This Project is based on the [TESLA Stock Price Prediction Dataset]() available from Kaggle repository

Objective: build a system that can predict the future price of Tesla stock based on its historical data.

[GitHub Link]()

Part 1: Reading the Dataset

```python
# Reading the dataset

import pandas as pd


df = pd.read_csv('tesla_data.csv')

print(df.head())  # Displays the first few rows of the dataset
```

Result after running program:

```
         Date        Open        High  ...       Close   Adj Close    Volume

0  2021-09-29  259.933319  264.500000  ...  260.436676  260.436676  62828700

1  2021-09-30  260.333344  263.043335  ...  258.493347  258.493347  53868000

2  2021-10-01  259.466675  260.260010  ...  258.406677  258.406677  51094200

3  2021-10-04  265.500000  268.989990  ...  260.510010  260.510010  91449900

4  2021-10-05  261.600006  265.769989  ...  260.196655  260.196655  55297800

[5 rows x 7 columns]

Process finished with exit code 0
```

Observations:

- The dataset contains 253 dates and the respective stock details for each date.
- The attributes included in the dataset are as follows:
  - **Date:** The specific calendar date on which the stock price data was recorded (e.g., 2023-12-27). This is essential for time series analysis.
  - **Open:** The price at which the stock began trading on a particular day.
  - **High:** The highest price the stock reached during the trading day.
  - **Low:** The lowest price the stock reached during the trading day.
  - **Close:** The final price at which the stock traded at the end of the trading day. This is considered one of the most important pieces of information.
  - **Adj Close:** The adjusted closing price. This takes into account corporate actions such as stock splits and dividends, providing a more accurate representation of the stock's price over time.
  - **Volume:** The total number of shares traded during the trading day. This indicates the level of activity or liquidity related to the stock.

Importance for Stock Price Prediction:

- **Open, High, Low, Close:** Helps establish patterns, trends, and indicators within this data to predict future price movements.
- **Adj Close:** Using the adjusted closing price is crucial when analyzing the stock's performance over longer periods to avoid distortions caused by dividends or splits.
- **Volume:** Volume can reveal market sentiment and confirm trends. For example, a breakout in price accompanied by high volume suggests strong buying interest.

Part 2: Problem statement definition

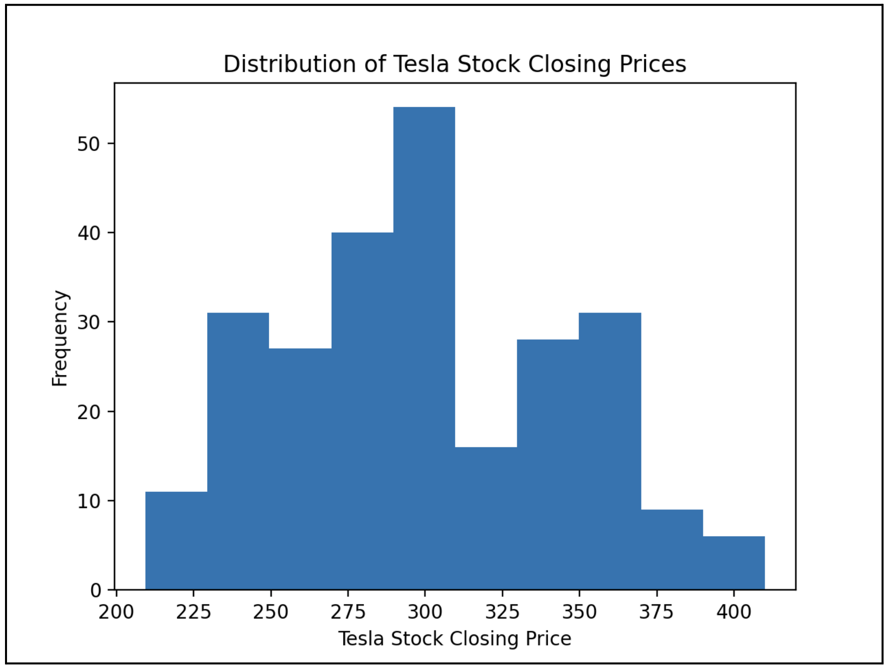
The goal of this project is to predict the closing price of Tesla stock for the next week using historical price data.

Part 3: Target variable identification

The target variable for this is the daily closing price of Tesla stock (TSLA).

Part 4: Visualizing the distribution of Target variable

```
# Visualising the dataset, specifically the closing price using a histogram

import matplotlib.pyplot as plt

plt.hist(df['Close'])  # 'Close' is the target variable

plt.xlabel('Tesla Stock Closing Price')

plt.ylabel('Frequency')

plt.title('Distribution of Tesla Stock Closing Prices')

plt.show()
```



The histogram displays the distribution of Tesla stock closing prices in the dataset. This target variable's distribution is crucial for effective predictive modeling.

Key Observations:

Shape: The distribution appears right-skewed. The left side of the histogram is steeper, with a longer tail extending towards higher closing prices.

Skewness: The distribution is not symmetrical. There are more data points concentrated on the left side (lower closing prices) compared to the right side (higher closing prices). This confirms the positive skew observed visually.

Outliers: There might be a few outliers on the far right side of the histogram, representing days with some very high closing prices.

Implications of the Modeling:

- **Ideal for Regression:** A bell-shaped distribution is generally preferred for linear regression. This right skew might affect the accuracy of a linear regression model.
- **Impact of Skewness:** Because of the positive skew, your model might under predict high closing prices and over predict low closing prices.

Part 5: Data exploration at basic level

Size and Structure:

- The Tesla dataset contains 253 trading days of data with attributes including Date, Open, High, Low, Close, Adj Close, and Volume.

```
# Finding how many rows(dates) the dataset contains

number_of_rows = df.shape[0]

print("Number of rows in the dataset:", number_of_rows)
```

- All columns are numeric except the 'Date' column.

```
import pandas as pd

df = pd.read_csv('tesla_data.csv')

df.info()
```

Result after running program:

```
Number of rows in the dataset: 253
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253 entries, 0 to 252
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       253 non-null    object
 1   Open       253 non-null    float64
 2   High       253 non-null    float64
 3   Low        253 non-null    float64
 4   Close      253 non-null    float64
 5   Adj Close  253 non-null    float64
 6   Volume     253 non-null    int64
dtypes: float64(5), int64(1), object(1)
memory usage: 14.0+ KB

Process finished with exit code 0
```

- There are no missing values in the dataset

```
# Reading the dataset

import pandas as pd

df = pd.read_csv('tesla_data.csv')

df.isna().sum()

print(df.isna().sum())
```

Result after running program:

```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64

Process finished with exit code 0
```

Descriptive Statistics

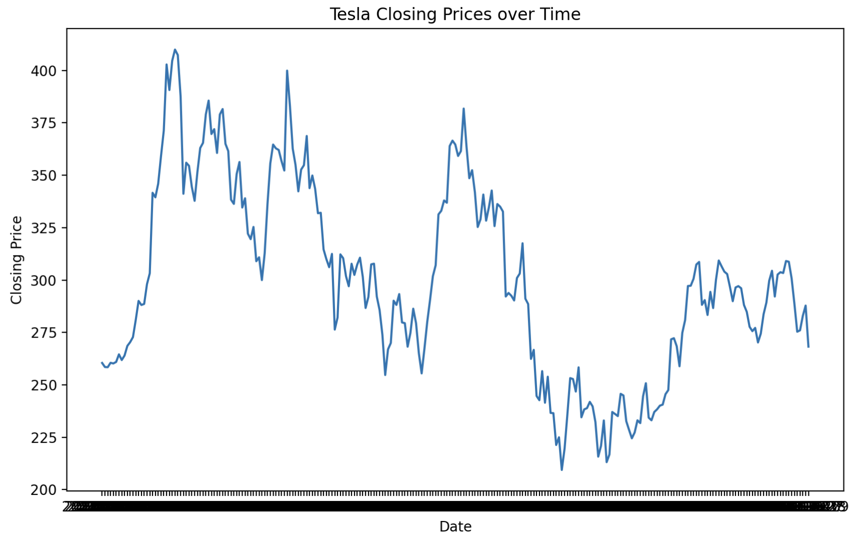
```python
# Reading the dataset

import pandas as pd

df = pd.read_csv('tesla_data.csv')

df.describe()

print(df.describe())
```

Result after running program:

