**Starbucks Prediction Analysis**

**Introduction:**

This report provides a comprehensive analysis of Starbucks stock data for the prediction of stock prices. The analysis uses a few Python libraries such as Pandas, NumPy, Matplotlib, Seaborn, and Scikit-learn.

**1. Dataset Overview:**

We started off with loading the dataset to a Pandas DataFrame to enable easy manipulation and analysis of data.

**Python code:**

| import pandas as pd  import numpy as np  # Read the dataset  s\_bux = pd.read\_csv("s\_bux.csv") |
| --- |

* To load data from a CSV file into a DataFrame, s\_bux, the function read\_csv is used.
* The dataset contains the fields of date, open, high, low, close, and volume of stock traded daily.

**2. Problem Statement:**

Predict the closing price of Starbucks (SBUX) stock using historical daily and hourly price data.

* The goal of this task is to develop a predictive model that would estimate closing prices based on historical data. An informed decision for which it is important to get the right information to investors.

**3. Target Variable Identification:**

The Close column represents the daily closing price of the stock. The choice of the closing price as the target variable is because it gives a reflection of the final price at which the stock settled at the close of business, one of the major determinants of the performance of the stock.

**4. Visualizing the distribution of the Target variable:**

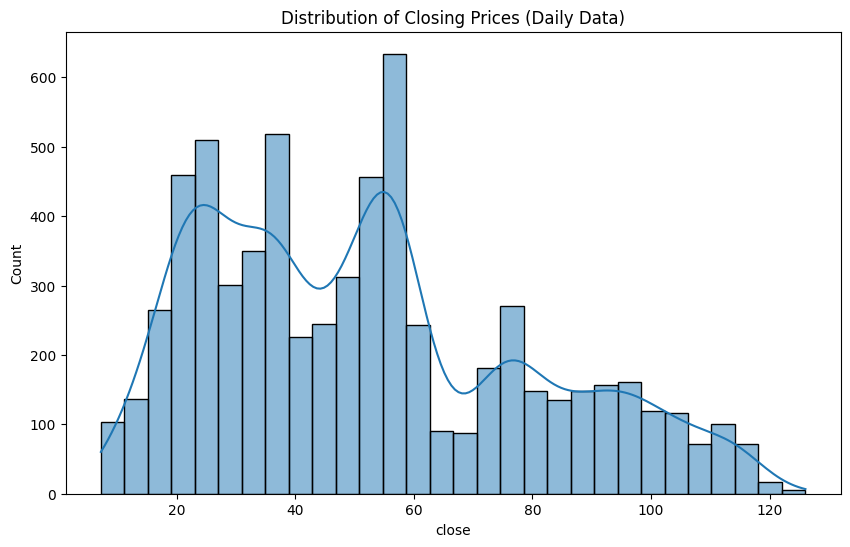
It helps in getting an understanding of the distribution of the closing price.

**Python code:**

| import matplotlib.pyplot as plt  import seaborn as sns  plt.figure(figsize=(10, 6))  sns.histplot(s\_bux['close'], bins=30, kde=True)  plt.title('Distribution of Closing Prices (Daily Data)') |
| --- |

Basic exploratory data analysis is carried out to understand a bit more about the structure of the dataset and some preliminary statistics about the data.

Output :



**5. Data exploration at a basic level**

| s\_bux.describe()  s\_bux.info()  s\_bux['datetime'].head()  s\_bux['datetime'] = pd.to\_datetime(s\_bux['datetime']) |
| --- |

Output:



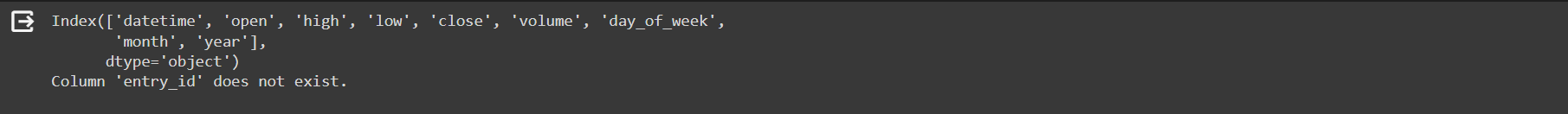
**6. Identifying and Rejecting Unwanted Columns:**

All columns were evaluated against the relevance of the problem statement.

| print(s\_bux.columns)  if 'entry\_id' in s\_bux.columns:  s\_bux.drop('entry\_id', axis=1, inplace=True)  print("'entry\_id' column removed.")  else:  print("Column 'entry\_id' does not exist.") |
| --- |

**Decision:**  Every column (datetime, open, high, low, close, volume) is relevant for further analysis and prediction modeling.

