STOCK PRICE ANALYSIS

[Presented by- GROUP 5: PROCHETA ROY, PRATYUSH BHATTACHARYA, RICHICK BASAK, ARCKODIPTA BHATTACHARYA, M.PRIYADARSHINI MANICKAM, NALLAPPA KODIVENDLA, SKHANDA KUMAR]

The Python script illustrates the process of analyzing Tata Motors' stock data through libraries like Pandas, Matplotlib, and Scikit-Learn. It constructs a RandomForestClassifier model to predict future stock performance. The script preprocesses data, generates visualizations, and evaluates the model's accuracy. Furthermore, it implements backtesting, a technique to assess models' performance over time. The script's comprehensive approach demonstrates the steps involved in data analysis, model building, and performance assessment within the context of stock prediction for Tata Motors.

Importing Necessary Libraries:

In the process of analyzing Tata Motors' stock price using Python, we are leveraging a set of essential libraries to facilitate different aspects of the analysis. These libraries enable us to efficiently manipulate, visualize, and process stock price data.

1. ***NumPy*** : NumPy is a fundamental library for numerical computations in Python. It provides support for working with arrays and matrices, allowing us to perform various mathematical and statistical operations efficiently. In our analysis, NumPy can help with handling and processing numerical data related to Tata Motors' stock price.

2. ***Pandas*** : Pandas is a versatile data manipulation library that offers powerful data structures, such as DataFrames, to organize and manipulate structured data. By using Pandas, we can efficiently load and manage historical stock price data, making it easier to perform data transformations and calculations.

3. ***Matplotlib.pyplot*** : Matplotlib is a widely used plotting library that provides tools for creating various types of visualizations, such as line plots, bar charts, and

scatter plots. The pyplot module of Matplotlib enables you to create informative visual representations of Tata Motors' stock price trends over time.

6. ***Warnings*** : The 'warnings' library is used to manage the display of warnings generated by Python. This can be particularly useful when dealing with potential data inconsistencies or changes in library behavior during our analysis.

7. ***yfinance*** : The 'yfinance' library is employed to fetch stock price data from Yahoo Finance. This library provides a convenient way to access historical and real-time stock price information for further analysis.

By combining these libraries, we will be able to load Tata Motors' stock price data, perform numerical calculations, organize and transform data efficiently, visualize trends and patterns, and handle potential warnings that might arise during the analysis process. Overall, this comprehensive approach utilizing various Python libraries will empower us to gain valuable insights into Tata Motors' stock performance.

Fetching Stock Data Along With Data Exploration and Information (Understanding the Dataset):

We gathered Tata Motors' historical stock data from BSE, a crucial dataset for in-depth stock price analysis using Python.

1. `Tata_Motors_Ltd.index` shows the date index, indicating when the data points were recorded.

2. `Tata_Motors_Ltd.head()` displays the 5 initial records, providing a snapshot of the dataset's starting entries.

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
2000-01-03 00:00:00+05:30	16.114738	16.114738	16.114738	16.114738	0	0.0	0.0
2000-01-04 00:00:00+05:30	16.114738	16.114738	16.114738	16.114738	0	0.0	0.0
2000-01-05 00:00:00+05:30	16.114738	16.114738	16.114738	16.114738	0	0.0	0.0
2000-01-06 00:00:00+05:30	16.114738	16.114738	16.114738	16.114738	0	0.0	0.0
2000-01-07 00:00:00+05:30	16.114738	16.114738	16.114738	16.114738	0	0.0	0.0

3. `Tata_Motors_Ltd.tail()` reveals the concluding 5 records, offering a glimpse into the dataset's final entries.

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
2023-08-10 00:00:00+05:30	623.000000	628.250000	601.599976	618.099976	1410506	0.0	0.0
2023-08-11 00:00:00+05:30	620.900024	624.500000	610.299988	611.700012	689271	0.0	0.0
2023-08-14 00:00:00+05:30	611.049988	611.700012	594.599976	607.150024	533603	0.0	0.0
2023-08-16 00:00:00+05:30	605.299988	621.150024	597.250000	618.799988	587907	0.0	0.0
2023-08-17 00:00:00+05:30	619.349976	621.900024	611.000000	613.549988	902619	0.0	0.0

4. `Tata_Motors_Ltd.shape` indicates the size of the dataset, specifying the number of rows and columns.
Here we have the output as (5875, 7) which implies that there are 5875 observations as on 17/8/2023 and 7 columns in the dataset.

5. `Tata_Motors_Ltd.columns` lists the column names, helping identify the type of information stored.

```
Index(['Open', 'High', 'Low', 'Close', 'Volume', 'Dividends', 'Stock Splits'], dtype='object')
```

Here we have 7 columns whose names are printed in the output.

