Understanding Stock Price Dynamics &Tailoring Investment Approaches of Tata Consultancy Services Limited

MSc FinTech and Business Analytics

Module: Artificial Intelligence and Machine Learning

Module Code: FNCE043W

Module Leader: Dr. Yumei Yao

Student Name: Sarika Patel

Student ID: W2019695

Word Count: 2183 words (Excluding cover page, Index, References, Appendix)

TABLE OF CONTENT

CONTENTS

ABSTRACT

This study examines how Tata Consultancy Services (TCS) stock price changes were predicted using machine learning approaches, namely Logistic Regression and Extra Trees classification algorithms, between January 2014 and December 2023. The study finds patterns and trends in historical stock data using in-depth Exploratory Data Analysis (EDA), which informs the forecast models. The research evaluates the models' performance using cross-validation, calculating accuracy, precision, recall, and F1-scores. It also compares the cumulative returns of conventional market strategies with those based on machine learning forecasts. This brief study casts doubt on machine learning's ability to anticipate stock price increases properly and offers a critical perspective on the technology's predictive capabilities in the financial industry. It also adds to the larger conversation about the application of machine learning.

1.INTRODUCTION

Tata Consultancy Services Limited (TCS) is an Indian multinational information technology (IT) services and consulting company headquartered in Mumbai. It functions in 150 sites around 46 countries and is a member of the Tata Group. It was said in September 2023 that TCS employed more than 616,000 people globally. TCS is the most valuable IT service brand globally, the second-largest Indian firm by market value, and the leading Big Tech (India) company. The company has generated consolidated revenues of US $27.9 billion in the year ended March 31, 2023, and is listed on the BSE and the NSE in India. (Tata Company, 2024)

2.DATA COLLECTION AND PROCESSING

A. HISTORICAL DATA

Collection of Tata Consultancy Services (TCS) daily stock data spanning from January 2014 to December 2023 was conducted utilizing Yahoo Finance. The dataset comprises 2444 rows, encompassing essential metrics such as open, high, low, and close prices, as well as adjusted close prices and stock volume. Data that was incomplete or missing with rows was eliminated.

B. FEATURE ENGINEERING

Generated featured from the processed data include:

i. The daily high-low ('H-L') and open-close ('O-C') differentials represent intraday price volatility and changes in stock prices throughout the trading day.

Formula: H-L= High Price- Low Price

O-C= Closing Price- Opening Price

By monitoring H-L and O-C differences, market participants gain valuable insights into market dynamics, price volatility, and potential trading opportunities, enhancing their decision-making processes in the financial markets.

ii. Moving averages ('3d MA', '10d MA', and '30d MA') show how the closing price changes over short, medium, and longer periods.

Formula: n-Day MA = Sum of closing prices for the last n days / n

By analysing these moving averages, market participants gain insights into short and medium-term price dynamics.

iii. The closing price's standard deviation ('Std_dev') measures daily price volatility over a 5-day rolling window.

Formula:

$$\text{Std\_dev}_t = \sqrt{\frac{\sum_{i=1}^{t}(X_i - \bar{X})^2}{t}}$$

Where:

- X_i is each closing price at time i.
- $\bar{X}$ is the mean closing price up to time t.
- The summation is performed over all closing prices up to time t.

This rolling window approach allows us to calculate the standard deviation dynamically over successive 5-day periods, capturing short-term price volatility in the financial markets.

iv. 'Price_Rise' - binary column which predicts price increases based on historical data.

$$\text{Price\_Rise}_t = \begin{cases} 1 & \text{if } \text{Close}_{t+1} > \text{Close}_t \\ 0 & \text{otherwise} \end{cases}$$

Where:

- Close_{t+1} is the closing price at the next time step.
- Close_t is the closing price at the current time step.
- Price_Rise_t is the binary column indicating whether the price rises (1) or not (0).

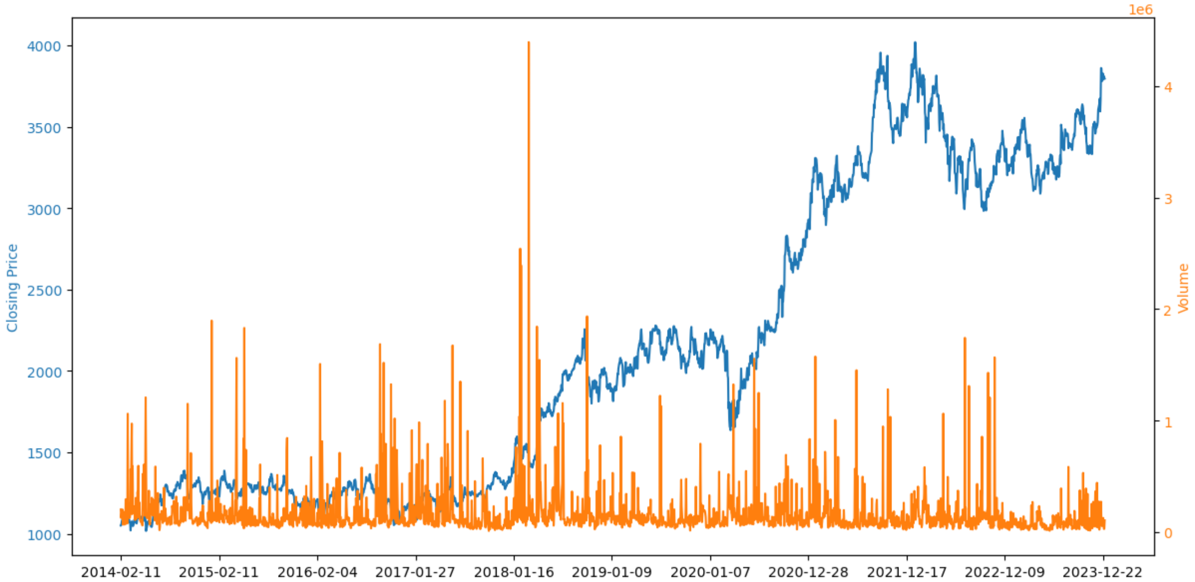
The 'Price_Rise' column is a binary indicator used to predict price movements based on historical data. Typically, it assigns a value of 1 when the price is predicted to rise and a value of 0 when the price is predicted to either stay the same or fall.

By removing rows with missing values and choosing the most pertinent TCS fields for analysis, the dataset was further refined.

3.EXPLORATORY DATA ANALYSIS OF FEATURES

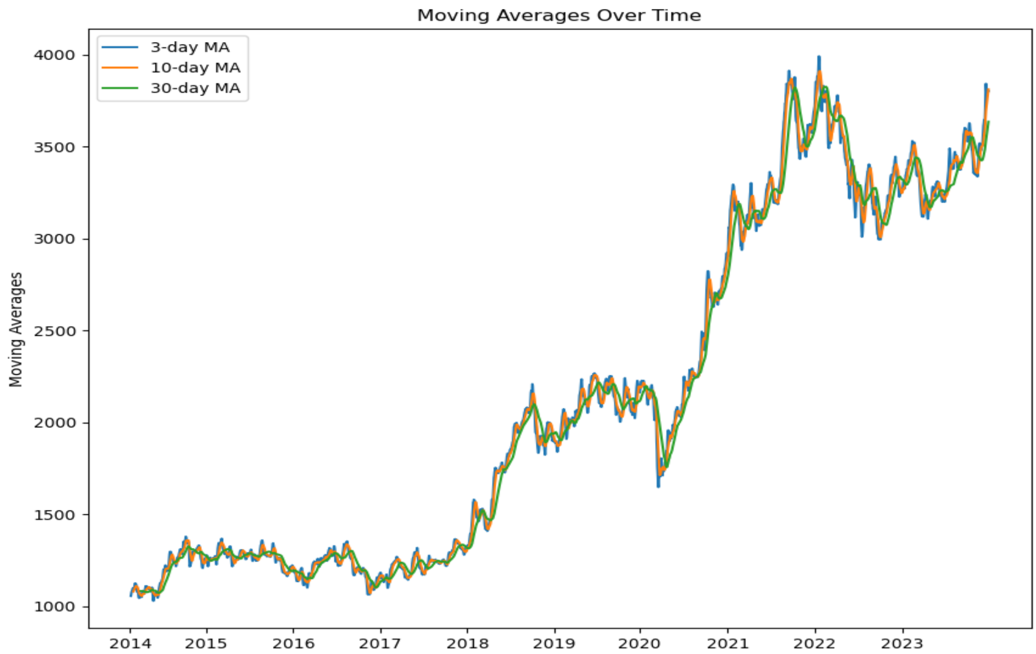
Before using advance statistical approaches, exploratory data analysis, or EDA, is a vital first stage in data analysis that aims to comprehend the key traits and patterns included within a dataset. In EDA, analysts usually look at the variables' distribution, central tendency, and dispersion, spot any outliers or missing data, and use graphs or charts to show the correlations between the variables. (IBM, 2024) EDA can provide insights into the data's structure, any correlations between variables, and any possible problems that would need to be fixed before additional modelling or analysis.

Figure 1: Closing price and volume Relationship.