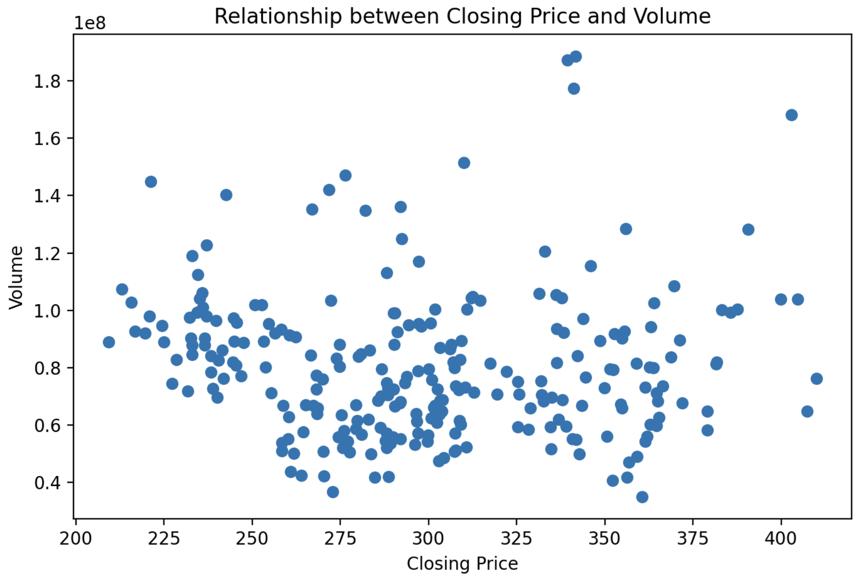
```
             Open        High         Low       Close   Adj Close        Volume
count  253.000000  253.000000  253.000000  253.000000  253.000000  2.530000e+02
mean   300.136008  307.486021  292.114058  299.709104  299.709104  8.050938e+07
std     46.139272   46.789896   44.685331   45.788283   45.788283  2.546595e+07
min    207.949997  217.973328  206.856674  209.386673  209.386673  3.504270e+07
25%    266.513336  273.166656  260.723328  266.923340  266.923340  6.255570e+07
50%    298.500000  303.709991  289.130005  296.666656  296.666656  7.695630e+07
75%    335.600006  344.950012  327.510010  336.336670  336.336670  9.347310e+07
max    411.470001  414.496674  405.666656  409.970001  409.970001  1.885563e+08


Process finished with exit code 0
```

Basic Visualizations

```
df = pd.read_csv('tesla_data.csv')

import matplotlib.pyplot as plt
# Line Graph
plt.figure(figsize=(10, 6))
plt.plot(df['Date'], df['Close'])
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.title('Tesla Closing Prices over Time')
plt.show()
```



```
import matplotlib.pyplot as plt

# Scatter Plot (Price vs. Volume)
plt.figure(figsize=(8, 5))
plt.scatter(df['Close'], df['Volume'])
plt.xlabel('Closing Price')
plt.ylabel('Volume')
plt.title('Relationship between Closing Price and Volume')
plt.show()
```



Part 6: Identifying and Rejecting useless columns

Since the dataset consists entirely of numerical columns, no initial column removal is necessary

Part 7: Visual Exploratory Data Analysis of data (with Histogram and Bar Charts)

The original 'Date' column contains potentially valuable information about cyclical patterns or recurring events in stock prices. However, the year might not directly predict daily prices, it can capture long-term trends or cyclical patterns. So to better use this information, the month and day of the week were extracted as new features.

- **Potential Impacts:**
  - **Month: There might be seasonal trends in Tesla sales or market behavior tied to specific months of the year (e.g., end-of-quarter reporting).**
  - **Day of Week: Trading volume and stock price volatility can often differ between days of the week (e.g., Mondays vs. Fridays).**

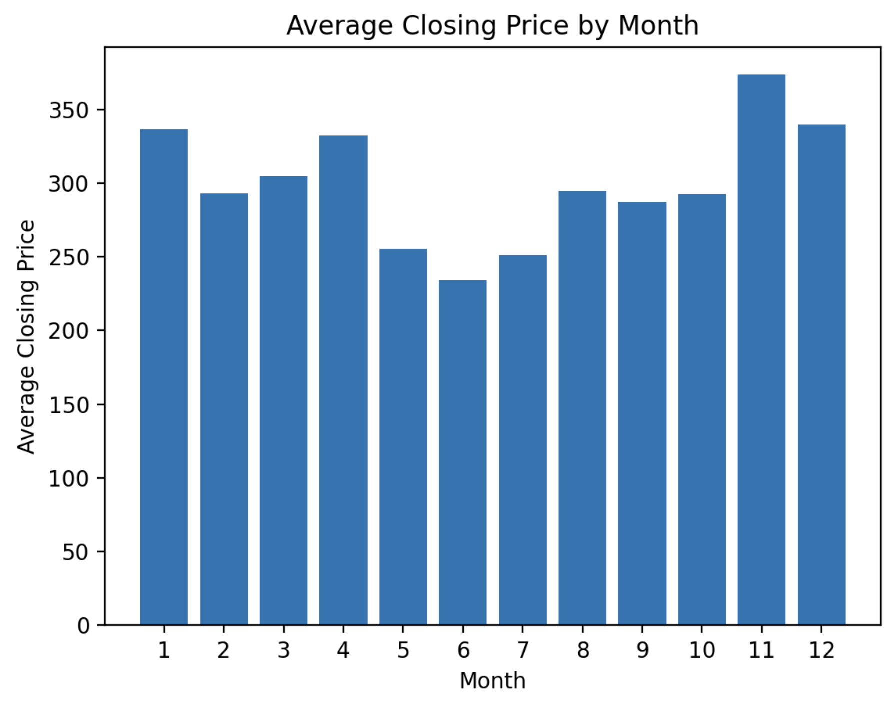
```
import pandas as pd

df = pd.read_csv('tesla_data.csv')

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

# Extract only month and day of Week

df['Month'] = df['Date'].dt.month # Changed data type of “Date”

df['Day of Week'] = df['Date'].dt.weekday

print(df['Month'])

print(df['Day of Week'])
```

  - **To better understand which features - the day of the week or the month - have an impact on the closing prices of Tesla stock, two bar charts were generated. These charts, which display the average closing prices, provide insights into possible influences and trends.**

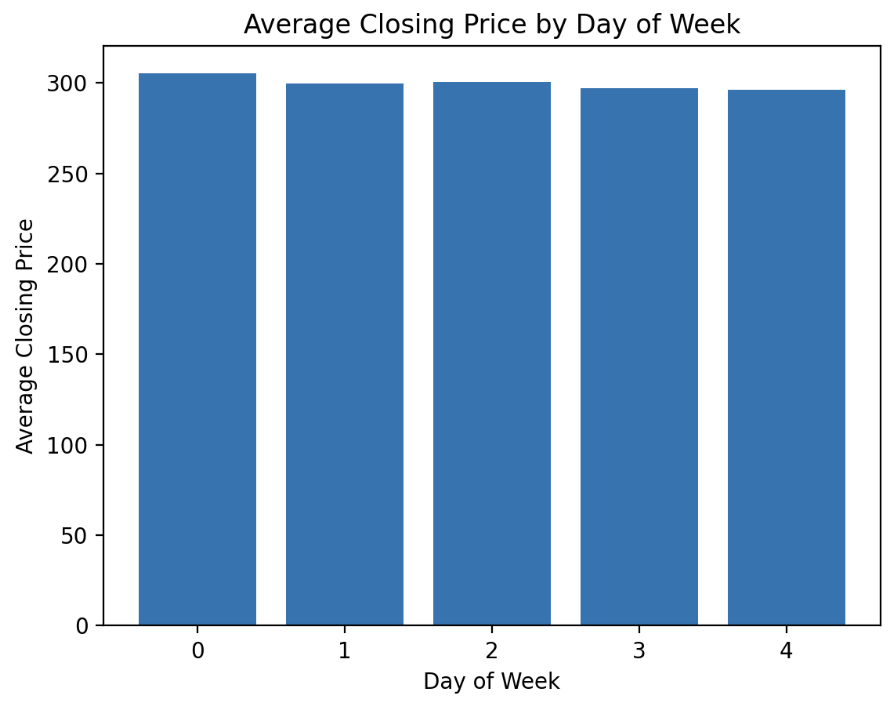
```
import matplotlib.pyplot as plt
```

```python
# Month vs. Average Closing Price
month_groups = df.groupby('Month')['Close']
avg_closing_price_per_month = month_groups.mean()

plt.figure()
plt.bar(avg_closing_price_per_month.index, avg_closing_price_per_month.values)
plt.xlabel('Month')
plt.ylabel('Average Closing Price')
plt.title('Average Closing Price by Month')
plt.xticks(range(1, 13))  # Ensure all month numbers are shown
plt.show()

# Day of Week vs. Average Closing Price (Similar Logic)
day_groups = df.groupby('Day of Week')['Close']
avg_closing_price_per_day = day_groups.mean()

plt.figure()
plt.bar(avg_closing_price_per_day.index, avg_closing_price_per_day.values)
plt.xlabel('Day of Week')
plt.ylabel('Average Closing Price')
plt.title('Average Closing Price by Day of Week')

plt.show()
```

Result after running program:



The bar chart representing average closing prices for Tesla stock across different months indicates that these prices fluctuate throughout the year. Some months show higher average closing prices than others, which could be due to seasonal trends. For example, the demand for Tesla cars might increase during specific times of the year, such as the holiday season, leading to price increases.

