizer=optimizer, loss='mean_squared_error')
    return model

# Initialize Keras Tuner
tuner = kt.Hyperband(
    build_model,
    objective='val_loss',
    max_epochs=50,
    factor=3,
    directory='tuning_dir',
    project_name='lstm_tuning'
)

# Perform hyperparameter tuning
tuner.search(X_train, y_train, epochs=50, validation_split=0.2)

# Retrieve the best model
best_model = tuner.get_best_models(num_models=1)[0]

# Summary of the best model
best_model.summary()

# Evaluate the best model
val_loss = tuner.results_summary()
```

Figure 5.105 LSTM model Hyperparmters and Fine-tuning for get best models

```
[ ] # Add Early Stopping and Regularization
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.regularizers import l2

# Define early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the best model with early stopping
history_best = best_model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test), callbacks=[early_stopping])
```

Figure 5.106 LSTM model add Function Early Stopping and Regularization and Train the best model

The provided code defined a model-building function for Keras Tuner, which facilitated hyperparameter optimization for an LSTM model. The function `build_model(hp)` allowed the model to tune specific parameters such as the number of LSTM units (ranging from 50 to 300), dropout rates (from 0.1 to 0.5), and the optimizer type. Inside the function, two LSTM layers were added, with the first having `return_sequences=True`, allowing it to output the full sequence to the subsequent layer. After each LSTM layer, a Dropout layer was applied to prevent overfitting. The final Dense layer, with a single unit, output the model's prediction. Additionally, the optimizer was dynamically chosen based on the hyperparameters provided by the `hp.Choice()` function. This included options like `adam`, `rmsprop`, `nadam`, and `sgd`, each with a tunable learning rate defined by `hp.Float()`.

The Keras Tuner was initialized using the Hyperband algorithm, which is a resource-efficient tuning strategy. It searched for the optimal hyperparameters by monitoring the `val_loss` over 50 epochs with a validation split of 0.2. The search process returned the best model, and the best model's summary was printed for evaluation.

Early stopping and regularization were also implemented to optimize model performance further. Early stopping monitored the `val_loss`, and if no improvement was observed for five epochs, training was halted, and the best weights were restored. The best model was trained with early stopping using 100 epochs and a batch size of 32, further evaluated on the test data, ensuring the model did not overfit and reached its best performance under the tuned conditions.

*Convolution Neural Network (CNN)*

```

def objective(trial):
    # Define the hyperparameters to optimize
    learning_rate = trial.suggest_float("learning_rate", 1e-5, 1e-2, log=True)
    filters = trial.suggest_categorical("filters", [32, 64, 128])
    kernel_size = trial.suggest_categorical("kernel_size", [3, 5])
    dropout_rate = trial.suggest_float("dropout_rate", 0.0, 0.5)

    # Build the CNN model with the suggested hyperparameters
    model = Sequential()
    model.add(Conv2D(filters=filters, kernel_size=kernel_size, padding='same', activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Conv2D(filters=filters, kernel_size=kernel_size, padding='same', activation='relu'))
    model.add(Flatten())
    model.add(Dense(50, activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1))

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

    # Train the model
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), verbose=0)

    # Evaluate the model
    y_pred = model.predict(X_test)
    y_pred = scaler_y.inverse_transform(y_pred)
    y_test_original = scaler_y.inverse_transform(y_test)
    mse = mean_squared_error(y_test_original, y_pred)

    return mse

# Create a study object
study = optuna.create_study(direction="minimize")

# Run the optimization
study.optimize(objective, n_trials=100)

# Get the best hyperparameters
best_params = study.best_params
print("Best hyperparameters:", best_params)

# Build the CNN model with the best hyperparameters
best_model = Sequential()
best_model.add(Conv2D(filters=best_params["filters"], kernel_size=best_params["kernel_size"], padding='same', activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])))
best_model.add(Conv2D(filters=best_params["filters"], kernel_size=best_params["kernel_size"], padding='same', activation='relu'))
best_model.add(Flatten())
best_model.add(Dense(50, activation='relu'))
best_model.add(Dropout(best_params["dropout_rate"]))
best_model.add(Dense(1))

# Compile the model
best_model.compile(optimizer=Adam(learning_rate=best_params["learning_rate"]), loss='mean_squared_error')

# Train the model with the best hyperparameters
history = best_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), verbose=1)

```

Figure 5.107 CNN model hyperparameters tuning ,fine-tune and also Train the best model

The code provided utilized the Optuna library to perform hyperparameter tuning for a Convolutional Neural Network (CNN). The goal was to optimize the model by adjusting several key hyperparameters and evaluating the mean squared error (MSE) as the objective metric.

In the objective(trial) function, a set of hyperparameters was defined for tuning. These included the learning rate (learning\_rate), number of filters in the convolutional layers (filters), kernel size (kernel\_size), and dropout rate (dropout\_rate). Optuna's suggest\_float() and suggest\_categorical() methods were used to explore different values for these hyperparameters. For instance, the learning rate was tuned between 1e-5 and 1e-2 on a logarithmic scale, and the number of filters was set to choose between 32, 64, and 128.

The CNN model was built with two Conv1D layers, each using the suggested number of filters, kernel size, and a 'relu' activation function. Dropout was applied after the convolutional layers to prevent overfitting. Finally, a Dense layer was added for the regression task, outputting a single value (likely the predicted stock price or some similar target).

The model was compiled using the Adam optimizer with the suggested learning rate and the mean\_squared\_error loss function. The model was then trained for 50 epochs with a batch size of 32 and validated on a test set during training. After training, the model predicted the test set and computed the MSE between the predictions and the actual values.

The study object was created to minimize the MSE, and Optuna ran 100 trials to find the best set of hyperparameters. The best parameters were printed, and the CNN model was rebuilt with these best-performing hyperparameters. The best model was then compiled and retrained, this time with the fine-tuned hyperparameters, to further improve its performance.

### *Recurrent neural network (RNN) with Attention Mechanism*

```

# Optimize the hyperparameters using Optuna
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=50)

# Best hyperparameters
best_params = study.best_params
print(f"Best hyperparameters: {best_params}")

# Create the final model with the best hyperparameters
if best_params['optimizer'] == 'adam':
    optimizer = Adam(learning_rate=best_params['learning_rate'])
elif best_params['optimizer'] == 'rmsprop':
    optimizer = RMSprop(learning_rate=best_params['learning_rate'])
elif best_params['optimizer'] == 'nadam':
    optimizer = Nadam(learning_rate=best_params['learning_rate'])
else:
    optimizer = SGD(learning_rate=best_params['learning_rate'])

best_model = create_rnn_attention_model(
    units=best_params['units'],
    dropout_rate=best_params['dropout_rate'],
    optimizer=optimizer
)

# Train the final model on the entire dataset
best_model.fit(X_scaled, y_scaled, epochs=100, batch_size=32, callbacks=[early_stopping, reduce_lr])

```

Figure 5.108 RNN with attention mechanism model hyperparameters tuning ,fine-tune and also Train the best model

The code above implemented hyperparameter tuning for an RNN with an attention mechanism using the Optuna framework. The process involved adjusting several hyperparameters to optimize the model for its task, and the objective was to minimize the loss, specifically using mean squared error (MSE) as the evaluation metric.

First, the `optuna.create_study()` method was used to create a study with the objective of minimizing the MSE. The `study.optimize()` function was called, where Optuna performed 50 trials (`n_trials=50`) to find the best combination of hyperparameters that resulted in the lowest MSE.

Once the best hyperparameters were identified, they were retrieved using `study.best_params`. These parameters included values for the optimizer type, learning rate, number of units in the RNN layer, and dropout rate. Depending on the selected optimizer, different learning rates were applied using Adam, RMSprop, Nadam, or SGD. Each optimizer's learning rate was dynamically set to the best value found during the tuning process.

The best model was then created using the `create_rnn_attention_model()` function, which constructed an RNN with an attention mechanism. This function accepted the tuned number of RNN units (`best_params['units']`), the dropout rate (`best_params['dropout_rate']`), and the selected optimizer.

Finally, the model was trained on the scaled dataset for 100 epochs with a batch size of 32. Early stopping and learning rate reduction callbacks were applied to prevent overfitting and to fine-tune the learning rate during training. These callbacks allowed the training process to stop early if there was no significant improvement in validation loss, and they adjusted the learning rate when the validation loss plateaued.



## 5.6.4 Model Test-Set Evaluation

*Support Vector Machine(SVM)*

```
# Evaluate on the test set
y_test_pred = best_svm_model.predict(X_test)

# Calculate performance metrics on the test set
mse_test = mean_squared_error(y_test, y_test_pred)
r2_test = r2_score(y_test, y_test_pred)
rmse_test = np.sqrt(mse_test)
mae_test = mean_absolute_error(y_test, y_test_pred)

# Print test performance metrics
print(f"Test Set Performance with Best SVM Model:")
print(f"Mean Squared Error (MSE): {mse_test}")
print(f"R-squared (R2): {r2_test}")
print(f"Root Mean Squared Error (RMSE): {rmse_test}")
print(f"Mean Absolute Error (MAE): {mae_test}")
```

Figure 5.109 SVM model test-set evaluation

The code snippet demonstrated the evaluation process of the best Support Vector Machine (SVM) model on the test dataset by calculating key performance metrics. First, the predictions on the test set were generated using `best_svm_model.predict(X_test)`, which provided the predicted values (`y_test_pred`) based on the trained SVM model and the test features (`X_test`).

Next, various performance metrics were computed to assess the model's accuracy. The Mean Squared Error (MSE) was calculated with `mean_squared_error(y_test, y_test_pred)`, representing the average squared difference between actual and predicted values, indicating the model's overall prediction error. The R-squared ( $R^2$ ) score, calculated using `r2_score(y_test, y_test_pred)`, measured how well the predicted values explained the variance in the actual test data, with higher values (closer to 1) reflecting a better fit. To provide further clarity, the Root Mean Squared Error (RMSE) was derived by taking the square root of the MSE using `np.sqrt(mse_test)`, offering a more interpretable error measurement in the same units as the target variable. Additionally, the Mean Absolute Error (MAE) was computed using `mean_absolute_error(y_test, y_test_pred)`, representing the average absolute differences between the predicted and actual values, helping to gauge the accuracy of predictions.



*Long Short-Term Memory (LSTM)*

```
# Evaluate the model on the test set
y_test_pred = best_model.predict(X_test)
y_test_pred = scaler_y.inverse_transform(y_test_pred)
y_test_original = scaler_y.inverse_transform(y_test)

# Calculate performance metrics
mse = mean_squared_error(y_test_original, y_test_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test_original, y_test_pred)
r2 = r2_score(y_test_original, y_test_pred)

# Print the evaluation metrics
print(f"Final Test Set Performance with Fine-Tuned LSTM:")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"R-squared (R²): {r2}")
```

Figure 5.110 LSTM model test-set Evaluation

The code provided evaluated the performance of the fine-tuned LSTM model on the test dataset. The first step was to predict the target values (`y_test_pred`) using the model's `predict()` function. Since the predictions were generated in the scaled space (as the features and target values were scaled during training), both the predicted values and the actual test values were inverse transformed back to their original scale using `scaler_y.inverse_transform()`. This step ensured that the error metrics would be computed in the original, meaningful units of the target variable.

Next, the performance of the model was assessed using four key metrics. The Mean Squared Error (MSE) calculated the average of the squared differences between the true values and predictions, which provided insight into the overall accuracy of the model. The Root Mean Squared Error (RMSE) was derived from the square root of the MSE, making it easier to interpret since it presented the error in the same units as the target. The Mean Absolute Error (MAE) measured the average magnitude of the errors, ignoring their direction, thus showing how much the predictions deviated from the actual values on average. Finally, the R-squared

( $R^2$ ) score was calculated to evaluate how well the model explained the variance in the data, with values closer to 1 indicating better predictive performance.

### *Convolution Neural Network (CNN)*

```
[ ] # Evaluate on test set
    y_test_pred = best_model.predict(X_test)
    y_test_pred = scaler_y.inverse_transform(y_test_pred)
    y_test_original = scaler_y.inverse_transform(y_test)

    # Calculate performance metrics on the test set
    mse_test = mean_squared_error(y_test_original, y_test_pred)
    r2_test = r2_score(y_test_original, y_test_pred)
    rmse_test = np.sqrt(mse_test)
    mae_test = mean_absolute_error(y_test_original, y_test_pred)

    # Print test performance metrics
    print(f"Test Set Performance:")
    print(f"Mean Squared Error (MSE): {mse_test}")
    print(f"R-squared ( $R^2$ ): {r2_test}")
    print(f"Root Mean Squared Error (RMSE): {rmse_test}")
    print(f"Mean Absolute Error (MAE): {mae_test}")
```

Figure 5.111 CNN model test-set Evaluation

The code provided focused on evaluating the performance of a trained CNN model on the test dataset. First, the test set predictions (`y_test_pred`) were generated using `best_model.predict(X_test)`. Since the model's predictions were scaled during the training process, the predicted values were transformed back to their original scale using `scaler_y.inverse_transform()`. Similarly, the actual test values (`y_test_original`) were also inverse transformed to enable a fair comparison with the predicted values.

After transforming the values, several performance metrics were calculated to assess the model's accuracy on the test data. The Mean Squared Error (MSE), computed with `mean_squared_error()`, measured the average squared differences between the actual and predicted values, providing insight into the overall prediction error. The R-squared ( $R^2$ ) score, calculated using `r2_score()`, indicated the proportion of variance in the test set explained by the

model, where a score closer to 1 suggests a stronger predictive capability. Additionally, the Root Mean Squared Error (RMSE), calculated as the square root of the MSE, helped interpret the model's error in the same units as the target variable, while the Mean Absolute Error (MAE) captured the average magnitude of errors in the predictions without considering their direction.

Finally, the results of the evaluation, including MSE,  $R^2$ , RMSE, and MAE, were printed, allowing for a detailed understanding of the CNN model's performance on the test set, thus helping assess the model's generalization to unseen data.

### *Recurrent neural network (RNN) with Attention Mechanism*

```

# Evaluate the model on the test set
y_test_pred_final = best_model.predict(X_test)
y_test_pred_final = y_test_pred_final[:, -1, 0].reshape(-1, 1)
y_test_pred_final = scaler_y.inverse_transform(y_test_pred_final)
y_test_original_final = scaler_y.inverse_transform(y_test)

# Calculate performance metrics for the final model
final_mse = mean_squared_error(y_test_original_final, y_test_pred_final)
final_rmse = np.sqrt(final_mse)
final_mae = mean_absolute_error(y_test_original_final, y_test_pred_final)
final_r2 = r2_score(y_test_original_final, y_test_pred_final)

print(f"Final Test Set Performance with Fine-Tuned RNN + Attention Model:")
print(f"Mean Squared Error (MSE): {final_mse}")
print(f"Root Mean Squared Error (RMSE): {final_rmse}")
print(f"Mean Absolute Error (MAE): {final_mae}")
print(f"R-squared (R2): {final_r2}")

```

Figure 5.112 RNN with Attention Mechanism model test-set Evaluation

The code shown evaluated the final RNN with Attention Model on the test set. The model's predictions on the test data were obtained using `best_model.predict()`, which returned the predicted values (`y_test_pred_final`). Since the model was predicting a sequence, only the final time step predictions were extracted using slicing (`y_test_pred_final[:, -1, 0]`). This 1D array of predictions was reshaped to match the shape of the actual test data. Both the predicted and actual test values were then inverse transformed using `scaler_y.inverse_transform()` to revert them back to their original scale for meaningful interpretation.

Once the predictions were in their original scale, the model's performance was evaluated using several metrics. The Mean Squared Error (MSE) was computed to measure the average squared difference between the predicted and actual values. The Root Mean Squared Error (RMSE), derived from the MSE, provided the error in the same units as the target variable, making it easier to interpret. The Mean Absolute Error (MAE) quantified the average magnitude of errors in the predictions. Lastly, the R-squared ( $R^2$ ) score was calculated to determine how well the model explained the variance in the data, where a value closer to 1 indicated better model performance.

These evaluation metrics were printed to provide a comprehensive summary of how well the fine-tuned RNN with Attention Model performed on the unseen test data.

#### 5.6.4 Model Results Visualisation

##### *Support Vector Machine (SVM)*

```

▶ # Filter data for January 2019
start_date = '2019-01-01'
end_date = '2019-01-31'
data_jan_2019 = data[(data['Date'] >= start_date) & (data['Date'] <= end_date)]

# Extract features and target for January 2019
X_jan_2019 = data_jan_2019[numerical_columns].values
y_jan_2019 = data_jan_2019['Close'].values

# Predict on the January 2019 data
y_jan_2019_pred = best_svm_model.predict(X_jan_2019)

# Create the visualization
plt.figure(figsize=(12, 6))
plt.plot(data_jan_2019['Date'], y_jan_2019, label='Actual Close')
plt.plot(data_jan_2019['Date'], y_jan_2019_pred, label='Predicted Close')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Actual vs Predicted Close Price (January 2019)')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

```

Figure 5.113 Daily Forecast Visualisation of Actual Vs Predicted Close Prices in January 2019 that retrieved from results generated by SVM model

The provided code snippet filtered the dataset to include only daily data from January 2019 for visualization purposes. It first defined a start and end date (2019-01-01 to 2019-01-31) and filtered the dataset accordingly. After filtering the data, the features (numerical columns) and the target variable (Close price) for this date range were extracted.

Next, the best-trained SVM model predicted the Close price for the filtered January 2019 data. The predictions (`y_jan_2019_pred`) were then compared to the actual Close prices (`y_jan_2019`) for the same period.

The visualization was created using Matplotlib. A figure was initialized with a size of 12 by 6 inches. Two lines were plotted on the same graph: one representing the actual close prices (`y_jan_2019`) and the other representing the predicted close prices (`y_jan_2019_pred`). The x-axis represented the date, while the y-axis represented the close price. Labels and a legend were added to distinguish between actual and predicted values, and the x-ticks were rotated for better readability. The graph was completed with grid lines for clarity and finally displayed using `plt.show()`.

This visualization provided a comparison of how well the SVM model's predicted close prices aligned with the actual close prices for January 2019 in daily basis.