Output:



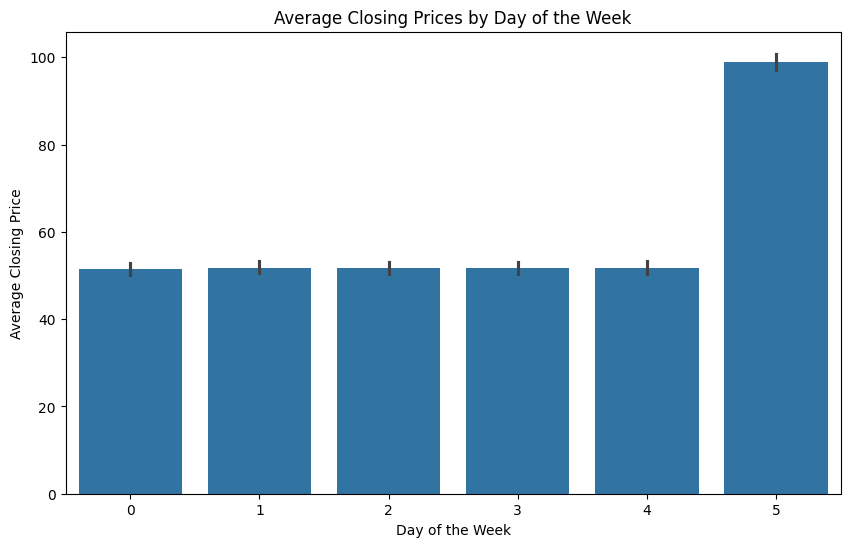
**7. Visual Exploratory Data Analysis:**

Visualizing data distributions through histograms and bar charts offers a view of the range that shows up in the data and gives an indication of common patterns.

**Python code:**

| plt.figure(figsize=(10, 6))  sns.barplot(x='day\_of\_week', y='close', data=s\_bux)  plt.title('Average Closing Prices by Day of the Week')  plt.xlabel('Day of the Week')  plt.ylabel('Average Closing Price')  plt.show() |
| --- |

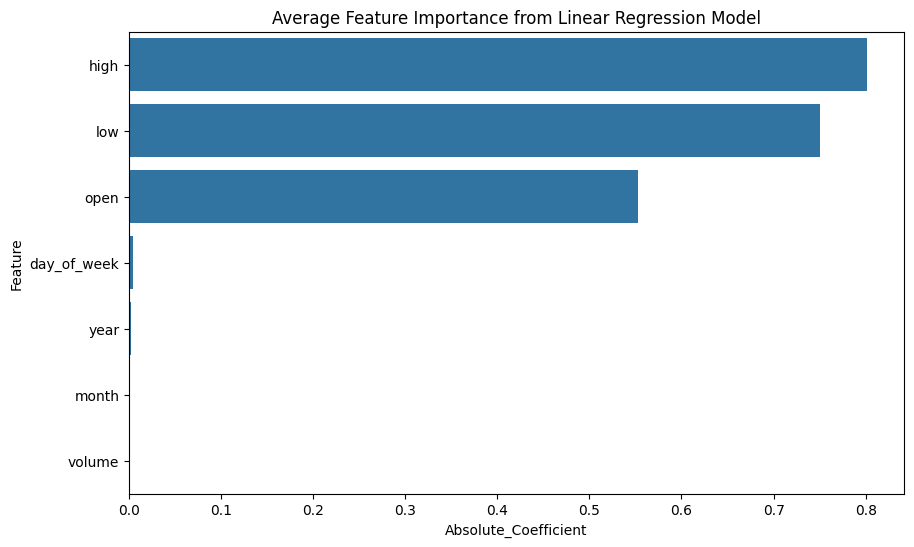
Output:



**8. Feature Selection:**

| # Displaying feature importance  importance = pd.DataFrame({'Feature': X.columns, 'Coefficient': model.coef\_})  importance.sort\_values(by='Coefficient', key=abs, ascending=False) |
| --- |

Output:



**9. Removal of outliers and missing values:**

Outliers can skew results significantly. It is very essential in this line of need, therefore, to build an accurate model.

