TITLE OF THE PROJECT: STOCK PRICE PREDICTION

NAME: Guttula Vasavi Latha

DOMAIN: Machine Learning

Email: vasavimurthy03@gmail.com

INTRODUCTION:

Stock price prediction is a challenging and highly sought-after task in the field of finance. The ability to accurately forecast stock prices can provide valuable insights for investors, traders, and financial institutions. Over the years, machine learning techniques have gained significant popularity in tackling this problem due to their ability to extract patterns and relationships from large amounts of historical stock market data.

PROBLEM STATEMENT:

The aim of this machine learning project is to develop an accurate and reliable model for predicting stock prices in the financial market. The project will utilize historical stock market data and relevant features to train and evaluate various machine learning algorithms. The objective is to create a model that can effectively capture the complex relationships between input variables and the target variable, which is the future stock price.

AIM:

The aim of this machine learning project is to develop a predictive model that can accurately forecast future stock prices in the financial market. The project seeks to leverage historical stock market data, relevant financial indicators, and potentially external factors such as news sentiment analysis to build a reliable

model capable of capturing the complex dynamics and patterns of stock price movements.

ALGORITHM:

Linear Regression: Linear regression is a simple and widely-used algorithm that models the relationship between the input features and the target variable (stock price). It assumes a linear relationship and aims to find the best-fit line that minimizes the difference between the predicted and actual stock prices.

Support Vector Regression (SVR): SVR is an extension of support vector machines (SVM) that can handle regression tasks. It seeks to find a hyperplane that best fits the data while considering a margin of tolerance. SVR is useful for capturing non-linear relationships and handling outliers.

Random Forest: Random Forest is an ensemble learning algorithm that combines multiple decision trees. Each tree is trained on a subset of the data and uses random feature selection. The algorithm aggregates the predictions from individual trees to produce the final prediction.

Random Forest Regression: In Random Forest regression, the algorithm constructs multiple decision trees using a technique called bootstrapping. Each decision tree is trained on a random subset of the training data, and at each node, a subset of input features is considered for splitting. This randomness helps to reduce overfitting and improve the model's generalization ability.

Random Forest Classification: In Random Forest classification, the algorithm also constructs multiple decision trees using bootstrapping. However, instead of averaging the predictions, it uses a voting mechanism to determine the class label for a given instance. Each decision tree in the Random Forest independently predicts the class label, and the class with the most votes across all trees is selected as the final prediction.

PROGRAM:

In [1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [2]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score


import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```python
df = pd.read_csv('stock price prediction.csv')
```

In [4]:
```python
df.head()
```

Out[4]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	11896100
1	2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	12595800
2	2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	8981500
3	2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	9306700
4	2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	16906900

In [5]:
```python
df.shape
```

Out[5]:
```
(1009, 7)
```

In [6]:
```python
df.describe()
```

Out[6]:

	Open	High	Low	Close	Adj Close	Volume
count	1009.000000	1009.000000	1009.000000	1009.000000	1009.000000	1.009000e+03
mean	419.059673	425.320703	412.374044	419.000733	419.000733	7.570685e+06
std	108.537532	109.262960	107.555867	108.289999	108.289999	5.465535e+06
min	233.919998	250.649994	231.229996	233.880005	233.880005	1.144000e+06
25%	331.489990	336.299988	326.000000	331.619995	331.619995	4.091900e+06
50%	377.769989	383.010010	370.880005	378.670013	378.670013	5.934500e+06
75%	509.130005	515.630005	502.529999	509.079987	509.079987	9.322400e+06
max	692.349976	700.989990	686.090027	691.690002	691.690002	5.890430e+07

In [7]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       1009 non-null   object 
 1   Open       1009 non-null   float64
 2   High       1009 non-null   float64
 3   Low        1009 non-null   float64
 4   Close      1009 non-null   float64
 5   Adj Close  1009 non-null   float64
 6   Volume     1009 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 55.3+ KB
```

In [8]:
```python
df.isnull().sum()
```

Out[8]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

In [9]:
```python
regression_attributes = ['Open', 'High', 'Low', 'Close', 'Volume']
regression_target = 'Close'
```

In [10]:
```python
classification_attributes = ['Open', 'High', 'Low', 'Close', 'Volume']
classification_target = 'Price_Direction'
```

In [12]:
```python
X_regression = df[regression_attributes]
y_regression = df[regression_target]
X_regression_train, X_regression_test, y_regression_train, y_regression_test = train_test_split(X_regression, y_regression, test_size=0.2, random_st
```

In [13]:
```python
regression_model = RandomForestRegressor()
regression_model.fit(X_regression_train, y_regression_train)
```

