Project Report

Statistics For Data Science

Semester — II

HDFC Bank Stock Prices

By Mahir Mavani

Reg No :- 2411021240003

Git Hub Link :- https://github.com/MahirMavani/sds-assignment

Department of Computer Application

Alliance University

Chandapura — Anekal Main Road, Anekal

Bengaluru — 562 106

April 2025

Introduction

- This project focuses on data analysis using Python, specifically on analyzing historical stock data of HDFC Bank. It combines various data science techniques to uncover insights from time series financial data using libraries like pandas, matplotlib, and seaborn.

Project Overview

- The goal of the project is to load and analyze the stock market data of HDFC Bank, clean and preprocess the dataset, perform exploratory data analysis (EDA), and generate insightful visualizations. The data was sourced in CSV format and includes key information such as open, close, high, low prices, and trading volume over a period of time.

Project Insights

- A detailed understanding of the data types and structure of financial datasets.
- Visualization helped in identifying trends in stock prices and volumes over time.
- Correlation analysis revealed the relationships between different stock indicators like closing price, volume, and high-low range.

Project Challenges

- Handling missing values and ensuring the dataset was clean for analysis.
- Choosing the right types of graphs and charts to visualize financial trends meaningfully.
- Managing large amounts of time-series data efficiently for smoother analysis.

Project Goals

- Develop proficiency in using Python for data analysis.
- Understand stock data behavior through visualization.
- Learn to summarize and interpret real-world financial datasets.
- Improve problem-solving skills using data science tools.

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
df=pd.read_csv(r"D:\downloads\HDFCBANK.NS.csv")
df
```

```
         Date      Open      High       Low     Close  \
0  1996-01-01  3.030000  3.030000  2.925000  2.980000
1  1996-01-02  2.980000  3.025000  2.950000  2.975000
2  1996-01-03  2.975000  2.995000  2.950000  2.985000
```

```
3     1996-01-04     2.985000     2.980000     2.940000     2.965000
4     1996-01-05     2.965000     2.980000     2.950000     2.960000
...          ...          ...          ...          ...          ...
6562  2022-01-17  1530.000000  1556.000000  1519.150024  1521.500000
6563  2022-01-18  1533.000000  1550.900024  1523.000000  1529.250000
6564  2022-01-19  1534.000000  1539.750000  1513.349976  1518.449951
6565  2022-01-20  1528.449951  1528.500000  1500.099976  1509.000000
6566  2022-01-21  1500.000000  1529.800049  1485.599976  1521.599976

        Adj Close      Volume
0        2.417746    350000.0
1        2.413689    412000.0
2        2.421803    284000.0
3        2.405575    282000.0
4        2.401519    189000.0
...           ...         ...
6562  1521.500000  11494686.0
6563  1529.250000   6170576.0
6564  1518.449951   7158813.0
6565  1509.000000   7598923.0
6566  1521.599976   5768339.0

[6567 rows x 7 columns]
```

Check for missing values and data types

```
print('Missing values:')
print(df.isnull().sum())

print('\nData types:')
print(df.dtypes)
```

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

Data types:
Date          datetime64[ns]
Open                 float64
High                 float64
Low                  float64
Close                float64
Adj Close            float64
Volume               float64
```

```
Next_Close           float64
dtype: object
```

A brief descriptive statistics overview

```
df.describe()
```

```
                                Date         Open         High          Low  \
count                           6552  6552.000000  6552.000000  6552.000000
mean   2008-11-07 22:28:21.098901248   337.005717   340.404149   333.226588
min              1996-01-01 00:00:00     2.435000     2.435000     2.395000
25%              2002-04-10 18:00:00    23.753749    24.090000    23.303750
50%              2008-09-22 12:00:00   133.282501   136.925003   130.095001
75%              2015-06-04 06:00:00   513.462509   518.424988   509.062492
max              2022-01-20 00:00:00  1705.000000  1725.000000  1671.000000
std                              NaN   428.946730   432.700515   424.688284

             Close    Adj Close        Volume   Next_Close
count  6552.000000  6552.000000  6.552000e+03  6552.000000
mean    336.894411   328.001586  4.941659e+06   337.122263
min       2.435000     1.975574  0.000000e+00     2.435000
25%      23.645000    20.435167  1.560665e+06    23.648750
50%     133.197502   121.573917  3.426695e+06   133.272499
75%     513.737488   495.121086  6.362624e+06   513.825012
max    1688.699951  1688.699951  2.011300e+08  1688.699951
std     428.701835   426.924740  5.714749e+06   428.930570
```