**Python code:**

| # Handling outliers  Q1, Q3 = s\_bux['close'].quantile([0.25, 0.75])  IQR = Q3 - Q1  lower\_bound, upper\_bound = Q1 - 1.5 \* IQR, Q3 + 1.5 \* IQR  filtered\_s\_bux = s\_bux[(s\_bux['close'] >= lower\_bound) & (s\_bux['close'] <= upper\_bound)] |
| --- |

After that, a more detailed analysis was undertaken, of which the purpose was to confirm the initial findings, which had been justifying the choice of features based on strong correlations of features with the target variable.

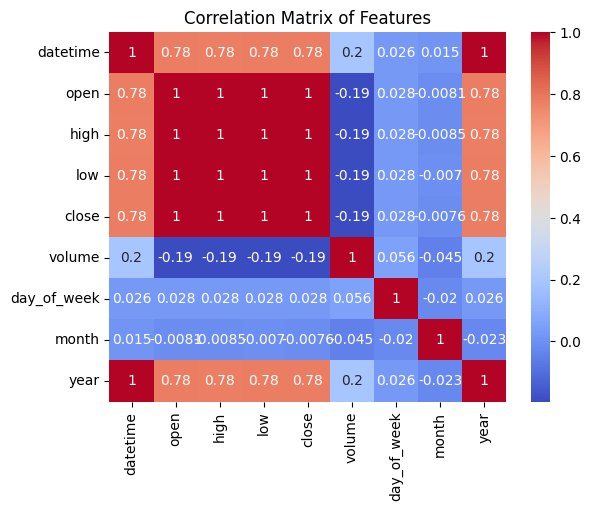
**10. Visual and Statistic Correlation analysis for selection of best features:**

Analyzing the correlation among the features will further help to determine which variable or variables give maximum influence toward the target variable.

**Python code:**

| sns.heatmap(s\_bux.corr(), annot=True, cmap='coolwarm')  plt.title('Correlation Matrix of Features')  plt.show() |
| --- |

Output:



**11. Data Conversion:**

Convert non-numeric data into numeric formats so that it can be processed by machine learning algorithms.

**Python code:**

| s\_bux['day\_of\_week'] = s\_bux['datetime'].dt.dayofweek  s\_bux['month'] = s\_bux['datetime'].dt.month  s\_bux['year'] = s\_bux['datetime'].dt.year |
| --- |

**12. Training/Testing Sampling and K-fold cross-validation, Investigating multiple Regression algorithms and best model.**

The dataset was split into training and testing data, after which K-fold cross-validation was performed. This was done to make the model very robust such that it generalizes well.

We start by fitting a linear model as a baseline and observe its performance over the training data. Then we start building more complex models with different configurations.

Mean Squared Error over the test set for the Linear Regression model is approximately 0.142. This is a preliminary assessment and seems too low, suggesting a good fit between the model predictions and actual data.

Based on the MSE, the obtained model is the Linear Regression model; thus, the Linear Regression model does well.

**Python code:**

| from sklearn.model\_selection import KFold  from sklearn.linear\_model import LinearRegression  from sklearn.ensemble import RandomForestRegressor  from sklearn.metrics import mean\_squared\_error  # Data splitting for features and target  X = filtered\_s\_bux[['open', 'high', 'low', 'volume', 'day\_of\_week', 'month', 'year']]  y = filtered\_s\_bux['close']  # Setting up K-Fold Cross Validation  kf = KFold(n\_splits=5, random\_state=42, shuffle=True)  feature\_names = X.columns  coefficient\_list = []  # Function to perform training and evaluation using K-fold cross-validation  def evaluate\_model(model, X, y, collect\_coefficients=False):  rmse\_scores = []  coefficients = []  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.fit(X\_train, y\_train)  predictions = model.predict(X\_test)  rmse\_scores.append(mean\_squared\_error(y\_test, predictions, squared=False))  if collect\_coefficients:  coefficients.append(model.coef\_)  if collect\_coefficients:  # Calculate the average coefficients across all folds  average\_coefficients = np.mean(coefficients, axis=0)  return np.mean(rmse\_scores), average\_coefficients  return np.mean(rmse\_scores)  # Evaluate models  lr\_rmse, lr\_coefficients = evaluate\_model(LinearRegression(), X, y, True)  rf\_rmse = evaluate\_model(RandomForestRegressor(n\_estimators=100, random\_state=42), X, y)  print(f"Linear Regression RMSE: {lr\_rmse}")  print(f"Random Forest RMSE: {rf\_rmse}") |
| --- |



