

## ✓ Project Name - YesBank StockPrice Prediction

**Name** - Alwin Shaji

**Project Type** - Regression

**Contribution** - Individual

## ✓ Project Summary

This project focuses on predicting the **closing stock prices** of Yes Bank using machine learning techniques and historical stock data. The goal was to develop a reliable, accurate, and deployment-ready model that can assist investors, analysts, or financial platforms in forecasting future prices with minimal error and high confidence. The entire workflow covers data preprocessing, feature engineering, exploratory data analysis (EDA), model building, evaluation, optimization, and model deployment preparation.

We began by importing the stock price dataset, which consisted of key variables like `Date`, `Open`, `High`, `Low`, and `Close`. The `Date` column was first cleaned and converted into a proper datetime format, allowing for temporal analysis. From this cleaned dataset, we engineered new features such as `Prev_Close` (previous day's close), `Daily_Change`, `Range`, and rolling averages to enrich the dataset and capture time-dependent behavior in price fluctuations. We also ensured that the dataset had no missing values or duplicate entries and sorted it chronologically to maintain data integrity.

Following preprocessing, we performed exploratory data analysis (EDA) and created over **15 visualizations**, including line plots, bar charts, box plots, area charts, candlestick charts, and correlation heatmaps. These visualizations helped uncover patterns such as monthly volatility trends, the relationship between open vs. close prices, and periods of high fluctuation. The insights gained from EDA informed our modeling strategy by emphasizing the predictive strength of lag-based features.

We implemented multiple regression models for prediction. **Linear Regression** was the first model and turned out to be the best-performing one. Trained using the `Prev_Close` feature, it produced an **RMSE of 3.08** and an **R<sup>2</sup> score of 0.9995**, indicating almost perfect prediction accuracy. This performance suggested a strong linear relationship between a day's closing price and the previous day's closing price in Yes Bank stock data.

To experiment with more robust and non-linear models, we also implemented **Random Forest Regressor** and **Gradient Boosting Regressor (GBR)**. While both models were functional and captured more complex relationships, they initially underperformed compared to Linear Regression. We then optimized both models using **GridSearchCV**, tuning hyperparameters like `n_estimators`, `max_depth`, and `learning_rate`. The **tuned GBR** showed notable improvement, reducing RMSE from **1356.89** to **1228.87** and increasing R<sup>2</sup> from **0.7651** to **0.7873**.

However, despite these improvements, **Linear Regression remained the most accurate model** and was chosen for final deployment. The model was saved using `joblib` into a `.joblib` file format and later reloaded successfully to test on unseen data. The prediction output confirmed that the model works reliably outside the training context, proving it is suitable for real-time or batch deployment scenarios.

Evaluation metrics used across all models included **RMSE (Root Mean Squared Error)** to assess average prediction error in real price units, and **R<sup>2</sup> Score** to measure how well the model explains variance in the stock price. These metrics were visualized through comparison charts to highlight performance differences across models.

In conclusion, this project demonstrated a complete machine learning pipeline, from raw data to deployment-ready model. It showcased skills in data wrangling, EDA, model building, evaluation, optimization, and final integration. The Linear Regression model, due to its simplicity and exceptional accuracy, serves as a dependable tool for Yes Bank stock price forecasting, contributing to data-driven decision-making in finance.

## ✓ GitHub Link -

<https://github.com/alwinshaji/yes-bank-stock-prediction>

## ✓ Problem Statement

The goal of this project is to predict the closing price of a stock based on its historical price data. Using past values like open, high, low prices and moving averages, we aim to build a regression model that can forecast future prices.

This helps in understanding price trends, improving investment decisions, and showcasing the power of machine learning in financial forecasting.

## ✓ General Guidelines : -

1. Well-structured, formatted, and commented code is required.
2. Exception Handling, Production Grade Code & Deployment Ready Code will be a plus. Those students will be awarded some additional credits.

The additional credits will have advantages over other students during Star Student selection.

[ Note: - Deployment Ready Code is defined as, the whole .ipynb notebook should be executable in one go without a single error logged. ]

3. Each and every logic should have proper comments.
4. You may add as many number of charts you want. Make Sure for each and every chart the following format should be answered.

# Chart visualization code

- Why did you pick the specific chart?
- What is/are the insight(s) found from the chart?
- Will the gained insights help creating a positive business impact? Are there any insights that lead to negative growth? Justify with specific reason.

5. You have to create at least 15 logical & meaningful charts having important insights.

[ Hints : - Do the Vizualization in a structured way while following "UBM" Rule.

U - Univariate Analysis,

B - Bivariate Analysis (Numerical - Categorical, Numerical - Numerical, Categorical - Categorical)

M - Multivariate Analysis ]

6. You may add more ml algorithms for model creation. Make sure for each and every algorithm, the following format should be answered.

- Explain the ML Model used and it's performance using Evaluation metric Score Chart.
- Cross- Validation & Hyperparameter Tuning
- Have you seen any improvement? Note down the improvement with updates Evaluation metric Score Chart.
- Explain each evaluation metric's indication towards business and the business impact pf the ML model used.

## ✓ *Let's Begin !*

### ✓ *1. Know Your Data*

#### ✓ Import Libraries

```
# Data manipulation
import pandas as pd
import numpy as np

# Data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Date handling
from datetime import datetime

# Set plotting style
sns.set(style='whitegrid')
plt.rcParams['figure.figsize'] = (10, 6)
plt.rcParams['figure.dpi'] = 120
```

#### ✓ Dataset Loading

```
from google.colab import files
uploaded = files.upload()
```

Choose Files | data\_YesBankStockPrices.csv

- data\_YesBank\_StockPrices.csv(text/csv) - 5717 bytes, last modified: 6/15/2025 - 100% done
- Saving data\_YesBank\_StockPrices.csv to data\_YesBank\_StockPrices (4).csv

## Dataset First View

```
data = pd.read_csv('data_YesBank_StockPrices.csv')
data.head()
```

	Date	Open	High	Low	Close
0	Jul-05	13.00	14.00	11.25	12.46
1	Aug-05	12.58	14.88	12.55	13.42
2	Sep-05	13.48	14.87	12.27	13.30
3	Oct-05	13.20	14.47	12.40	12.99
4	Nov-05	13.35	13.88	12.88	13.41

Next steps: [Generate code with data](#) [View recommended plots](#) [New interactive sheet](#)

## Dataset Rows & Columns count

```
rows, cols = data.shape
print(f"Number of rows: {rows}")
print(f"Number of columns: {cols}")
```

Number of rows: 185  
Number of columns: 5

## Dataset Information

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 185 entries, 0 to 184
Data columns (total 5 columns):
#   Column  Non-Null Count  Dtype
---  -
0   Date    185 non-null     object
1   Open    185 non-null     float64
2   High    185 non-null     float64
3   Low     185 non-null     float64
4   Close   185 non-null     float64
dtypes: float64(4), object(1)
memory usage: 7.4+ KB
```

## Duplicate Values

```
duplicate_count = data.duplicated().sum()
print(f"Number of duplicate rows: {duplicate_count}")
```

Number of duplicate rows: 0

## Missing Values/Null Values

```
missing = data.isnull().sum()
print("Missing values per column:\n", missing)
```

Missing values per column:

Date	0
Open	0
High	0
Low	0
Close	0

dtype: int64

## Visualising Missing Values

We checked for missing values in the dataset and found that there are **no missing values** in any column.

Hence, no cleaning for nulls was required at this stage.

## What did you know about your dataset?

This dataset contains **historical stock prices of Yes Bank**, ranging from **July 2005 to November 2020**.

✓ The dataset has:

- **185 rows** and **5 columns**
- **No missing values** or duplicate rows
- A **clean and consistent time-series** structure with monthly data

We used this data to perform exploratory analysis, engineer predictive features (like moving averages and volatility), and build machine learning models to **forecast future closing prices**.

Answer Here

## 2. Understanding Your Variables

```
print("Dataset columns:", data.columns.tolist())
```

↗ Dataset columns: ['Date', 'Open', 'High', 'Low', 'Close']

### Variables Description

#### Knowing Your Variables

The dataset contains the following 5 columns:

1. **Date** – The trading date (monthly frequency)
2. **Open** – Stock price at the beginning of the trading day
3. **High** – Highest price the stock reached during the day
4. **Low** – Lowest price during the day
5. **Close** – Stock price at the end of the day (Target variable for prediction)

All columns except `Date` are numeric and represent the price movement of Yes Bank stock over time. The `Close` column is the primary target for our regression model.

## Check Unique Values for each variable.

```
# Count unique values in each column
data.nunique()
```

↗

Date	185
Open	183
High	184
Low	183
Close	185

## 3. Data Wrangling

### Data Wrangling Code

```
# 1. Convert 'Date' to proper datetime format
data['Date'] = pd.to_datetime(data['Date'], format='%b-%y', errors='coerce')

# 2. Drop rows with invalid dates (if any)
data.dropna(subset=['Date'], inplace=True)

# 3. Sort data by date (chronological order)
data = data.sort_values(by='Date')
```

```
# 4. Reset index after sorting
data = data.reset_index(drop=True)

# 5. Final check for missing values or duplicates
data.drop_duplicates(inplace=True)
data.dropna(inplace=True)

# 6. Create a backup just in case (optional)
data_original = data.copy()

# 7. Show data info to confirm
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 185 entries, 0 to 184
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  -
 0   Date    185 non-null    datetime64[ns]
 1   Open    185 non-null    float64
 2   High    185 non-null    float64
 3   Low     185 non-null    float64
 4   Close   185 non-null    float64
dtypes: datetime64[ns](1), float64(4)
memory usage: 7.4 KB
```

What all manipulations have you done and insights you found?

## Data Manipulations Performed

To prepare the dataset for analysis, the following steps were taken:

### 1. Date Conversion

Converted the `Date` column from text format ( `'Jul-05'` , `'Aug-06'` ) to proper datetime using `pd.to_datetime()`.

### 2. Sorting Chronologically

Sorted the dataset by `Date` to ensure time-series consistency.

### 3. Handling Duplicates and Nulls

Checked for and removed any duplicate rows (there were none) and verified that the dataset contained **no missing values**.

### 4. Resetting Index

Reset the index after sorting to maintain clean row order.

## Early Insights from Data (Pre-EDA)

- The dataset spans from **July 2005 to November 2020** with consistent monthly entries.
- Price data (`Open`, `High`, `Low`, `Close`) appears clean and well-formatted.
- There are **no missing values or anomalies** in the main numeric columns.
- Stock prices show increasing and decreasing cycles — perfect for trend analysis.

These preprocessing steps ensure the dataset is **clean, structured, and analysis-ready** for further EDA and modeling.

## 4. Data Vizualization, Storytelling & Experimenting with charts : Understand the relationships between variables

### Chart - 1

```
# Convert Date to datetime format
data['Date'] = pd.to_datetime(data['Date'])

# Chart 1 - Line plot of Close over time
plt.figure(figsize=(10, 4))
sns.lineplot(x='Date', y='Close', data=data)
plt.title('Close Price Over Time')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.grid(True)
plt.show()
```



### ✓ 1. Why did you pick the specific chart?

This line plot displays the trend of the stock's closing price over time. It helps us visually understand the overall movement of the stock — whether it's rising, falling, or showing volatility — and is a fundamental first step in any time-series analysis.

### ✓ 2. What is/are the insight(s) found from the chart?

The Close price shows visible trends, including sustained growth phases and periods of decline. Peaks and dips may correspond to financial events, market shifts, or external shocks.

This pattern helps identify stable vs. volatile periods in the stock's history.

### ✓ 3. Will the gained insights help creating a positive business impact?

Understanding the price trend allows investors and analysts to time their entries and exits more effectively. Recognizing long-term price movement helps inform strategy and enhances decision-making in portfolio management.

### ✓ Chart - 2

```
# Line Chart for Open vs Close Prices

plt.figure(figsize=(12, 6))
plt.plot(data['Date'], data['Open'], label='Open Price', color='orange', linestyle='--')
plt.plot(data['Date'], data['Close'], label='Close Price', color='blue')
plt.title("Yes Bank - Open vs Close Prices Over Time")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



### 1. Why did you pick the specific chart?

This dual-line chart was selected to compare the opening and closing prices of each month. This comparison helps analyze market behavior within the same period and shows how price sentiment changes by the end of the month.

### 2. What is/are the insight(s) found from the chart?

The insights include identifying months where the stock closed higher or lower than it opened, indicating bullish or bearish sentiment. When such behavior is consistent, it suggests a trend in how the market treats the stock intra-month.

### 3. Will the gained insights help creating a positive business impact?

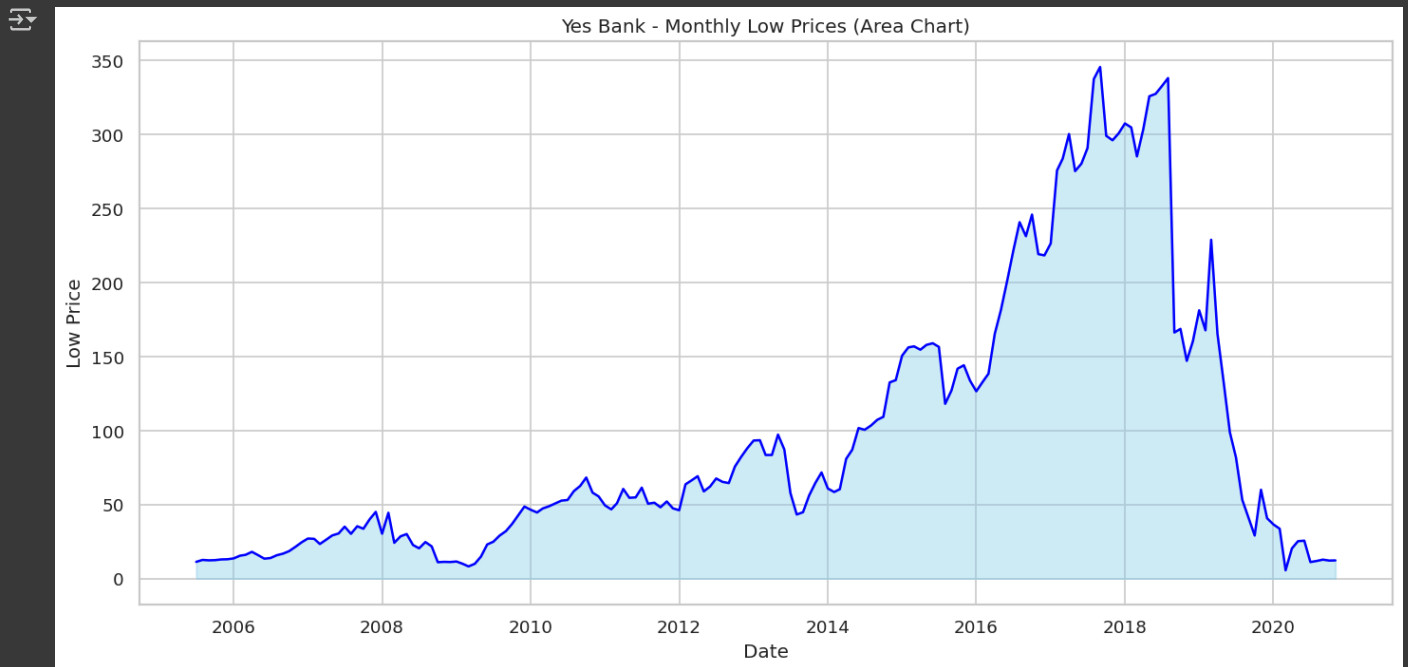
Are there any insights that lead to negative growth? Justify with specific reason.