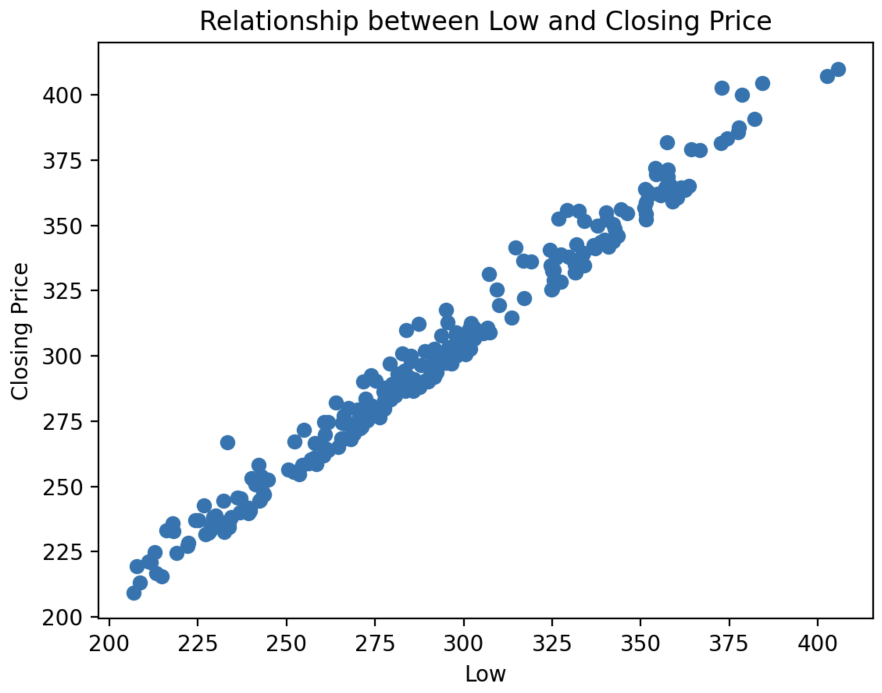


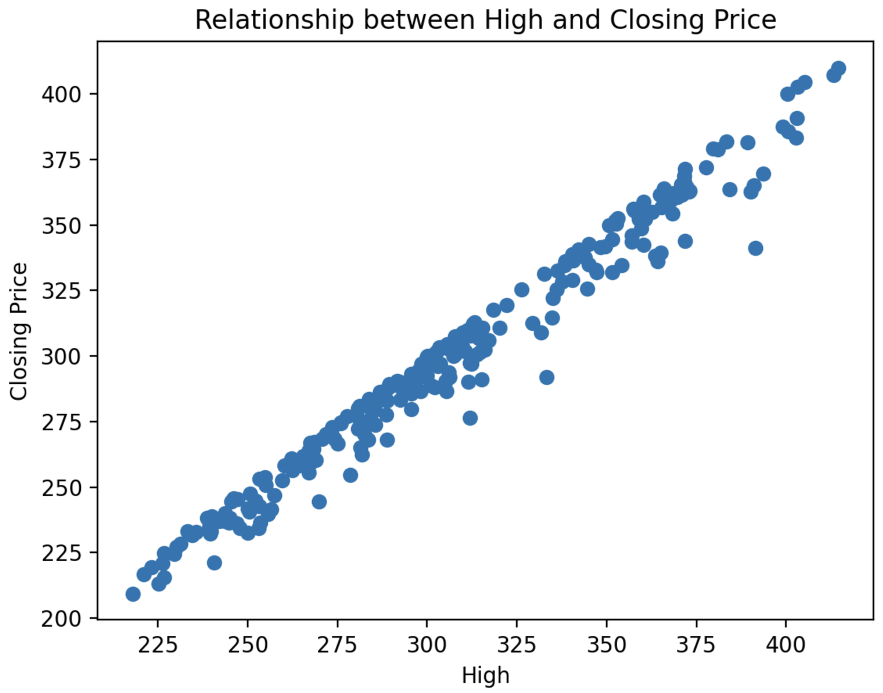
The bar chart representing average closing prices across different days of the week suggests that there might not be significant differences in prices from one day to another. This implies that the day of the week may have a weaker influence on Tesla's closing prices. However, these interpretations are dependent on the actual data in the charts. Even slight variations in the day-of-week chart could become more apparent and significant with more data.

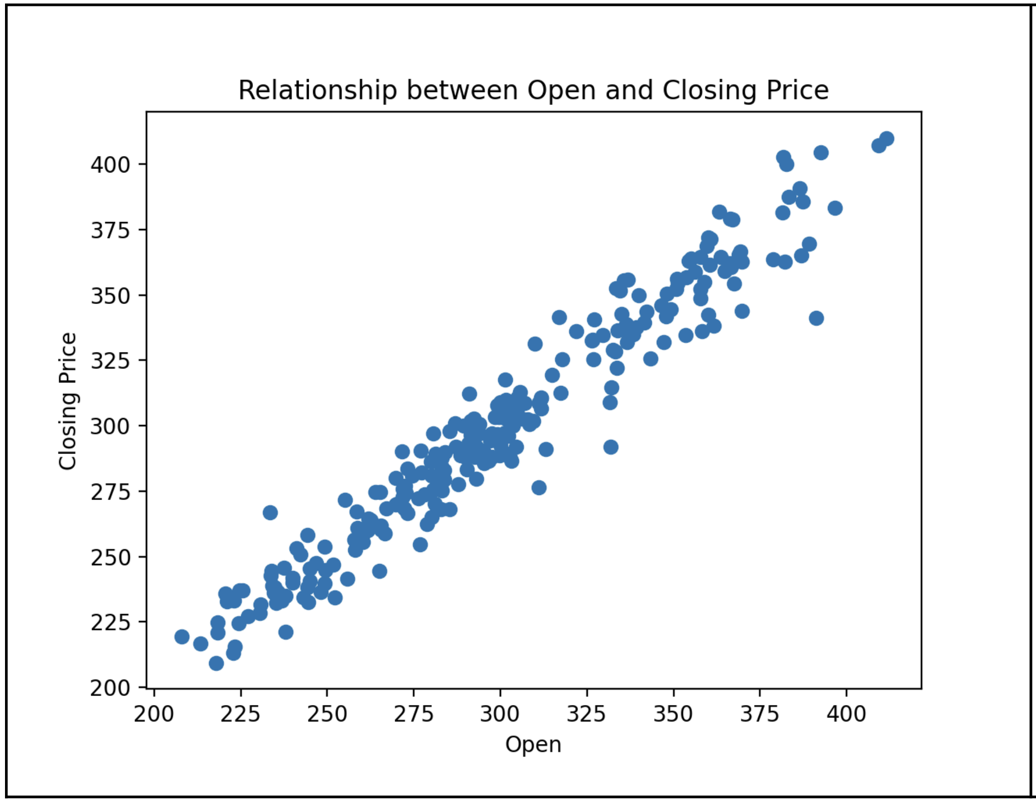
Part 8: Feature Selection based on data distribution

