UNIVERSITY OF WESTMINSTER

Microsoft Corporation

Predicting Stock Price Movements: A Machine Learning Approach Applied to Microsoft Corporation Data

WORD COUNT: 2198, EXCLUDING COVER PAGE, TABLE OF CONTENTS, ABSTRACT, FORMULAS, TABLES, APPENDIX, CODES, AND THE OUTPUTS OF CODES.

Artificial Intelligence and Machine Learning in Finance Services (7FNCE043W)

Module Leader: Dr. Yumei Yao

Versha Sandesh
Student ID: 20395827
MSc Fintech with Business Analytics

Table of Contents

Abstract 2
1. Introduction 3
2. Retrieving Data of Microsoft Corporation Stock (MSFT) 3
Data Loading and Preprocessing 3
3. Features Creation 4
4. Exploratory Data Analysis (EDA) 5
Descriptive Statistics 5
Data Visualisation 6
Correlation Analysis 8
5. Machine Learning Classification Methods 9
Logistic Regression 10
Extra Trees Classifier 10
6. Cross-Validation Analysis 10
Cross Validation of Logistic Regression 10
Cross Validation of Extra Trees Classifier 11
7. Evaluation of Classification Methods 11
8. Predicting the MSFT Price using the Logistic Regression Model 12
Classification Report 12
Confusion Matrix 13
ROC Plot 13
9. Market Returns and Strategy Returns 14
10. Cumulative Market and Cumulative Strategy Returns 15
11. Interpretation and Discussion 15
References 17
Appendix: Coding 19

Abstract

This study explores the application of machine learning (ML) techniques to predict stock price movements using historical data of Microsoft Corporation (MSFT) from January 2014 to December 2023. Utilising logistic regression, the analysis demonstrates the model's ability to predict MSFT stock price rises with an accuracy rate of approximately 51%. The study aims to address the hypothesis regarding the predictive capabilities of machine learning in stock price movements. Results indicate a moderate level of accuracy in predicting MSFT stock prices using ML technology. While ML provides valuable insights into stock price dynamics, its limitations must be acknowledged, such as the inability to forecast future events and market changes. The findings emphasise the necessity of human judgment, market expertise, and risk management in investment decision-making processes.

1. Introduction

Microsoft Corporation, founded in 1975 by Bill Gates and Paul Allen, is a leading American multinational technology company renowned for its software, hardware, and cloud computing services (Zachary, et al., 2024).

With flagship products like Windows operating systems, Office productivity suite, and Azure cloud platform, Microsoft drives digital transformation across various industries with the mission to empower individuals and organisations worldwide (Microsoft, n.d.).

Background Information of Microsoft Corporation	
Head Office	Microsoft Corporation One Microsoft Way Redmond Washington USA
Industry	Information Technology
Web Address	www.microsoft.com
Financial Year End	June
NASDAQ Ticker	MSFT

Microsoft has demonstrated impressive stock and financial performance in recent years, with reported revenues of US$211,915 million for FY2023, marking a 6.9% increase over FY2022 (MarketLine, 2023). As a key player in the technology sector, Microsoft attracts investors seeking stable returns and growth opportunities, supported by its solid balance sheet and commitment to long-term growth.

This report proposes leveraging machine learning models and techniques to predict stock trends in Microsoft's data, offering insights to investors for informed decision-making regarding their investments in Microsoft stock, thereby optimising returns and managing risks effectively. The analysis aims to evaluate the effectiveness of machine learning in accurately predicting fluctuations in Microsoft's stock price.

2. Retrieving Data of Microsoft Corporation Stock (MSFT)

Data Loading and Preprocessing

Importing Data

Financial data for Microsoft Corporation stock analysis is sourced from Yahoo Finance using the 'yahooquery' Python package within a Python Jupyter Notebook. The ticker symbol 'MSFT' retrieves comprehensive data, including stock prices, historical records, company details, and financial statements (Python Package Index (PyPI), n.d.).

The dataset focuses on daily MSFT stock prices spanning from 01 January 2014 to 31 December 2023. This dataset comprises 2516 rows and 7 columns, containing key metrics such as open, high, low, close, volume, adjclose, and dividends (Yahoo Finance, 2024).

Preparing Data

Following data loading, preprocessing steps are initiated to enhance data quality, which may involve tasks like noise removal or complexity reduction (Yao, 2024).

For the MSFT dataset, initial preprocessing involves dropping columns 'adjclose' and 'dividends' to streamline the dataset, retaining only volume, open, high, low, and close columns. Subsequently, missing values are checked to ensure data completeness and integrity.

3. Features Creation

To analyse and predict MSFT stock market movements, the following features have been created:

Daily Price Fluctuations

H-L (High – Low): Indicates the daily high to low price range, reflecting intraday volatility.

C-O (Close-Open): Represents the difference between closing and opening prices, reflecting if the stock closed higher or lower than its opening price.

Rolling Moving Averages

$$\text{n-day Moving Average} = \frac{Sum\ of\ Closing\ Prices\ for\ the\ last\ n\ Days}{n}$$

3day MA (3-day Moving Average): Identifies short-term trends using a 3-day window.

10day MA (10-day Moving Average): Provides a longer-term trend perspective with a 10-day window.

30day MA (30-day Moving Average): Offers a broader view of stock price trend over 30 days.

Standard Deviation

Measures the dispersion of closing prices around the mean over a 5-day window, indicating volatility.

$$\text{Standard Deviation} = \sqrt{\frac{\sum_{i=1}^{t}(x_i - \bar{x})^2}{t}}$$

Where:
x_i is each closing price at time i
$\bar{x}$ is the mean closing price up to time t

Price_Rise

A binary indicator (1 for price rise, 0 otherwise) determines if the stock price rises or falls compared to the next day.

$$(\text{Price\_Rise})_t = \begin{cases} 1 \rightarrow & if\ Close_{t+1} > Close_t \\ 0 \rightarrow & otherwise \end{cases}$$

Where:
$Close_{t+1}$ is the closing price at next time step
$Close_t$ is the closing price at the current time step

These features collectively provide insights into MSFT stock price movements, volatility, and trends across various time frames. They serve as valuable inputs for machine learning models or statistical analysis to predict, identify patterns, or gain insights into market behaviour (Raval, 2024).

4. Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is an essential initial step in understanding dataset structure, patterns, and relationships, aiding in effective data manipulation for insights (IBM, n.d.).

Through descriptive statistics, data visualisation, and correlation analysis, EDA examines key features in the Microsoft stock dataset, including daily price differences, moving averages, standard deviations, and price movement indicators.

Descriptive Statistics

The descriptive statistic on MSFT for EDA is given below, which shows key information about data, such as minimum and maximum value for price, total count, and data type. It also verifies that there are no missing values in the dataset.

	volume	open	high	low	close	H-L	C-O	3day MA	10day MA	30day MA	Std_dev	Pric
count	2.486000e+03	2486.000000	2486.000000	2486.000000	2486.000000	2486.000000	2486.000000	2486.000000	2486.000000	2486.000000	2486.000000	2486.
mean	2.999795e+07	151.169373	152.676227	149.620177	151.224554	3.056050	0.055181	150.952656	150.478178	149.123866	2.112427	0.
std	1.382120e+07	101.449258	102.503234	100.357030	101.480702	2.924295	2.562814	101.310759	101.008163	100.144333	2.155757	0.
min	7.425600e+06	37.220001	37.740002	37.189999	37.419998	0.240002	-15.670013	37.416667	36.806000	36.460000	0.050301	0.
25%	2.161240e+07	56.602499	56.952501	56.222499	56.692500	0.850002	-0.650002	56.613333	56.427500	54.957083	0.529041	0.
50%	2.678650e+07	111.415001	112.195000	110.389999	111.704998	1.849998	0.065002	110.970000	110.306499	109.459833	1.300255	1.
75%	3.407525e+07	243.834999	245.877499	241.410000	244.327496	4.509979	0.790001	243.892497	243.327501	242.100499	3.017633	1.
max	2.025224e+08	383.760010	384.299988	378.160004	382.700012	24.609985	22.079987	380.153341	377.088004	373.869002	14.000517	1.

Extracting Data Information

```
<class 'pandas.core.frame.DataFrame'>
Index: 2486 entries, 30 to 2515
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   date        2486 non-null   object
 1   volume      2486 non-null   int64
 2   open        2486 non-null   float64
 3   high        2486 non-null   float64
 4   low         2486 non-null   float64
 5   close       2486 non-null   float64
 6   H-L         2486 non-null   float64
 7   C-O         2486 non-null   float64
 8   3day MA     2486 non-null   float64
 9   10day MA    2486 non-null   float64
 10  30day MA    2486 non-null   float64
 11  Std_dev     2486 non-null   float64
 12  Price_Rise  2486 non-null   int32
dtypes: float64(10), int32(1), int64(1), object(1)
memory usage: 262.2+ KB
```

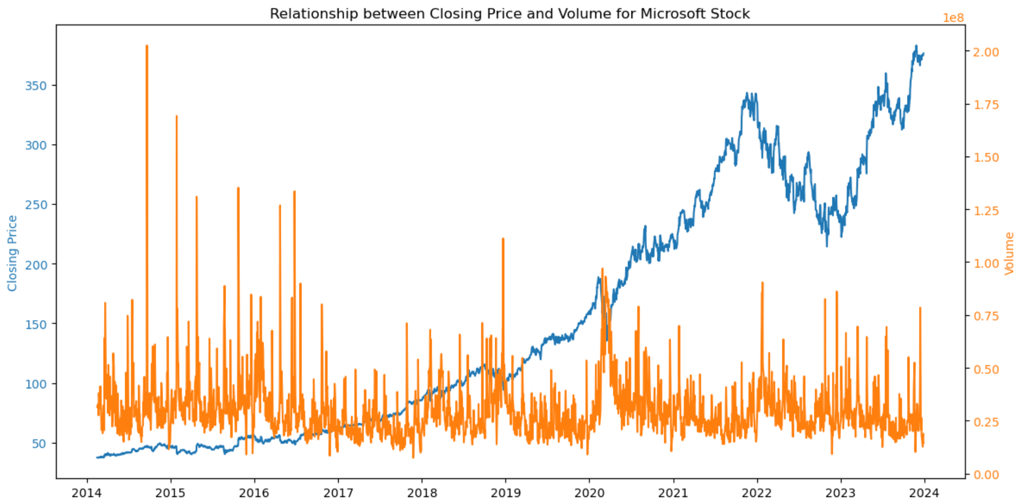
Checked for Missing Values

```
date          0
volume        0
open          0
high          0
low           0
close         0
H-L           0
C-O           0
3day MA       0
10day MA      0
30day MA      0
Std_dev       0
Price_Rise    0
dtype: int64
```

Data Visualisation

Figure 1 illustrates an inverse relationship between the Closing Price and Volume of Microsoft stock, suggesting that as the closing price decreases, the trading volume tends to increase and vice versa. Additionally, the graph depicts a general upward trend in the closing price of MSFT over the observed period, accompanied by significant trading volume, indicating robust market demand for Microsoft stock.

Figure 1



Subsequently, the moving averages have been plotted to observe the trend in MSFT prices over the years. As the rolling window increases, the trend line gets smoother. However, the variation remains similar. It can be noticed in Figure 2 that the average price of MSFT stock has surpassed the value of $350 by the end of the year 2023 and is still showing an upward trend.