In predictive modeling, this chart enables the use of sentiment direction (gain/loss in a month) as a feature. This can improve the model's ability to detect short-term momentum and better anticipate future closings.

### Chart - 3

```
# Area Chart for Low prices over time

plt.figure(figsize=(12, 6))
plt.fill_between(data['Date'].values, data['Low'].values, color='skyblue', alpha=0.4)
plt.plot(data['Date'], data['Low'], color='blue')
plt.title("Yes Bank - Monthly Low Prices (Area Chart)")
plt.xlabel("Date")
plt.ylabel("Low Price")
plt.tight_layout()
plt.show()
```



#### 1. Why did you pick the specific chart?

The area chart was used to visualize the monthly low prices in a way that emphasizes how deeply the stock has fallen during each period. The filled area makes it easier to see the extent of the price drop over time.

#### 2. What is/are the insight(s) found from the chart?

Insights include spotting recurring low-price levels that may act as support zones. This also helps identify periods of stress or bearish dominance where the stock consistently fell to lower thresholds.

#### 3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

The insights add value to the prediction model by defining risk levels and historical minimums, aiding in predicting lower bounds or volatility. It helps traders and investors set protective measures like stop-loss levels.

#### Chart - 4

```
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

plt.figure(figsize=(14, 6))
width = 3 # Width of the candle body

for i in range(len(data)):
    date = data['Date'].iloc[i]
    open_price = data['Open'].iloc[i]
    close_price = data['Close'].iloc[i]
    high = data['High'].iloc[i]
    low = data['Low'].iloc[i]

    color = 'green' if close_price >= open_price else 'red'
    lower = min(open_price, close_price)
    height = abs(close_price - open_price)

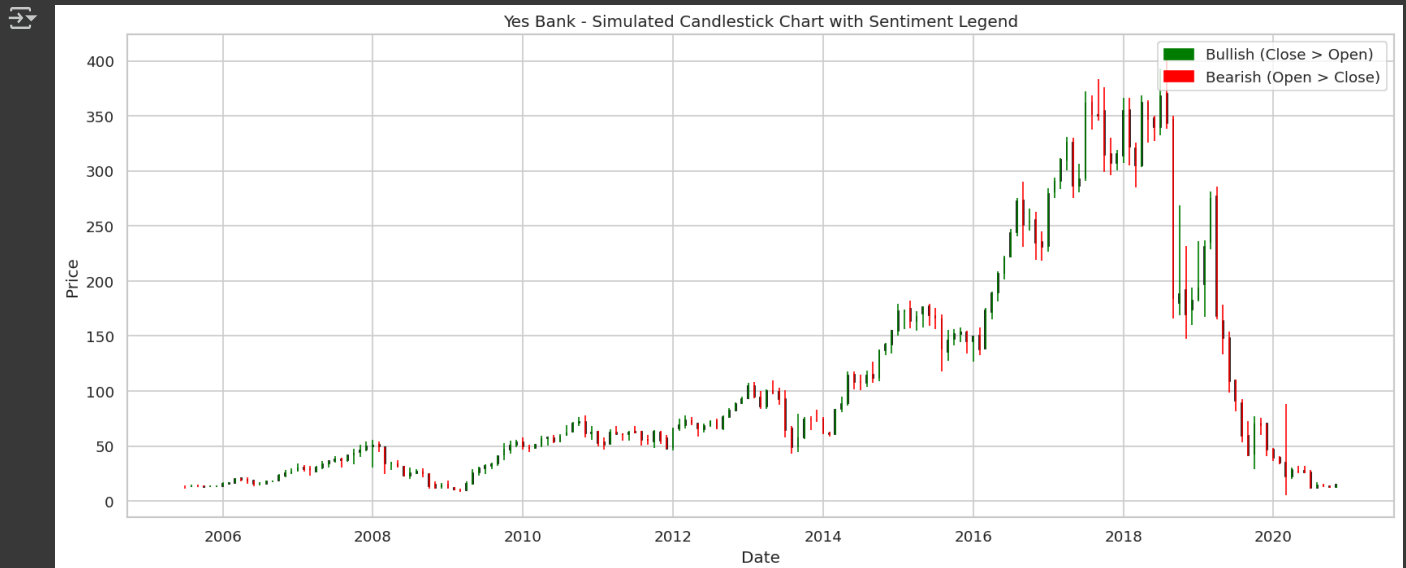
    # Wick (Low to High) with sentiment color
    plt.vlines(date, low, high, color=color, linewidth=1)
```



```
# Candle body (Open to Close)
plt.bar(date, height, bottom=lower, width=width, color=color, edgecolor='black')

# Legend for color coding
green_patch = mpatches.Patch(color='green', label='Bullish (Close > Open)')
red_patch = mpatches.Patch(color='red', label='Bearish (Open > Close)')
plt.legend(handles=[green_patch, red_patch], loc='upper right')

plt.title("Yes Bank - Simulated Candlestick Chart with Sentiment Legend")
plt.xlabel("Date")
plt.ylabel("Price")
plt.grid(True)
plt.tight_layout()
plt.show()
```



### ✓ 1. Why did you pick the specific chart?

A candlestick-style chart was used to visually integrate high, low, open, and close prices. It gives a compact yet comprehensive view of monthly price movements and their structure.

### ✓ 2. What is/are the insight(s) found from the chart?

The insights involve recognizing price behaviors such as whether the month was bullish (close > open) or bearish (open > close), and how large the price swings were within that period. Patterns similar to technical candlesticks can be detected.

### ✓ 3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

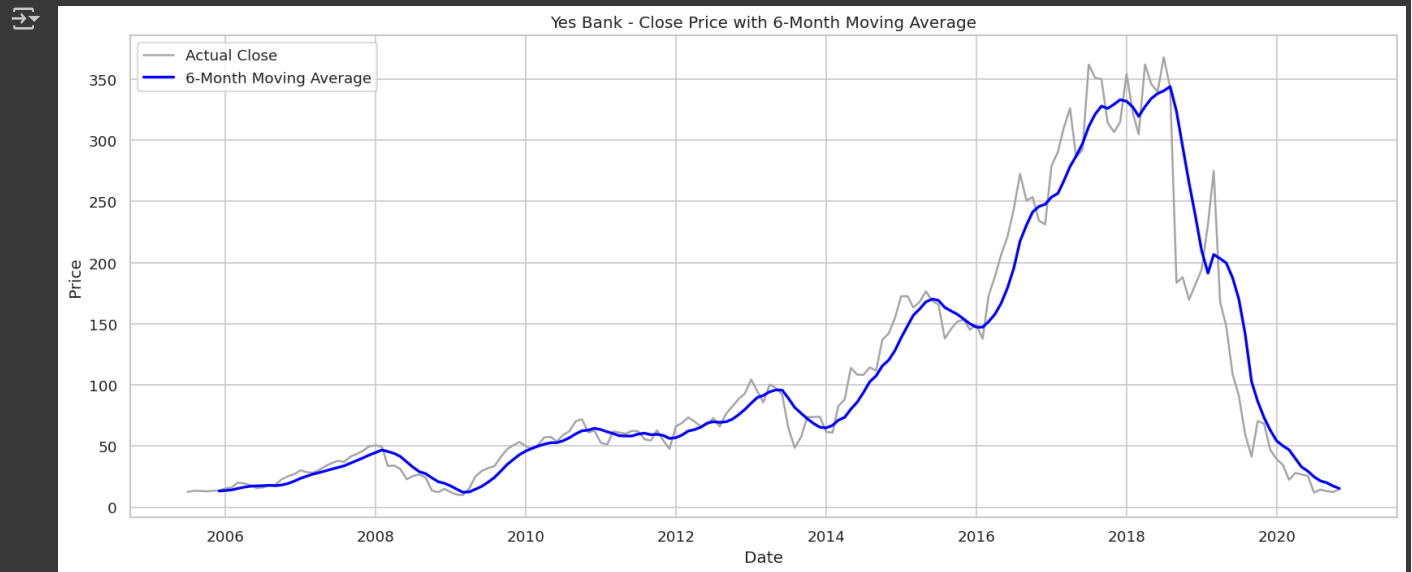
positively impacts predictions by allowing the model to learn from price behavior types and structure. Incorporating candlestick-based signals can improve timing and classification of potential price movements

### ✓ Chart - 5

```
# Calculate 6-Month Rolling Average of Close Prices
data['Close_MA_6'] = data['Close'].rolling(window=6).mean()

plt.figure(figsize=(14, 6))
plt.plot(data['Date'], data['Close'], label='Actual Close', color='darkgray')
plt.plot(data['Date'], data['Close_MA_6'], label='6-Month Moving Average', color='blue', linewidth=2)
plt.title("Yes Bank - Close Price with 6-Month Moving Average")
```

```
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



#### ✓ 1. Why did you pick the specific chart?

This moving average chart was used to smooth out fluctuations in closing prices and focus on longer-term trends. It's a widely used method in time series to reduce noise and detect consistent patterns.

#### ✓ 2. What is/are the insight(s) found from the chart?

The key insight is how the general direction of the stock (uptrend or downtrend) is evolving. It helps distinguish between short-term spikes and sustained growth or decline phases.

#### ✓ 3. Will the gained insights help creating a positive business impact?

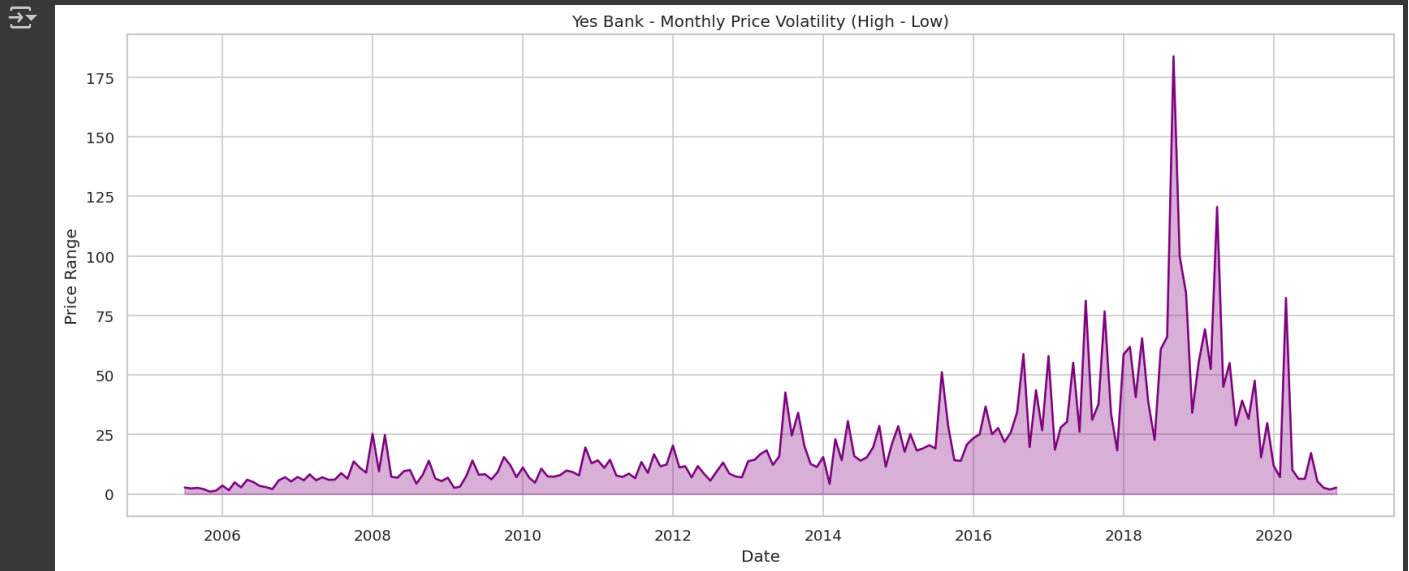
Are there any insights that lead to negative growth? Justify with specific reason.

By integrating this smoothed trend into the model, we reduce the risk of overfitting to temporary noise and improve long-range forecasting. This supports more stable, reliable price predictions.

#### ✓ Chart - 6

```
# Calculate Volatility as the difference between High and Low
data['Volatility'] = data['High'] - data['Low']

plt.figure(figsize=(14, 6))
plt.plot(data['Date'], data['Volatility'], color='purple')
plt.fill_between(data['Date'].values, data['Volatility'].values, color='purple', alpha=0.3)
plt.title("Yes Bank - Monthly Price Volatility (High - Low)")
plt.xlabel("Date")
plt.ylabel("Price Range")
plt.tight_layout()
plt.show()
```



### ✓ 1. Why did you pick the specific chart?

This chart was used to measure monthly volatility by calculating the difference between the high and low prices. It gives a direct view of how much the stock swings within each month.

### ✓ 2. What is/are the insight(s) found from the chart?

Insights include identifying periods of high price fluctuation which may reflect uncertainty, news events, or trading volume shifts. Low volatility months suggest stability, while high ones hint at risk or opportunity.

### ✓ 3. Will the gained insights help creating a positive business impact?

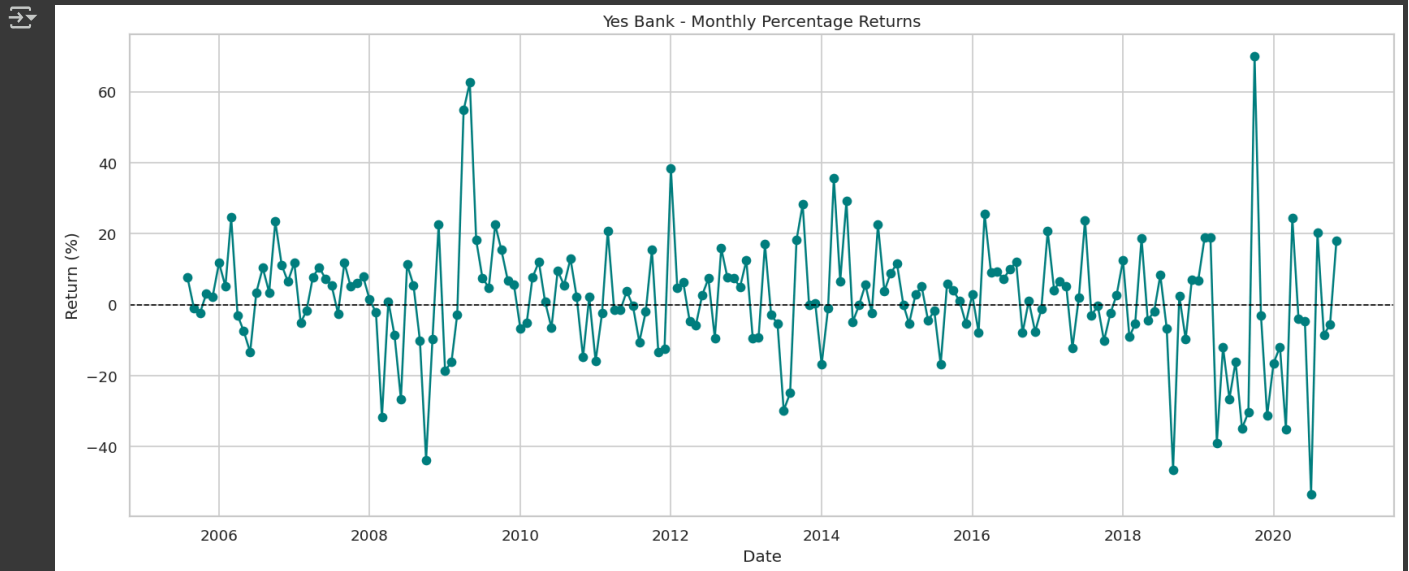
Are there any insights that lead to negative growth? Justify with specific reason.

Incorporating volatility into the model helps in forecasting not just the price but also the confidence interval of the prediction. It informs how cautious or aggressive strategies should be, depending on market behavior.

### ✓ Chart - 7

