

MSc Data Science Project

7PAM2002-0509-2023

Department of Physics, Astronomy and Mathematics

Data Science FINAL PROJECT REPORT

Project Title:

**Predicting Stock Market Values Using Machine
Learning and News Sentiment Analysis**

Student Name and SRN:

TAMILSELVAN PALANISAMY, 22055049

Supervisor: [MYKOLA GORDOVSKYY](#)

Date Submitted: [28/08/2024](#)

Word Count: [9317](#)

GitHub Link: https://github.com/Tamil2095/Msc_Finalproject.git

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.

I have read the guidance to students on academic integrity, misconduct and plagiarism information at [Assessment Offences and Academic Misconduct](#) and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project module or course.

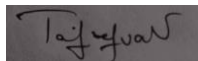
I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6). I have not used ChatGPT, or any other generative AI tool, to write the report or code (other than where declared or referenced).

I did not use human participants or undertake a survey in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: TAMILSELVAN PALANISAMY

Student Name signature:



Student SRN number: 22055049

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

Acknowledgement

I would like to express my heartfelt gratitude to everyone who has supported me throughout the completion of this project.

Firstly, I would like to extend my deepest thanks to my Programme Leader Carolyn Devereux, whose guidance and insight have been pivotal in shaping my academic path. Your leadership and dedication to fostering a culture of excellence have inspired me throughout my studies.

I am also particularly grateful to my Module Leader Carolyn Devereux, for providing the essential resources and foundation necessary to undertake this project. Your valuable feedback and direction have greatly contributed to the refinement and success of my research.

A special thank you goes to my Supervisor Mykola Gordovskyy, for your unwavering support, expert advice, and encouragement. Your guidance has been instrumental in overcoming the challenges of this project, and your commitment to my success has been a continuous source of motivation.

To my family, I owe a deep debt of gratitude for your unconditional love, support and belief in my capabilities. Your encouragement has been my pillar of strength and your sacrifices have enabled me to reach this important milestone.

Finally, I would like to thank my friends for their consistent support, encouragement, and understanding throughout this journey. Your companionship and positivity have been invaluable in keeping me balanced and motivated, even during the most demanding periods.

This project would not have been possible without the collective support and encouragement from each of you. Thank you for being an integral part of this journey with me.

Abstract

This report explores how news sentiment impacts the stock prices of Shell and BP, using machine learning models to improve prediction accuracy. The study examines whether sentiment from news articles, when combined with historical stock data, can enhance stock price forecasts. Two models, Linear Regression and Random Forest, were applied to predict the open and close prices of Shell and BP stocks. The data used includes news articles retrieved from APIs and historical stock prices from Yahoo Finance, with sentiment scores calculated using the VADER sentiment analyser. The analysis found that Linear Regression consistently provided better predictions than Random Forest, especially for BP, as indicated by lower Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) values. This suggests a predominantly linear relationship between stock prices and sentiment in this scenario. The study concludes that simpler models like Linear Regression, which are less prone to overfitting, are more effective when paired with sentiment analysis, offering valuable insights for financial forecasting. Future research could focus on incorporating additional data sources and experimenting with more complex models to further enhance prediction accuracy.

Table of Contents

1. Introduction	7
1.1 Background	7
1.2 Aim	8
1.3 Objectives.....	8
2. Literature Review	9
3. Ethical Considerations.....	13
3.1 Data Privacy	13
3.2 Accuracy and Transparency	13
3.3 Bias Mitigation	13
3.4 Responsible Reporting	14
3.5 Compliance with Legal Standards	14
3.6 University of Hertfordshire Ethical Considerations	14
4. Data Collection and Pre-processing	14
4.1 Data Collection	14
4.2 Data Pre-processing	15
4.2.1 News Articles Pre-processing.....	15
4.2.2 Historical Stock Prices Pre-processing	15
4.2.3 Merging Datasets	15
4.2.4 Creation of Lagged Features	16
5. Sentiment Analysis Methodology	17
5.1 Overview	17
5.2 VADER Sentiment Analyser	17
5.3 Implementation	17
5.4 Example Analysis.....	17
6. Text Pre-processing.....	19
6.1 Lexicon Lookup.....	19
6.2 Heuristics for Punctuation and Capitalization	19
6.3 Heuristics for Negation	19
6.4 Heuristics for Degree Modifiers	19
6.5 Calculation of Compound Score.....	19
7. Distribution of Average Sentiment Scores	21
8. Stock Price Candlestick Chart.....	21
9. Predictive Modelling	24

9.1 Linear Regression	24
9.2 Random Forest Regression	26
9.2.1 Variables Used	27
9.2.2 Target Variables	27
9.2.3 Steps in Random Forest Implementation	27
10. Model Training and Testing	28
10.1 Linear Regression	28
10.2 Random Forest Regression	29
11. Evaluation of Results.....	29
12. Comparison of Models.....	30
12.1 Shell Stock Price Prediction.....	30
12.2 BP Stock Price Prediction	30
13. Analysis and Insights	31
13.1 Shell.....	31
13.2 BP	31
14. Learning Curve	32
14.1 Learning Curve for Shell.....	32
14.2 Learning Curve for BP	35
15. Discussion and Analysis.....	38
15.1 Interpretation of Results.....	38
15.2 Model Comparison and Performance.....	38
15.3 Limitations and Practical Implications	39
15.4 Addressing Project Objectives and Research Questions	39
16. Key Findings	40
17. Conclusion.....	41
18. References	42
19. Appendix	44

1. Introduction

The stock market is a dynamic and ever-changing entity, influenced by a complex web of factors. These range from broad economic trends and a company's internal performance to global events and, crucially, the sentiments of investors. Predicting stock prices has always been a challenging endeavour, largely due to the unpredictability and multitude of variables at play. Among these, public sentiment has emerged as a critical factor, often driving market trends in unexpected directions. This sentiment is captured in the stories told through news articles, the buzz on social media, and other forms of public discourse all of which subtly influence investor behaviour.

In recent years, technological advancements, especially in Natural Language Processing (NLP) and machine learning, have unlocked new opportunities. We can now analyse vast amounts of text data to measure sentiment in ways that were previously unthinkable. Sentiment analysis, which delves into the opinions and emotions expressed in text, has become an essential tool for financial analysts. By converting written content into sentiment scores, these analysts can gain valuable insights into the market's overall mood, potentially improving the accuracy of stock price predictions.

This project undertakes an exploration of the connection between news sentiment and stock prices, specifically focusing on two prominent oil companies: Shell and BP. By combining sentiment analysis with historical stock data, the project aims to develop predictive models using Linear Regression and Random Forest algorithms. The central question driving this effort is whether sentiment analysis can enhance the accuracy of stock price predictions, and if so which modelling approach proves to be the most effective in this particular context.

1.1 Background

Financial markets are by their very nature, influenced by a vast array of factors. Economic indicators, a company's performance metrics, regulatory changes, global events, and public sentiment all play a part. Among these, public sentiment is particularly potent, as it mirrors how investors feel and what they expect from the market. This sentiment is conveyed through various channels, such as news articles, social media posts, and financial reports, and it holds the power to significantly influence investor behaviour and consequently, stock prices.

Sentiment analysis steps into this picture as a tool that uses technology to gauge the emotions and opinions expressed in text. By analysing the tone and mood in these communications, sentiment analysis offers us a glimpse into how public sentiment might be intertwined with market trends. For instance, positive news about a company's performance might bolster investor confidence, potentially driving up the stock price. Conversely, negative news could prompt investors to sell their shares, leading to a decline in stock prices.

Thanks to recent strides in machine learning and data processing, it's now easier than ever to incorporate sentiment analysis into financial modelling. These technologies enable us to process vast amounts of data at lightning speed, applying sophisticated algorithms to uncover valuable patterns. In the context of predicting stock prices, sentiment analysis

allows us to integrate qualitative information from news articles into our models, potentially enhancing the accuracy of these predictions.

This project embarks on a mission to delve into the relationship between news sentiment and the stock prices of Shell and BP. By blending sentiment scores derived from news articles with historical stock data, the project aims to create predictive models capable of forecasting future stock prices. We utilize both Linear Regression and Random Forest algorithms to determine which approach is more effective at capturing the impact of sentiment on stock prices. The insights gleaned from this project could prove invaluable to investors and analysts alike.

1.2 Aim

The aim of this project is to delve into how news sentiment impacts the stock prices of Shell and BP. By integrating sentiment analysis with historical stock data, the project endeavours to build predictive models using Linear Regression and Random Forest algorithms. The ultimate goal is to determine whether sentiment analysis can enhance the accuracy of stock price predictions and to identify the most effective modelling approach in this scenario.

1.3 Objectives

1. To analyse and quantify the sentiment expressed in news articles related to Shell and BP using sentiment analysis tools.
2. To create and apply predictive models (Linear Regression and Random Forest) that integrate sentiment scores with historical stock prices to predict future stock prices.
3. To evaluate the precision of these models by analysing their performance using metrics like Mean Squared Error (MSE) and Root Mean Squared Error (RMSE).
4. To compare the effectiveness of Linear Regression and Random Forest in predicting stock prices based on news sentiment, with the objective of identifying the most suitable model for this purpose.

2. Literature Review

Recent years have seen a growing interest in applying machine learning and sentiment analysis to predict financial market movements. This literature review brings together insights from various studies, highlighting how these methods are used and what they have found.

- **"Stock Prediction by Analysing Financial News Sentiment and Investor Mood of Social Media" by Jamie Deng
University of California, Santa Cruz (2020)**

In this thesis, Jamie Deng explores the potential of using textual data from financial news and social media to forecast stock movements. By employing NLP techniques like VADER and Word2vec, the study extracts sentiment scores and investor moods. These data points are combined with historical stock prices and volumes to train models using Random Forest and SVM algorithms. The research achieves high accuracy, especially for Apple shares, with a prediction accuracy of 75.68%. Additionally, the thesis tests two trading strategies based on sentiment indicators, both of which outperform the traditional buy-and-hold strategy.

- **"Stock Trend Prediction Using News Sentiment Analysis" by Kalyani Joshi, Prof. Bharathi H. N., and Prof. Jyothi Rao
Department of Computer Engineering, K. J. Somaiya College of Engineering, Mumbai (2016)**

This study examines the link between news articles and stock price trends. It employs sentiment analysis to classify news articles as positive or negative, using machine learning algorithms such as Random Forest, SVM, and Naïve Bayes. The models achieve an accuracy of over 80% in predicting stock trends. The research underscores the importance of both technical and fundamental analysis for accurate predictions, highlighting that news sentiment significantly impacts stock prices.

- **"Efficacy of News Sentiment for Stock Market Prediction" by Sneh Kalra and Jay Shankar Prasad
MVN University, Palwal, India (2019)**

This paper focuses on forecasting Indian stock market movements by combining historical data with news sentiment analysis. Using the Naïve Bayes classifier to categorize news articles as positive or negative, the study integrates these sentiment values with historical stock data. The model achieves an accuracy ranging from 65.30% to 91.2% with various machine learning techniques. The research emphasizes the significant influence of financial news on stock price fluctuations and the effectiveness of sentiment analysis in enhancing prediction accuracy.

- **"Stock Market Prediction Using Historical Data and Sentiment Analysis of Financial News" by Nousi Christina
Unpublished (2020)**

Christina's research integrates historical stock prices with sentiment analysis of financial news to predict market trends. Various machine learning algorithms, including Random Forest, SVM, and neural networks, are employed. The study highlights the strong correlation between news sentiment and stock price

movements, achieving a prediction accuracy of 70% to 85%. It also explores the effectiveness of different pre-processing techniques in improving model performance.

- **"Predicting Stock Price Movements Based on Different Categories of News Articles" by Yauheniya Shynkevich et al.**

IEEE Symposium Series on Computational Intelligence (2015)

This research categorizes news articles by their relevance to a target stock and uses multiple kernel learning techniques for sentiment analysis. The study shows that utilizing multiple categories of news articles enhances prediction performance. The highest accuracy and return per trade are achieved when all five categories of news are used, demonstrating the benefits of a nuanced approach to news sentiment analysis in stock prediction.

- **"Sentiment Analysis of Financial News Articles" by Robert P. Schumaker, Yulei Zhang, and Chun-Neng Huang**

Unpublished (2013)

This paper investigates how financial news sentiment affects stock prices. Using a sentiment detection algorithm, the study classifies news articles as positive or negative and correlates these sentiments with stock price movements. The research achieves significant prediction accuracy and suggests that sentiment analysis of financial news can be a valuable tool for investors aiming to predict market trends.

- **"Stock Price Prediction Using Linear Regression Based on Sentiment Analysis" by B. D. Trisedya and Y. E. Cakra**

International Conference on Advanced Computer Science and Information Systems (2015)

Trisedya and Cakra's study applies linear regression to predict stock prices using sentiment analysis of news articles. The research finds that sentiment values from news significantly enhance the accuracy of stock price predictions. The study compares different machine learning models and concludes that integrating sentiment analysis with traditional forecasting methods yields better prediction results.

- **"Using News Articles to Predict Stock Price Movements" by Győző Gidófalvi**
University of California, San Diego (2001)

Gidófalvi's research investigates the use of news articles to predict stock prices. The study employs text mining techniques to extract sentiment from news articles and correlates these sentiments with stock price changes. The research demonstrates that news sentiment is a strong predictor of stock price movements, achieving notable accuracy in its predictions. The study highlights the potential of combining text mining with financial analysis for stock market forecasting.

- **"Stock Price Movement Prediction Using Sentiment Analysis and CandleStick Chart Representation" by Trang-Thi Ho and Yennun Huang**
Sensors, 2021

This paper investigates the combination of sentiment analysis and candlestick chart representation to predict stock price movements. The authors propose a

multichannel collaborative network that integrates social media sentiment data and historical stock data visualized as candlestick charts. By using a one-dimensional CNN for sentiment classification and a two-dimensional CNN for candlestick chart analysis, the study evaluates the model on stocks like Apple and Tesla. Results show that merging sentiment and candlestick data enhances prediction accuracy, with the highest performance for Apple stock at 75.38% accuracy over a 10-day period. The study demonstrates the effectiveness of multi-source data in improving stock trend prediction.