Figure 2

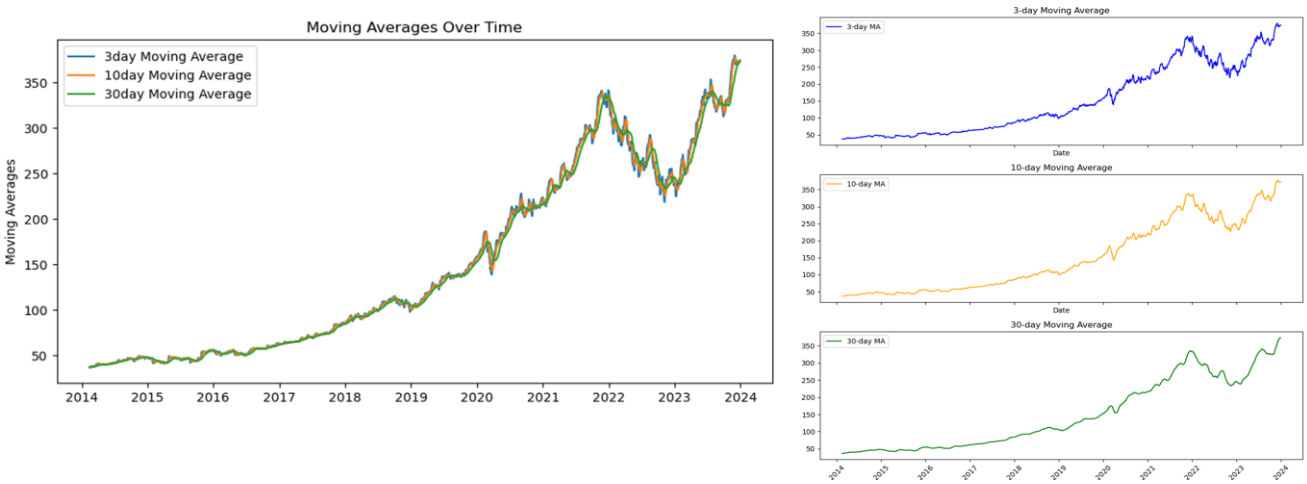
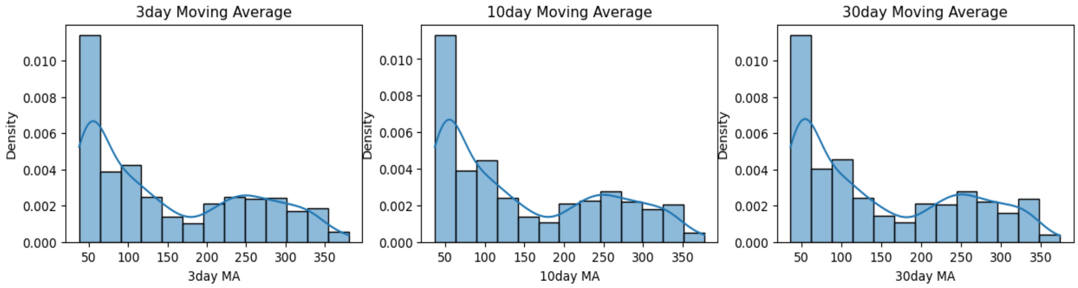


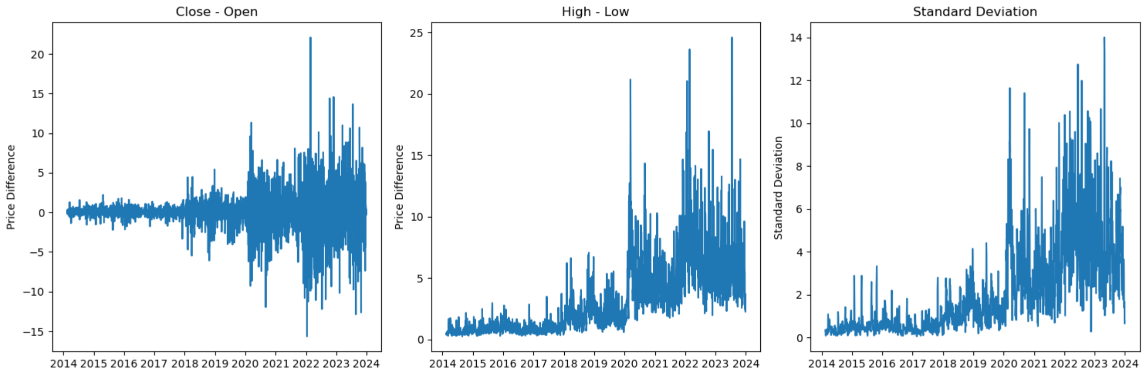
Figure 3 displays multiple histogram plots side by side, allowing for easy comparison of the distributions of different moving average features with their density.

Figure 3



In Figure 4, the features C-O and H-L illustrate that the price differences in MSFT have been consistently increasing over the last decade, accompanied by a higher standard deviation in recent years. This observation suggests that as the fluctuations in price have intensified, the standard deviation has also risen, indicating a higher risk associated with MSFT stock.

Figure 4



In Figure 5, the cluster plot reveals as the price difference approaches a minimum value (near 0), the trade volume tends to be higher. Figure 6 reinforces this observation by demonstrating that as the standard deviation decreases (lower volatility), there is an upward trend in trade volume, indicating increased trading activity during periods of stability.

Figure 5

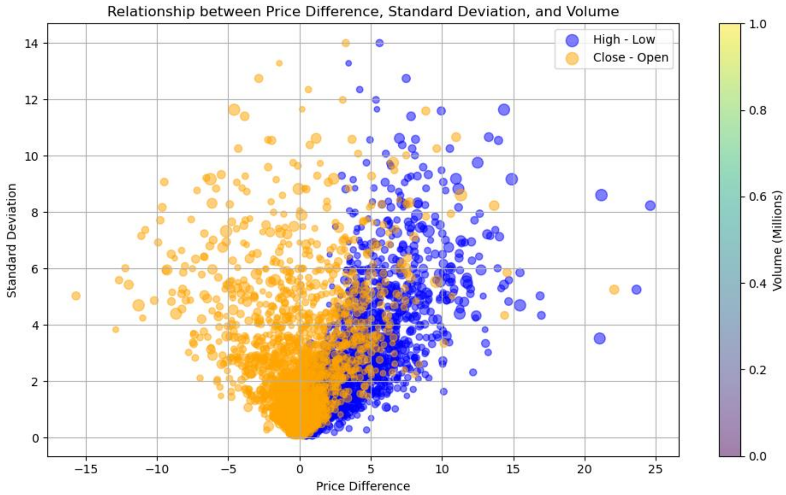


Figure 6

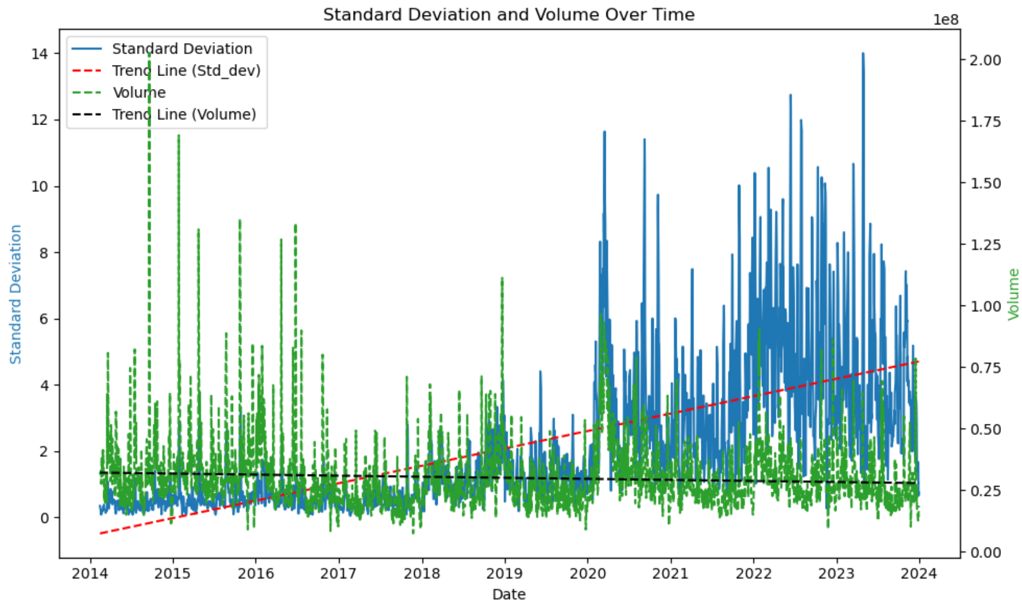
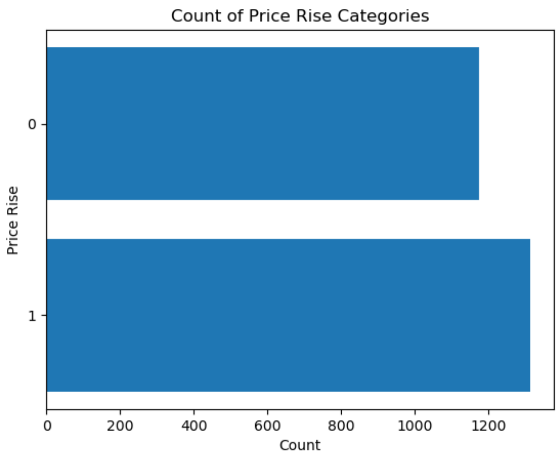


Figure 7 shows the count of price rise (Price_Rise = 1) instances surpasses price decline (Price_Rise = 0), highlighting an overall upward trend in MSFT stock.

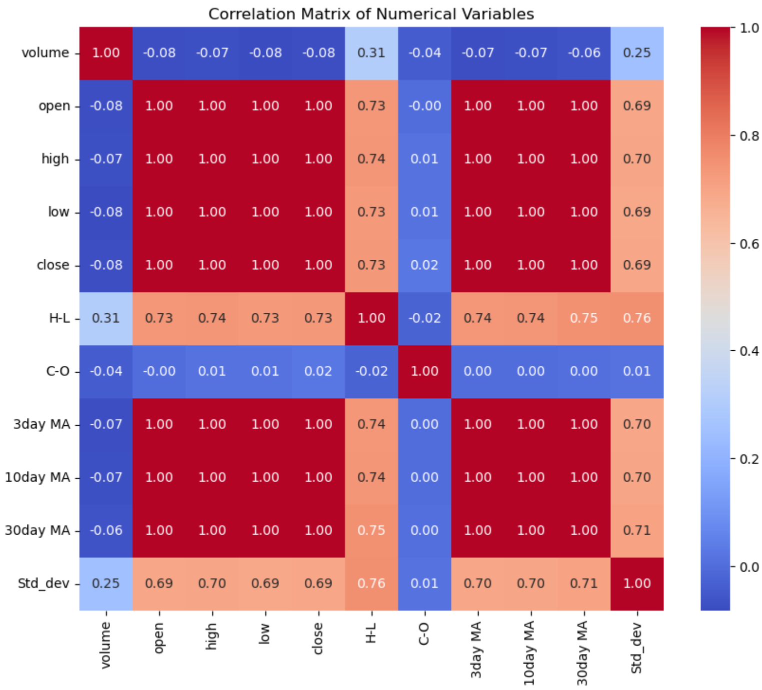
Figure 7



Correlation Analysis

The correlation matrix for the created features of the MSFT stock indicates that trading volume is moderately correlated with the feature representing the difference between high and low prices (31%) and the standard deviation (25%). Although the correlation is not very high, it suggests that changes in price differences and volatility have some impact on the trading volume of MSFT. Additionally, the correlation matrix reveals intercorrelations among all features, indicating complex relationships within the dataset.

Figure 8



These findings underscore the complex relationships within the dataset and provide valuable insights for understanding the dynamics of MSFT stock price movements.

5. Machine Learning Classification Methods

In machine learning classification, models are trained to learn patterns from training data and evaluated using separate test data before making predictions on new, unseen data (Keita, 2022). In this report, two machine learning models, Logistic Regression and Extra Trees Classifier, were evaluated for their effectiveness in predicting future price movements of Microsoft stock.

The classification task undertaken here is Binary Classification, which involves predicting one of two classes (Brownlee, 2020). For this purpose, the binary feature "Price_Rise" was created specifically to forecast price increases in Microsoft stock and was set as the target variable (Y) for the models, while the remaining created features, as outlined in Section 3, were assigned to the independent variable (X).

Preprocessing of the MSFT data involved splitting the data into training (80%) and test (20%) sets, followed by standardisation using StandardScaler to ensure consistent scaling across features (Yao, 2024).