Convert 'Date' column to datetime

```
df['Date'] = pd.to_datetime(df['Date'])
df.head(5)
```

```
        Date   Open   High    Low  Close  Adj Close    Volume
0 1996-01-01  3.030  3.030  2.925  2.980   2.417746  350000.0
1 1996-01-02  2.980  3.025  2.950  2.975   2.413689  412000.0
2 1996-01-03  2.975  2.995  2.950  2.985   2.421803  284000.0
3 1996-01-04  2.985  2.980  2.940  2.965   2.405575  282000.0
4 1996-01-05  2.965  2.980  2.950  2.960   2.401519  189000.0
```

Sort by date

```
df = df.sort_values(by='Date')
df.head(5)
```

```
        Date   Open   High    Low  Close  Adj Close    Volume
0 1996-01-01  3.030  3.030  2.925  2.980   2.417746  350000.0
1 1996-01-02  2.980  3.025  2.950  2.975   2.413689  412000.0
2 1996-01-03  2.975  2.995  2.950  2.985   2.421803  284000.0
3 1996-01-04  2.985  2.980  2.940  2.965   2.405575  282000.0
4 1996-01-05  2.965  2.980  2.950  2.960   2.401519  189000.0
```

Check basic info

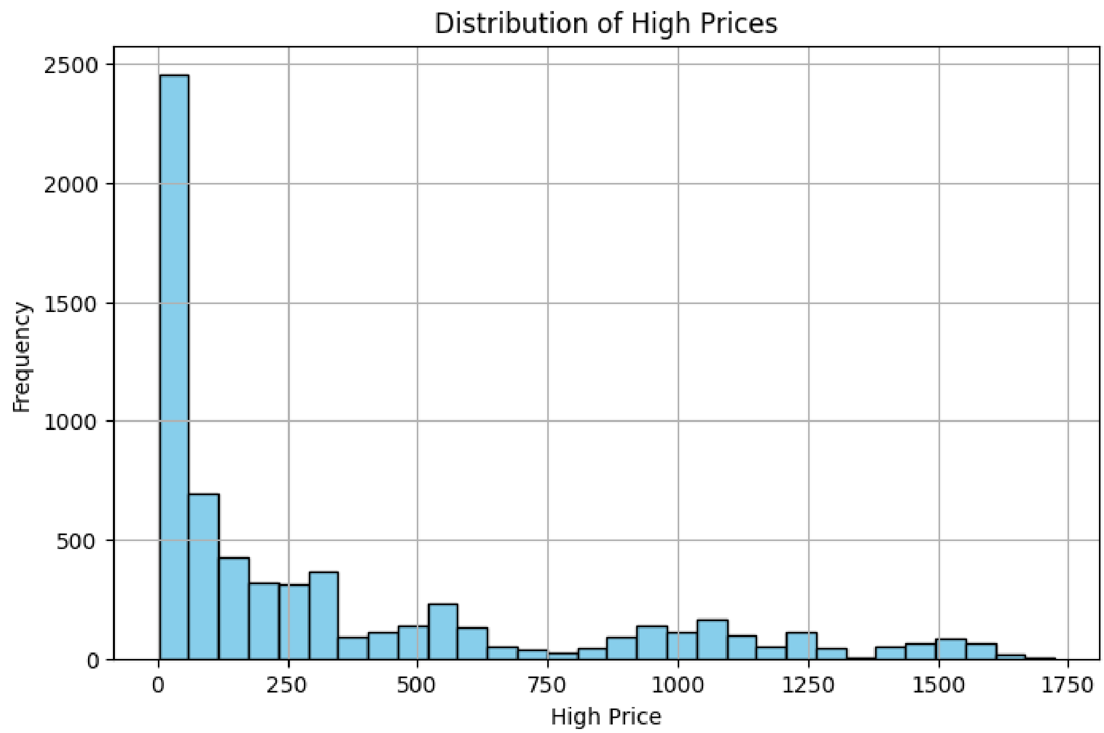
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6567 entries, 0 to 6566
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       6567 non-null   datetime64[ns]
 1   Open       6560 non-null   float64
 2   High       6560 non-null   float64
 3   Low        6560 non-null   float64
 4   Close      6560 non-null   float64
 5   Adj Close  6560 non-null   float64
 6   Volume     6560 non-null   float64
dtypes: datetime64[ns](1), float64(6)
memory usage: 359.3 KB
None
```

Visualizing the Data

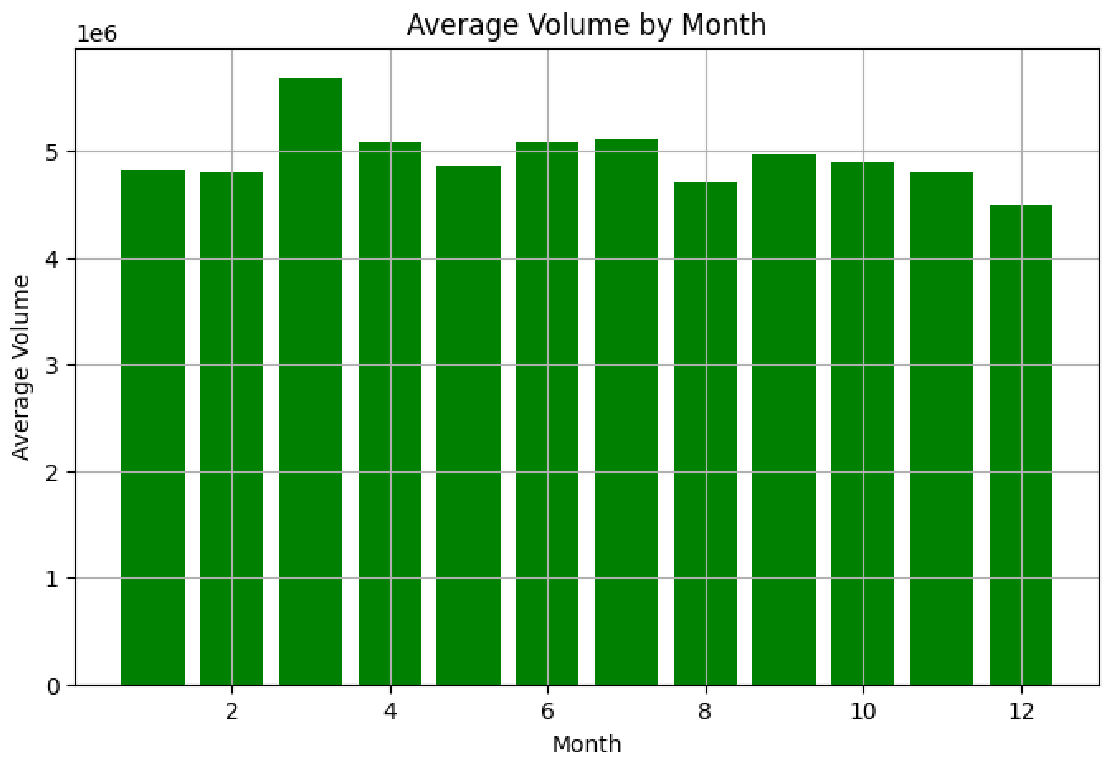
Histogram of 'High' prices