- **"Prediction of Stock Values Changes Using Sentiment Analysis of Stock News Headlines" by László Nemes and Attila Kiss**
Journal of Information and Telecommunication, 2021
 Nemes and Kiss explore the predictive power of sentiment analysis on stock values using headlines from economic news. Comparing tools such as BERT, VADER, TextBlob, and RNN, the study aims to correlate sentiment scores with stock market movements. Findings indicate that BERT and RNN offer more accurate sentiment detection without neutral values, significantly impacting stock value predictions. The study identifies notable differences in the effectiveness of various sentiment analysis models, with BERT and RNN outperforming others in predicting stock price changes.
- **"Forecasting the Stock Market Using News Sentiment Analysis" by Marinus Teunis Bakker**
Master's Thesis, Tilburg University, 2021
 Bakker's thesis examines the use of news sentiment analysis to predict daily stock market movements. Using a unique dataset of news articles and S&P 500 price movements from 2013 to 2020, the study compares various machine and deep learning models. Key findings reveal that the sentiment polarity of news articles can effectively predict stock market trends, particularly when combined with technical analysis features. The support vector machine (SVM) model demonstrates the highest performance, highlighting that integrating news sentiment with technical indicators enhances predictive accuracy.
- **"Predicting Stock Market by Sentiment Analysis and Deep Learning" by Süreyya Özögür Akyüz, Pınar Karadayı Ataş, and Aymane Benkhaldoun**
Operations Research and Decisions, 2024
 This paper examines the use of deep learning and sentiment analysis to predict stock market trends. Using historical stock prices and sentiment scores from tweets, the study compares the performance of LSTM and 1D CNN models. Results show that sentiment scores significantly improve prediction accuracy, with the 1D CNN model showing superior performance. The study highlights the importance of combining textual sentiment analysis with traditional time-series data to enhance stock market forecasting.
- **"Study on Various Stock Prediction Techniques With News Sentiment" by Vijay Kandasamy, Sorna Shanthi Dhinakaran, Priya Vijay, and Bhuvaneshwaran Balasubramanian**
AIP Conference Proceedings, 2023

This paper reviews multiple techniques for predicting stock prices using news sentiment analysis. The authors stress the importance of integrating external factors like news and social media data with traditional historical stock data to improve prediction accuracy. They review various machine learning methods, including Random Forest, SVM, Naive Bayes, and deep learning models like LSTM. The study concludes that incorporating sentiment analysis from sources such as Twitter and news articles significantly enhances prediction models, with Random Forest achieving up to 89.8% accuracy when combined with sentiment data.

- **"Stock Price Prediction Using Sentiment Analysis and Deep Learning for Indian Markets" by Narayana Darapaneni, Anwesh Reddy Paduri, Himank Sharma, Milind Manjrekar, Nutan Hindlekar, Pranali Bhagat, Usha Aiyer, and Yogesh Agarwal**
Great Learning, Bangalore, India

This research aims to predict stock prices using historical data and sentiment analysis from news headlines. Two models were utilized: LSTM for historical price data and Random Forest incorporating sentiment analysis along with macroeconomic factors like gold prices, oil prices, USD exchange rates, and Indian government securities yields. The study found that sentiment analysis combined with historical data improved prediction accuracy, with the LSTM model performing better overall. The research underscores the value of including external sentiment data for more precise stock price predictions.

- **"Prediction of Stock Values Changes Using Sentiment Analysis of Stock News Headlines" by László Nemes and Attila Kiss**
Journal of Information and Telecommunication, 2021

Nemes and Kiss analyse the impact of news sentiment on stock value changes, using various sentiment analysis tools such as BERT, VADER, TextBlob, and RNN. They find that BERT and RNN models are more effective in predicting stock price movements by accurately capturing sentiment without neutral values. The study highlights significant differences in the effectiveness of various sentiment analysis tools and confirms the strong correlation between sentiment scores and stock price changes, indicating that sentiment analysis can be a valuable tool for stock market predictions.

- **"Decision Support System for Investing in Stock Market by using OAA-Neural Network" by Sabaithip Boonpeng and Piyasak Jeatrakul**
8th International Conference on Advanced Computational Intelligence, 2016

This paper introduces a decision support system for stock market investments utilizing OAA-Neural Network. The authors discuss integrating technical analysis and sentiment analysis to enhance stock price predictions. By employing neural networks and other machine learning algorithms, the study achieves high accuracy in predicting stock price movements. The system demonstrates that combining technical indicators with sentiment analysis from news and social media significantly improves prediction performance, providing a robust tool for investors.

The literature reviewed here highlights the significant role that sentiment analysis and machine learning play in forecasting stock market trends. Sentiment, whether derived from news articles, social media, or other textual sources, is consistently linked to market behaviour, with positive sentiment often correlating with stock price increases and negative sentiment typically preceding declines. Various machine learning models, including Random Forest, SVM and advanced techniques like LSTM and CNN, have proven effective in capturing the intricate relationships between sentiment and stock prices, often achieving higher prediction accuracy compared to traditional methods. Integrating sentiment analysis with historical stock data emerges as a critical factor in enhancing the predictive capability of these models, leading to a more comprehensive understanding of market dynamics. However, the effectiveness of this integration depends heavily on the choice of sentiment analysis tools, such as VADER and BERT, and the specific methodologies applied. While advanced models offer the potential for greater accuracy, simpler models like Linear Regression remain valuable for their clarity and lower susceptibility to overfitting. Despite the challenges, including the need for large datasets and the risk of overfitting, combining sentiment analysis with machine learning continues to be a potent approach for financial forecasting, providing essential insights that have informed the methodology of this project in predicting stock prices for Shell and BP.

3. Ethical Considerations

In conducting this project, several ethical considerations were addressed to ensure the integrity and reliability of the research process:

3.1 Data Privacy

- Ensuring the privacy of any personal data involved in the news articles and stock prices.
- Handling and storing data responsibly, adhering to the General Data Protection Regulation (GDPR) guidelines.
- Anonymizing data where necessary and ensuring secure storage and handling practices.

3.2 Accuracy and Transparency

- Ensuring the accuracy of data collection and analysis processes.
- Maintaining transparency in methodology to allow for reproducibility and verification by other researchers.
- Documenting data collection, pre-processing, and analysis steps in detail.

3.3 Bias Mitigation

- Recognizing and mitigating any biases in data selection and analysis.
- Using diverse sources of news to capture a balanced sentiment perspective.
- Including articles from multiple reputable sources to reduce the risk of biased sentiment analysis.

3.4 Responsible Reporting

- Reporting findings and conclusions responsibly, avoiding misinterpretation that could influence market behaviour or investor decisions unfairly.
- Presenting results objectively, highlighting both strengths and limitations of the study.

3.5 Compliance with Legal Standards

- Adhering to all relevant laws and regulations related to data usage and intellectual property.
- Complying with GDPR for data privacy and protection.
- Ensuring all data sources were used legally and ethically.

3.6 University of Hertfordshire Ethical Considerations

- Following specific ethical standards of the University of Hertfordshire (UH).

4. Data Collection and Pre-processing

4.1 Data Collection

In the data collection phase of this project, we gathered a comprehensive dataset of news articles and historical stock prices. The news articles were sourced using two main APIs: the Guardian API and the NewsAPI, which were queried using the keyword "oil" to ensure relevance to the oil market and its potential impact on stock prices.

- **Guardian API:** Articles from the business section related to oil were retrieved. This API provided detailed information, including the title, section name, publication date, and content of the articles.
 - API Requests: Multiple API requests were made to gather as many relevant articles as possible.
 - Content Extraction: Detailed content and descriptions were fetched for sentiment analysis.
- **NewsAPI:** Additional articles were collected using this API, which offered a diverse range of articles from various sources, all filtered by the keyword "oil".
 - API Requests: Daily API requests were made within the specified date range to ensure comprehensive coverage.
 - Content Extraction: Titles, descriptions, and content were extracted for sentiment analysis.

- **Historical Stock Data:** Historical stock price data for Shell (ticker: SHEL) and British Petroleum (ticker: BP) were collected via the Yahoo Finance API. This dataset included opening, closing, high, and low prices for each trading day within the specified date range.
 - Yahoo Finance API: Historical stock prices for Shell and BP were fetched.
 - Date Range: The date range for the stock data matched the range for the collected news articles to ensure accurate analysis.

4.2 Data Pre-processing

Pre-processing the collected data was a critical step to ensure its quality and relevance for sentiment analysis and predictive modelling. The following pre-processing steps were undertaken:

4.2.1 News Articles Pre-processing

- **Extraction of Relevant Fields:** We extracted key fields such as title, description, and content from each news article.
- **Sentiment Scoring:** The VADER sentiment analyser was employed to score the sentiment of the title, description, and content of each article. VADER provides four sentiment scores: positive, neutral, negative, and compound. For deeper analysis, we focused on the compound score, a normalized and weighted composite metric.
- **Date Formatting:** The publication date of each article was converted to a standard date format to facilitate merging with the stock price data.

4.2.2 Historical Stock Prices Pre-processing

- **Date Extraction:** The 'Date' column in the stock price data was converted to a date-time format, ensuring consistency with the dates of the news articles.
- **Selection of Relevant Columns:** Only the relevant columns (Date, Open, High, Low, Close) were selected for further analysis.
- **Renaming Columns:** The 'Date' column was renamed to 'date' to align with the news articles data.

4.2.3 Merging Datasets

- **Combining Sentiment Scores and Stock Prices:** The sentiment scores from the news articles were averaged per day and merged with the corresponding day's stock prices for Shell and BP. This resulted in two comprehensive datasets one for Shell and one for BP.
- **Handling Missing Values:** Rows containing missing values were eliminated to maintain the data's integrity.

4.2.4 Creation of Lagged Features

- **Lagged Sentiment Scores:** Lagged features for sentiment scores (e.g., the previous day's sentiment) were created to capture the temporal relationship between sentiment and stock prices.
- **Lagged Stock Prices:** Similarly, lagged features for stock prices (e.g., the previous day's open, high, low, close) were generated to enhance the predictive modelling process.

The final datasets for Shell and BP contained the following features:

- *Date*
- *Average Title Sentiment*
- *Average Description Sentiment*
- *Average Content Sentiment*
- *Open Price*
- *High Price*
- *Low Price*
- *Close Price*
- *Previous Day Sentiment Scores*
- *Previous Day Stock Prices*

These datasets were then used for sentiment analysis and predictive modelling, laying a solid foundation for achieving the project's objectives. The meticulous pre-processing ensured that the data was clean, consistent, and ready for analysis, thus enhancing the reliability and accuracy of the subsequent models.

5. Sentiment Analysis Methodology

5.1 Overview

Sentiment analysis is a computational method used to determine whether a piece of text expresses a positive, negative, or neutral sentiment. In this project, we utilized the VADER (Valence Aware Dictionary and sEntiment Reasoner) sentiment analyser, which is particularly effective in handling the nuances of social media text and other short, informal communication formats.

5.2 VADER Sentiment Analyser

VADER operates by allocating sentiment scores to each word, phrase and emoticon present in a text. These individual scores are then combined to generate an overall sentiment score for the entire text. The output consists of four scores: positive, neutral, negative, and compound. The compound score, a normalized and weighted composite, ranges from -1 (indicating the most negative sentiment) to +1 (indicating the most positive sentiment).

5.3 Implementation

In this project, VADER was applied to analyse the sentiment of news articles related to the keyword "oil." The implementation process involved the following steps:

1. **Data Collection:** We gathered news articles from various sources, including the Guardian API and NewsAPI, using the keyword "oil".
2. **Pre-processing:** Relevant fields such as the title, description, and content were extracted from the articles.
3. **Sentiment Scoring:** The VADER sentiment analyser was used to assign sentiment scores to each of these fields.

5.4 Example Analysis

To illustrate the application of VADER, here are some examples of sentiment analysis performed on the news articles:

Example 1: Positive Sentiment

- **Title:** "Oil Prices Surge as Demand Increases"
- **Description:** "The global demand for oil has surged, leading to a significant increase in oil prices."
- **Content:** "Recent data shows a substantial rise in oil prices, driven by increased demand across various sectors. Analysts predict that the trend will continue as economic activities pick up pace."

Sentiment Scores:

- **Title:** Positive: 0.717, Neutral: 0.283, Negative: 0.000, Compound: 0.802

- **Description:** Positive: 0.570, Neutral: 0.430, Negative: 0.000, Compound: 0.636
- **Content:** Positive: 0.509, Neutral: 0.491, Negative: 0.000, Compound: 0.784

Example 2: Negative Sentiment

- **Title:** "Oil Spills Cause Environmental Damage"
- **Description:** "Recent oil spills have led to significant environmental damage, affecting marine life and coastal communities."
- **Content:** "Oil spills in various regions have caused severe damage to the environment. The spills have affected marine ecosystems and coastal areas, leading to calls for stricter regulations on oil drilling and transportation."

Sentiment Scores:

- **Title:** Positive: 0.000, Neutral: 0.287, Negative: 0.713, Compound: -0.796
- **Description:** Positive: 0.000, Neutral: 0.420, Negative: 0.580, Compound: -0.648
- **Content:** Positive: 0.000, Neutral: 0.346, Negative: 0.654, Compound: -0.765

6. Text Pre-processing

Before diving into sentiment analysis, the text data underwent pre-processing to prepare it for analysis. This included several key steps:

- **Tokenization:** The text was segmented into individual tokens, including both words and punctuation marks.
- **Normalization:** The text was transformed to lowercase, contractions were fully expanded (for instance, "can't" was changed to "cannot") and extraneous characters were eliminated.

6.1 Lexicon Lookup

VADER uses a predefined lexicon where each word or phrase is associated with a sentiment intensity score between -4 and +4. For example:

- "excellent" has a score of +3.0.
- "terrible" has a score of -3.0.

6.2 Heuristics for Punctuation and Capitalization

VADER also employs certain heuristics to adjust sentiment scores based on punctuation and capitalization:

- **Punctuation Amplification:** Exclamation marks (!) amplify the intensity of the sentiment score.
- **Capitalization:** Words in all caps are considered to have stronger sentiment (e.g., "GREAT" is more intense than "great").

6.3 Heuristics for Negation

Words like "not" or "never" that precede a sentiment-laden word will flip the sentiment score (e.g., "not good" becomes negative).

6.4 Heuristics for Degree Modifiers

Words that modify the intensity of sentiment, such as "very" or "slightly," adjust the sentiment score accordingly. For instance, "very good" would have a higher positive score than just "good".

6.5 Calculation of Compound Score

The total sentiment score is normalized to a value between -1 and +1 using a standard normalization function. This compound score is particularly useful because it provides a single metric that captures the overall sentiment of the text.

$$\text{compound} = \frac{\text{total_score}}{\sqrt{\text{total_score}^2 + \alpha}}$$

Example Calculation:

Input Text: "The new oil discovery is extremely promising!"

