Cocacola stock price

Introduction

The aim of this project is to analyze and predict Coca-Cola's stock price using Machine Learning models. The project involves collecting historical stock data, performing exploratory data analysis (EDA), and applying predictive algorithms like Random Forest, Decision Tree, and LSTM to forecast future trends.

```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.svm import SVR
from sklearn.linear_model import SGDRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense, LSTM
```

Dataset Description

The dataset contains historical Coca-Cola stock data with columns such as Date, Open, High, Low, Close, and Volume. The 'Close' price column was used as the target variable for prediction. The dataset was loaded using pandas and examined for missing values, structure, and descriptive statistics.

loading Dataset

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

Choose Files Coca-Cola_...tory (1).csv
Coca-Cola_stock_history (1).csv(text/csv) - 1233831 bytes, last modified: 10/14/2025 - 100% done

```
Saving Coca-Cola_stock_history (1).csv to Coca-Cola_stock_history (1).csv
```

```
import pandas as pd
cc=pd.read_csv("Coca-Cola_stock_history (1).csv")
cc.head()
```

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	1962-01-02	0.050016	0.051378	0.050016	0.050016	806400	0.0	0
1	1962-01-03	0.049273	0.049273	0.048159	0.048902	1574400	0.0	0
2	1962-01-04	0.049026	0.049645	0.049026	0.049273	844800	0.0	0
3	1962-01-05	0.049273	0.049892	0.048035	0.048159	1420800	0.0	0
4	1962-01-08	0.047787	0.047787	0.046735	0.047664	2035200	0.0	0

Next steps: Generate code with cc New interactive sheet

```
cc.columns
```

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

```
cc.index
```

```
RangeIndex(start=0, stop=15311, step=1)
```

```
cc.tail()
```

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
15306	2022-10-20 00:00:00-04:00	55.770000	55.919998	54.959999	55.080002	16905100	0.0	0
15307	2022-10-21 00:00:00-04:00	55.000000	56.110001	54.990002	55.959999	15028000	0.0	0
15308	2022-10-24 00:00:00-04:00	56.639999	57.730000	56.570000	57.570000	17416700	0.0	0
15309	2022-10-25 00:00:00-04:00	59.040001	59.110001	57.750000	58.950001	28829900	0.0	0
15310	2022-10-26 00:00:00-04:00	59.009998	59.779999	58.860001	59.389999	15831400	0.0	0

```
cc.shape
```

```
(15311, 8)
```

Exploratory Data Analysis (EDA)

In this stage, the data was analyzed to understand its structure and patterns. The dataset showed no significant missing values. Descriptive statistics such as mean, median, and standard deviation were calculated. Visualization using Matplotlib displayed the closing price trends over time, helping to understand market movements and volatility.

```
print(cc.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15311 entries, 0 to 15310
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Date          15311 non-null  object 
 1   Open          15311 non-null  float64
 2   High          15311 non-null  float64
 3   Low           15311 non-null  float64
 4   Close         15311 non-null  float64
 5   Volume        15311 non-null  int64  
 6   Dividends     15311 non-null  float64
 7   Stock Splits  15311 non-null  int64  
dtypes: float64(5), int64(2), object(1)
memory usage: 957.1+ KB
None
```

```
print(cc.describe())
```

```
               Open          High           Low         Close        Volume  \
count  15311.000000  15311.000000  15311.000000  15311.000000  1.531100e+04   
mean      11.812883     11.906708     11.717375     11.815409  9.139213e+06   
std       15.025726     15.133336     14.915580     15.026316  7.957947e+06   
min        0.037154      0.037279      0.034890      0.037028  7.680000e+04   
25%        0.238453      0.240305      0.236415      0.238312  2.889600e+06   
50%        4.935146      4.980985      4.884242      4.937339  7.708800e+06   
75%       17.383926     17.612844     17.168283     17.415106  1.307130e+07   
max       66.037933     66.235058     64.776308     65.259270  1.241690e+08   

          Dividends  Stock Splits  
count  15311.000000  15311.000000  
mean       0.001678      0.001110  
std        0.021302      0.049148  
min        0.000000      0.000000  
25%        0.000000      0.000000  
50%        0.000000      0.000000  
75%        0.000000      0.000000  
max        0.440000      3.000000  
```

```
print(cc.isnull().sum())
```

```
Date            0
Open            0
High            0
Low             0
```

```
Close           0
Volume          0
Dividends       0
Stock Splits    0
dtype: int64
```

```
# Visualizing the closing price history
plt.figure(figsize=(16,8))
plt.title("Close Price History")
plt.plot(cc["Close"])
plt.xlabel("Date", fontsize=18)
plt.ylabel("Close Price USD ($)", fontsize=18)
plt.show()
```