6. `Tata_Motors_Ltd.info()` furnishes a comprehensive overview of the dataset, including data types and the count of non-missing values.

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 5875 entries, 2000-01-03 00:00:00+05:30 to 2023-08-17 00:00:00+05:30
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Open          5875 non-null   float64
 1   High          5875 non-null   float64
 2   Low           5875 non-null   float64
 3   Close         5875 non-null   float64
 4   Volume        5875 non-null   int64
 5   Dividends     5875 non-null   float64
 6   Stock Splits  5875 non-null   float64
dtypes: float64(6), int64(1)
memory usage: 367.2 KB
```

These operations assist in understanding the dataset's structure, contents, and coverage. The index provides a timeline, while the `head()` and `tail()` functions reveal sample data. Dimensions from `shape` aid in assessing dataset size. Column names are important for reference, and `info()` offers insights into the dataset's quality and content completeness

Preprocessing the Dataset for Analysis:

1. The "Dividends" and "Stock Splits" columns are deleted from the dataset using the `del` command, as they are not required for further analysis.

2. A new "Tomorrow" column is added to the dataset. It contains the "Close" prices shifted one day ahead, reflecting the stock's closing price for the following day.

3. A "Target" column is generated to indicate whether the stock price has increased the next day. It is computed by comparing "Tomorrow" and "Close" prices and assigning a binary value based on the comparison.

4. The dataset is now filtered to include data starting from January 1, 2002, using the `.loc` method. This narrows the focus to data relevant to the analysis.

The dataset now looks as follows

Date	Open	High	Low	Close	Volume	Tomorrow	Target
2023-08-10 00:00:00+05:30	623.000000	628.250000	601.599976	618.099976	1410506	611.700012	0
2023-08-11 00:00:00+05:30	620.900024	624.500000	610.299988	611.700012	689271	607.150024	0
2023-08-14 00:00:00+05:30	611.049988	611.700012	594.599976	607.150024	533603	618.799988	1
2023-08-16 00:00:00+05:30	605.299988	621.150024	597.250000	618.799988	587907	613.549988	0
2023-08-17 00:00:00+05:30	619.349976	621.900024	611.000000	613.549988	902619	NaN	0

5. Further, we check for missing values in the dataset. The `.isnull().sum()` function is used to count missing values in each column of the dataset, assisting in identifying data gaps.

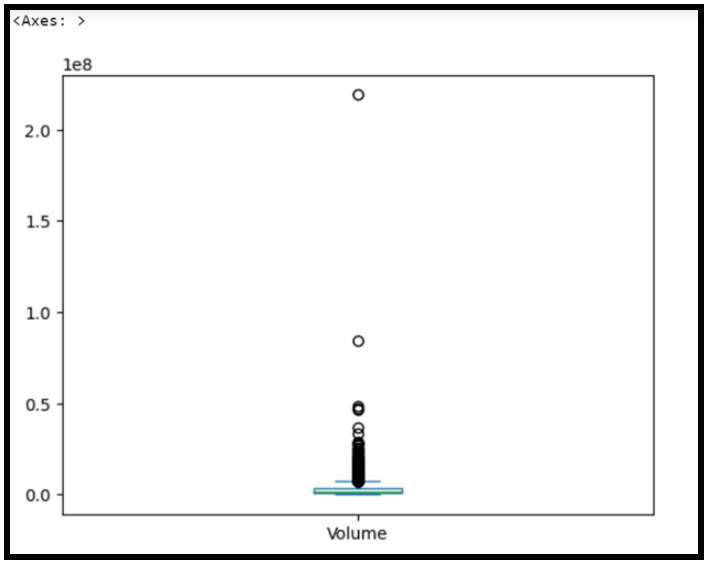
```
Open        0
High        0
Low         0
Close       0
Volume      0
Tomorrow    1
Target      0
dtype: int64
```

6. Since just 1 row has 1 missing value, removing it does not affect the accuracy of the dataset. Hence the row containing the missing value is removed , ensuring the dataset is complete and consistent for analysis.

These preprocessing steps involve cleaning the dataset by removing irrelevant columns, creating relevant features for analysis, converting data for classification, selecting a specific timeframe, identifying and handling missing values, resulting in a refined and usable dataset for further exploration.

To visualize the distribution of trading volume:

1. We analyze the total volume of Tata Motors stocks traded from 2002 till 2023.
2. A box plot is created. This type of plot shows the distribution of data points, including the median, quartiles, and potential outliers.