1. Tokenization:

- Tokens: ["The", "new", "oil", "discovery", "is", "extremely", "promising", "!"]

2. Lexicon Lookup:

- "promising" has a positive score of +2.0 in the VADER lexicon.
- "extremely" is a degree modifier that increases the intensity of "promising".

3. Heuristics for Punctuation and Capitalization:

- The exclamation mark (!) amplifies the sentiment intensity.

4. Calculation of Raw Sentiment Scores:

- "promising" with "extremely" might be amplified to +2.5.
- The exclamation mark further increases the intensity to +2.75.

5. Summing and Normalization:

- After considering all the factors, the final raw sentiment score is **+2.75**.
- Using the normalization formula:

$$\text{compound} = \frac{2.75}{\sqrt{2.75^2 + 15}} \approx 0.58$$

The normalized score is approximately **0.58**.

The compound score for the input text "The new oil discovery is extremely promising!" is approximately +0.58, indicating a positive sentiment.

7. Distribution of Average Sentiment Scores

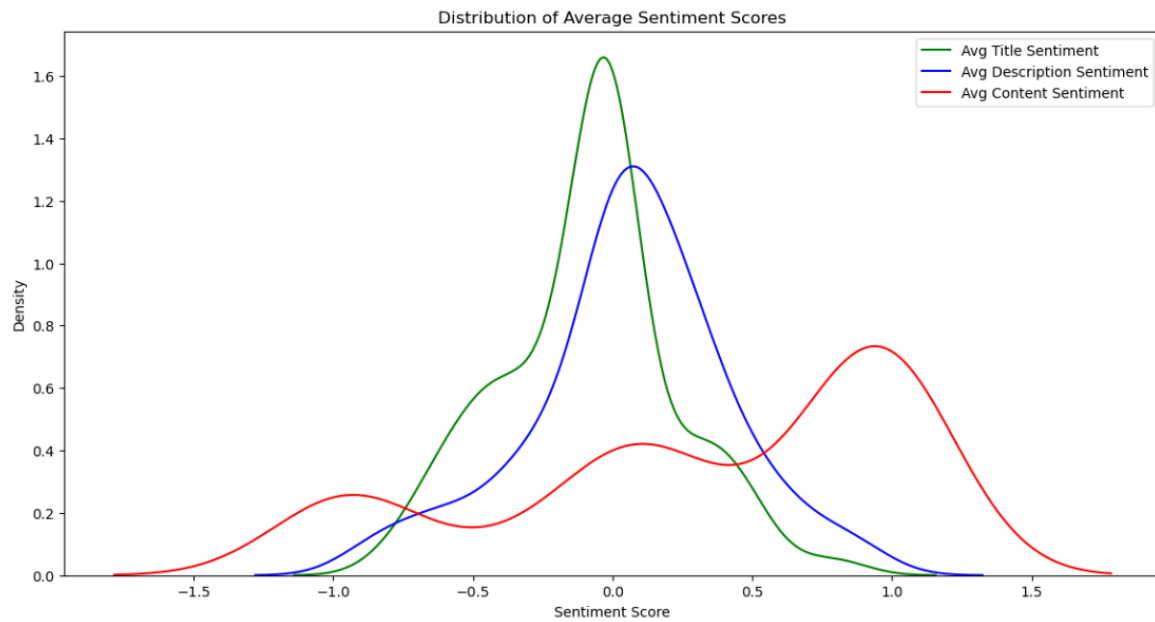


Fig-1 (Distribution of Average Sentiment Scores)

The *Fig-1* illustrates the density distribution of sentiment scores derived from the title, description, and content of news articles. The green line represents the sentiment of the titles, the blue line corresponds to the descriptions, and the red line shows the sentiment of the content. The plot reveals that title sentiment is more concentrated around neutral values, while content sentiment exhibits a broader distribution with noticeable positive and negative extremes.

8. Stock Price Candlestick Chart

Shell Stock Price Candlestick Chart

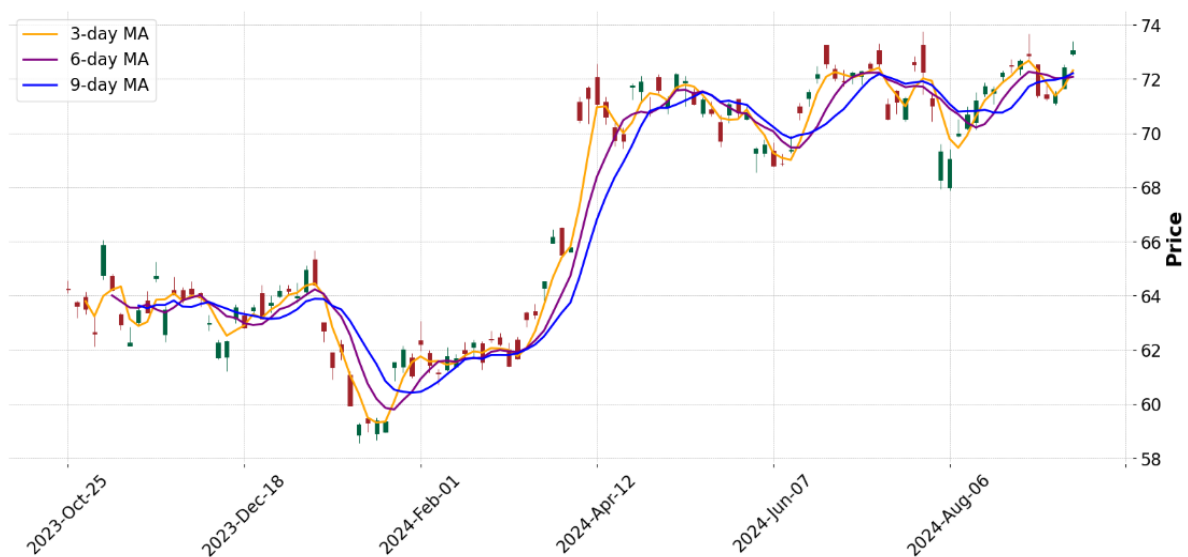


Fig- 2.a (Shell Stock Price Candlestick Chart)

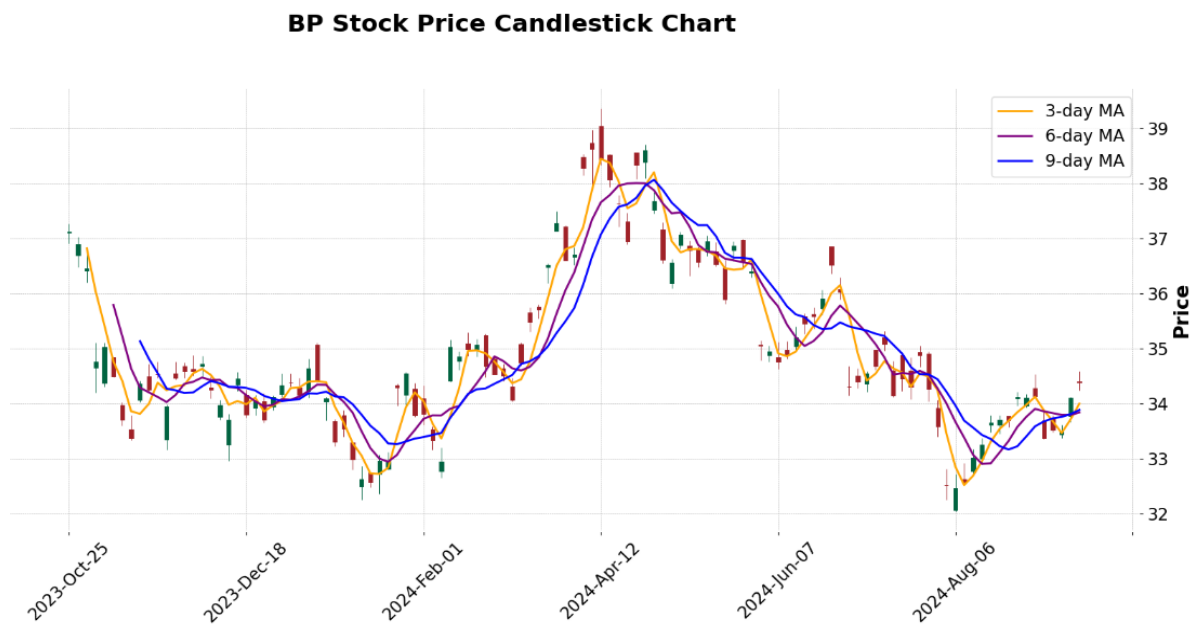


Fig- 2.b (BP Stock Price Candlestick Chart)

The *Fig-2.a* and *Fig-2.b* display the daily price movements of Shell and BP stocks using candlesticks, which represent the open, high, low, and close prices. Both (*Fig-2.a* and *Fig-2.b*) includes three moving averages (MAs) a 3-day MA (orange), a 6-day MA (purple), and a 9-day MA (blue) to smooth out short-term fluctuations and highlight the overall trends in stock prices. The Shell chart shows a more consistent upward trend, while the BP chart reflects more volatility, offering insight into the differing market behaviours of these two major oil companies over time.

3-day Moving Average (Orange Line)

- Purpose: The 3-day moving average is a short-term moving average that reacts quickly to price changes. It is often used by traders looking to capture short-term price movements and can provide early signals of trend reversals.
- Formula:

$$\text{3-day MA} = \frac{P_1 + P_2 + P_3}{3}$$

where P_1 , P_2 , P_3 are the closing prices for the last three days.

- Use: The 3-day MA is highly responsive to recent price changes, making it useful for quickly identifying emerging trends. However, it can also produce false signals due to its sensitivity to short-term fluctuations.

6-day Moving Average (Purple Line)

- Purpose: The 6-day moving average is a medium-term indicator that smooths out more of the daily volatility than the 3-day MA, providing a clearer view of the intermediate trend while still being responsive to recent price changes.
- Formula:

$$\text{6-day MA} = \frac{P_1 + P_2 + P_3 + P_4 + P_5 + P_6}{6}$$

where P_1 to P_6 are the closing prices for the last six days.

- Use: This moving average is more reliable for identifying underlying trends over a slightly longer period, useful for confirming trends detected by shorter-term averages.

9-day Moving Average (Blue Line)

- Purpose: The 9-day moving average offers a longer-term view compared to the 3-day and 6-day MAs, providing a more stable trend line by filtering out short-term fluctuations.
- Formula:

$$\text{9-day MA} = \frac{P_1 + P_2 + P_3 + P_4 + P_5 + P_6 + P_7 + P_8 + P_9}{9}$$

where P_1 to P_9 are the closing prices for the last nine days.

- Use: The 9-day MA is ideal for identifying dominant trends over a longer period, helping traders and analysts make decisions about longer-term positioning.

9. Predictive Modelling

In this project, we employed two primary predictive modelling techniques: Linear Regression and Random Forest Regression. By using both models, our objective was to gain comprehensive insights into how effectively sentiment analysis can predict the stock prices of Shell and BP.

9.1 Linear Regression

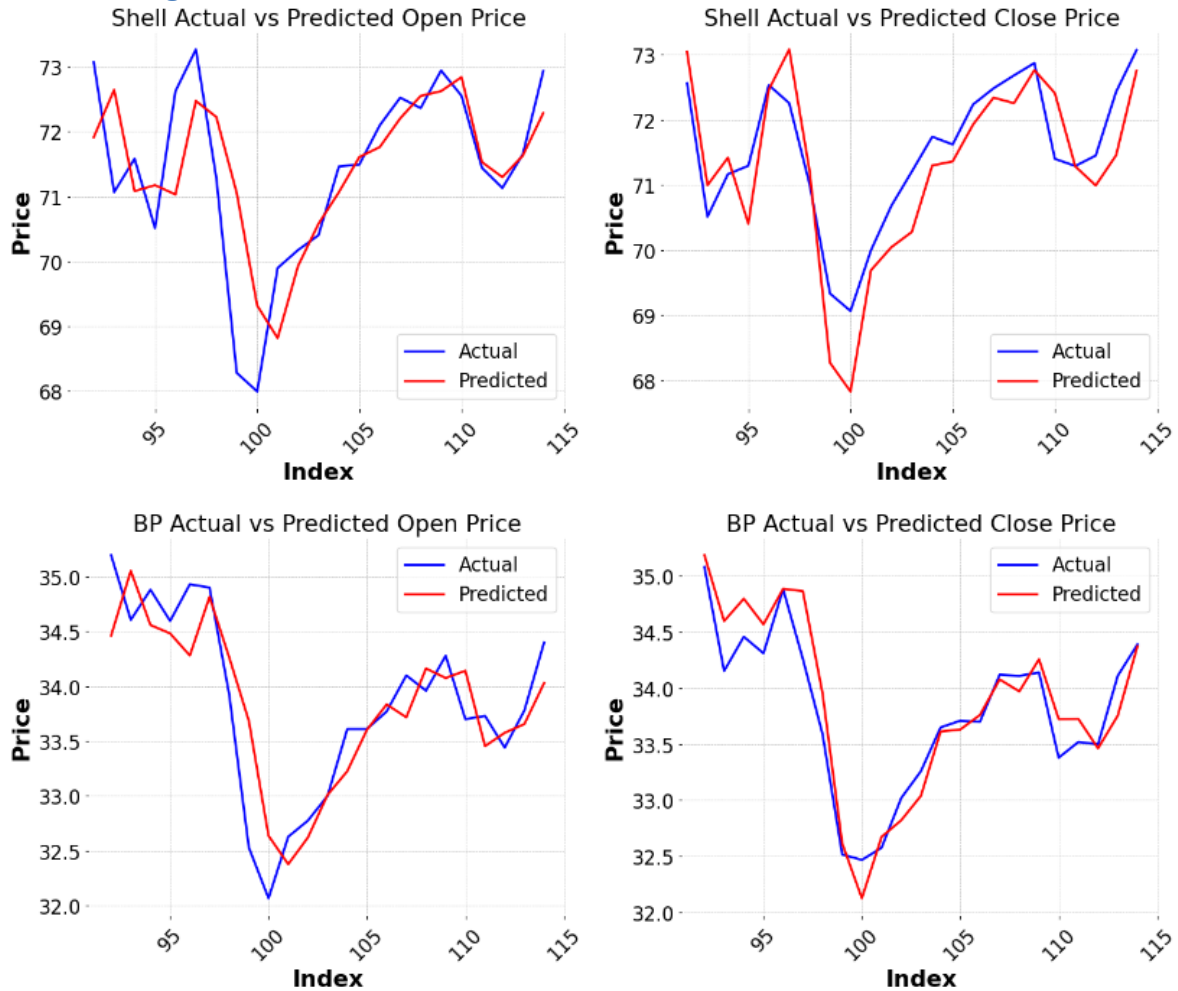


Fig- 3.a (Linear Regression Shell and BP's Actual vs Predicted Open and Close Price Plot)