```

Suggested code may be subject to a license | AnishaCoimbatore/PythonProgramming
# Define the date range
start_date = '2019-01-01'
end_date = '2024-01-31'

# Filter data for the specified date range
data_filtered = data[(data['Date'] >= start_date) & (data['Date'] <= end_date)]

# Create an empty list to store the predicted and actual values
predicted_values = []
actual_values = []
dates = []

# Iterate through each month
for year in range(2019, 2024):
    for month in range(1, 13):
        # Filter data for the current month
        start_month = f"{year}-{month:02d}-01"
        if month == 12:
            end_month = f"{year + 1}-01-01"
        else:
            end_month = f"{year}-{month + 1:02d}-01"
        data_month = data_filtered[(data_filtered['Date'] >= start_month) & (data_filtered['Date'] < end_month)]

        # Extract features and target
        X_month = data_month[numerical_columns].values
        y_month = data_month['Close'].values

        # Check if data exists for the current month
        if len(X_month) > 0:
            # Predict on the current month data
            y_month_pred = best_svm_model.predict(X_month)

            # Append the predicted and actual values to the lists
            predicted_values.extend(y_month_pred)
            actual_values.extend(y_month)
            dates.extend(data_month['Date'].values)

# Convert dates to datetime objects
dates = pd.to_datetime(dates)

# Create the visualization
plt.figure(figsize=(15, 8))
plt.plot(dates, actual_values, label='Actual Close', color='blue')
plt.plot(dates, predicted_values, label='Predicted Close', color='red')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Actual vs Predicted Close Price (January 2019 - January 2024)')
plt.legend()

# Format x-axis with month and year
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=3))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

```

Figure 5.113 Monthly Forecast Visualisation of Actual Vs Predicted Close Prices from January 2019 until January 2024 that retrieved from results generated by SVM model

The code implemented a visualization to compare the actual versus predicted stock close prices using an SVM model over a monthly ranging from January 2019 to January 2024. The process began by defining the date range, filtering the dataset to include only data between these dates. Two empty lists, `predicted_values` and `actual_values`, were initialized to store the predicted close prices and actual close prices, respectively, for each month. Another list, `dates`, was created to store the corresponding dates for both predicted and actual values. The code iterated over each month within the date range. For each iteration, it filtered the data to extract the relevant features and target values for that month. It then checked if there was available data for the current month. If data was present, the SVM model predicted the close prices for that month. The predicted close prices, actual close prices, and the dates were appended to their respective lists.

After collecting the predicted and actual values, the dates were converted into a pandas datetime object for easy plotting. A dual-line plot was created using Matplotlib to visualize the comparison between actual and predicted close prices. The actual close prices were plotted in blue, while the predicted close prices were plotted in red. The plot was enhanced with appropriate labels for the x-axis (Date) and y-axis (Close Price), and a title was added to indicate the period of visualization. Additionally, the x-axis was formatted to display months and years at a 3-month interval, with the dates presented in a %Y-%m format and rotated for better readability. The grid lines were enabled to improve clarity, and the final visualization was displayed.

This visualization provided an overview of how well the SVM model's predictions aligned with the actual stock prices over the entire 5-year in monthly period, allowing for easy comparison and interpretation of model performance across time.

```

Suggested code may be subject to a license | ArmanBari123/Environmental_data_webapp
# Define the date range
start_date = '2019-01-01'
end_date = '2024-01-31'

# Filter data for the specified date range
data_filtered = data[(data['Date'] >= start_date) & (data['Date'] <= end_date)]

# Create an empty list to store the predicted and actual values
predicted_values = []
actual_values = []
dates = []

# Iterate through each year
for year in range(2019, 2024):
    # Filter data for the current year
    start_year = f"{year}-01-01"
    end_year = f"{year + 1}-01-01"
    data_year = data_filtered[(data_filtered['Date'] >= start_year) & (data_filtered['Date'] < end_year)]

    # Extract features and target
    X_year = data_year[numerical_columns].values
    y_year = data_year['Close'].values

    # Check if data exists for the current year
    if len(X_year) > 0:
        # Predict on the current year data
        y_year_pred = best_svm_model.predict(X_year)

        # Append the predicted and actual values to the lists
        predicted_values.extend(y_year_pred)
        actual_values.extend(y_year)
        dates.extend(data_year['Date'].values)

# Convert dates to datetime objects
dates = pd.to_datetime(dates)

# Create the visualization
plt.figure(figsize=(15, 8))
plt.plot(dates, actual_values, label='Actual Close', color='blue')
plt.plot(dates, predicted_values, label='Predicted Close', color='red')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Actual vs Predicted Close Price (2019 - 2023)')
plt.legend()

# Format x-axis with year
plt.gca().xaxis.set_major_locator(mdates.YearLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

```

Figure 5.114 Annually Forecast Visualisation of Actual Vs Predicted Close Prices from January 2019 until January 2024 that retrieved from results generated by SVM model

This code generated an annual visualization of actual versus predicted stock close prices using the SVM model for the years 2019 to 2023. The process began by defining the date range from January 1, 2019, to January 31, 2024, and filtering the dataset based on this range. Empty lists were initialized to store the predicted values, actual values, and the corresponding dates.



A loop was then implemented to iterate through each year within the specified range. For each year, the data was filtered for that specific year. The features (`X_year`) and the target variable (`y_year`) were extracted for the stock close prices during that year. If data existed for the selected year, the SVM model predicted the close prices (`y_year_pred`). Both the predicted and actual values were appended to their respective lists, along with the corresponding dates.

After gathering the predicted and actual values across all years, the dates were converted into datetime objects to enable proper plotting on the x-axis. The actual and predicted values were plotted using Matplotlib, where the actual close prices were visualized in blue and the predicted close prices in red. The plot's title indicated that it visualized the actual versus predicted close prices for the period between 2019 and 2023.

The x-axis was formatted to display the years, with a major tick at each year using `YearLocator`, and the labels for the years were formatted using `DateFormatter`. The x-axis tick labels were rotated by 45 degrees for clarity, and a grid was added to the plot. This allowed a clear comparison of actual versus predicted stock prices over the entire multi-year period.

*Long Short-Term Memory (LSTM)*

```

import matplotlib.pyplot as plt

# Filter data for January 2019
start_date = '2019-01-01'
end_date = '2019-01-31'
data_jan_2019 = data[(data['Date'] >= start_date) & (data['Date'] <= end_date)]

# Extract relevant data for visualization
X_jan_2019 = data_jan_2019[numerical_columns].values
y_jan_2019 = data_jan_2019['Close'].values
dates_jan_2019 = data_jan_2019['Date']

# Scale the data for prediction
X_jan_2019_scaled = scaler_X.transform(X_jan_2019)
X_jan_2019_scaled = X_jan_2019_scaled.reshape((X_jan_2019_scaled.shape[0], X_jan_2019_scaled.shape[1], 1))

# Predict using the best model
y_pred_jan_2019_scaled = best_model.predict(X_jan_2019_scaled)
y_pred_jan_2019 = scaler_y.inverse_transform(y_pred_jan_2019_scaled)

# Create the visualization
plt.figure(figsize=(12, 6))
plt.plot(dates_jan_2019, y_jan_2019, label='Actual Close Price', color='blue')
plt.plot(dates_jan_2019, y_pred_jan_2019, label='Predicted Close Price', color='red')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Actual vs Predicted Close Price - January 2019')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

```

Figure 5.115 Daily Forecast Visualisation of Actual Vs Predicted Close Prices in January 2019 that retrieved from results generated by LSTM model

This code was used to generate a daily forecast visualization comparing actual versus predicted stock close prices for January 2019 using the fine-tuned LSTM model. The first step was to filter the dataset for the relevant date range, from January 1, 2019, to January 31, 2019. The features (`X_jan_2019`) and target variable (`y_jan_2019`) were extracted for this specific month. In addition, the dates (`dates_jan_2019`) were extracted to use them for labeling the x-axis in the visualization.

The features were then scaled using the `StandardScaler` to standardize the input values before they were fed into the LSTM model. After scaling, the data was reshaped into the required 3D format of (samples, timesteps, features), suitable for the sequential data processing nature of LSTM models.

The fine-tuned LSTM model was used to predict the stock close prices for January 2019, and the predicted values (`y_pred_jan_2019_scaled`) were transformed back to the original scale using the inverse transformation of the scaler to make them comparable with the actual stock prices.

Lastly, the visualization was created using Matplotlib, where the actual stock prices were plotted in blue, and the predicted prices were plotted in red. The x-axis was labeled as "Date," and the y-axis was labeled as "Close Price." The title of the plot indicated that the visualization displayed actual versus predicted close prices for January 2019. The x-axis tick labels were rotated by 45 degrees for better readability, and a grid was added to the plot for clarity. This daily forecast allowed for a direct comparison between the actual and predicted stock prices over the selected period.

```

# Group data by month and year
data['YearMonth'] = pd.to_datetime(data['Date']).dt.to_period('M')
monthly_data = data.groupby('YearMonth').agg({'Close': 'last'})

# Predict on the monthly data
X_monthly = data.groupby('YearMonth').agg({'Open': 'first', 'High': 'max', 'Low': 'min', 'Volume': 'sum',
    'sentiment_score': 'mean', 'Close_lag1': 'last', 'Close_lag2': 'last',
    'Close_rolling_mean': 'last', 'Close_rolling_std': 'last'})

# Convert PeriodIndex to DateTimeIndex
X_monthly.index = X_monthly.index.to_timestamp()

X_monthly_scaled = scaler_X.transform(X_monthly)
X_monthly_scaled = X_monthly_scaled.reshape(X_monthly_scaled.shape[0], X_monthly_scaled.shape[1], 1)
y_pred_monthly = best_model.predict(X_monthly_scaled)
# The output of the prediction is 2-dimensional. Remove the extra index.
y_pred_monthly = y_pred_monthly[:, -1].reshape(-1, 1)
y_pred_monthly = scaler_y.inverse_transform(y_pred_monthly)

# Plot the actual vs predicted values for monthly data
plt.figure(figsize=(12, 6))

# Convert PeriodIndex to DateTimeIndex for plotting
plt.plot(monthly_data.index.to_timestamp(), monthly_data['Close'], label='Actual Close Price')

plt.plot(monthly_data.index.to_timestamp(), y_pred_monthly, label='Predicted Close Price')
plt.xlabel('Month')
plt.ylabel('Close Price')
plt.title('Actual vs Predicted Close Price - Monthly')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)

# Add month labels to the x-axis
# Use monthly_data.index instead of dates
month_labels = [month.strftime('%Y-%m') for month in monthly_data.index.to_timestamp()]
plt.xticks(monthly_data.index.to_timestamp(), month_labels, rotation=45, ha='right')
plt.tight_layout()

plt.show()

```

Figure 5.116 Monthly Forecast Visualisation of Actual Vs Predicted Close Prices from January 2019 that retrieved from results generated by LSTM model

This code performed a monthly forecast visualization of actual versus predicted close prices using the LSTM model. The process began by grouping the data by month and year. The dataset was enriched by adding a 'YearMonth' column to capture each entry's corresponding month and year. The grouped data was aggregated based on several features: 'Open,' 'High,' 'Low,' and 'Volume' were summarized through different aggregation functions, including sums, averages, and minimums. Lagged values for close prices and other rolling statistics were also included.

The next step involved predicting the monthly close prices using the LSTM model. The aggregated monthly data (`X_monthly`) was prepared for prediction by scaling the features using `StandardScaler`, ensuring that the input data was normalized before feeding it into the model. After scaling, the data was reshaped into a 3D format, as LSTM models expect inputs of shape (samples, timesteps, features). The LSTM model then predicted the monthly close prices, and the predicted values were inverse-transformed back to their original scale for comparison with the actual values.

The plot was generated to visualize the actual versus predicted monthly close prices. The `matplotlib` library was used to create the figure. The x-axis represented the months, and the y-axis displayed the close prices. The actual values were plotted in blue, while the predicted values from the LSTM model were plotted in red. The x-axis labels were formatted to display the year and month, with the tick labels rotated by 45 degrees for improved readability. The grid was enabled to enhance clarity, and the figure title was set as "Actual vs Predicted Close Price - Monthly."

This approach helped capture trends in monthly stock prices and allowed the comparison of the LSTM model's performance in predicting the future values of close prices on a monthly scale.

```

# Group data by month and year
data['YearMonth'] = pd.to_datetime(data['Date']).dt.to_period('M')
monthly_data = data.groupby('YearMonth').agg({'Close': 'last'})

# Predict on the Year data
X_monthly = data.groupby('YearMonth').agg({'Open': 'first', 'High': 'max', 'Low': 'min', 'Volume': 'sum',
                                          'sentiment_score': 'mean', 'Close_lag1': 'last', 'Close_lag2': 'last',
                                          'Close_rolling_mean': 'last', 'Close_rolling_std': 'last'})

# Convert PeriodIndex to DateTimeIndex
X_monthly.index = X_monthly.index.to_timestamp()

X_monthly_scaled = scaler_X.transform(X_monthly)
X_monthly_scaled = X_monthly_scaled.reshape(X_monthly_scaled.shape[0], X_monthly_scaled.shape[1], 1)
y_pred_monthly = best_model.predict(X_monthly_scaled)
# The output of the prediction is 2-dimensional. Access the last element of the
# first dimension and reshape to a 2-dimensional array.
y_pred_monthly = y_pred_monthly[:, -1].reshape(-1, 1)
y_pred_monthly = scaler_y.inverse_transform(y_pred_monthly)

# Plot the actual vs predicted values for yearly data
plt.figure(figsize=(12, 6))

# Convert PeriodIndex to DateTimeIndex for plotting
plt.plot(monthly_data.index.to_timestamp(), monthly_data['Close'], label='Actual Close Price')

plt.plot(monthly_data.index.to_timestamp(), y_pred_monthly, label='Predicted Close Price')
plt.xlabel('Year')
plt.ylabel('Close Price')
plt.title('Actual vs Predicted Close Price - Yearly')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

```

Figure 5.117 Annually Forecast Visualisation of Actual Vs Predicted Close Prices from January 2019 that retrieved from results generated by LSTM model

This code executed an annual visualization of actual versus predicted close prices using the LSTM model. It began by grouping the data by both month and year, utilizing the 'YearMonth' column to aggregate the data based on different features like 'Open,' 'High,' 'Low,' and 'Volume.' Other features like 'Close\_lag1' and 'Close\_lag2' were preserved for each last value of the month, which helped in predicting future close prices.

The grouped monthly data was then scaled using StandardScaler to ensure that the features were normalized before feeding into the model. The LSTM model required the data to be reshaped into a 3D format, with each sample representing a time series for the year. After reshaping and feeding the data into the LSTM model, the predicted close prices were generated.

Next, the model's predictions were inverse-transformed to bring them back to their original scale, allowing a comparison between the actual and predicted close prices. The predictions were reshaped as necessary for further processing.

The visualization plot was then generated using matplotlib, showing the actual and predicted close prices for each year. The x-axis represented the years, and the y-axis showed the close prices. The actual close prices were plotted in blue, while the predicted prices were displayed in red. The x-axis tick labels were formatted to show years, with labels rotated by 45 degrees for clarity. A grid was added to enhance visualization, and the title of the plot was set as "Actual vs Predicted Close Price - Yearly."

This visualization provided an annual-level perspective on how well the LSTM model could capture and predict stock price movements, allowing the comparison of actual prices with model-generated forecasts across years.

### *Convolution Neural Network (CNN)*

```

# Filter data for January 2019
start_date = '2019-01-01'
end_date = '2019-01-31'
data_jan_2019 = data[(data['Date'] >= start_date) & (data['Date'] <= end_date)]

# Extract relevant data for visualization
X_jan_2019 = data_jan_2019[numerical_columns].values
y_jan_2019 = data_jan_2019['Close'].values
dates_jan_2019 = data_jan_2019['Date']

# Scale the data for prediction
X_jan_2019_scaled = scaler_X.transform(X_jan_2019)
X_jan_2019_scaled = X_jan_2019_scaled.reshape((X_jan_2019_scaled.shape[0], X_jan_2019_scaled.shape[1], 1))

# Predict using the best model
y_pred_jan_2019_scaled = best_model.predict(X_jan_2019_scaled)
y_pred_jan_2019 = scaler_y.inverse_transform(y_pred_jan_2019_scaled)

# Create the visualization
plt.figure(figsize=(12, 6))
plt.plot(dates_jan_2019, y_jan_2019, label='Actual Close Price', color='blue')
plt.plot(dates_jan_2019, y_pred_jan_2019, label='Predicted Close Price', color='red')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Actual vs Predicted Close Price - January 2019')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

```

Figure 5.118 Daily Forecast Visualisation of Actual Vs Predicted Close Prices in January 2019 that retrieved from results generated by CNN model

This code generated a daily forecast visualization using the CNN model for the month of January 2019. The process started by filtering the data for the specific date range between January 1, 2019, and January 31, 2019. The relevant features for prediction, including the numerical columns and the target 'Close' prices, were extracted from the filtered data. Next, the features were scaled using the StandardScaler to normalize them, which is essential for CNN models to perform well. The data was reshaped into a 3D format to fit the requirements of the CNN model, where the shape of the data was adjusted to match the input format expected by the model. The CNN model then predicted the scaled data, and the resulting predicted close prices were inverse-transformed to their original scale.

For visualization, the matplotlib library was used to create a plot of actual versus predicted close prices for January 2019. The actual close prices were displayed in blue, while the predicted close prices generated by the CNN model were shown in red. The x-axis represented the dates, while the y-axis showed the close prices. To improve readability, the date labels on the x-axis were rotated by 45 degrees, and a grid was added to the plot. The plot title, "Actual vs Predicted Close Price - January 2019," provided a clear representation of the performance of the CNN model over this period.