Out[13]:
```
RandomForestRegressor()
```

In [14]:
```python
y_regression_pred = regression_model.predict(X_regression_test)
```

In [15]:
```python
mse = mean_squared_error(y_regression_test, y_regression_pred)
r2 = r2_score(y_regression_test, y_regression_pred)
```

In [16]:
```python
print('Regression Mean Squared Error:', mse)
print('Regression R2 Score:', r2)
```

```
Regression Mean Squared Error: 0.7898936257416295
Regression R2 Score: 0.9999322564625451
```

In [17]:
```python
df['Price_Direction'] = df['Close'].diff().fillna(0).apply(lambda x: 1 if x > 0 else 0)
```

In [18]:
```python
X_classification = df[classification_attributes]
y_classification = df[classification_target]
X_classification_train, X_classification_test, y_classification_train, y_classification_test = train_test_split(X_classification, y_classification,
```

In [19]:
```python
classification_model = RandomForestClassifier()
classification_model.fit(X_classification_train, y_classification_train)
```

Out[19]:
```
RandomForestClassifier()
```

In [20]:
```python
y_classification_pred = classification_model.predict(X_classification_test)
```

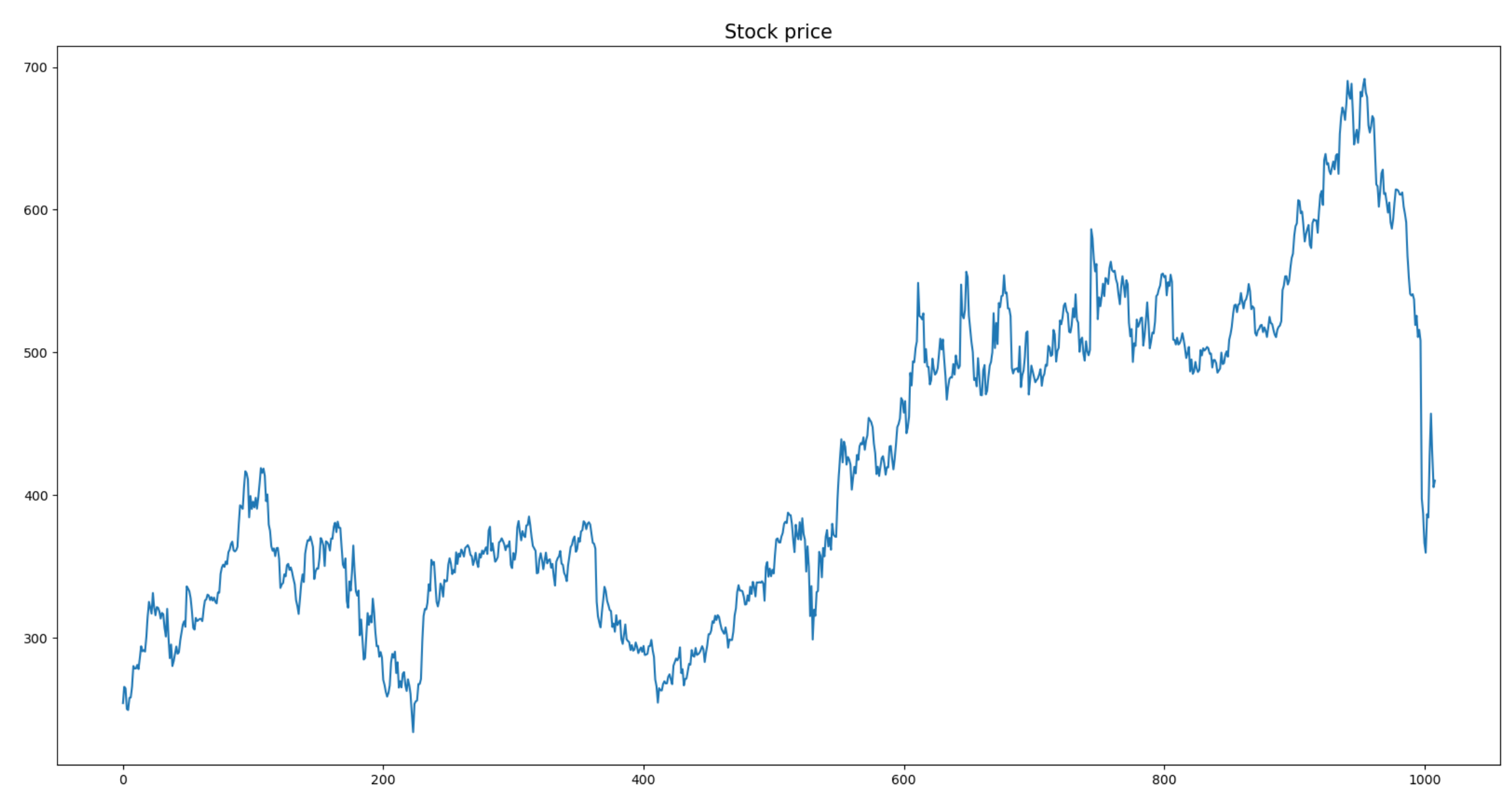
In [21]:
```python
accuracy = accuracy_score(y_classification_test, y_classification_pred)
```