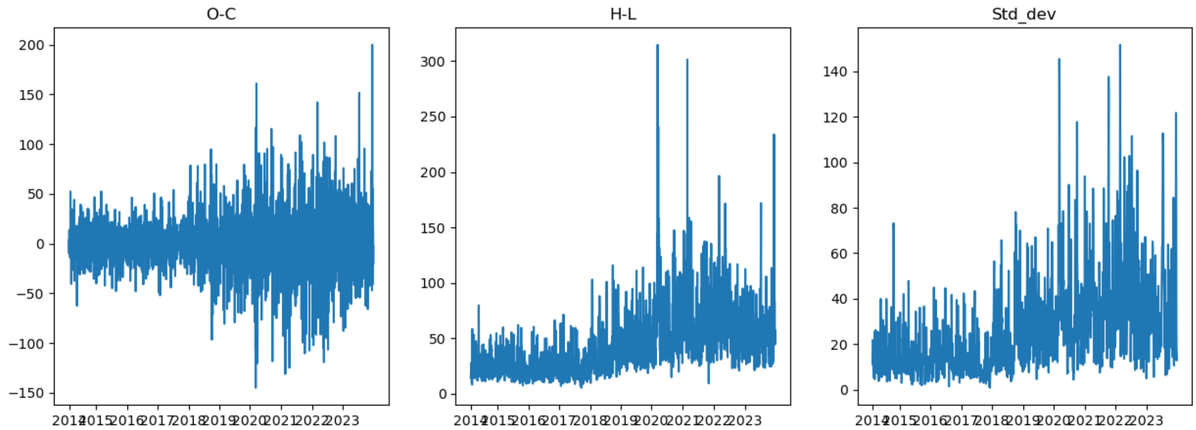
The chart depicts TCS stock's progress, showing an upward trajectory in the closing price against trading volume peaks. These peaks might reflect key events but do not consistently correlate with the stock's price, suggesting a complex interplay between volume and price dynamics. (Fig.1)

Figure 2: Plot for 3day, 10day, 30day MA



The moving averages such as '3-day MA', '10-day MA', and '30-day MA' help reduce short-term fluctuations in data. These moving averages exhibit comparable trends over time (Fig.1) and display similar distribution patterns (Fig.4). As the rolling window widens, the resulting trend line appears smoother, coinciding with a general upward trend in closing prices, with values frequently exceeding INR 3500 across the observed timeframe.

Figure:3. O-C, H-L and std_deviation



The 'H-L' and 'O-C' functions provide data on price fluctuations and market sentiment. The market condition in "H-L," also referred to as the daily price range, indicates that it normally rises with time, and the O-C price difference is larger as time goes on (Fig.3). The daily standard deviation moves in the same way as the "H-L" characteristic, which indicates that a larger "Std_dev" corresponds to a wider range of possible price movement and a higher degree of risk.

Figure 4: 3day,10day,30days MA histogram and density

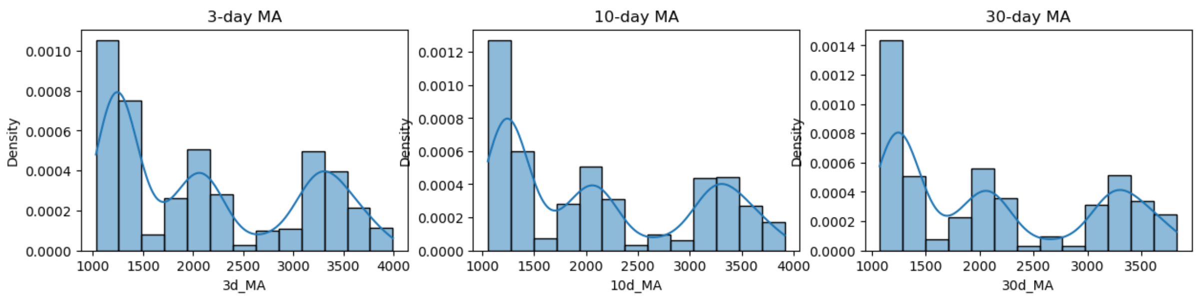
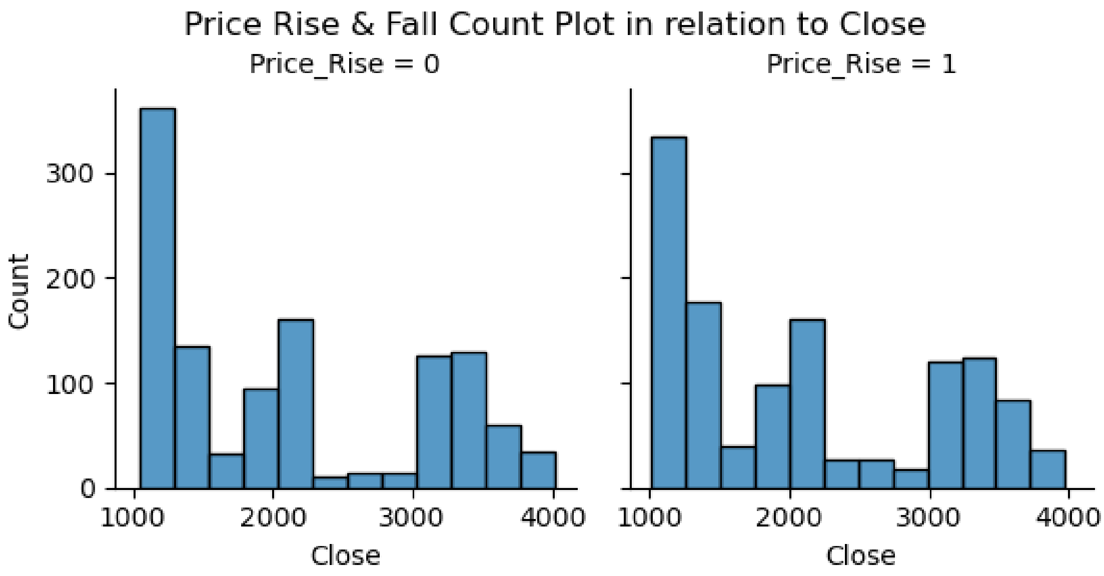
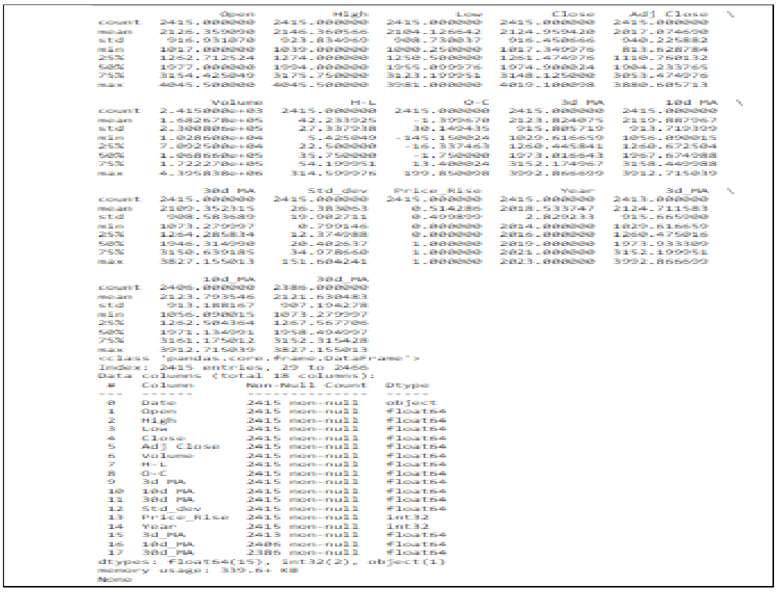


Figure 5: Price Rise & Fall Count Plot in relation to 'Close' and 'Std_dev'



The price rise and fall binary column (derived from our dataset's 'Price_Rise' feature; see Fig. 5) exhibits a negligible difference in the counts of upward and downward movements. The closing price and standard deviation predictions are generally somewhat positive oriented, showing significant consistency in some regions where the data points are densely clustered.