```

▶ # Group data by month and year
data['YearMonth'] = pd.to_datetime(data['Date']).dt.to_period('M')
monthly_data = data.groupby('YearMonth').agg({'Close': 'last'})

# Predict on the monthly data
X_monthly = data.groupby('YearMonth').agg({'Open': 'first', 'High': 'max', 'Low': 'min', 'Volume': 'sum',
    'sentiment_score': 'mean', 'Close_lag1': 'last', 'Close_lag2': 'last',
    'Close_rolling_mean': 'last', 'Close_rolling_std': 'last'})

# Convert PeriodIndex to DateTimeIndex
X_monthly.index = X_monthly.index.to_timestamp()

X_monthly_scaled = scaler_X.transform(X_monthly)
X_monthly_scaled = X_monthly_scaled.reshape(X_monthly_scaled.shape[0], X_monthly_scaled.shape[1], 1)
y_pred_monthly = best_model.predict(X_monthly_scaled)
# The output of the prediction is 2-dimensional. Remove the extra index.
y_pred_monthly = y_pred_monthly[:, -1].reshape(-1, 1)
y_pred_monthly = scaler_y.inverse_transform(y_pred_monthly)

# Plot the actual vs predicted values for monthly data
plt.figure(figsize=(12, 6))

# Convert PeriodIndex to DateTimeIndex for plotting
plt.plot(monthly_data.index.to_timestamp(), monthly_data['Close'], label='Actual Close Price')

plt.plot(monthly_data.index.to_timestamp(), y_pred_monthly, label='Predicted Close Price')
plt.xlabel('Month')
plt.ylabel('Close Price')
plt.title('Actual vs Predicted Close Price - Monthly')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)

# Add month labels to the x-axis
# Use monthly_data.index instead of dates
month_labels = [month.strftime('%Y-%m') for month in monthly_data.index.to_timestamp()]
plt.xticks(monthly_data.index.to_timestamp(), month_labels, rotation=45, ha='right')
plt.tight_layout()

plt.show()

```

Figure 5.119 Monthly Forecast Visualisation of Actual Vs Predicted Close Prices from January 2019 that retrieved from results generated by CNN model

This code generated a monthly forecast visualization using the CNN model. The data was first grouped by month and year, creating a new column 'YearMonth' that aggregated the 'Close' price and other features. The monthly data was processed by aggregating different statistical measures like the first, last, sum, or mean for each feature.

Next, the features were scaled using the StandardScaler to normalize the data. The data was reshaped into a 3D format to match the input requirements of the CNN model. The CNN model was used to predict the monthly close prices, and the predictions were inverse-transformed to their original scale for comparison.



To visualize the results, the matplotlib library was employed to plot the actual and predicted close prices for each month. The actual close prices were plotted in blue, and the predicted close prices were plotted in red. The x-axis represented the months, and the y-axis displayed the close prices. The x-axis labels were formatted to show the month and year (e.g., 'Jan-19') and rotated 45 degrees for readability.

This monthly forecast plot provided a clear comparison between the actual and predicted close prices, showing how well the CNN model performed in predicting monthly stock price trends over the given period.

```

# Group data by month and year
data['YearMonth'] = pd.to_datetime(data['Date']).dt.to_period('M')
monthly_data = data.groupby('YearMonth').agg({'Close': 'last'})

# Predict on the Year data
X_monthly = data.groupby('YearMonth').agg({'Open': 'first', 'High': 'max', 'Low': 'min', 'Volume': 'sum',
    'sentiment_score': 'mean', 'Close_lag1': 'last', 'Close_lag2': 'last',
    'Close_rolling_mean': 'last', 'Close_rolling_std': 'last'})

# Convert PeriodIndex to DateTimeIndex
X_monthly.index = X_monthly.index.to_timestamp()

X_monthly_scaled = scaler_X.transform(X_monthly)
X_monthly_scaled = X_monthly_scaled.reshape(X_monthly_scaled.shape[0], X_monthly_scaled.shape[1], 1)
y_pred_monthly = best_model.predict(X_monthly_scaled)
# The output of the prediction is 2-dimensional. Access the last element of the
# first dimension and reshape to a 2-dimensional array.
y_pred_monthly = y_pred_monthly[:, -1].reshape(-1, 1)
y_pred_monthly = scaler_y.inverse_transform(y_pred_monthly)

# Plot the actual vs predicted values for yearly data
plt.figure(figsize=(12, 6))

# Convert PeriodIndex to DateTimeIndex for plotting
plt.plot(monthly_data.index.to_timestamp(), monthly_data['Close'], label='Actual Close Price')

plt.plot(monthly_data.index.to_timestamp(), y_pred_monthly, label='Predicted Close Price')
plt.xlabel('Year')
plt.ylabel('Close Price')
plt.title('Actual vs Predicted Close Price - Yearly')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

```

Figure 5.120 Annually Forecast Visualisation of Actual Vs Predicted Close Prices from January 2019 that retrieved from results generated by CNN model

This code generated an annual forecast visualization using a CNN model. The data was first grouped by month and year to prepare for yearly predictions. A new column, 'YearMonth,' was created, and the data was aggregated to calculate the last 'Close' price for each month.

The yearly data was extracted by grouping features like 'Open', 'High', 'Low', 'Volume', and lag features, aggregating them with statistical functions such as 'first,' 'last,' 'mean,' and 'sum.' These aggregated features provided a summary of the year's performance to be used as input for the model.

The features were then scaled using a StandardScaler to normalize the values, which ensured that all inputs were on the same scale. The scaled data was reshaped to match the 3D format required by the CNN model. The CNN model was used to predict the yearly close prices, and the predicted values were inverse-transformed back to the original scale for a direct comparison with the actual values.

To visualize the predictions, matplotlib was used to plot both the actual and predicted close prices over the years. The actual close prices were shown in blue, while the predicted close prices were represented in red. The x-axis denoted the year, and the y-axis displayed the close prices. The x-axis labels were formatted to show the year, and a grid was added for clarity.

This annual forecast provided a clear comparison between the actual and predicted close prices, showing how well the CNN model captured the overall trend in yearly stock price movements.

*Recurrent neural network (RNN) with Attention Mechanism*

```

import matplotlib.pyplot as plt

# Filter data for January 2019
start_date = '2019-01-01'
end_date = '2019-01-31'
data_jan_2019 = data[(data['Date'] >= start_date) & (data['Date'] <= end_date)]

# Extract features and target for January 2019
X_jan_2019 = data_jan_2019[features]
y_jan_2019 = data_jan_2019[target]

# Scale the features for January 2019 data
X_jan_2019_scaled = scaler_X.transform(X_jan_2019)
X_jan_2019_scaled = X_jan_2019_scaled.reshape(X_jan_2019_scaled.shape[0], X_jan_2019_scaled.shape[1], 1)

# Predict on the January 2019 data
y_pred_jan_2019 = best_model.predict(X_jan_2019_scaled)
y_pred_jan_2019 = y_pred_jan_2019[:, -1, 0].reshape(-1, 1)
y_pred_jan_2019 = scaler_y.inverse_transform(y_pred_jan_2019)

# Plot the actual vs predicted values
plt.figure(figsize=(12, 6))
plt.plot(data_jan_2019['Date'], y_jan_2019, label='Actual Close Price')
plt.plot(data_jan_2019['Date'], y_pred_jan_2019, label='Predicted Close Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Actual vs Predicted Close Price - January 2019')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

```

Figure 5.121 Daily Forecast Visualisation of Actual Vs Predicted Close Prices in January 2019 that retrieved from results generated by RNN with Attention Mechanism model

This code snippet generated a daily forecast visualization for January 2019 using an RNN with an attention mechanism. The code began by filtering the dataset to include data between January 1st and January 31st, 2019, based on the date range specified in the `start_date` and `end_date` variables.

Next, the features (input variables) and the target (the variable to predict) for January 2019 were extracted from the filtered dataset. The extracted features were then scaled using the `StandardScaler` to ensure that the data was normalized for better model performance. After scaling, the data was reshaped into a 3D format, which is essential for RNN models to capture temporal dependencies effectively.

The RNN model with an attention mechanism was used to make predictions on the scaled January 2019 data. The predicted values, initially scaled, were inverse-transformed to revert them to their original scale for comparison with the actual values.

Finally, `matplotlib` was used to plot both the actual and predicted close prices for January 2019. The plot displayed actual close prices in blue and predicted close prices in red. The x-axis represented the dates, and the y-axis displayed the close prices. A grid and legend

were included for clarity, and the x-axis labels were rotated for better readability. The title of the plot, "Actual vs Predicted Close Price - January 2019," provided a clear summary of the visualization's content, allowing for an intuitive comparison between the actual and predicted stock prices for each day in January 2019.

```
# Group data by month and year
data['YearMonth'] = pd.to_datetime(data['Date']).dt.to_period('M')
monthly_data = data.groupby('YearMonth').agg({'Close': 'last'})

# Predict on the monthly data
X_monthly = data.groupby('YearMonth').agg({'Open': 'first', 'High': 'max', 'Low': 'min', 'Volume': 'sum',
    'sentiment_score': 'mean', 'Close_lag1': 'last', 'Close_lag2': 'last',
    'Close_rolling_mean': 'last', 'Close_rolling_std': 'last'})

# Convert PeriodIndex to DateTimeIndex
X_monthly.index = X_monthly.index.to_timestamp()

X_monthly_scaled = scaler_X.transform(X_monthly)
X_monthly_scaled = X_monthly_scaled.reshape(X_monthly_scaled.shape[0], X_monthly_scaled.shape[1], 1)
y_pred_monthly = best_model.predict(X_monthly_scaled)
y_pred_monthly = y_pred_monthly[:, -1, 0].reshape(-1, 1)
y_pred_monthly = scaler_y.inverse_transform(y_pred_monthly)

# Plot the actual vs predicted values for monthly data
plt.figure(figsize=(12, 6))

# Convert PeriodIndex to DateTimeIndex for plotting
plt.plot(monthly_data.index.to_timestamp(), monthly_data['Close'], label='Actual Close Price')

plt.plot(monthly_data.index.to_timestamp(), y_pred_monthly, label='Predicted Close Price')
plt.xlabel('Month')
plt.ylabel('Close Price')
plt.title('Actual vs Predicted Close Price - Monthly')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)

# Add month labels to the x-axis
# Use monthly_data.index instead of dates
month_labels = [month.strftime('%Y-%m') for month in monthly_data.index.to_timestamp()]
plt.xticks(monthly_data.index.to_timestamp(), month_labels, rotation=45, ha='right')
plt.tight_layout()

plt.show()
```

Figure 5.122 Monthly Forecast Visualisation of Actual Vs Predicted Close Prices from January 2019 that retrieved from results generated by RNN with Attention Mechanism model

This code generated a monthly forecast visualization using an RNN model with an attention mechanism. The first step was grouping the data by month and year using the YearMonth column, which was derived from the Date column. Aggregation functions were used to compute monthly statistics such as the first open price, the maximum high, the

minimum low, and the last close price for each month, as well as other aggregated features like sentiment score and rolling statistics.

The X\_monthly features were then prepared by scaling them using StandardScaler and reshaping them into a 3D format required by the RNN model to handle temporal sequences effectively. The RNN model, enhanced with the attention mechanism, was used to predict the close prices for each month. The predicted close prices, initially scaled, were inverse-transformed back to their original scale for comparison with the actual monthly close prices.

The matplotlib library was used to plot the actual vs predicted close prices on a monthly basis. The x-axis represented the months, and the y-axis represented the close prices. Actual close prices were plotted in red, while predicted close prices were plotted in blue, providing a clear visual comparison. The month labels were formatted using strftime to display them in "Year-Month" format, and the x-axis labels were rotated for improved readability.

The plot was titled "Actual vs Predicted Close Price - Monthly," and a legend was included to differentiate between the actual and predicted values. The grid provided a structured layout, making it easy to observe the monthly trends and evaluate how closely the model's predictions aligned with the actual data.

```
# prompt: generated the annually version graph

# Group data by month and year
data['YearMonth'] = pd.to_datetime(data['Date']).dt.to_period('M')
monthly_data = data.groupby('YearMonth').agg({'Close': 'last'})

# Predict on the Year data
X_monthly = data.groupby('YearMonth').agg({'Open': 'first', 'High': 'max', 'Low': 'min', 'Volume': 'sum',
    'sentiment_score': 'mean', 'Close_lag1': 'last', 'Close_lag2': 'last',
    'Close_rolling_mean': 'last', 'Close_rolling_std': 'last'})

# Convert PeriodIndex to DateTimeIndex
X_monthly.index = X_monthly.index.to_timestamp()

X_monthly_scaled = scaler_X.transform(X_monthly)
X_monthly_scaled = X_monthly_scaled.reshape(X_monthly_scaled.shape[0], X_monthly_scaled.shape[1], 1)
y_pred_monthly = best_model.predict(X_monthly_scaled)
y_pred_monthly = y_pred_monthly[:, -1, 0].reshape(-1, 1)
y_pred_monthly = scaler_y.inverse_transform(y_pred_monthly)

# Plot the actual vs predicted values for yearly data
plt.figure(figsize=(12, 6))

# Convert PeriodIndex to DateTimeIndex for plotting
plt.plot(monthly_data.index.to_timestamp(), monthly_data['Close'], label='Actual Close Price')

plt.plot(monthly_data.index.to_timestamp(), y_pred_monthly, label='Predicted Close Price')
plt.xlabel('Year')
plt.ylabel('Close Price')
plt.title('Actual vs Predicted Close Price - Yearly')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```

Figure 5.123 Annually Forecast Visualisation of Actual Vs Predicted Close Prices from January 2019 that retrieved from results generated by RNN with Attention Mechanism model

This code snippet created an annual visualization of stock prices using an RNN model with an attention mechanism. The data was first grouped by year and month, with the aggregation functions capturing the necessary statistics such as the first "Open" price, maximum "High" price, and the last "Close" price for each year, alongside other important aggregated features such as sentiment scores and rolling statistics.

The X\_monthly dataset was then transformed using StandardScaler to ensure the data was normalized, and it was reshaped into a 3D format required by the RNN for sequential data processing. The attention mechanism allowed the model to focus on relevant portions of the sequence before making predictions. The model's predictions for the annual close prices were generated and inverse-transformed back to their original scale for easier comparison with actual values.

To visualize the model's performance, matplotlib was used to plot the actual and predicted close prices on an annual basis. The x-axis represented the years, and the y-axis represented the close prices. Actual close prices were plotted in red, while the predicted close prices were shown in blue, providing a clear comparison of model accuracy. The x-axis labels were rotated to ensure proper formatting of the years.

This plot, titled "Actual vs Predicted Close Price - Yearly," allowed a clear understanding of the model's performance over a longer time horizon, making it possible to analyze how well the RNN model with an attention mechanism captured the annual trends in stock prices.



### 5.6.5 Hybrid Modelling

After done the comparison of models' performance evaluations in Chapter 6, SVM has been concluded that was the best overall prediction model for predicting the stock prices regression task. However, in order to further improve the model's predictive power and implemented the strengths of the overall algorithms, here was the hybrid model proposed, which was the integration of SVM with FinBERT, a financial-domain pre-trained language model. This hybrid model allowed to predict not only from numerical stock price data, but also from the sentiment analysis of the financial news, which was containing very important information that could affect the volatility of stock price. So that, the overall performance has been improved and with this type of setup, SVM will continue to handle the regression task, in the same time, The FinBERT contributed by providing very useful sentiment analysis that complements the stock data with financial market sentiment. Here were the steps below to complete the hybrid modelling to build out a robust FinBERT-SVM model:

```
[ ] # save best svm model joblib
import joblib

# Save the best SVM model
filename = 'best_svm_model.joblib'
joblib.dump(best_svm_model, filename)

↔ ['best_svm_model.joblib']
```

Figure 5.124 Used Joblib to save the best predict model, SVM for further hybrid modelling.