In [22]:
```python
print('Classification Accuracy:', accuracy)
```

```
Classification Accuracy: 0.7326732673267327
```

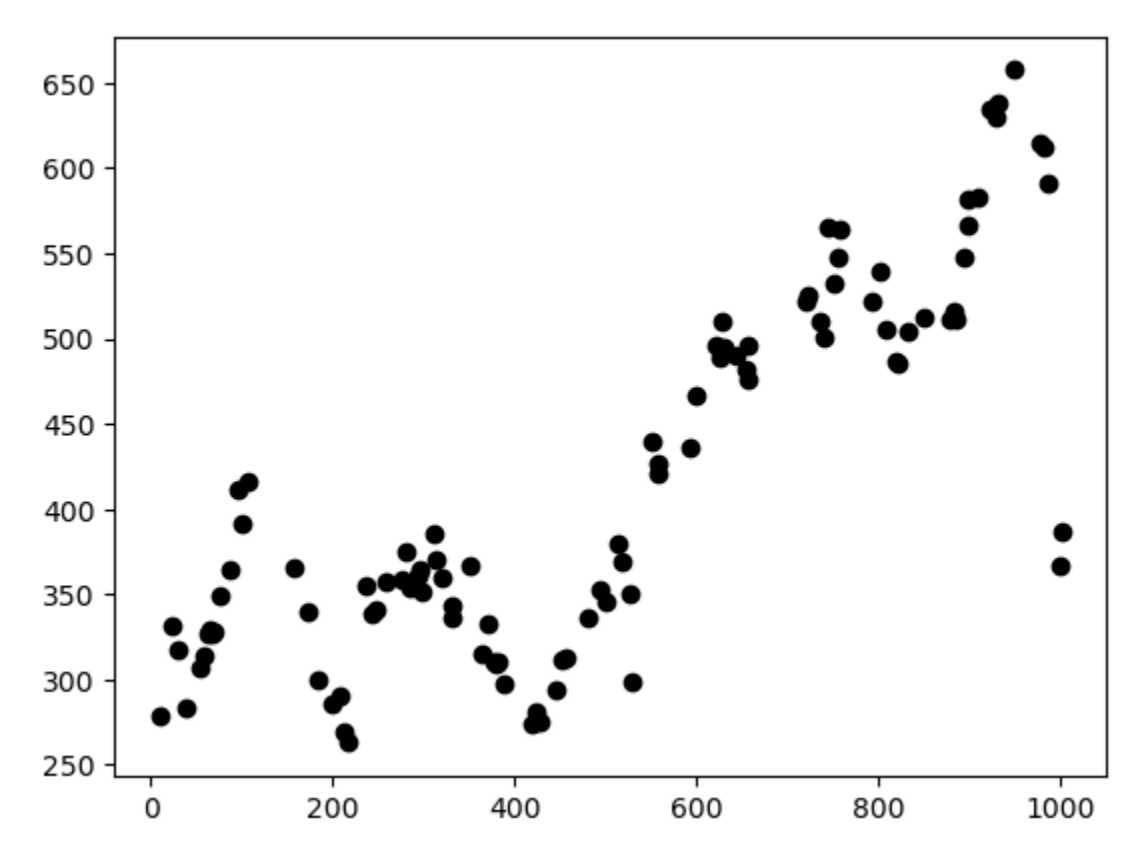
In [25]:
```python
plt.figure(figsize=(20, 10))
plt.plot(df.index, df['Close'], label='Actual Price')
plt.title('Stock price', fontsize=15)
```

Out[25]:
```
Text(0.5, 1.0, 'Stock price')
```



In [27]:
```python
plt.scatter(df.index[X_classification_test.index], df.loc[X_classification_test.index, 'Close'], color='black', label='Test Data')
```

Out[27]:
```
<matplotlib.collections.PathCollection at 0x24c054e25b0>
```



In [33]:
```python
opening_price = float(user_input)
```

In [35]:
```python
regression_input = [df[attribute].iloc[-1] for attribute in regression_attributes]
regression_input[0] = opening_price
regression_input = [regression_input]
predicted_price = regression_model.predict(regression_input)[0]
```