```
import matplotlib.pyplot as plt
import pandas as pd

plt.figure(figsize=(8, 5))
plt.hist(df['High'], bins=30, color='skyblue', edgecolor='black')
plt.title("Distribution of High Prices")
plt.xlabel("High Price")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()
```



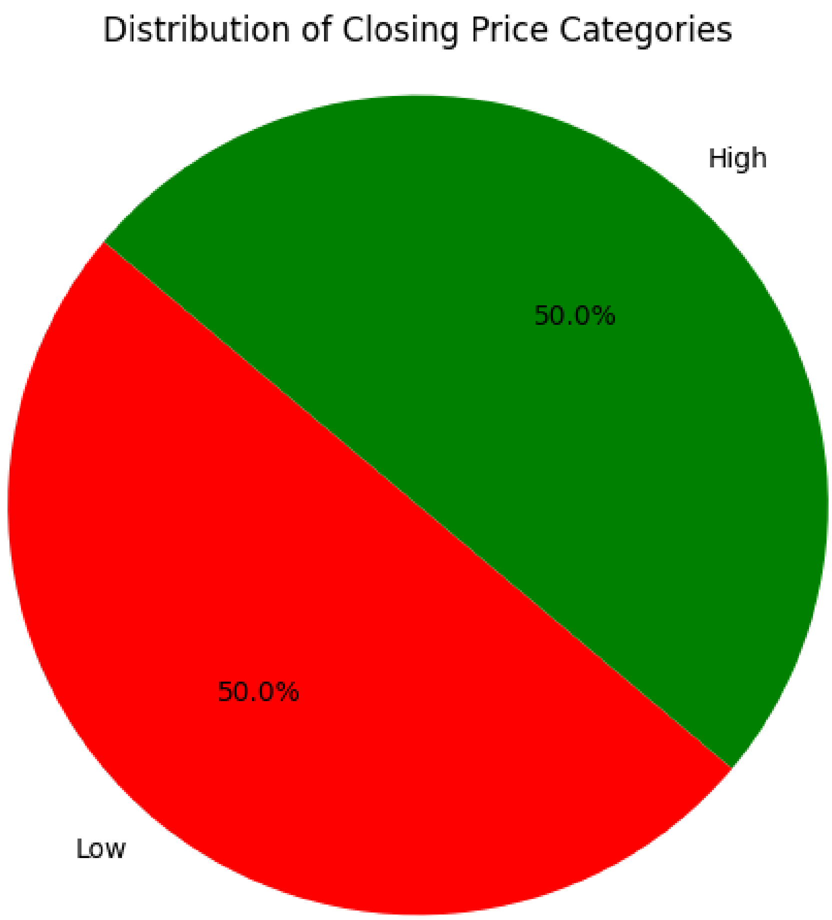
Bar plot