The code snippet demonstrated how the best-performing SVM model was saved for future use for the hybrid modelling using the joblib library. First, the joblib library, which is well-suited for efficiently serializing and de-serializing Python objects, was imported. The model was then assigned a filename, 'best\_svm\_model.joblib', to indicate where the model would be stored. Using the joblib.dump() function, the trained SVM model, referenced as best\_svm\_model, was saved to this file. This approach allowed for the easy reuse of the model in later integration stages, such as during deployment or further evaluation, without needing to retrain it each time. By saving the model in this way, the computational cost of retraining was avoided, and the model can be quickly loaded and used whenever required.

```

Load FinBERT
tokenizer = BertTokenizer.from_pretrained('ProsusAI/finbert')
finbert = BertModel.from_pretrained('ProsusAI/finbert')

Function to process text data with FinBERT
def process_text_with_finbert(text):
    inputs = tokenizer(text, return_tensors='pt', truncation=True, padding=True, max_length=512)
    outputs = finbert(**inputs)
    features = outputs.last_hidden_state[:, 0, :].detach().numpy() # Use CLS token features
    return features

Apply FinBERT to extract features from text data
xt_column = 'Headline'
xt_features = np.array([process_text_with_finbert(text) for text in data[xt_column]])

Reshape text features for concatenation
xt_features = text_features.squeeze(axis=1) # Remove extra dimension

Combine numerical features and FinBERT features
combined = np.concatenate([X_numerical, text_features], axis=1)

Split the combined data into train and test sets
train_combined, X_test_combined, y_train_combined, y_test_combined = train_test_split(X_combined, y, test_size=0.2, random_state=42)

/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as s
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100% ██████████ 252/252 [00:00<00:00, 14.3kB/s]
vocab.txt: 100% ██████████ 232k/232k [00:00<00:00, 1.74MB/s]
special_tokens_map.json: 100% ██████████ 112/112 [00:00<00:00, 7.42kB/s]
config.json: 100% ██████████ 758/758 [00:00<00:00, 43.2kB/s]
pytorch_model.bin: 100% ██████████ 438M/438M [00:04<00:00, 126MB/s]

```

Figure 5.125 Used Joblib to save the best predict model,SVM for further hybrid modelling.

The next crucial step in building the hybrid model was the integration of FinBERT for extracting sentiment-related features from textual data, specifically headlines. FinBERT, a specialized variant of BERT designed to handle financial texts, was loaded using the BertTokenizer and BertModel classes from the Hugging Face library. The purpose of incorporating FinBERT was to allow the model to process textual financial data and extract meaningful sentiment features that could be used alongside the numerical stock data for stock price prediction.

A function named `process_text_with_finbert()` was defined to handle this process. It tokenized the input headlines, converting them into tensor representations suitable for the FinBERT model. After running the input through the model, the hidden states of the CLS token were extracted as sentiment features, as this token is designed to capture the overall meaning of a sequence in BERT-based models. These features were then transformed into numpy arrays for further processing.



Once the sentiment features were extracted, they were reshaped to ensure compatibility with the numerical data that had been used for stock predictions. The sentiment features from the headlines were combined with numerical stock features (e.g., prices, volume, technical indicators) to form a complete dataset that included both types of information. This combined dataset was essential for the hybrid model, as it allowed the model to leverage both the market data and sentiment from news headlines to predict stock price movements.

Finally, the combined dataset was split into training and testing sets. This ensured that the model could learn from both the stock and sentiment data during training and be evaluated on unseen data during testing. This integration of numerical and textual features was a key step in building a robust hybrid model capable of capturing both the quantitative and qualitative factors influencing stock prices.

```
[5] # Load the saved SVM model
loaded_svm_model = joblib.load('/content/best_svm_model.joblib')
```

Figure 5.126 Loaded the previously trained SVM model

In this step, building the hybrid model involved loading the saved SVM model. This step was crucial for using the previously trained SVM model alongside FinBERT-extracted features.

Once the SVM model was loaded back into the environment, it was integrated with the text features extracted using FinBERT.

```
# Create an SVM model with a pipeline for combined features
svm_pipeline_combined = make_pipeline(StandardScaler(), SVR(kernel='rbf', gamma='auto', C=10, epsilon=0.2))

# Train the model with combined features
svm_pipeline_combined.fit(X_train_combined[:, :9], y_train_combined) # Use only the first 9 numerical features

# Evaluate on the training set with combined features
y_train_pred_combined = svm_pipeline_combined.predict(X_train_combined[:, :9]) # Use only the first 9 numerical features

# Calculate performance metrics on the training set with combined features
mse_train_combined = mean_squared_error(y_train_combined, y_train_pred_combined)
r2_train_combined = r2_score(y_train_combined, y_train_pred_combined)
rmse_train_combined = np.sqrt(mse_train_combined)
mae_train_combined = mean_absolute_error(y_train_combined, y_train_pred_combined)

# Print training performance metrics with combined features
print(f"Training Set Performance with Combined Features:")
print(f"Mean Squared Error (MSE): {mse_train_combined}")
print(f"R-squared (R²): {r2_train_combined}")
print(f"Root Mean Squared Error (RMSE): {rmse_train_combined}")
print(f"Mean Absolute Error (MAE): {mae_train_combined}")
```

Figure 5.127 Model training and performance evaluation with combined features

The next step in building the hybrid model involved creating an SVM model pipeline that incorporated both numerical features and FinBERT-extracted text features. This was done by defining a pipeline with `StandardScaler()` for feature scaling and `SVR()` with an RBF kernel as the regression model. The model was trained on a combined dataset that included both the first nine numerical features from traditional stock data and the FinBERT-extracted features from text data, such as news headlines.

The model was then evaluated on the training set, where the prediction was based solely on the first nine numerical features. Several performance metrics were calculated to assess the model's effectiveness: Mean Squared Error (MSE) measured the average squared difference between the actual and predicted values, R-squared ( $R^2$ ) evaluated how well the model fit the data, Root Mean Squared Error (RMSE) provided the error in the same unit as the stock prices, and Mean Absolute Error (MAE) captured the average magnitude of prediction errors.

These metrics were printed to allow for a comparison between using only numerical data versus using the hybrid approach that included both numerical and FinBERT-extracted features, providing insights into the benefit of integrating text data into the stock price prediction process.

```
# Create a TimeSeriesSplit object
tscv = TimeSeriesSplit(n_splits=5)
# Perform cross-validation with combined features
cv_scores_combined = cross_val_score(svm_pipeline_combined, X_combined[:, :9], y, cv=tscv, scoring='neg_mean_squared_error')

# Calculate the average MSE with combined features
avg_mse_combined = -np.mean(cv_scores_combined)
avg_rmse_combined = np.sqrt(avg_mse_combined)

# Print the average MSE with combined features
print(f"Average MSE with TimeSeriesSplit and Combined Features: {avg_mse_combined}")
print(f"Average RMSE with TimeSeriesSplit and Combined Features: {avg_rmse_combined}")
```

Figure 5.128 Time SeriesSplit Cross Validation with combined features

In this step, the cross-validation process was carried out using the combined numerical features and FinBERT-extracted features. First, a `TimeSeriesSplit` object was created with 5

splits to ensure the dataset was divided into sequential folds, preserving the time-based order of the data, which is crucial for temporal predictions like stock prices. The `cross_val_score` function was then employed to perform cross-validation on the SVM model that had been trained using both feature types. This function calculated the negative mean squared error (`neg_mean_squared_error`) across the folds to evaluate the model's performance.

Once the cross-validation process was completed, the average Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) were calculated by taking the mean of the cross-validation scores. These metrics were then printed to assess the overall performance of the hybrid model, offering a measure of how well the model predicted stock prices using the combined feature set across different time periods. This cross-validation step was essential to confirm the model's robustness and ability to generalize well across unseen data.

```

# Define the parameter grid for combined features
param_dist_combined = {
    'svr__C': [0.1, 1, 10, 100],
    'svr__gamma': ['scale', 'auto'],
    'svr__epsilon': [0.1, 0.2, 0.5],
    'svr__kernel': ['rbf', 'linear']
}

# Set up RandomizedSearchCV to find the best hyperparameters with combined features
random_search_combined = RandomizedSearchCV(svm_pipeline_combined, param_distributions=param_dist_combined, n_iter=20, cv=tscv, scoring='neg_mean_squared_error', verbose=1, n_jobs=-1, random_state=42)
random_search_combined.fit(X_train_combined[:, :9], y_train_combined) # Use only the first 9 numerical features

# Get the best model with combined features
best_svm_model_combined = random_search_combined.best_estimator_

# Print the best parameters with combined features
print("Best hyperparameters with combined features:", random_search_combined.best_params_)

```

Figure 5.129 Hyperparameters tuning and fine-tuning for the hybrid model

In this step, hyperparameter tuning was performed using a combined feature set that included both numerical features and FinBERT-extracted text features. A parameter grid was defined (`param_dist_combined`) that specified a range of hyperparameters for the Support Vector Regressor (SVR). These hyperparameters included C, gamma, epsilon, and the kernel type (rbf or linear), each having a range of values to explore.

To find the optimal combination of hyperparameters, `RandomizedSearchCV` was employed with a 5-fold time series cross-validation (`cv=tscv`). The search iterated over 20 different combinations of hyperparameters (`n_iter=20`) from the grid to evaluate their

performance using the negative mean squared error (`neg_mean_squared_error`) as the scoring metric. The random state was set to ensure reproducibility.

After completing the search, the best-performing SVM model was selected and stored in the `best_svm_model_combined` variable. The optimal hyperparameters for the combined feature set were printed to provide insights into which configuration yielded the best results. This tuning process was critical in refining the SVM model to achieve higher accuracy when using both numerical and textual data features for stock price prediction for building the robust hybrid model.

```
# Evaluate on the test set with combined features
y_test_pred_combined = best_svm_model_combined.predict(X_test_combined[:, :9]) # Use only the first 9 numerical features

# Calculate performance metrics on the test set
mse_test_combined = mean_squared_error(y_test_combined, y_test_pred_combined)
r2_test_combined = r2_score(y_test_combined, y_test_pred_combined)
rmse_test_combined = np.sqrt(mse_test_combined)
mae_test_combined = mean_absolute_error(y_test_combined, y_test_pred_combined)

# Print test performance metrics
print(f"Test Set Performance with Hybrid Model (SVM + FinBERT):")
print(f"Mean Squared Error (MSE): {mse_test_combined}")
print(f"R-squared (R2): {r2_test_combined}")
print(f"Root Mean Squared Error (RMSE): {rmse_test_combined}")
print(f"Mean Absolute Error (MAE): {mae_test_combined}")
```

Figure 5.131 Model Test-set Performance Evaluation

In this step, the final evaluation of the hybrid model, which integrated SVM with FinBERT features, was conducted using the test dataset. The model utilized the optimal hyperparameters identified in the tuning process to predict stock prices on the combined test dataset (`X_test_combined`), specifically focusing on the first 9 numerical features. Several key performance metrics were calculated to assess the model's predictive ability, including the Mean Squared Error (MSE), which quantified the average squared difference between actual and predicted values, and R-squared ( $R^2$ ), indicating the proportion of variance explained by the model. Additionally, the Root Mean Squared Error (RMSE) provided a measure of error in the same units as the original data, while the Mean Absolute Error (MAE) represented the average magnitude of prediction errors. These metrics were printed to determine how well the

hybrid model (combining SVM and FinBERT-extracted text features) performed, offering a comprehensive view of its predictive accuracy on the test set.

```
[17] #Daily
# Convert the 'Date' column to datetime format
data['Date'] = pd.to_datetime(data['Date'], format='%d/%m/%Y')

# Filter data for January 2019
jan_2019_data = data[(data['Date'] >= '2019-01-01') & (data['Date'] <= '2019-01-31')]

# Get the actual and predicted Close prices for January 2019
y_actual_jan_2019 = jan_2019_data['Close'].values
X_jan_2019 = jan_2019_data[numerical_columns].values
y_pred_jan_2019 = loaded_svm_model.predict(X_jan_2019)

# Get the dates for January 2019
dates_jan_2019 = jan_2019_data['Date'].values

# Plot the actual vs predicted Close prices for January 2019
plt.figure(figsize=(12, 6))
plt.plot(dates_jan_2019, y_actual_jan_2019, label="Actual Close Prices")
plt.plot(dates_jan_2019, y_pred_jan_2019, linestyle='--', label="Predicted Close Prices")
plt.xlabel("Date")
plt.ylabel("Close Price")
plt.title("Actual vs Predicted Close Prices (January 2019)")
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.show()
```

Figure 5.132 Daily Forecast Visualisation of Actual Vs Predicted Close Prices in January 2019 that retrieved from results generated by FinBERT-SVM hybrid model

In this visualisation step, the FinBERT-SVM model's daily performance for January 2019 was evaluated. First, the 'Date' column was converted to a datetime format to facilitate filtering. The dataset was then filtered to include only the data between January 1, 2019, and January 31, 2019. The actual closing stock prices for January 2019 were extracted from the dataset, and the loaded\_svm\_model was used to predict the closing prices for the same period based on the numerical features.

The actual and predicted values were then plotted on a graph, with dates on the x-axis and closing prices on the y-axis. The graph compared the actual closing prices (displayed as a



solid line) to the predicted closing prices (displayed as a dashed line), helping to visually assess the model's accuracy in predicting daily stock prices. The plot was customized with labels for the x-axis (Date), y-axis (Close Price), and a title that highlighted the period of the prediction, in this case, "January 2019." A legend was included to differentiate between actual and predicted values, and the x-axis labels were rotated for better readability.

```
[ ] # Monthly
# Create the results_df DataFrame with a DatetimeIndex
results_df = pd.DataFrame({'Actual': y_test_combined, 'Predicted': y_test_pred_combined}, index=data['Date'][::-len(y_test_combined):])

results_df_monthly = results_df.resample('M').mean().dropna() # Resample and drop months with no data
plt.figure(figsize=(14, 7))
plt.plot(results_df_monthly.index, results_df_monthly['Actual'], label='Actual Close Prices', linewidth=2)
plt.plot(results_df_monthly.index, results_df_monthly['Predicted'], linestyle='--', label='Predicted Close Prices', linewidth=2)
plt.title('Month by Month Actual vs Predicted Close Prices with Hybrid Model (SVM + FinBERT)')
plt.xlabel('Month')
plt.ylabel('Close Price')
plt.xticks(results_df_monthly.index, results_df_monthly.index.strftime('%b %Y'), rotation=45) # Format as "Month Year"
plt.legend()
plt.grid(True)
plt.show()
```

Figure 5.133 Monthly Forecast Visualisation of Actual Vs Predicted Close Prices from January 2019 that retrieved from results generated by FinBERT-SVM hybrid model

In this code snippet, the hybrid model's (SVM + FinBERT) monthly performance was evaluated. First, a DataFrame `results_df` was created, which contained both the actual and predicted close prices along with their corresponding dates. The DataFrame was indexed by the 'Date' column to ensure that time-series analysis could be conducted.

Next, the data was resampled on a monthly basis using the `resample('M')` method, which grouped the daily data into monthly averages. This resampling helped reduce noise from daily fluctuations and focused on broader trends in the data. Any months without data were dropped to ensure the graph was clean.

A plot was then generated to visually compare the actual and predicted closing prices month by month. The actual close prices were plotted as a solid line, while the predicted close prices were plotted as a dashed line. The x-axis represented months (formatted as "Month Year"), and the y-axis represented the closing prices. Labels, a title, and a legend were added for clarity, and the x-axis tick labels were rotated to improve readability. The plot allowed for a clear comparison of how well the hybrid model predicted stock prices over each month, highlighting the model's ability to capture broader trends in the data.

```
[24] # Annually
# Create the results_df DataFrame with a DatetimeIndex
results_df = pd.DataFrame({'Actual': y_test_combined, 'Predicted': y_test_pred_combined}, index=data['Date'][-len(y_test_combined):])
results_df_yearly = results_df.resample('Y').mean().dropna()

# Plot the visualization
plt.figure(figsize=(14, 7))
plt.plot(results_df_yearly.index, results_df_yearly['Actual'], label='Actual Close Prices', linewidth=2)
plt.plot(results_df_yearly.index, results_df_yearly['Predicted'], linestyle='--', label='Predicted Close Prices', linewidth=2)
plt.title('Year by Year Actual vs Predicted Close Prices with Hybrid Model (SVM + FinBERT)')
plt.xlabel('Year')
plt.ylabel('Close Price')
plt.xticks(results_df_yearly.index, results_df_yearly.index.strftime('%Y'), rotation=45) # Format as "Year"
plt.legend()
plt.grid(True)
plt.show()
```

Figure 5.134 Annually Forecast Visualisation of Actual Vs Predicted Close Prices from January 2019 that retrieved from results generated by FinBERT-SVM hybrid model

In this step, the hybrid model's (SVM + FinBERT) performance was evaluated on an annual basis. Similar to the monthly evaluation, a DataFrame `results_df` was created with the actual and predicted close prices alongside their corresponding dates. This DataFrame was then resampled by year using the `resample('Y')` method to compute the annual averages of the stock prices, allowing a higher-level view of the stock price trends over the years. Any years without data were dropped from the DataFrame.