The plot provides insights into the spread of trading volume data, with the box indicating the interquartile range and the whiskers indicating data range. We see that there are quite a number of outliers and removing them from the model would affect the model adversely whereas here keeping the outliers is crucial. Hence, the outliers are not removed.

To enhance the data's quality for further analysis the data for only the volumes > 0 are taken. This step is essential to eliminate entries with zero trade volume, which might introduce errors or lack relevance for analysis.

By applying this data filtering, we ensure that the dataset consists of meaningful and accurate trading volume values. This is crucial for producing reliable insights during subsequent analysis. In summary, the boxplot helps us comprehend the data's distribution and outlier presence, while the filtering code contributes to refining the dataset, aligning it with accurate trading volume values.

After the dataset underwent changes due to the implemented code, two important functions, `Tata_Motors_Ltd.describe()` and `Tata_Motors_Ltd.info()`, offer a wealth of insights that illuminate the dataset's new characteristics and structure.

1. The invocation of the `describe()` function serves as a gateway to a comprehensive statistical overview of the dataset. Each numerical column in the DataFrame is subjected

to scrutiny, unveiling a multitude of key statistical metrics. These encompass central tendencies, dispersions, and quartiles, facilitating a profound understanding of the data's distribution and magnitude. Notably, post-processing alterations on the dataset are distinctly reflected in the statistical summary. Consequently, the description encapsulates the essence of the refined and transformed data, providing invaluable quantitative insights.

	Open	High	Low	Close	Volume	Tomorrow	Target
count	5318.000000	5318.000000	5318.000000	5318.000000	5.318000e+03	5318.000000	5318.000000
mean	220.757377	223.888882	217.199458	220.361284	2.813754e+06	220.474258	0.507146
std	168.931564	170.955194	166.608023	168.645457	4.743110e+06	168.707382	0.499996
min	7.375733	7.375733	7.375733	7.375733	6.819000e+04	7.375733	0.000000
25%	74.780851	76.537500	72.263389	74.563824	7.304540e+05	74.612499	0.000000
50%	178.960922	181.747523	176.000000	178.525108	1.575864e+06	178.606285	1.000000
75%	378.961548	383.961043	373.323152	377.684181	3.351340e+06	377.718864	1.000000
max	646.200012	665.299988	639.000000	640.599976	2.190277e+08	640.599976	1.000000

2. Concurrently, the `info()` function engages in divulging intricate specifics about the dataset. It unveils essential attributes, including the counts of non-null values, elucidation of data types within each column, and meticulous memory utilization. In the wake of the code modifications, the `info()` function unfurls an updated, real-time panorama of the dataset's morphology. This comprehensive glimpse distinctly accentuates the influence of filtration and preprocessing steps on the dataset's fundamental structure and composition.

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 5318 entries, 2002-01-02 00:00:00+05:30 to 2023-08-16 00:00:00+05:3
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Open      5318 non-null   float64
 1   High      5318 non-null   float64
 2   Low       5318 non-null   float64
 3   Close     5318 non-null   float64
 4   Volume    5318 non-null   int64
 5   Tomorrow  5318 non-null   float64
 6   Target    5318 non-null   int64
dtypes: float64(5), int64(2)
memory usage: 332.4 KB
```