pre-processing data

The Close column was scaled using MinMaxScaler to normalize the data between 0 and 1. The dataset was then split into training (80%) and testing (20%) sets. A window of 60 days of past data was used to predict the next day's stock price, as part of the time series modeling process.#

```
# Using the 'Close' column for prediction
data = cc.filter(["Close"])
dataset = data.values
```

LSTM Model

```
# Scaling the data
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)
```

Splitting the data into training and testing sets

```
# Splitting the data into training and testing sets
train_size = int(len(dataset) * 0.8)
train_data = scaled_data[0:train_size, :]
test_data = scaled_data[train_size - 60:, :]

# Preparing the data for LSTM model
def create_dataset(data, look_back=60):
    X, y = [], []
    for i in range(len(data) - look_back):
        X.append(data[i:(i + look_back), 0])
        y.append(data[i + look_back, 0])
    return np.array(X), np.array(y)

look_back = 60
X_train, y_train = create_dataset(train_data, look_back)
X_test, y_test = create_dataset(test_data, look_back)
```

```
# Reshape input to be [samples, time steps, features] for LSTM
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

Training and Building the LSTM Model

Multiple regression models were tested including Support Vector Regressor (SVR), Decision Tree Regressor, and Random Forest Regressor. Additionally, an LSTM (Long Short-Term Memory) neural network model was prepared for sequential prediction. These models were chosen to capture both linear and non-linear dependencies in stock price movements.

```
lstm_model = Sequential()
lstm_model.add(LSTM(50, return_sequences=True, input_shape=(0, 1)))
lstm_model.add(LSTM(50, return_sequences=False))
lstm_model.add(Dense(25))
lstm_model.add(Dense(1))
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/rnn/rnn.py:199: UserWarning: Do not pass an `input_shape`/`input_dim` argu
  super().__init__(**kwargs)
```

```
data = cc.filter(["Close"])
dataset = data.values
```

```
# Scaling the data
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)
```

```
# Splitting the data into training and testing sets
train_size = int(len(dataset) * 0.8)
train_data = scaled_data[0:train_size, :]
test_data = scaled_data[train_size - 60:, :]

# Preparing the data for LSTM model
def create_dataset(data, look_back=60):
    X, y = [], []
    for i in range(len(data) - look_back):
        X.append(data[i:(i + look_back), 0])
        y.append(data[i + look_back, 0])
    return np.array(X), np.array(y)

look_back = 60
X_train, y_train = create_dataset(train_data, look_back)
X_test, y_test = create_dataset(test_data, look_back)
```

```
lstm_model.compile(optimizer='adam', loss='mean_squared_error')
lstm_model.fit(X_train, y_train, batch_size=1, epochs=1)
```

```
12188/12188 ━━━━━━━━━━━━━━━━━━━━ 301s 24ms/step - loss: 1.3347e-04
<keras.src.callbacks.history.History at 0x7f7cfbf3c380>
```