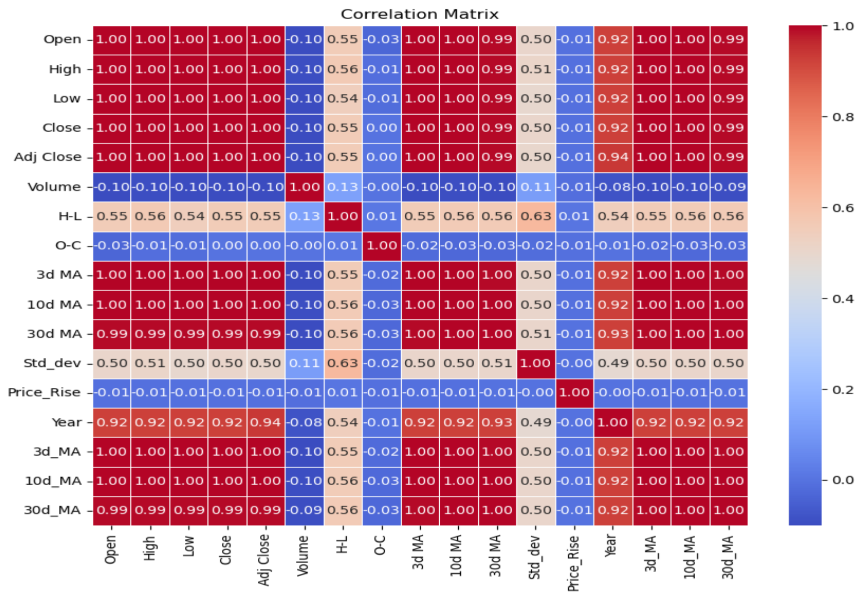
Figure 6: Descriptive Statistics



```
              Open         High          Low        Close    Adj Close  \
count  2415.000000  2415.000000  2415.000000  2415.000000  2415.000000
mean   2126.359090  2146.360566  2104.126642  2124.959420  2017.074690
std     916.931070   923.834060   908.730037   916.450666   940.225882
min    1017.000000  1039.000000  1000.250000  1017.349976   813.628784
25%    1262.712524  1274.000000  1250.500000  1261.474976  1110.760132
50%    1977.000000  1994.000000  1955.099976  1974.900024  1904.233765
75%    3154.425049  3175.750000  3123.199951  3148.125000  3053.474976
max    4045.500000  4045.500000  3981.000000  4019.100098  3880.605713

             Volume          H-L          O-C        3d MA       10d MA  \
count  2.415000e+03  2415.000000  2415.000000  2415.000000  2415.000000
mean   1.682678e+05    42.233925    -1.399670  2123.824075  2119.887967
std    2.300806e+05    27.337938    30.149435   915.805719   913.719399
min    1.028600e+04     5.425049  -145.150024  1029.616659  1056.090015
25%    7.092500e+04    22.500000   -16.337463  1260.445841  1260.672504
50%    1.068660e+05    35.750000    -1.750000  1973.016643  1967.674988
75%    1.722270e+05    54.199951    13.400024  3152.174967  3158.440988
max    4.395838e+06   314.599976   199.850098  3992.866699  3912.715039

             30d MA      Std_dev   Price_Rise         Year        3d_MA  \
count  2415.000000  2415.000000  2415.000000  2415.000000  2413.000000
mean   2109.352315    26.383063     0.514286  2018.533747  2124.711583
std     908.583689    19.982711     0.499899     2.829233   915.665900
min    1073.279997     0.799146     0.000000  2014.000000  1029.616659
25%    1264.285834    12.374988     0.000000  2016.000000  1260.475016
50%    1946.314990    20.402637     1.000000  2019.000000  1973.933300
75%    3150.639185    34.978660     1.000000  2021.000000  3152.199951
max    3827.155013   151.604241     1.000000  2023.000000  3992.866699

             10d_MA       30d_MA
count  2406.000000  2386.000000
mean   2123.793546  2121.630483
std     913.188167   907.194278
min    1056.090015  1073.279997
25%    1262.584364  1267.567706
50%    1971.134991  1958.494997
75%    3161.175012  3152.315428
max    3912.715039  3827.155013
<class 'pandas.core.frame.DataFrame'>
Index: 2415 entries, 29 to 2466
Data columns (total 18 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         2415 non-null   object
 1   Open         2415 non-null   float64
 2   High         2415 non-null   float64
 3   Low          2415 non-null   float64
 4   Close        2415 non-null   float64
 5   Adj Close    2415 non-null   float64
 6   Volume       2415 non-null   float64
 7   H-L          2415 non-null   float64
 8   O-C          2415 non-null   float64
 9   3d MA        2415 non-null   float64
 10  10d MA       2415 non-null   float64
 11  30d MA       2415 non-null   float64
 12  Std_dev      2415 non-null   float64
 13  Price_Rise   2415 non-null   int32
 14  Year         2415 non-null   int32
 15  3d_MA        2413 non-null   float64
 16  10d_MA       2406 non-null   float64
 17  30d_MA       2386 non-null   float64
dtypes: float64(15), int32(2), object(1)
memory usage: 339.6+ KB
None
```

From the descriptive statistics tables (Fig.6) a total of 2415 rows are observed due to data reduction techniques (previously 2444 rows). The TCS stock dataset, after data refinement, shows a notable growth in mean closing price from 2014 to 2023. Price volatility is evident from the standard deviation, and the nearly balanced 'Price_Rise' indicates an equal distribution of gain and loss days. Moving averages smooth out short-term fluctuations, underlining the stock's long-term upward trajectory.

Figure 7: correlation matrix of data variables



The correlation matrix for TCS shows a strong positive correlation among the stock's Open, High, Low, Close, and Adjusted Close prices, indicative of their typical co-movement. A notable negative correlation between 'Year' and price variables may point to a long-term downtrend (Fig.7). 'Volume' and 'Price_Rise' display weak correlations with price metrics,

4.MACHINE LEARNING CLASSIFICATION

Two machine learning models, Logistic Regression and Extra Trees Classifier, were assessed for their efficacy in predicting stock price movements using historical data from TCS. Features ranging from price differentials (H-L) to volatility (Std_dev) were utilized.

The dataset was split into training and testing sets, with a 20% test size, and feature scaling was applied using StandardScaler for enhanced model performance evaluation. This analysis aimed to determine the most effective model for predicting stock price trends based on historical data.

A.LOGISTIC REGRESSION

Logistic Regression, a statistical method for binary classification, applies a logistic function to estimate the probability of an event based on inputs. (David G. Kleinbaum , Mitchel Klein, 2010) When predicting stock movements, such as for TCS, it evaluates linear relationships between features like price range and volatility to forecast price trends, as detailed in section 6.

Formula:

$$P(y=1|x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$$

Where:

- $P(y=1|x)$ is the probability of the positive class.
- β_0 is the intercept.
- β_1 is the coefficient for the feature x.

B. EXTRA TREES

Extra Trees, an ensemble of decision trees, randomizes split points and features to build diverse trees, reducing overfitting and increasing robustness to noise. This method, faster than many tree-based algorithms, excels in both classification and regression tasks (Yazan Ahmad Alsariera; Victor Elijah Adeyemo; Abdullateef Oluwagbemiga Balogun, 2020), as highlighted in section 6.

5.CROSS VALIDATION ACCURACY ANALYSIS

Cross-validation rigorously tests a model's ability to generalize by rotating through different training and testing data subsets, thus ensuring the model's reliability, and guiding the optimization of its settings.

Formula:

$$\text{Mean Accuracy} = \frac{1}{k}\sum_{i=1}^{k} \text{Accuracy}_i$$

where Accuracy_i is the accuracy of the model on the i-th fold.

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

A. CROSS VALIDATION - LOGISTIC REGRESSION

The logistic regression model, as evaluated through 5-fold cross-validation, exhibits an average accuracy of 49%. This performance level indicates that the model performs marginally better than random chance when generalized to the entire dataset, and the low standard deviation signifies stable performance across various data splits.

B. CROSS VALIDATION - EXTRA TREES

The Extra Trees Classifier, upon evaluation with cross-validation, yielded an average accuracy of 48%. This result suggests that the model's predictive capability is nearly equivalent to random guessing. Its performance indicates low variability, suggesting consistent model behaviour across various subsets of the dataset.

6.EVALUATION OF CLASSIFIERS RESULTS

The Logistic Regression model, with an overall accuracy of 49%, showed proficiency in predicting stock price increases with a 49% precision and 88% recall, but it was less effective at predicting when prices would not rise, with a precision of 46% and a recall of only 10%. The model's f1-scores reflect this imbalance, suggesting it's better at forecasting price rises than declines.

	precision	recall	f1-score	support
0	0.46	0.10	0.17	244
1	0.49	0.88	0.63	239
accuracy			0.49	483
macro avg	0.48	0.49	0.40	483
weighted avg	0.48	0.49	0.40	483

he Extra Trees Classifier achieved a balanced performance with an overall accuracy of 48%, showing near-equal precision in predicting both 'no rise' and 'price rise' outcomes at around 49% and 48% respectively. It was better at identifying 'price rise' cases with a 59% success rate, compared to 38% for 'no rise' scenarios. The f1-scores suggest moderate effectiveness across predictions, highlighting its consistent but modest ability to forecast stock movements.