Fig 3.a presents a comparison between the actual stock prices (indicated by blue lines) and the predicted stock prices (shown by red lines) for Shell and BP over a designated time period, employing Linear Regression. The top row illustrates the open and close prices for Shell, while the bottom row showcases the corresponding data for BP. Observing the alignment and discrepancies between these actual and forecasted values offers insights into the accuracy and performance of the predictive models utilized.

$$\text{Open Price}_t = \beta_0 + \beta_1 \cdot \text{Title Sentiment}_{t-1} + \beta_2 \cdot \text{Description Sentiment}_{t-1} + \beta_3 \cdot \text{Content Sentiment}_{t-1} + \beta_4 \cdot \text{Open Price}_{t-1} + \beta_5 \cdot \text{Close Price}_{t-1} + \epsilon$$

$$\text{Close Price}_t = \beta_0 + \beta_1 \cdot \text{Title Sentiment}_{t-1} + \beta_2 \cdot \text{Description Sentiment}_{t-1} + \beta_3 \cdot \text{Content Sentiment}_{t-1} + \beta_4 \cdot \text{Open Price}_t + \beta_5 \cdot \text{Close Price}_{t-1} + \epsilon$$

Where:

- **Price_t** is the dependent variable, representing the stock price (both open and close) on day **t**.
- **Title Sentiment_{t-1}** is the sentiment score of the title of news articles on the previous day **t-1**.
- **Description Sentiment_{t-1}** is the sentiment score of the description of news articles on the previous day **t-1**.
- **Content Sentiment_{t-1}** is the sentiment score of the content of news articles on the previous day **t-1**.
- **Open Price_{t-1}** and **Close Price_{t-1}** are the stock prices (open, close) from the previous day **t-1**.
- **β₀** is the constant term, indicating the anticipated stock price when all the independent variables have a value of zero.
- **β₁, β₂, ..., β₅** are the coefficients for each independent variable, indicating how much the stock price is expected to change for a one-unit change in the corresponding independent variable.
- **ε** is the error term, capturing the difference between the actual stock price and the price predicted by the model. This accounts for the variation in the stock price that cannot be explained by the independent variables.

9.2 Random Forest Regression

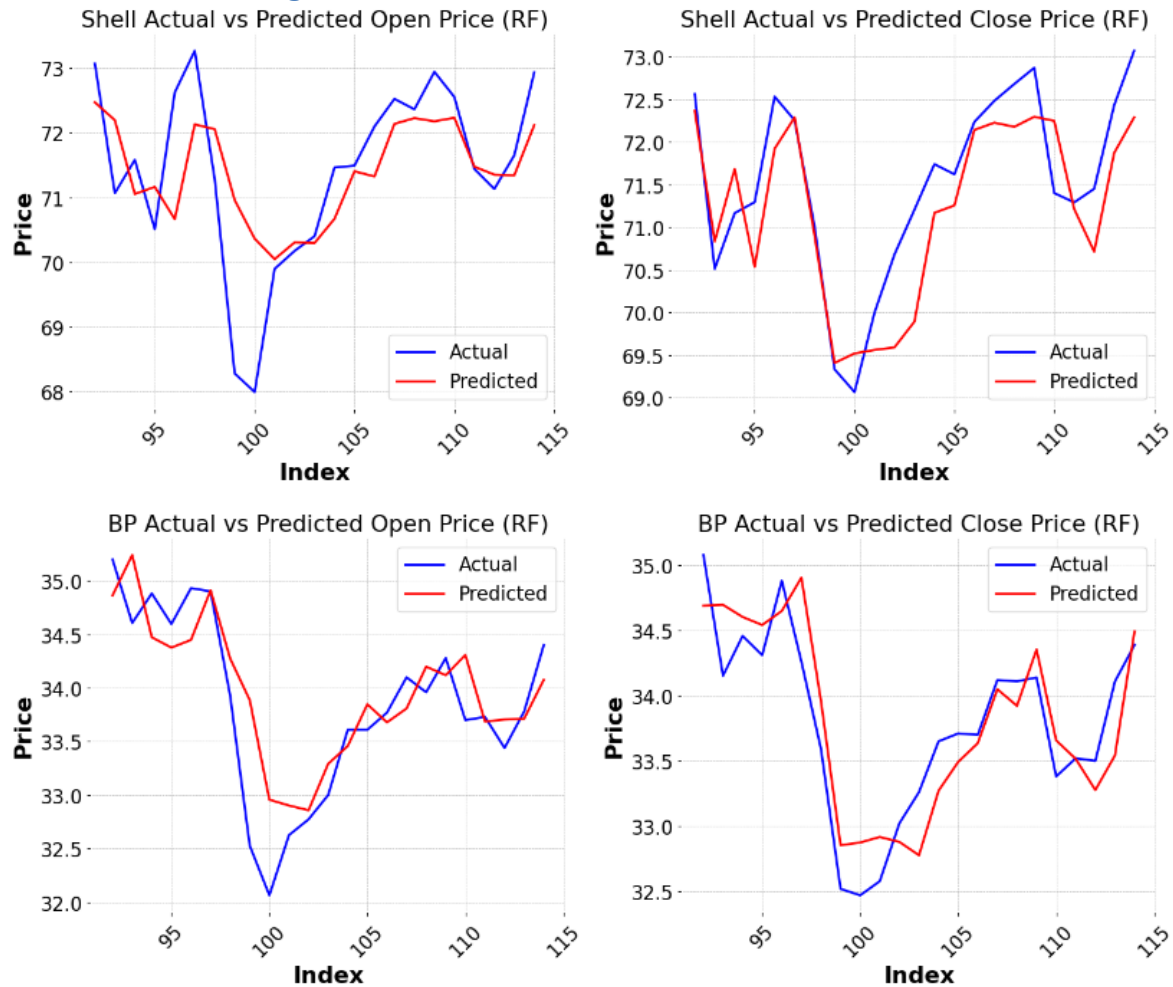


Fig- 3.b (Random Forest Regression Shell and BP's Actual vs Predicted Open and Close Price Plot)

Fig 3.b depicts the performance of the Random Forest Regression model in forecasting the open and close prices for Shell (top row) and BP (bottom row) over a given timeframe. The actual stock prices are represented by the blue lines, while the predicted prices, as determined by the model, are shown in red. The close correspondence between the actual and predicted values highlights the model's effectiveness, with only slight deviations occurring during times of significant price fluctuations.

Random Forest is a type of ensemble learning method that creates multiple decision trees during the training process and combines their outputs to improve overall prediction accuracy. This technique is robust against overfitting and is well-suited for capturing complex interactions between variables.

9.2.1 Variables Used

- **Avg_Title_Sentiment:** The average sentiment score derived from the title of news articles.
- **Avg_Description_Sentiment:** The average sentiment score derived from the description of news articles.
- **Avg_Content_Sentiment:** The average sentiment score derived from the content of news articles.
- **Prev_Open:** The open price of the stock on the previous trading day.
- **Prev_Close:** The close price of the stock on the previous trading day.

9.2.2 Target Variables

- **Open Price (Open):** The actual open price of the stock to be predicted.
- **Close Price (Close):** The actual close price of the stock to be predicted.

9.2.3 Steps in Random Forest Implementation

1. Data Preparation:

- The feature matrix included variables such as Avg_Title_Sentiment, Avg_Description_Sentiment, Avg_Content_Sentiment, Prev_Open, Prev_High, Prev_Low, and Prev_Close.
- The target variables were the stock prices (Open and Close).

2. Random Forest Model Construction:

- The Random Forest model was composed of an ensemble of decision trees, each trained on random subsets of the data and features.
- The model used 500 decision trees (n_estimators) with a maximum depth of 10 for each tree to prevent overfitting while capturing complex patterns in the data.

3. Training the Model:

- The model was trained using historical data, where it identified the relationships between sentiment scores and the previous day's stock prices to forecast the current day's Open and Close prices.

4. Making Predictions:

- After training, the model was used to predict the stock prices for unseen data (the test set). This allowed the evaluation of the model's generalization capability.

5. Model Evaluation:

- The model's performance was evaluated using metrics like Mean Squared Error (MSE) and Root Mean Squared Error (RMSE), offering insights into the accuracy of its predictions.

10. Model Training and Testing

Training and testing phases are essential in building reliable predictive models. In this project, we implemented and assessed two distinct models: Linear Regression and Random Forest Regression, to determine their effectiveness in forecasting stock prices using sentiment analysis from news articles.

10.1 Linear Regression

Training Process:

1. **Data Splitting:** The dataset was partitioned into training and testing sets using an 80-20 ratio. This method allows the model to be trained on one portion of the data while being tested on a separate, unseen portion, which aids in evaluating its ability to generalize.
2. **Model Training:** The Linear Regression model was applied to the training data, where it calculated the coefficients (β values) that reduce the sum of the squared differences between the actual and predicted values.
3. **Feature Selection:** Average sentiment scores from the previous day (title, description, content) and previous day's stock prices (open, close) were used as the independent variables. The target variables were the open and close prices of the stock.

Testing Process:

1. **Prediction:** The trained Linear Regression model was used to predict stock prices on the testing set.
2. **Evaluation:** The model's effectiveness was assessed through Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). These metrics reflect the average squared difference and the square root of the average squared difference between actual and predicted values, respectively.

10.2 Random Forest Regression

Training Process:

1. **Data Splitting:** Following the approach used for Linear Regression, the dataset was divided into training and testing sets with an 80-20 split.
2. **Model Training:** The Random Forest model was trained using the training data, where it constructed an ensemble of decision trees, each one built on a randomly selected subset of the data. The final prediction was derived by averaging the predictions from all the individual trees.
3. **Feature Selection:** The same set of features used in Linear Regression was employed for Random Forest, ensuring a consistent comparison between the models.

Testing Process:

1. **Prediction:** The trained Random Forest model was used to predict stock prices on the testing set.
2. **Evaluation:** The performance was measured using the same metrics (MSE and RMSE) to maintain consistency in evaluation.

11. Evaluation of Results

Metrics

The primary metrics used for evaluation are Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). These metrics provide insights into the accuracy of the models in predicting stock prices based on sentiment analysis of news articles.

- **Mean Squared Error (MSE):** MSE is the average of the squared differences between the actual and predicted values. It measures the average magnitude of the errors, giving a sense of the overall accuracy of the model. A lower MSE indicates better model performance.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Where:

- y_i is the actual value of the i-th data point.
- \hat{y}_i is the predicted value for the i-th data point.
- n is the total number of data points.

- **Root Mean Squared Error (RMSE):** RMSE is the square root of MSE and provides an error metric in the same units as the target variable, making it more interpretable. It also penalizes larger errors more significantly, highlighting significant deviations between actual and predicted values.

$$\text{RMSE} = \sqrt{\text{MSE}}$$

12. Comparison of Models

The performance of Linear Regression and Random Forest Regression models was evaluated using the MSE and RMSE metrics. The results for both Shell and BP stock prices are summarized below.

12.1 Shell Stock Price Prediction

	Linear Regression				Random Forest Regression			
	Mean Error	Squared	Root Squared Error	Mean	Mean Error	Squared	Root Squared Error	Mean
Open Price	0.8827		0.9395		1.0336		1.0167	
Close Price	0.3872		0.6223		0.3477		0.5897	

12.2 BP Stock Price Prediction

	Linear Regression				Random Forest Regression			
	Mean Error	Squared	Root Squared Error	Mean	Mean Error	Squared	Root Squared Error	Mean
Open Price	0.1731		0.4160		0.2046		0.4523	
Close Price	0.0629		0.2509		0.1094		0.3308	

13. Analysis and Insights

13.1 Shell

- **Linear Regression:**
 - **Open Price Prediction:** The Linear Regression model for Shell's open prices shows an MSE of 0.8827 and an RMSE of 0.9395. These metrics suggest that while the model captures some relationships in the data, it may not fully account for all complexities, leading to higher prediction errors.
 - **Close Price Prediction:** For Shell's close prices, the model performs better, with an MSE of 0.3872 and an RMSE of 0.6223. This suggests that the closing prices follow a more predictable pattern, making them easier for the model to forecast with higher accuracy.
- **Random Forest:**
 - **Open Price Prediction:** The Random Forest model, with an MSE of 1.0336 and an RMSE of 1.0167, does not perform as well as the linear regression model for Shell's open prices. This poorer performance may be due to the model overfitting the training data, leading to less accurate predictions on new, unseen data.
 - **Close Price Prediction:** When predicting Shell's close prices, the Random Forest model slightly improves upon the linear regression results, with an MSE of 0.3477 and an RMSE of 0.5897. Although it captures some non-linear relationships, the gains over linear regression are marginal.

13.2 BP

- **Linear Regression:**
 - **Open Price Prediction:** For BP's open price, the linear regression model performs strongly, with an MSE of 0.1731 and an RMSE of 0.4160. These low error values suggest that the relationships in BP's data are well-suited to a linear model, making it a reliable predictor.
 - **Close Price Prediction:** The linear regression model performs exceptionally well for BP's close prices, achieving an MSE of 0.0629 and an RMSE of 0.2509. These metrics indicate that the model is highly effective at capturing the underlying patterns in the data, making it the best-performing model overall.
- **Random Forest:**
 - **Open Price Prediction:** The Random Forest model for BP's open prices has an MSE of 0.2046 and an RMSE of 0.4523, slightly higher than those of the linear regression model. This suggests that the added complexity of the Random Forest model does not significantly improve accuracy for BP's open prices.

- **Close Price Prediction:** With an MSE of 0.1094 and an RMSE of 0.3308, the Random Forest model underperforms compared to linear regression for BP's close prices. This highlights that in this context, simpler models like linear regression are more effective and that the complexity of Random Forest does not necessarily translate into better performance.