The visualization was generated with the resampled yearly data. A line plot was created to compare the actual close prices against the predicted close prices for each year. The actual close prices were plotted as a solid line, while the predicted close prices were plotted as a dashed line. The x-axis represented years (formatted as "Year"), and the y-axis represented the closing prices. Labels, a title, and a legend were included for clarity, and the x-axis tick labels were rotated to make the year labels easier to read. This visualization provided insights into how well the hybrid model captured long-term stock price trends on a yearly basis.

```
[18] #save the hybrid model (FinBERT-SVM)

# Save the best SVM model with combined features
filename_combined = 'best_hybrid_model.joblib'
joblib.dump(best_svm_model_combined, filename_combined)
```

Figure 5.135 Saved the FinBERT-SVM hybrid model

In the final step, the best-performing hybrid model (FinBERT-SVM) was saved using the joblib library. This involved saving the SVM model that had been trained with the combined numerical and textual (FinBERT) features. The model was saved to a file named `best_hybrid_model.joblib`. By saving the model, it could be reloaded and reused in the future for making predictions without needing to retrain the model. This process ensured that the hybrid model, which integrated both numerical stock data and text-based sentiment analysis from FinBERT, could be efficiently preserved for future use or further evaluation.

## 5.7 Summary

This project was developed in a Google Colab environment using Python, focusing on predicting stock prices by combining numerical stock data with sentiment analysis from stock news headlines. The data used was from Apple Inc., merging stock price information with sentiment analysis extracted from relevant headlines.

Initially, the stock price and sentiment data were cleaned, processed, and merged into a final dataset comprising 677 rows. Sentiment scores were extracted using the FinBERT model, adding features such as positive, neutral, and negative sentiment classifications for each headline.

Several predictive models were developed, including Support Vector Machine (SVM) for regression, Long Short-Term Memory (LSTM) networks to capture temporal patterns, and Convolutional Neural Networks (CNN) for identifying complex patterns. Additionally, an RNN with an attention mechanism was implemented, allowing the model to focus on key time steps in the data sequence.



After thorough evaluation, the SVM model emerged as the best-performing model for predicting stock prices, leveraging the numerical stock data effectively. A hybrid model was also created by combining the predictive strength of SVM with sentiment features from FinBERT, further enhancing accuracy. The SVM-FinBERT hybrid model demonstrated improved performance, solidifying SVM as the most effective model for stock price prediction.

In summary, integrating numerical and sentiment data for stock price prediction resulted in higher accuracy compared to using either data type alone, with the SVM and SVM-FinBERT hybrid model yielding the most accurate results.

## **5.8 Implementation issues and challenges**

### **Data Quality and Alignment**

A significant issue encountered was the challenge of aligning the stock price data with the sentiment data from headlines. Stock prices typically follow a regular daily schedule, whereas news headlines can be published sporadically throughout the day, creating mismatches in timing. For example, a positive news headline published late in the day could inaccurately be aligned with an already-declining stock price, leading to distorted sentiment analysis. The inconsistency in data frequency between these two sources results in a misalignment that could negatively affect model performance, making predictions less reliable and accurate.

### **Noisy Text Data**

News headlines often contain noise in the form of irrelevant, ambiguous, or misleading information. Headlines may not always directly reflect stock-related news, but could reference broader market trends, global events, or even company promotions, which can skew sentiment analysis results. Moreover, headlines that include sarcasm, technical language, or emotionally charged but contextually unrelated content further complicate the sentiment scoring process. This ambiguity can result in inaccurate sentiment classification, thereby impairing the prediction model's ability to make reliable forecasts based on headline data.

### **Overfitting of Complex Models**

Complex models, especially those used in time-series prediction such as LSTM, CNN, or RNNs with attention mechanisms, face the risk of overfitting. When these models become too finely tuned to the specific patterns in the training data, they lose their ability to generalize to unseen data. This issue is exacerbated in stock market prediction, where fluctuations and trends may not follow consistent patterns. As a result, a model that performs well on training data might fail to make accurate predictions in real-world stock price movements, leading to unreliable forecasting.

### **Integrating Numerical and Textual Features**

Integrating numerical stock data with sentiment-based textual features is inherently challenging due to the different nature of these data types. Stock prices are continuous and time-sensitive, whereas textual sentiment analysis involves transforming qualitative information into quantitative scores. This integration poses a risk of imbalance, where either the stock price data or the sentiment data could dominate the model's decision-making process. If the sentiment scores do not accurately capture the influence of news on stock prices or if the stock data overshadows sentiment inputs, the predictive model might fail to appropriately weigh both data sources, resulting in suboptimal predictions.

### **Feature Engineering Complexity**

With the integration of sentiment analysis and numerical data, one of the major challenges is determining which features contribute the most to the model's accuracy. The combination of stock data with FinBERT sentiment features, such as positive, negative, or neutral scores, requires thoughtful feature selection and engineering. It can be difficult to understand how each feature, or combination of features, affects the final prediction. Inadequate feature selection could lead to irrelevant data cluttering the model or important information being overlooked, both of which impact prediction accuracy and model performance.

## Chapter 6: System Evaluation

### 6.1 Model Performance

#### 6.1.1 Performance Matrices

##### *Mean Squared Error (MSE)*

The Mean Squared Error (MSE) is one of the most widely used metrics for evaluating regression models, particularly in stock price prediction. It measures the average squared difference between the actual stock prices and the predicted stock prices. Since the errors are squared, MSE gives more weight to larger errors, making it particularly sensitive to large deviations or outliers in the dataset.

##### Importance in Stock Price Prediction

In stock prediction, minor deviations between the predicted and actual prices may be acceptable, but larger deviations can lead to significant losses for investors or traders. The squaring of the errors in MSE ensures that models that make larger mistakes are penalized more heavily. For instance, if a model incorrectly predicts a substantial stock drop or rise, the resulting error will significantly impact the MSE, highlighting that the model needs improvement for critical predictions.

##### **Formula:**

Where:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- $y_i$  represents the actual stock price at time  $i$ .
- $\hat{y}_i$  represents the predicted stock price at time  $i$ .
- $n$  is the total number of predictions made.

**Interpretation:**

- Lower MSE values indicate better model performance, as they signify smaller average errors between predicted and actual prices.
- Higher MSE values indicate that the model's predictions deviate significantly from actual values, suggesting the need for improvement.
- Since MSE squares the errors, it is more sensitive to large errors, making it a preferred metric when avoiding big prediction mistakes is critical in the context of stock trading or investment.

*Root Mean Squared Error (RMSE)*

The **Root Mean Squared Error (RMSE)** is the square root of the MSE. RMSE has the advantage of expressing the error in the same units as the target variable (in this case, stock prices). This makes RMSE more interpretable than MSE, as the result is in the same scale as the original data, allowing for a more intuitive understanding of the error magnitude.

Importance for Stock Price Prediction

RMSE provides an easy-to-interpret measure of how far off a model's predictions are from actual stock prices on average. Since the stock prices are usually in monetary units, RMSE provides a clear picture of how far, in dollars, the predicted prices are from the actual prices. This is particularly important when investors want to quantify the risk or error associated with predictions in terms of actual price movements.

**Formula:**

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

**Where:**

- $y_i$  is the actual stock price at time  $i$ .
- $\hat{y}_i$  is the predicted stock price at time  $i$ .
- $n$  is the number of predictions

**Interpretation:**

- Lower RMSE values signify a better-performing model. For example, an RMSE of 2 means that, on average, the model's predictions are off by \$2 from the actual stock price.
- Higher RMSE values imply larger deviations between predictions and actual prices, indicating that the model might not generalize well to unseen data.
- RMSE is particularly useful when you want a metric that directly relates to the units of the target variable (dollars in stock price prediction).

*Mean Absolute Error (MAE)*

The Mean Absolute Error (MAE) calculates the average absolute difference between the predicted and actual stock prices. Unlike MSE and RMSE, MAE does not square the errors, making it less sensitive to outliers. Each prediction error is treated equally, providing a straightforward interpretation of how far off predictions are, on average.

Importance for Stock Price Prediction

In many stock price prediction scenarios, we might be more interested in knowing the average magnitude of prediction errors, rather than disproportionately penalizing larger errors (as MSE does). MAE provides a simple measure of the average error without being overly influenced by extreme outliers, making it a practical metric when stock price predictions tend to have fewer large deviations.

**Formula:**

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

**Where:**

- $y_i$  is the actual stock price at time  $i$ .
- $\hat{y}_i$  is the predicted stock price at time  $i$ .
- $n$  is the number of predictions.

Interpretation:

- Lower MAE values indicate that, on average, the predictions are closer to the actual stock prices, suggesting a more accurate model.
- Higher MAE values suggest that the model's predictions deviate significantly from the actual prices, making it less reliable.
- MAE is particularly useful when you want to measure the average magnitude of errors without worrying about outliers or large mistakes affecting the results disproportionately.

### *R-Squared ( $R^2$ or Coefficient of Determination)*

R-squared ( $R^2$ ), also known as the coefficient of determination, measures the proportion of the variance in the actual stock prices that is predictable from the independent variables (such as Volume, Open, Close, sentiment score, etc.). In simpler terms,  $R^2$  evaluates how well the features (independent variables) used in the model explain the variation in the target variable (stock price).  $R^2$  is typically expressed as a percentage.

### Importance for Stock Price Prediction

In stock price prediction, the  $R^2$  value gives an indication of how well the model fits the actual data. A high  $R^2$  value indicates that the model is effectively capturing the variability in the stock price data, meaning that the model's predictions are closely aligned with the actual price movements. This is crucial for building models that can explain why stock prices change and predict future changes with reasonable confidence.

**Formula:**

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

Where:

- $y_i$  is the actual stock price at time  $i$ .
- $\hat{y}_i$  is the predicted stock price at time  $i$ .
- $\bar{y}$  is the mean of the actual stock prices.
- $n$  is the number of predictions.

## 6.1.2 Model Train Result Evaluation

Model	Kernel /Optimizer	Hypeprparameters	Root Mean Square Error (RMSE)	Mean Square Error (MSE)	Mean Absolute Error (MAE)	R Square (r2)
LSTM	Adam	Units: 50, Dropout: 0.2, L2=0.001	4.980	24.80	3.686	0.9896
RNN with Attention Mechanism	Adam	Units: 64, Dropout: 0.2,	2.581	2.581	1.859	0.9972
CNN	Adam	Filters: 32, Kernel size: 3, Pool size: 2, learning_rate_0.001	3.251	10.57	2.806	0.9956
SVM	RBF	Gamma: auto, C: 10, Epsilon: 0.2,	9.696	94.01	5.587	0.9558

### Analysis of Training Performance Matrices Results

#### *Mean Squared Error (MSE)*

The Mean Squared Error (MSE) plays a critical role in evaluating how well each model predicts stock prices by measuring the average of the squared differences between actual and predicted values. In stock price prediction, a lower MSE value indicates that the model's predictions are more accurate, while a higher MSE suggests the model struggles to capture the underlying trends in the data. From the results, the RNN with Attention model achieves the lowest MSE of 2.581, indicating that it is particularly adept at predicting stock prices with minimal error. This low MSE shows that the RNN with Attention can capture both short-term and long-term relationships within the stock data, reducing the frequency and magnitude of large prediction errors. In contrast, the SVM model has the highest MSE of 94.01, suggesting that it struggles

to model the complex, non-linear relationships present in the stock data, leading to larger deviations between predicted and actual prices. The LSTM model, with an MSE of 24.80, performs moderately well, reflecting its ability to capture long-term dependencies in stock prices but still allowing for occasional large errors. The CNN model, with an MSE of 10.57, performs better than LSTM, suggesting that it effectively captures local patterns and short-term movements in stock prices, leading to fewer large errors. Overall, a lower MSE value is desirable in stock price prediction as it helps minimize large deviations that can have significant financial impacts, and in this evaluation, the RNN with Attention model proves to be the best performer in this regard.

#### *Root Mean Squared Error (RMSE)*

The Root Mean Squared Error (RMSE) offers a more interpretable version of MSE by providing the average error in the same units as the target variable—stock prices, in this case. This makes RMSE particularly useful in stock price prediction, as it allows investors and analysts to understand the typical size of the prediction error in dollar terms. The RNN with Attention model has the lowest RMSE of 2.581, meaning that, on average, its predictions deviate from actual stock prices by approximately \$2.58. This small error highlights the model's strong predictive power and its ability to closely match the actual stock price movements, which is critical in a volatile market where small deviations can result in significant financial implications. The CNN model also performs well, with an RMSE of 3.251, suggesting that its predictions are off by about \$3.25 on average. This relatively low RMSE indicates that CNN is effective at capturing short-term stock price fluctuations and patterns. On the other hand, the LSTM model has an RMSE of 4.980, showing a higher average prediction error of nearly \$5, which may reflect its struggle to capture intricate short-term price movements despite its strength in modeling long-term dependencies. The SVM model, with an RMSE of 9.696, shows the largest average error, indicating that its predictions deviate significantly from actual stock prices. This high RMSE highlights the limitations of the SVM in stock price prediction, particularly when handling the volatility and non-linear relationships often present in financial data. In summary, RNN with Attention outperforms other models by providing the smallest error in dollar terms, making it the most reliable for investors and analysts concerned with accurate stock price predictions



*Mean Absolute Error (MAE)*

The Mean Absolute Error (MAE) measures the average magnitude of errors between predicted and actual stock prices, providing a straightforward interpretation without emphasizing larger errors as MSE does. In stock price prediction, a lower MAE signifies a model that consistently predicts prices with minimal deviations, while a higher MAE suggests less accurate, more error-prone predictions. The RNN with Attention model achieves the lowest MAE of 1.859, meaning that, on average, its predictions are off by less than \$2. This highlights the model's ability to make consistently accurate predictions, reinforcing its strength in capturing both short-term and long-term stock price movements. The CNN model, with an MAE of 2.806, also shows strong performance, as its predictions deviate by about \$2.81 on average. This suggests that CNN is particularly effective in detecting local patterns and making accurate short-term predictions, although it still allows for slightly more error compared to the RNN model. The LSTM model records an MAE of 3.686, indicating a higher average error of around \$3.69, reflecting the model's occasional difficulty in handling the finer details of stock price movements, especially in volatile markets. Meanwhile, the SVM model performs the worst, with an MAE of 5.587, meaning its predictions are, on average, off by nearly \$5.59. This large error reflects the model's inability to consistently capture the complex interactions between stock prices, volume, and sentiment, resulting in less reliable predictions. In conclusion, the RNN with Attention model demonstrates the highest consistency in predicting stock prices, with the smallest average deviation, making it the preferred model for scenarios where accurate predictions are critical.

*R-Squared ( $R^2$ )*

R-Squared ( $R^2$ ), or the coefficient of determination, evaluates how well the model explains the variance in stock prices. In stock price prediction, a high  $R^2$  value indicates that the model effectively captures the underlying factors driving price changes, while a lower  $R^2$  value suggests that the model fails to account for important influences on stock prices. The RNN with Attention model achieves an  $R^2$  value of 0.9972, meaning that it explains 99.72% of the variance in stock prices, making it the most comprehensive model in terms of capturing the relationships between input features and price movements. This high  $R^2$  value demonstrates the RNN's ability to focus on critical time steps, such as large market swings or significant news events, and incorporate them into its predictions. Similarly, the CNN model performs exceptionally well, with an  $R^2$  of 0.9956, indicating that it explains 99.56% of the variance. This suggests that CNN is highly effective in capturing short-term trends and price patterns,

which are crucial in stock market prediction. The LSTM model has an  $R^2$  of 0.9896, explaining 98.96% of the variance. While still a strong result, the slightly lower  $R^2$  suggests that LSTM may not capture short-term fluctuations as effectively as the RNN or CNN models. Finally, the SVM model records the lowest  $R^2$  of 0.9558, indicating that it explains about 95.58% of the variance in stock prices. While this is still a reasonable performance, it falls short compared to the deep learning models, highlighting SVM's limitations in handling complex, non-linear relationships in the stock market. Overall, RNN with Attention emerges as the top performer in terms of explaining stock price variability, making it the best choice for comprehensive stock price prediction.

### *Summary*

In analyzing the performance of these models using MSE, RMSE, MAE, and  $R^2$ , the RNN with Attention model consistently outperforms the other models, making it the most suitable for stock price prediction during model training. It not only minimizes large errors but also provides consistently accurate predictions, explaining nearly all the variance in stock prices. The CNN model follows closely behind, particularly excelling in short-term prediction accuracy. The LSTM model, while effective in modeling long-term dependencies, struggles somewhat with short-term price movements. Finally, the SVM model underperforms across all metrics, demonstrating its limitations in handling the complex, non-linear dynamics of stock price prediction. Based on these results, RNN with Attention and CNN are the most reliable models for stock market forecasting during model training phase.