Classification Report:

	precision	recall	f1-score	support
0	0.49	0.38	0.42	244
1	0.48	0.59	0.53	239
accuracy			0.48	483
macro avg	0.48	0.49	0.48	483
weighted avg	0.48	0.48	0.48	483

When applied to a 20% dataset split, both classifiers showed similar performance. Logistic regression excelled in predicting price rises, indicated by higher recall and f1-score for this outcome, while the Extra Trees Classifier delivered more balanced results across scenarios, with comparable precision and recall for both 'no rise' and 'rise' predictions.

```
In [46]:
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer, accuracy_score

# Perform cross-validation
accuracy_scores = cross_val_score(model_lr, X, Y, cv=5, scoring=make_scorer(accuracy_score))

# Print mean and standard deviation of accuracy
print(f"Mean Accuracy: {accuracy_scores.mean():.2f}")
print(f"Standard Deviation: {accuracy_scores.std():.2f}")
```

```
Mean Accuracy: 0.49
Standard Deviation: 0.02
```

```
In [47]:
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer, accuracy_score

# Create an Extra Trees Classifier model
model_et = ExtraTreesClassifier()

# Perform cross-validation
accuracy_scores = cross_val_score(model_et, X, Y, cv=5, scoring=make_scorer(accuracy_score))

# Print mean and standard deviation of accuracy
print(f"Mean Accuracy: {accuracy_scores.mean():.2f}")
print(f"Standard Deviation Accuracy: {accuracy_scores.std():.2f}")
```

```
Mean Accuracy: 0.50
Standard Deviation Accuracy: 0.02
```

7.PREDICTION OF PRICE USING X_ TEST DATA: EXTRA TREE

The choice to proceed with the Extra Trees model, as discussed in section 5, is justified by its superior performance on the split dataset, where it surpassed the logistic regression model in effectiveness. (by Matthew F. Dixon; Igor Halperin; Paul A. Biloko, 2020)

A. CLASSIFICATION REPORT

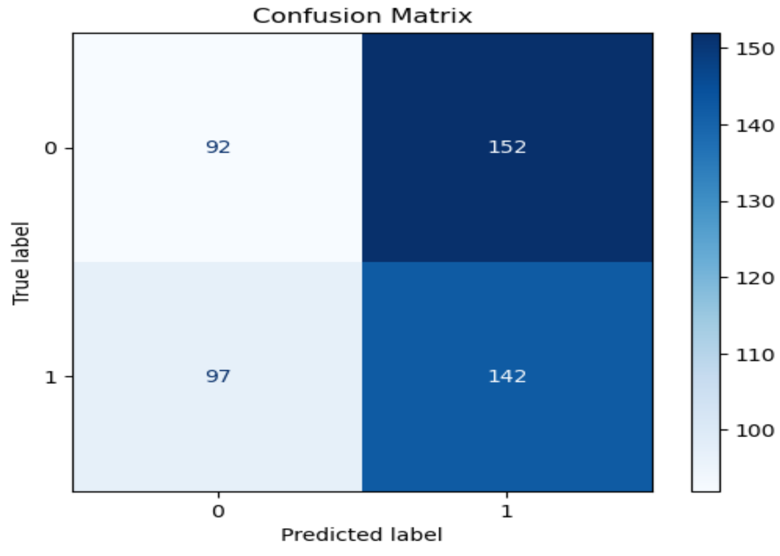
The classification report evaluates model performance on binary classification through precision (accuracy of class predictions), recall (proportion of actual class instances correctly identified), F1-score (balance between precision and recall), and support (instances per class), offering a comprehensive view of effectiveness in classifying instances. (Buchanan, Bonnie G, 2019)

```
Classification Report:
              precision    recall  f1-score   support

           0       0.49      0.38      0.42       244
           1       0.48      0.59      0.53       239

    accuracy                           0.48       483
   macro avg       0.48      0.49      0.48       483
weighted avg       0.48      0.48      0.48       483
```

B. CONFUSION MATRIX

The confusion matrix shows the model's prediction accuracy: 92 correct for 'no price rise' and 142 for 'price rise', but with 152 and 97 mispredictions for each category, respectively. It highlights the model's stronger performance in identifying 'price rise' scenarios. (Fig.8).

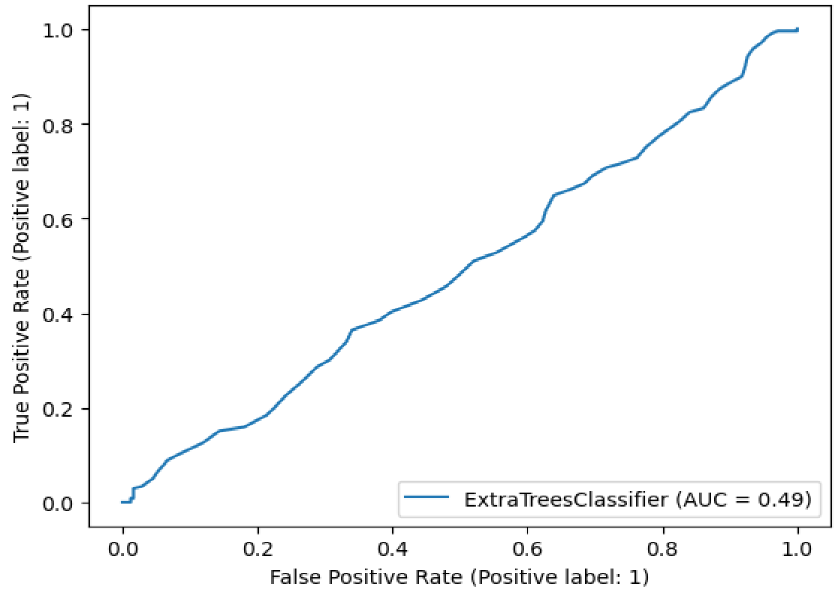
Figure 8: Confusion Matrix of model (Extra Tree model)



C.ROC CURVE

The ROC curve for the Extra Trees Classifier, with an AUC of 0.49, demonstrates the model's balance between sensitivity and specificity, suggesting its performance in predicting 'price rise' is like random guessing. (Fig.9).

Figure 9: ROC Curve of model (Extra Tree classification model)



8. MARKET AND STRATEGY RETURNS

"Market Returns" represent hypothetical earnings from holding a stock, while "Strategy Returns" are based on trading strategies informed by machine learning predictions, such as those from the Extra Trees model. In our analysis, model forecasts are added to a 'Y_predicted' column to calculate these returns. "Tomorrows Returns" tracks daily logarithmic returns, adjusting for the natural fluctuation between consecutive days' closing prices, ensuring alignment for strategic analysis. (Yumei Yao, 2024)

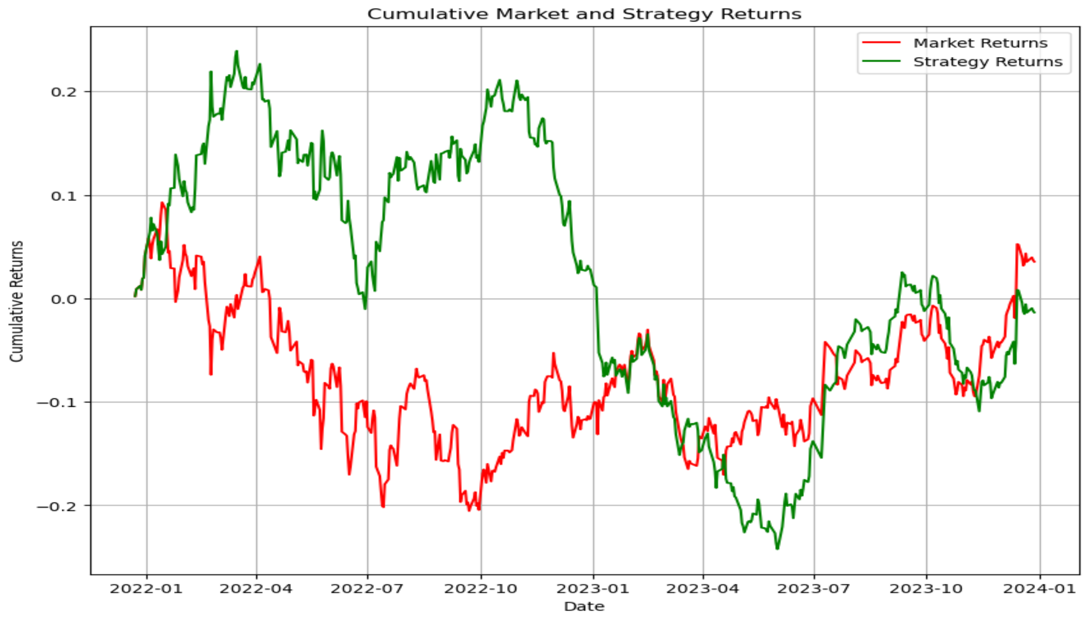
$$\text{Market Return} = \frac{\text{Closing Price}_{\text{today}} - \text{Closing Price}_{\text{yesterday}}}{\text{Closing Price}_{\text{yesterday}}}$$

Strategy returns are derived by adopting long positions for true 'Y_predicted' and short positions for false. The 'Strategy Returns' column, initialized at zero, records 'Tomorrows Returns'—positively for predicted price rises and negatively for predicted drops, simulating buying or selling the stock accordingly. (Yumei Yao, 2024)

9.CUMMULATIVE RETURNS ANALYSIS

Cumulative return analysis evaluates investment strategy effectiveness, asset performance, and decision profitability, guiding long-term investment insights and strategy adjustments. It underpins trading strategy assessments, with market and strategy cumulative returns calculated using the cumsum() function.

Figure 10: cumulative market and strategy return based on Y_ Prediction



The graph tracks cumulative returns from early 2022 to early 2024, contrasting market performance with a predictive strategy's returns. Initially, market returns outshine the strategy, but from April 2022 onwards, the two approaches' performances begin to align and occasionally switch leads. The strategy notably outperforms the market amid volatility from late 2022 to 2023, suggesting its potential superiority during unstable market conditions. By the period's end, the strategy notably rebounds above market returns, indicating its potential advantage over standard market-following approaches in turbulent times.