```
# Convert Date to datetime
df['Date'] = pd.to_datetime(df['Date'])
df['Month'] = df['Date'].dt.month

# Average volume by month
monthly_avg = df.groupby('Month')['Volume'].mean()

# Bar plot
plt.figure(figsize=(8, 5))
plt.bar(monthly_avg.index, monthly_avg.values, color='green')
plt.title("Average Volume by Month")
plt.xlabel("Month")
plt.ylabel("Average Volume")
plt.grid(True)
plt.show()
```



Pie chart of categories

```
median_close = df['Close'].median()
df['Close_Category'] = df['Close'].apply(lambda x: 'High' if x > median_close
else 'Low')

counts = df['Close_Category'].value_counts()

plt.figure(figsize=(6, 6))
plt.pie(counts, labels=counts.index, autopct='%1.1f%%',
colors=['red','green'], startangle=140)
plt.title("Distribution of Closing Price Categories")
plt.axis('equal')
plt.show()
```



Creating a Box Plot

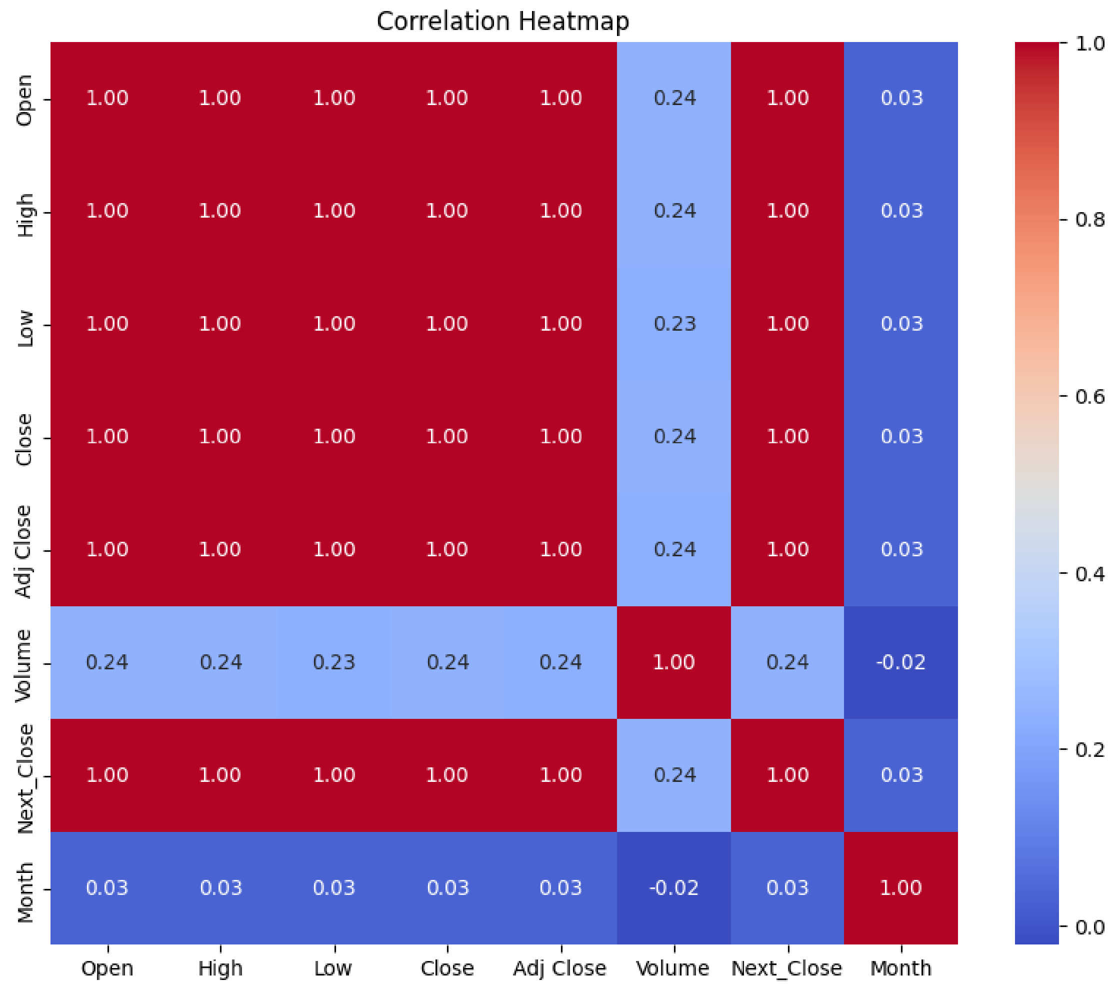
```
# Convert date and extract month
df['Date'] = pd.to_datetime(df['Date'])
df['Month'] = df['Date'].dt.month