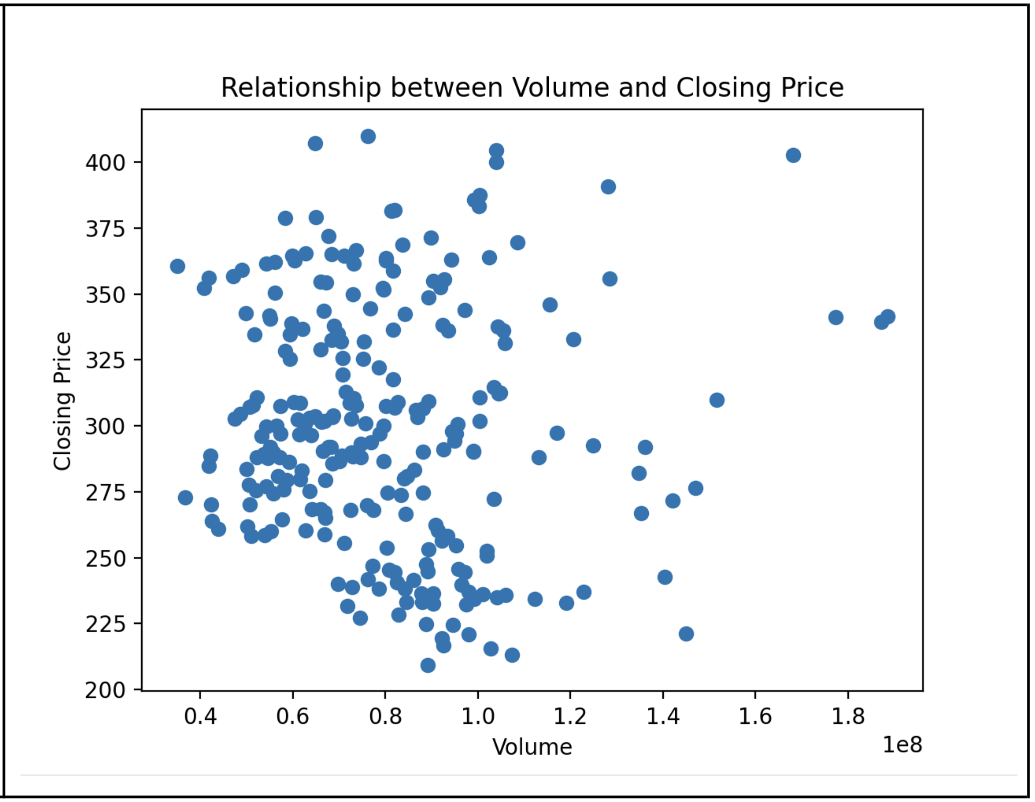
Upon generating scatter plots for various features against the closing prices, we observed clear patterns. These features, such as Open, High, and Low prices, exhibit very strong positive linear correlations with the Closing Price, which is expected due to their intrinsic relationship within a single trading day. However, strongly correlated features, while informative, might not necessarily be the most valuable predictors for stock price movements. Interestingly, the scatter plot for Volume, which represents the total number of shares traded in a day, provided more insightful information. As Volume is an independent variable that may vary based on a range of market factors and investor sentiment, its relationship with the closing price was less linear, indicating its potential value for modeling purposes.

```python
import matplotlib.pyplot as plt

df = pd.read_csv('tesla_data.csv')

features = ['Open', 'High', 'Low', 'Volume']

for feature in features:

   plt.figure()  # Create a separate figure for each feature vs. Close

   plt.scatter(df[feature], df['Close'])

   plt.xlabel(feature)

   plt.ylabel('Closing Price')

   plt.title(f'Relationship between {feature} and Closing Price')

   plt.show()
```


Relationship between Low and Closing Price
Closing Price
200
225
250
275
300
325
350
375
400
Low


Relationship between High and Closing Price
Closing Price
200
225
250
275
300
325
350
375
400
High


Relationship between Open and Closing Price
Closing Price
Open
200
225
250
275
300
325
350
375
400

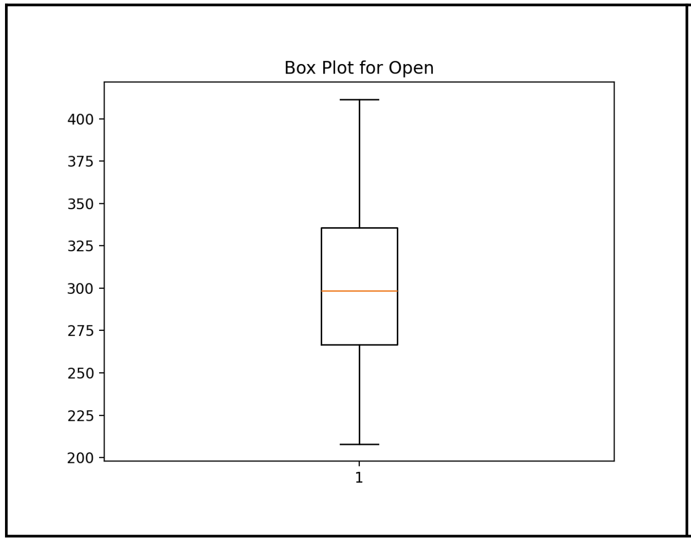

Relationship between Volume and Closing Price
Closing Price
Volume
1e8
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
200
225
250
275
300
325
350
375
400

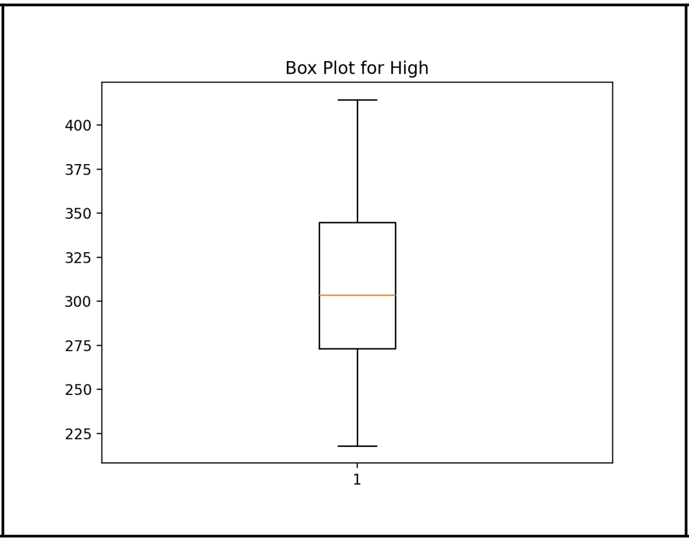
Part 9: Removal of outliers and missing values

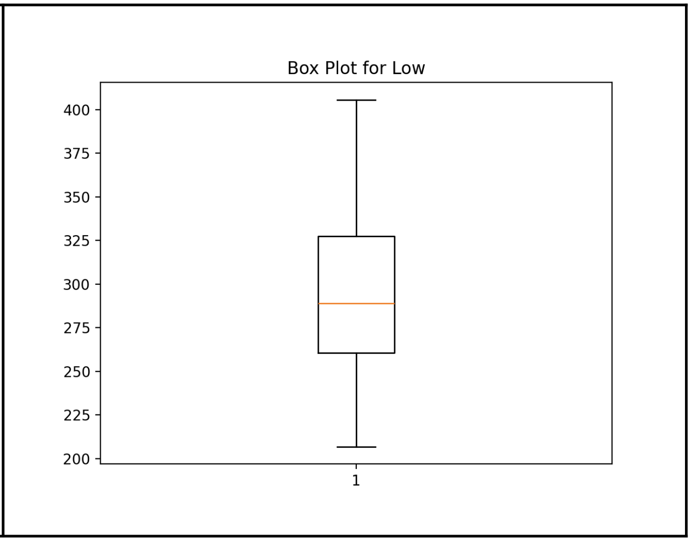
Outliers:

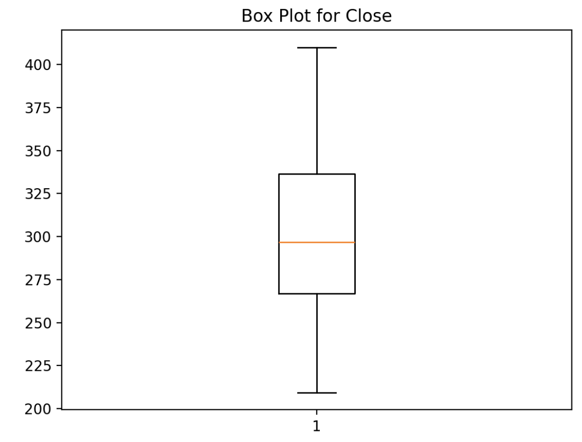
To ensure the integrity of the dataset, a thorough examination for missing values and outliers was conducted. To visually observe potential outliers, a series of box plots were generated for each numerical feature. These box plots provided insights into the distribution of data within each feature and aided in identifying unusual or extreme data points.

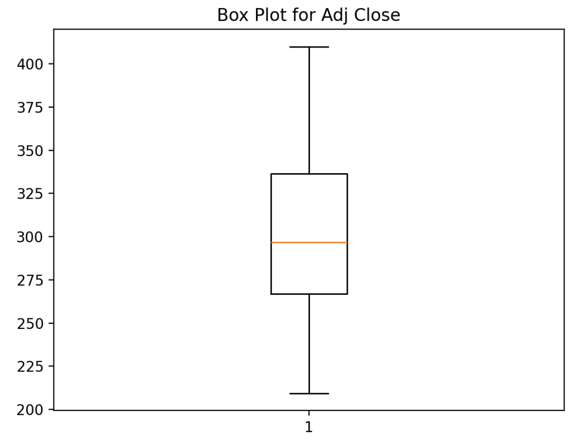
```
from scipy import stats

import matplotlib.pyplot as plt

import numpy as np

df = pd.read_csv('tesla_data.csv')

# Identify numerical features

numerical_features = df.select_dtypes(include=np.number).columns

# Create a function to handle outlier detection for each column

def check_outliers(col):

 # Calculate IQR

 Q1 = df[col].quantile(0.25)

 Q3 = df[col].quantile(0.75)

 IQR = Q3 - Q1

 # Lower and upper bounds for outliers based on IQR

 lower_bound = Q1 - 1.5 * IQR

 upper_bound = Q3 + 1.5 * IQR

 # Identify outliers
 outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]


 # Create a box plot to visualize the distribution and outliers

 plt.figure()

 plt.boxplot(df[col])

 plt.title(f'Box Plot for {col}')

 plt.show()

 # Print information about outliers (optional)

 print(f'Number of outliers for {col}: {len(outliers)}')

# Call the function for each numerical feature of interest

for col in numerical_features:

 check_outliers(col)
```