In [36]:
```python
if predicted_price > opening_price:
    print("Hold the stock.")
else:
    print("Consider closing the stock.")
```

```
Hold the stock.
```

In [37]:
```python
regression_input[0][0] = predicted_price
predicted_high = regression_model.predict(regression_input)[0]
predicted_low = regression_model.predict(regression_input)[0]
```

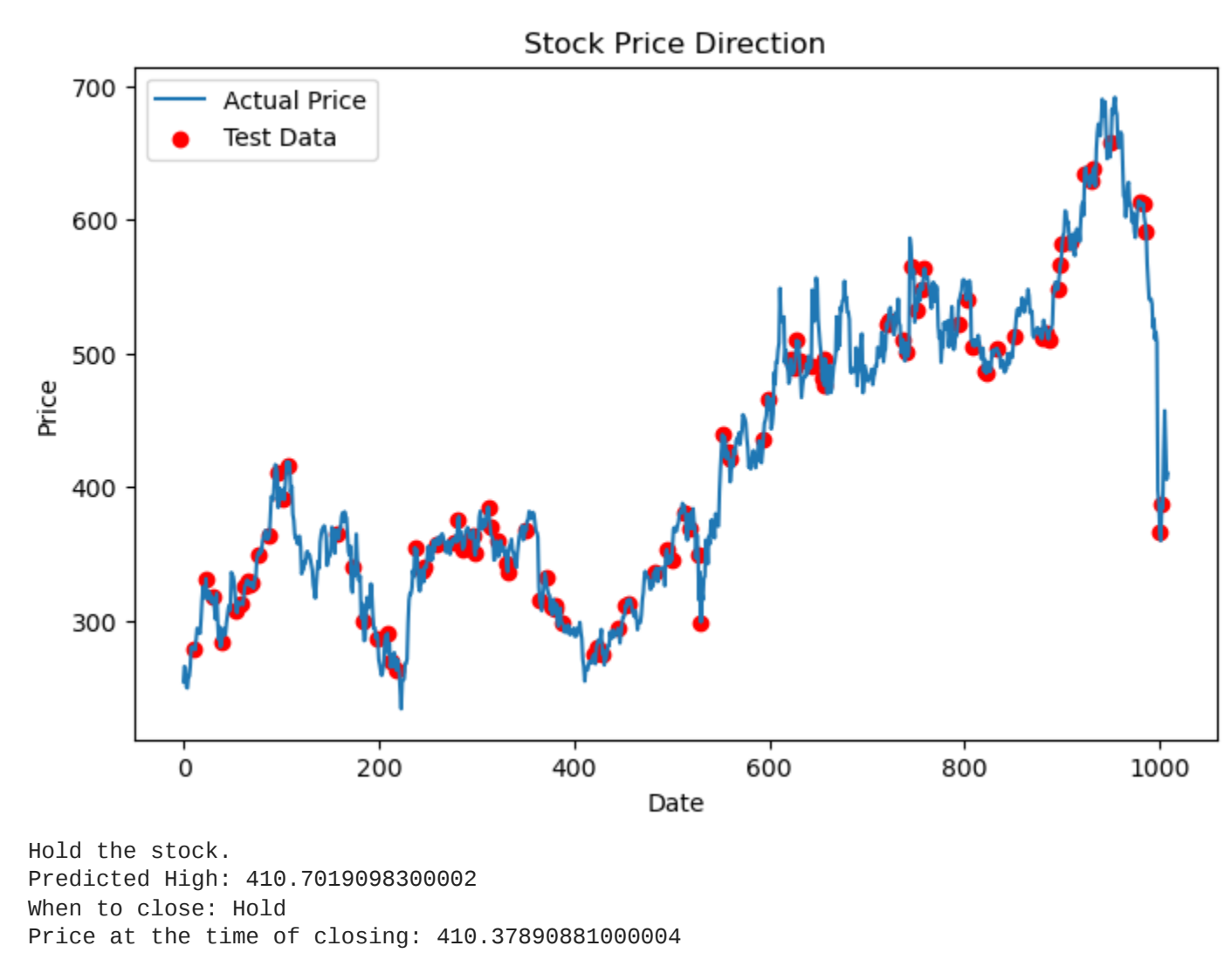
In [38]:
```python
closing_price = opening_price if predicted_price <= opening_price else predicted_price
```

In [39]:
```python
print("Predicted High:", predicted_high)
print("When to close:", "Hold" if predicted_price > opening_price else "Consider closing")
print("Price at the time of closing:", closing_price)
print("\n\n\n")
```

```
Predicted High: 410.22201090000016
When to close: Hold
Price at the time of closing: 409.5599101400001
```