```
lstm_model.compile(optimizer='adam', loss='mean_squared_error')
lstm_model.fit(X_train, y_train, batch_size=1, epochs=1)
```

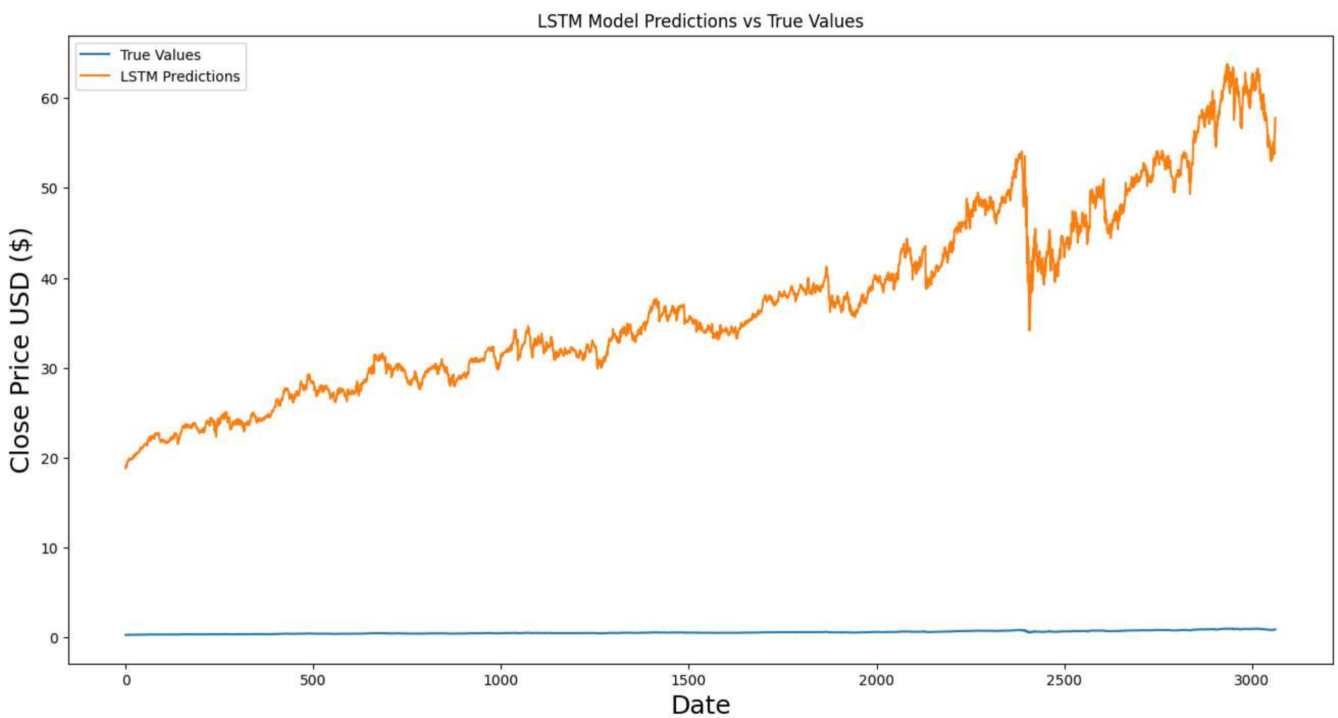
```
12188/12188 ━━━━━━━━━━━━━━━━━━━━ 295s 24ms/step - loss: 3.0791e-05
<keras.src.callbacks.history.History at 0x7f7cfc31bd40>
```

```
lstm_predictions = lstm_model.predict(X_test)
lstm_predictions = scaler.inverse_transform(lstm_predictions)

rmse = np.sqrt(np.mean(((lstm_predictions - y_test) ** 2)))
print('LSTM Model RMSE:', rmse)
```

```
96/96 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step
LSTM Model RMSE: 37.947840251746456
```

```
# Visualizing the Results for LSTM
plt.figure(figsize=(16,8))
plt.title('LSTM Model Predictions vs True Values')
plt.plot(y_test, label='True Values')
plt.plot(lstm_predictions, label='LSTM Predictions')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.legend()
plt.show()
```



SVR Model

```
# Preparing the data for SVR model
X = data.index.values.reshape(-1, 1)
y = data['Close'].values
```