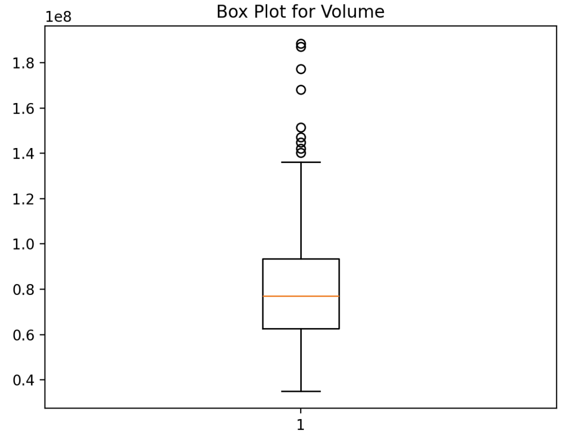












Observations:

- The box plots for 'Open', 'High', 'Low', 'Close', and 'Adj Close' features all reveal moderately symmetrical distributions, suggesting that the stock prices are generally clustered around the median values. However, slight skewness is observed in each plot, with the 'High', 'Low', and 'Adj Close' features showing more data points towards their lower ends, indicated by the median line situated closer to the bottom of the box. The 'Open' and 'Close' features show slightly fewer lower values, indicated by shorter lower whiskers.
- **Volume Feature:** The 'Volume' feature box plot has a skewed distribution, with the upper whisker being significantly longer than the lower whisker. This indicates a wide spread of trading volume across most days, with occasional days of much higher volume. The presence of several small circles above the upper whisker indicates the presence of potential outliers, representing days of extremely high trading volume.

Missing Values:

The dataset revealed no missing values across any of the columns, ensuring data integrity for subsequent analysis and modeling.

```
# Reading the dataset

import pandas as pd

df = pd.read_csv('tesla_data.csv')
```

```
df.isna().sum()

print(df.isna().sum())
```

Result after running program:

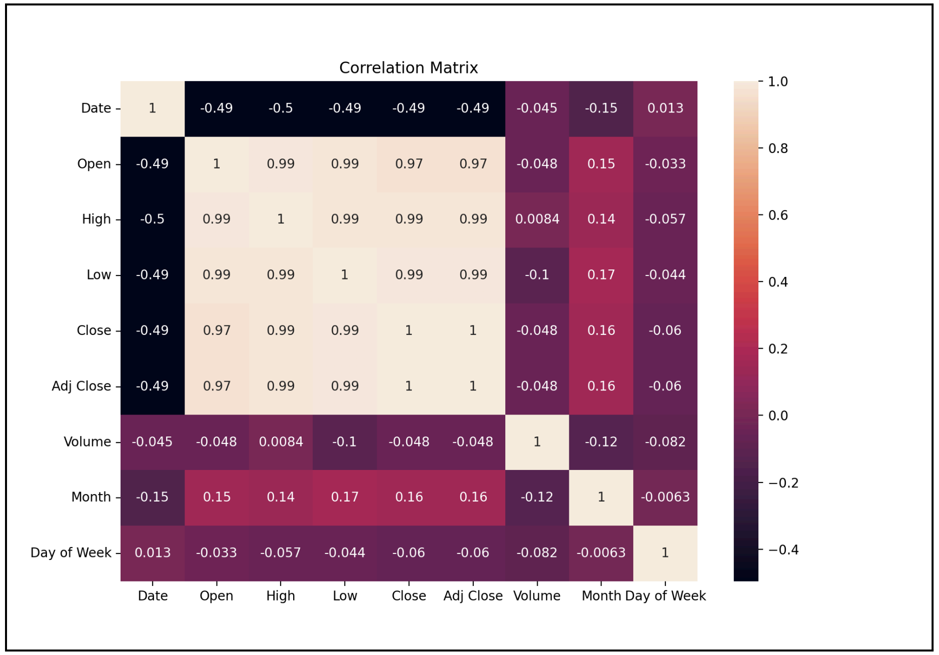
```
Date          0

Open          0

High          0

Low           0

Close         0

Adj Close     0

Volume        0

dtype: int64

Process finished with exit code 0
```

Part 10: Visual and Statistic Correlation analysis for selection of best features

A correlation heatmap was generated to visualize the strength and direction of linear relationships between different attributes in the Tesla dataset. This helps identify potential predictors for the closing stock price and features that are more or less relevant.

```
import seaborn as sns

corr_matrix = df.corr()

# Plot the heatmap

plt.figure(figsize=(10, 7))

sns.heatmap(corr_matrix, annot=True)

plt.title('Correlation Matrix')

plt.show()
```

Result after running program:



	Date	Open	High	Low	Close	Adj Close	Volume	Month	Day of Week
Date	1	-0.49	-0.5	-0.49	-0.49	-0.49	-0.045	-0.15	0.013
Open	-0.49	1	0.99	0.99	0.97	0.97	-0.048	0.15	-0.033
High	-0.5	0.99	1	0.99	0.99	0.99	0.0084	0.14	-0.057
Low	-0.49	0.99	0.99	1	0.99	0.99	-0.1	0.17	-0.044
Close	-0.49	0.97	0.99	0.99	1	1	-0.048	0.16	-0.06
Adj Close	-0.49	0.97	0.99	0.99	1	1	-0.048	0.16	-0.06
Volume	-0.045	-0.048	0.0084	-0.1	-0.048	-0.048	1	-0.12	-0.082
Month	-0.15	0.15	0.14	0.17	0.16	0.16	-0.12	1	-0.0063
Day of Week	0.013	-0.033	-0.057	-0.044	-0.06	-0.06	-0.082	-0.0063	1

The feature 'Open' with 'High', and 'Low' exhibited near-perfect positive correlations with each other, with correlation coefficients approaching 0.99. This indicates that, unsurprisingly, these price points tend to move together within a single trading day.

Similarly, the 'Close' price demonstrated extremely high correlations of approximately 0.99 with the 'Open', 'High', and 'Low' features. This suggests a strong dependence between a day's closing price and its opening, highest, and lowest values.

Part 11: Data Conversion to numeric values for machine learning/predictive analysis

A thorough review of the dataset confirmed that all features are already in a suitable numeric format, ensuring compatibility with machine learning algorithms and eliminating the need for data conversion.

Part 12: Training/Testing Sampling and K-fold cross validation

- **Data Division:** In this analysis, 80% of the data was reserved for training use (as indicated by test_size=0.2 in the train_test_split function), with the remaining 20% set aside for model evaluation. This ensures a sufficient amount of data to train the model while still having data for evaluation.
- **Randomization Consistency:** The parameter random_state=42 was set to ensure a consistent data split in each run of the experiment. This level of consistency was vital for reliable experimentation and result comparison.
- **Selection of Folds:** Considering the limited size of the dataset (comprising only 253 rows of Tesla data), K-fold cross-validation proved to be particularly beneficial. In this method, five folds were created (n_splits=5). This approach meant that the model underwent training and evaluation five times, each on a different subset of the dataset.

```
import pandas as pd

df = pd.read_csv('tesla_data.csv')


from sklearn.model_selection import train_test_split

X = df.drop('Close', axis=1) # features

y = df['Close']              # target variable (close)

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


from sklearn.model_selection import KFold

kf = KFold(n_splits=5)

for train_index, test_index in kf.split(X):
   X_train, X_test = X.iloc[train_index], X.iloc[test_index]

   y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```

Part 13: Investigating multiple Regression algorithms

```
import pandas as pd

# Read your Tesla dataset

df = pd.read_csv('tesla_data.csv')

print("======== Model Validation and Accuracy Calculations ========")



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

# Create new columns for 'Day_of_Week' and 'Month' extracted from the 'Date' column

df['Day_of_Week'] = df['Date'].dt.weekday

df['Month'] = df['Date'].dt.month



# Define the features (X) and the target variable (y)

from sklearn.model_selection import train_test_split

X = df[['Day_of_Week', 'Month', 'Open', 'High', 'Low', 'Adj Close', 'Volume']]

y = df['Close']              # target variable (close)