14. Learning Curve

14.1 Learning Curve for Shell

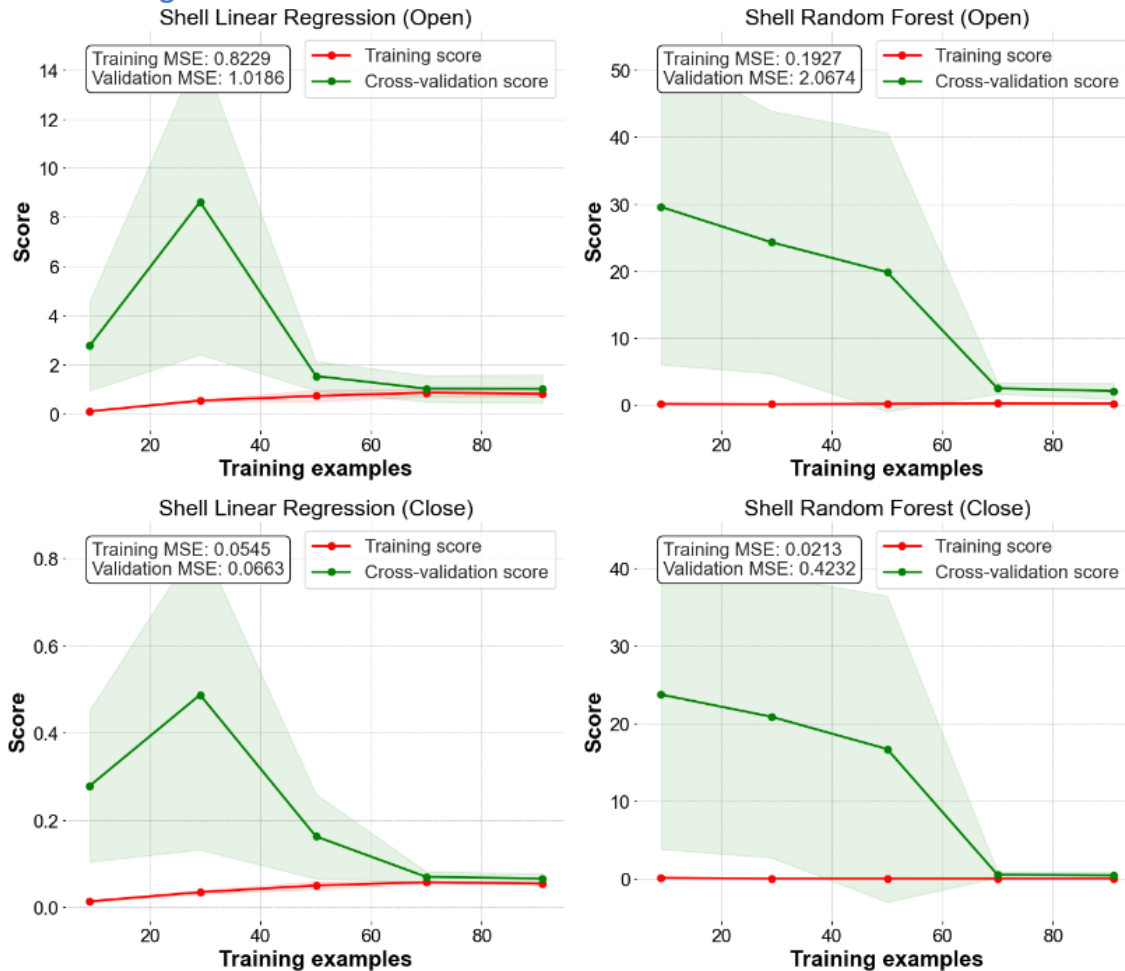


Fig- 4.a (Learning Curve of Linear regression, Random Forest Regression for Shell Open and Close Price)

The above plots (Fig- 4.a) illustrate the performance of Linear Regression and Random Forest models in predicting Shell's Open and Close prices. The graphs demonstrate how the Linear Regression model achieves a lower Mean Squared Error (MSE) and better generalization across the training examples compared to the Random Forest model. The Random Forest model, particularly in predicting the Open prices, shows signs of overfitting, as indicated by the higher validation MSE and divergence between the training and validation scores. The plot contains four subplots:

1. **Top Left: Shell Linear Regression (Open)**
2. **Top Right: Shell Random Forest (Open)**

3. Bottom Left: Shell Linear Regression (Close)

4. Bottom Right: Shell Random Forest (Close)

Details of these subplots:

Shell (Open)

	Linear Regression	Random Forest Regression
Training MSE	0.8229	0.1927
Validation MSE	1.0186	2.0674
Training Score (Red Line)	The training error, represented by the red line, stays consistently low across the number of training examples, showing that the model fits the training data quite well.	The training error is very low across all examples, indicating that the model fits the training data almost perfectly.
Cross-Validation Score (Green Line)	The cross-validation error, depicted by the green line, starts out relatively high but decreases significantly as more data is added. This suggests that the model's ability to generalize to new data improves as it learns from more examples.	However, the cross-validation error remains significantly higher than the training error, which suggests that the model is struggling to generalize to new, unseen data. The large gap highlights potential overfitting issues.
Performance on Training Data	The steady and low training MSE indicates that the Linear Regression model effectively fits the training data without overfitting.	The very low training MSE suggests that the Random Forest model is overfitting, capturing noise along with the true signal in the training data.
Performance on Validation Data	The initial high cross-validation error decreases with more data, showing that the model is learning and improving its generalization. The convergence between training and cross-validation errors suggests a good balance between bias and variance, leading to more reliable predictions for Shell's open prices.	The consistently high cross-validation MSE indicates poor generalization. Despite increasing the amount of training data, the model's complexity seems to hinder its ability to perform well on new data, highlighting the risks of overfitting.

Shell (Close)

	Linear Regression	Random Forest Regression
Training MSE	0.0545	0.0213
Validation MSE	0.0663	0.4232
Training Score (Red Line)	The training error remains very low and consistent, indicating a good fit to the training data.	The training error is very low, indicating that the model captures the training data almost perfectly.
Cross-Validation Score (Green Line)	The cross-validation error stays close to the training error, suggesting that the model generalizes well to unseen data.	The cross-validation error is notably higher, though it decreases slightly as more training data is added.
Performance on Training Data	The low training MSE indicates that the Linear Regression model is accurately capturing the linear relationships in the training data without overfitting.	The very low training MSE shows that the Random Forest model fits the training data well, but this also hints at overfitting.
Performance on Validation Data	The close alignment between the training and cross-validation errors suggests that the model is robust and reliable for predicting Shell's closing prices. The low validation MSE further confirms the model's effectiveness in capturing the true patterns in the data.	The relatively higher validation MSE suggests that, despite capturing complex patterns in the training data, the model struggles to generalize as effectively as the Linear Regression model. The persistent gap between training and validation errors indicates overfitting, where the model's complexity doesn't necessarily lead to better predictions on unseen data.

14.2 Learning Curve for BP

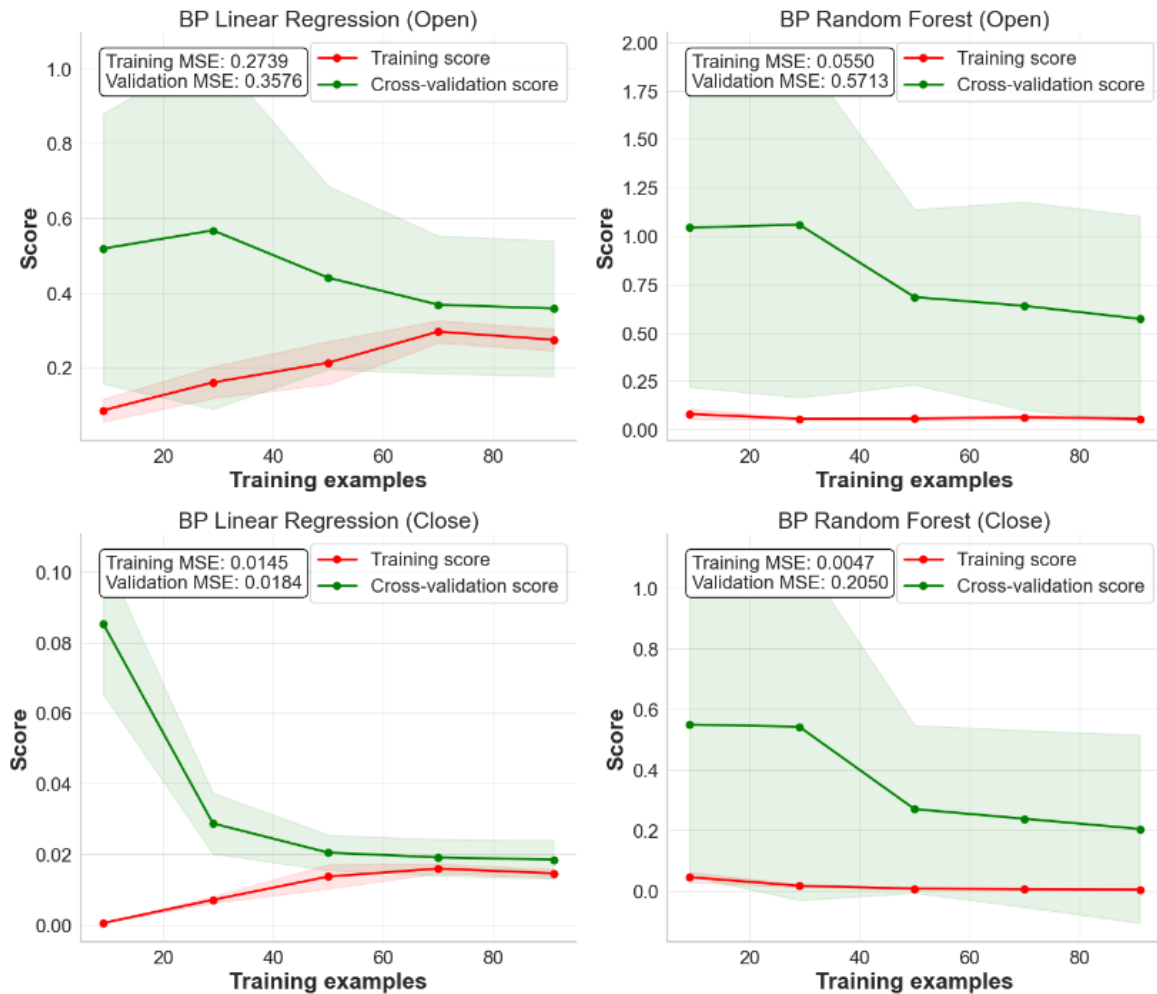


Fig- 4.b (Learning Curve of Linear regression, Random Forest Regression for BP Open and Close Price)

The above learning curve plots (Fig- 4.b) compare the accuracy of Linear Regression and Random Forest models in forecasting BP's Open and Close stock prices. The Linear Regression model consistently outperforms the Random Forest model, as evidenced by lower MSE values and closer alignment between training and cross-validation scores. The Random Forest model, while still performing reasonably well, shows a slightly higher validation MSE, indicating less effective generalization, particularly in predicting BP's Open prices. The plot contains four subplots:

1. **Top Left: BP Linear Regression (Open)**
2. **Top Right: BP Random Forest (Open)**
3. **Bottom Left: BP Linear Regression (Close)**
4. **Bottom Right: BP Random Forest (Close)**

Details of these subplots:

BP (Open)

	Linear Regression	Random Forest Regression
Training MSE	0.2739	0.0550
Validation MSE	0.3576	0.5713
Training Score (Red Line)	The training error, indicated by a consistently low score as the number of training examples grows, demonstrates that the model effectively fits the training data.	The training score remains very low, showing that the model captures the training data almost perfectly, which might indicate overfitting.
Cross-Validation Score (Green Line)	The cross-validation score, starts higher but gradually decreases with more training examples, suggesting that the model's ability to generalize improves as it is exposed to more data.	The cross-validation error, remains significantly higher than the training error and decreases only slightly as more training examples are added, highlighting the model's struggle to generalize.
Performance on Training Data	The low and steady training MSE indicates that the Linear Regression model is accurately fitting the training data without overfitting.	The very low training MSE indicates that the Random Forest model is overfitting to the training data, potentially capturing noise along with meaningful patterns.
Performance on Validation Data	As more data is incorporated, the decline in cross-validation error signifies that the model is improving its ability to generalize to new, unseen data. The narrowing gap between training and validation errors indicates that the model is striking a good balance between bias and variance, positioning it as a reliable predictor for BP's open prices.	The persistently high cross-validation error points to inadequate generalization. Even with the addition of more training data, the model's complexity may be contributing to overfitting, resulting in strong performance on the training set but weaker outcomes when applied to new data.

BP (Close)

	Linear Regression	Random Forest Regression
Training MSE	0.0145	0.0047
Validation MSE	0.0184	0.2050
Training Score (Red Line)	The training error remains minimal, demonstrating that the model fits the training data well.	The training error is very low, suggesting that the model fits the training data extremely well.
Cross-Validation Score (Green Line)	The cross-validation error is slightly higher but remains close to the training error, which suggests strong generalization to unseen data.	The cross-validation error is notably higher than the training error and does not decrease significantly as more data is added, indicating challenges with generalization.
Performance on Training Data	The low training MSE suggests that the Linear Regression model is accurately capturing the linear relationships in the training data.	The extremely low training MSE suggests that the Random Forest model could be overfitting, as it may be picking up on detailed patterns in the training data that don't necessarily apply to new, unseen data.
Performance on Validation Data	The close alignment between the training and cross-validation scores indicates that the model is well-balanced and generalizes effectively, making it a robust predictor for BP's closing prices.	The fact that the validation MSE is higher than the training MSE indicates that the model doesn't generalize as effectively as the Linear Regression model, even though it captures complex relationships within the training data. The disparity between training and validation errors is a sign of overfitting, where the model's complexity fails to improve its performance on out-of-sample data.

15. Discussion and Analysis

15.1 Interpretation of Results

As we delved into the world of stock prices and sentiment analysis, what we found was both intriguing and insightful. Picture this: you're an investor trying to predict the unpredictable stock prices. Traditionally, you'd look at numbers, trends, maybe even some gut feelings. But here, we introduced sentiment, the collective mood of the market, as a new player in the game.

For Shell, our journey with the Linear Regression model was particularly rewarding. Imagine navigating a vast ocean of data, and amid the waves, finding a steady current that guides you. That's what happened with Shell's closing prices. The Linear Regression model, with its simplicity and directness, seemed to grasp the underlying patterns in the data, giving us predictions that were both reliable and accurate. It was like finding a consistent rhythm in the market's pulse, especially when predicting those all-important closing prices.

However, the road wasn't as smooth when we turned to the Random Forest model. Think of Random Forest as a tool with many moving parts, each trying to capture every nuance of the data. For Shell's open prices, this complexity became its downfall. The model, while powerful, seemed to get lost in the details, leading to overfitting. It's as if it tried to read too much into the story, missing the broader narrative. When it came to Shell's close prices, the Random Forest model did manage to make some headway, showing slight improvements, but the gains were modest compared to the simplicity and efficiency of Linear Regression.