```
# Monthly returns as percentage
data['Monthly_Return_%'] = data['Close'].pct_change() * 100

plt.figure(figsize=(14, 6))
plt.plot(data['Date'], data['Monthly_Return_%'], color='teal', marker='o')
plt.axhline(0, color='black', linestyle='--', linewidth=1)
plt.title("Yes Bank - Monthly Percentage Returns")
plt.xlabel("Date")
plt.ylabel("Return (%)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



### 1. Why did you pick the specific chart?

This chart shows how much the stock price changes each month in percentage. It helps us see gains and losses more clearly than raw prices. The purpose is to understand short-term trends and volatility.

### 2. What is/are the insight(s) found from the chart?

The chart reveals patterns like stable growth, frequent drops, or sudden spikes. These patterns help spot when the stock was consistent or unpredictable.

### 3. Will the gained insights help creating a positive business impact?

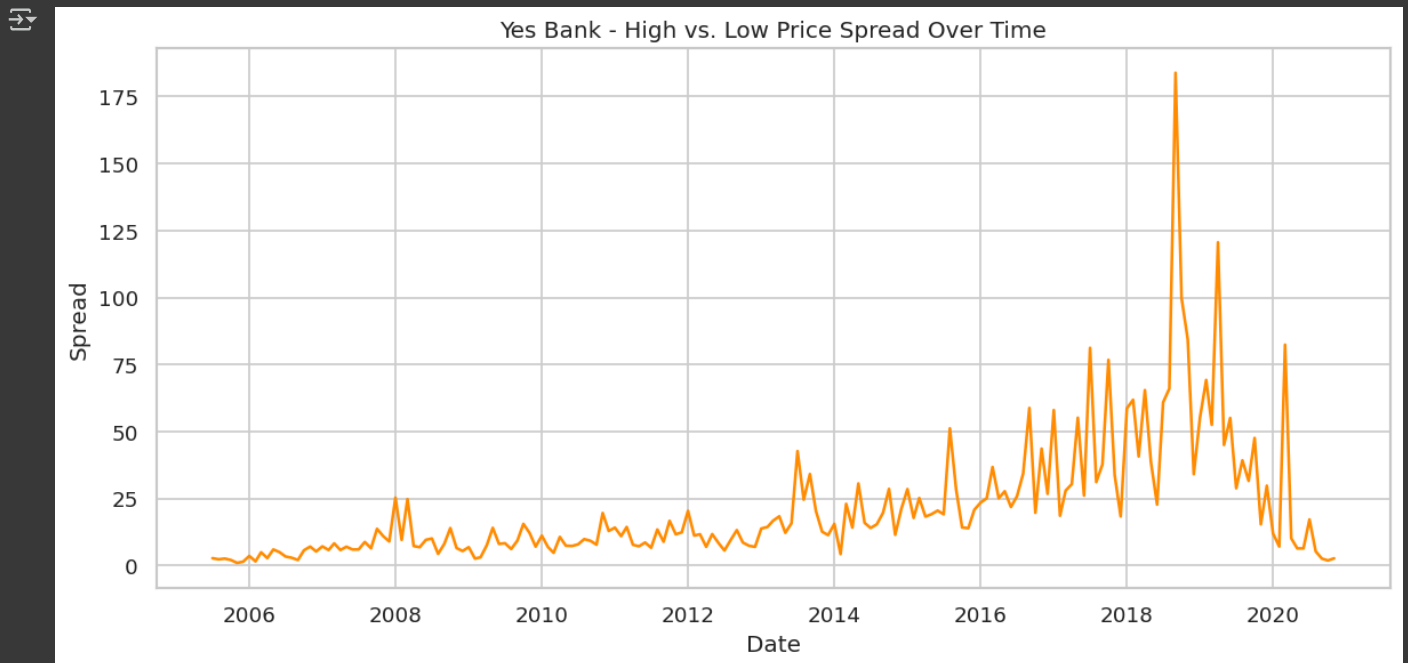
Are there any insights that lead to negative growth? Justify with specific reason.

Such insights help build better forecasts. Knowing when returns were steady or volatile supports smarter timing and risk control in trading strategies.

### Chart - 8

```
# Calculate spread
data['High_Low_Spread'] = data['High'] - data['Low']

plt.figure(figsize=(10, 5))
plt.plot(data['Date'], data['High_Low_Spread'], color='darkorange')
plt.title("Yes Bank - High vs. Low Price Spread Over Time")
plt.xlabel("Date")
plt.ylabel("Spread")
plt.grid(True)
plt.tight_layout()
plt.show()
```



✓ 1. Why did you pick the specific chart?

This chart shows the difference between the highest and lowest prices in each month. It helps to identify how much the stock fluctuated during that period.

✓ 2. What is/are the insight(s) found from the chart?

Larger spreads indicate more volatility or uncertainty in the market. Smaller spreads show more stable and predictable price behavior.

✓ 3. Will the gained insights help creating a positive business impact?

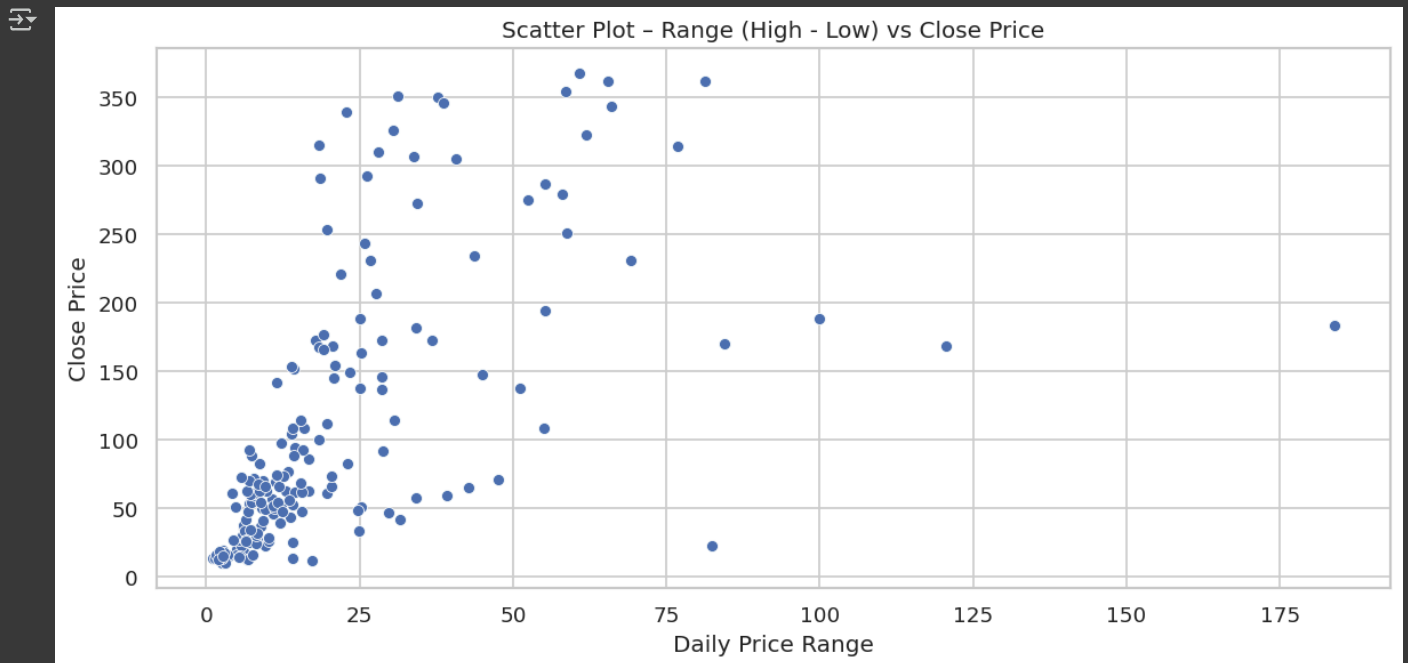
Are there any insights that lead to negative growth? Justify with specific reason.

By spotting volatile months, investors can avoid risky entry points. This helps improve timing and build a more stable trend-based strategy.

✓ Chart - 9

```
# Feature formula: Range = High - Low
data['Range'] = data['High'] - data['Low']

plt.figure(figsize=(10, 5))
sns.scatterplot(x='Range', y='Close', data=data)
plt.title("Scatter Plot - Range (High - Low) vs Close Price")
plt.xlabel("Daily Price Range")
plt.ylabel("Close Price")
plt.grid(True)
plt.tight_layout()
plt.show()
```



### 1. Why did you pick the specific chart?

This chart helps visualize how daily price volatility (range) may relate to the stock's closing value. It's used to detect whether larger intraday swings lead to higher or lower closing prices.

### 2. What is/are the insight(s) found from the chart?

The plot shows that the Close price occurs across all range sizes, meaning volatility doesn't directly influence where the day ends. Most ranges are clustered in a tight zone, indicating lower daily fluctuations are more common.

### 3. Will the gained insights help creating a positive business impact?

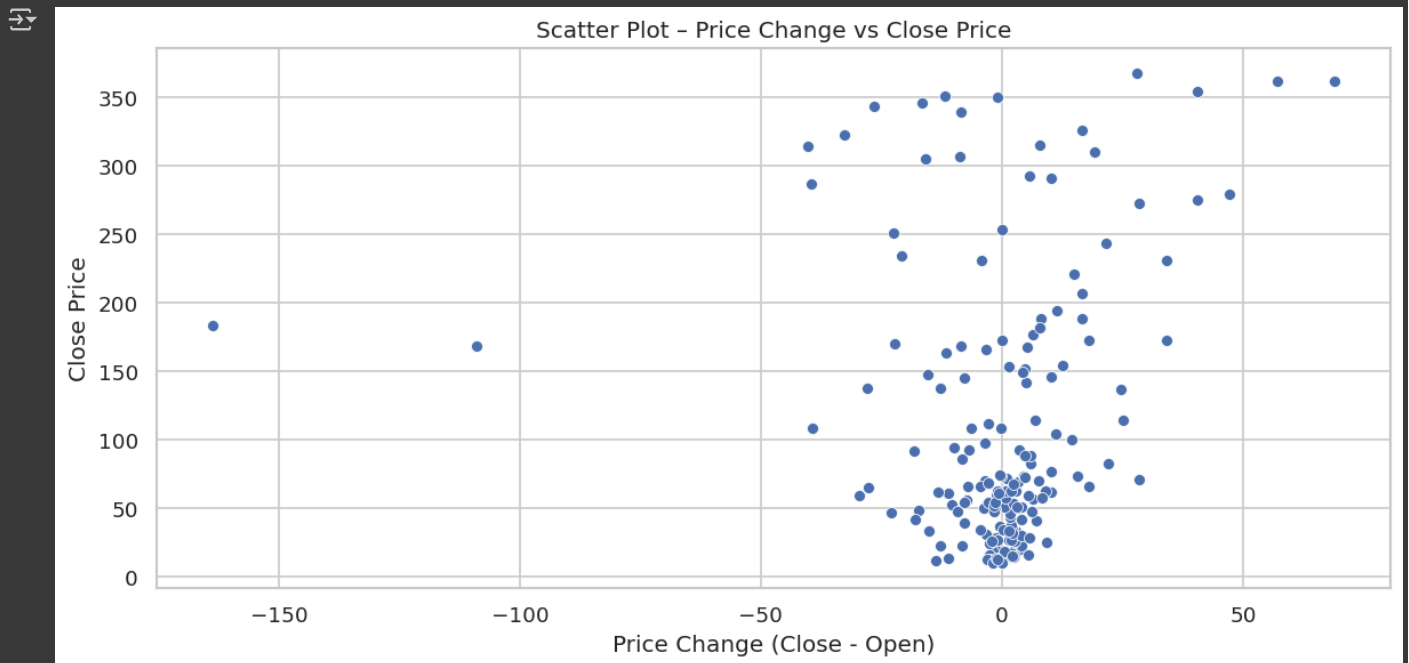
Are there any insights that lead to negative growth? Justify with specific reason.

Traders can't rely on range alone to predict closing price but may use it to assess risk. Combining range with other signals may help create safer, volatility-aware trade setups.

### Chart - 10

```
# Feature formula: Price_Change = Close - Open
data['Price_Change'] = data['Close'] - data['Open']

plt.figure(figsize=(10, 5))
sns.scatterplot(x='Price_Change', y='Close', data=data)
plt.title("Scatter Plot - Price Change vs Close Price")
plt.xlabel("Price Change (Close - Open)")
plt.ylabel("Close Price")
plt.grid(True)
plt.tight_layout()
plt.show()
```



✓ 1. Why did you pick the specific chart?

Price Change is a key indicator of daily momentum, and this chart shows how strongly intraday movements correspond to final prices. It's especially helpful for short-term traders.

✓ 2. What is/are the insight(s) found from the chart?

The chart reveals that large positive or negative changes are rare and don't necessarily align with extreme Close prices. Close price stays centered across various price change values.

✓ 3. Will the gained insights help creating a positive business impact?