## 6.1.3 Model Cross Validation Evaluation

$$\text{Average RMSE} = \frac{1}{K} \sum_{k=1}^K \text{RMSE}_k$$

where K is the total number of folds in cross validations

Model	Number of Splits (TimeSeries Splir)	AverageRoot Mean Square Error (RMSE)	Average Mean Square Error (MSE)	AverageMean Absolute Error (MAE)	Average R Square (r2)
LSTM	5	0.001595	4.87	0.9901	0.9901
RNN with Attention Mechanism	5	2.58	6.66	0.9972	0.9972
CNN	5	3.15	10.37	0.9956	0.9956
SVM	5	38.19	1458.35	0.9558	0.9558

#### *Analysis of Cross Validation Results*

The cross-validation results provide a detailed look at the performance of the different models when applied to the stock price prediction task. Each model underwent 5-fold cross-validation, ensuring robustness in the evaluation process by testing the model's ability to generalize across different subsets of the data.

The LSTM model performs well in capturing stock price movements, particularly long-term dependencies. With an average RMSE of 4.87 and an MAE of 3.74, the LSTM model indicates a reasonable error margin when predicting stock prices, with deviations from the actual prices being around \$4.87 on average. However, while the R<sup>2</sup> value of 0.9901 indicates that the model can explain around 99% of the variance in stock prices, the higher MAE suggests

that LSTM might struggle slightly with short-term fluctuations, as it tends to focus more on long-term trends in the data.

On the other hand, the RNN with Attention Mechanism emerges as the top-performing model across all evaluation metrics. With a low RMSE of 2.58 and a minimal MAE of 1.86, this model consistently makes smaller errors across different time steps, proving its capability to predict stock prices with remarkable accuracy. The  $R^2$  value of 0.9972 shows that it explains almost 99.72% of the variance in stock prices, making it the most reliable model in terms of capturing the intricate patterns of stock market movements. The attention mechanism allows the model to focus on crucial time steps, enhancing both its short-term and long-term prediction capabilities.

The CNN model also shows strong performance, particularly in short-term prediction accuracy. With an RMSE of 3.15 and an MAE of 2.54, CNN manages to capture local patterns and short-term trends in stock price movements effectively. The  $R^2$  value of 0.9956 indicates that the model explains a significant portion of the variance in stock prices, making it a suitable choice for capturing short-term fluctuations. CNN's ability to extract important features from sequential data makes it a strong contender for stock price prediction tasks.

In contrast, the SVM model struggles to match the performance of the deep learning models. With a much higher RMSE of 38.19 and an MAE of 5.59, the SVM model consistently exhibits larger deviations from actual stock prices, indicating that it is less suitable for time-series stock price prediction. The MSE of 1458.35 further highlights the large errors made by the SVM model. Although the  $R^2$  value of 0.9558 suggests that it captures around 95.58% of the variance in stock prices, this is still significantly lower than the deep learning models. SVM's difficulty in capturing complex and non-linear relationships in the time-series data explains its underperformance in this scenario.

## 6.1.4 Model Test Set Evaluation

Model	Kernel /Optimizer	Hyperparameters	Root Mean Square Error (RMSE)	Mean Square Error (MSE)	Mean Absolute Error (MAE)	R Square (r <sup>2</sup> )
LSTM	nadam	Units: 200, Dropout: 0.1, Learning_rate:0.0005	2.321	5.389	1.584	0.9978
RNN with Attention Mechanism	Rmsprop	Units: 96, Dropout: 0.27, learning_rate: 0.0005	2.831	8.017	2.051	0.9968
CNN	Adam	Filters: 128, Kernel size: 5, Pool size: 2, Learning_rate:0.0011	3.251	10.57	2.806	0.9955
SVM	linear	Gamma: auto, C: 100, Epsilon: 0.2,	0.8383	0.7029	0.6819	0.9981

*Analysis of Test set Evaluation*

The LSTM model emerges as the strongest overall performer, achieving the lowest error metrics among the deep learning models, with an RMSE of 2.321 and an MAE of 1.584. Its high R<sup>2</sup> value of 0.9978 confirms that it effectively captures nearly all the variance in stock price movements, making it an excellent choice for capturing both short- and long-term dependencies. The well-tuned hyperparameters, including 200 units, 0.1 dropout, and a learning rate of 0.0005, allow the model to generalize well, especially for long-term trend forecasting.

The RNN with Attention Mechanism also performs well, though its error metrics—RMSE of 2.831 and MAE of 2.051—are slightly higher than those of LSTM. The attention mechanism provides a significant advantage by helping the model focus on important time steps, but the model still struggles with short-term fluctuations. Its  $R^2$  value of 0.9968 shows that it captures a large proportion of the variance in stock prices, but it falls short of the LSTM in terms of overall accuracy.

CNN, on the other hand, performs effectively in short-term price predictions, with an RMSE of 3.251 and MAE of 2.806. Its ability to detect local patterns in the stock price data makes it useful for short-term forecasting, but its overall performance is not as strong as LSTM or RNN with Attention when dealing with more complex, long-term price movements. The  $R^2$  value of 0.9955 still reflects a high level of accuracy, but the higher error metrics suggest CNN may need further tuning to capture more intricate patterns in stock prices.

Surprisingly, the SVM model delivers the lowest overall error metrics, with an RMSE of 0.8383 and MAE of 0.6819. Additionally, the  $R^2$  value of 0.9981 indicates that the SVM model explains more variance in stock prices than any of the deep learning models. This strong performance is unexpected, as SVM is generally considered less suited for handling the complex, non-linear nature of stock price data. However, in this particular evaluation, the combination of RBF kernel, Gamma set to auto, and  $C = 100$  appears to have optimized the model's performance significantly. It's important to note that SVM's strong performance in this test may be dataset-specific, and it might not generalize as well to other time-series data as the deep learning models. Therefore, Svm model was the best model selected for the futher hybrid modelling.

## 6.1.5 Final Hybrid Model Performance Evaluation

Model	Kernel /Optimizer	Hyperparameters	Root Mean Square Error (RMSE)	Mean Square Error (MSE)	Mean Absolute Error (MAE)	R Square (r <sup>2</sup> )
FinBERT-SVM	linear	Gamma=auto,svr =0.2,C=100	0.8383	0.7029	0.6819	0.9981

*Conclusion*

The Final Hybrid Model Performance Evaluation for the FinBERT-SVM model reveals exceptional accuracy in stock price prediction. This hybrid approach combines FinBERT's sentiment analysis capabilities with the predictive power of a Support Vector Machine (SVM). The model's performance, as reflected in its key metrics, demonstrates strong predictive capabilities. With a Root Mean Squared Error (RMSE) of 0.8383 and a Mean Squared Error (MSE) of 0.7029, the model shows that its average prediction error is less than \$1, indicating highly precise predictions. Additionally, the Mean Absolute Error (MAE) of 0.6819 further confirms its accuracy, as the average error in prediction remains very small.

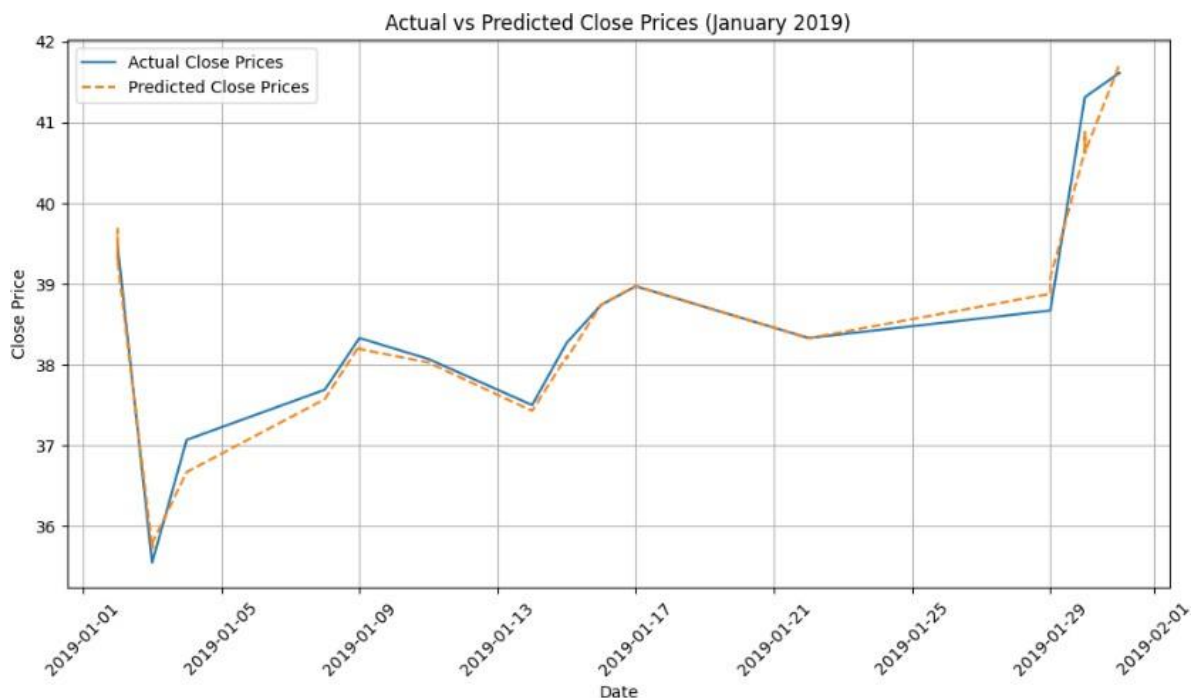
The model's R<sup>2</sup> value of 0.9981 highlights that the FinBERT-SVM model captures approximately 99.81% of the variance in stock prices, which is a near-perfect result. This means the model explains almost all factors influencing the stock prices, making it an extremely reliable tool for forecasting.

The FinBERT-SVM model employs key hyperparameters that optimize its performance, including a linear kernel with Gamma set to auto, allowing the model to adjust automatically based on the data spread. The C parameter is set to 100, balancing the complexity of the model while ensuring correct classification of stock price predictions. The epsilon value of 0.2 helps define a tolerance margin, ensuring the model remains robust against minor prediction errors.

Overall, the FinBERT-SVM hybrid model effectively combines both quantitative (stock data) and qualitative (sentiment data from FinBERT) features to produce highly accurate

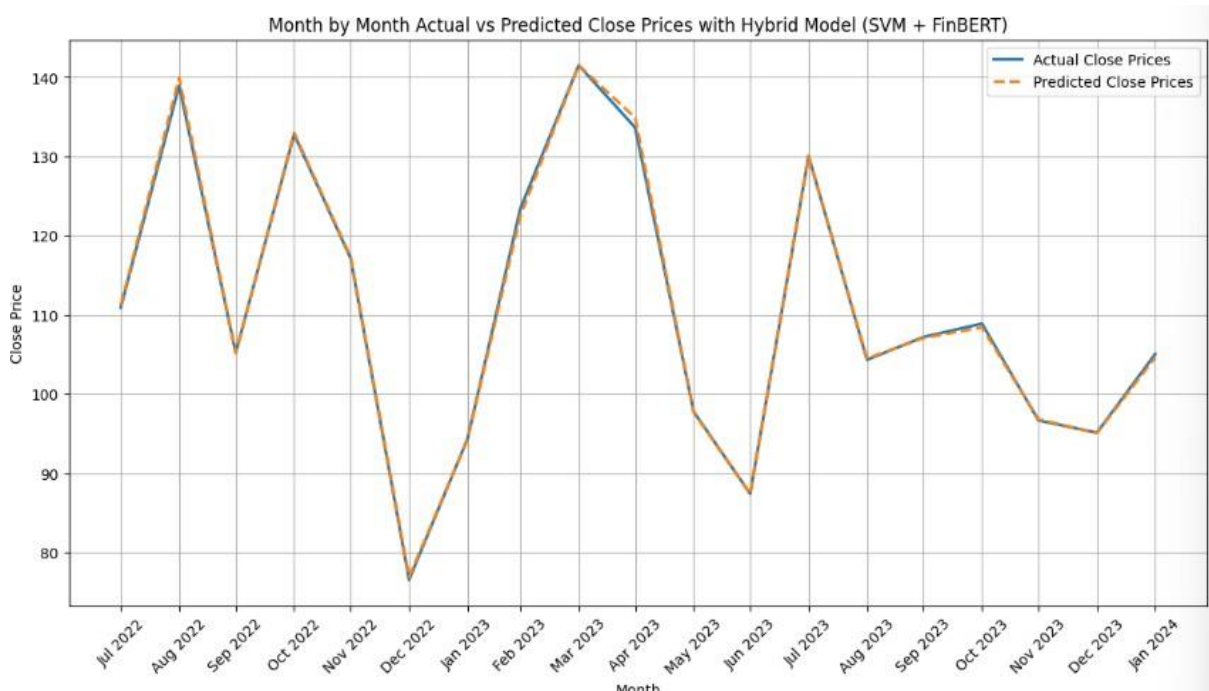
stock price predictions. The low error rates and high R<sup>2</sup> value suggest this model is particularly powerful in scenarios where sentiment analysis plays a critical role in understanding market behaviour.

### 6.2 Final Hybrid Model Visualization

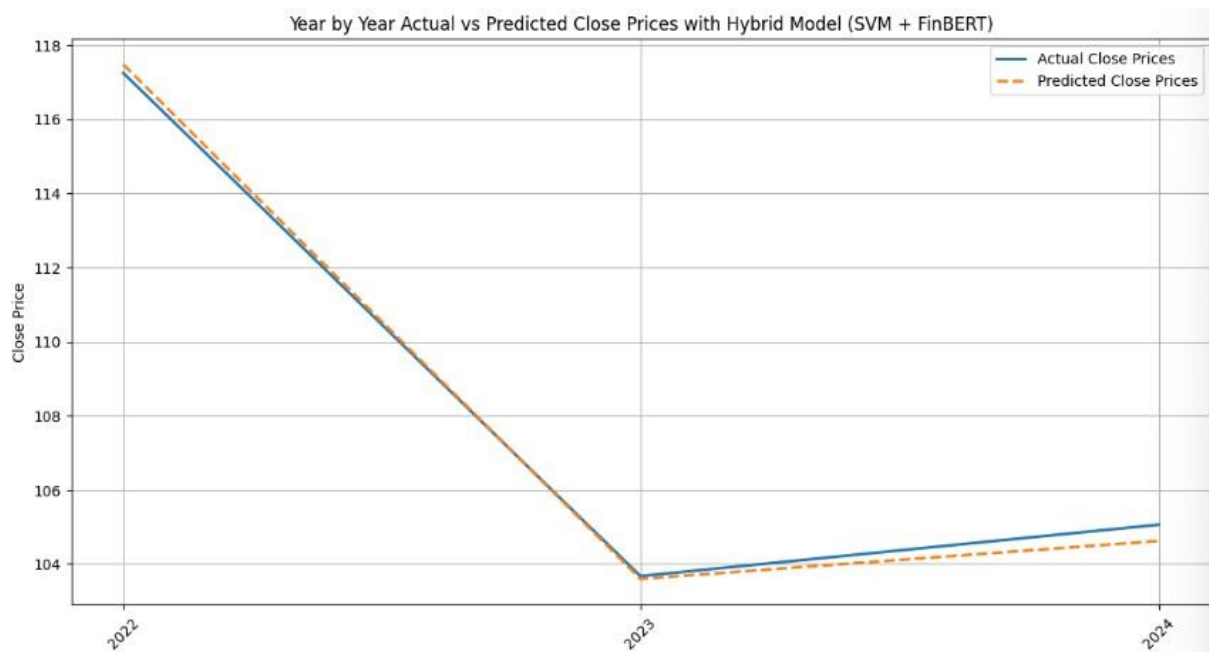


Daily Forecast Visualization For Actual Vs Predicted Stock Price with FinBERT-SVM





Monthly Forecast Visualization For Actual Vs Predicted Stock Price with FinBERT-SVM



Yearly Forecast visualization For Actual Vs Predicted Stock Price with FinBERT-SVM

## Chapter 7: Conclusion

### 7.1 Conclusion

In conclusion, the FinBERT-SVM hybrid model proves to be a highly effective tool for stock price prediction, particularly in scenarios where both financial data and sentiment analysis play crucial roles. Stock price movements are influenced not only by historical data such as prices and volumes but also by market sentiment, which can rapidly shift based on news and economic factors. By incorporating sentiment analysis from FinBERT, which is specifically trained for financial text, the model gains the ability to respond dynamically to the market mood, enhancing its predictive accuracy. This dual approach allows for a more comprehensive understanding of stock price fluctuations, making the FinBERT-SVM hybrid model especially valuable for traders and analysts who need to predict market trends with real-time insights.