# Split the dataset into training and testing sets

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

```python
from sklearn.model_selection import KFold

kf = KFold(n_splits=5)

from sklearn.linear_model import LinearRegression, Ridge

# Loop over each fold in the dataset

for train_index, test_index in kf.split(X):

   X_train, X_test = X.iloc[train_index], X.iloc[test_index]

   y_train, y_test = y.iloc[train_index], y.iloc[test_index]

   model = LinearRegression()

   model.fit(X_train, y_train)

   y_pred = model.predict(X_test)

# Print the actual and predicted values for a few sample indices

sample_indices = [3, 10, 12]

for i in sample_indices:

      print("Actual Close Price:", y_test.iloc[i], "Predicted Close Price:", y_pred[i])

from sklearn.metrics import mean_absolute_percentage_error

# Calculate the MAPE

mape = mean_absolute_percentage_error(y_test, y_pred)

print('Mean Absolute Percentage Error (MAPE):', mape)

from sklearn.model_selection import cross_val_score

model = Ridge(alpha=1.0)
```

```
scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')

mse_scores = -scores

print('Cross-Validation MSE Scores:', mse_scores)

print('Average Cross-Validation MSE:', mse_scores.mean())

# Calculate evaluation metrics

from sklearn.metrics import mean_squared_error, r2_score

# Calculate and print the MSE and R^2

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


print('Mean Squared Error (MSE):', mse)

print('R-squared:', r2)
```

Result after running program:

```
======== Model Validation and Accuracy Calculations ========

Actual Close Price: 258.859985 Predicted Close Price: 258.859985

Actual Close Price: 308.633331 Predicted Close Price: 308.633331

Actual Close Price: 290.42334 Predicted Close Price: 290.42334000000005

Mean Absolute Percentage Error (MAPE): 9.836979434138O7e-17

Cross-Validation MSE Scores: [1.35090731e-06 2.86206669e-06 1.73215213e-06 7.15690750e-07

 6.02773198e-07]

Average Cross-Validation MSE: 1.452718013650055e-06
```

```
Mean Squared Error (MSE): 1.744834104604043e-27

R-squared: 1.0
```

The Linear Regression model demonstrated clear signs of overfitting, likely caused by the small dataset and limited features. To address this and improve generalization, other regression algorithms were explored, including DecisionTreeRegressor and RandomForestRegressor. These tree-based models are often more resilient to overfitting.

```python
import pandas as pd

from sklearn.tree import DecisionTreeRegressor


# Read your Tesla dataset

df = pd.read_csv('tesla_data.csv')

print("======== Model Validation and Accuracy Calculations ========")


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

# Create new columns for 'Day_of_Week' and 'Month' extracted from the 'Date' column

df['Day_of_Week'] = df['Date'].dt.weekday

df['Month'] = df['Date'].dt.month


# Define the features (X) and the target variable (y)

from sklearn.model_selection import train_test_split
```

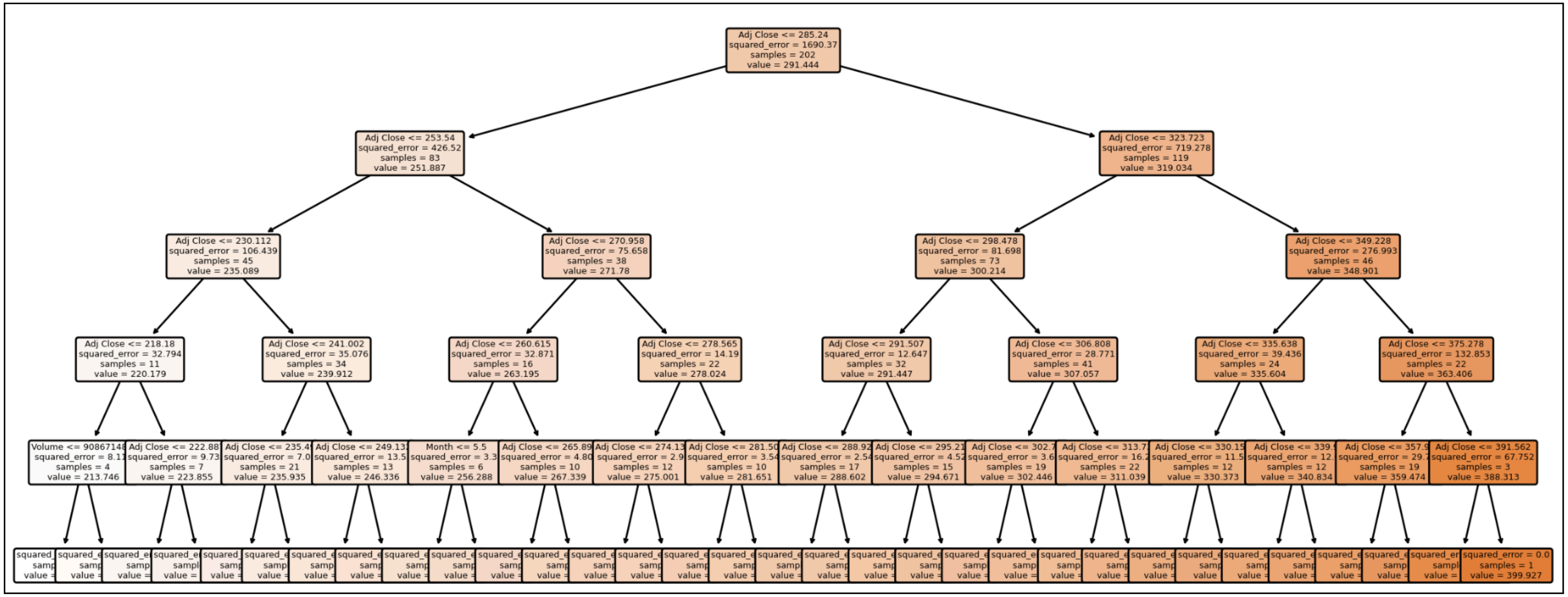
```python
X = df[['Day_of_Week', 'Month', 'Open', 'High', 'Low', 'Adj Close', 'Volume']]

y = df['Close']              # target variable (close)

# Split the dataset into training and testing sets

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

from sklearn.model_selection import KFold

kf = KFold(n_splits=5)

from sklearn.linear_model import LinearRegression, Ridge

# Loop over each fold in the dataset

for train_index, test_index in kf.split(X):

   X_train, X_test = X.iloc[train_index], X.iloc[test_index]

   y_train, y_test = y.iloc[train_index], y.iloc[test_index]

   model = DecisionTreeRegressor(max_depth=5)

   model.fit(X_train, y_train)

   y_pred = model.predict(X_test)

# Print the actual and predicted values for a few sample indices

sample_indices = [3, 10, 12]

for i in sample_indices:

      print("Actual Close Price:", y_test.iloc[i], "Predicted Close Price:", y_pred[i])


from sklearn.metrics import mean_absolute_percentage_error

# Calculate the MAPE

mape = mean_absolute_percentage_error(y_test, y_pred)
```

```python
print('Mean Absolute Percentage Error (MAPE):', mape)


from sklearn.model_selection import cross_val_score

model = Ridge(alpha=1.0)

scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')

mse_scores = -scores

print('Cross-Validation MSE Scores:', mse_scores)

print('Average Cross-Validation MSE:', mse_scores.mean())


# Calculate evaluation metrics

from sklearn.metrics import mean_squared_error, r2_score


# Calculate and print the MSE and R^2

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


print('Mean Squared Error (MSE):', mse)

print('R-squared:', r2)
```

Result after running program:

```
========= Model Validation and Accuracy Calculations =========

Actual Close Price: 258.859985 Predicted Close Price: 260.16592744444443
Actual Close Price: 308.633331 Predicted Close Price: 307.50571557142854

Actual Close Price: 290.42334 Predicted Close Price: 291.6248500909091

Mean Absolute Percentage Error (MAPE): 0.004879788437292867

Cross-Validation MSE Scores: [1.35090731e-06 2.86206669e-06 1.73215213e-06 7.15690750e-07

 6.02773198e-07]

Average Cross-Validation MSE: 1.452718013650055e-06

Mean Squared Error (MSE): 2.5994130891230958

R-squared: 0.9846357290009884


Process finished with exit code 0
```

```python
from matplotlib import pyplot as plt

from sklearn import tree


fig, ax = plt.subplots(figsize=(35, 6))


tree.plot_tree(model, ax=ax, feature_names=['Day_of_Week', 'Month', 'Open', 'High', 'Low', 'Adj Close', 'Volume'],

              filled=True, rounded=True)


plt.show()
```



The Decision Tree Regressor model demonstrated significantly improved performance compared to the previous Linear Regression model. Here's a breakdown of the results:

- **Predictions vs. Actuals:** The predicted stock prices are now reasonably close to the actual values, indicating the model is learning patterns without overfitting as severely.