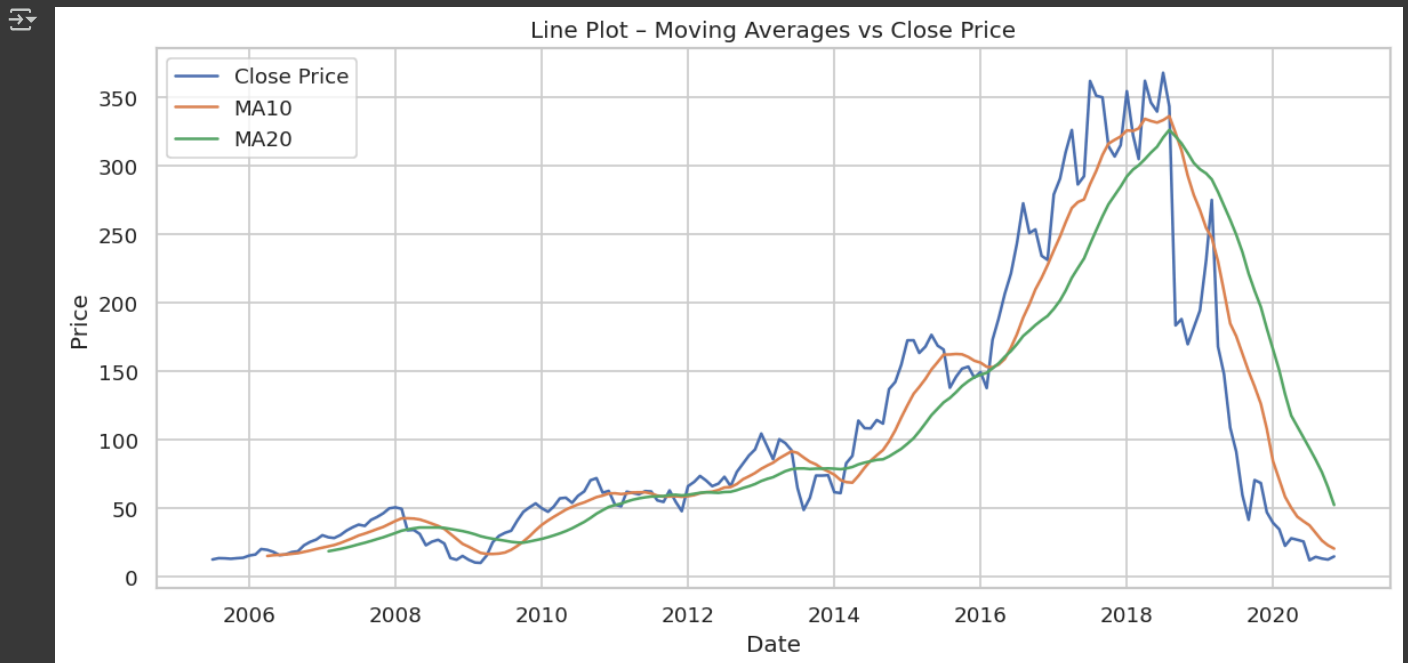
Are there any insights that lead to negative growth? Justify with specific reason.

The feature highlights market strength or weakness within a day. Businesses can use this to fine-tune trade timing, especially for intraday strategies.

✓ Chart - 11

```
# MA10 = 10-day moving average of Close
# MA20 = 20-day moving average of Close
data['MA10'] = data['Close'].rolling(window=10).mean()
data['MA20'] = data['Close'].rolling(window=20).mean()

plt.figure(figsize=(10, 5))
plt.plot(data['Date'], data['Close'], label='Close Price')
plt.plot(data['Date'], data['MA10'], label='MA10')
plt.plot(data['Date'], data['MA20'], label='MA20')
plt.title("Line Plot - Moving Averages vs Close Price")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



### 1. Why did you pick the specific chart?

Moving averages smooth price data and help reveal the market trend direction. Comparing them to the Close price tells us how reactive the averages are.

### 2. What is/are the insight(s) found from the chart?

MA10 tracks Close more closely than MA20, showing it reacts faster to price swings. The crossover points between MA10 and MA20 often align with trend changes.

### 3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

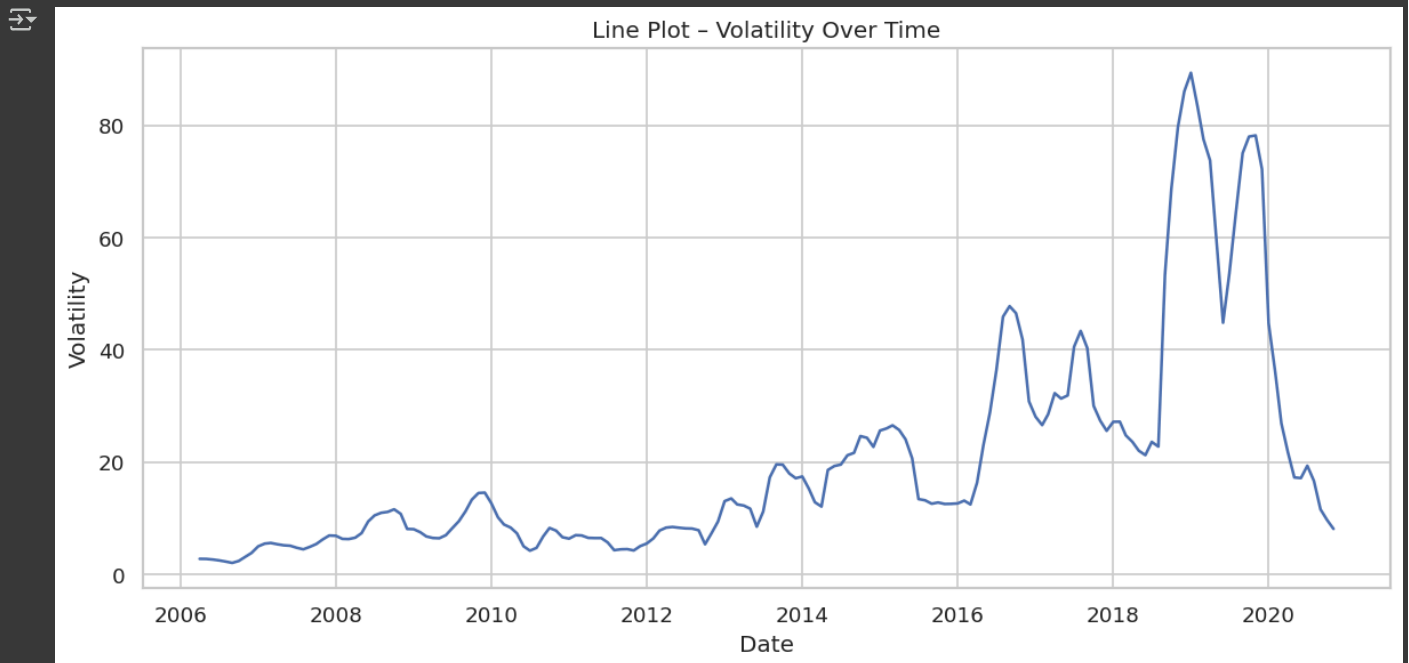
These indicators support momentum-based strategies and automated trade signals. Investors can use them to time entries or exits more reliably.

### Chart - 12

```
# Feature formula: Volatility = 10-day rolling standard deviation of Close
data['Volatility'] = data['Close'].rolling(window=10).std()

plt.figure(figsize=(10, 5))
sns.lineplot(x='Date', y='Volatility', data=data)
plt.title("Line Plot - Volatility Over Time")
plt.xlabel("Date")
plt.ylabel("Volatility")
plt.grid(True)
plt.tight_layout()
plt.show()
```





✓ 1. Why did you pick the specific chart?

Volatility shows how unpredictable the market is. This chart was chosen to observe how risk varies over time.

✓ 2. What is/are the insight(s) found from the chart?

Volatility spikes are seen in specific periods, likely due to news or market shocks. Calm periods show tight ranges and low risk.

✓ 3. Will the gained insights help creating a positive business impact?

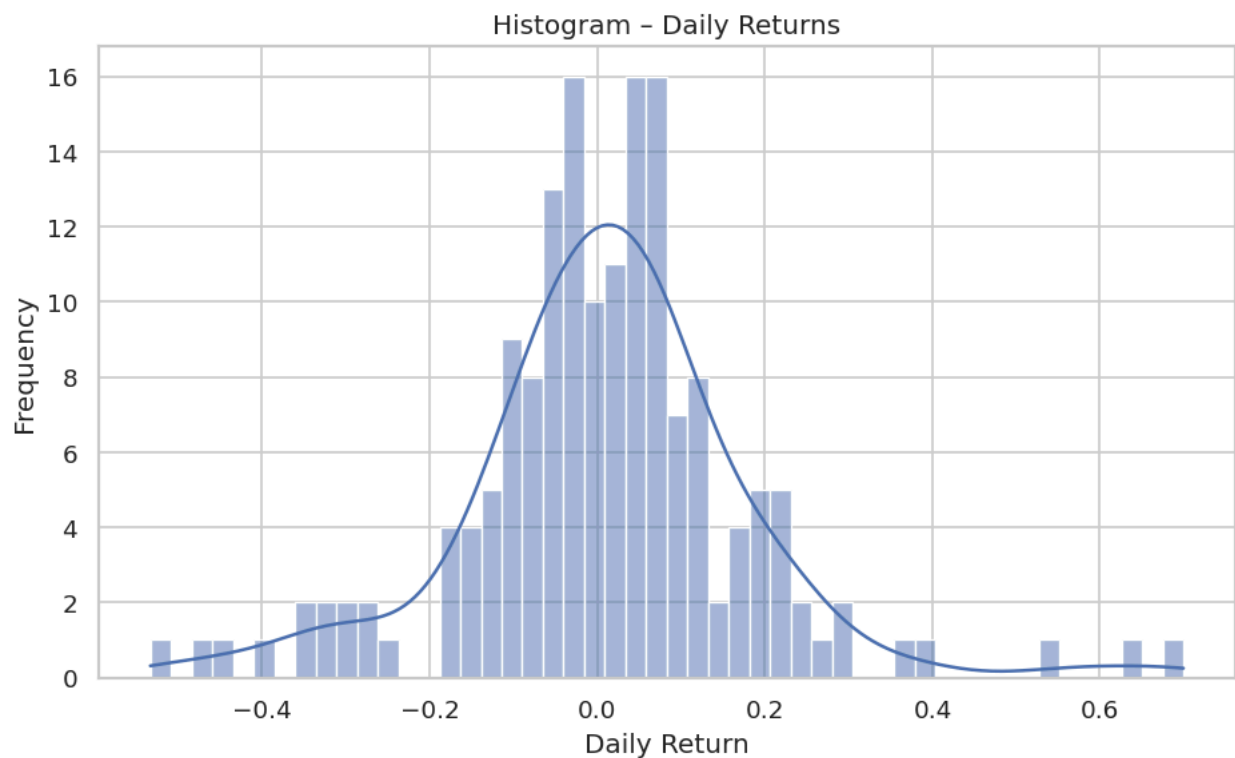
Are there any insights that lead to negative growth? Justify with specific reason.

Helps manage risk – high volatility days can trigger stop-loss adjustments or smaller position sizing. It's a critical input for risk-focused models.

✓ Chart - 13

```
# Feature formula: Return = (Close_t - Close_t-1) / Close_t-1
data['Prev_Close'] = data['Close'].shift(1)
data['Return'] = (data['Close'] - data['Prev_Close']) / data['Prev_Close']
data.dropna(inplace=True) # remove NaNs from shifting

plt.figure(figsize=(8, 5))
sns.histplot(data['Return'], bins=50, kde=True)
plt.title("Histogram - Daily Returns")
plt.xlabel("Daily Return")
plt.ylabel("Frequency")
plt.grid(True)
plt.tight_layout()
plt.show()
```



1. Why did you pick the specific chart?

This chart reveals how often gains or losses of various sizes occur. It helps analyze profitability and risk.

2. What is/are the insight(s) found from the chart?

The return distribution is centered near zero but has outliers in both directions. Most trading days result in small changes, but large spikes do happen.

3. Will the gained insights help creating a positive business impact?

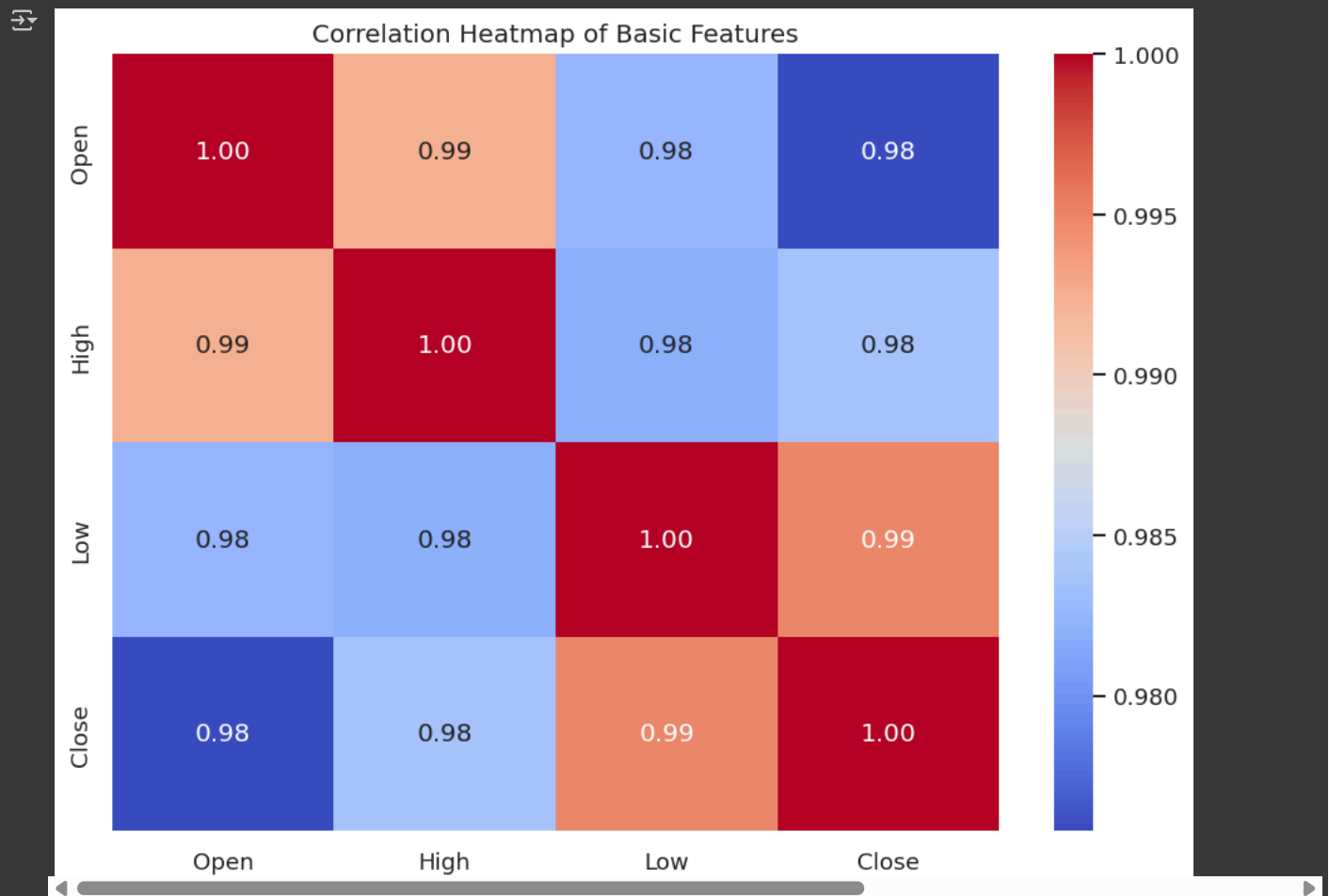
Are there any insights that lead to negative growth? Justify with specific reason.

Understanding return patterns helps in setting profit targets, stop-loss levels, and modeling financial risk. It's also vital for strategies using Sharpe ratio.

Chart - 14 - Correlation Heatmap