10.INTERPRETATION AND DISCUSSION

Analysing TCS stock prices using the Extra Trees Classifier revealed a model accuracy of around 48%, which suggests that while the model has some predictive ability, it's not highly reliable for forecasting stock price movements. This outcome aligns with the efficient market hypothesis, which holds that stocks reflect all available information, making prediction challenging. The model's performance underscores the complex and often unpredictable nature of financial markets, where numerous variables can influence stock prices beyond historical data.

11.CONCLUSION

The analysis of TCS stock with the Extra Trees Classifier indicates that machine learning can capture certain stock market trends to an extent, but the model's predictive power is limited. The performance is reflective of market complexities and unpredictability, highlighting the difficulty in forecasting prices with high accuracy due to unexpected market events and the challenge of overfitting. Although machine learning adds analytical value, it should be complemented by traditional financial analysis due to the volatile nature of the markets and the limitations of existing models. For future advancements, research should aim at enhancing the adaptability and robustness of predictive models under varied market conditions.

12.REFERENCES

David G. Kleinbaum , Mitchel Klein. (2010). *Logistic regression.* USA: Springer New York, NY.

Buchanan, Bonnie G. (2019). *Artificial intelligence in finance.* Finland: The Alan Turing Institute.

by Matthew F. Dixon; Igor Halperin; Paul A. Biloko. (2020). *Machine learning in finance: from theory to practice.* UK: Springer International Publishing.

IBM. (2024, 03 24). *EDA*. Retrieved from IBM: https://www.ibm.com/topics/exploratory-data-analysis

Tata Company. (2024, march 24). *TATA Companies*. Retrieved from Tata Consultancy Services: https://www.tata.com/business/tcs

Yazan Ahmad Alsariera; Victor Elijah Adeyemo; Abdullateef Oluwagbemiga Balogun. (2020). *AI Meta-Learners and Extra-Trees Algorithm.* NY: IEEE.

Yumei Yao. (2024, March 24). Seminar 6. *AI& ML*. London, UK, UK: university of westminster.

13.APPENDICES

1.Imported Library

In [4]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [5]:
```
#Import libraries and metrics
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import accuracy_score,roc_curve, classification_report
from sklearn.metrics import confusion_matrix, make_scorer, roc_auc_score
from sklearn.metrics import RocCurveDisplay, ConfusionMatrixDisplay
```

2.Historical Data (TCS Uploaded) & Cleaned

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

In [13]:
```
import pandas as pd
file_path = 'TCS.BO.csv'
df = pd.read_csv(file_path)
print(df.head())
```

```
         Date         Open         High          Low        Close   Adj Close  \
0  2014-01-01  1090.000000  1092.500000  1075.525024  1076.849976  858.149536
1  2014-01-02  1080.000000  1093.574951  1075.000000  1081.150024  861.576538
2  2014-01-03  1087.500000  1114.125000  1075.500000  1111.000000  885.364075
3  2014-01-06  1113.000000  1121.824951  1098.500000  1119.800049  892.376953
4  2014-01-07  1119.150024  1127.400024  1100.300049  1104.099976  879.865173

     Volume
0   73300.0
1  117324.0
2  267204.0
3  145172.0
4  135104.0
```

In [19]:
```
df_cleaned = df.dropna()
print(df_cleaned)
```

```
            Date         Open         High          Low        Close  \
0     2014-01-01  1090.000000  1092.500000  1075.525024  1076.849976
1     2014-01-02  1080.000000  1093.574951  1075.000000  1081.150024
2     2014-01-03  1087.500000  1114.125000  1075.500000  1111.000000
3     2014-01-06  1113.000000  1121.824951  1098.500000  1119.800049
4     2014-01-07  1119.150024  1127.400024  1100.300049  1104.099976
...          ...          ...          ...          ...          ...
2462  2023-12-22  3796.300049  3845.949951  3762.699951  3825.300049
2463  2023-12-26  3780.100098  3833.850098  3780.100098  3794.600098
2464  2023-12-27  3795.550049  3818.000000  3768.100098  3810.800049
2465  2023-12-28  3822.100098  3838.250000  3793.750000  3801.050049
2466  2023-12-29  3797.850098  3822.949951  3766.050049  3794.949951

        Adj Close    Volume
0      858.149536   73300.0
1      861.576538  117324.0
2      885.364075  267204.0
3      892.376953  145172.0
4      879.865173  135104.0
...           ...       ...
2462  3825.300049  127163.0
2463  3794.600098   70216.0
2464  3810.800049   28290.0
2465  3801.050049   29256.0
2466  3794.949951  105711.0
```

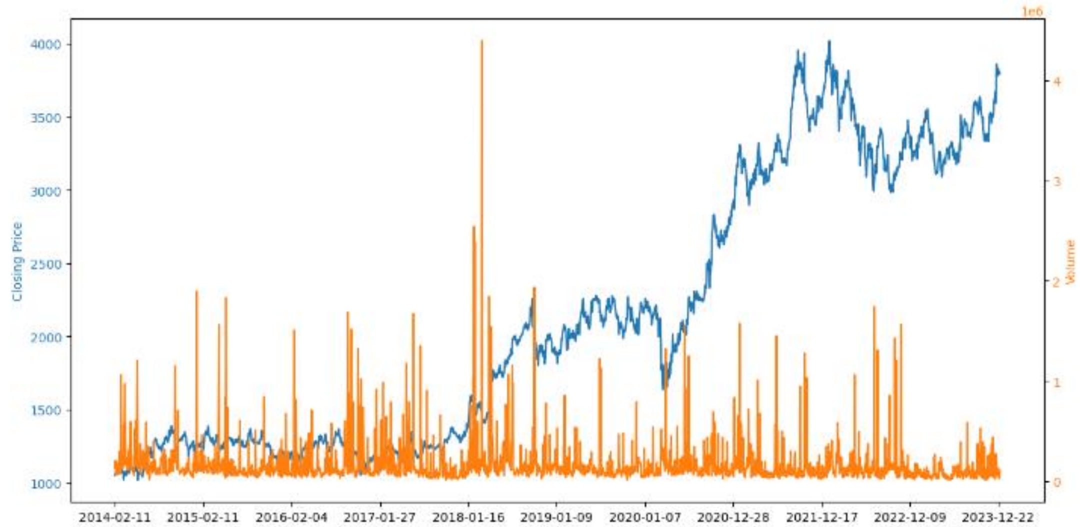
3.Feature Engineering

In [20]:
```python
# Calculate 'H-L' and 'O-C' differences
df_cleaned['H-L'] = df_cleaned['High'] - df_cleaned['Low']
df_cleaned['O-C'] = df_cleaned['Close'] - df_cleaned['Open']

# Calculate moving averages
df_cleaned['3d MA'] = df_cleaned['Close'].rolling(window=3).mean()
df_cleaned['10d MA'] = df_cleaned['Close'].rolling(window=10).mean()
df_cleaned['30d MA'] = df_cleaned['Close'].rolling(window=30).mean()

# Calculate standard deviation
df_cleaned['Std_dev'] = df_cleaned['Close'].rolling(window=5).std()

# Create 'Price_Rise' column
df_cleaned['Price_Rise'] = (df_cleaned['Close'].shift(-1) > df_cleaned['Close']).astype(int)

# Drop last row as it will have NaN values for 'Price_Rise'
df_cleaned = df_cleaned.dropna()

# Display the cleaned DataFrame
print(df_cleaned)
```

```
            Date         Open         High          Low        Close  \
29    2014-02-11  1047.500000  1063.000000  1047.500000  1050.500000
30    2014-02-12  1061.849976  1061.900024  1047.550049  1051.900024
31    2014-02-13  1054.500000  1069.500000  1049.500000  1066.875000
32    2014-02-14  1069.500000  1092.199951  1064.250000  1083.125000
33    2014-02-17  1084.500000  1093.500000  1074.775024  1084.300049
...          ...          ...          ...          ...          ...
2462  2023-12-22  3796.300049  3845.949951  3762.699951  3825.300049
2463  2023-12-26  3780.100098  3833.850098  3780.100098  3794.600098
2464  2023-12-27  3795.550049  3818.000000  3768.100098  3810.800049
2465  2023-12-28  3822.100098  3838.250000  3793.750000  3801.050049
2466  2023-12-29  3797.850098  3822.949951  3766.050049  3794.949951

        Adj Close    Volume        H-L        O-C        3d MA       10d MA  \
29     840.140564  202536.0  15.500000   3.000000  1056.341675  1085.967493
30     841.260193  126528.0  14.349975  -9.949952  1049.750000  1080.679993
31     853.236633  165496.0  20.000000  12.375000  1056.425008  1076.472498
32     866.232483  149938.0  27.949951  13.625000  1067.300008  1072.975000
33     867.172363  202532.0  18.724976  -0.199951  1078.100016  1071.572510
...           ...       ...        ...        ...          ...          ...
2462  3825.300049  127163.0  83.250000  29.000000  3798.366699  3749.985034
2463  3794.600098   70216.0  53.750000  14.500000  3803.116699  3765.225049
2464  3810.800049   28290.0  49.899902  15.250000  3810.233399  3779.100049
2465  3801.050049   29256.0  44.500000 -21.050049  3802.150065  3799.830054
2466  3794.949951  105711.0  56.899902  -2.900147  3802.266683  3812.665039

           30d MA    Std_dev  Price_Rise
29    1115.374162  21.631318           1
30    1114.542497  17.325573           1
31    1114.066663  10.983936           1
32    1113.137496  15.088916           1
33    1111.954163  16.271172           0
...           ...        ...         ...
2462  3577.036678  30.812797           0
```