```
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

```
# Scaling the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

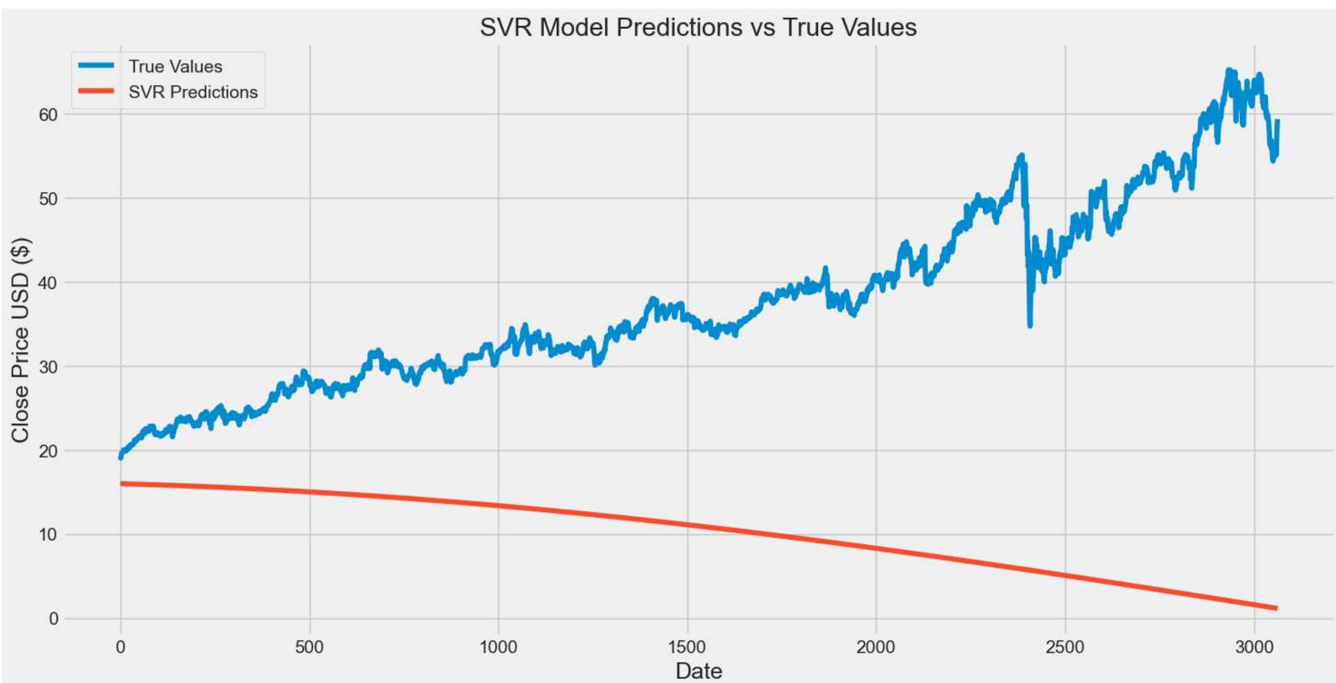
```
# Training the SVR Model
svr_model = SVR(kernel='rbf', C=1e3, gamma=0.1)
svr_model.fit(X_train_scaled, y_train)
```

```
SVR
SVR(C=1000.0, gamma=0.1)
```

```
# Predicting and evaluating the SVR model
svr_predictions = svr_model.predict(X_test_scaled)
svr_rmse = np.sqrt(mean_squared_error(y_test, svr_predictions))
print('SVR Model RMSE:', svr_rmse)
```

```
SVR Model RMSE: 31.388272265004012
```

```
# Visualizing the Results for SVR
plt.figure(figsize=(16,8))
plt.title('SVR Model Predictions vs True Values')
plt.plot(y_test, label='True Values')
plt.plot(svr_predictions, label='SVR Predictions')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.legend()
plt.show()
```



SGD Regressor Model

```
# Preparing the data for SGD Regressor model
X = data.index.values.reshape(-1, 1)
y = data['Close'].values
```