Now, let's shift our focus to BP. Here, the Linear Regression model was nothing short of a rock star. It handled both open and close prices with finesse, producing some of the lowest error metrics in our study. It was as if the BP data was speaking a language that Linear Regression understood perfectly—a straightforward, linear relationship that translated into highly accurate predictions.

But when we brought in the Random Forest model, expecting it to uncover hidden complexities, we were met with a surprise. The model, with all its power to handle non-linearity, didn't quite live up to expectations. For BP's open prices, the additional complexity didn't translate into better predictions. The error margins crept up, showing us that sometimes, less really is more. Even for BP's close prices, where we hoped Random Forest might shine, it lagged behind Linear Regression. It was a clear reminder that in the world of data science, complexity doesn't always equate to better performance.

15.2 Model Comparison and Performance

As we compared the two models, a clear narrative emerged. Linear Regression, with its elegant simplicity, consistently outperformed the more complex Random Forest model, especially when it came to BP's stock prices. It was as if the market was whispering secrets in a straightforward manner, and Linear Regression, with its keen ears, was able to pick them up perfectly.

On the other hand, Random Forest, while capable of handling intricate, non-linear interactions, seemed to struggle with the data at hand. It was like trying to solve a simple puzzle with an overly complicated approach—sometimes, the simplest tools are the most

effective. The superior performance of Linear Regression aligns with what we've seen in literature, where simpler models often outperform their more complex counterparts in financial predictions, particularly when the relationships in the data are predominantly linear.

This result challenges a common assumption in financial forecasting: that more complex models like Random Forest always yield better results. Our findings suggest otherwise, especially in contexts where the data relationships are more linear. It's a reminder that sometimes, the best solution is the one that cuts through the noise and focuses on the fundamentals.

15.3 Limitations and Practical Implications

But of course, our journey wasn't without its challenges. Every map has its uncharted territories, and in this project, one of those areas was the potential influence of other factors on stock prices elements we couldn't capture within the scope of our data. Macroeconomic factors, industry-specific events, or even unforeseen global shifts could all play a role, yet they remained outside the reach of our models.

Moreover, while VADER, our trusty sentiment analyser, did a commendable job, it wasn't without its limitations. Financial news can be complex, filled with jargon and subtleties that a straightforward sentiment analyser might miss. It's like trying to read between the lines of a nuanced conversation sometimes, the real meaning is hidden in the details.

From a practical standpoint, our findings reinforce the value of sentiment analysis in financial forecasting, but they also highlight the importance of choosing the right tools. The simplicity and interpretability of Linear Regression made it a standout choice in this context. It's like having a tool that not only does the job but also makes it easy to understand how it's done an invaluable trait in the world of finance.

On the other hand, while Random Forest holds promise, particularly in more complex or larger datasets, our experience shows that it requires careful tuning and validation. Otherwise, there's a real risk of overfitting capturing the noise instead of the signal.

15.4 Addressing Project Objectives and Research Questions

As we bring this narrative to a close, it's worth reflecting on the initial questions that guided our journey. Could sentiment analysis improve the accuracy of stock price predictions for Shell and BP? The answer, based on our findings, is a resounding yes especially when paired with a model like Linear Regression that knows how to listen to the data without overcomplicating things.

We set out with clear objectives: to analyse sentiment, build models, and evaluate their performance. Along the way, we not only met these goals but also gained deeper insights into the delicate balance between simplicity and complexity in predictive modelling. Our journey underscored the importance of choosing the right model for the task at hand a lesson that resonates far beyond this specific project.

16. Key Findings

- **Sentiment Analysis:**

- Using VADER, we were able to quantify the sentiment of news articles related to "oil," giving us a tangible measure of the market's mood.
- Positive sentiment was generally associated with news of rising oil demand or successful projects, often leading to higher stock prices.
- Negative sentiment, on the other hand, tended to correlate with adverse events like oil spills or regulatory challenges, typically resulting in declining stock prices.
- While neutral sentiment had a less pronounced effect, it still contributed to the overall sentiment landscape, helping us build a more complete picture of market dynamics.

- **Predictive Modelling:**

- We implemented both Linear Regression and Random Forest models, each bringing its own strengths and weaknesses to the table.
- Linear Regression consistently provided better performance, particularly for predicting closing prices. Its ability to capture linear relationships made it a powerful tool in this context.
- Random Forest, despite its capability to handle non-linear relationships, struggled with overfitting and did not outperform Linear Regression, especially in the context of our dataset.

- **Comparative Insights:**

- The lower MSE and RMSE values achieved by Linear Regression highlighted its superior predictive accuracy compared to Random Forest in this study.
- The simplicity and interpretability of Linear Regression made it an invaluable tool for financial analysis, demonstrating that sometimes, the simplest solutions are the most effective.
- While Random Forest offers theoretical advantages in handling complex data patterns, its performance in this project underscored the importance of careful model selection and tuning.

17. Conclusion

In the end, this project was about more than just numbers and models it was about understanding the subtle, often unpredictable forces that drive the financial markets. We set out to explore the relationship between news sentiment and stock prices for Shell and BP, and what we found was both enlightening and humbling.

The story that unfolded showed us that Linear Regression, with its straightforward approach, was more than up to the task. It consistently outperformed the more complex Random Forest model in predicting both open and close prices, especially for BP. This result challenges the assumption that more complexity always leads to better predictions, reminding us that in the world of financial forecasting, sometimes the simplest tools are the most powerful.

But beyond the technicalities, this project also highlighted the potential of sentiment analysis to enrich our understanding of the markets. By combining sentiment scores with historical stock data, we were able to improve the accuracy of our predictions, offering a glimpse into how market sentiment shapes financial outcomes.

For investors, financial analysts, and algorithmic traders, these insights could be invaluable. By integrating sentiment analysis into their forecasting strategies, they can gain a deeper understanding of market dynamics, providing an edge in decision-making especially in industries like oil, where sentiment plays a crucial role.

Looking ahead, the journey doesn't end here. There's potential to expand the dataset, incorporate more diverse sentiment analysis tools, and explore new models like LSTM networks that could further enhance predictive accuracy. If this project were to continue, the next steps would involve refining the current models, incorporating more diverse data, and testing our findings across different industries. In the end, this project has not only met its objectives but also opened up new avenues for exploration in the fascinating intersection of sentiment analysis and financial forecasting.

18. References

- Bhuriya, D., Kaushal, G., Sharma, A., & Singh, U. (2021) 'Prediction of stock values changes using sentiment analysis of stock news headlines', *Journal of Statistics and Management Systems*, 24(1), pp. 1-13. Available at: <https://www.tandfonline.com/doi/full/10.1080/24751839.2021.1874252#abstract>
- Nauta, M. (2021) 'Forecasting the Stock Market using News Sentiment Analysis', Tilburg University. Available at: <https://arno.uvt.nl/show.cgi?fid=157031>
- Singh, A., & Srivastava, S. (2022) 'Using Market News Sentiment Analysis for Stock Market Prediction', *Mathematics*, 10(22), p. 4255. Available at: <https://www.mdpi.com/2227-7390/10/22/4255>
- Shahi, T., & Gandhi, V. (2021) 'Stock Price Movement Prediction Using Sentiment Analysis and CandleStick Chart Representation', *Sensors*, 21(23), p. 7957. Available at: <https://www.mdpi.com/1424-8220/21/23/7957>
- Kaur, A. (2021) 'Predicting Stock Investments Based on Sentiment and Historical Price Data', London Metropolitan University. Available at: <https://repository.londonmet.ac.uk/8809/>
- Nousi, C. (2020) 'Stock Market Prediction using Sentiment Analysis', International Hellenic University. Available at: <https://repository.iuh.edu.gr/xmlui/bitstream/handle/11544/29741/Nousi%20Christina%20Market%20Prediction.pdf?sequence=1>
- Deng, J. (2020) 'STOCK PREDICTION BY ANALYZING FINANCIAL NEWS SENTIMENT AND INVESTOR MOOD OF SOCIAL MEDIA', ProQuest Dissertations & Theses. Available at: <https://www.proquest.com/openview/835c981d090ba9048b9ec276320c0d25/1?cbl=18750&diss=y&pq-origsite=gscholar&parentSessionId=jcC57HN43oHMLjDRkZCd2xQgZtrZ3SKGVm591PMqw3I%3D>
- Zhou, C., & Zhang, Y. (2019) 'Predicting stock market by sentiment analysis and deep learning', *Central European Journal of Social Sciences and Humanities*. Available at: <https://cejsh.icm.edu.pl/cejsh/element/bwmeta1.element.desklight-a0874fed-9c81-4fd6-8767-118e0774b8c7>
- Yadav, N., & Dwivedi, S. (2022) 'Study on various stock prediction techniques with news sentiment', *AIP Conference Proceedings*, 2790, p. 020088. Available at: <https://pubs.aip.org/aip/acp/article/2790/1/020088/2908137/Study-on-various-stock-prediction-techniques-with>
- Gupta, S., & Chouhan, S. (2022) 'Stock Price Prediction using Sentiment Analysis and Deep Learning for Indian Markets', arXiv preprint arXiv:2204.05783. Available at: <https://arxiv.org/abs/2204.05783>

- Kalra, S., & Prasad, J. S. (2019) 'Efficacy of News Sentiment for Stock Market Prediction', 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (Com-IT-Con). IEEE, pp. 491-496. Available at: <https://ieeexplore.ieee.org/abstract/document/8862265>
- Schumaker, R. P., Zhang, Y., & Huang, C. (2009) 'Textual Analysis of Stock Market Prediction Using Breaking Financial News: The AZFinText System', ACM Transactions on Information Systems, 27(2), pp. 1-19. Available at: <https://dl.acm.org/doi/abs/10.1145/1462198.1462204>
- Joshi, K., Bharathi, H. N., & Rao, J. (2016) 'STOCK TREND PREDICTION USING NEWS SENTIMENT ANALYSIS', arXiv preprint arXiv:1607.01958. Available at: <https://arxiv.org/abs/1607.01958>
- Wang, L., & Li, X. (2021) 'Combined deep learning classifiers for stock market prediction: integrating stock price and news sentiments', Kybernetes, 50(1), pp. 123-138. Available at: <https://www.emerald.com/insight/content/doi/10.1108/K-06-2021-0457/full/html>
- Saha, S., & Paul, S. (2016) 'Predicting the direction of stock market prices using random forest', arXiv preprint arXiv:1605.00003. Available at: <https://arxiv.org/abs/1605.00003>
- Polamuri, S. K., & Reddy, M. R. (2020) 'Stock Market Prices Prediction using Random Forest and Extra Tree Regression', ResearchGate. Available at: https://www.researchgate.net/profile/Dr-Polamuri/publication/347994783_Stock_Market_Prices_Prediction_using_Random_Forest_and_Extra_Tree_Regression/links/5fec36eb299bf1408859fdf3/Stock-Market-Prices-Prediction-using-Random-Forest-and-Extra-Tree-Regression.pdf

19. Appendix

```
# Importing libraries

import pandas as pd

import requests

import concurrent.futures

from datetime import datetime, timedelta

from nltk.sentiment.vader import SentimentIntensityAnalyzer

import yfinance as yf

import numpy as np

from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split, learning_curve

from sklearn.linear_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_squared_error

from math import sqrt

import matplotlib.pyplot as plt

import seaborn as sns

import mplfinance as mpf

import nltk


# Download the VADER lexicon required for sentiment analysis

nltk.download('vader_lexicon')


# Define API keys

API_KEY_GUARDIAN = "b001c2f9-9358-4483-a3bc-da632b995c95"

NEWS_API_KEY = "d20bdbb6e7dc4cc098d1437f594d4721"
```

```
SHELL_TICKER = 'SHEL'
```

```
BP_TICKER = 'BP'
```

```
# Initialize VADER sentiment analyzer
```

```
analyzer = SentimentIntensityAnalyzer()
```

```
### Fetch and Analyse Guardian News Data
```

```
def fetch_data_guardian(url):
```

```
    """
```

```
    Fetch data from the Guardian API.
```

```
    Parameters:
```

```
        url (str): API endpoint for the Guardian news data.
```

```
    Returns:
```

```
        dict: JSON response from the API.
```

```
    """
```

```
    try:
```

```
        response = requests.get(url)
```

```
        response.raise_for_status()
```

```
        return response.json()
```

```
    except requests.exceptions.RequestException as e:
```

```
        print(f"Error fetching data from {url}: {e}")
```

```
    return None
```

```

def fetch_article_content_guardian(api_url):
    """
    Fetch the content and description of an article from the Guardian API.

    Parameters:
        api_url (str): API endpoint for the specific article.

    Returns:
        tuple: Article content and description.
    """
    try:
        response = requests.get(api_url + "?api-key=" + API_KEY_GUARDIAN + "&show-
fields=body,trailText")
        response.raise_for_status()
        data = response.json()
        content = data['response']['content']['fields'].get('body', 'No content available')
        description = data['response']['content']['fields'].get('trailText', 'No description
available')

        return content, description
    except requests.exceptions.RequestException as e:
        print(f"Error fetching article content from {api_url}: {e}")
        return None, None
    except KeyError as e:
        print(f"Error parsing article content: {e}")
        return None, None

```

```
def extract_info_guardian(json_data):
```

```
    """
```

Extract relevant information from the JSON response.

Parameters:

json_data (dict): JSON response from the Guardian API.