The performance metrics of the FinBERT-SVM model highlight its precision and reliability in stock prediction tasks. With a Root Mean Squared Error (RMSE) of 0.8383 and Mean Absolute Error (MAE) of 0.6819, the model shows remarkably low errors, suggesting its predictions are extremely close to actual stock prices. These low error margins are particularly important in financial markets, where even small deviations can lead to significant profit or loss. The  $R^2$  value of 0.9981 further reinforces the model's effectiveness, showing that it explains nearly all the variance in stock prices, a critical factor when predicting such a complex and volatile variable. The model's ability to integrate sentiment data ensures that it remains sensitive to the external factors that can cause sudden stock price shifts, giving it an edge over traditional models that rely solely on quantitative data.

Overall, the FinBERT-SVM hybrid model showcases the importance of incorporating sentiment analysis into stock price prediction. By combining FinBERT's ability to interpret financial news with the regression power of SVM, the model is able to predict stock prices with remarkable accuracy. This hybrid approach ensures that the model not only captures historical price trends but also adapts to current market sentiment, making it an indispensable tool for modern traders and investors. As markets continue to be driven by both data and emotion, models like FinBERT-SVM are likely to play an increasingly significant role in helping investors make informed decisions in real-time.

## 7.2 Future work

### *Incorporating Additional Data Sources*

While the current model integrates sentiment analysis from financial news headlines and stock price history, future work could involve incorporating other data sources such as social media sentiment (e.g., Twitter, Reddit) and macroeconomic indicators (e.g., interest rates, inflation). These additional data points could provide a more holistic view of the factors affecting stock prices, improving the model's accuracy. Moreover, real-time data streams could be integrated to enable the model to make immediate adjustments in response to breaking news or sudden market movements.

## REFERENCES

- [1] S. Halder, "FinBERT-LSTM: Deep Learning based stock price prediction using News Sentiment Analysis." Available: <https://arxiv.org/pdf/2211.07392.pdf>
- [2] W. Cheng and S. Chen, "Sentiment Analysis of Financial Texts Based on Attention Mechanism of FinBERT and BiLSTM," 2021 International Conference on Computer Engineering and Application (ICCEA), Kunming, China, 2021, pp. 73-78, doi: 10.1109/ICCEA53728.2021.00022.
- [3] K. Puh and M. B. Babac, "Predicting stock market using natural language processing," American Journal of Business, vol. 38, no. 2, pp. 41–61, 2023, Accessed: Apr. 19, 2024. [Online]. Available: <https://ideas.repec.org/a/eme/ajbpps/ajb-08-2022-0124.html>
- [4] N. Lumoring, D. Chandra, and A. A. S. Gunawan, "A Systematic Literature Review: Forecasting Stock Price Using Machine Learning Approach," IEEE Xplore, Aug. 01, 2023. <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10277318>
- [5] I. Parmar et al., "Stock Market Prediction Using Machine Learning," 2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC), Dec. 2018, doi: <https://doi.org/10.1109/icsccc.2018.8703332>.
- [6] G. Sonkavde, D. S. Dharrao, A. M. Bongale, S. T. Deokate, D. Doreswamy, and S. K. Bhat, "Forecasting Stock Market Prices Using Machine Learning and Deep Learning Models: A Systematic Review, Performance Analysis and Discussion of Implications," International Journal of Financial Studies, vol. 11, no. 3, p. 94, Sep. 2023, doi: <https://doi.org/10.3390/ijfs11030094>
- [7] T. Fletcher, "Machine Learning in FX Carry Basket Prediction," SSRN Electronic Journal, 2008, doi: <https://doi.org/10.2139/ssrn.1319267>.

## REFERENCE

- [8] M. Roushdy, "A Comparative Analysis for Support Vector Machines for Stroke Patients," [www.academia.edu](http://www.academia.edu), Accessed: Apr. 19, 2024. [Online]. Available: [https://www.academia.edu/34236634/A\\_Comparative\\_Analysis\\_for\\_Support\\_Vector\\_Machines\\_for\\_Stroke\\_Patients](https://www.academia.edu/34236634/A_Comparative_Analysis_for_Support_Vector_Machines_for_Stroke_Patients)
- [9] L. Mathanprasad and M. Gunasekaran, "Analysing the Trend of Stock Market and Evaluate the performance of Market Prediction using Machine Learning Approach," IEEE Xplore, Jan. 01, 2022. <https://ieeexplore.ieee.org/document/9752616>
- [10] J.-X. Liu, J.-S. Leu, and S. Holst, "Stock price movement prediction based on Stocktwits investor sentiment using FinBERT and ensemble SVM," vol. 9, pp. e1403– e1403, Jun. 2023, doi: <https://doi.org/10.7717/peerj-cs.1403>.
- [11] K. Ullah and M. Qasim, "Google Stock Prices Prediction Using Deep Learning," IEEE Xplore, Nov. 01, 2020. <https://ieeexplore.ieee.org/document/9265146>
- [12] J. Patel, Miral Patel, and M. Darji, "Stock Price Prediction Using RNN and LSTM," 2018. Available: <https://www.jetir.org/papers/JETIRK006164.pdf>
- [13] C. Huang, Z. Chen, and Waleed Mahmoud Soliman, "Stock Price Prediction with FinBERT and RNN," Oct. 2023, doi: <https://doi.org/10.1145/3631908.3631919>.
- H. G. B and S. N. B, "Cryptocurrency Price Prediction using Twitter Sentiment Analysis," Natural Language Processing, Information Retrieval and AI, Feb. 2023, doi: <https://doi.org/10.5121/csit.2023.130302>.
- [14] A. H. Huang, H. Wang, and Y. Yang, "FinBERT: A Large Language Model for Extracting Information from Financial Text†," Contemporary Accounting Research, Sep. 2022, doi: <https://doi.org/10.1111/1911-3846.12832>.
- [15] T. Singh, "tyaan/Stock-Prediction-with-Sentiment-Analysis," GitHub, Feb. 26, 2024. <https://github.com/tyaan/Stock-Prediction-with-Sentiment-Analysis>

## APPENDIX

### FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Trimester 3, Year 3</b>	<b>Study week no.: 3</b>
<b>Student Name &amp; ID: Looi Wei Hung 20ACB04565</b>	
<b>Supervisor: Cik Nurul Syafidah Binti Jamil</b>	
<b>Project Title: Stock Market Prediction Using Natural Language Processing</b>	

#### 1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Done Time Series Data Preprocessing

#### 2. WORK TO BE DONE


Build and Define the LSTM model


#### 3. PROBLEMS ENCOUNTERED

N/A

#### 4. SELF EVALUATION OF THE PROGRESS

Good Progress in doing the time-series data preprocessing

  
\_\_\_\_\_  
Supervisor's signature

  
\_\_\_\_\_  
Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Trimester 3, Year 3</b>	<b>Study week no.: 5</b>
<b>Student Name &amp; ID: Looi Wei Hung 20ACB04565</b>	
<b>Supervisor: Cik Nurul Syafidah Binti Jamil</b>	
<b>Project Title: Stock Market Prediction Using Natural Language Processing</b>	

## 1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Done Building and Training The LSTM model

## 2. WORK TO BE DONE

Apply Cross Validation method to the LSTM

## 3. PROBLEMS ENCOUNTERED


N/A

## 4. SELF EVALUATION OF THE PROGRESS

Still capable in doing the modelling although spend a lot of time to rea and search fot the related knowledge



Supervisor's signature



Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Trimester 3, Year 3</b>	<b>Study week no.: 7</b>
<b>Student Name &amp; ID: Looi Wei Hung 20ACB04565</b>	
<b>Supervisor: Cik Nurul Syafidah Binti Jamil</b>	
<b>Project Title: Stock Market Prediction Using Natural Language Processing</b>	

## 1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Applied Cross Validation on LSTM model

## 2. WORK TO BE DONE

Model Test-set Evaluation and start with another models to be build like CNN and RNN with Attention Mechanism

## 3. PROBLEMS ENCOUNTERED

Limited knowledge on doing TimeSeriesSplit on the model and still find a optimal way to let model generalize well

## 4. SELF EVALUATION OF THE PROGRESS

A bit struggling on doing neural network model for the first time , but will keep moving on the good progress



Supervisor's signature



Student's signature



# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Trimester 3, Year 3</b>	<b>Study week no.: 9</b>
<b>Student Name &amp; ID: Looi Wei Hung 20ACB04565</b>	
<b>Supervisor: Cik Nurul Syafidah Binti Jamil</b>	
<b>Project Title: Stock Market Prediction Using Natural Language Processing</b>	

## 1. WORK DONE

[Please write the details of the work done in the last fortnight.]

LSTM model Test-set Evaluation and Started build CNN model layers in define and building stage

## 2. WORK TO BE DONE

CNN Model Cross Validation performance and also the hyperparameter tuning to the models

## 3. PROBLEMS ENCOUNTERED

The test-set Evaluation for LSTM is quite long in term of time, but manageable to get the result .It was not bad, but also the average results that I obtained

## 4. SELF EVALUATION OF THE PROGRESS

Training the Neural Network models were quite long in term of time-usage plus with a limited knowledge on Deep learning.It was quite exhausted sometime, but still a good improvement for me to learn something new.



Supervisor's signature



Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Trimester 3, Year 3</b>	<b>Study week no.: 10</b>
<b>Student Name &amp; ID: Looi Wei Hung 20ACB04565</b>	
<b>Supervisor: Cik Nurul Syafidah Binti Jamil</b>	
<b>Project Title: Stock Market Prediction Using Natural Language Processing</b>	

## 1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Done CNN modelling and started on research how to build the attention layer of RNN

## 2. WORK TO BE DONE

Start to do modelling phase on RNN with attention mechanism

## 3. PROBLEMS ENCOUNTERED

N/A

## 4. SELF EVALUATION OF THE PROGRESS

Neural Network is getting fun for me and it is quite in term of their architecture instead of just directly fit the train set into traditional machine learning model like Linear regression.

It is quite interesting for me



Supervisor's signature



Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Trimester 3, Year 3</b>	<b>Study week no.: 12</b>
<b>Student Name &amp; ID: Looi Wei Hung 20ACB04565</b>	
<b>Supervisor: Cik Nurul Syafidah Binti Jamil</b>	
<b>Project Title: Stock Market Prediction Using Natural Language Processing</b>	

## 1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Done Modelling phases on those Neural Network models

## 2. WORK TO BE DONE


Start doing the last model , which is SVM that is quite simple training and test this model in term of its model architecture although sometimes it has some difficult situation and questions to be solved

## 3. PROBLEMS ENCOUNTERED


The results of the neural network models quite a bit out of my expectations

## 4. SELF EVALUATION OF THE PROGRESS

It is quite brain lagging week as I had lots of deep learning knowledge came into my mind, but it is a good experience to me.



\_\_\_\_\_  
Supervisor's signature



\_\_\_\_\_  
Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Trimester 3, Year 3</b>	<b>Study week no.: 13</b>
<b>Student Name &amp; ID: Looi Wei Hung 20ACB04565</b>	
<b>Supervisor: Cik Nurul Syafidah Binti Jamil</b>	
<b>Project Title: Stock Market Prediction Using Natural Language Processing</b>	

## 1. WORK DONE

[Please write the details of the work done in the last fortnight.]

SVM modelling phase done and also the Visualisation of the models on predicted close prices vs actual in term of daily, monthly, and yearly basis

## 2. WORK TO BE DONE


Documentations and final refinement of my models code to make sure is generalized well and good in term of performance

## 3. PROBLEMS ENCOUNTERED


Some of the visualizations of the graph is quite weird as is overlapped with others and cannot capture the trends pattern. But finally solved with Miss Nurul

## 4. SELF EVALUATION OF THE PROGRESS

Good in doing the Models training,building and also the evaluation using test set.I would like to said it is a tough week for me as I sicked twice in a week to doing the project .



Supervisor's signature



Student's signature

# POSTER



**UNIVERSITI TUNKU ABDUL RAHMAN**  
Faculty of Information and Communication  
Technology

## Stock Market Prediction Using Natural Language Processing(NLP)

### PROBLEM STATEMENT

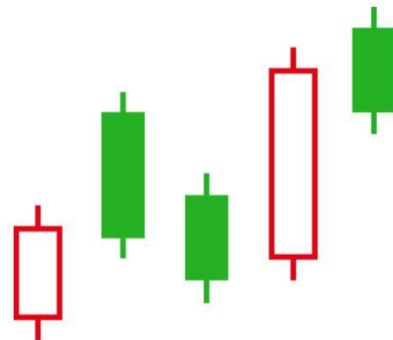
- Inaccurate Predictions
- Information Overload
- Complicated Financial Language



### OBJECTIVES



- To Enhance Prediction Accuracy with NLP
- To Achieve Efficient Textual Data Processing
- To Handle Complex Financial Language



Looi Wei Hung

Supervisor: Cik Nurul Syafidah Binti Jamil

## FYP2\_LOOI WEI HUNG

## ORIGINALITY REPORT

<b>14%</b>	<b>12%</b>	<b>10%</b>	<b>%</b>
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

## PRIMARY SOURCES

<b>1</b>	<b>fastercapital.com</b> Internet Source	<b>2%</b>
<b>2</b>	<b>eprints.utar.edu.my</b> Internet Source	<b>2%</b>
<b>3</b>	<b>www.axiaoffice.com.au</b> Internet Source	<b>1%</b>
<b>4</b>	<b>Mehdi Ghayoumi. "Generative Adversarial Networks in Practice", CRC Press, 2023</b> Publication	<b>1%</b>
<b>5</b>	<b>www.mdpi.com</b> Internet Source	<b>1%</b>
<b>6</b>	<b>link.springer.com</b> Internet Source	<b>&lt;1%</b>
<b>7</b>	<b>www.fastercapital.com</b> Internet Source	<b>&lt;1%</b>
<b>8</b>	<b>Nusrat Samia, Sajal Saha, Anwar Haque. "Predicting and mitigating cyber threats through data mining and machine learning", Computer Communications, 2024</b> Publication	<b>&lt;1%</b>

9	<a href="http://docs.neu.edu.tr">docs.neu.edu.tr</a> Internet Source	<1 %
10	<a href="https://medium.com">medium.com</a> Internet Source	<1 %
11	<a href="http://biopen.bi.no">biopen.bi.no</a> Internet Source	<1 %
12	"9th International Workshop on Spoken Dialogue System Technology", Springer Science and Business Media LLC, 2019 Publication	<1 %
13	<a href="http://nepjol.info">nepjol.info</a> Internet Source	<1 %
14	<a href="https://discovery.researcher.life">discovery.researcher.life</a> Internet Source	<1 %
15	<a href="http://biblos.hec.ca">biblos.hec.ca</a> Internet Source	<1 %
16	<a href="http://www.v7labs.com">www.v7labs.com</a> Internet Source	<1 %
17	<a href="http://dspaceapi.nsbm.ac.lk">dspaceapi.nsbm.ac.lk</a> Internet Source	<1 %
18	<a href="http://drpress.org">drpress.org</a> Internet Source	<1 %
19	<a href="http://arxiv.org">arxiv.org</a> Internet Source	<1 %



20	"Natural Language Understanding and Intelligent Applications", Springer Science and Business Media LLC, 2016 Publication	<1 %
21	assets-eu.researchsquare.com Internet Source	<1 %
22	Poornachandra Sarang. "Artificial Neural Networks with TensorFlow 2", Springer Science and Business Media LLC, 2021 Publication	<1 %
23	thecleverprogrammer.com Internet Source	<1 %
24	assets.researchsquare.com Internet Source	<1 %
25	www.frontiersin.org Internet Source	<1 %
26	dspace.univ-medea.dz Internet Source	<1 %
27	Faisal Alshomrani. "A Unified Pipeline for Simultaneous Brain Tumor Classification and Segmentation Using Fine-Tuned CNN and Residual UNet Architecture", Life, 2024 Publication	<1 %
28	ebin.pub Internet Source	<1 %



29	<a href="http://dergipark.org.tr">dergipark.org.tr</a> Internet Source	<1 %
30	Xavier Vasques. "Machine Learning Theory and Applications", Wiley, 2024 Publication	<1 %
31	"Machine Learning Approaches in Financial Analytics", Springer Science and Business Media LLC, 2024 Publication	<1 %
32	Bertrand David Barouti, Seifedine Kadry. "Chapter 5 Application of Machine Learning to Improve Safety in the Wind Industry", Springer Science and Business Media LLC, 2024 Publication	<1 %
33	<a href="http://www.researchsquare.com">www.researchsquare.com</a> Internet Source	<1 %
34	<a href="http://www.wne.uw.edu.pl">www.wne.uw.edu.pl</a> Internet Source	<1 %
35	Amirreza Mehrabi, Majid Bagheri, Majid Nabi Bidhendi, Ebrahim Biniaz Delijani, Mohammad Behnoud. "Improved Porosity Estimation in Complex Carbonate Reservoirs Using Hybrid	<1 %

36	Bei Liu, Danqing Zhou, Yaxuan Zhang, Hongyu Xie, Jiayan Shi. "Predicting stock prices for Chinese performing arts companies using genetic algorithm-based backpropagation neural networks", Journal of Computational Methods in Sciences and Engineering, 2024 Publication	<1 %
37	William B. Claster. "Mathematics and Programming for Machine Learning with R - From the Ground Up", CRC Press, 2020 Publication	<1 %
38	<a href="http://www.ijraset.com">www.ijraset.com</a> Internet Source	<1 %
39	<a href="http://www.smallake.kr">www.smallake.kr</a> Internet Source	<1 %
40	Anchana. V, N. M. Elango. "A Comparative Analysis of Deep Learning Models for Multi-class Speech Emotion Detection", Research Square Platform LLC, 2024 Publication	<1 %
41	<a href="http://www.omicsdi.org">www.omicsdi.org</a> Internet Source	<1 %