# Box plot: Close price distribution across months
plt.figure(figsize=(10, 6))
sns.boxplot(x='Month', y='Close', data=df, palette='Set3')
plt.title("Box Plot of Close Prices by Month")
plt.xlabel("Month")
plt.ylabel("Closing Price")
plt.grid(True)
plt.show()
```

```
C:\Users\Mahir\AppData\Local\Temp\ipykernel_32212\781908893.py:7:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.
```

```
sns.boxplot(x='Month', y='Close', data=df, palette='Set3')
```

Creating a Correlation Heatmap

```
if 'Date' in df.columns:
    df['Date'] = pd.to_datetime(df['Date'])
numeric_df = df.select_dtypes(include='number')
correlation_matrix = numeric_df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```

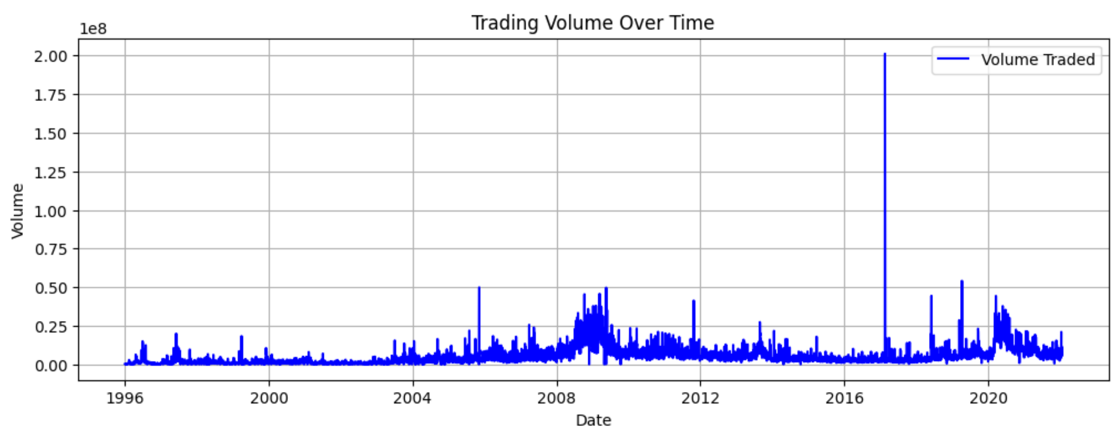


Plotting closing price over time

```
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Close'], label='Close Price', color='green')
plt.title("HDFC Bank Stock Price Over Time")
plt.xlabel("Date")
plt.ylabel("Closing Price")
plt.grid(True)
plt.legend()
plt.show()
```

Displaying Volume trends over the time

```
plt.figure(figsize=(12, 4))
plt.plot(df['Date'], df['Volume'], label='Volume Traded', color='blue')
plt.title("Trading Volume Over Time")
plt.xlabel("Date")
plt.ylabel("Volume")
plt.grid(True)
plt.legend()
plt.show()
```



Creating log feature for prediction

```
df['Next_Close'] = df['Close'].shift(-1)
df = df.dropna()
```