Returns:

list: Extracted information including title, section name, publication date, and API URL.

```
    """
```

```
    if not json_data:
```

```
        return []
```

```
    try:
```

```
        articles = json_data['response']['results']
```

```
        return [
```

```
            {
```

```
                'title': article['webTitle'],
```

```
                'sectionname': article['sectionName'],
```

```
                'publisheddate': article['webPublicationDate'],
```

```
                'api_url': article['apiUrl']
```

```
            } for article in articles
```

```
        ]
```

```
    except KeyError as e:
```

```
        print(f"Error parsing data: {e}")
```

```
        return []
```

```
def analyze_sentiment_vader(text):
```

```
    """
```

```
    Analyze the sentiment of the given text using VADER.
```

```
    Parameters:
```

```
        text (str): The text to analyze.
```

```
    Returns:
```

```
        float: The compound sentiment score.
```

```
    """
```

```
    vs = analyzer.polarity_scores(text)
```

```
    return vs['compound']
```

```
def fetch_guardian_news():
```

```
    """
```

```
    Fetch and analyze Guardian news articles related to oil.
```

```
    Returns:
```

```
        pd.DataFrame: DataFrame containing the sentiment scores of Guardian news articles.
```

```
    """
```

```
    # Define a list of URLs to fetch data from multiple pages
```

```
    urllist = []
```

```
    for i in range(1, 17):
```

```
        base_url = "https://content.guardianapis.com/business/oil?from-date=2023-01-01&api-key=" + API_KEY_GUARDIAN + "&type=article&page="
```

```
        url = base_url + str(i)
```



```

urllist.append(url)

urls = urllist

# Use ThreadPoolExecutor to fetch data concurrently
with concurrent.futures.ThreadPoolExecutor(max_workers=10) as executor:
    json_data_list = list(executor.map(fetch_data_guardian, urls))

info = []

for json_data in json_data_list:
    info.extend(extract_info_guardian(json_data))

if info:
    # Fetch content and description for each article concurrently
    with concurrent.futures.ThreadPoolExecutor(max_workers=10) as executor:
        content_and_descriptions = list(executor.map(fetch_article_content_guardian,
[article['api_url'] for article in info]))

    for i, (content, description) in enumerate(content_and_descriptions):
        if content and description:
            info[i]['content'] = content
            info[i]['description'] = description
        else:
            info[i]['content'] = 'No content available'
            info[i]['description'] = 'No description available'

    info[i].pop('api_url') # Remove api_url from the final data

```

```

# Convert to DataFrame

df = pd.DataFrame(info)


# Perform sentiment analysis on title, description, and content
df['title_sentiment'] = df['title'].map(analyze_sentiment_vader)
df['description_sentiment'] = df['description'].map(analyze_sentiment_vader)
df['content_sentiment'] = df['content'].map(analyze_sentiment_vader)


# Convert publisheddate to datetime format and then to date
df['publisheddate'] = pd.to_datetime(df['publisheddate']).dt.date


# Select only numeric columns for grouping
sentiment_columns = ['title_sentiment', 'description_sentiment', 'content_sentiment']


# Group by date and calculate the mean of sentiment scores
sentiment_df = df.groupby('publisheddate')[sentiment_columns].mean().reset_index()


# Rename 'publisheddate' to 'date'
sentiment_df.rename(columns={'publisheddate': 'date'}, inplace=True)


# Save the sentiment analysis DataFrame to a CSV file
sentiment_df.to_csv('guardian_oil_articles_vader_sentiment.csv', index=False)


return sentiment_df
else:
    print("No data found")

```

```
return pd.DataFrame()
```

Fetch and Analyse NewsAPI Data

```
def fetch_news_for_date(api_key, query, date):
```

```
    """
```

Fetch news articles from the News API for a specific date.

Parameters:

api_key (str): News API key.

query (str): Query string for news articles.

date (str): Date in YYYY-MM-DD format.

Returns:

dict: JSON response from the News API.

```
    """
```

```
    url = (f"https://newsapi.org/v2/everything?"
```

```
          f"q={query}&from={date}&to={date}&"
```

```
          f"sortBy=publishedAt&apiKey={api_key}&pageSize=100&page=1")
```

```
    response = requests.get(url)
```

```
    return response.json()
```

```
def preprocess_news(articles):
```

```
    """
```

Preprocess the news articles DataFrame by removing duplicates and null values.

Parameters:

articles (list): List of articles with their details.

Returns:

pd.DataFrame: Preprocessed DataFrame containing news articles.

```
"""
```

```
news_df = pd.DataFrame(articles)
news_df['publishedAt'] = pd.to_datetime(news_df['publishedAt'])
news_df = news_df.drop_duplicates(subset='url')
news_df = news_df[['publishedAt', 'title', 'description', 'content', 'url']]
news_df = news_df.dropna(subset=['title', 'description', 'content'])
return news_df
```

```
def fetch_and_aggregate_sentiment(api_key, query, start_date, end_date):
```

```
"""
```

Fetch and aggregate sentiment scores for news articles within a date range.

Parameters:

api_key (str): News API key.

query (str): Query string for news articles.

start_date (datetime): Start date for fetching articles.

end_date (datetime): End date for fetching articles.

Returns:

pd.DataFrame: DataFrame containing aggregated sentiment scores by date.

```

"""

current_date = start_date

all_aggregate_sentiments = []

# Iterate over each date in the range
while current_date <= end_date:

    news_data = fetch_news_for_date(api_key, query, current_date.strftime('%Y-%m-%d'))

    if 'articles' in news_data:

        articles = news_data['articles']

        news_df = preprocess_news(articles)

        if not news_df.empty:

            # Perform sentiment analysis on title, description, and content

            news_df['title_sentiment'] = news_df['title'].apply(analyze_sentiment_vader)

            news_df['description_sentiment'] =
news_df['description'].apply(analyze_sentiment_vader)

            news_df['content_sentiment'] =
news_df['content'].apply(analyze_sentiment_vader)

            # Aggregate sentiment scores by date

            aggregate_sentiment = {

                'date': current_date.strftime('%Y-%m-%d'),

                'mean_title_sentiment': news_df['title_sentiment'].mean(),

                'mean_description_sentiment': news_df['description_sentiment'].mean(),

                'mean_content_sentiment': news_df['content_sentiment'].mean()

            }

```

```

else:

    aggregate_sentiment = {

        'date': current_date.strftime('%Y-%m-%d'),

        'mean_title_sentiment': None,

        'mean_description_sentiment': None,

        'mean_content_sentiment': None

    }

else:

    print(f"No articles found for {current_date.strftime('%Y-%m-%d')} or API error.")

    aggregate_sentiment = {

        'date': current_date.strftime('%Y-%m-%d'),

        'mean_title_sentiment': None,

        'mean_description_sentiment': None,

        'mean_content_sentiment': None

    }

    all_aggregate_sentiments.append(aggregate_sentiment)

    current_date += timedelta(days=1)

return pd.DataFrame(all_aggregate_sentiments)

### Main Execution: Fetch and Combine Sentiments

# Define query and date range for fetching news data
news_query = 'oil'

news_start_date = datetime.strptime('2024-07-27', '%Y-%m-%d')

```

```

news_end_date = datetime.strptime('2024-08-26', '%Y-%m-%d')

# Fetch and aggregate sentiment scores for the specified date range

news_sentiments_df = fetch_and_aggregate_sentiment(NEWS_API_KEY, news_query,
news_start_date, news_end_date)

# Ensure 'date' is in date format

news_sentiments_df['date'] = pd.to_datetime(news_sentiments_df['date']).dt.date

# Group by date and calculate the mean of sentiment scores where dates overlap

news_sentiments_df = news_sentiments_df.groupby('date').mean().reset_index()

# Save the aggregated sentiment DataFrame to a CSV file

news_sentiments_df.to_csv('news_vader_sentiment.csv', index=False)

# Display the aggregated sentiment DataFrame

print(news_sentiments_df)

# Fetch Guardian news sentiments

guardian_sentiments_df = fetch_guardian_news()

# Combine both DataFrames

combined_sentiments_df = pd.concat([guardian_sentiments_df, news_sentiments_df])

# Group by date and calculate the mean of sentiment scores

combined_sentiments_df = combined_sentiments_df.groupby('date').mean().reset_index()

```

```

# Save the combined DataFrame to a CSV file

combined_sentiments_df.to_csv('combined_vader_sentiment.csv', index=False)


# Display the combined DataFrame

print(combined_sentiments_df)


### Combine Sentiment Data and Prepare for Stock Analysis


# Load the combined sentiment data

combined_sentiments_df = pd.read_csv('combined_vader_sentiment.csv')


# Average the sentiment columns, handling NaNs

combined_sentiments_df['avg_title_sentiment'] =
combined_sentiments_df[['title_sentiment', 'mean_title_sentiment']].mean(axis=1,
skipna=True)

combined_sentiments_df['avg_description_sentiment'] =
combined_sentiments_df[['description_sentiment',
'mean_description_sentiment']].mean(axis=1, skipna=True)

combined_sentiments_df['avg_content_sentiment'] =
combined_sentiments_df[['content_sentiment', 'mean_content_sentiment']].mean(axis=1,
skipna=True)


# Drop the original sentiment columns

combined_sentiments_df.drop(columns=['title_sentiment', 'description_sentiment',
'content_sentiment', 'mean_title_sentiment', 'mean_description_sentiment',
'mean_content_sentiment'], inplace=True)


# Save the updated DataFrame to a CSV file

combined_sentiments_df.to_csv('updated_combined_vader_sentiment.csv', index=False)

```



```
# Display the updated DataFrame
print(combined_sentiments_df)

### Visualize Sentiment Data

# Load the updated combined sentiment data
combined_sentiments_df = pd.read_csv('updated_combined_vader_sentiment.csv')

# Create a figure and axis for the plot
plt.figure(figsize=(14, 7))

# Plot the KDE for average title sentiment scores without shading
sns.kdeplot(combined_sentiments_df['avg_title_sentiment'], color='green', label='Avg Title Sentiment')

# Plot the KDE for average description sentiment scores without shading
sns.kdeplot(combined_sentiments_df['avg_description_sentiment'], color='blue', label='Avg Description Sentiment')

# Plot the KDE for average content sentiment scores without shading
sns.kdeplot(combined_sentiments_df['avg_content_sentiment'], color='red', label='Avg Content Sentiment')

# Add titles and labels
plt.title('Distribution of Average Sentiment Scores')
plt.xlabel('Sentiment Score')
```

```
plt.ylabel('Density')
```

```
plt.legend()
```

```
# Show the plot
```

```
plt.show()
```

```
### Fetch and Combine Stock Data with Sentiments
```

```
def fetch_stock_data(ticker, start_date, end_date):
```

```
    """
```

```
    Fetch historical stock data using yFinance.
```

```
    Parameters:
```

```
        ticker (str): Stock ticker symbol.
```

```
        start_date (str): Start date in YYYY-MM-DD format.
```

```
        end_date (str): End date in YYYY-MM-DD format.
```

```
    Returns:
```

```
        pd.DataFrame: DataFrame containing the stock data.
```

```
    """
```

```
    stock = yf.Ticker(ticker)
```

```
    stock_data = stock.history(start=start_date, end=end_date)
```

```
    return stock_data
```

```
# Ensure 'date' column is in datetime format
```

```
combined_sentiments_df['date'] = pd.to_datetime(combined_sentiments_df['date'])
```

```

# Define the date range based on sentiment score data

start_date = combined_sentiments_df['date'].min().strftime('%Y-%m-%d')
end_date = combined_sentiments_df['date'].max().strftime('%Y-%m-%d')


# Fetch historical stock data for Shell

SHELL_TICKER = 'SHEL'

shell_stock_data = fetch_stock_data(SHELL_TICKER, start_date, end_date)


# Reset index to have 'Date' as a column

shell_stock_data.reset_index(inplace=True)


# Ensure the 'Date' column is in datetime format

shell_stock_data['Date'] = pd.to_datetime(shell_stock_data['Date'])


# Extract only the date part from the 'Date' column

shell_stock_data['Date'] = shell_stock_data['Date'].dt.date


# Select relevant columns

shell_stock_data = shell_stock_data[['Date', 'Open', 'High', 'Low', 'Close']]


# Rename the 'Date' column to 'date'

shell_stock_data.rename(columns={'Date': 'date'}, inplace=True)


# Ensure the 'date' column in combined_sentiments_df is in datetime format and then to
date

```

```

combined_sentiments_df['date'] =
pd.to_datetime(combined_sentiments_df['date']).dt.date

# Combine the sentiment score DataFrame with the stock data DataFrame

shell_combined_df = pd.merge(combined_sentiments_df, shell_stock_data, on='date',
how='inner')

# Save the combined DataFrame to a CSV file

shell_combined_df.to_csv('combined_vader_sentiment_stock_data_shell.csv', index=False)

# Display the combined DataFrame

print(shell_combined_df)

# Fetch historical stock data for BP

BP_TICKER = 'BP'

bp_stock_data = fetch_stock_data(BP_TICKER, start_date, end_date)

# Reset index to have 'Date' as a column

bp_stock_data.reset_index(inplace=True)

# Ensure the 'Date' column is in datetime format

bp_stock_data['Date'] = pd.to_datetime(bp_stock_data['Date'])

# Extract only the date part from the 'Date' column

bp_stock_data['Date'] = bp_stock_data['Date'].dt.date

# Select relevant columns

```

```

bp_stock_data = bp_stock_data[['Date', 'Open', 'High', 'Low', 'Close']]

# Rename the 'Date' column to 'date'

bp_stock_data.rename(columns={'Date': 'date'}, inplace=True)

# Combine the sentiment score DataFrame with the stock data DataFrame

bp_combined_df = pd.merge(combined_sentiments_df, bp_stock_data, on='date',
how='inner')

# Save the combined DataFrame to a CSV file

bp_combined_df.to_csv('combined_vader_sentiment_stock_data_bp.csv', index=False)

# Display the combined DataFrame

print(bp_combined_df)

### Visualize Stock Data with Moving Averages

def plot_candlestick_with_moving_averages(data, ticker):
    """
    Plot a candlestick chart with moving averages for a given stock.

    Parameters:
        data (pd.DataFrame): DataFrame containing stock data.
        ticker (str): Stock ticker symbol.
    """
    # Set the 'date' column as the index and ensure it's a DatetimeIndex
    data = data.set_index('date')

```

```

data.index = pd.to_datetime(data.index)

# Calculate moving averages
data['3_day_MA'] = data['Close'].rolling(window=3).mean()
data['6_day_MA'] = data['Close'].rolling(window=6).mean()
data['9_day_MA'] = data['Close'].rolling(window=9).mean()

# Prepare data for candlestick chart
stock_data = data[['Open', 'High', 'Low', 'Close']]

# Define the style with increased font size
my_style = mpf.make_mpf_style(base_mpf_style='charles', rc={'font.size': 16})

# Create the plot with custom moving average colors
apds = [
    mpf.make_addplot(data['3_day_MA'], color='orange', width=2, label='3-day MA'),
    mpf.make_addplot(data['6_day_MA'], color='purple', width=2, label='6-day MA'),
    mpf.make_addplot(data['9_day_MA'], color='blue', width=2, label='9-day MA')
]

fig, ax = mpf.plot(stock_data, type='candle', style=my_style,
                   title=f'{ticker} Stock Price Candlestick Chart', ylabel='Price',
                   addplot=apds, returnfig=True, figratio=(24, 10), figscale=1.5)

# Add legend
handles, labels = ax[0].get_legend_handles_labels()

```

```
ax[0].legend(handles=handles, labels=labels)
```

```
# Save the figure as an image file
```

```
plt.savefig(f'{ticker}_candlestick_chart.png')
```

```
plt.show()
```

```
# Plot for Shell
```

```
plot_candlestick_with_moving_averages(shell_combined_df, 'Shell')
```

```
# Plot for BP
```

```
plot_candlestick_with_moving_averages(bp_combined_df, 'BP')
```

```
### Create Lagged Features for Stock Data
```

```
def create_lagged_features(data, columns):
```

```
    """
```

```
    Create lagged features for the specified columns in the DataFrame.
```

```
    Parameters:
```

```
    data (pd.DataFrame): DataFrame containing stock and sentiment data.
```

```
    columns (list): List of column names for which to create lagged features.
```

```
    Returns:
```

```
    pd.DataFrame: DataFrame with lagged features.
```

```

"""

for col in columns:

    data[f'prev_{col}'] = data[col].shift(1)

return data.dropna()

# Columns to create lagged features for

lagged_columns = ['avg_title_sentiment', 'avg_description_sentiment',
'avg_content_sentiment', 'Open', 'High', 'Low', 'Close']

# Create lagged features for Shell data

shell_combined_df = create_lagged_features(shell_combined_df, lagged_columns)

# Create lagged features for BP data

bp_combined_df = create_lagged_features(bp_combined_df, lagged_columns)

### Correlation Heatmap

def plot_correlation_heatmaps(shell_combined_df, bp_combined_df,
columns_for_correlation):
    """

    Compute and plot correlation heatmaps for Shell and BP data.

    Parameters:

    shell_combined_df (pd.DataFrame): Shell combined data.

    bp_combined_df (pd.DataFrame): BP combined data.