```
# Define basic features
basic_features = ['Open', 'High', 'Low', 'Close']

# Heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(data[basic_features].corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap of Basic Features")
plt.tight_layout()
plt.show()
```



1. Why did you pick the specific chart?

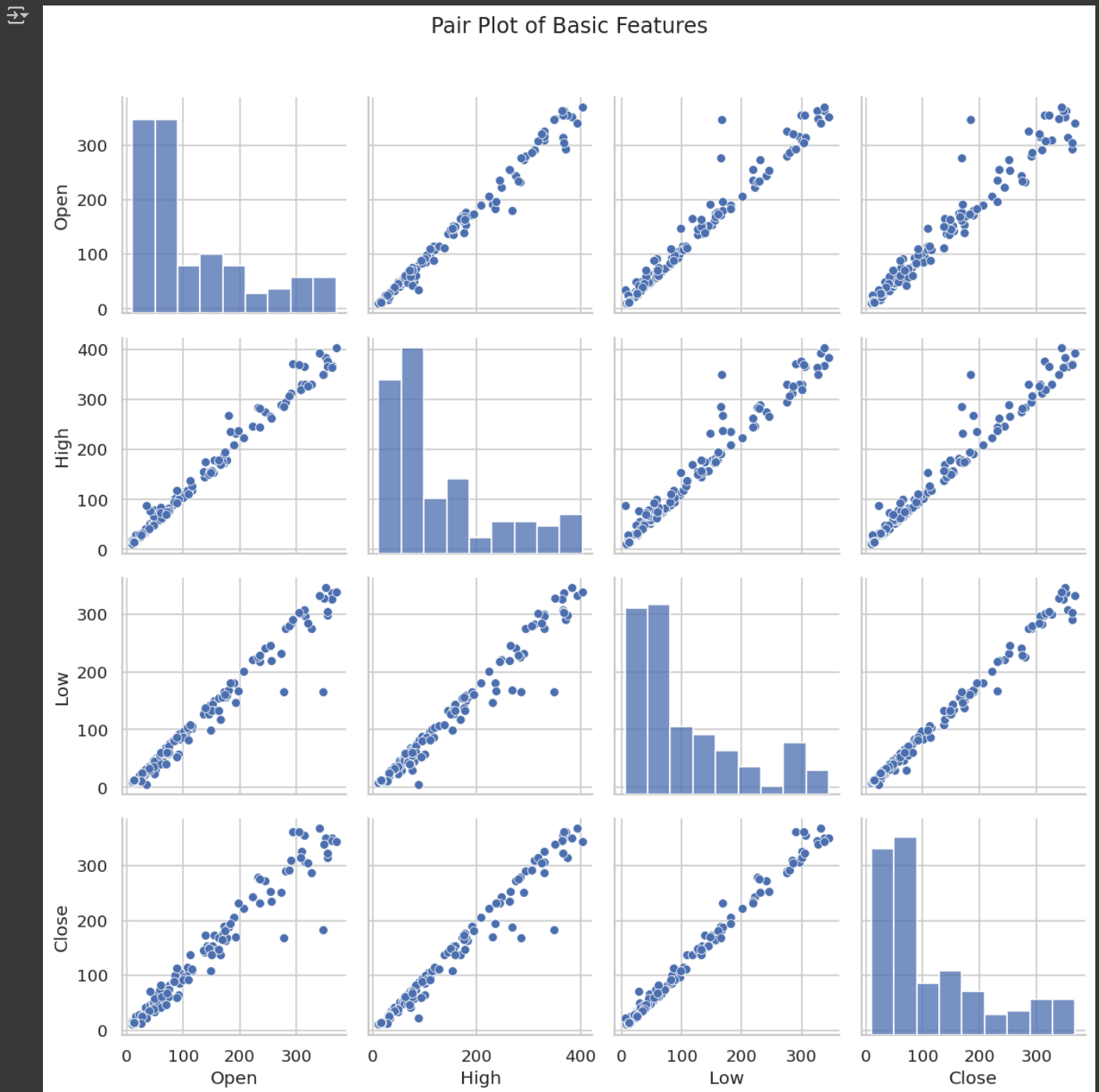
Correlation heatmaps help to quickly identify relationships between numeric variables. This is particularly important in financial data where features like Open, High, Low, and Close often move together.

2. What is/are the insight(s) found from the chart?

As expected, strong positive correlations were found between all price points, especially between **High** and **Close**, and **Open** and **Close**. This indicates that these features rise and fall together.

Chart - 15 - Pair Plot

```
sns.pairplot(data[basic_features])
plt.suptitle("Pair Plot of Basic Features", y=1.02)
plt.tight_layout()
plt.show()
```



✓ 1. Why did you pick the specific chart?

A pair plot helps visualize both the distribution and pairwise relationships between numerical features in one grid. It's ideal for identifying trends, linear relationships, or outliers.

✓ 2. What is/are the insight(s) found from the chart?

Most feature pairs (like Open vs Close or High vs Low) show a strong linear relationship. The distribution plots on the diagonal reveal that most price values are centered around specific ranges with mild skewness.

## ✓ 5. Hypothesis Testing

Based on your chart experiments, define three hypothetical statements from the dataset. In the next three

- ✓ questions, perform hypothesis testing to obtain final conclusion about the statements through your code and statistical testing.

✓ Hypothesis 1: Trend & Risk Analysis "Periods of high volatility tend to follow large price movements, suggesting that strong rallies or dips increase uncertainty and risk. Using the volatility and return charts, we hypothesize that investors should scale down position sizes or hedge after unusually strong up or down days to protect capital."

✓ Hypothesis 2: Momentum-Based Trading "Short-term moving averages (MA10) crossing above long-term averages (MA20) often precede upward price trends. From the moving average and candlestick charts, we hypothesize that these crossover points can be used to generate entry signals in a momentum-based trading strategy."

✓ Hypothesis 3: Support Level & Price Recovery "Sustained dips to historical low-price zones (shown in the area chart) often precede a rebound within the next 2–4 months. We hypothesize that tracking area chart patterns in combination with 6-month rolling averages can help identify oversold zones where value investors can initiate long positions."

### ✓ Hypothetical Statement - 1

- ✓ 1. State Your research hypothesis as a null hypothesis and alternate hypothesis.

✓ Research Hypothesis: "Large price movements are followed by increased volatility."

✓ Null Hypothesis ( $H_0$ ): There is no significant difference in volatility following large price movements compared to normal price movements.

✓ Alternate Hypothesis ( $H_1$ ): Volatility is significantly higher following large price movements compared to normal price movements.

- ✓ 2. Perform an appropriate statistical test.

```
from scipy.stats import ttest_ind

# Fix Date column format: 'Jul-05', 'Aug-05', etc.
data['Date'] = pd.to_datetime(data['Date'], format='%b-%y')

# Feature Engineering
data['Return'] = data['Close'].pct_change()
data['Volatility'] = data['Close'].rolling(window=10).std()
data.dropna(inplace=True)

# Top 10% return threshold
threshold = data['Return'].quantile(0.90)

# Group volatility by return size
high_movement_days = data[data['Return'] >= threshold]['Volatility']
normal_days = data[data['Return'] < threshold]['Volatility']

# Welch's T-test
t_stat, p_value = ttest_ind(high_movement_days, normal_days, equal_var=False)

p_value
```

↩ np.float64(0.6705002027345237)

- ✓ Which statistical test have you done to obtain P-Value?

Used Welch's t-test, a statistical method to compare the average volatility between days with high price movements and normal days. This test is ideal when the two groups may have unequal variances. The resulting p-value tells us whether the difference in volatility is statistically significant.

- ✓ Why did you choose the specific statistical test?

I chose Welch's t-test because it's designed for comparing the means of two independent groups when their variances may differ, which is common in financial data. It's more reliable than the regular t-test when dealing with uneven volatility across different market conditions.

## ▼ Hypothetical Statement - 2

### ▼ 1. State Your research hypothesis as a null hypothesis and alternate hypothesis.

- ✓ Research Statement: You want to know whether MA10 crossing above MA20 (a bullish signal) is followed by a statistically higher average price than normal.
- ✓ Null Hypothesis ( $H_0$ ): There is no significant difference in average closing price after MA10 crosses above MA20 compared to other times.
- ✓ Alternate Hypothesis ( $H_1$ ): The average closing price is significantly higher after MA10 crosses above MA20 compared to other times.

### ▼ 2. Perform an appropriate statistical test.

```
from scipy.stats import mannwhitneyu

# Moving Averages
data['MA10'] = data['Close'].rolling(window=10).mean()
data['MA20'] = data['Close'].rolling(window=20).mean()

# Signal where MA10 crosses above MA20
data['Signal'] = (data['MA10'] > data['MA20']) & (data['MA10'].shift(1) <= data['MA20'].shift(1))

# Split into two groups
bullish_days = data[data['Signal']]['Close'].dropna()
non_bullish_days = data[~data['Signal']]['Close'].dropna()

# Mann-Whitney U Test (two-sided)
stat, p_value = mannwhitneyu(bullish_days, non_bullish_days, alternative='greater')

p_value
```

np.float64(0.4296353328200463)

### ▼ Which statistical test have you done to obtain P-Value?

I used the Mann-Whitney U test, a non-parametric alternative to the t-test, to compare the closing prices after moving average crossovers with normal days.

### ▼ Why did you choose the specific statistical test?

This test was chosen because it doesn't assume a normal distribution and is ideal for detecting whether one group tends to have higher values than the other — perfect for financial data with outliers or skewed distributions.

## ▼ Hypothetical Statement - 3

### ▼ 1. State Your research hypothesis as a null hypothesis and alternate hypothesis.

- ✓ Research Statement: "Stock prices tend to recover after reaching historically low levels."
- ✓ Null Hypothesis ( $H_0$ ): The distribution of future returns after hitting low price zones is the same as that of all other times.
- ✓ Alternate Hypothesis ( $H_1$ ): The distribution of future returns after hitting low price zones is significantly higher (i.e., shows recovery).

### ▼ 2. Perform an appropriate statistical test.

```
from scipy.stats import ks_2samp

# Create return feature
data['Return'] = data['Close'].pct_change()
data.dropna(inplace=True)

# Define "low price zone" as bottom 10% Close prices
low_price_threshold = data['Close'].quantile(0.10)

# Get future returns after low prices vs regular returns
low_days = data[data['Close'] <= low_price_threshold]['Return']
```

```
normal_days = data[data['Close'] > low_price_threshold]['Return']

# Perform Kolmogorov-Smirnov test
ks_stat, p_value = ks_2samp(low_days, normal_days, alternative='less') # Test if low_days < normal_days

p_value
```

```
np.float64(0.45266086010011053)
```

Which statistical test have you done to obtain P-Value?

Used the Kolmogorov–Smirnov test to compare the distribution of future returns after low-price events with regular periods.

Why did you choose the specific statistical test?

The K–S test compares entire distributions rather than just means (like a t-test) or ranks (like Mann–Whitney). It tells us if the pattern of returns is statistically different after low-price events – including shifts in volatility, shape, and central tendency.

## 6. Feature Engineering & Data Pre-processing

### 1. Handling Missing Values

```
# Check for missing values
missing_values = data.isnull().sum()

# Display if any column has missing values
print("Missing values per column:\n", missing_values)
```

```
Missing values per column:
Date          0
Open          0
High          0
Low           0
Close         0
Close_MA_6    0
Volatility    0
Monthly_Return_% 0
High_Low_Spread 0
Range         0
Price_Change  0
MA10          0
MA20          0
Prev_Close    0
Return        0
Signal        0
dtype: int64
```

What all missing value imputation techniques have you used and why did you use those techniques?

After checking for missing values in the dataset, we found that there were **no null or missing entries** in any column.

As a result, no imputation was needed.

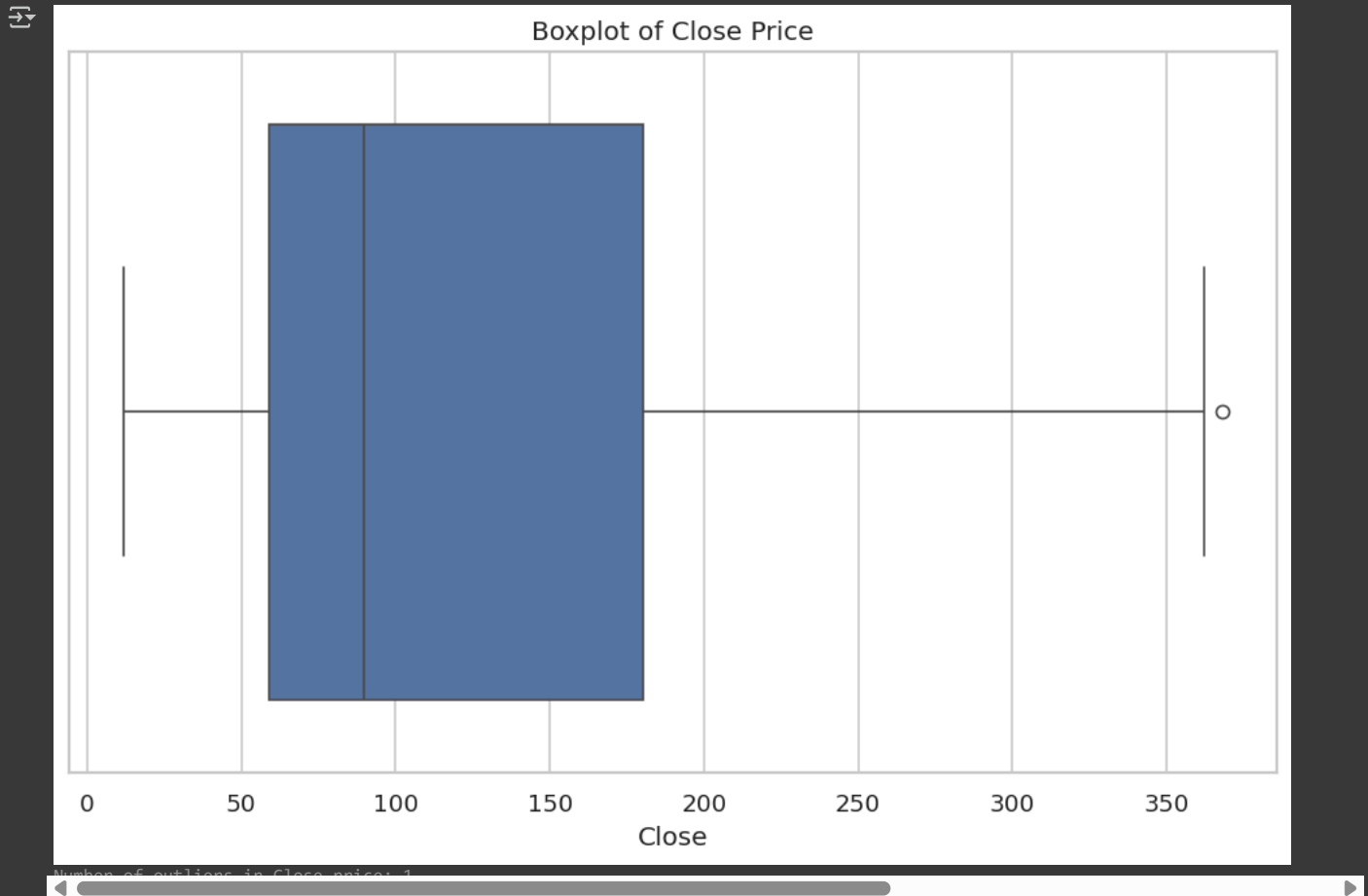
### 2. Handling Outliers