4.EDA of created Features

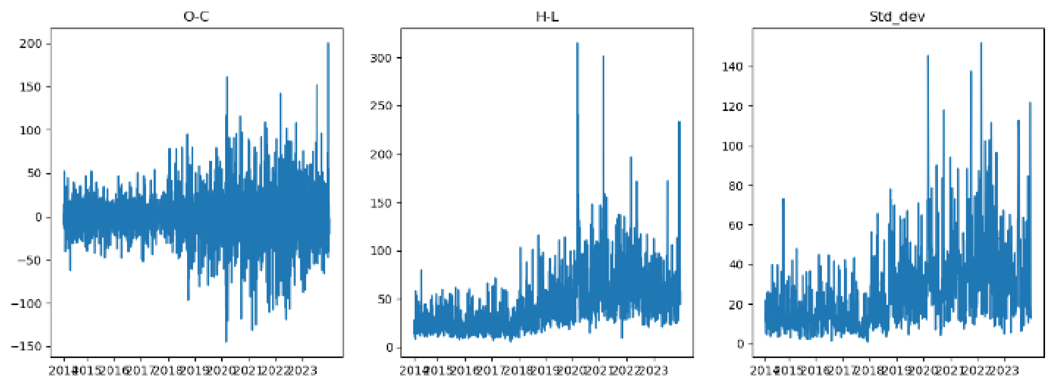
In [26]:
```python
import pandas as pd
import matplotlib.pyplot as plt

# Assuming your DataFrame is named df_cleaned

# Create a larger figure and axis objects
fig, ax1 = plt.subplots(figsize=(14, 7))

# Plot Closing Price on the first y-axis
color = 'tab:blue'
ax1.set_ylabel('Closing Price', color=color)
ax1.plot(df_cleaned['Date'], df_cleaned['Close'], label='Close', color=color)
ax1.tick_params(axis='y', labelcolor=color)

# Create a second y-axis sharing the same x-axis
ax2 = ax1.twinx()

# Plot Volume on the second y-axis
color = 'tab:orange'
ax2.set_ylabel('Volume', color=color)
ax2.plot(df_cleaned['Date'], df_cleaned['Volume'], label='Volume', color=color)
ax2.tick_params(axis='y', labelcolor=color)

# Customize x-axis ticks to show only years
years = pd.to_datetime(df_cleaned['Date']).dt.year.unique()
plt.xticks(ticks=df_cleaned['Date'][::len(df_cleaned)//len(years)], labels=years, rotation=45)

# Add title and legend
plt.title('Closing Price and Volume Relationship')
fig.tight_layout()
plt.show()
```



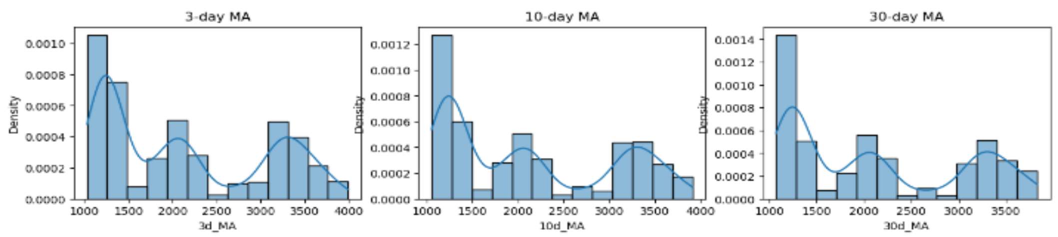
In [33]:

```python
import matplotlib.pyplot as plt

# Assuming your DataFrame is named df_cleaned
# Make sure you have already calculated the moving averages

# Extract years from the 'Date' column
years = pd.to_datetime(df_cleaned['Date']).dt.year.unique()

# Get the indices corresponding to the beginning of each year
year_indices = []
for year in years:
    year_indices.append(df_cleaned[df_cleaned['Date'].str.contains(str(year))].index[0])

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

plt.plot(df_cleaned['3d_MA'], label='3-day MA')
plt.plot(df_cleaned['10d_MA'], label='10-day MA')
plt.plot(df_cleaned['30d_MA'], label='30-day MA')

# Set x-axis ticks to correspond to the indices where the years change
plt.xticks(ticks=year_indices, labels=years)

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

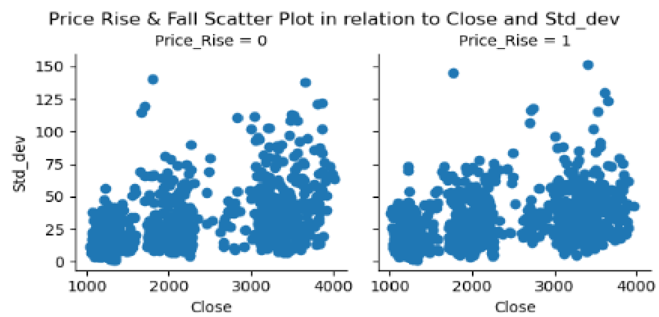
```python
In [35]: import matplotlib.pyplot as plt


         # Extract years from the 'Date' column
         years = pd.to_datetime(df_cleaned['Date']).dt.year.unique()

         fig, axes = plt.subplots(1, 3, figsize=(15, 5))

         for ax in axes:
             ax.set_xticks(year_indices)
             ax.set_xticklabels(years)

         axes[0].plot(df_cleaned['O-C'])
         axes[0].set_title('O-C')

         axes[1].plot(df_cleaned['H-L'])
         axes[1].set_title('H-L')

         axes[2].plot(df_cleaned['Std_dev'])
         axes[2].set_title('Std_dev')

         plt.show()
```




```python
In [37]: import seaborn as sns
         import matplotlib.pyplot as plt


         fig, axes = plt.subplots(1, 3, figsize=(15,3))  # 1 row, 3 columns

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

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

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

         plt.show()
```

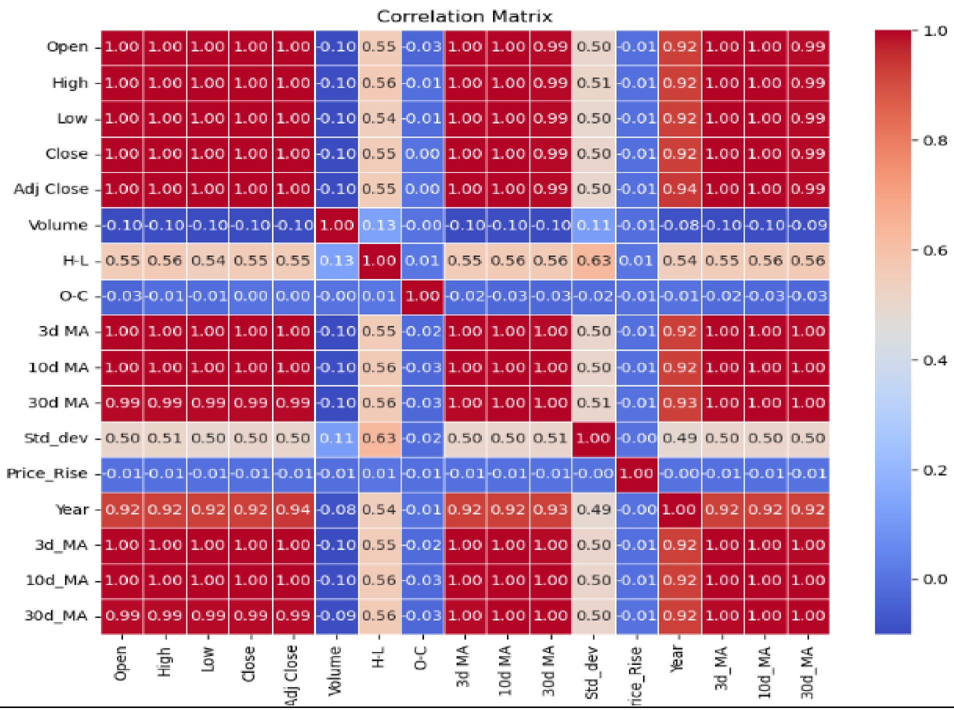


```python
In [38]:  import seaborn as sns
           import matplotlib.pyplot as plt

           # Price Rise & Fall Count Plot
           chart_count = sns.FacetGrid(df_cleaned, col='Price_Rise')
           chart_count.map(sns.histplot, 'Close')
           plt.suptitle('Price Rise & Fall Count Plot in relation to Close', y=1.02)
           plt.show()