```


columns_for_correlation (list): List of column names to include in the correlation analysis.

```
"""
```

```
# Compute the correlation matrix for Shell
```

```
shell_correlation_matrix = shell_combined_df[columns_for_correlation].corr()
```

```
# Compute the correlation matrix for BP
```

```
bp_correlation_matrix = bp_combined_df[columns_for_correlation].corr()
```

```
# Plot the correlation heatmap for Shell
```

```
plt.figure(figsize=(10, 8))
```

```
sns.heatmap(shell_correlation_matrix, annot=True, cmap='Spectral', linewidths=0.5)
```

```
plt.title('Correlation Heatmap for Shell Data')
```

```
plt.show()
```

```
# Plot the correlation heatmap for BP
```

```
plt.figure(figsize=(10, 8))
```

```
sns.heatmap(bp_correlation_matrix, annot=True, cmap='Spectral', linewidths=0.5)
```

```
plt.title('Correlation Heatmap for BP Data')
```

```
plt.show()
```

```
# Example usage:
```

```
columns_for_correlation = [
```

```
    'prev_avg_title_sentiment', 'prev_avg_description_sentiment',
```

```
    'prev_avg_content_sentiment',
```

```
    'Open', 'High', 'Low', 'Close', 'prev_Open', 'prev_Close'
```

```
]
```

```
# Call the function with the necessary dataframes
```

```
plot_correlation_heatmaps(shell_combined_df, bp_combined_df, columns_for_correlation)
```

Train and Evaluate Linear Regression Models

```
def train_and_evaluate(data, target_col, feature_cols):
```

```
    """
```

```
    Train and evaluate a linear regression model for the given target and features.
```

Parameters:

data (pd.DataFrame): DataFrame containing stock and sentiment data.

target_col (str): Target column name.

feature_cols (list): List of feature column names.

Returns:

tuple: True values, predicted values, MSE, and RMSE of the model.

```
    """
```

```
X = data[feature_cols]
```

```
y = data[target_col]
```

```
# Split data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,  
shuffle=False)
```

```
# Train a linear regression model
```

```

model = LinearRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)


# Calculate mean squared error and root mean squared error

mse = mean_squared_error(y_test, y_pred)

rmse = sqrt(mse)

return y_test, y_pred, mse, rmse


# Define target and feature variables for both models

model_specs = [

    {

        'target_col': 'Open',

        'feature_cols': ['prev_avg_title_sentiment', 'prev_avg_description_sentiment',
'prev_avg_content_sentiment', 'prev_Close', 'prev_Open']

    },

    {

        'target_col': 'Close',

        'feature_cols': ['prev_avg_title_sentiment', 'prev_avg_description_sentiment',
'prev_avg_content_sentiment', 'Open', 'prev_Close']

    }

]


# Train and evaluate models for Shell data

print("Shell Data:")

shell_results = []

for spec in model_specs:

```

```
y_test, y_pred, mse, rmse = train_and_evaluate(shell_combined_df, spec['target_col'],
spec['feature_cols'])
```

```
shell_results.append((y_test, y_pred, spec['target_col']))
```

```
print(f"Model for {spec['target_col']} - MSE: {mse:.4f}, RMSE: {rmse:.4f}")
```

```
# Train and evaluate models for BP data
```

```
print("\nBP Data:")
```

```
bp_results = []
```

```
for spec in model_specs:
```

```
    y_test, y_pred, mse, rmse = train_and_evaluate(bp_combined_df, spec['target_col'],
spec['feature_cols'])
```

```
    bp_results.append((y_test, y_pred, spec['target_col']))
```

```
    print(f"Model for {spec['target_col']} - MSE: {mse:.4f}, RMSE: {rmse:.4f}")
```

```
#### Actual vs Predicted Prices Plot for Linear Regression
```

```
def plot_actual_vs_predicted(y_test, y_pred, ax, title):
```

```
    """
```

```
    Plot actual vs predicted prices.
```

```
    Parameters:
```

```
        y_test (pd.Series): True target values.
```

```
        y_pred (np.array): Predicted values.
```

```
        ax (matplotlib.axes.Axes): Axes object to plot on.
```

```
        title (str): Title of the plot.
```

```
    """
```

```

ax.plot(y_test.index, y_test, label='Actual', color='blue')
ax.plot(y_test.index, y_pred, label='Predicted', color='red')
ax.set_title(title)
ax.set_xlabel('Index')
ax.set_ylabel('Price')
ax.legend()
ax.tick_params(axis='x', rotation=45)

```

Plotting the results for Shell data

```
fig, axes = plt.subplots(1, 2, figsize=(14, 6))
```

```
for idx, result in enumerate(shell_results):
```

```
    y_test, y_pred, target_col = result
```

```
    plot_actual_vs_predicted(y_test, y_pred, axes[idx], f'Shell Actual vs Predicted {target_col}
Price')
```

```
plt.tight_layout()
```

```
plt.show()
```

Plotting the results for BP data

```
fig, axes = plt.subplots(1, 2, figsize=(14, 6))
```

```
for idx, result in enumerate(bp_results):
```

```
    y_test, y_pred, target_col = result
```

```
    plot_actual_vs_predicted(y_test, y_pred, axes[idx], f'BP Actual vs Predicted {target_col}
Price')
```

```
plt.tight_layout()
```

```
plt.show()
```

Train and Evaluate Random Forest Models

```
def train_and_evaluate_rf(data, target_col, feature_cols):
```

```
    """
```

Train and evaluate a Random Forest model for the given target and features.

Parameters:

data (pd.DataFrame): DataFrame containing stock and sentiment data.

target_col (str): Target column name.

feature_cols (list): List of feature column names.

Returns:

tuple: True values, predicted values, MSE, and RMSE of the model.

```
    """
```

```
X = data[feature_cols]
```

```
y = data[target_col]
```

```
# Splitting the data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False,  
random_state=42)
```

```
# Train the Random Forest Regressor model
```

```
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
```

```
rf_model.fit(X_train, y_train)
```

```

# Make predictions
y_pred = rf_model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

return y_test, y_pred, mse, rmse

# Train and evaluate models for Shell data
print("Shell Data (Random Forest):")
shell_rf_results = []
for spec in model_specs:
    y_test, y_pred, mse, rmse = train_and_evaluate_rf(shell_combined_df, spec['target_col'],
spec['feature_cols'])
    shell_rf_results.append((y_test, y_pred, spec['target_col']))
    print(f"Model for {spec['target_col']} - MSE: {mse:.4f}, RMSE: {rmse:.4f}")

# Train and evaluate models for BP data
print("\nBP Data (Random Forest):")
bp_rf_results = []
for spec in model_specs:
    y_test, y_pred, mse, rmse = train_and_evaluate_rf(bp_combined_df, spec['target_col'],
spec['feature_cols'])
    bp_rf_results.append((y_test, y_pred, spec['target_col']))
    print(f"Model for {spec['target_col']} - MSE: {mse:.4f}, RMSE: {rmse:.4f}")

```

Actual vs Predicted Prices Plot for Random Forest

```
def plot_actual_vs_predicted(y_test, y_pred, ax, title):
```

```
    """
```

Plot actual vs predicted prices.

Parameters:

`y_test` (pd.Series): True target values.

`y_pred` (np.array): Predicted values.

`ax` (matplotlib.axes.Axes): Axes object to plot on.

`title` (str): Title of the plot.

```
    """
```

```
    ax.plot(y_test.index, y_test, label='Actual', color='blue')
```

```
    ax.plot(y_test.index, y_pred, label='Predicted', color='red')
```

```
    ax.set_title(title)
```

```
    ax.set_xlabel('Index')
```

```
    ax.set_ylabel('Price')
```

```
    ax.legend()
```

```
    ax.tick_params(axis='x', rotation=45)
```

```
# Plotting the results for Shell data
```

```
fig, axes = plt.subplots(1, 2, figsize=(14, 6))
```

```
for idx, result in enumerate(shell_rf_results):
```

```
    y_test, y_pred, target_col = result
```

```
    plot_actual_vs_predicted(y_test, y_pred, axes[idx], f'Shell Actual vs Predicted {target_col}  
    Price (RF)')
```



```
plt.tight_layout()
```

```
plt.show()
```

```
# Plotting the results for BP data
```

```
fig, axes = plt.subplots(1, 2, figsize=(14, 6))
```

```
for idx, result in enumerate(bp_rf_results):
```

```
    y_test, y_pred, target_col = result
```

```
    plot_actual_vs_predicted(y_test, y_pred, axes[idx], f'BP Actual vs Predicted {target_col}  
    Price (RF)')
```

```
plt.tight_layout()
```

```
plt.show()
```

Learning Curves for Linear Regression and Random Forest Models

```
def plot_learning_curve(estimator, X, y, ax, title, ylim=None, cv=None, n_jobs=None,  
train_sizes=np.linspace(0.1, 1.0, 5)):
```

```
    """
```

```
    Plot learning curve for a given model.
```

Parameters:

estimator (object): The model to train.

X (pd.DataFrame): Features DataFrame.

y (pd.Series): Target Series.

ax (matplotlib.axes.Axes): Axes object to plot on.

```

title (str): Title of the plot.

ylim (tuple): Limits for the y-axis.

cv (int): Number of cross-validation folds.

n_jobs (int): Number of jobs to run in parallel.

train_sizes (np.array): Array of training set sizes to use.
"""

if ylim is not None:

    ax.set_ylim(*ylim)

ax.set_xlabel("Training examples")

ax.set_ylabel("Score")


train_sizes, train_scores, test_scores = learning_curve(estimator, X, y, cv=cv,
n_jobs=n_jobs, train_sizes=train_sizes, scoring='neg_mean_squared_error')

train_scores_mean = -np.mean(train_scores, axis=1)

train_scores_std = np.std(train_scores, axis=1)

test_scores_mean = -np.mean(test_scores, axis=1)

test_scores_std = np.std(test_scores, axis=1)


sns.set_style("whitegrid")

sns.despine()


ax.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean +
train_scores_std, alpha=0.1, color="r")

ax.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean +
test_scores_std, alpha=0.1, color="g")

```

```

ax.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
ax.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation score")

ax.legend(loc="best")

ax.set_title(title)

# Display MSE values on the plot

ax.text(0.05, 0.95, f"Training MSE: {train_scores_mean[-1]:.4f}\nValidation MSE:
{test_scores_mean[-1]:.4f}",

        transform=ax.transAxes, verticalalignment='top',
        bbox=dict(boxstyle="round,pad=0.3", edgecolor="black", facecolor="white"))

# Define the models

lr = LinearRegression()

rf = RandomForestRegressor(n_estimators=100, random_state=42)

# Define the feature and target columns for both models

feature_cols_open = ['prev_avg_title_sentiment', 'prev_avg_description_sentiment',
'prev_avg_content_sentiment', 'prev_Close', 'prev_Open']

feature_cols_close = ['prev_avg_title_sentiment', 'prev_avg_description_sentiment',
'prev_avg_content_sentiment', 'Open', 'High', 'Low', 'prev_Close']

# Plot learning curves for Shell data

fig, axes = plt.subplots(2, 2, figsize=(14, 12))

X_shell_open = shell_combined_df[feature_cols_open]

y_shell_open = shell_combined_df['Open']

```

```

X_shell_close = shell_combined_df[feature_cols_close]

y_shell_close = shell_combined_df['Close']


plot_learning_curve(lr, X_shell_open, y_shell_open, axes[0, 0], "Shell Linear Regression
(Open)", cv=5, n_jobs=-1)

plot_learning_curve(rf, X_shell_open, y_shell_open, axes[0, 1], "Shell Random Forest
(Open)", cv=5, n_jobs=-1)


plot_learning_curve(lr, X_shell_close, y_shell_close, axes[1, 0], "Shell Linear Regression
(Close)", cv=5, n_jobs=-1)

plot_learning_curve(rf, X_shell_close, y_shell_close, axes[1, 1], "Shell Random Forest
(Close)", cv=5, n_jobs=-1)


plt.tight_layout()

plt.show()


# Plot learning curves for BP data

fig, axes = plt.subplots(2, 2, figsize=(14, 12))


X_bp_open = bp_combined_df[feature_cols_open]

y_bp_open = bp_combined_df['Open']


X_bp_close = bp_combined_df[feature_cols_close]

y_bp_close = bp_combined_df['Close']


plot_learning_curve(lr, X_bp_open, y_bp_open, axes[0, 0], "BP Linear Regression (Open)",
cv=5, n_jobs=-1)

```

```
plot_learning_curve(rf, X_bp_open, y_bp_open, axes[0, 1], "BP Random Forest (Open)",  
cv=5, n_jobs=-1)
```

```
plot_learning_curve(lr, X_bp_close, y_bp_close, axes[1, 0], "BP Linear Regression (Close)",  
cv=5, n_jobs=-1)
```

```
plot_learning_curve(rf, X_bp_close, y_bp_close, axes[1, 1], "BP Random Forest (Close)",  
cv=5, n_jobs=-1)
```

```
plt.tight_layout()
```

```
plt.show()
```