In a summative context, the `describe()` function imparts a profound understanding of the dataset's statistical underpinnings, while the `info()` function magnifies our grasp on the

restructured dataset. This dynamic duo empowers us with a multidimensional comprehension of the refined data's essence, characteristics, and framework.

3. We now see the opening stock price in 2002.

```
7.375732898712158
```

Whereas, currently the closing price of the stock rose to

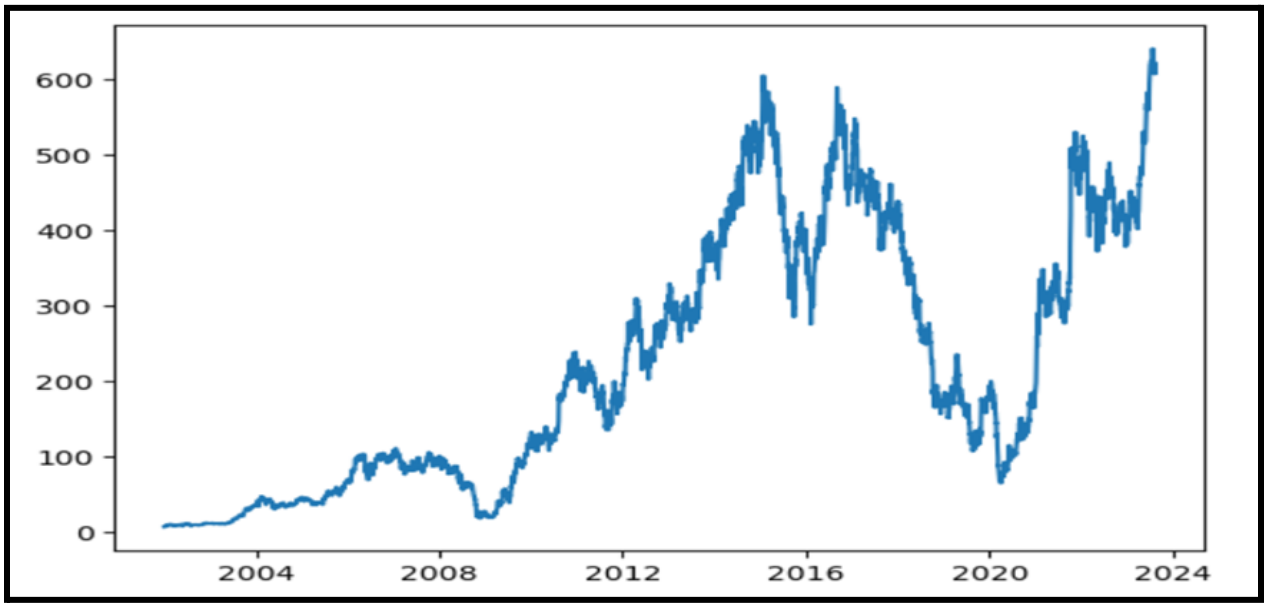
```
640.5999755859375
```

Visualizations :

We use Matplotlib library to create a line plot of the closing prices of Tata Motors' stock. The data for the plot is retrieved from the "Close" column of the `**Tata_Motors_Ltd**` DataFrame. The x-axis of the plot represents the time (e.g., trading days), and the y-axis represents the closing prices of the stock.

The graph plots the historical closing prices of Tata Motors' stock over time, and then displays the plot for analysis. This visualization allows you to observe trends, fluctuations, and patterns in the stock's closing prices, aiding in the interpretation of its historical performance and potential insights for future investment decisions.

The figure illustrates how Tata Motors' stock prices changed over several years. Starting from the year 2004 and going up to 2016, the closing prices of the stock moved in an upward direction. This indicates that during this period, the stock generally became more valuable, with its prices increasing.However, in the years between 2016 and 2020, a different pattern emerged. During this period, the closing prices of Tata Motors' stock went down. This means that the stock's value decreased, and investors might have experienced a decline in the value of their investments.Interestingly, after the year 2020, the situation changed again. From 2020 to 2024, the closing prices of the stock started to rise once more. This suggests that the stock's value began to increase again, potentially leading to better returns for investors who held the stock during this period.The figure provides a visual representation of how Tata Motors' stock prices evolved over time. It shows a period of growth from 2004 to 2016, a decline from 2016 to 2020, and a subsequent rise from 2020 to 2024. These trends offer insights into the stock's historical performance and the changing market conditions that influenced its value.



We now use a line plot that represents both the opening and closing prices of Tata Motors' stock over time. The data for the plot is taken from the "Open" and "Close" columns of the `**Tata_Motors_Ltd**` DataFrame. The x-axis of the plot represents the time (e.g., trading days), while the y-axis represents the stock prices.

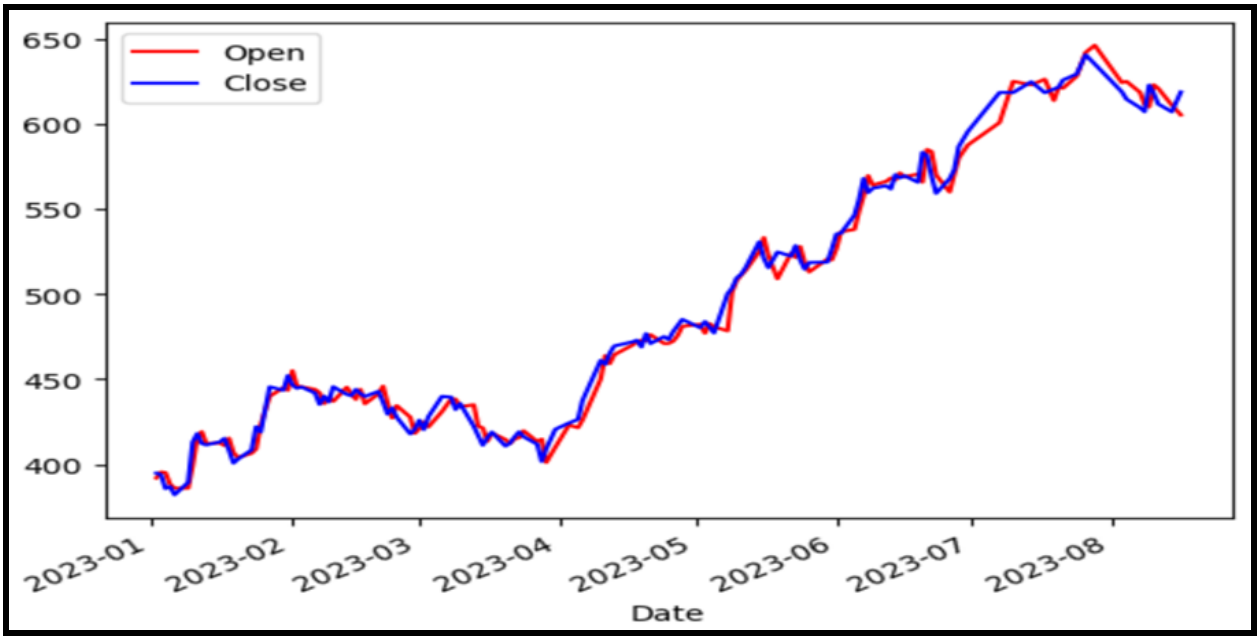
The code snippet as a whole sets up a figure with specified dimensions, generates a line plot showing both the opening and closing prices of Tata Motors' stock over time, and then displays the plot for analysis. This visualization enables us to simultaneously observe the trends in both opening and closing prices, providing insights into how these prices have fluctuated over the recorded period and potentially revealing patterns and correlations between these two values.