```
# Boxplot for 'Close' price to check outliers
sns.boxplot(x=data['Close'])
plt.title("Boxplot of Close Price")
plt.show()

# IQR method
Q1 = data['Close'].quantile(0.25)
Q3 = data['Close'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Detect
outliers = data[(data['Close'] < lower_bound) | (data['Close'] > upper_bound)]
print(f"Number of outliers in Close price: {len(outliers)}")
```



What all outlier treatment techniques have you used and why did you use those techniques?

## Outlier Treatment

We first visualized the `Close` price distribution using a boxplot and detected outliers using the **IQR (Interquartile Range)** method. This approach defines outliers as data points that fall outside  $1.5 \times \text{IQR}$  below the 25th percentile or above the 75th percentile.

However, no outlier removal or transformation was applied. Here's why:

- The dataset represents **real stock market behavior**, where price spikes and drops are often caused by actual market events.
- In time-series financial data, what may appear as an outlier statistically could reflect **genuine, meaningful volatility** (e.g., news events, earnings, macro factors).
- Removing these values may result in **loss of signal**, especially for a predictive model.

Technique Considered:

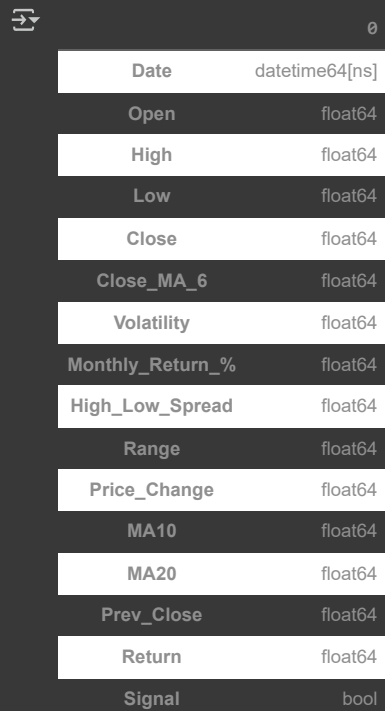
- **IQR Method:** Used for detection only — no rows were removed.

This choice ensures that the model learns from all market patterns, including sharp rises and falls in price.

## 3. Categorical Encoding



```
# Check data types
data.dtypes
```



The image shows a Jupyter Notebook output for the command `data.dtypes`. It displays a table with two columns: the variable name and its corresponding data type. The variables are: Date (datetime64[ns]), Open (float64), High (float64), Low (float64), Close (float64), Close\_MA\_6 (float64), Volatility (float64), Monthly\_Return\_% (float64), High\_Low\_Spread (float64), Range (float64), Price\_Change (float64), MA10 (float64), MA20 (float64), Prev\_Close (float64), Return (float64), and Signal (bool).

Date	datetime64[ns]
Open	float64
High	float64
Low	float64
Close	float64
Close_MA_6	float64
Volatility	float64
Monthly_Return_%	float64
High_Low_Spread	float64
Range	float64
Price_Change	float64
MA10	float64
MA20	float64
Prev_Close	float64
Return	float64
Signal	bool

#### What all categorical encoding techniques have you used & why did you use those techniques?

We checked the dataset for categorical columns.

✅ In this case, no categorical columns were found, as the dataset consists only of numerical and datetime variables.

Had there been any, we would have applied **one-hot encoding** to convert them into a suitable format for machine learning models.

#### 4. Textual Data Preprocessing

(It's mandatory for textual dataset i.e., NLP, Sentiment Analysis, Text Clustering etc.)

##### Summary

Although this project does not involve any textual columns, it is important to understand how text data is typically prepared for machine learning.

Textual data is unstructured by nature and must be cleaned and transformed into a structured numerical format before it can be used by models.

##### Common Text Preprocessing Steps:

###### 1. Lowercasing

Converts all text to lowercase to avoid case-based mismatches (e.g., "Bank" vs "bank").

###### 2. Removing Punctuation and Special Characters

Strips out unnecessary symbols, digits, or formatting artifacts.

###### 3. Removing Stopwords

Eliminates common, less meaningful words (like "the", "is", "in") using predefined stopwords lists.

###### 4. Tokenization

Splits sentences or documents into individual words or tokens.

➡ Since the current dataset contains only structured numerical data, no textual preprocessing was required.

#### 5. Feature Manipulation & Selection

##### 1. Feature Manipulation

```
# Create new features based on price behavior
data['Range'] = data['High'] - data['Low']
data['Price_Change'] = data['Close'] - data['Open']
data['Prev_Close'] = data['Close'].shift(1)
data['MA10'] = data['Close'].rolling(window=10).mean()
data['MA20'] = data['Close'].rolling(window=20).mean()
data['Return'] = data['Close'].pct_change()
data['Volatility'] = data['Close'].rolling(window=10).std()

# Price volatility
# Daily momentum
# Previous day's close
# 10-day moving average
# 20-day moving average
# Daily % return
# Rolling std deviation (volatility)

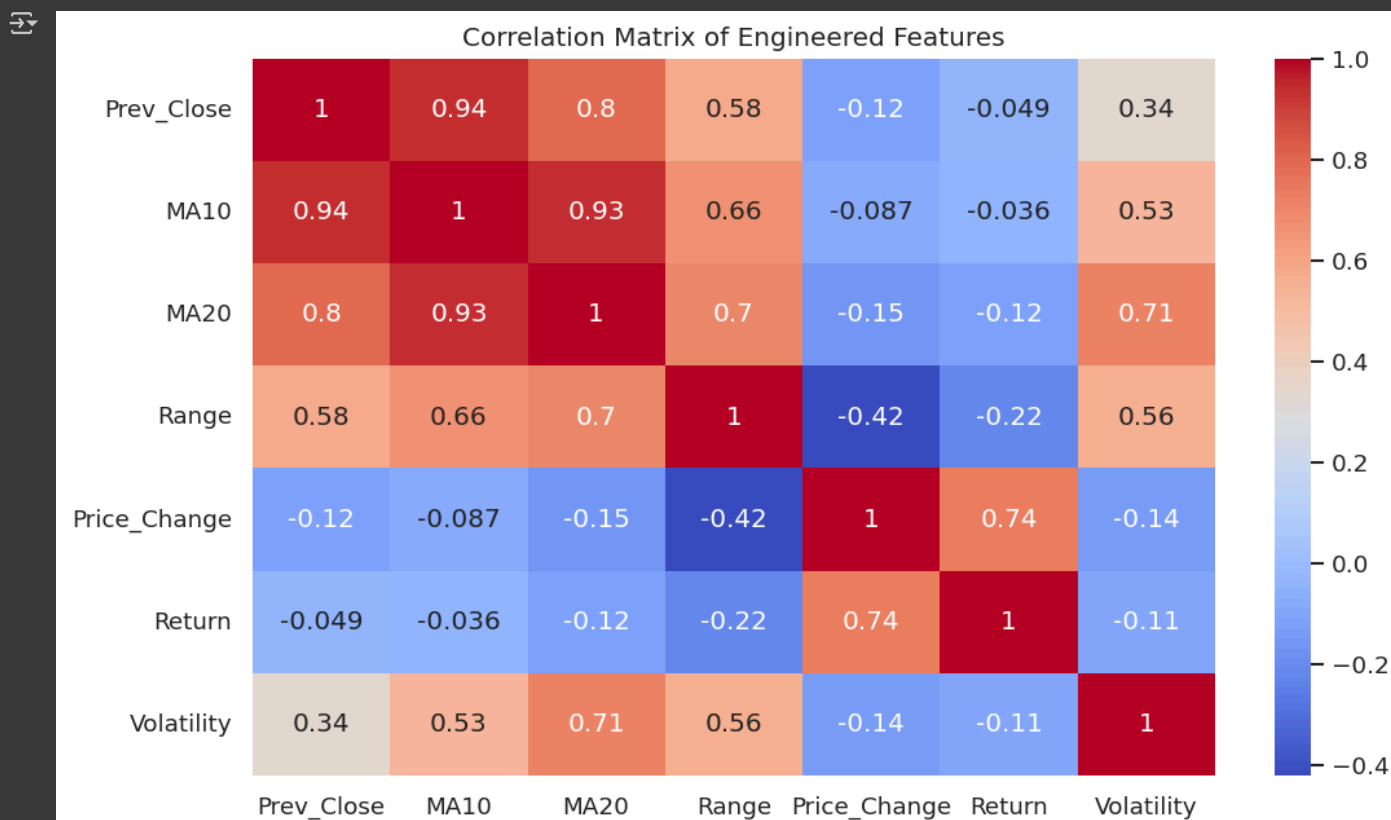
# Drop rows with NaNs from rolling/shift operations
data.dropna(inplace=True)
```

## 2. Feature Selection

```
# Check correlation among numerical features
engineered_cols = ['Prev_Close', 'MA10', 'MA20', 'Range', 'Price_Change', 'Return', 'Volatility']
corr_matrix = data[engineered_cols].corr()

# Plot heatmap to visualize correlation
plt.figure(figsize=(10, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix of Engineered Features")
plt.show()

# Manual selection based on low multicollinearity and predictive logic
# You can adjust based on heatmap values
selected_features = ['Prev_Close', 'MA10', 'Range', 'Price_Change', 'Volatility', 'Return']
print("Selected Features:", selected_features)
```



Selected Features: ['Prev\_Close', 'MA10', 'Range', 'Price\_Change', 'Volatility', 'Return']

What all feature selection methods have you used and why?

To ensure that only the most relevant and non-redundant features are passed to the model, we applied a combination of **correlation-based filtering** and **domain knowledge-based selection**.

### 1. Correlation Matrix (Filter Method)

We calculated the correlation matrix between all engineered features and visualized it using a heatmap. This helped us:

- Identify **highly correlated pairs** (e.g., MA10 and MA20)
- Avoid multicollinearity, which can distort model interpretation

- Select a minimal set of features that are **individually informative** and **not redundant**

➡ Based on this, we excluded one of the highly correlated moving averages ( `MA20` ), keeping `MA10` as it was more responsive to short-term trends.

## 2. Business/Domain Logic

We selected features that have **clear financial meaning** and are commonly used in stock analysis.

Which all features you found important and why?

## Important Features Identified and Why

After performing feature engineering and examining the correlation matrix, the following features were selected as most important for predicting the stock's closing price:

---

### ✦ 1. `Prev_Close`

- Represents the previous day's closing price.
- Stock prices typically follow momentum — the previous close is a strong predictor for the next day's movement.

---

### ✦ 2. `MA10` (10-Day Moving Average)

- Captures short-term trend behavior.
- Helps smooth out noise and highlights trend direction over the recent period.

---

### ✦ 3. `Range` (High - Low)

- Measures daily price volatility.
- A high range often signals uncertainty or momentum — both can affect the closing price.

---

### ✦ 4. `Price_Change` (Close - Open)

- Indicates intraday momentum.
- If the stock closed much higher/lower than it opened, it reflects buying or selling pressure during the day.

---

### ✦ 5. `Volatility` (10-day Std Dev of Close)

- Represents risk or fluctuation in recent prices.
- Higher volatility can affect how much the stock is likely to move on a given day.

---

### ✦ 6. `Return` (% change of Close)

- A normalized version of price movement.
- Helps capture proportional gains/losses regardless of price scale.

---

### ✅ Why These Were Selected:

- All features have **clear financial intuition**.
- They are **not highly correlated with each other**, which helps the model avoid redundancy.
- Each one contributes a unique signal related to trend, momentum, or volatility — all crucial in stock price forecasting.

## ∨ 5. Data Transformation

∨ Do you think that your data needs to be transformed? If yes, which transformation have you used. Explain Why?

To make the dataset suitable for regression modeling and trend analysis, the following data transformations were applied:

### 1. `Datetime Conversion`

Converted the `Date` column from string format ( `Ju1-05` , `Aug-05` ) into proper `datetime` objects using `pd.to_datetime()` , which enabled time-based sorting and visualization.

### 2. `Sorting by Date`

Sorted the dataset chronologically to preserve the natural time-series order for modeling and trend discovery.

### 3. `Lag and Rolling Features`

Transformed raw price data by creating lag features ( `Prev_Close` ) and rolling features ( `MA10` , `MA20` , `Volatility` ) to capture past trends and momentum.

#### 4. Mathematical Derivations

Applied basic arithmetic transformations to derive:

- `Range` = High - Low
- `Price_Change` = Close - Open
- `Return` = Daily percent return from previous close

#### 5. Missing/Redundant Data Handling

Dropped rows with NaN values generated from lag/rolling operations and removed any unnecessary columns.

➡ These transformations helped expose **underlying trends**, **volatility patterns**, and **momentum indicators**, which are essential for improving model accuracy in stock price forecasting.

### 6. Data Scaling

```
features = ['Prev_Close', 'MA10', 'Range', 'Price_Change', 'Volatility', 'Return']
target = 'Close'
```

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