```
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

```
# Scaling the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

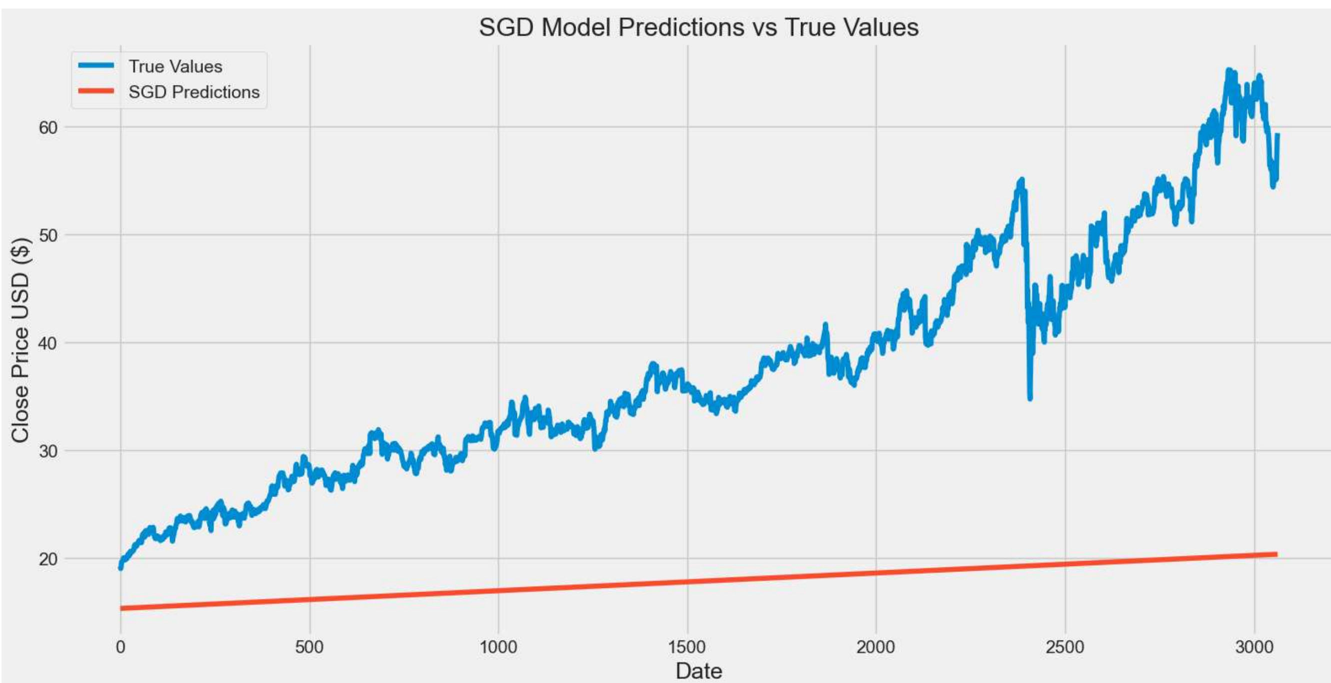
```
# Training the SGD Regressor Model
sgd_model = SGDRegressor(max_iter=1000, tol=1e-3)
sgd_model.fit(X_train_scaled, y_train)
```

```
SGDRegressor
SGDRegressor()
```

```
# Predicting and evaluating the SGD model
sgd_predictions = sgd_model.predict(X_test_scaled)
sgd_rmse = np.sqrt(mean_squared_error(y_test, sgd_predictions))
print('SGD Model RMSE:', sgd_rmse)
```

```
SGD Model RMSE: 21.895595810423227
```

```
# Visualizing the Results for SGD
plt.figure(figsize=(16,8))
plt.title('SGD Model Predictions vs True Values')
plt.plot(y_test, label='True Values')
plt.plot(sgd_predictions, label='SGD Predictions')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.legend()
plt.show()
```



Decision Tree Regressor Model

```
# Preparing the data for Decision Tree Regressor model
X = data.index.values.reshape(-1, 1)
y = data['Close'].values
```