The following graph shows that over the years both the opening and closing prices have a similar trend.

.
We now create a new DataFrame named `Tata_Motors_Ltd_2023` containing a subset of data from the original DataFrame `Tata_Motors_Ltd`. This subset includes all the data starting from January 1, 2023, and onward, allowing for focused analysis or further processing specific to the year 2023.

A line plot of the "Close" prices of Tata Motors Ltd for the year 2023 is drawn where the x-axis of the plot will correspond to the dates from the DataFrame's index, and the y-axis will correspond to the "Close" prices.This kind of visualization can help us to understand the trend and fluctuations in the closing prices of Tata Motors Ltd throughout the year 2023.

The codeTata_Motors_Ltd_2023.plot.line(y=["Open", "Close"], color=["r", "b"], use_index=True)
creates a line plot with two lines: one representing the "Open" prices and another representing the "Close" prices of Tata Motors Ltd for the year 2023. The x-axis will show the dates from the DataFrame's index, and the y-axis will show the respective prices. The different colors (red and blue) make it easy to distinguish between the two lines. This type of visualization helps us to compare the opening and closing prices over time and understand their relationship and trends.



The red line represents the opening prices, and the blue line represents the closing prices. When we look at the plot, we notice that the red and blue lines have a similar trend. This means that the movement of the opening prices tends to follow a similar pattern as the movement of the closing prices throughout the year 2023. If the red line (opening prices) generally goes up or down, you'll likely see a similar movement in the blue line (closing prices).

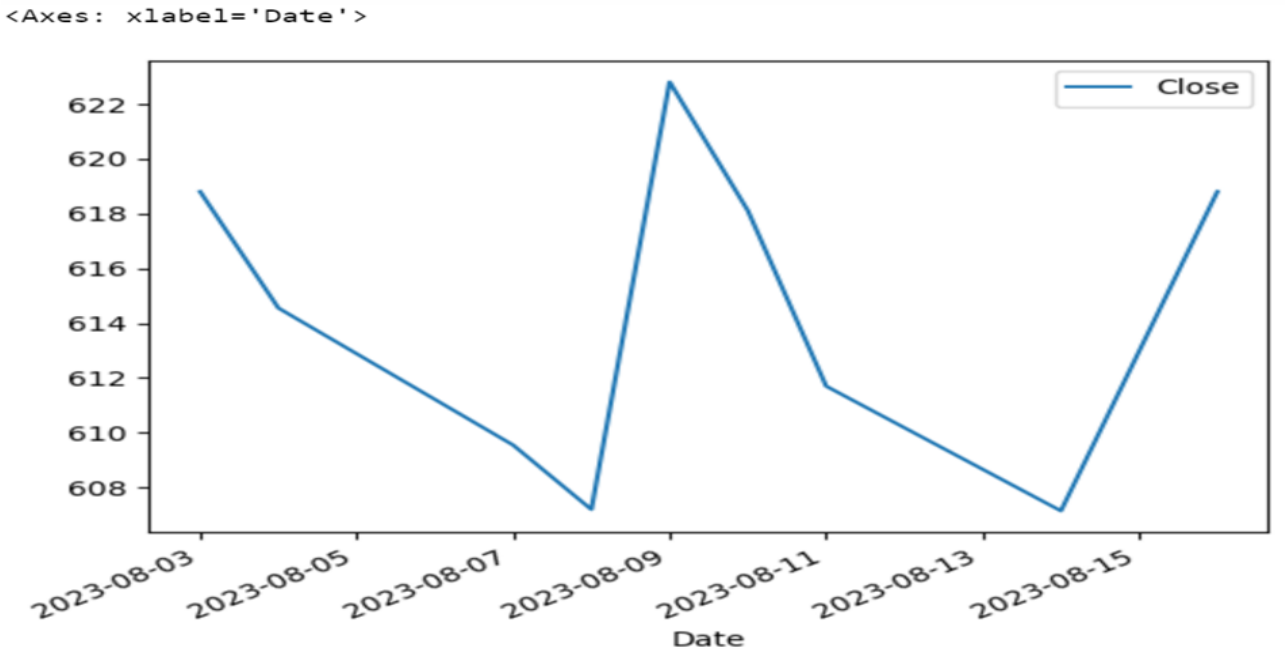
This type of visualization helps us quickly assess how closely related the opening and closing prices are over time. If they consistently move together, it could indicate that market sentiment or external factors affecting the stock are influencing both the opening and closing prices in a similar way. On the other hand, if there are instances where they diverge significantly, it might suggest that there are factors driving changes in the opening and closing prices independently.