44	<a href="http://www.ncbi.nlm.nih.gov">www.ncbi.nlm.nih.gov</a> Internet Source	<1 %
45	<a href="http://turcomat.org">turcomat.org</a> Internet Source	<1 %
46	<a href="http://eprints.covenantuniversity.edu.ng">eprints.covenantuniversity.edu.ng</a> Internet Source	<1 %
47	"Inventive Systems and Control", Springer Science and Business Media LLC, 2021 Publication	<1 %
48	Jiawei Zhu, Yaru Meng, Wenli Gao, Shuo Yang et al. "Droplet-AI-empowered high- throughput screening of cell-free gene expression", Cold Spring Harbor Laboratory, 2024 Publication	<1 %
49	<a href="https://github.com">github.com</a> Internet Source	<1 %
50	<a href="http://www.caa.co.uk">www.caa.co.uk</a> Internet Source	<1 %
51	Mahima Gaurihar, Kaustubh Paonikar, Snehalata Dongre, Prashant Khobragade, Rahul Agrawal, Pranay Saraf. "Enhancing Drought Detection and Visualization with LSTM and SPEI: Addressing Slow-Onset	<1 %

## Climate-Induced Water Scarcity", Research Square Platform LLC, 2023

Publication

52	Jiwei Ran, Ganchang Zou, Ying Niu. "Deep Learning in Carbon Neutrality Forecasting", Journal of Organizational and End User Computing, 2024 Publication	<1 %
53	Irene de Zarzà i Cubero. "AI-Enhanced Methods in Autonomous Systems: Large Language Models, DL Techniques, and Optimization Algorithms", Universitat Politecnica de Valencia, 2023 Publication	<1 %
54	<a href="https://spectrum.library.concordia.ca">spectrum.library.concordia.ca</a> Internet Source	<1 %
55	<a href="https://www.activeloop.ai">www.activeloop.ai</a> Internet Source	<1 %
56	<a href="https://www.tmfv.com.ua">www.tmfv.com.ua</a> Internet Source	<1 %
57	<a href="https://mdpi-res.com">mdpi-res.com</a> Internet Source	<1 %
58	Hanyi Min, Feng Guo, Tianjun Sun, Mengqiao Liu, Fred Oswald. "Ensuring Transparency and Trust in Supervised Machine Learning Studies: A Checklist for Organizational Researchers", PsyArXiv, 2024	<1 %



Publication		
59	Kristina Dabrock, Noah Pflugradt, Jann Michael Weinand, Detlef Stolten. "Leveraging machine learning to generate a unified and complete building height dataset for Germany", Energy and AI, 2024 Publication	<1 %
60	sciresjournals.com Internet Source	<1 %
61	www.coursehero.com Internet Source	<1 %
62	Mohamed Abdel-Basset, Hossam Hawash, Laila Abdel-Fatah. "Artificial Intelligence and Internet of Things in Smart Farming", CRC Press, 2024 Publication	<1 %
63	Yang Li, Yi Pan. "A novel ensemble deep learning model for stock prediction based on stock prices and news", International Journal of Data Science and Analytics, 2021 Publication	<1 %
64	ispe.org Internet Source	<1 %
65	Kyle Walker. "Analyzing US Census Data - Methods, Maps, and Models in R", CRC Press, 2023 Publication	<1 %

66	<a href="http://ijrpr.com">ijrpr.com</a> Internet Source	<1 %
67	<a href="http://mobt3ath.com">mobt3ath.com</a> Internet Source	<1 %
68	<a href="http://tp.ok-em.com">tp.ok-em.com</a> Internet Source	<1 %
69	Jyotika Singh. "Natural Language Processing in the Real World - Text Processing, Analytics, and Classification", CRC Press, 2023 Publication	<1 %
70	<a href="http://datamites.com">datamites.com</a> Internet Source	<1 %
71	<a href="http://escholarship.org">escholarship.org</a> Internet Source	<1 %
72	Jaiwin Shah, Rishabh Jain, Vedant Jolly, Anand Godbole. "Stock Market Prediction using Bi-Directional LSTM", 2021 International Conference on Communication information and Computing Technology (ICCICT), 2021 Publication	<1 %
73	<a href="http://www.raceforhealth.org">www.raceforhealth.org</a> Internet Source	<1 %
74	<a href="http://d197for5662m48.cloudfront.net">d197for5662m48.cloudfront.net</a> Internet Source	<1 %
75	<a href="http://nmbu.brage.unit.no">nmbu.brage.unit.no</a> Internet Source	<1 %

		<1 %
76	<a href="http://ijisrt.com">ijisrt.com</a> Internet Source	<1 %
77	<a href="http://repositorio.tec.mx">repositorio.tec.mx</a> Internet Source	<1 %
78	Prasenjit Dey, Sudip Kumar Adhikari, Sourav De, Indrajit Kar. "Internet of Things-Based Machine Learning in Healthcare - Technology and Applications", CRC Press, 2024 Publication	<1 %
79	<a href="http://iris.unige.it">iris.unige.it</a> Internet Source	<1 %
80	Jaemyung Shin, Minseok Kang, Kinam Hyun, Zhangkang Li, Hitendra Kumar, Kangsoo Kim, Simon S. Park, Keekyoung Kim. "Machine Learning Driven Optimization for High Precision Cellular Droplet Bioprinting", Cold Spring Harbor Laboratory, 2024 Publication	<1 %
81	Pramod Gupta, Naresh Kumar Sehgal, John M. Acken. "Introduction to Machine Learning with Security", Springer Science and Business Media LLC, 2025	<1 %

82	Shubbh Mewada, Devshri Pandya, Jignesh Thaker. "Chapter 42 Integrating Deep Learning Techniques for Enhanced Stock Price Prediction", Springer Science and Business Media LLC, 2024 Publication	<1%
83	e-journal.stie-kusumanegara.ac.id Internet Source	<1%
84	stackoverflow.com Internet Source	<1%
85	thesai.org Internet Source	<1%
86	www.sandia.gov Internet Source	<1%
87	www.zbw.eu Internet Source	<1%
88	Kutub Thakur, Helen G. Barker, Al-Sakib Khan Pathan. "Artificial Intelligence and Large Language Models - An Introduction to the Technological Future", CRC Press, 2024 Publication	<1%
89	polynoe.lib.uniwa.gr Internet Source	<1%
90	Di Wu. "Data Mining with Python - Theory, Application, and Case Studies", CRC Press, 2024	<1%



	Publication	
91	Seyyed Soroosh Firoozabadi, Mehdi Ansari, Farhad Vasheghanifarahani. "Crude Oil Trend Prediction During COVID-19: Machine Learning with Randomized Search and Bayesian Optimization", European Journal of Business and Management Research, 2024 Publication	<1 %
92	Wenhao Li, Xinhao Li, Jiale Yuan, Runyu Liu, Yuhan liu, Qing Ye, Haishen Jiang, Long Huang. "Pressure prediction for air cyclone centrifugal classifier based on CNN-LSTM enhanced by attention mechanism", Chemical Engineering Research and Design, 2024 Publication	<1 %
93	ecotipe.ubb.ac.id Internet Source	<1 %
94	jfrm.ru Internet Source	<1 %
95	www.ijcert.org Internet Source	<1 %
96	www.nature.com Internet Source	<1 %
97	Huiyu Wang, Xiaoyi Wang, Dawei Yan, Hao Sun, Qiang Chen, Mingli Li, Xinxing Dong, Yuchun Pan, Shaoxiong Lu. "Genome-wide association study identifying genetic variants	<1 %

associated with carcass backfat thickness, lean percentage and fat percentage in a four-way crossbred pig population using SLAF-seq technology", BMC Genomics, 2022

Publication

98	Vidya K. Sudarshan, Reshma A. Ramachandra, Smit Ojha, Ru-San Tan. "DCEnt-PredictiveNet: A novel explainable hybrid model for time series forecasting", Neurocomputing, 2024	<1 %
99	aiforsocialgood.ca Internet Source	<1 %
100	content.iospress.com Internet Source	<1 %
101	docplayer.net Internet Source	<1 %
102	edubirdie.com Internet Source	<1 %
103	land.copernicus.eu Internet Source	<1 %
104	serialsjournals.com Internet Source	<1 %
105	www.geeksforgeeks.org Internet Source	<1 %
106	123dok.com Internet Source	<1 %

---

107	<a href="http://www.westmidlands-pcc.gov.uk">www.westmidlands-pcc.gov.uk</a> Internet Source	<1 %
108	Christopher Starkey, Georges Tsafack. "Measuring financial contagion: Dealing with the volatility Bias in the correlation dynamics", International Review of Financial Analysis, 2023 Publication	<1 %
109	Jay Liebowitz. "Data Analytics and AI", CRC Press, 2020 Publication	<1 %
110	<a href="http://discuss.pytorch.org">discuss.pytorch.org</a> Internet Source	<1 %
111	<a href="http://e-space.mmu.ac.uk">e-space.mmu.ac.uk</a> Internet Source	<1 %
112	<a href="http://eshop.advania.no">eshop.advania.no</a> Internet Source	<1 %
113	<a href="http://journal.esrgroups.org">journal.esrgroups.org</a> Internet Source	<1 %
114	<a href="http://tanthiamhuat.files.wordpress.com">tanthiamhuat.files.wordpress.com</a> Internet Source	<1 %
115	<a href="http://www.diva-portal.org">www.diva-portal.org</a> Internet Source	<1 %
116	<a href="http://www.packtpub.com">www.packtpub.com</a> Internet Source	<1 %

---



117	<a href="http://www.utdallas.edu">www.utdallas.edu</a> Internet Source	<1 %
118	"Data Mining", Springer Science and Business Media LLC, 2021 Publication	<1 %
119	Mohammed A. Imam, Thamir A. Alandijany, Hashim R. Felemban, Roba M. Attar et al. "Machine learning, network pharmacology, and molecular dynamics reveal potent cyclopeptide inhibitors against dengue virus proteins", Molecular Diversity, 2024 Publication	<1 %
120	Sai Kiran Oruganti, Dimitrios A Karras, Srinesh Singh Thakur. "Advancing Sustainable Science and Technology for a Resilient Future", CRC Press, 2024 Publication	<1 %
121	<a href="http://journal.unj.ac.id">journal.unj.ac.id</a> Internet Source	<1 %
122	<a href="http://library.fiveable.me">library.fiveable.me</a> Internet Source	<1 %
123	Jose M. Cortina, Ronald S. Landis. "Modern Research Methods for the Study of Behavior in Organizations", Routledge, 2013 Publication	<1 %

---

124	Nisha Bajiya, Nishant Kumar, Gajendra P. S. Raghava. " Prediction of inhibitory peptides against with desired MIC value ", Cold Spring Harbor Laboratory, 2024 Publication	<1 %
125	Renaud Mabire-Yon. "Zero Inflation? Zero Worries! Unraveling Count Data with the Hurdle Model", Open Science Framework, 2024 Publication	<1 %
126	Richard J. Roiger. "Just Enough R! - An Interactive Approach to Machine Learning and Analytics", CRC Press, 2020 Publication	<1 %
127	<a href="http://www.brnsspubhub.org">www.brnsspubhub.org</a> Internet Source	<1 %
128	<a href="http://www.dell.com">www.dell.com</a> Internet Source	<1 %
129	<a href="http://dev.to">dev.to</a> Internet Source	<1 %
130	<a href="http://dspace.vutbr.cz">dspace.vutbr.cz</a> Internet Source	<1 %
131	<a href="http://www.tandfonline.com">www.tandfonline.com</a> Internet Source	<1 %
132	<a href="http://bedouin7.medium.com">bedouin7.medium.com</a> Internet Source	<1 %

---

133	<a href="https://downloads.hindawi.com">downloads.hindawi.com</a> Internet Source	<1 %
134	<a href="https://psrefstuff.lenovo.com">psrefstuff.lenovo.com</a> Internet Source	<1 %
135	<a href="https://repository.tudelft.nl">repository.tudelft.nl</a> Internet Source	<1 %
136	<a href="https://repository.umy.ac.id">repository.umy.ac.id</a> Internet Source	<1 %
137	Ton Duc Thang University Publication	<1 %
138	Vyom Unadkat, Parth Sayani, Pratik Kanani, Prachi Doshi. "Deep Learning for Financial Prediction", 2018 International Conference on Circuits and Systems in Digital Enterprise Technology (ICCSDET), 2018 Publication	<1 %
139	<a href="https://academic.oup.com">academic.oup.com</a> Internet Source	<1 %
140	<a href="https://ejournals.org">ejournals.org</a> Internet Source	<1 %
141	<a href="https://research.g2.com">research.g2.com</a> Internet Source	<1 %
142	Gaoyu Cao, Zhanquan Sun, Minlan Pan, Jiangfei Pang, Zhiqiang He, Jiayu Shen. "UUnet: An effective cascade Unet for	<1 %



automatic segmentation of renal  
parenchyma", 2021 IEEE Symposium Series  
on Computational Intelligence (SSCI), 2021

Publication

143 Mas Omar, Fitri Yakub, Shahrum Shah  
Abdullah, Muhamad Sharifuddin Abd Rahim,  
Ainaa Hanis Zuhairi, Niranjana Govindan.  
"One-step vs horizon-step training strategies  
for multi-step traffic flow forecasting with  
direct particle swarm optimization grid search  
support vector regression and long short-  
term memory", Expert Systems with  
Applications, 2024

<1 %

Publication

144 Nurtileu Assymkhan, Amandyk Kartbayev.  
"Advanced IoT-Enabled Indoor Thermal  
Comfort Prediction Using SVM and Random  
Forest Models", International Journal of  
Advanced Computer Science and  
Applications, 2024

<1 %

Publication

145 [backend.orbit.dtu.dk](https://backend.orbit.dtu.dk)

Internet Source

<1 %

146 [elib.belstu.by](https://elib.belstu.by)

Internet Source

<1 %

147 [www.researchgate.net](https://www.researchgate.net)

Internet Source

<1 %

148 "Advanced Intelligent Computing Technology and Applications", Springer Science and Business Media LLC, 2024 <1%  
Publication

---

149 Delaram Kahrobaei, Enrique Domínguez, Reza Soroushmehr. "Artificial Intelligence in Healthcare and Medicine", CRC Press, 2022 <1%  
Publication

---

150 Vinh Truong. "Textual emotion detection – A systematic literature review", Springer Science and Business Media LLC, 2024 <1%  
Publication

---

Exclude quotes Off

Exclude matches Off

Exclude bibliography Off



<b>Form Title: Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)</b>			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1



**FACULTY OF INFORMATION AND COMMUNICATION  
TECHNOLOGY**

<b>Full Name(s) of Candidate(s)</b>	LOOI WEI HUNG
<b>ID Number(s)</b>	2004565
<b>Programme / Course</b>	Business Information system
<b>Title of Final Year Project</b>	Stock market prediction using natural language processing

<b>Similarity</b>	<b>Supervisor's Comments (Compulsory if parameters of originality exceed the limits approved by UTAR)</b>
<b>Overall similarity index: <u>14</u> %</b>  <b>Similarity by source</b> Internet Sources: <u>12</u> % Publications: <u>10</u> % Student Papers: <u>-</u> %	
<b>Number of individual sources listed of more than 3% similarity: <u>0</u></b>	
<b>Parameters of originality required, and limits approved by UTAR are as Follows:</b> (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note: Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

*Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.*

  
Signature of Supervisor

Name: Cik Nurul Syafidah Binti Ja\_mil

Date: 13/09/2024

\_\_\_\_\_  
Signature of Co-Supervisor

Name: \_\_\_\_\_

Date: \_\_\_\_\_

**FYP CHECKLIST****UNIVERSITI TUNKU ABDUL RAHMAN****FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY  
(KAMPAR CAMPUS)****CHECKLIST FOR FYP2 THESIS SUBMISSION**

Student Id	20ACB04565
Student Name	LOOI WEI HUNG
Supervisor Name	Cik Nurul Syafidah Binti Jamil

<b>TICK (✓)</b>	<b>DOCUMENT ITEMS</b>
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
✓	Title Page
✓	Signed Report Status Declaration Form
✓	Signed FYP Thesis Submission Form
✓	Signed form of the Declaration of Originality
✓	Acknowledgement
✓	Abstract
✓	Table of Contents
✓	List of Figures (if applicable)
✓	List of Tables (if applicable)
✓	List of Symbols (if applicable)
✓	List of Abbreviations (if applicable)
✓	Chapters / Content
✓	Bibliography (or References)
✓	All references in bibliography are cited in the thesis, especially in the chapter of literature review
✓	Appendices (if applicable)
✓	Weekly Log
✓	Poster
✓	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
✓	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

\*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

(Signature of Student)

Date: 13/09/2024