Logistic Regression

Logistic regression is a supervised machine learning algorithm utilised primarily for binary classification tasks. It estimates the probability of an outcome, event, or observation in one of two possible categories. The model produces a binary outcome, commonly represented as 0 (indicating no price rise) or 1 (suggesting a price rise) (Kanade, 2022). The result of the Logistic Regression model is presented in Sections 6 and 7.

Extra Trees Classifier

The Extra Trees Classifier is an ensemble machine-learning approach for predictive tasks, including stock price prediction. It functions by training numerous randomised decision trees, referred to as extra-trees, on different subsets of the dataset and then averaging their predictions to generate a final output (scikit-learn, n.d.). This algorithm creates a diverse array of trees by randomly selecting subsets of features and thresholds for node splitting, ensuring they are uncorrelated and varied. This diversity enhances the model's capability to capture intricate patterns within the dataset, thereby improving its predictive accuracy (ArcGIS Pro, n.d.). The outcome is presented in the next section.

6. Cross-Validation Analysis

Cross-validation (CV) is a crucial technique for evaluating machine learning models and assessing their accuracy and generalisation capabilities. It involves dividing the dataset into training and testing subsets, repeatedly training the model on different subsets, and evaluating its performance. By averaging performance metrics across iterations, CV offers a reliable estimate of the model's performance and helps to mitigate issues such as overfitting or underfitting (Joby, 2021). The cross_val_score to perform k-fold cross-validation with 5 folds (cv=5) is used for both models.

$$\text{Mean Accuracy} = \frac{Number\ of\ Correct\ Classifications}{Total\ Number\ of\ Test\ Cases}$$

$$\text{Standard Deviation} = \sqrt{\frac{1}{k}\sum_{i=1}^{k}(Accuracy_i - Mean\ Accuracy)^2}$$

Cross Validation of Logistic Regression

The logistic regression model achieved accuracy scores ranging from 0.5251 to 0.5340 across five different dataset subsets. The Mean Accuracy is approximately 0.53 (53%), with a Standard Deviation of around 0.0045 (0.45%).

```
Cross-Validation Scores for Logistic Regression:
[0.52512563 0.53517588 0.52512563 0.52644836 0.53400504]
Mean Accuracy based on Cross-Validation Score: 0.53
Standard Deviation of Cross-Validation Scores: 0.0045
```

Cross Validation of Extra Trees Classifier

For the Extra Trees Classifier, accuracy scores ranged from 0.4095 to 0.5214 across the five subsets. The Mean Accuracy is approximately 0.46 (46%), with a Standard Deviation of about 0.0356 (3.56%).

```
Cross-Validation Scores for ExtraTreesClassifier:
[0.46733668 0.40954774 0.45728643 0.52141058 0.45843829]
Mean Accuracy based on Cross-Validation Score: 0.46
Standard Deviation of Cross-Validation Scores: 0.0356
```

7. Evaluation of Classification Methods

A classification report for each model is generated to evaluate both models.

Classification Report: Logistic Regression

	precision	recall	f1-score	support
0	0.51	0.34	0.41	246
1	0.51	0.68	0.59	252
accuracy			0.51	498
macro avg	0.51	0.51	0.50	498
weighted avg	0.51	0.51	0.50	498

Classification Report: Extra Trees Classifier

	precision	recall	f1-score	support
0	0.49	0.42	0.45	246
1	0.50	0.57	0.53	252
accuracy			0.50	498
macro avg	0.50	0.50	0.49	498
weighted avg	0.50	0.50	0.49	498

The logistic regression model achieved an accuracy of 51%, indicating the overall proportion of correctly identified instances in the model.

- For class 0 (no price rise in MSFT), the precision is 51%, meaning that out of all instances predicted as no price rise, 51% were correctly predicted, and the recall indicates that only 34% of all actual instances of no price rise were correctly identified as such by the model.
- For class 1 (price rise in MSFT), the precision indicates that 51% of the instances predicted as price rise were correct, and the recall shows that 68% of all actual instances of price rise were correctly identified.

The Extra Trees Classifier Model achieved an overall Accuracy of 50% for indicating the proportion of correctly identified instances.

- For class 0, the precision is 49% and recall is 42%.
- For class 1, the precision is 50%, and recall is 57%.

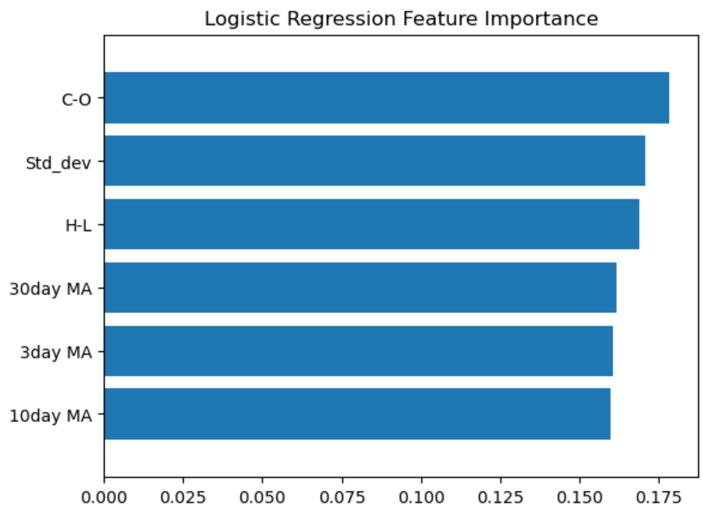
Based on cross-validation (Section 6), where logistic regression has a higher mean accuracy (0.53) compared to the ExtraTreesClassifier (0.46), suggesting that logistic regression performs better on average in predicting the target variable. Logistic regression also has a much lower standard deviation (0.0045) compared to the ExtraTreesClassifier (0.0356), indicating that logistic regression's performance is more consistent across folds compared to the ExtraTreesClassifier.

The Logistic Regression model demonstrates slightly higher accuracy and better performance compared to the Extra Trees Classifier model. Therefore, logistic regression is recommended for predicting price movements in MSFT stock.

8. Predicting the MSFT Price using the Logistic Regression Model

The Logistic Regression model, chosen based on its accuracy score, utilises X_test-scaled data for predicting Microsoft stock price movements. Figure 9 showcases feature rankings for the Logistic Regression Model, with opening and closing price differences identified as most important, followed by standard deviation, highlighting the significance of these features in influencing the model's predictions.

Figure 9



Classification Report

The classification report is a performance evaluation metric in machine learning which is used to show the precision, recall, F1-score, and support score of the trained classification model, as well as the weighted average of these metrics across all classes. The F1-score is a weighted mean of precision and recall, with values closer to 1.0 indicating better performance (Kharwal, 2021).

Classification Report: Logistic Regression

	precision	recall	f1-score	support
0	0.51	0.34	0.41	246
1	0.51	0.68	0.59	252
accuracy			0.51	498
macro avg	0.51	0.51	0.50	498
weighted avg	0.51	0.51	0.50	498

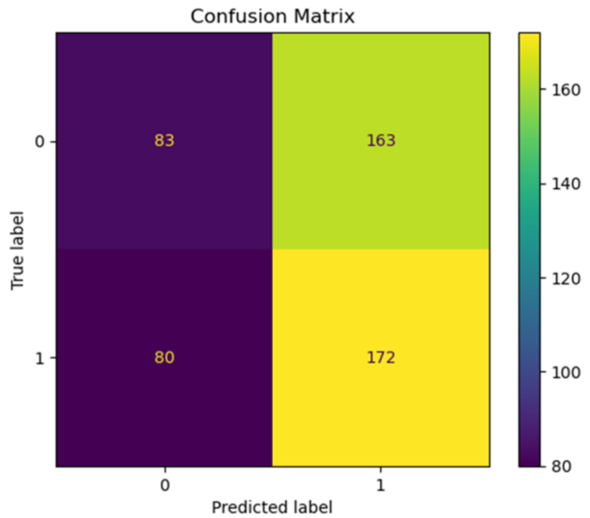
As mentioned in Section 7, the Logistic regression model classification report has an accuracy rate of 51%. F1-scores indicate 41% for class 0 (no price rise) and 59% for class 1 (price rise), reflecting moderate performance in predicting price movements, with a slightly higher performance in predicting price rises compared to no price rises.

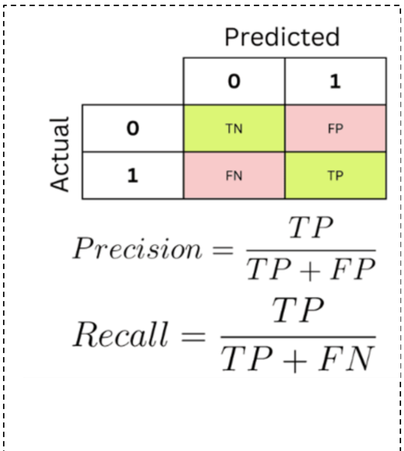
Confusion Matrix

A confusion matrix is a table used to assess the performance of a machine learning model's performance by comparing its predicted and actual outcomes. It provides a breakdown of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for each class in a classification problem. The matrix gives a quick snapshot for understanding where the model is making errors and can help fine-tune the model for better performance (LinkedIn, n.d.).

The confusion matrix for the logistic regression model on the prediction of MSFT stock, as depicted in Figure 10, reveals the model's performance in classifying instances into different classes. According to the matrix, the model accurately classifies 83 instances of 'no-rise/decline in price (0)' and 172 instances of 'rise in price (1)'. However, 163 instances of no-rise/decline are incorrectly labelled as a price rise, and 80 instances of actual price rises are misclassified.

Figure 10





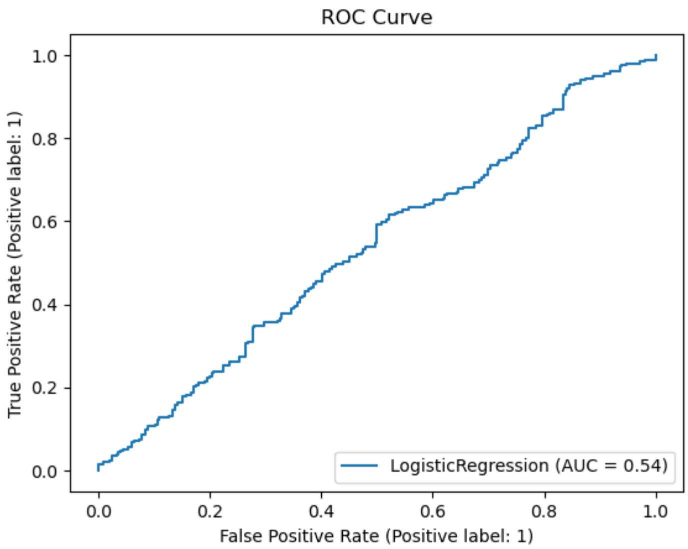
Source: (Selvaraj, 2022)

ROC Plot

The Receive Operating Characteristics (ROC) Curve is the plot of the true positive rate (TPR) against the false positive rate (FPR) (Google for Developers, 2022). ROC curves typically feature TPR on the Y axis and FPR on the X axis. A larger area under the curve (AUC) is usually better, as it indicates a higher true positive rate and a lower false positive rate across different thresholds (scikit-learn, n.d.).

The ROC curve (Figure 11) illustrates an Area Under the Curve (AUC) of 54%. While surpassing random guessing (AUC > 0.5), it suggests a modest level of predictive capability for the logistic regression model on the MSFT dataset.

Figure 11



9. Market Returns and Strategy Returns

A trading strategy is implemented based on predictions made by a logistic regression model on the dataset. First, the original dataset is copied to trade_dataset, and a new column, 'Y_pred', is created to store the values of predictions made by the logistic regression model. To ensure consistency, rows containing NaN values (corresponding to the initial rows without predictions) are dropped from the trade_dataset (Yao, 2024). Subsequently, with predicted stock values available, strategy returns can be computed.