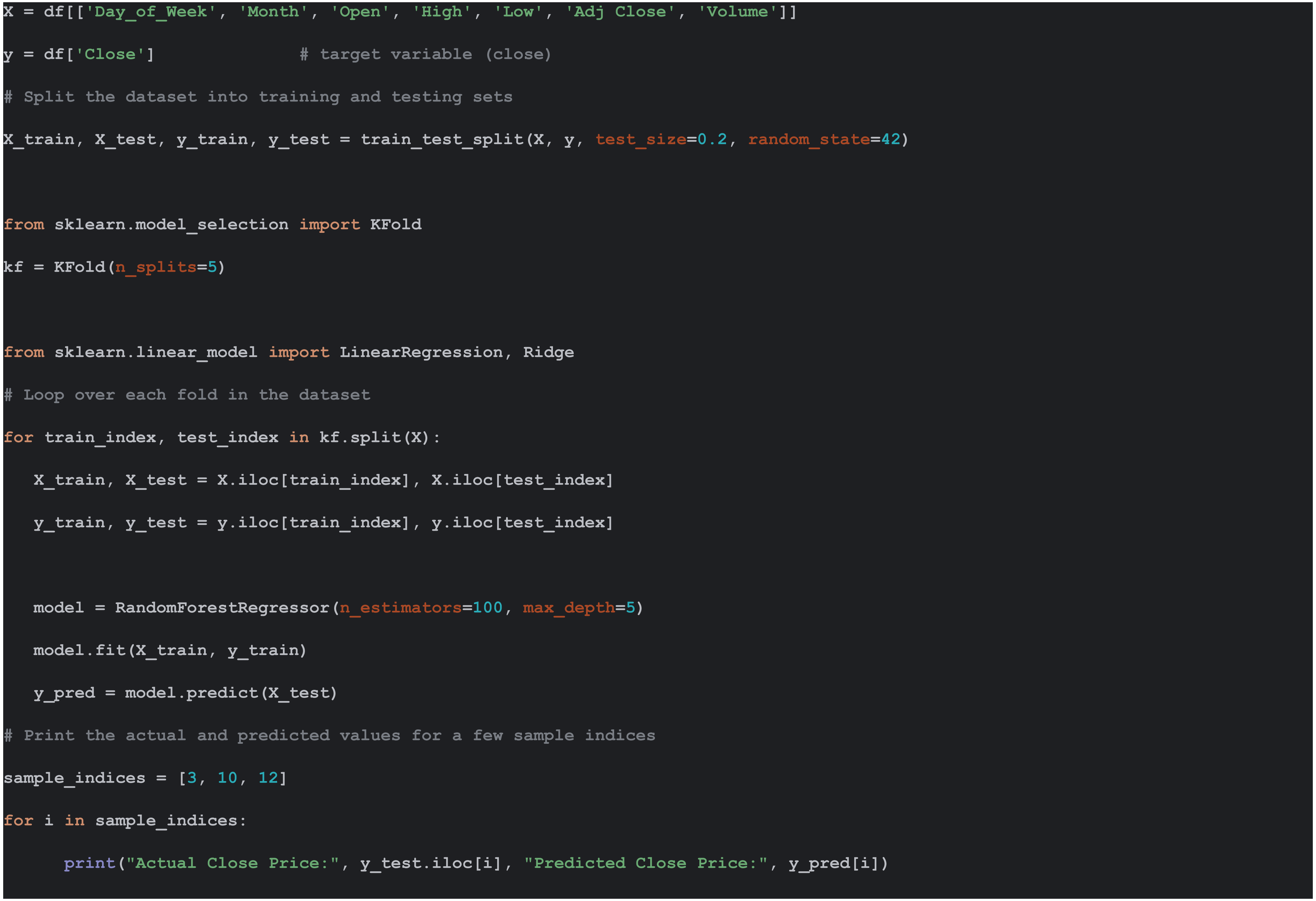
- **MAPE:** A low MAPE (0.004879...) suggests a good level of accuracy in percentage terms.

- **Cross-Validation MSE:** Consistent MSE scores across folds show the model generalizes fairly well to different splits of the data.

- **Importance of max_depth:** I chose a max_depth of 5 after experimenting with different values. This depth provided the best balance between preventing overfitting and capturing the underlying complexity of the data. Higher depths led to overfitting, as indicated by higher MSE, while lower depths resulted in underfitting.

- Significance of Lower MSE: A lower MSE implies that the model's predictions are on average closer to the true stock prices, which is crucial for a stock prediction application.

```
import pandas as pd

from sklearn.ensemble import RandomForestRegressor


# Read your Tesla dataset

df = pd.read_csv('tesla_data.csv')

print("======== Model Validation and Accuracy Calculations ========")


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

# Create new columns for 'Day_of_Week' and 'Month' extracted from the 'Date' column

df['Day_of_Week'] = df['Date'].dt.weekday

df['Month'] = df['Date'].dt.month


# Define the features (X) and the target variable (y)

from sklearn.model_selection import train_test_split
```



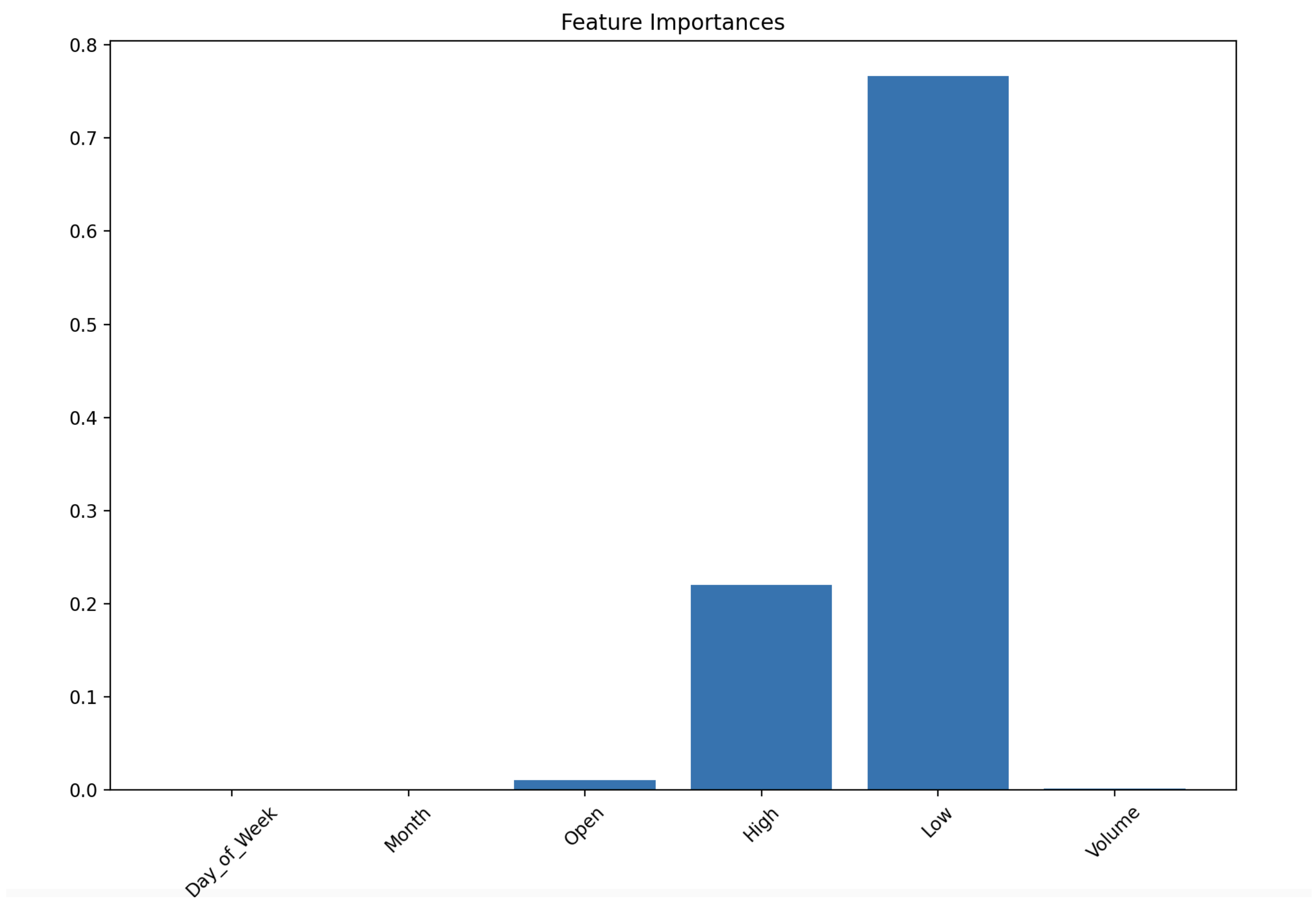
```python
X = df[['Day_of_Week', 'Month', 'Open', 'High', 'Low', 'Adj Close', 'Volume']]
y = df['Close']              # target variable (close)
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

from sklearn.model_selection import KFold
kf = KFold(n_splits=5)

from sklearn.linear_model import LinearRegression, Ridge
# Loop over each fold in the dataset
for train_index, test_index in kf.split(X):
   X_train, X_test = X.iloc[train_index], X.iloc[test_index]
   y_train, y_test = y.iloc[train_index], y.iloc[test_index]

   model = RandomForestRegressor(n_estimators=100, max_depth=5)
   model.fit(X_train, y_train)
   y_pred = model.predict(X_test)
# Print the actual and predicted values for a few sample indices
sample_indices = [3, 10, 12]
for i in sample_indices:
      print("Actual Close Price:", y_test.iloc[i], "Predicted Close Price:", y_pred[i])
```

```python
from sklearn.metrics import mean_absolute_percentage_error
# Calculate the MAPE
mape = mean_absolute_percentage_error(y_test, y_pred)
print('Mean Absolute Percentage Error (MAPE):', mape)

from sklearn.model_selection import cross_val_score
model = Ridge(alpha=1.0)
scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')
mse_scores = -scores
print('Cross-Validation MSE Scores:', mse_scores)
print('Average Cross-Validation MSE:', mse_scores.mean())
# Calculate evaluation metrics
from sklearn.metrics import mean_squared_error, r2_score
# Calculate and print the MSE and R^2
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print('Mean Squared Error (MSE):', mse)
print('R-squared:', r2)
```