```
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

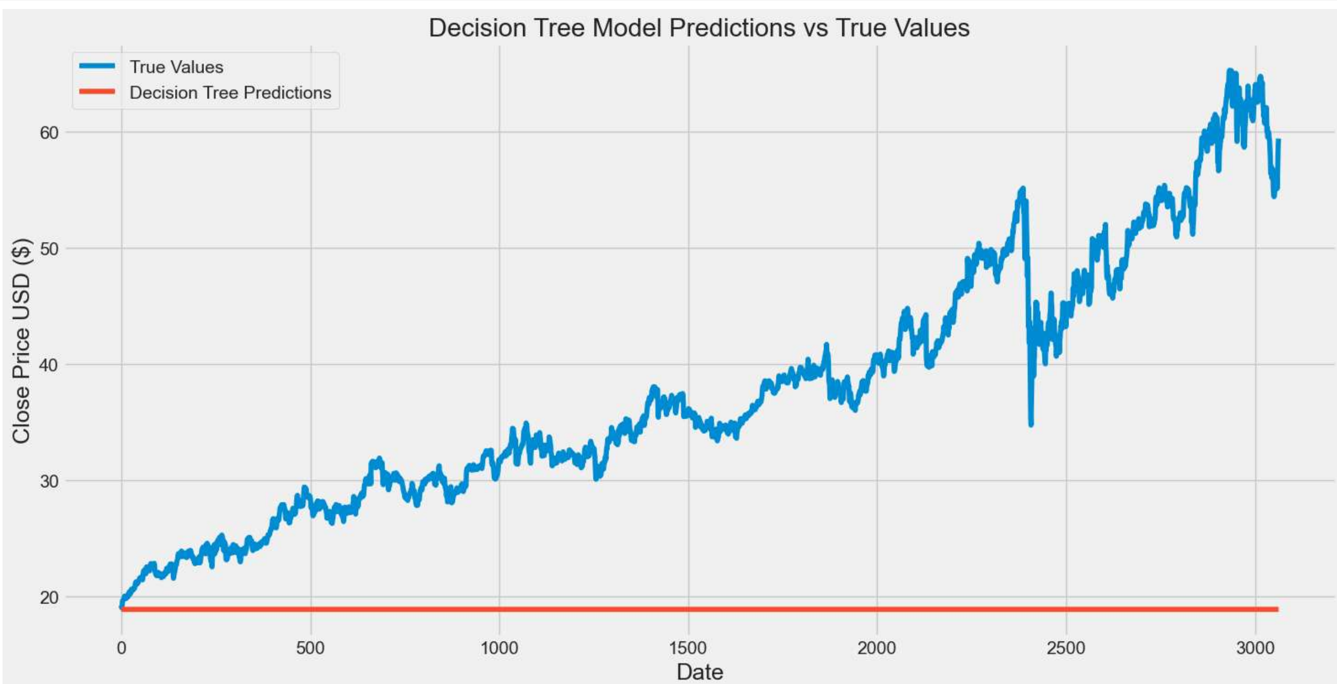
```
# Training the Decision Tree Regressor Model
dtr_model = DecisionTreeRegressor()
dtr_model.fit(X_train, y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

```
# Predicting and evaluating the Decision Tree model
dtr_predictions = dtr_model.predict(X_test)
dtr_rmse = np.sqrt(mean_squared_error(y_test, dtr_predictions))
print('Decision Tree Model RMSE:', dtr_rmse)
```

```
Decision Tree Model RMSE: 21.608274306195128
```

```
# Visualizing the Results for Decision Tree
plt.figure(figsize=(16,8))
plt.title('Decision Tree Model Predictions vs True Values')
plt.plot(y_test, label='True Values')
plt.plot(dtr_predictions, label='Decision Tree Predictions')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.legend()
plt.show()
```



Random Forest Regressor Model

```
# Preparing the data for Random Forest Regressor model
X = data.index.values.reshape(-1, 1)
y = data['Close'].values
```