Market Returns

For calculating market returns, a new column named "Tomorrows Returns" is created to calculate the returns for the next trading day. These returns are computed as the natural logarithm of the ratio between the closing price of the next day (close.shift(-1)) and the closing price of the current day (close), representing the percentage change in price from one day to the next. The values are then shifted upwards by one element to align tomorrow's returns with today's prices.

$$Market\ Returns = \ln\left(\frac{Close_{next\ day}}{Close_{current\ day}}\right)$$

Strategy Returns

Another new column, "Strategy Returns", is created to capture the returns the trading strategy generates. The strategy is based on the predictions (y_pred_logistic) made by the logistic regression model. If the model predicts a positive outcome (1), the strategy buys the asset, resulting in positive value equivalent to tomorrow's returns. Conversely, if the model predicts a negative outcome (0), the strategy sells or shorts the asset, leading to negative value equivalent to tomorrow's returns. By using the np.where() function, the values stored in the 'Tomorrow's Returns' column are then assigned to the 'Strategy Returns' column if the 'Y_pred' column stores 'True' (a long position); otherwise, the negative of the values is recorded (a short position).

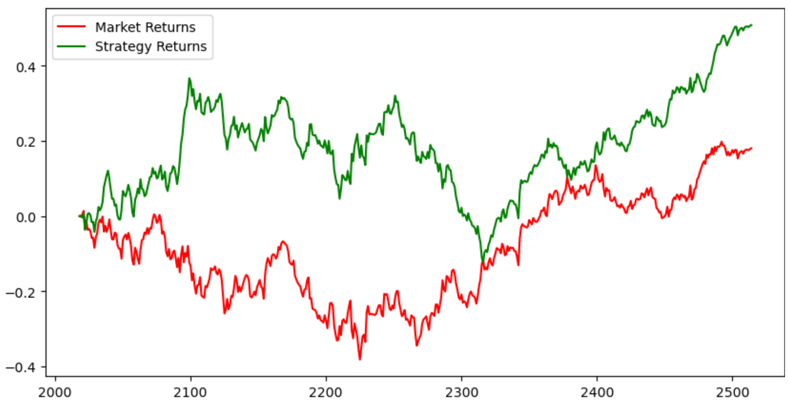
10. Cumulative Market and Cumulative Strategy Returns

Following the daily market and strategy returns calculation, cumulative returns are computed to provide a comprehensive perspective on the total accumulated returns over time. Cumulative returns are particularly valuable for assessing investment or trading strategies' overall performance and profitability.

To facilitate this analysis, two new columns for cumulative market and strategy returns were generated using the cumsum() function and added to the 'trade_dataset'. These columns capture the aggregation of returns over the entire dataset period.

Additionally, a plot of the cumulative returns trend is depicted in Figure 12. The trend line of cumulative strategy returns consistently surpasses cumulative market returns, signifying the strategy's outperformance over the specified period. This indicates that the strategy to time the long or short position has generated higher returns than simply holding the asset or investing in the market index.

Figure 12



11. Interpretation and Discussion

The analysis of Microsoft Corporation's historical stock data demonstrates the application of machine learning (ML) techniques in predicting stock price movements. While ML can predict MSFT stock price rises to some extent, it's crucial to acknowledge the inherent limitations of such predictions, as no model can achieve 100% accuracy (Choudhary, 2023).

Interpretation

This report utilised machine learning techniques, particularly logistic regression, to predict price movements in Microsoft (MSFT) stock data. Through exploratory data analysis, feature engineering, and model training, we could generate predictions about whether the stock price would rise or fall on the next trading day.

The logistic regression model achieved an accuracy rate of approximately 51%, indicating a slightly better predictive capability than random chance. Further analysis of precision, recall, and F1-score provided insights into the model's proficiency in identifying positive and negative price movements.

Additionally, cumulative returns analysis demonstrated the trading strategy's performance based on the model's predictions, indicating potential for outperforming the market.

Discussion

While machine learning technology can predict stock prices based on current market conditions, it cannot foresee future events or account for changes in market dynamics. According to the efficient market hypothesis, achieving 100% accuracy in predicting stock market movements is nearly impossible (Khan, et al., 2023). Additionally, k-fold cross-validation evaluation may introduce "Look-Ahead Bias," failing to capture changes in market conditions that could impact predictions (Milosevic, n.d.).

The random walk theory suggests that stock prices move unpredictably, challenging the idea that traders can time the market or profit from patterns in stock prices (Smith, 2023).

Conclusion

While machine learning can provide valuable insights and predictions regarding stock price movements, it should be viewed as a complementary tool rather than a definitive solution. Human judgment, market knowledge, and risk management are crucial in investment decision-making processes. Machine learning solutions provide data-driven guidelines that enable investors to make informed decisions; however, continuous monitoring and refinement of machine learning models are necessary to adapt to changing market conditions and ensure their predictive accuracy over time.

References

ArcGIS Pro, n.d. *How Extra trees classification and regression algorithm works.* [Online]
Available at: https://pro.arcgis.com/en/pro-app/latest/tool-reference/geoai/how-extra-tree-classification-and-regression-works.htm
[Accessed 31 March 2024].

Brownlee, J., 2020. *4 Types of Classification Tasks in Machine Learning.* [Online]
Available at: https://machinelearningmastery.com/types-of-classification-in-machine-learning/
[Accessed 31 March 2024].

Choudhary, P., 2023. *How Machine Learning Facilitates In Forecasting Stock Prices.* [Online]
Available at: https://globalfintechseries.com/machine-learning/how-machine-learning-facilitates-in-forecasting-stock-prices/
[Accessed 01 April 2024].

Google for Developers, 2022. *Classification: ROC Curve and AUC.* [Online]
Available at: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc#:~:text=An%20ROC%20curve%20(receiver%20operating,False%20Positive%20Rate
[Accessed 01 April 2024].

IBM, n.d. *What is exploratory data analysis (EDA)?.* [Online]
Available at: https://www.ibm.com/topics/exploratory-data-analysis
[Accessed 30 March 2024].

Joby, A., 2021. *What Is Cross-Validation? Comparing Machine Learning Models.* [Online]
Available at: https://learn.g2.com/cross-validation#:~:text=Cross%2Dvalidation%20(CV)%20is,this%20sample%20to%20evaluate%20it.
[Accessed 31 March 2024].

Kanade, V., 2022. *What Is Logistic Regression? Equation, Assumptions, Types, and Best Practices.* [Online]
Available at: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-logistic-regression/#:~:text=Logistic%20regression%20is%20a%20supervised%20machine%20learning%20algorithm%20that%20accomplishes,1%2C%20or%20true%2Ffalse.
[Accessed 31 March 2024].

Keita, Z., 2022. *Classification in Machine Learning: An Introduction.* [Online]
Available at: https://www.datacamp.com/blog/classification-machine-learning
[Accessed 31 March 2024].

Khan, A. H. et al., 2023. A performance comparison of machine learning models for stock market prediction with novel investment strategy. *National Library of Medicine,* 18(9).

Kharwal, A., 2021. *Classification Report in Machine Learning.* [Online]
Available at: https://thecleverprogrammer.com/2021/07/07/classification-report-in-machine-learning/
[Accessed 31 March 2024].

LinkedIn, n.d. *What is the difference between a confusion matrix and a classification report?.* [Online]
Available at: https://www.linkedin.com/advice/3/what-difference-between-confusion-matrix-classification-hsehf
[Accessed 01 April 2024].

MarketLine, 2023. *Microsoft Corporation MarketLine Company Profile,* s.l.: s.n.

Microsoft, n.d. *Facts about Microsoft.* [Online]
Available at: https://news.microsoft.com/facts-about-microsoft/
[Accessed 29 March 2024].

Milosevic, N., n.d. *Equity forecast: Predicting Long Term Stock Price Movement using Machine Learning,* Manchester, UK: University of Manchester.

Python Package Index (PyPI), n.d. *yahooquery 2.3.7.* [Online]
Available at: https://pypi.org/project/yahooquery/
[Accessed 30 March 2024].

Raval, P., 2024. *Stock Price Prediction using Machine Learning with Source Code.* [Online]
Available at: https://www.projectpro.io/article/stock-price-prediction-using-machine-learning-project/571
[Accessed 30 March 2024].

scikit-learn, n.d. *Multiclass Receiver Operating Characteristic (ROC).* [Online]
Available at: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
[Accessed 01 April 2024].

scikit-learn, n.d. *sklearn.ensemble.ExtraTreesClassifier.* [Online]
Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html#:~:text=An%20extra%2Dtrees%20classifier.,accuracy%20and%20control%20over%2Dfitting.
[Accessed 31 March 2024].

Selvaraj, N., 2022. *Confusion Matrix, Precision, and Recall Explained.* [Online]
Available at: https://www.kdnuggets.com/2022/11/confusion-matrix-precision-recall-explained.html
[Accessed 01 April 2024].

Smith, T., 2023. *Random Walk Theory: Definition, How It's Used, and Example.* [Online]
Available at:
https://www.investopedia.com/terms/r/randomwalktheory.asp#:~:text=Random%20walk%20theory%20suggests%20that,and%20reflects%20all%20available%20information.
[Accessed 01 April 2024].

Yahoo Finance, 2024. *Microsoft Corporation.* [Online]
Available at: https://uk.finance.yahoo.com/quote/MSFT
[Accessed 30 March 2024].

Yao, Y., 2024. *Lecture Notes - Artificial Intelligence and Machine Learning.* s.l.:University of Westminster.

Zachary, G. P., Hall, M. & Montevirgen, K., 2024. *Encyclopaedia Britannica.* [Online]
Available at: https://www.britannica.com/topic/Microsoft-Corporation
[Accessed 29 March 2024].

Appendix: Coding

The jupyter notebook file for all relevant coding can be found on the below link:

https://github.com/vershasandesh/AI-ML/blob/main/AI%20and%20ML%20-%20Coursework%201%20-%2020395827.ipynb

The screenshots of codes are provided below in same sequence as the report headings.

REFER SECTION 2: RETRIEVING MICROSOFT'S STOCK INFORMATION

Microsoft Corporation

Getting Microsoft's Stock Data

In [18]:
```
! pip install yahooquery
```

```
Requirement already satisfied: yahooquery in c:\users\sande\anaconda3\lib\site-packages (2.3.7)
Requirement already satisfied: beautifulsoup4<5.0.0,>=4.12.2 in c:\users\sande\anaconda3\lib\site-packages (from yahooquery)
(4.12.2)
Requirement already satisfied: lxml<5.0.0,>=4.9.3 in c:\users\sande\anaconda3\lib\site-packages (from yahooquery) (4.9.3)
Requirement already satisfied: pandas<3.0.0,>=2.0.3 in c:\users\sande\anaconda3\lib\site-packages (from yahooquery) (2.0.3)
Requirement already satisfied: requests<3.0.0,>=2.31.0 in c:\users\sande\anaconda3\lib\site-packages (from yahooquery) (2.31.0)
Requirement already satisfied: requests-futures<2.0.0,>=1.0.1 in c:\users\sande\anaconda3\lib\site-packages (from yahooquery)
(1.0.1)
Requirement already satisfied: tqdm<5.0.0,>=4.65.0 in c:\users\sande\anaconda3\lib\site-packages (from yahooquery) (4.65.0)
Requirement already satisfied: soupsieve>1.2 in c:\users\sande\anaconda3\lib\site-packages (from beautifulsoup4<5.0.0,>=4.12.2-
>yahooquery) (2.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\sande\anaconda3\lib\site-packages (from pandas<3.0.0,>=2.0.3-
>yahooquery) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\sande\anaconda3\lib\site-packages (from pandas<3.0.0,>=2.0.3->yahooquer
y) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\sande\anaconda3\lib\site-packages (from pandas<3.0.0,>=2.0.3->yahooqu
ery) (2023.3)
Requirement already satisfied: numpy>=1.21.0 in c:\users\sande\anaconda3\lib\site-packages (from pandas<3.0.0,>=2.0.3->yahooque
ry) (1.24.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\sande\anaconda3\lib\site-packages (from requests<3.0.0,>=2.
31.0->yahooquery) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\sande\anaconda3\lib\site-packages (from requests<3.0.0,>=2.31.0->yahooq
uery) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\sande\anaconda3\lib\site-packages (from requests<3.0.0,>=2.31.0->
yahooquery) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\sande\anaconda3\lib\site-packages (from requests<3.0.0,>=2.31.0->
```