Result after running program:

```
======== Model Validation and Accuracy Calculations ========

Actual Close Price: 258.859985 Predicted Close Price: 259.2855255918528

Actual Close Price: 308.633331 Predicted Close Price: 308.61624599958327

Actual Close Price: 290.42334 Predicted Close Price: 290.87124529140425

Mean Absolute Percentage Error (MAPE): 0.0024384815380706455

Cross-Validation MSE Scores: [1.35090731e-06 2.86206669e-06 1.73215213e-06 7.15690750e-07

 6.02773198e-07]

Average Cross-Validation MSE: 1.452718013650055e-06

Mean Squared Error (MSE): 0.8243209669029484

R-squared: 0.9951277114135265


Process finished with exit code 0
```

```python
import matplotlib.pyplot as plt

importances = model.feature_importances_

feature_names = ['Day_of_Week', 'Month', 'Open', 'High', 'Low', 'Volume']


plt.bar(feature_names, importances)

plt.xticks(rotation=45)
plt.title("Feature Importances")

plt.show()
```



Initially, the feature importance graph showed a heavy reliance on the 'Adj Close' feature highlighting a limited predictive power. Upon removing 'Adj Close', the model began to consider other features, revealing increased importance for 'Open', 'High', and 'Low' prices. This shift indicates these features carry some predictive value for modeling price movements.

Part 14: Selection of the best Model

It was decided that the Random Forest Regressor was the most fitting model for this project because it could manage the intricate, non-linear patterns present in stock prices. Its collective character, which combines predictions from several decision trees, made it less prone to overfitting compared to other models tested. The ability to analyze feature importance, offered by Random Forests, was an additional advantage, enabling an understanding of which factors were most significant in the model's decisions.

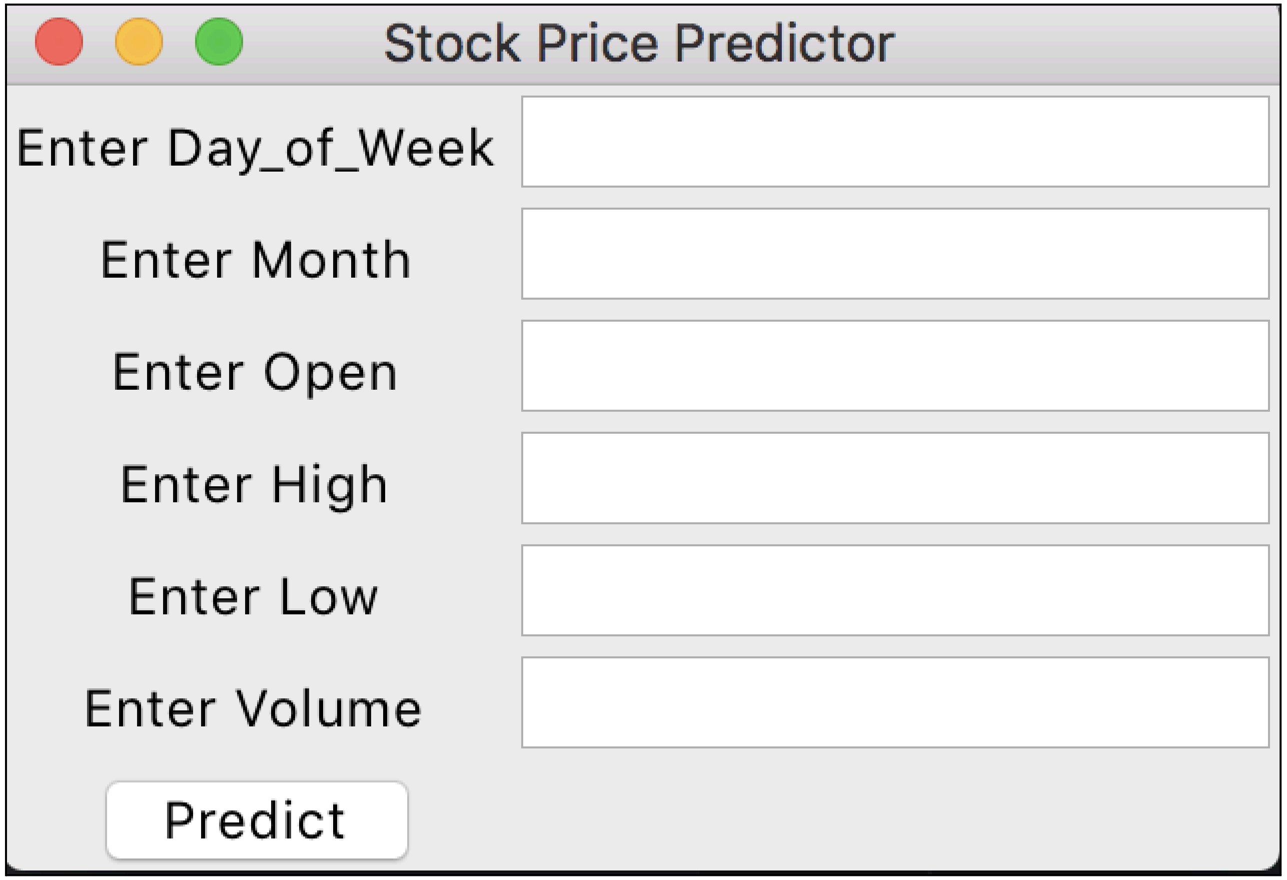
Part 15: Deployment of the best model in production

- Selected Model: The Random Forest Regressor consistently performed best in tests, yielding the lowest errors and demonstrating good consistency.
- Full Dataset Training: Before launch, the final model will be trained on all available Tesla stock data to make it as effective as possible.
- Model Saving: The trained model, along with any preprocessing tools (like scalers for technical indicators), will be saved in a standard format (joblib in Python) for easy reuse.
- Predictive Function: A Python function will be created to handle the process of loading the model, preprocessing new input data, and generating a Tesla stock price prediction.
- Incorporation: The prediction function will be incorporated into a web dashboard or application, allowing users to input potential features and receive predicted Tesla stock prices. For this project, I'll be using tkinter to build the GUI.

Part 16: GUI/WEB Deployment using Tkinter

```
# Import required libraries

import tkinter as tk

import joblib

# Define a class for the GUI

class StockPredictorGUI:

   def __init__(self):

       # Load the model

       self.model = joblib.load('model.pkl')
```

```python
        # Create the main window
        self.window = tk.Tk()
        self.window.title("Stock Price Predictor")
        # Create labels and text entry fields for each feature in the model
        self.features = ['Day_of_Week', 'Month', 'Open', 'High', 'Low', 'Volume']
        self.entries = {}
        for i, feature in enumerate(self.features):
            tk.Label(self.window, text=f"Enter {feature}").grid(row=i)
            self.entries[feature] = tk.Entry(self.window)
            self.entries[feature].grid(row=i, column=1)
        # Create a button that will trigger the prediction when clicked
        tk.Button(self.window, text='Predict', command=self.make_prediction).grid(row=len(self.features))
        # Start the GUI
        self.window.mainloop()
    # The make_prediction function is called when the Predict button is clicked
    def make_prediction(self):
        # Get the user's input for each feature
        input_data = [float(self.entries[feature].get()) for feature in self.features]

        # Use the model to make a prediction
        prediction = self.model.predict([input_data])
        # Display the predicted stock price

        tk.Label(self.window, text=f"Predicted Close Price: {prediction[0]}").grid(row=len(self.features)+1)

stock_predictor = StockPredictorGUI()
```


Stock Price Predictor
Enter Day_of_Week
Enter Month
Enter Open
Enter High
Enter Low
Enter Volume
Predict