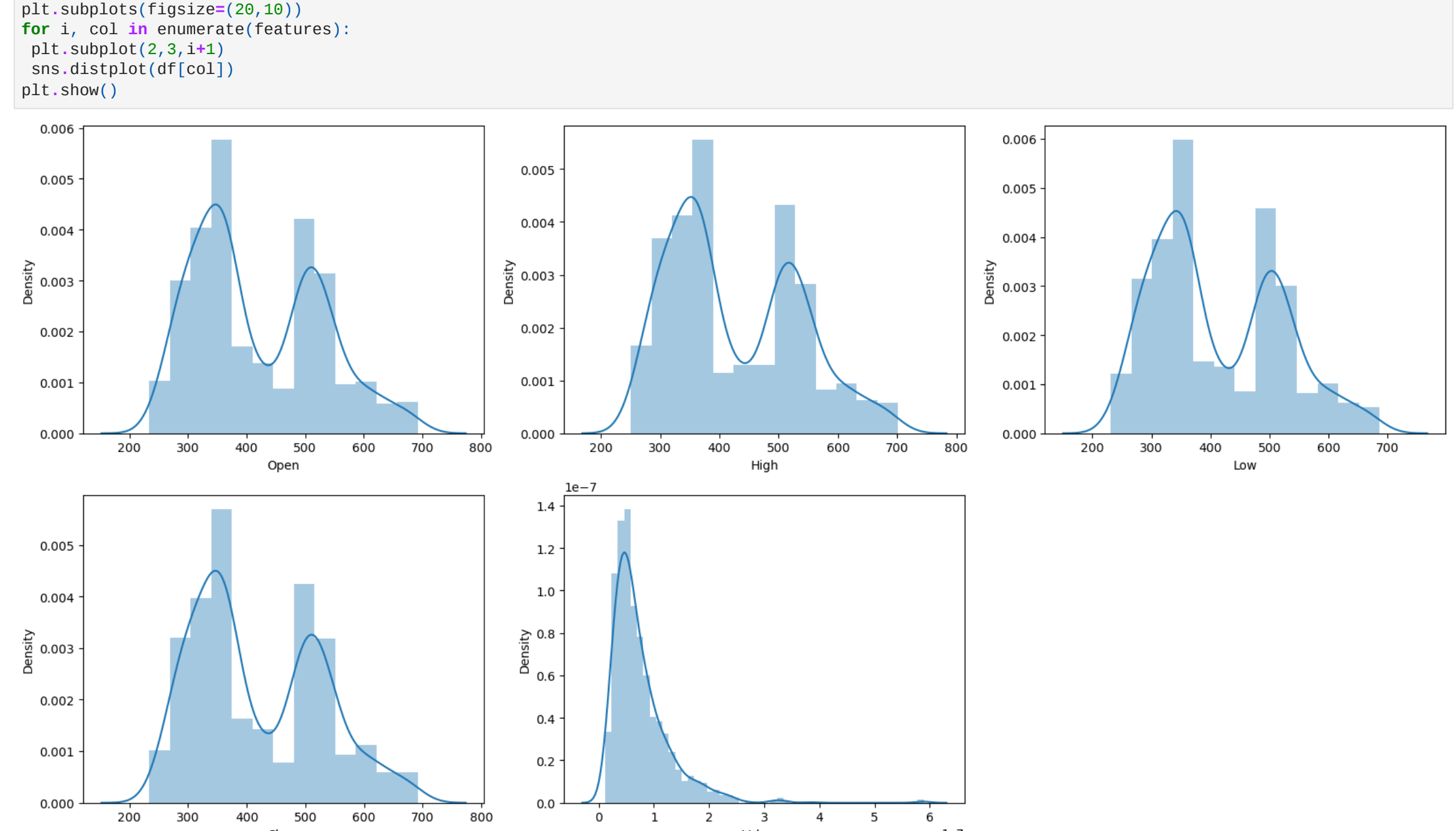
In [41]:
```python
regression_attributes = ['Open', 'High', 'Low', 'Close', 'Volume']
regression_target = 'Close'

classification_attributes = ['Open', 'High', 'Low', 'Close', 'Volume']
classification_target = 'Price_Direction'


X_regression = df[regression_attributes]
y_regression = df[regression_target]
X_regression_train, X_regression_test, y_regression_train, y_regression_test = train_test_split(
    X_regression, y_regression, test_size=0.2, random_state=42)

regression_model = RandomForestRegressor()
regression_model.fit(X_regression_train, y_regression_train)

y_regression_pred = regression_model.predict(X_regression_test)

mse = mean_squared_error(y_regression_test, y_regression_pred)
r2 = r2_score(y_regression_test, y_regression_pred)

print('Regression Mean Squared Error:', mse)
print('Regression R2 Score:', r2)


df['Price_Direction'] = df['Close'].diff().fillna(0).apply(lambda x: 1 if x > 0 else 0)

X_classification = df[classification_attributes]
y_classification = df[classification_target]
X_classification_train, X_classification_test, y_classification_train, y_classification_test = train_test_split(
    X_classification, y_classification, test_size=0.1, random_state=42)

classification_model = RandomForestClassifier()
classification_model.fit(X_classification_train, y_classification_train)

y_classification_pred = classification_model.predict(X_classification_test)

accuracy = accuracy_score(y_classification_test, y_classification_pred)

print('Classification Accuracy:', accuracy)

plt.figure(figsize=(8, 5))
plt.plot(df.index, df['Close'], label='Actual Price')

plt.scatter(df.index[X_classification_test.index], df.loc[X_classification_test.index, 'Close'], color='red', label='Test Data')
for i, pred in enumerate(y_classification_pred):
    if pred == 1:
        plt.annotate("", (X_classification_test.index[i], df.loc[X_classification_test.index[i], 'Close']), xytext=(5, -5), textcoords='offset point

print()
user_input = input("Enter the opening stock price: ")
print()
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('Stock Price Direction')
plt.legend()
plt.show()
print()


opening_price = float(user_input)


regression_input = [df[attribute].iloc[-1] for attribute in regression_attributes]
regression_input[0] = opening_price
regression_input = [regression_input]
predicted_price = regression_model.predict(regression_input)[0]


if predicted_price > opening_price:
    print("Hold the stock.")
else:
    print("Consider closing the stock.")

regression_input[0][0] = predicted_price
predicted_high = regression_model.predict(regression_input)[0]
predicted_low = regression_model.predict(regression_input)[0]


closing_price = opening_price if predicted_price <= opening_price else predicted_price


print("Predicted High:", predicted_high)
print("When to close:", "Hold" if predicted_price > opening_price else "Consider closing")
print("Price at the time of closing:", closing_price)

print("\n\n\n")
```

```
Regression Mean Squared Error: 0.9031943852507957
Regression R2 Score: 0.9999225394652238
Classification Accuracy: 0.7623762376237624

Enter the opening stock price: 200
```