In [257]:
```
from yahooquery import Ticker
msft = Ticker('msft')
```

Data Loading

In [258]:
```
msft.history()  #This shows to the recent data by default
```

Out[258]:

symbol	date	open	high	low	close	volume	adjclose	dividends
msft	2024-01-02	373.859985	375.899994	366.769989	370.869995	25258600	370.185425	0.0
	2024-01-03	369.010010	373.260010	368.510010	370.600006	23083500	369.915924	0.0
	2024-01-04	370.670013	373.100006	367.170013	367.940002	20901500	367.260834	0.0
	2024-01-05	368.970001	372.059998	366.500000	367.750000	20987000	367.071198	0.0
	2024-01-08	369.299988	375.200012	369.010010	374.690002	23134000	373.998383	0.0
	...	...	...	...	...	...	...	...
	2024-03-22	429.700012	429.859985	426.070007	428.739990	17636500	428.739990	0.0
	2024-03-25	425.239990	427.410004	421.609985	422.859985	18060500	422.859985	0.0
	2024-03-26	425.609985	425.989990	421.350006	421.649994	16725600	421.649994	0.0
	2024-03-27	424.440002	424.450012	419.010010	421.429993	16705000	421.429993	0.0
	2024-03-28	420.959991	421.869995	419.119995	420.720001	21871200	420.720001	0.0

61 rows × 7 columns

In [313]: df … data

Out[313]:

In [260]:
```
# Save data in memory to disk
df.to_csv('MSFT_stock_data.csv')
```

Data Preprocessing

Import requried libraries

In [261]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [262]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import RocCurveDisplay
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import cross_val_score
import seaborn as sns
%matplotlib inline
```

In [263]:
```
pd.options.mode.chained_assignment = None
```

Data Reduction

In [264]:
```
dataset = pd.read_csv('MSFT_stock_data.csv')
dataset = dataset.dropna()                    # Reducing the data by dropping unnecessary columns, like adjclose, dividends, etc
dataset = dataset[['date','volume','open', 'high', 'low', 'close']]
dataset.head()
```

Out[264]:

	date	volume	open	high	low	close
0	2014-01-02	30632200	37.349998	37.400002	37.099998	37.160000
1	2014-01-03	31134800	37.200001	37.220001	36.599998	36.910000
2	2014-01-06	43603700	36.849998	36.889999	36.110001	36.130001
3	2014-01-07	35802800	36.330002	36.490002	36.209999	36.410000
4	2014-01-08	59971700	36.000000	36.139999	35.580002	35.759998

REFER SECTION 3: FEATURES CREATION

Features Creation

In [265]:
```
dataset['H-L'] = dataset['high'] - dataset['low']
dataset['C-O'] = dataset['close'] - dataset['open']
dataset['3day MA'] = dataset['close'].shift(1).rolling(window = 3).mean()
dataset['10day MA'] = dataset['close'].shift(1).rolling(window = 10).mean()

dataset['30day MA'] = dataset['close'].shift(1).rolling(window = 30).mean()
dataset['Std_dev']= dataset['close'].rolling(5).std()

dataset['Price_Rise'] = np.where(dataset['close'].shift(-1) > dataset['close'], 1, 0)
dataset = dataset.dropna()
dataset.head()
```

Out[265]:

	date	volume	open	high	low	close	H-L	C-O	3day MA	10day MA	30day MA	Std_dev	Price_Rise
30	2014-02-14	31407500	37.389999	37.779999	37.330002	37.619999	0.449997	0.230000	37.416667	36.828	36.460000	0.349472	0
31	2014-02-18	32834000	37.630001	37.779999	37.410000	37.419998	0.369999	-0.210003	37.566667	36.806	36.475333	0.182949	1
32	2014-02-19	29750400	37.220001	37.750000	37.209999	37.509998	0.540001	0.289997	37.549999	36.900	36.492333	0.087350	1
33	2014-02-20	27526100	37.570000	37.869999	37.400002	37.750000	0.469997	0.180000	37.516665	37.016	36.538333	0.124379	1
34	2014-02-21	38021300	37.939999	38.349998	37.860001	37.980000	0.489998	0.040001	37.559999	37.209	36.583000	0.219158	0

REFER SECTION 4: EXPLORATORY DATA ANALYSIS (EDA)

Exploratory Data Analysis (EDA)

Descriptive Statistics

```
In [266]: dataset.describe()    # To show the summary statistics
```

Out[266]:

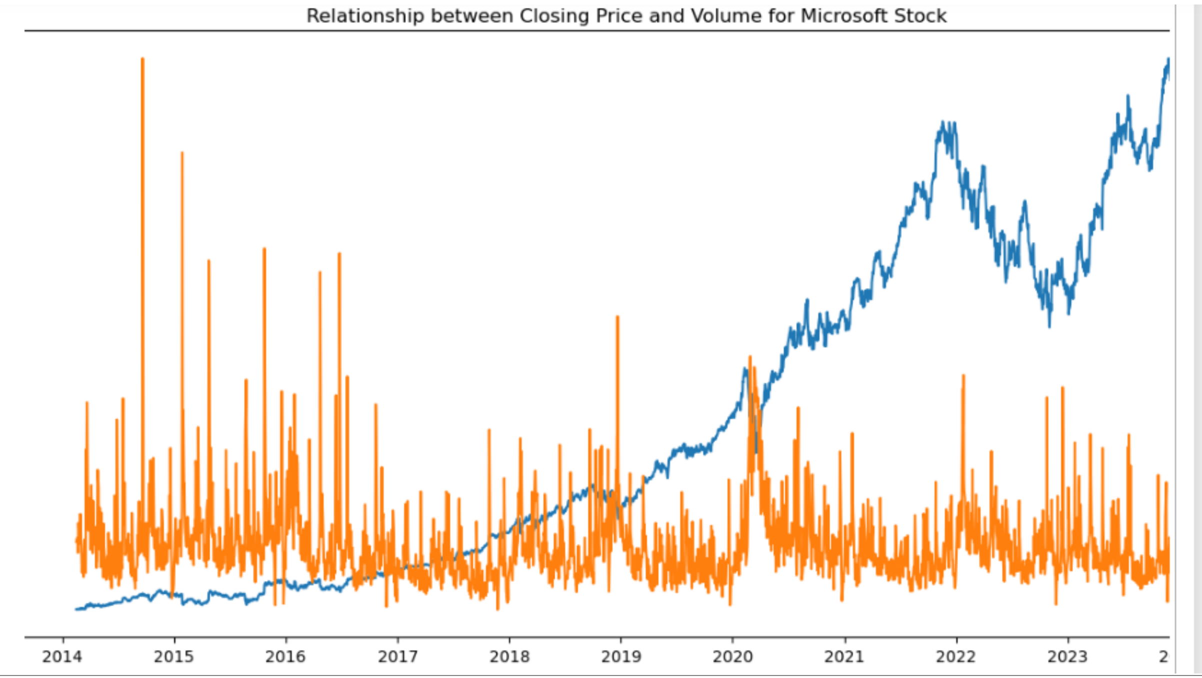
	volume	open	high	low	close	H-L	C-O	3day MA	10day MA	30day MA	Std_dev	Pric
count	2.486000e+03	2486.000000	2486.000000	2486.000000	2486.000000	2486.000000	2486.000000	2486.000000	2486.000000	2486.000000	2486.000000	2486.
mean	2.999795e+07	151.169373	152.676227	149.620177	151.224554	3.056050	0.055181	150.952656	150.478178	149.123866	2.112427	0.
std	1.382120e+07	101.449258	102.503234	100.357030	101.480702	2.924295	2.562814	101.310759	101.008163	100.144333	2.155757	0.
min	7.425600e+06	37.220001	37.740002	37.189999	37.419998	0.240002	-15.670013	37.416667	36.806000	36.460000	0.050301	0.
25%	2.161240e+07	56.602499	56.952501	56.222499	56.692500	0.850002	-0.650002	56.613333	56.427500	54.957083	0.529041	0.
50%	2.678650e+07	111.415001	112.195000	110.389999	111.704998	1.849998	0.065002	110.970000	110.306499	109.459833	1.300255	1.
75%	3.407525e+07	243.834999	245.877499	241.410000	244.327496	4.509979	0.790001	243.892497	243.327501	242.100499	3.017633	1.
max	2.025224e+08	383.760010	384.299988	378.160004	382.700012	24.609985	22.079987	380.153341	377.088004	373.869002	14.000517	1.

```
In [267]: dataset.info()

<class 'pandas.core.frame.DataFrame'>
Index: 2486 entries, 30 to 2515
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   date        2486 non-null   object
 1   volume      2486 non-null   int64
 2   open        2486 non-null   float64
 3   high        2486 non-null   float64
 4   low         2486 non-null   float64
 5   close       2486 non-null   float64
 6   H-L         2486 non-null   float64
 7   C-O         2486 non-null   float64
 8   3day MA     2486 non-null   float64
 9   10day MA    2486 non-null   float64
 10  30day MA    2486 non-null   float64
 11  Std_dev     2486 non-null   float64
 12  Price_Rise  2486 non-null   int32
dtypes: float64(10), int32(1), int64(1), object(1)
memory usage: 262.2+ KB
```

```
In [268]: print(dataset.isnull().sum()) # Checking for missing values

date          0
volume        0
open          0
high          0
low           0
close         0
H-L           0
C-O           0
3day MA       0
10day MA      0
30day MA      0
Std_dev       0
Price_Rise    0
dtype: int64
```

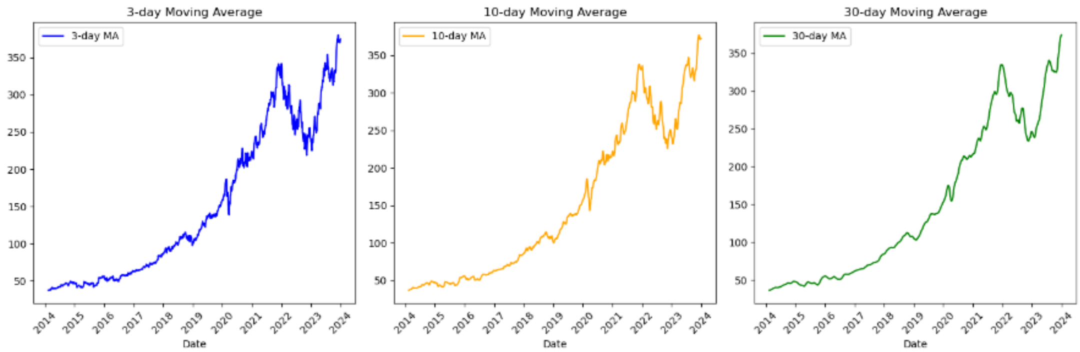
Data Visualisation: Plot the data

In [269]:
```
dataset['date'] = pd.to_datetime(dataset['date'])

fig, ax1 = plt.subplots(figsize=(12, 6))

color1 = 'tab:blue'
ax1.set_ylabel('Closing Price', color=color1)
ax1.plot(dataset['date'], dataset['close'], label='Close', color=color1)  # Using 'date' column for x-axis
ax1.tick_params(axis='y', labelcolor=color1)

ax2 = ax1.twinx()
color2 = 'tab:orange'
ax2.set_ylabel('Volume', color=color2)
ax2.plot(dataset['date'], dataset['volume'], label='Volume', color=color2)  # Using 'date' column for x-axis
ax2.tick_params(axis='y', labelcolor=color2)

plt.title('Relationship between Closing Price and Volume for Microsoft Stock')
fig.tight_layout()
plt.show()
```



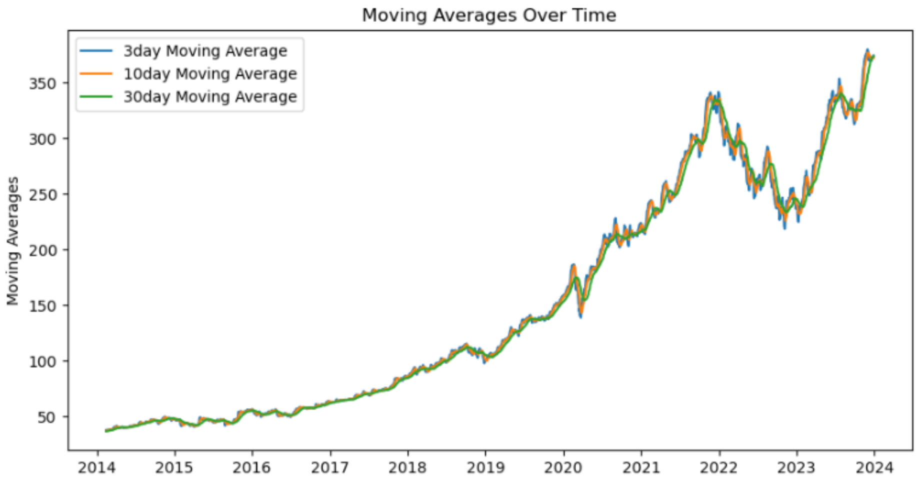
In [316]:

```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5), sharex=True)

# Plot 3-day moving average in the first subplot
axes[0].plot(dataset['date'], dataset['3day MA'], label='3-day MA', color='blue')
axes[0].set_title('3-day Moving Average')

# Plot 10-day moving average in the second subplot
axes[1].plot(dataset['date'], dataset['10day MA'], label='10-day MA', color='orange')
axes[1].set_title('10-day Moving Average')

# Plot 30-day moving average in the third subplot
axes[2].plot(dataset['date'], dataset['30day MA'], label='30-day MA', color='green')
axes[2].set_title('30-day Moving Average')

# Set common xlabel for all subplots
for ax in axes:
    ax.set_xlabel('Date')

# Rotate x-axis labels for better readability
for ax in axes:
    plt.sca(ax)
    plt.xticks(rotation=45)

# Add legend to each subplot
for ax in axes:
    ax.legend()

plt.tight_layout()  # Adjust subplots to prevent overlap
plt.show()
```

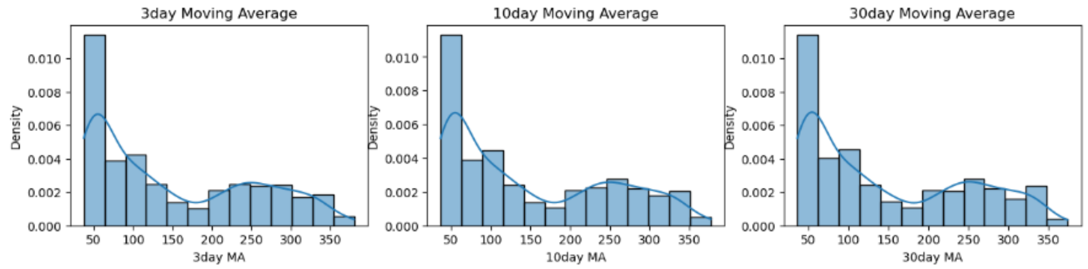


In [271]:

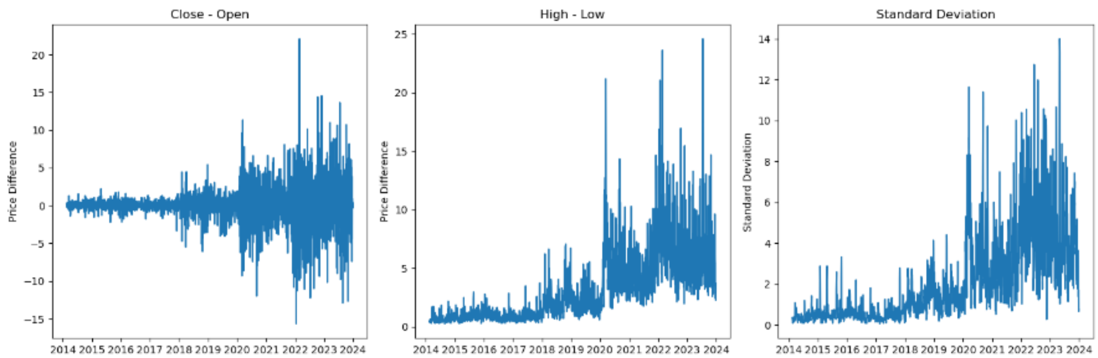
```
plt.figure(figsize=(10,5))  # Comparing all moving averages over time

plt.plot(dataset['date'], dataset['3day MA'], label='3day Moving Average')
plt.plot(dataset['date'], dataset['10day MA'], label='10day Moving Average')
plt.plot(dataset['date'], dataset['30day MA'], label='30day Moving Average')

plt.title('Moving Averages Over Time')
plt.ylabel('Moving Averages')
plt.legend()
plt.show()
```



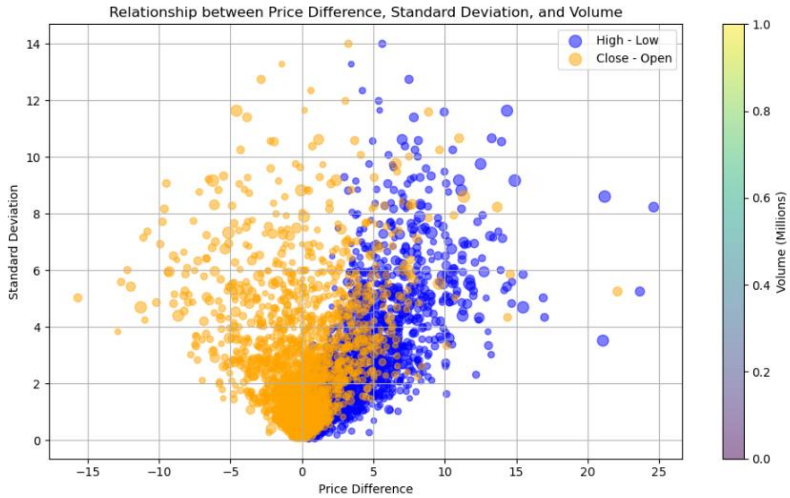
```
In [275]: fig, axes = plt.subplots(1, 3, figsize=(15, 3))  # Creating subplots in 1 row and 3 columns

sns.histplot(data=dataset, x="3day MA", kde=True, stat="density", ax=axes[0])
axes[0].set_title('3day Moving Average')

sns.histplot(data=dataset, x="10day MA", kde=True, stat="density", ax=axes[1])
axes[1].set_title('10day Moving Average')

sns.histplot(data=dataset, x="30day MA", kde=True, stat="density", ax=axes[2])
axes[2].set_title('30day Moving Average')

plt.show()
```



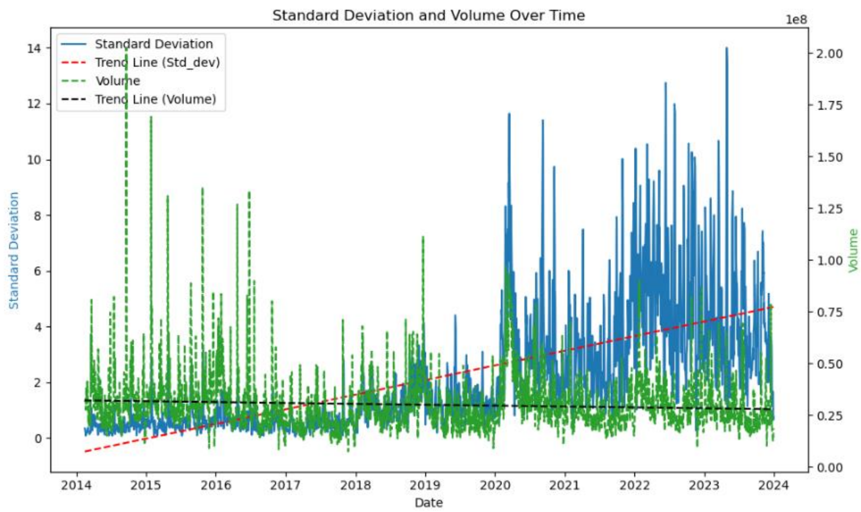

```
In [272]: fig, axes = plt.subplots(1, 3, figsize=(15,5))

axes[0].plot(dataset['date'], dataset['C-O'])       # Plot Price difference (Close - Open) in the first subplot
axes[0].set_title('Close - Open')
axes[0].set_ylabel('Price Difference')
axes[1].plot(dataset['date'], dataset['H-L'])       # Plot Price Difference (High - Low) in the second subplot
axes[1].set_title('High - Low')
axes[1].set_ylabel('Price Difference')
axes[2].plot(dataset['date'], dataset['Std_dev'])   # Plot Standard Deviation in the third subplot
axes[2].set_title('Standard Deviation')
axes[2].set_ylabel('Standard Deviation')

plt.tight_layout()  # Adjust subplots to prevent overlap
plt.show()
```



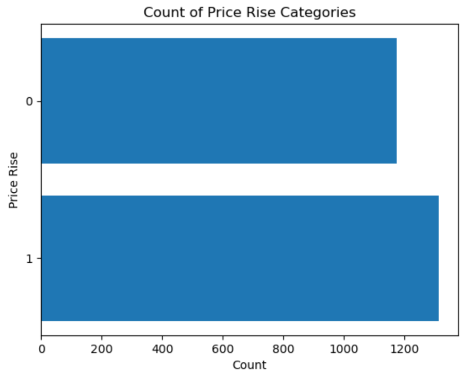

```
In [273]: plt.figure(figsize=(10, 6))

# Scatter plot of Price Difference vs. Standard Deviation with marker size based on volume
plt.scatter(dataset['H-L'], dataset['Std_dev'], s=dataset['volume']/1e6, c='blue', alpha=0.5, label='High - Low')
plt.scatter(dataset['C-O'], dataset['Std_dev'], s=dataset['volume']/1e6, c='orange', alpha=0.5, label='Close - Open')

plt.xlabel('Price Difference')
plt.ylabel('Standard Deviation')
plt.title('Relationship between Price Difference, Standard Deviation, and Volume')
plt.legend()

# Add color bar indicating the range of volume
plt.colorbar(label='Volume (Millions)')

plt.grid(True)
plt.tight_layout()
plt.show()
```



In [274]:
```
fig, ax1 = plt.subplots(figsize=(10, 6))

# Plot standard deviation on the primary y-axis with trend line
color1 = 'tab:blue'
ax1.set_xlabel('Date')
ax1.set_ylabel('Standard Deviation', color=color1)
ax1.plot(dataset['date'], dataset['Std_dev'], color=color1, label='Standard Deviation')

# Calculate trend line for standard deviation
z = np.polyfit(dataset.index, dataset['Std_dev'], 1)
p = np.poly1d(z)
ax1.plot(dataset['date'], p(dataset.index), color='red', linestyle='--', label='Trend Line (Std_dev)')

# Create a secondary y-axis for volume
ax2 = ax1.twinx()
color2 = 'tab:green'
ax2.set_ylabel('Volume', color=color2)
ax2.plot(dataset['date'], dataset['volume'], color=color2, linestyle='--', label='Volume')

# Calculate trend line for volume
z_volume = np.polyfit(dataset.index, dataset['volume'], 1)
p_volume = np.poly1d(z_volume)
ax2.plot(dataset['date'], p_volume(dataset.index), color='black', linestyle='--', label='Trend Line (Volume)')

# Combine the legends
lines1, labels1 = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax1.legend(lines1 + lines2, labels1 + labels2, loc='upper left')