           # Price Rise & Fall Scatter Plot
           chart_scatter = sns.FacetGrid(df_cleaned, col='Price_Rise')
           chart_scatter.map(plt.scatter, 'Close', 'Std_dev')
           plt.suptitle('Price Rise & Fall Scatter Plot in relation to Close and Std_dev', y=1.02)
           plt.show()
```



5.Correlation Matrix

```python
In [41]:  import seaborn as sns
           import matplotlib.pyplot as plt

           # Calculate correlation matrix excluding non-numeric columns
           corr_matrix = df_cleaned.corr(method='pearson', min_periods=1, numeric_only=True)

           # Plot correlation matrix
           plt.figure(figsize=(10, 8))
           sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
           plt.title('Correlation Matrix')
           plt.show()
```



6.Train and Test Data Splitting

```
In [43]:  from sklearn.model_selection import train_test_split

          # 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)

          # Display the testing set features
          print(X_test)

                 H-L        O-C       3d MA      10d MA      30d MA    Std_dev
          1967  40.949951  22.300049  3633.750000  3606.595044  3556.254997  40.325331
          1968  58.599853  -5.750000  3654.350016  3609.965039  3562.326660  45.779676
          1969  46.850098  25.550049  3676.200033  3618.510034  3568.028328  34.340124
          1970  30.449951  -7.699951  3691.033366  3626.960034  3573.141667  30.188729
          1971  35.050049  -6.850098  3698.666667  3639.315015  3577.741667  18.588547
          ...        ...        ...         ...         ...         ...        ...
          2462  83.250000  29.000000  3798.366699  3749.985034  3577.036678  30.812797
          2463  53.750000  14.500000  3803.116699  3765.225049  3591.915015  18.113791
          2464  49.899902  15.250000  3810.233399  3779.100049  3607.848348  17.911341
          2465  44.500000 -21.050049  3802.150065  3799.830054  3623.493351  14.216267
          2466  56.899902  -2.900147  3802.266683  3812.665039  3636.681681  12.941387

          [483 rows x 6 columns]
```

7.Logistic regression

```
In [45]:  from sklearn.linear_model import LogisticRegression
          from sklearn.metrics import classification_report

          # Create a Logistic Regression model
          model_lr = LogisticRegression(random_state=101)

          # Train the model
          model_lr.fit(X_train, Y_train)

          # Predict on the test set
          Y_pred = model_lr.predict(X_test)

          # Print the classification report
          print(classification_report(Y_test, Y_pred))

                        precision    recall  f1-score   support

                     0       0.46      0.10      0.17       244
                     1       0.49      0.88      0.63       239

              accuracy                           0.49       483
             macro avg       0.48      0.49      0.40       483
          weighted avg       0.48      0.49      0.40       483
```

8.Extra Tree classification

```
In [48]:  from sklearn.ensemble import ExtraTreesClassifier
          from sklearn.metrics import classification_report

          # Create an Extra Trees Classifier model
          model_et = ExtraTreesClassifier(random_state=101)

          # Train the model
          model_et.fit(X_train, Y_train)

          # Predict on the test set
          Y_pred = model_et.predict(X_test)

          # Print classification report
          classification_rep = classification_report(Y_test, Y_pred)
          print("Classification Report:\n", classification_rep)

          Classification Report:
                         precision    recall  f1-score   support

                     0       0.49      0.38      0.42       244
                     1       0.48      0.59      0.53       239

              accuracy                           0.48       483
             macro avg       0.48      0.49      0.48       483
          weighted avg       0.48      0.48      0.48       483
```

9.CrossValidation

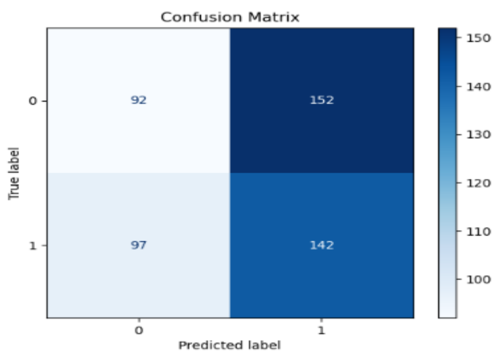
```
In [46]:  from sklearn.model_selection import cross_val_score
          from sklearn.metrics import make_scorer, accuracy_score

          # Perform cross-validation
          accuracy_scores = cross_val_score(model_lr, X, Y, cv=5, scoring=make_scorer(accuracy_score))

          # Print mean and standard deviation of accuracy
          print(f"Mean Accuracy: {accuracy_scores.mean():.2f}")
          print(f"Standard Deviation: {accuracy_scores.std():.2f}")

          Mean Accuracy: 0.49
          Standard Deviation: 0.02

In [47]:  from sklearn.ensemble import ExtraTreesClassifier
          from sklearn.model_selection import cross_val_score
          from sklearn.metrics import make_scorer, accuracy_score

          # Create an Extra Trees Classifier model
          model_et = ExtraTreesClassifier()

          # Perform cross-validation
          accuracy_scores = cross_val_score(model_et, X, Y, cv=5, scoring=make_scorer(accuracy_score))

          # Print mean and standard deviation of accuracy
          print(f"Mean Accuracy: {accuracy_scores.mean():.2f}")
          print(f"Standard Deviation Accuracy: {accuracy_scores.std():.2f}")

          Mean Accuracy: 0.50
          Standard Deviation Accuracy: 0.02
```

10.Prediction of Price Rise Using extra Tree on X test Data

```
In [48]:  from sklearn.ensemble import ExtraTreesClassifier
          from sklearn.metrics import classification_report

          # Create an Extra Trees Classifier model
          model_et = ExtraTreesClassifier(random_state=101)

          # Train the model
          model_et.fit(X_train, Y_train)

          # Predict on the test set
          Y_pred = model_et.predict(X_test)

          # Print classification report
          classification_rep = classification_report(Y_test, Y_pred)
          print("Classification Report:\n", classification_rep)

          Classification Report:
                         precision    recall  f1-score   support

                     0       0.49      0.38      0.42       244
                     1       0.48      0.59      0.53       239

              accuracy                           0.48       483
             macro avg       0.48      0.49      0.48       483
          weighted avg       0.48      0.48      0.48       483
```

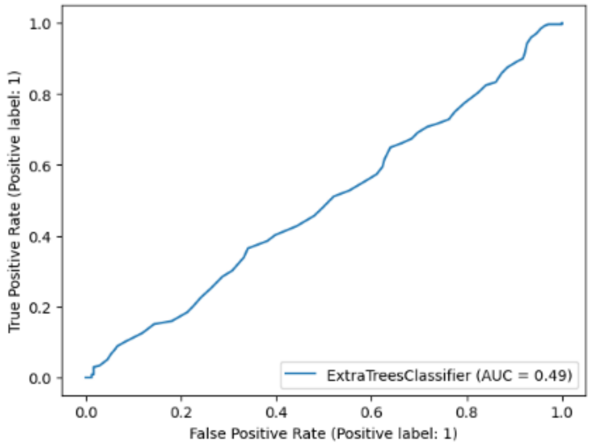
11.Confusion Matrix

```
In [49]:  from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

          # Calculate the confusion matrix
          conf_matrix = confusion_matrix(Y_test, Y_pred)

          # Display the confusion matrix
          disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=model_et.classes_)
          disp.plot(cmap='Blues')
          plt.title('Confusion Matrix')
          plt.show()
```



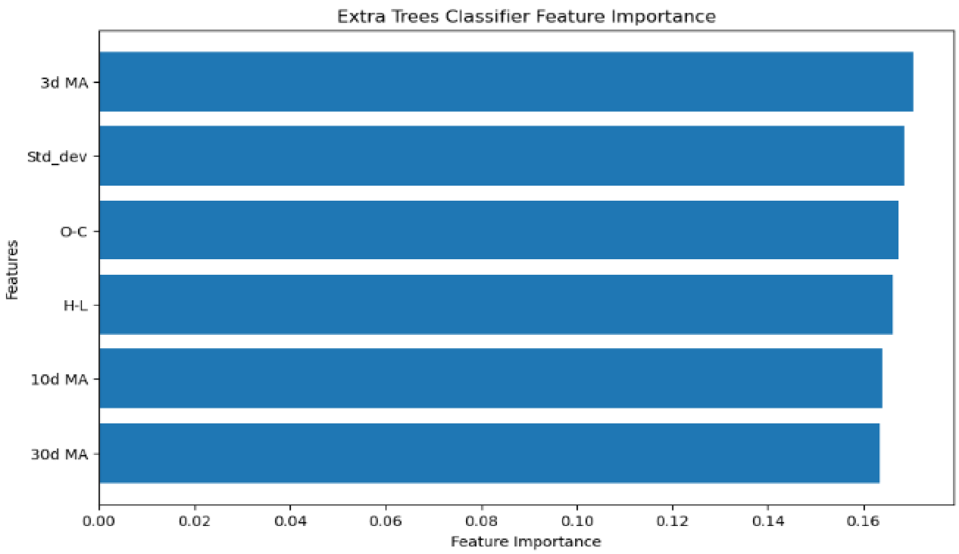
12.ROC Curve of the model (Extra Tree)