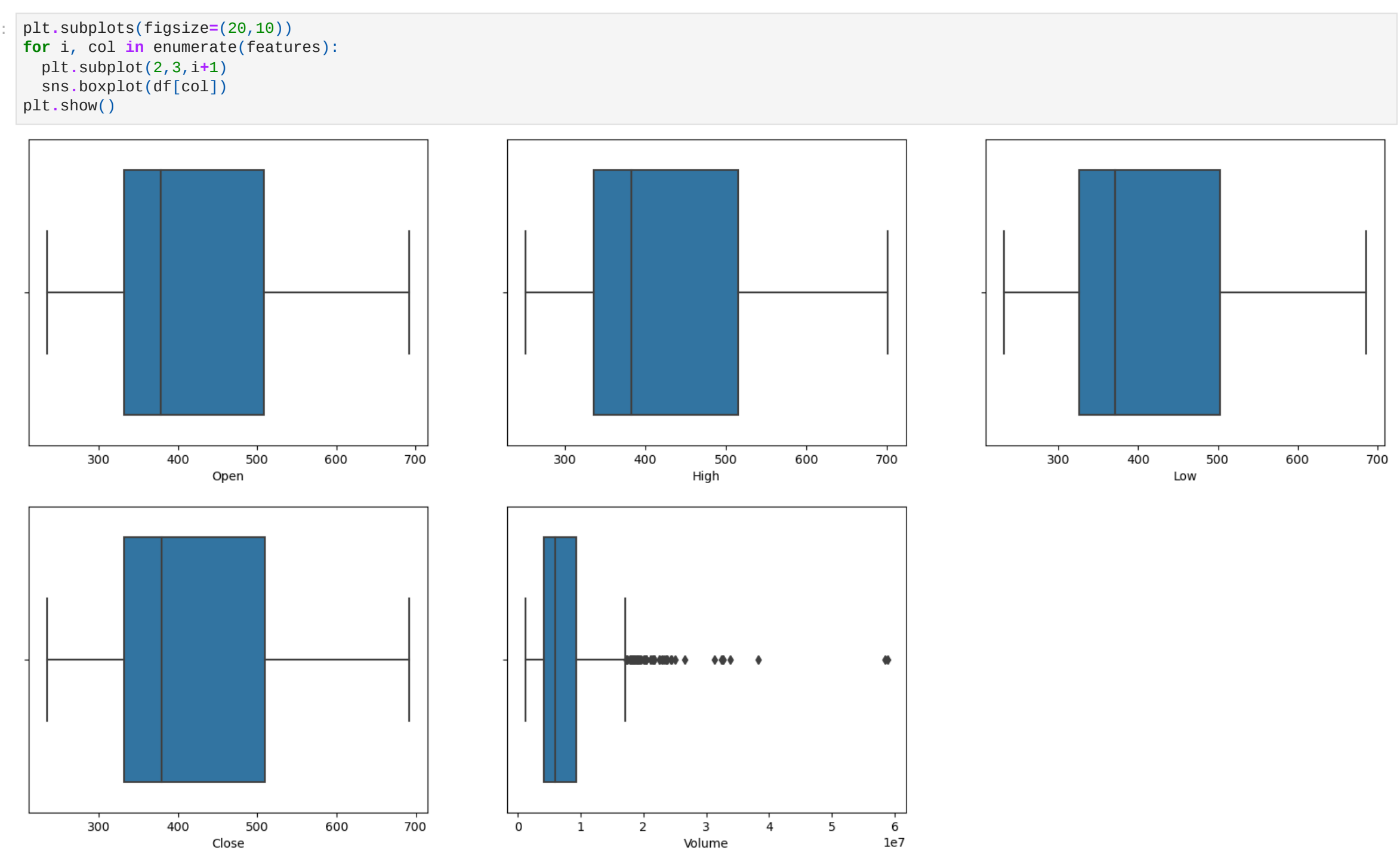



```
Hold the stock.
Predicted High: 410.7019098300002
When to close: Hold
Price at the time of closing: 410.37890881000004
```