# Set title
plt.title('Standard Deviation and Volume Over Time')

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

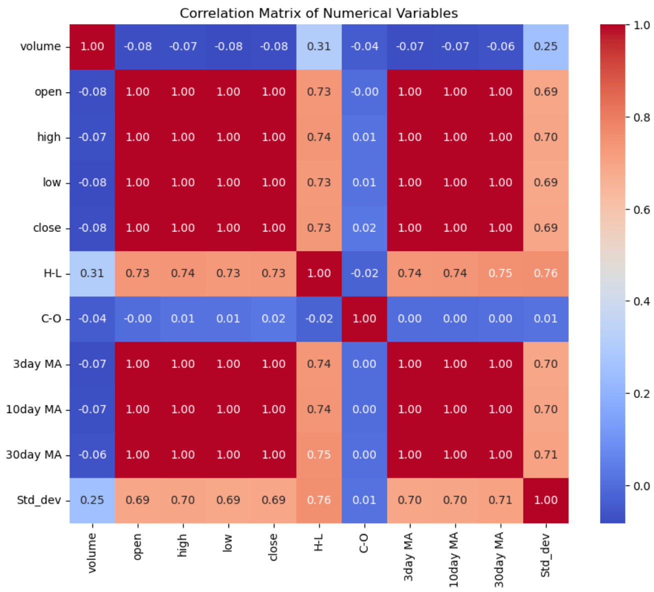


In [278]:
```
# Calculate the count of occurrences for each category of Price_Rise
price_rise_counts = dataset['Price_Rise'].value_counts()

# Define the range of values for the y-axis
range1 = range(len(price_rise_counts))

# Create a horizontal bar plot
plt.title('Count of Price Rise Categories')
plt.barh(range1, price_rise_counts)

# Set y-axis ticks and labels
plt.yticks(range1, price_rise_counts.index)
plt.ylabel('Price Rise')
plt.xlabel('Count')

plt.show()
```



Correlation Matrix

In [276]:
```
numerical_cols = ['volume', 'open', 'high', 'low', 'close', 'H-L', 'C-O', '3day MA', '10day MA', '30day MA', 'Std_dev']

# Calculate the correlation matrix
correlation_matrix = dataset[numerical_cols].corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", square=True)

plt.title('Correlation Matrix of Numerical Variables')
plt.show()
```



	volume	open	high	low	close	H-L	C-O	3day MA	10day MA	30day MA	Std_dev
volume	1.00	-0.08	-0.07	-0.08	-0.08	0.31	-0.04	-0.07	-0.07	-0.06	0.25
open	-0.08	1.00	1.00	1.00	1.00	0.73	-0.00	1.00	1.00	1.00	0.69
high	-0.07	1.00	1.00	1.00	1.00	0.74	0.01	1.00	1.00	1.00	0.70
low	-0.08	1.00	1.00	1.00	1.00	0.73	0.01	1.00	1.00	1.00	0.69
close	-0.08	1.00	1.00	1.00	1.00	0.73	0.02	1.00	1.00	1.00	0.69
H-L	0.31	0.73	0.74	0.73	0.73	1.00	-0.02	0.74	0.74	0.75	0.76
C-O	-0.04	-0.00	0.01	0.01	0.02	-0.02	1.00	0.00	0.00	0.00	0.01
3day MA	-0.07	1.00	1.00	1.00	1.00	0.74	0.00	1.00	1.00	1.00	0.70
10day MA	-0.07	1.00	1.00	1.00	1.00	0.74	0.00	1.00	1.00	1.00	0.70
30day MA	-0.06	1.00	1.00	1.00	1.00	0.75	0.00	1.00	1.00	1.00	0.71
Std_dev	0.25	0.69	0.70	0.69	0.69	0.76	0.01	0.70	0.70	0.71	1.00

REFER SECTION 5: MACHINE LEARNING CLASSIFICATION METHODS

Machine Learning Classification Methods

In [280]:
```
# Assigning data to the variables
# X includes columns from the 4th index (or fifth column) of the dataset up to the second-to-last column,
# and y corresponds to the last column that we want to predict (Price_Rise).
X = dataset.iloc[:, 6:-1]
Y = dataset.iloc[:, -1]
```

In [281]:
```
X
```

Out[281]:

	H-L	C-O	3day MA	10day MA	30day MA	Std_dev
30	0.449997	0.230000	37.416667	36.828000	36.460000	0.349472
31	0.369999	-0.210003	37.566667	36.806000	36.475333	0.182949
32	0.540001	0.289997	37.549999	36.900000	36.492333	0.087350
33	0.469997	0.180000	37.516665	37.016000	36.538333	0.124379
34	0.489998	0.040001	37.559999	37.209000	36.583000	0.219158
...	...	...	...	...	...	...
2511	2.470001	0.899994	372.473338	372.101001	372.826336	1.467821
2512	3.440002	-0.339996	372.913330	372.135999	373.289335	1.637413
2513	2.250000	0.380005	374.260000	372.472000	373.455668	1.668288
2514	2.299988	-0.089996	374.436666	372.441000	373.702002	0.655648
2515	3.679993	0.040009	374.670003	372.532001	373.869002	0.756560

2486 rows × 6 columns

In [282]:
```
Y
```

Out[282]:
```
30      0
31      1
32      1
33      1
34      0
       ..
2511    1
2512    0
2513    1
2514    1
2515    0
Name: Price_Rise, Length: 2486, dtype: int32
```

In [283]:
```
# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, shuffle=False)
X_test
```

Out[283]:

	H-L	C-O	3day MA	10day MA	30day MA	Std_dev
2018	7.210022	0.730011	326.713338	334.932007	332.480002	10.386779
2019	6.410004	-0.109985	319.756673	333.000006	331.686669	9.641449
2020	10.029999	4.779999	314.766673	330.935007	330.891002	6.504658
2021	6.719971	1.600006	314.063334	328.117004	330.377336	1.023867
2022	6.330017	-1.400024	314.430003	325.490005	329.655669	1.827799
...	...	...	...	...	...	...
2511	2.470001	0.899994	372.473338	372.101001	372.826336	1.467821
2512	3.440002	-0.339996	372.913330	372.135999	373.289335	1.637413
2513	2.250000	0.380005	374.260000	372.472000	373.455668	1.668288
2514	2.299988	-0.089996	374.436666	372.441000	373.702002	0.655648
2515	3.679993	0.040009	374.670003	372.532001	373.869002	0.756560

498 rows × 6 columns

In [284]:
```
        # Initialize StandardScaler
        sc = StandardScaler()

        # Fit and transform X_train
        X_train_scaled = sc.fit_transform(X_train)

        # Transform X_test
        X_test_scaled = sc.transform(X_test)
```

Logistic Regression

In [285]:
```
# Initialize logistic regression model
log_reg = LogisticRegression()

# Train the model
log_reg.fit(X_train_scaled, Y_train)

# Predictions on the test set
y_pred_logistic = log_reg.predict(X_test_scaled)

# Evaluate the model
print (classification_report(Y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.51      0.34      0.41       246
           1       0.51      0.68      0.59       252

    accuracy                           0.51       498
   macro avg       0.51      0.51      0.50       498
weighted avg       0.51      0.51      0.50       498
```

Extra Trees Classifier

In [288]:
```
# Extra Trees Classifier Model
classifier = ExtraTreesClassifier(random_state=101)
# Train the Model
classifier.fit(X_train, Y_train)
```

Out[288]:
```
ExtraTreesClassifier
ExtraTreesClassifier(random_state=101)
```

REFER SECTION 6: CROSS-VALIDATION ANALYSIS

Cross-Validation

In [292]:
```
# Initialize logistic regression model
log_reg = LogisticRegression(random_state=101)

# Perform cross-validation for logistic regression
log_reg_scores = cross_val_score(log_reg, X_train, Y_train, cv=5)

# Print cross-validation scores for logistic regression
print("Cross-Validation Scores for Logistic Regression:")
print(log_reg_scores)
print("Mean Accuracy based on Cross-Validation Score: {:.2f}".format(round(log_reg_scores.mean(), 2)))
print("Standard Deviation of Cross-Validation Scores: {:.4f}".format(round(log_reg_scores.std(), 4)))
print()
```

```
Cross-Validation Scores for Logistic Regression:
[0.52512563 0.53517588 0.52512563 0.52644836 0.53400504]
Mean Accuracy based on Cross-Validation Score: 0.53
Standard Deviation of Cross-Validation Scores: 0.0045
```

In [293]:
```
# Perform cross-validation for ExtraTreesClassifier
extra_trees_scores = cross_val_score(classifier, X_train, Y_train, cv=5)

# Print cross-validation scores for ExtraTreesClassifier
print("Cross-Validation Scores for ExtraTreesClassifier:")
print(extra_trees_scores)
print("Mean Accuracy based on Cross-Validation Score: {:.2f}".format(round(extra_trees_scores.mean(), 2)))
print("Standard Deviation of Cross-Validation Scores: {:.4f}".format(round(extra_trees_scores.std(), 4)))
```

```
Cross-Validation Scores for ExtraTreesClassifier:
[0.46733668 0.40954774 0.45728643 0.52141058 0.45843829]
Mean Accuracy based on Cross-Validation Score: 0.46
Standard Deviation of Cross-Validation Scores: 0.0356
```

Refer Section 7: Evaluation of Classification Methods

```
In [294]: # Compare mean cross-validation scores
if log_reg_scores.mean() > extra_trees_scores.mean():
    print("Logistic Regression performs better with a mean cross-validation score of {:.2f} and standard deviation of {:.4f}".for
elif log_reg_scores.mean() < extra_trees_scores.mean():
    print("ExtraTreesClassifier performs better with a mean cross-validation score of {:.2f} and standard deviation of {:.4f}".fc
else:
    print("Both models have the same mean cross-validation score of {:.2f} and standard deviation of {:.4f}".format(log_reg_score
```

```
Logistic Regression performs better with a mean cross-validation score of 0.53 and standard deviation of 0.0045
```

Logistic Regression

```
In [285]: # Initialize logistic regression model
log_reg = LogisticRegression()

# Train the model
log_reg.fit(X_train_scaled, Y_train)

# Predictions on the test set
y_pred_logistic = log_reg.predict(X_test_scaled)

# Evaluate the model
print (classification_report(Y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.51      0.34      0.41       246
           1       0.51      0.68      0.59       252

    accuracy                           0.51       498
   macro avg       0.51      0.51      0.50       498
weighted avg       0.51      0.51      0.50       498
```

Extra Trees Classifier

```
In [288]: # Extra Trees Classifier Model
classifier = ExtraTreesClassifier(random_state=101)
# Train the Model
classifier.fit(X_train, Y_train)
```

```
Out[288]: ExtraTreesClassifier
ExtraTreesClassifier(random_state=101)
```

```
In [289]: Y_pred = classifier.predict(X_test)
print(classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.49      0.42      0.45       246
           1       0.50      0.57      0.53       252

    accuracy                           0.50       498
   macro avg       0.50      0.50      0.49       498
weighted avg       0.50      0.50      0.49       498
```

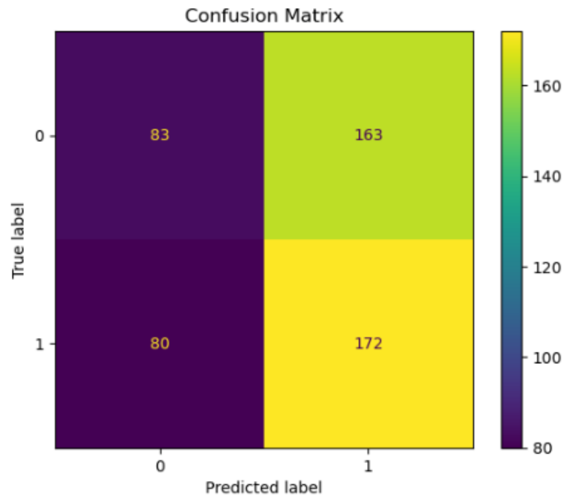
Refer Section 8: Predicting MSFT Price using Logistic Regression Model