```
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

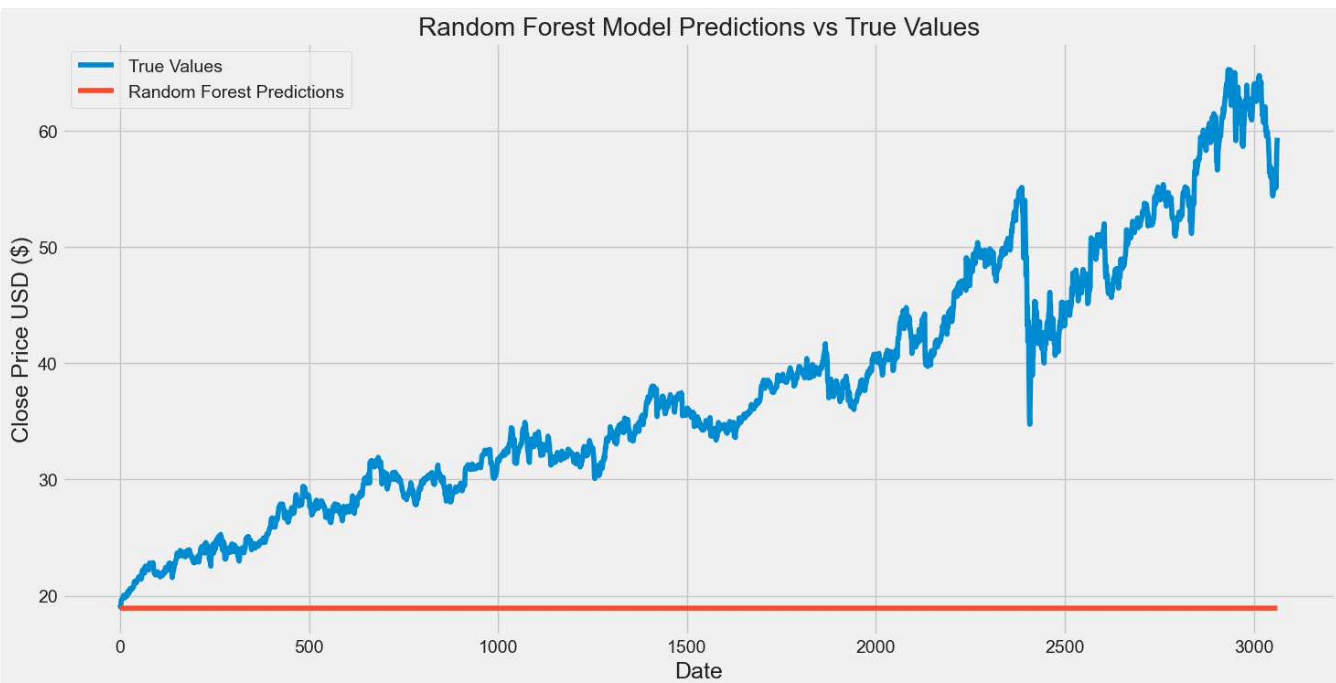
```
# Training the Random Forest Regressor Model
rfr_model = RandomForestRegressor(n_estimators=100)
rfr_model.fit(X_train, y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
# Predicting and evaluating the Random Forest model
rfr_predictions = rfr_model.predict(X_test)
rfr_rmse = np.sqrt(mean_squared_error(y_test, rfr_predictions))
print('Random Forest Model RMSE:', rfr_rmse)
```

```
Random Forest Model RMSE: 21.571138391366404
```

```
# Visualizing the Results for Random Forest
plt.figure(figsize=(16,8))
plt.title('Random Forest Model Predictions vs True Values')
plt.plot(y_test, label='True Values')
plt.plot(rfr_predictions, label='Random Forest Predictions')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.legend()
plt.show()
```



Conclusion

This project demonstrates the application of machine learning techniques in financial forecasting. By comparing different regression and deep learning models, valuable insights were gained into Coca-Cola's stock price behavior. The LSTM model, due to its ability to learn temporal patterns, proved particularly effective for this time series data. This internship project enhanced skills in data preprocessing, EDA, and ML modeling using Python libraries like Pandas, Sklearn, and Matplotlib.