```
In [50]: from sklearn.metrics import RocCurveDisplay

         # Plot ROC curve
         roc_disp = RocCurveDisplay.from_estimator(model_et, X_test, Y_test)
         roc_disp.plot()
         plt.title('ROC Curve')
         plt.show()
```



13.Importance of Classifier in model

```
In [52]: import numpy as np
         import matplotlib.pyplot as plt

         # Importance of classifiers
         feature_names = X.columns
         importance = model_et.feature_importances_
         indices = np.argsort(importance)
         range1 = range(len(importance))

         plt.figure(figsize=(10, 6))
         plt.title("Extra Trees Classifier Feature Importance")
         plt.barh(range1, importance[indices])
         plt.yticks(range1, feature_names[indices])
         plt.xlabel('Feature Importance')
         plt.ylabel('Features')
         plt.show()
```



14.Market and Strategic Return

In [55]:

```python
# Create a new column for market returns
df_cleaned['Market Returns'] = 0.0

# Calculate market returns
df_cleaned['Market Returns'] = np.log(df_cleaned['Close'] / df_cleaned['Close'].shift(1))

# Shift market returns by one day to represent tomorrow's returns
df_cleaned['Market Returns'] = df_cleaned['Market Returns'].shift(-1)

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

# Display the DataFrame
df_cleaned
```

Out[55]:

	Date	Open	High	Low	Close	Adj Close	Volume	H-L	O-C	3d MA	10d MA	30d MA	
1967	2021-12-23	3640.500000	3671.000000	3630.050049	3662.800049	3536.583496	62978.0	40.949951	22.300049	3633.750000	3606.595044	3556.254997	40
1968	2021-12-24	3676.000000	3703.899902	3645.300049	3670.250000	3543.776855	141802.0	58.599853	-5.750000	3654.350016	3609.965039	3562.326660	45
1969	2021-12-27	3670.000000	3700.000000	3653.149902	3695.550049	3568.204834	37807.0	46.850098	25.550049	3676.200033	3618.510034	3568.028328	34
1970	2021-12-28	3715.000000	3724.500000	3694.050049	3707.300049	3579.550049	34438.0	30.449951	-7.699951	3691.033366	3626.960034	3573.141667	30
1971	2021-12-29	3700.000000	3719.449951	3684.399902	3693.149902	3565.887207	75965.0	35.050049	-6.850098	3698.666667	3639.315015	3577.741667	18
...	...	...	...	...	...	...	...	...	...	...	...	...	
2461	2023-12-21	3734.399902	3805.000000	3734.399902	3789.449951	3789.449951	90556.0	70.600098	55.050049	3793.966716	3730.095020	3562.218343	37
2462	2023-12-22	3796.300049	3845.949951	3762.699951	3825.300049	3825.300049	127163.0	83.250000	29.000000	3798.366699	3749.985034	3577.036678	30
2463	2023-12-26	3780.100098	3833.850098	3780.100098	3794.600098	3794.600098	70216.0	53.750000	14.500000	3803.116699	3765.225049	3591.915015	18
2464	2023-12-27	3795.550049	3818.000000	3768.100098	3810.800049	3810.800049	28290.0	49.899902	15.250000	3810.233399	3779.100049	3607.848348	17
2465	2023-12-28	3822.100098	3838.250000	3793.750000	3801.050049	3801.050049	29256.0	44.500000	-21.050049	3802.150065	3799.830054	3623.493351	14

482 rows × 20 columns

In [56]:

```python
# Create a new column for strategy returns
df_cleaned['Strategy Returns'] = 0.0

# Compute strategy returns based on Y_pred
df_cleaned['Strategy Returns'] = np.where(df_cleaned['Y_pred'] == True,
                                          df_cleaned['Market Returns'],
                                          -df_cleaned['Market Returns'])

# Display the DataFrame
df_cleaned
```

Out[56]:

	Date	Open	High	Low	Close	Adj Close	Volume	H-L	O-C	3d MA	...	30d MA	Std_dev	F
1967	2021-12-23	3640.500000	3671.000000	3630.050049	3662.800049	3536.583496	62978.0	40.949951	22.300049	3633.750000	...	3556.254997	40.325331	
1968	2021-12-24	3676.000000	3703.899902	3645.300049	3670.250000	3543.776855	141802.0	58.599853	-5.750000	3654.350016	...	3562.326660	45.779676	
1969	2021-12-27	3670.000000	3700.000000	3653.149902	3695.550049	3568.204834	37807.0	46.850098	25.550049	3676.200033	...	3568.028328	34.340124	
1970	2021-12-28	3715.000000	3724.500000	3694.050049	3707.300049	3579.550049	34438.0	30.449951	-7.699951	3691.033366	...	3573.141667	30.188729	
1971	2021-12-29	3700.000000	3719.449951	3684.399902	3693.149902	3565.887207	75965.0	35.050049	-6.850098	3698.666667	...	3577.741667	18.588547	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2461	2023-12-21	3734.399902	3805.000000	3734.399902	3789.449951	3789.449951	90556.0	70.600098	55.050049	3793.966716	...	3562.218343	37.469436	
2462	2023-12-22	3796.300049	3845.949951	3762.699951	3825.300049	3825.300049	127163.0	83.250000	29.000000	3798.366699	...	3577.036678	30.812797	
2463	2023-12-26	3780.100098	3833.850098	3780.100098	3794.600098	3794.600098	70216.0	53.750000	14.500000	3803.116699	...	3591.915015	18.113791	
2464	2023-12-27	3795.550049	3818.000000	3768.100098	3810.800049	3810.800049	28290.0	49.899902	15.250000	3810.233399	...	3607.848348	17.911341	
2465	2023-12-28	3822.100098	3838.250000	3793.750000	3801.050049	3801.050049	29256.0	44.500000	-21.050049	3802.150065	...	3623.493351	14.216267	

482 rows × 21 columns

15.cummulative market and strategic return

```
In [57]:  # Compute cumulative market returns
          df_cleaned['Cumulative Market Returns'] = np.cumsum(df_cleaned['Market Returns'])

          # Compute cumulative strategy returns
          df_cleaned['Cumulative Strategy Returns'] = np.cumsum(df_cleaned['Strategy Returns'])

          # Display the DataFrame
          df_cleaned
```

Out[57]:

	Date	Open	High	Low	Close	Adj Close	Volume	H-L	O-C	3d MA	...	Price_Rise	Year	3c
1967	2021-12-23	3640.500000	3671.000000	3630.050049	3662.800049	3536.583496	62978.0	40.949951	22.300049	3633.750000	...	1	2021	3633.75
1968	2021-12-24	3676.000000	3703.899902	3645.300049	3670.250000	3543.776855	141802.0	58.599853	-5.750000	3654.350016	...	1	2021	3654.35
1969	2021-12-27	3670.000000	3700.000000	3653.149902	3695.550049	3568.204834	37807.0	46.850098	25.550049	3676.200033	...	1	2021	3676.20
1970	2021-12-28	3715.000000	3724.500000	3694.050049	3707.300049	3579.550049	34438.0	30.449951	-7.699951	3691.033366	...	0	2021	3691.03
1971	2021-12-29	3700.000000	3719.449951	3684.399902	3693.149902	3565.887207	75965.0	35.050049	-6.850098	3698.666667	...	1	2021	3698.66
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2461	2023-12-21	3734.399902	3805.000000	3734.399902	3789.449951	3789.449951	90556.0	70.600098	55.050049	3793.966716	...	1	2023	3793.96
2462	2023-12-22	3796.300049	3845.949951	3762.699951	3825.300049	3825.300049	127163.0	83.250000	29.000000	3798.366699	...	0	2023	3798.36
2463	2023-12-26	3780.100098	3833.850098	3780.100098	3794.600098	3794.600098	70216.0	53.750000	14.500000	3803.116699	...	1	2023	3803.11
2464	2023-12-27	3795.550049	3818.000000	3768.100098	3810.800049	3810.800049	28290.0	49.899902	15.250000	3810.233399	...	0	2023	3810.23
2465	2023-12-28	3822.100098	3838.250000	3793.750000	3801.050049	3801.050049	29256.0	44.500000	-21.050049	3802.150065	...	0	2023	3802.15

482 rows × 23 columns

```
In [73]:  # Convert 'Date' column to datetime format
          df_cleaned['Date'] = pd.to_datetime(df_cleaned['Date'])

          # Plot the cumulative market and strategy returns
          plt.figure(figsize=(10, 8))

          plt.plot(df_cleaned['Date'], df_cleaned['Cumulative Market Returns'], color='r', label='Market Returns')
          plt.plot(df_cleaned['Date'], df_cleaned['Cumulative Strategy Returns'], color='g', label='Strategy Returns')

          plt.title('Cumulative Market and Strategy Returns')
          plt.xlabel('Date')
          plt.ylabel('Cumulative Returns')
          plt.legend()
          plt.grid(True)

          plt.show()
```

GitHub Link (Python codes): https://github.com/xxsarikapatel/AI-ML-1Coursework/blob/main/AI%26%20ML%20CW.ipynb