Logistic Regression

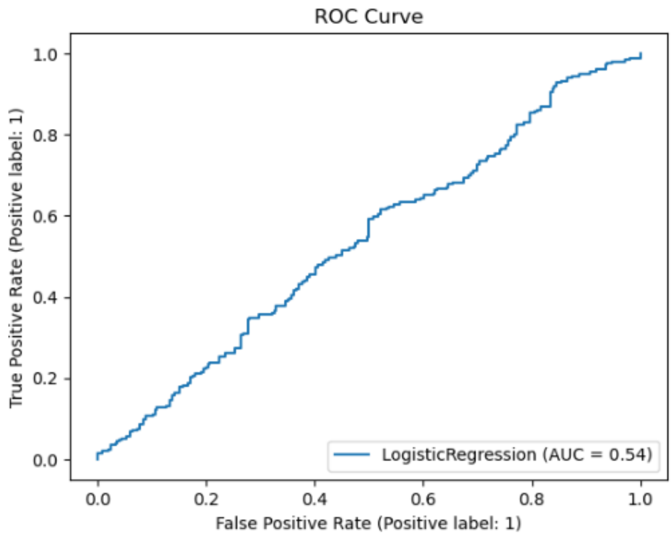
```
In [285]: # Initialize logistic regression model
log_reg = LogisticRegression()

# Train the model
log_reg.fit(X_train_scaled, Y_train)

# Predictions on the test set
y_pred_logistic = log_reg.predict(X_test_scaled)

# Evaluate the model
print (classification_report(Y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.51      0.34      0.41       246
           1       0.51      0.68      0.59       252

    accuracy                           0.51       498
   macro avg       0.51      0.51      0.50       498
weighted avg       0.51      0.51      0.50       498
```

```
In [286]: # Predictions on the test set
y_pred = log_reg.predict(X_test_scaled)

# Evaluate the model by means of a Confusion Matrix
matrix = ConfusionMatrixDisplay.from_estimator(log_reg, X_test_scaled, Y_test)

plt.title('Confusion Matrix')
plt.show(matrix)
```



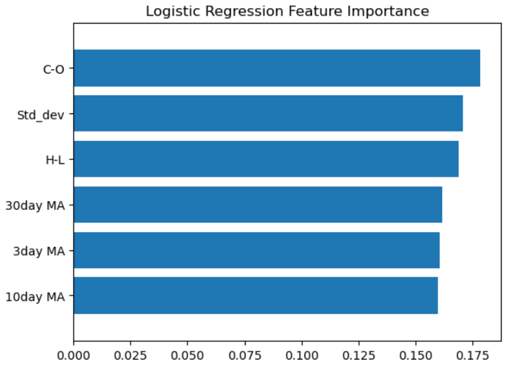

```
In [287]: # Create ROC curve display
roc_disp = RocCurveDisplay.from_estimator(log_reg, X_test_scaled, Y_test)

# Add title and show plot
plt.title('ROC Curve')
plt.show()
```



Features Importance

```
In [295]: feature_names=X.columns
```

```
In [296]: importance = classifier.feature_importances_
indices = np.argsort(importance)
range1 = range(len(importance[indices]))
plt.figure()
plt.title("Logistic Regression Feature Importance")
plt.barh(range1,importance[indices])
plt.yticks(range1, feature_names[indices])
plt.ylim([-1, len(range1)])
plt.show()
```



REFER SECTION 9: MARKET RETURNS AND STRATEGY RETURNS

Trading Strategies - Predicitng the Price using Logistic Regression Model

In [297]:
```
trade_dataset = dataset.copy() #Create a copy of original dataset for trade analysis
```

In [298]:
```
dataset['Y_pred'] = np.NaN
dataset.iloc[(len(dataset) - len(Y_pred)):,-1] = Y_pred
trade_dataset = dataset.dropna()
trade_dataset
```

	date	volume	open	high	low	close	H-L	C-O	3day MA	10day MA	30day MA	Std_dev	Price_Rise	Y_pred
2018	2022-01-06	39646100	313.149994	318.700012	311.489990	313.880005	7.210022	0.730011	326.713338	334.932007	332.480002	10.386779	1	0.0
2019	2022-01-07	32720000	314.149994	316.500000	310.089996	314.040009	6.410004	-0.109985	319.756673	333.000006	331.686669	9.641449	1	1.0
2020	2022-01-10	44289500	309.489990	314.720001	304.690002	314.269989	10.029999	4.779999	314.766673	330.935007	330.891002	6.504658	1	1.0
2021	2022-01-11	29386800	313.380005	316.609985	309.890015	314.980011	6.719971	1.600006	314.063334	328.117004	330.377336	1.023867	1	1.0
2022	2022-01-12	34372200	319.670013	323.410004	317.079987	318.269989	6.330017	-1.400024	314.430003	325.490005	329.655669	1.827799	0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2511	2023-12-22	17091100	373.679993	375.179993	372.709991	374.579987	2.470001	0.899994	372.473338	372.101001	372.826336	1.467821	1	0.0
2512	2023-12-26	12673100	375.000000	376.940002	373.500000	374.660004	3.440002	-0.339996	372.913330	372.135999	373.289335	1.637413	0	0.0
2513	2023-12-27	14905400	373.690002	375.059998	372.809998	374.070007	2.250000	0.380005	374.260000	372.472000	373.455668	1.668288	1	0.0
2514	2023-12-28	14327000	375.369995	376.459991	374.160004	375.279999	2.299988	-0.089996	374.436666	372.441000	373.702002	0.655648	1	0.0
2515	2023-12-29	18723000	376.000000	377.160004	373.480011	376.040009	3.679993	0.040009	374.670003	372.532001	373.869002	0.756560	0	0.0

98 rows × 14 columns

In [299]:
```
trade_dataset['Tomorrows Returns'] = 0. # Creating new column for Tomorrow's return
trade_dataset['Tomorrows Returns'] = np.log(trade_dataset['close'].shift(-1) / trade_dataset['close'])
trade_dataset
```

Out[299]:

	date	volume	open	high	low	close	H-L	C-O	3day MA	10day MA	30day MA	Std_dev	Price_Rise	Y_pre
2018	2022-01-06	39646100	313.149994	318.700012	311.489990	313.880005	7.210022	0.730011	326.713338	334.932007	332.480002	10.386779	1	0.
2019	2022-01-07	32720000	314.149994	316.500000	310.089996	314.040009	6.410004	-0.109985	319.756673	333.000006	331.686669	9.641449	1	1.
2020	2022-01-10	44289500	309.489990	314.720001	304.690002	314.269989	10.029999	4.779999	314.766673	330.935007	330.891002	6.504658	1	1.
2021	2022-01-11	29386800	313.380005	316.609985	309.890015	314.980011	6.719971	1.600006	314.063334	328.117004	330.377336	1.023867	1	1.
2022	2022-01-12	34372200	319.670013	323.410004	317.079987	318.269989	6.330017	-1.400024	314.430003	325.490005	329.655669	1.827799	0	1.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
2511	2023-12-22	17091100	373.679993	375.179993	372.709991	374.579987	2.470001	0.899994	372.473338	372.101001	372.826336	1.467821	1	0.
2512	2023-12-26	12673100	375.000000	376.940002	373.500000	374.660004	3.440002	-0.339996	372.913330	372.135999	373.289335	1.637413	0	0.
2513	2023-12-27	14905400	373.690002	375.059998	372.809998	374.070007	2.250000	0.380005	374.260000	372.472000	373.455668	1.668288	1	0.
2514	2023-12-28	14327000	375.369995	376.459991	374.160004	375.279999	2.299988	-0.089996	374.436666	372.441000	373.702002	0.655648	1	0.
2515	2023-12-29	18723000	376.000000	377.160004	373.480011	376.040009	3.679993	0.040009	374.670003	372.532001	373.869002	0.756560	0	0.

498 rows × 15 columns

In [300]:
```
trade_dataset['Strategy Returns'] = 0.  # Creating another column for Strategy Returns
trade_dataset['Strategy Returns'] = np.where(y_pred_logistic == 1, trade_dataset['Tomorrows Returns'], -trade_dataset['Tomorrows
trade_dataset
```

Out[300]:

open	high	low	close	H-L	C-O	3day MA	10day MA	30day MA	Std_dev	Price_Rise	Y_pred	Tomorrows Returns	Strategy Returns
313.149994	318.700012	311.489990	313.880005	7.210022	0.730011	326.713338	334.932007	332.480002	10.386779	1	0.0	0.000510	-0.000510
314.149994	316.500000	310.089996	314.040009	6.410004	-0.109985	319.756673	333.000006	331.686669	9.641449	1	1.0	0.000732	-0.000732
309.489990	314.720001	304.690002	314.269989	10.029999	4.779999	314.766673	330.935007	330.891002	6.504658	1	1.0	0.002257	-0.002257
313.380005	316.609985	309.890015	314.980011	6.719971	1.600006	314.063334	328.117004	330.377336	1.023867	1	1.0	0.010391	0.010391
319.670013	323.410004	317.079987	318.269989	6.330017	-1.400024	314.430003	325.490005	329.655669	1.827799	0	1.0	-0.043244	-0.043244
...	...	...	...	...	...	...	...	...	...	...	...	...	...
373.679993	375.179993	372.709991	374.579987	2.470001	0.899994	372.473338	372.101001	372.826336	1.467821	1	0.0	0.000214	0.000214
375.000000	376.940002	373.500000	374.660004	3.440002	-0.339996	372.913330	372.135999	373.289335	1.637413	0	0.0	-0.001576	-0.001576
373.690002	375.059998	372.809998	374.070007	2.250000	0.380005	374.260000	372.472000	373.455668	1.668288	1	0.0	0.003229	0.003229
375.369995	376.459991	374.160004	375.279999	2.299988	-0.089996	374.436666	372.441000	373.702002	0.655648	1	0.0	0.002023	0.002023
376.000000	377.160004	373.480011	376.040009	3.679993	0.040009	374.670003	372.532001	373.869002	0.756560	0	0.0	NaN	NaN

Refer Section 10: Cumulative Market and Strategy Returns

Cumulative Market Returns and Cumulative Strategy Returns

In [311]:
```
trade_dataset['Cumulative Market Returns'] = np.cumsum(trade_dataset['Tomorrows Returns'])
trade_dataset['Cumulative Strategy Returns'] = np.cumsum(trade_dataset['Strategy Returns'])
trade_dataset
```

Out[311]:

low	close	H-L	C-O	3day MA	10day MA	30day MA	Std_dev	Price_Rise	Y_pred	Tomorrows Returns	Strategy Returns	Cumulative Market Returns	Cumulative Strategy Returns
311.489990	313.880005	7.210022	0.730011	326.713338	334.932007	332.480002	10.386779	1	0.0	0.000510	-0.000510	0.000510	-0.000510
310.089996	314.040009	6.410004	-0.109985	319.756673	333.000006	331.686669	9.641449	1	1.0	0.000732	-0.000732	0.001242	-0.001242
304.690002	314.269989	10.029999	4.779999	314.766673	330.935007	330.891002	6.504658	1	1.0	0.002257	-0.002257	0.003498	-0.003498
309.890015	314.980011	6.719971	1.600006	314.063334	328.117004	330.377336	1.023867	1	1.0	0.010391	0.010391	0.013889	0.006892
317.079987	318.269989	6.330017	-1.400024	314.430003	325.490005	329.655669	1.827799	0	1.0	-0.043244	-0.043244	-0.029355	-0.036352
...	...	...	...	...	...	...	...	...	...	...	...	...	...
370.040009	373.540009	4.369995	0.980011	372.176666	371.842001	372.481669	1.393828	1	0.0	0.002780	0.002780	0.176795	0.504799
372.709991	374.579987	2.470001	0.899994	372.473338	372.101001	372.826336	1.467821	1	0.0	0.000214	0.000214	0.177008	0.505013
373.500000	374.660004	3.440002	-0.339996	372.913330	372.135999	373.289335	1.637413	0	0.0	-0.001576	-0.001576	0.175432	0.503437
372.809998	374.070007	2.250000	0.380005	374.260000	372.472000	373.455668	1.668288	1	0.0	0.003229	0.003229	0.178662	0.506666
374.160004	375.279999	2.299988	-0.089996	374.436666	372.441000	373.702002	0.655648	1	0.0	0.002023	0.002023	0.180685	0.508689

Plot of Cumulative Market and Strategy Returns

In [312]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,5))
plt.plot(trade_dataset['Cumulative Market Returns'], color='r', label='Market Returns')
plt.plot(trade_dataset['Cumulative Strategy Returns'], color='g', label='Strategy Returns')
plt.legend()
plt.show()
```