In essence, the similar trend between the red and blue lines indicates a correlation or similarity between the movement of opening and closing prices for Tata Motors Ltd in the year 2023.

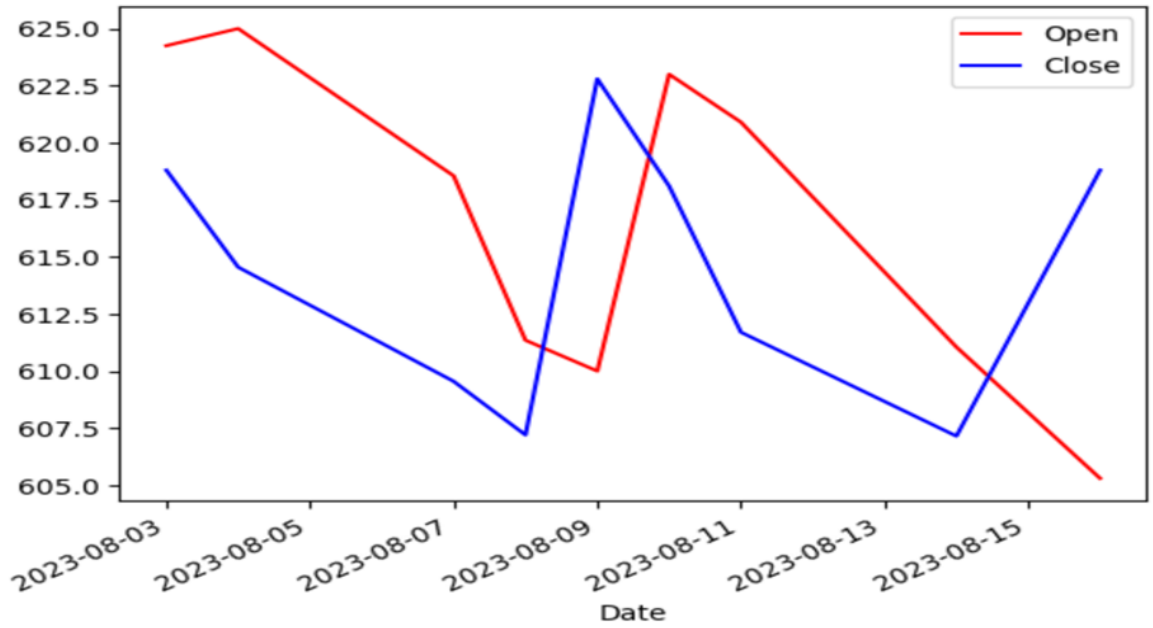
We will now analyze the performance of a stock in August 2023.We create a new DataFrame named `**Tata_Motors_Ltd_2023_Aug**` that contains a subset of data from the original DataFrame `**Tata_Motors_Ltd**`. This subset includes all the data from August 1, 2023, to the end of the dataset. This can be useful for focusing your analysis

specifically on the month of August 2023 and performing various analyses, visualizations, or calculations specific to that period.

The line plot of closing prices for Tata Motors Ltd in August 2023 provides a visual representation of the stock's performance during that specific month. It helps you quickly grasp the overall trend, patterns, and fluctuations in closing prices, enabling you to draw insights and make informed decisions related to the stock's behavior during that time.



The depicted figure illustrates the dynamic movement of closing prices for Tata Motors Ltd over a specific period. At the outset, there is a discernible downward trend as the closing prices experience a decline. However, this initial descent is followed by a noteworthy reversal, with prices gradually starting to ascend. This upward trajectory is not sustained indefinitely, as the plot indicates a subsequent dip in prices. Yet, the narrative takes another turn, with prices rebounding once more and ascending once again. In this series of fluctuations, we witness a pattern characterized by alternating rises and falls, underscoring the inherent volatility of the stock's performance. Such intricate price behavior underscores the complex interplay of market forces and external factors that shape the stock's journey during the depicted timeframe.