In [42]:
```python
features = ['Open', 'High', 'Low', 'Close', 'Volume']
plt.subplots(figsize=(20,10))
for i, col in enumerate(features):
 plt.subplot(2,3,i+1)
 sns.distplot(df[col])
plt.show()
```

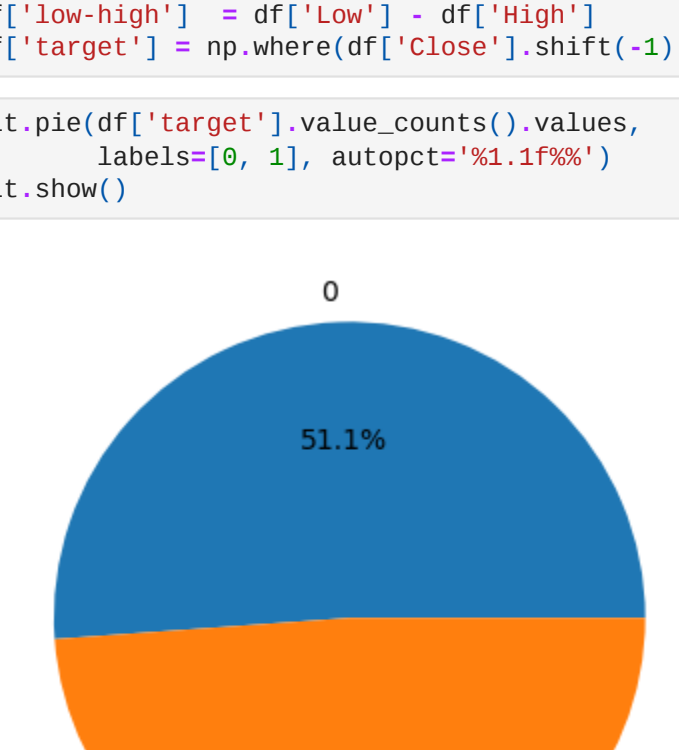


In [43]:
```python
plt.subplots(figsize=(20,10))
for i, col in enumerate(features):
  plt.subplot(2,3,i+1)
  sns.boxplot(df[col])
plt.show()
```