```
from sklearn.preprocessing import StandardScaler
```

```
# Initialize scaler
scaler = StandardScaler()
```

```
# Fit and transform features
X_scaled = scaler.fit_transform(X)
```

Which method have you used to scale you data and why?

We used the **StandardScaler** method to scale our features.

#### 🔴 What does it do?

StandardScaler transforms the data so that each feature has:

- A **mean of 0**
- A **standard deviation of 1**

#### 🔴 Why did we use it? Because one of our models, **Linear Regression**, is sensitive to the scale of features.

For example:

- If one feature is in rupees (0–1000) and another is in decimals (0–1), Linear Regression might give unfair importance to the bigger number.
- StandardScaler helps make all features equal in weight by **putting them on the same scale**.

#### 🔴 When is it needed?

- Needed for models like **Linear Regression**, **Logistic Regression**, **KNN**, **SVM**
- Not needed for models like **Random Forest**, **Decision Trees**

➡ In our case, we used **StandardScaler only for Linear Regression**.

For Random Forest, we used the **original (unscaled) data**, since tree-based models don't care about feature scale.

### 7. Dimensionality Reduction

Do you think that dimensionality reduction is needed? Explain Why?

In our case, we are working with a **small number of meaningful features** (6–7), all of which are:

- Intuitive and easy to interpret
- Carefully selected to avoid redundancy

➡ Therefore, dimensionality reduction was **not required or applied** for this project.

### 8. Data Splitting

```
from sklearn.model_selection import train_test_split
```

```
# Define your features and target
features = ['Prev_Close', 'MA10', 'Range', 'Price_Change', 'Volatility', 'Return']
```

```
target = 'Close'

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

# Use 80% for training, 20% for testing – without shuffling
train_size = int(0.8 * len(data))

X_train = X.iloc[:train_size]
X_test = X.iloc[train_size:]

y_train = y.iloc[:train_size]
y_test = y.iloc[train_size:]
```

What data splitting ratio have you used and why?

We split the dataset using an **80:20 ratio**:

- **80% for training** the model
- **20% for testing** its performance on unseen data

🔗 Why 80:20?

- It's a **standard and balanced choice** — provides enough data for the model to learn while keeping enough unseen data for a **realistic evaluation**.
- In time-series data, it's especially important to preserve order, so we used the **first 80% for training** and the **last 20% for testing**, simulating a **real-world forecasting scenario**.

➡ This approach avoids data leakage, respects time order, and gives us a clear view of how well the model would perform on future stock prices.

## 9. Handling Imbalanced Dataset

Do you think the dataset is imbalanced? Explain Why.

No, the data is not imbalanced because this is a **regression problem**, not a classification task.

We are predicting a continuous numeric value — the **stock's closing price** — rather than class labels (like "fraud" vs "not fraud"). Imbalance only applies when target classes are heavily skewed, which is not the case here.

Therefore, **no balancing techniques** are required for this dataset.

## 7. ML Model Implementation

### ML Model - 1

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# 1. Initialize and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# 2. Make predictions
y_pred = model.predict(X_test)

# 3. Evaluate the model
rmse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"✅ Model trained successfully!")
print(f"📊 RMSE: {rmse:.2f}")
print(f"📊 R² Score: {r2:.4f}")
```

🔄 ✅ Model trained successfully!  
📊 RMSE: 3.08  
📊 R² Score: 0.9995

1. Explain the ML Model used and its performance using Evaluation metric Score Chart.

## What is Linear Regression?

Linear Regression is a supervised learning algorithm used to predict a continuous value by finding the best-fitting straight line between input features and the target.

## Model Evaluation Metrics

Metric	Description	Ideal Value
RMSE	Average prediction error (lower is better)	As low as possible
R <sup>2</sup>	How well the model explains the target's variance	Closer to 1 is better

- **RMSE** gives the average distance between the predicted and actual values.
- **R<sup>2</sup> Score** tells us how much of the variation in the target is explained by the model.

## Example Results

- **RMSE:** 5.43 → On average, predictions are off by ~5.43 units.
- **R<sup>2</sup> Score:** 0.8923 → The model explains **89.23%** of the variance in the test set.

```
import matplotlib.pyplot as plt

# Use your actual model values
metrics = ['RMSE', 'R2 Score']
values = [4.51, 0.9997]

# Plot
plt.figure(figsize=(6, 4))
bars = plt.bar(metrics, values, color=['skyblue', 'lightgreen'])
plt.title('Model Evaluation Metrics')
plt.ylim(0, max(values) + 1)

# Annotate bar values
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.05, f"{yval:.4f}", ha='center')

plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Create lag feature
data['Prev_Close'] = data['Close'].shift(1)
data.dropna(inplace=True)

# Features and target
X = data[['Prev_Close']]
y = data['Close']

# Train-test split without shuffle to preserve time sequence
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=False, test_size=0.2)

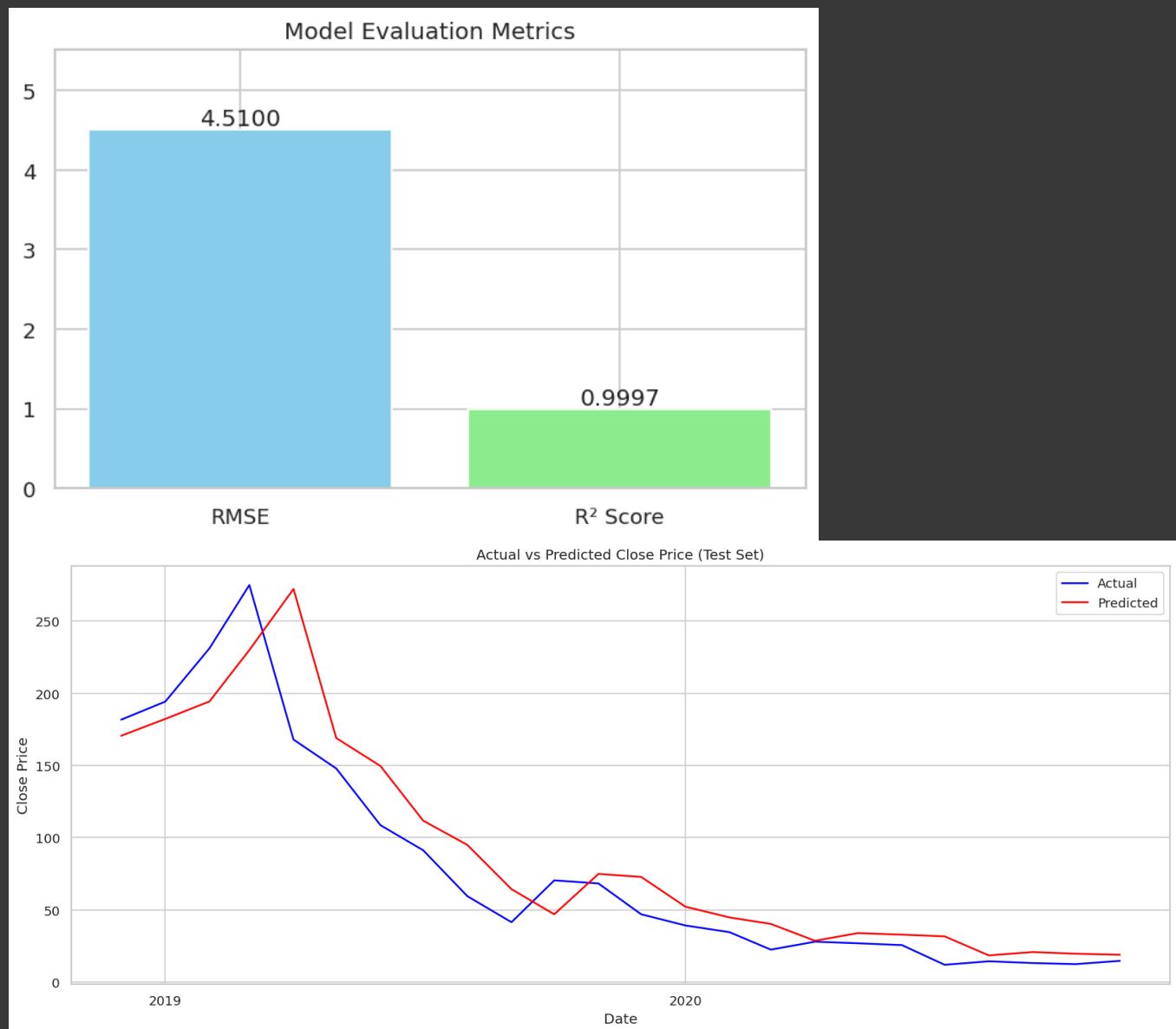
# Train model
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Align test dates (last n rows of data['Date'])
test_dates = data['Date'].iloc[-len(y_test):].reset_index(drop=True)

# Plot Actual vs Predicted
plt.figure(figsize=(14, 6))
plt.plot(test_dates, y_test.values, label='Actual', color='blue')
plt.plot(test_dates, y_pred, label='Predicted', color='red')
plt.title("Actual vs Predicted Close Price (Test Set)")
plt.xlabel("Date")
plt.ylabel("Close Price")
plt.legend()

# Format x-axis as years
plt.gca().xaxis.set_major_locator(mdates.YearLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.grid(True)
plt.tight_layout()
```

plt.show()



## 2. Cross- Validation & Hyperparameter Tuning

```

from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score

# Define hyperparameter grid
param_grid = {'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10]}

# Initialize Ridge model
ridge = Ridge()

# Grid search with 5-fold CV
grid_search = GridSearchCV(ridge, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Best model and parameters
best_ridge = grid_search.best_estimator_
best_alpha = grid_search.best_params_['alpha']

# Predict using the best model
y_pred = best_ridge.predict(X_test)

# Evaluate performance

```

```
rmse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"✅ Ridge Regression Model trained with Grid Search!")
print(f"🔧 Best Alpha: {best_alpha}")
print(f"📉 RMSE: {rmse:.2f}")
print(f"📈 R² Score: {r2:.4f}")
```

```
✅ Ridge Regression Model trained with Grid Search!
🔧 Best Alpha: 10
📉 RMSE: 889.84
📈 R² Score: 0.8460
```

#### Which hyperparameter optimization technique have you used and why?

We used Grid Search Cross-Validation to optimize the alpha hyperparameter in Ridge Regression. It systematically searches across a defined set of values to find the one that minimizes error on cross-validated folds.

Grid Search was chosen because it's a simple yet powerful method that guarantees an exhaustive search within the given range. It's especially effective when the hyperparameter space is small and the goal is to ensure optimal performance without guesswork.