The above figure represents both the opening and closing prices for Tata Motors Ltd in August 2023. There are distinct time periods during which the opening price surpasses the closing price, as well as periods where the closing price exceeds the opening price. These fluctuations reflect the varying dynamics of Tata Motors Ltd stock performance over the specified timeframe.

Opening Price Higher than Closing Price:

In the sections where the opening price (represented by the red line) is positioned above the closing price (blue line), it indicates that the stock began the trading session with a higher valuation than it concluded. This pattern suggests initial optimism or demand, potentially driven by positive news, market sentiment, or other factors influencing investors at the start of those trading periods.

Closing Price Higher than Opening Price:

Conversely, in the intervals where the closing price surpasses the opening price, the blue line ascends above the red line. This scenario implies that the stock experienced a gain in value throughout the trading day, indicating that the market sentiment improved as the trading session progressed. This kind of movement can be due to developments, announcements, or market trends that boosted investor confidence and led to an increase in the stock's value by the end of the trading period.

Overall, the alternating nature of opening and closing prices being higher than each other reflects the ebb and flow of market dynamics, as supply and demand, news events, economic indicators, and other influences continuously shape the stock's behavior within the defined time frame. This pattern underscores the ever-changing nature of stock market activities and the interplay of factors that impact stock prices.

Model Building (Random Forest Classifier) And Evaluation:

-Importing confusion_matrix and precision_score from scikit-learn.

1. **Confusion Matrix** :
This table presents the counts of true positive, true negative, false positive, and false negative predictions to summarize the performance of a classification model. It aids in the comprehension of how well the model is performing in terms of correctly and erroneously classifying cases. To create this matrix in Python, import the scikit-learn 'confusion_matrix' function.

2. Precision Score :
The number of occurrences that the model correctly forecasted as positive (true positives) out of all instances that it correctly predicted as positive (including true positives and false positives) is known as precision. It aids in the comprehension of the accuracy of the model's optimistic forecasts. To determine this metric in Scikit-Learn, import the 'precision_score' function. By importing these functions and applying them in the evaluation procedure, the accuracy of the model's predictions can be evaluated and decisions can be made regarding it's performance.
- Making predictions using the trained model on the test data.

1. ***Trained Model*** :

A machine learning model discovers patterns and relationships in the data after being trained using a labeled dataset (training data). These patterns provide the model the ability to anticipate or categorize previously undiscovered data. We have used all the observations except the last 100 observations for training the model.

2. ***Test Data*** :

A different dataset is used called the test dataset to gauge how effectively the trained model generalizes to fresh input. There are examples in this dataset that the model has never encountered before. It simulates actual situations so that the model's performance can be evaluated using hypothetical data. The last 100 observations are used for testing the model.

3. ***Making Predictions*** :

The attributes of instances from the test database is inported using a trained model. In order to predict target variables for each instance of the test data, this model is based on patterns it has learned in its training.

4. ***Output*** :

Class labels or numerical values for classification tasks are usually used in the model's predictions. These forecasts are presented in the test dataset to compare with real target values. The accuracy of the model is given by

```
0.44
```

The precision_score of the model is given by

```
0.6428571428571429
```

However, we can see that the model is just 44% accurate and hence we need a more robust model to improve the model.
Thus we now implement backtesting to improve the model's performance.

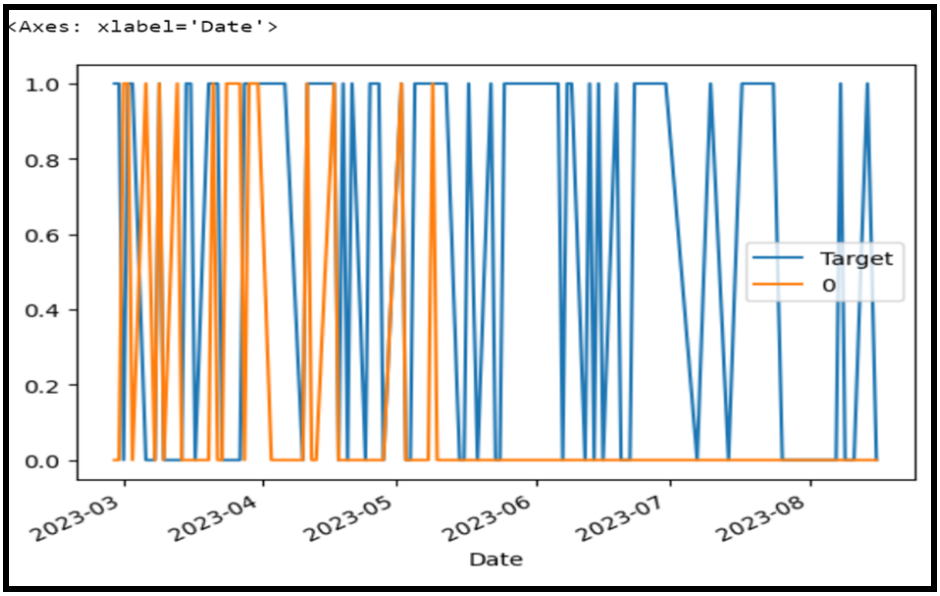
5. ***Evaluation*** :