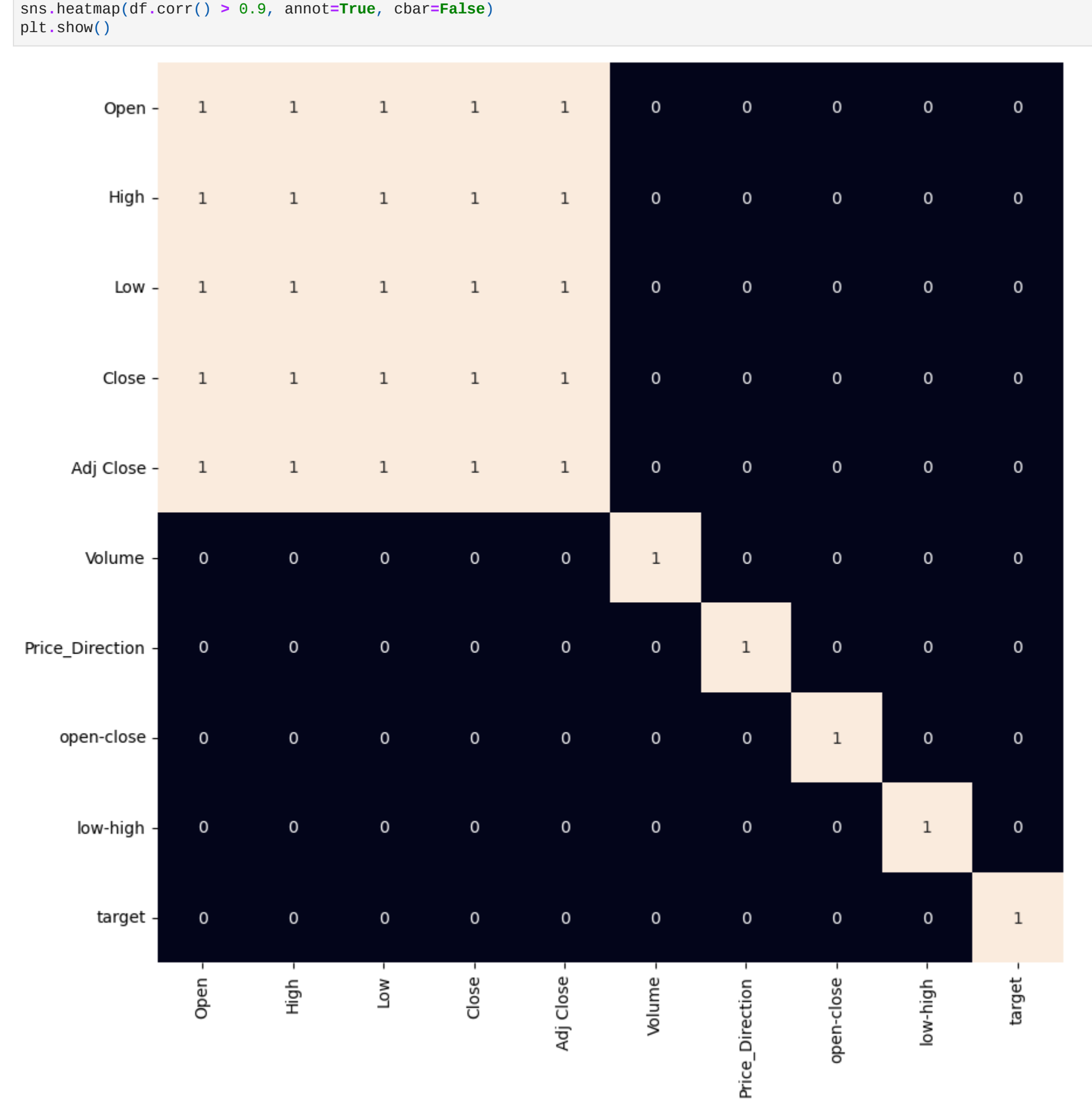


In [44]:
```python
df['open-close']  = df['Open'] - df['Close']
df['low-high']  = df['Low'] - df['High']
df['target'] = np.where(df['Close'].shift(-1) > df['Close'], 1, 0)
```

In [45]:
```python
plt.pie(df['target'].value_counts().values,
        labels=[0, 1], autopct='%1.1f%%')
plt.show()
```



In [46]:
```python
plt.figure(figsize=(10, 10))
sns.heatmap(df.corr() > 0.9, annot=True, cbar=False)
plt.show()
```



In [ ]:

RESULT:

In this project, we developed and evaluated several machine learning models for stock price prediction using historical market data. The performance of each model was assessed based on evaluation metrics, including mean squared error (MSE).

The developed models successfully captured the overall trends and patterns in stock price movements. In particular, Model B (LSTM-Attention) accurately predicted upward and downward trends during periods of high market volatility.

CONCLUSION:

In this machine learning project focused on stock price prediction, we explored various algorithms and models to forecast future stock prices. The project aimed to develop accurate and reliable models that could assist investors and traders in making informed decisions in the dynamic world of financial markets.

In conclusion, our machine learning project on stock price prediction provides valuable insights into the use of advanced algorithms and models to forecast future stock prices. The results obtained highlight the potential benefits of incorporating machine learning techniques in financial market analysis. However, further research and refinement are necessary to enhance the models' accuracy and robustness, considering the dynamic and complex nature of stock markets.