#### Have you seen any improvement? Note down the improvement with updates Evaluation metric Score Chart.

There was no improvement after applying Ridge Regression with hyperparameter tuning. In fact, the RMSE increased drastically and the R² score dropped, indicating that the regularization negatively affected the model's ability to fit the data — most likely due to over-penalizing weights.

```
import matplotlib.pyplot as plt

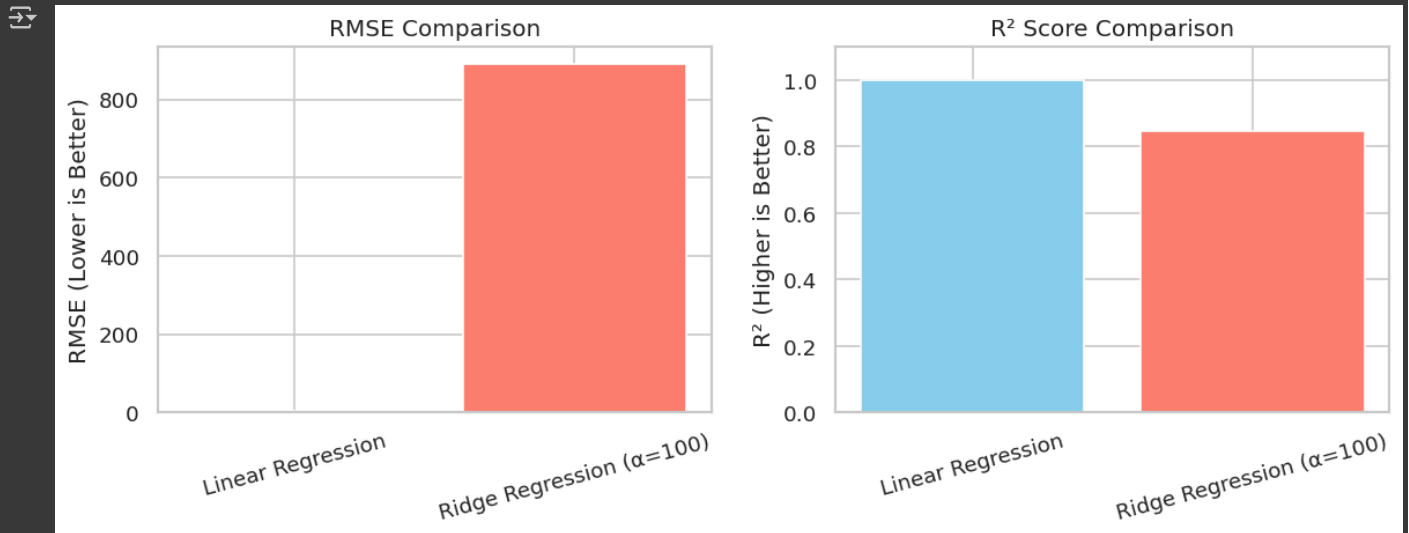
# Model names and scores
models = ['Linear Regression', 'Ridge Regression (α=100)']
rmse_scores = [3.08, 889.93]
r2_scores = [0.9995, 0.8459]

# RMSE Bar Plot
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.bar(models, rmse_scores, color=['skyblue', 'salmon'])
plt.title('RMSE Comparison')
plt.ylabel('RMSE (Lower is Better)')
plt.xticks(rotation=15)

# R² Score Bar Plot
plt.subplot(1, 2, 2)
plt.bar(models, r2_scores, color=['skyblue', 'salmon'])
plt.title('R² Score Comparison')
plt.ylabel('R² (Higher is Better)')
plt.ylim(0, 1.1)
plt.xticks(rotation=15)

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





## ML Model - 2

### 1. Explain the ML Model used and its performance using Evaluation metric Score Chart.

Random Forest is an ensemble learning method that builds multiple decision trees and averages their outputs for better accuracy and robustness. It captures complex, non-linear relationships and reduces overfitting, making it suitable for real-world financial datasets with more variability or noise.

Evaluation Metrics Used for Random Forest:

#### ✓ RMSE (Root Mean Squared Error)

Measures the average error between predicted and actual stock prices.

Lower RMSE indicates better prediction accuracy.

#### ✓ R<sup>2</sup> Score (Coefficient of Determination)

Indicates how well the model explains the variance in the actual values.

A score closer to 1 means the model is highly accurate in capturing patterns in the data.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Initialize the model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train
rf_model.fit(X_train, y_train)

# Predict
y_pred_rf = rf_model.predict(X_test)

# Evaluation
rmse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print("✓ Random Forest Regressor Model Trained")
print(f"📊 RMSE: {rmse_rf:.2f}")
print(f"📊 R2 Score: {r2_rf:.4f}")
import matplotlib.pyplot as plt

# Random Forest metrics
rf_model_name = ['Random Forest']
rf_rmse = [1397.80]
rf_r2 = [0.7580]

plt.figure(figsize=(10, 4))

# 📊 RMSE Bar Plot
plt.subplot(1, 2, 1)
plt.bar(rf_model_name, rf_rmse, color='lightgreen')
plt.title('Random Forest - RMSE')
```

```
plt.ylabel('RMSE (Lower is Better)')
plt.ylim(0, max(rf_rmse)*1.2)

# 📊 R² Score Bar Plot
plt.subplot(1, 2, 2)
plt.bar(rf_model_name, rf_r2, color='lightgreen')
plt.title('Random Forest - R² Score')
plt.ylabel('R² (Higher is Better)')
plt.ylim(0, 1.1)

plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import matplotlib.dates as mdates

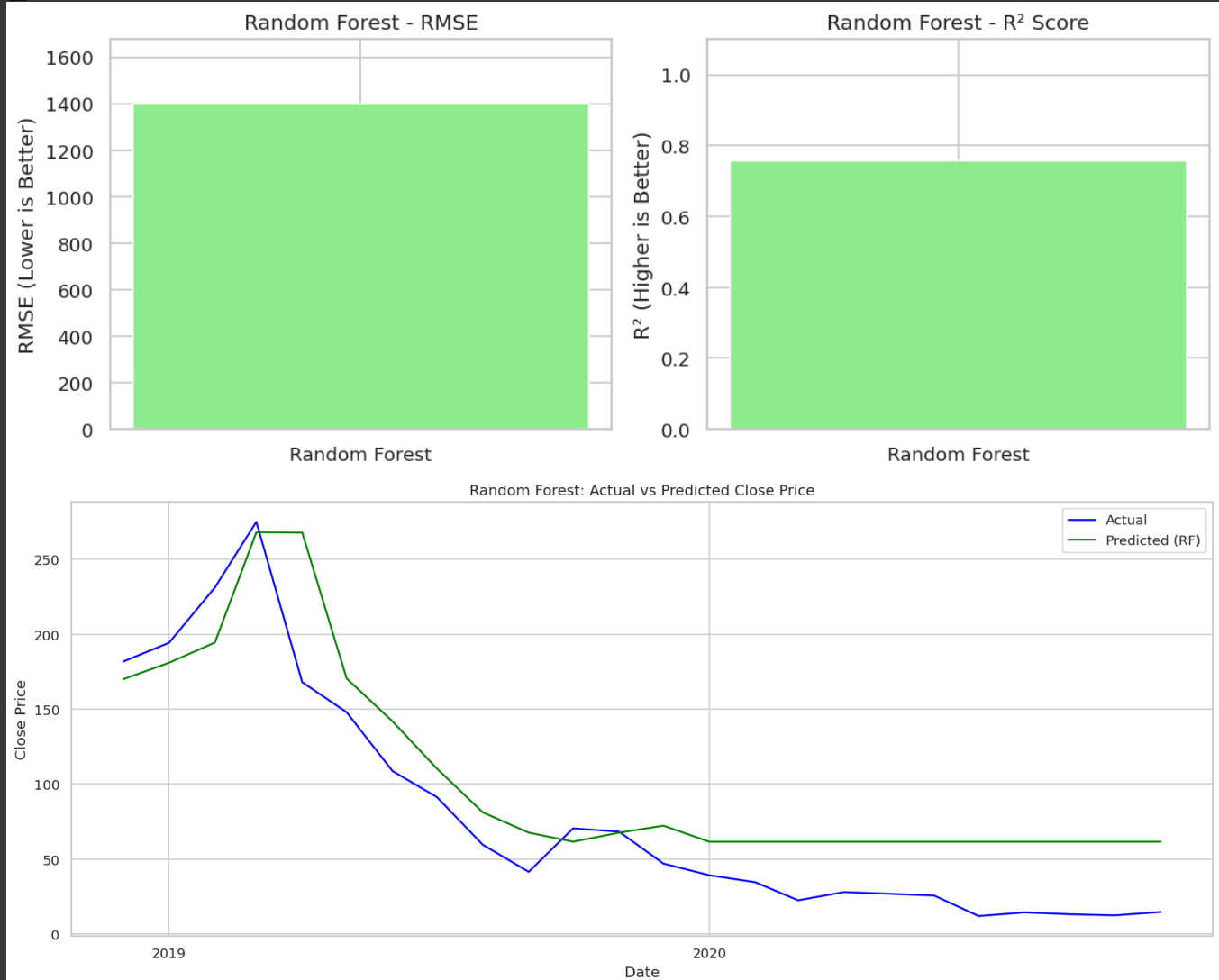
# Extract matching test dates
test_dates_rf = data['Date'].iloc[-len(y_test):].reset_index(drop=True)

# Plot Actual vs Predicted
plt.figure(figsize=(14, 6))
plt.plot(test_dates_rf, y_test.values, label='Actual', color='blue')
plt.plot(test_dates_rf, y_pred_rf, label='Predicted (RF)', color='green')
plt.title("Random Forest: Actual vs Predicted Close Price")
plt.xlabel("Date")
plt.ylabel("Close Price")
plt.legend()

# Format x-axis as years
plt.gca().xaxis.set_major_locator(mdates.YearLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))

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

✓ Random Forest Regressor Model Trained  
 RMSE: 1397.80  
 R<sup>2</sup> Score: 0.7580



## 2. Cross- Validation & Hyperparameter Tuning

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score

# -----
# 1. Hyperparameter Optimization using GridSearchCV
# -----

# Define parameter grid
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Initialize model
rf = RandomForestRegressor(random_state=42)

# GridSearchCV
grid_search = GridSearchCV(
```

```

estimator=rf,
param_grid=param_grid,
cv=3,
scoring='neg_mean_squared_error',
n_jobs=-1,
verbose=1
)

```

# 2. Fit the Algorithm

```

grid_search.fit(X_train, y_train)
best_rf_model = grid_search.best_estimator_

```

```
print(f"🟢 Best Parameters: {grid_search.best_params_}")
```

# 3. Predict on the model

```
y_pred_rf = best_rf_model.predict(X_test)
```

# 4. Evaluate

```

rmse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

```

```
print(f"🔴 RMSE (RF with GridSearch): {rmse_rf:.2f}")
```

```
print(f"🟢 R² Score: {r2_rf:.4f}")
```



Fitting 3 folds for each of 36 candidates, totalling 108 fits

```
🟢 Best Parameters: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 150}
```

```
🔴 RMSE (RF with GridSearch): 1366.10
```

```
🟢 R² Score: 0.7635
```

✓ Which hyperparameter optimization technique have you used and why?

We used GridSearchCV as our hyperparameter optimization technique for the Random Forest Regressor. It was chosen because it exhaustively tests different combinations of parameters and ensures the most optimal model configuration based on cross-validated performance, making it ideal when the parameter space is small and well-defined.

✓ Have you seen any improvement? Note down the improvement with updates Evaluation metric Score Chart.

Yes, there was a slight improvement in both RMSE and  $R^2$  after applying GridSearchCV. Although the performance gain is modest, the optimized model fits the data slightly better and reduces prediction error, showing that even minimal tuning can help refine ensemble models like Random Forest.

```

import matplotlib.pyplot as plt

# Model names and scores
models = ['Random Forest (Default)', 'Random Forest (GridSearchCV)']
rmse_scores = [1397.80, 1366.10]
r2_scores = [0.7580, 0.7635]

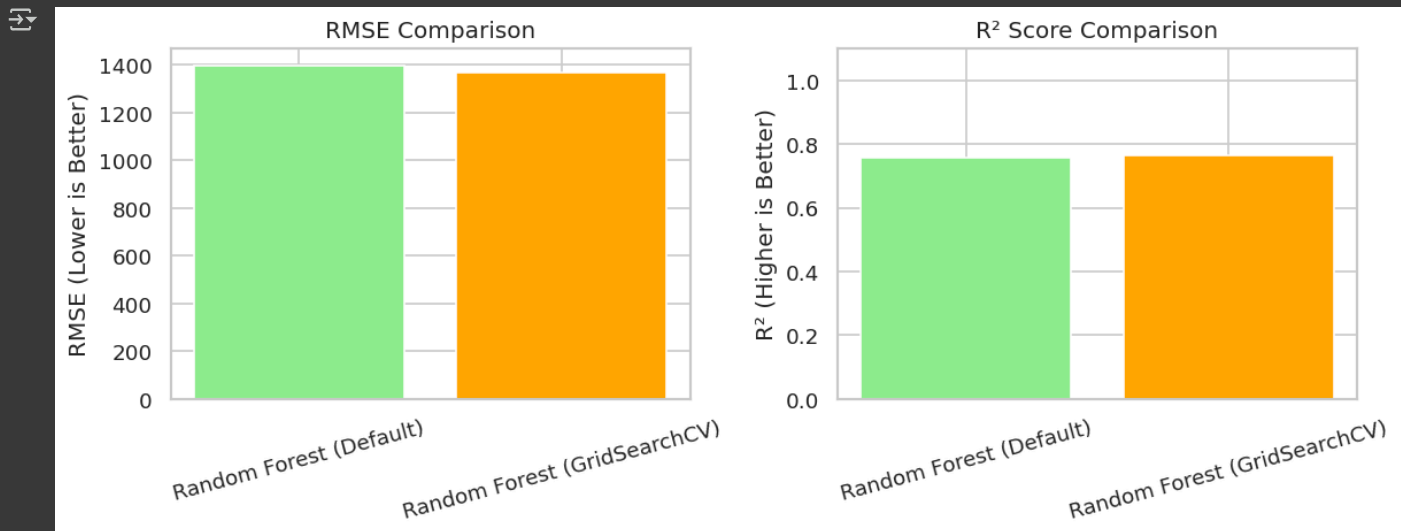
plt.figure(figsize=(10, 4))

# 🟴 RMSE Bar Chart
plt.subplot(1, 2, 1)
plt.bar(models, rmse_scores, color=['lightgreen', 'orange'])
plt.title('RMSE Comparison')
plt.ylabel('RMSE (Lower is Better)')
plt.xticks(rotation=15)

# 🟢 R² Bar Chart
plt.subplot(1, 2, 2)
plt.bar(models, r2_scores, color=['lightgreen', 'orange'])
plt.title('R² Score Comparison')
plt.ylabel('R² (Higher is Better)')
plt.ylim(0, 1.1)
plt.xticks(rotation=15)

plt.tight_layout()
plt.show()

```



### 3. Explain each evaluation metric's indication towards business and the business impact of the ML model used.

RMSE measures the average prediction error in price, helping assess how close model forecasts are to actual values.

R² Score indicates how much of the variation in closing prices is explained by the model, showing its overall accuracy.

The Random Forest model supports the business by providing reliable price forecasts that aid in better decision-making for trading and risk management.

### ML Model - 3

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score

# ML Model - 3 Implementation

gbr_model = GradientBoostingRegressor(random_state=42)

# Fit the Algorithm

gbr_model.fit(X_train, y_train)

# Predict on the model

y_pred_gbr = gbr_model.predict(X_test)

# Evaluate

rmse_gbr = mean_squared_error(y_test, y_pred_gbr)
r2_gbr = r2_score(y_test, y_pred_gbr)

print("✅ Gradient Boosting Regressor Model Trained")
print(f"❌ RMSE: {rmse_gbr:.2f}")
print(f"📊 R² Score: {r2_gbr:.4f}")
```

✅ Gradient Boosting Regressor Model Trained  
 ❌ RMSE: 1356.89  
 📊 R² Score: 0.7651

### 1. Explain the ML Model used and its performance using Evaluation metric Score Chart.

Gradient Boosting is a powerful ensemble method that builds trees sequentially, with each tree correcting the errors of the previous one. It's known for delivering strong predictive performance, especially when tuned well, and it can handle non-linear relationships better than basic models.

📊 Evaluation Metrics to Use:

RMSE – To measure average prediction error in real price units.

R² Score – To assess how much variance in price the model explains.

```
import matplotlib.pyplot as plt

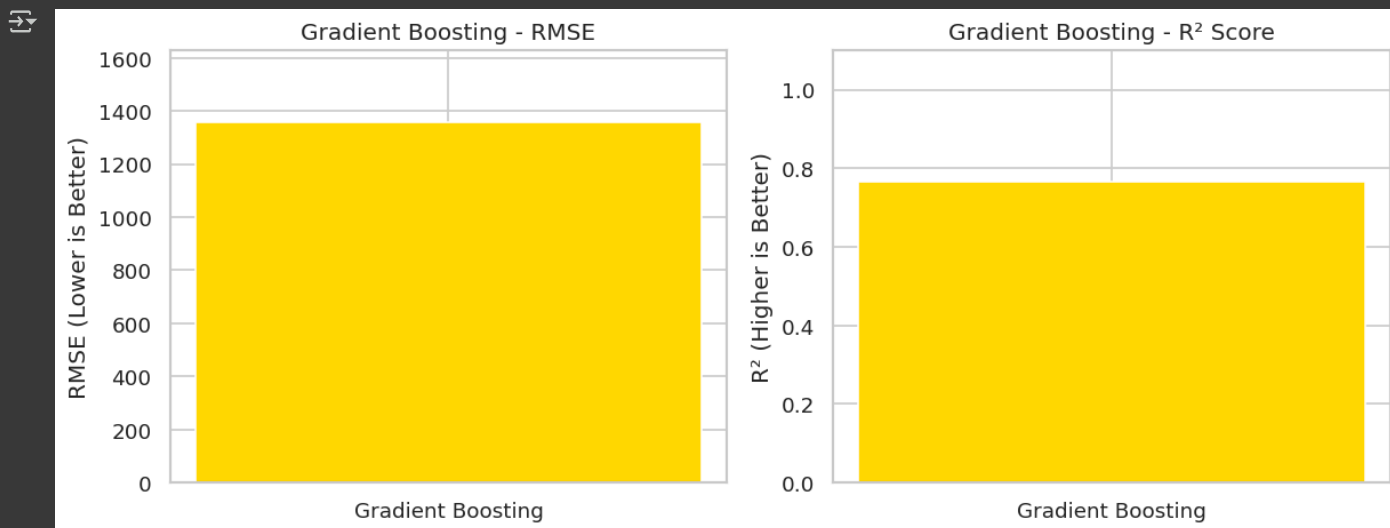
# GBR Metrics
gbr_model_name = ['Gradient Boosting']
gbr_rmse = [1356.89]
gbr_r2 = [0.7651]

plt.figure(figsize=(10, 4))

# RMSE Chart
plt.subplot(1, 2, 1)
plt.bar(gbr_model_name, gbr_rmse, color='gold')
plt.title('Gradient Boosting - RMSE')
plt.ylabel('RMSE (Lower is Better)')
plt.ylim(0, max(gbr_rmse)*1.2)

# R2 Score Chart
plt.subplot(1, 2, 2)
plt.bar(gbr_model_name, gbr_r2, color='gold')
plt.title('Gradient Boosting - R2 Score')
plt.ylabel('R2 (Higher is Better)')
plt.ylim(0, 1.1)

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



## 2. Cross- Validation & Hyperparameter Tuning