Calculating evaluation criteria like accuracy, precision is performed when the model's predicted values are compared with actual data in a test dataset. These metrics give an indication of how well the model performs on unseen data.
-Calculating the confusion matrix to evaluate prediction performance.
-Computing accuracy and printing it.
-Calculating precision score and plotting actual vs. predicted values.

1.True Positives (TP): These are instances that the model correctly predicts as positive. In other words, the model predicts a positive outcome and the actual outcome is actually positive.
2. True Negatives (TN): These are the instances where the correct prediction model is negative. The model predicts a negative outcome and the actual outcome is actually negative.
3. False Positives (FP): These are the instances where the model falsely predicts positive while the actual outcome is negative. This is also known as a "Type I Error".
4. False Negatives (FN): These are instances where the model falsely predicts negative when the actual result is positive. This is also known as a "Type II error".
The confusion matrix is typically organized like this:

	Actual Negative	Actual Positive
Predicted Positive	TP	FP
Predicted Negative	FN	TN

By calculating the counts of these four values, insights about the performance of the model can be gained. Based on the confusion matrix, different evaluation metrics can be calculated, such as:
- Accuracy: The proportion of correct predictions out of all predictions.
- Precision: The proportion of true positive predictions out of all positive predictions made by the model.
With regard to the confusion matrix from the given stock price data, it is found out that the TP, FP, FN, TN are 35, 5, 51, 9 respectively and thereby it is observed that the model is 44% accurate.
In Python, the ‘confusion_matrix’ function can be used from the scikit-learn library to compute the confusion matrix based on the model predictions and the actual labels of the test data. This matrix helps to understand where the model excels and where it can go wrong, leading to more informed decisions about model improvement.



This is the combined plot which shows the target and the actual values. It can be seen that the model prediction is mostly faulty.

Building a Robust Testing Method:

In addition to predictive modeling, the code encompasses a backtesting method. This methodology involves testing the predictive model's effectiveness by simulating its application on historical data. By assessing how well the model's predictions align with actual outcomes over different time periods, the code allows for a comprehensive evaluation of the model's performance under varying market conditions.
The backtest function simulates a backtesting process by iteratively splitting the provided data into training and test sets. For each iteration, the model is trained on the training data and used to predict target values for the test data. The predictions from each iteration are collected and concatenated into a DataFrame. .

Evaluation Metrics:
The code aims to evaluate the model's performance using two metrics:

Value Counts of Predictions:
The line predictions["Predictions"].value_counts() provides a count of each predicted value. This can help understand the distribution of the predicted stock price movements.

Precision Score:
The precision_score(predictions["Target"], predictions["Predictions"]) calculates the precision score, which assesses the accuracy of positive predictions made by the model. In the context of stock price prediction, precision can indicate how well the model predicts upward price movements.

Proportion of Target Values:
The line predictions["Target"].value_counts() / predictions.shape[0] computes the proportion of each class (likely corresponding to price movements) in the target variable. This can provide insights into the distribution of actual stock price movements.

```
0    0.504
1    0.496
Name: Target, dtype: float64
```

From analysing the robust model we can come to a conclusion that our model has improved and we have attained a precision of 72% percent. Let us say on Monday the stock price of X pvt.ltd is 30L. Let us say that our model predicts that on Tuesday the stock prices will be higher than 30L.
The robust model tells that this prediction of future price of the stock market may be 72 percent accurate.

Conclusion:

The provided code performs a comprehensive analysis of Tata Motors' stock data. It fetches historical stock information using the Yahoo Finance API and preprocesses it to remove any missing or irrelevant data. The data is then visualized using Matplotlib, highlighting the stock's closing and opening prices. The code builds a predictive model using the RandomForestClassifier from Scikit-Learn, utilizing key predictors like close and volume. This model is evaluated through confusion matrices, accuracy calculations, and precision scores to gauge its effectiveness in predicting stock performance. The script goes beyond simple model evaluation by implementing a backtesting method. This technique assesses the model's predictive power across different time periods, allowing for a more robust evaluation of its performance. In summary, the code provides a holistic approach to analyzing stock data, building a predictive model, and rigorously assessing its predictive capabilities.

----------X----------