Cretaing Features and target

```
X = df[['Open', 'High', 'Low', 'Close', 'Volume']]
y = df['Next_Close']
```

Train-test split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
shuffle=False)
```

Training model

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

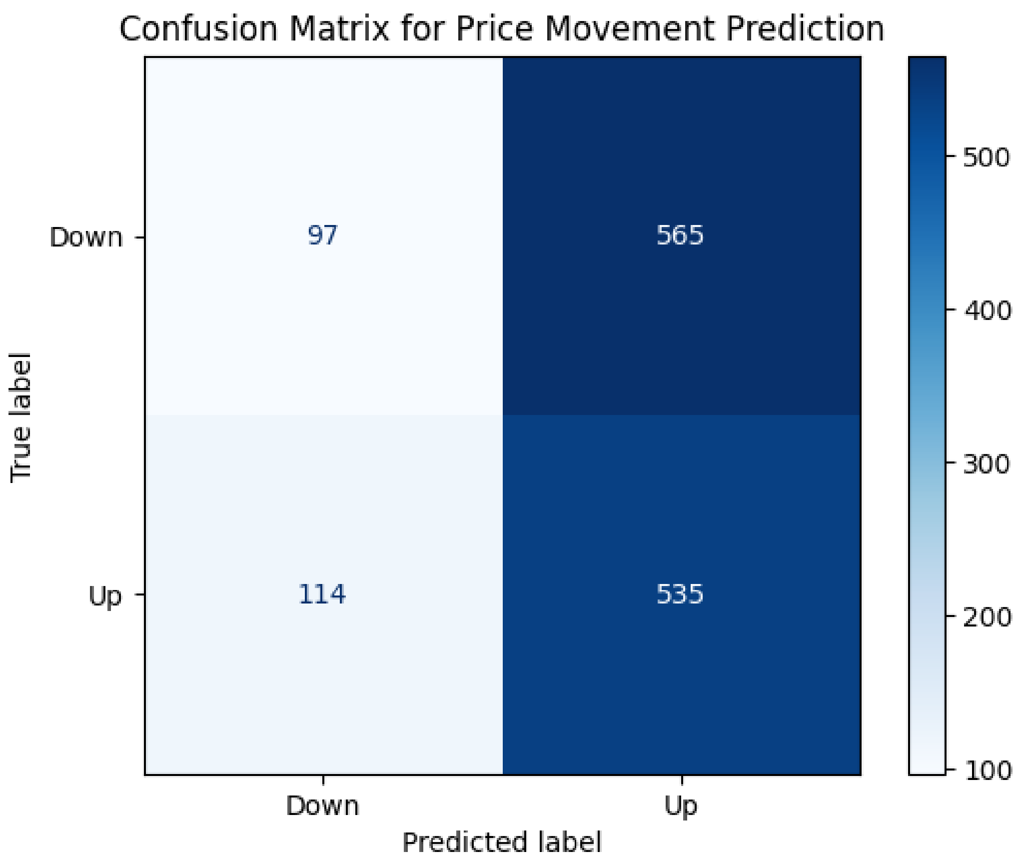
Evaluating the model that is trained

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"MAE: {mae:.2f}")
print(f"RMSE: {rmse:.2f}")
print(f"R² Score: {r2:.2f}")

MAE: 11.52
RMSE: 17.31
R² Score: 1.00
```

Creating a Confusion Matrix for model Evaluation

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Prepare data: classify price movement
df['Price_Up'] = (df['Close'].shift(-1) > df['Close']).astype(int)  # 1 =
price went up next day
df = df.dropna()  # drop last row with NaN target

# Features and target
X = df[['Open', 'High', 'Low', 'Close', 'Volume']]
y = df['Price_Up']

# Split 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)

# Train logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predict
```

```python
y_pred = model.predict(X_test)

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Down",
"Up"])
disp.plot(cmap='Blues')
plt.title("Confusion Matrix for Price Movement Prediction")
plt.show()
```



What I Learned from This Project

- How to read and manipulate CSV files using pandas.
- Techniques to identify and handle missing or inconsistent data.
- How to create and interpret visualizations using matplotlib and seaborn.
- Understanding of stock market terms and indicators through hands-on analysis

Conclusion

- This project was an insightful introduction to financial data analysis using Python. It improved my data handling and visualization skills and deepened my understanding of stock market trends. Working with real-world data presented unique challenges, which ultimately enhanced my analytical thinking and technical abilities.