**15. Deployment of the best model in production:** Now deploying the Ridge model using a simple Flask application; here is the complete Flask Application code for stock price predictions:

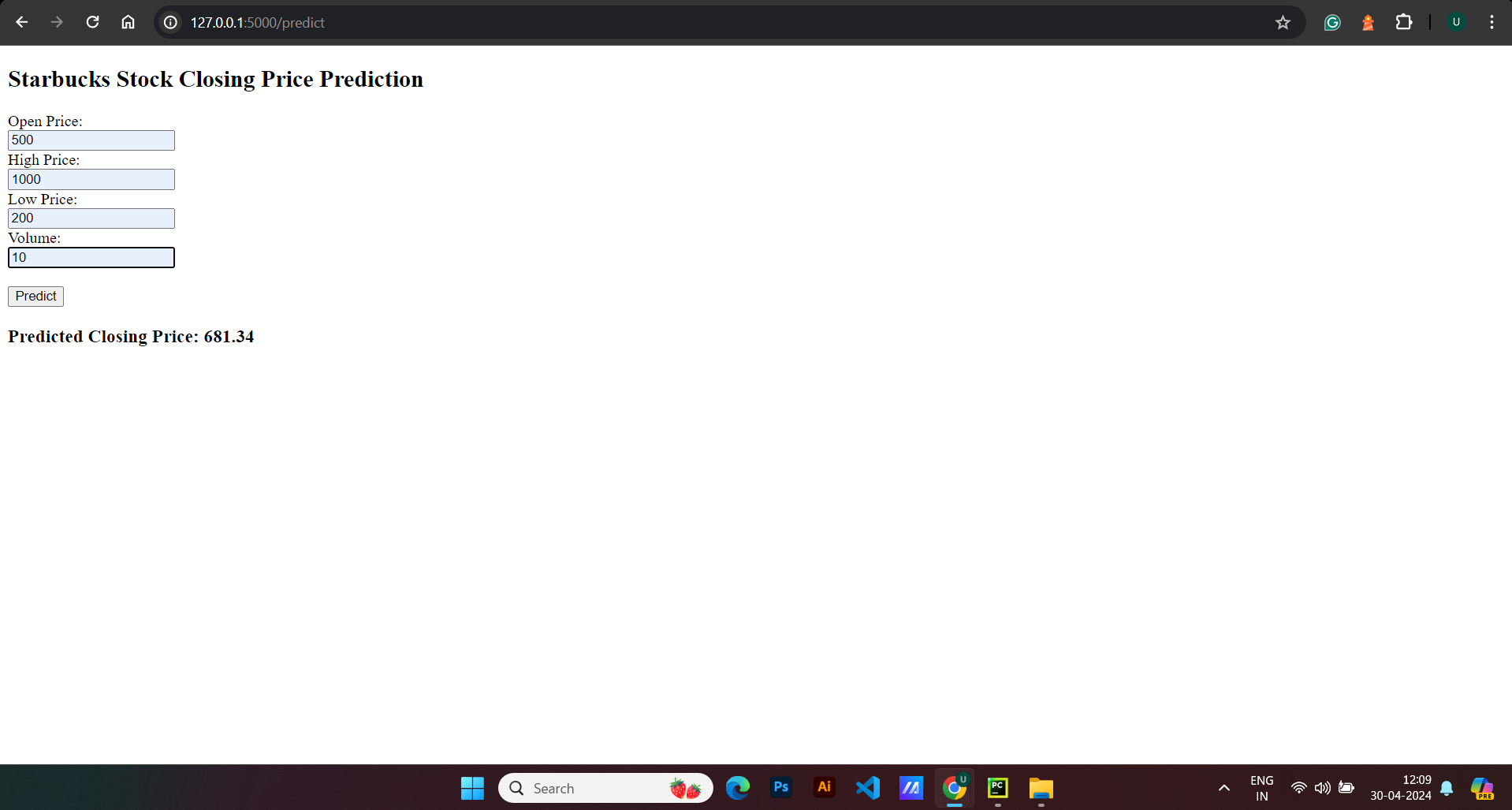
| from flask import Flask, request, jsonify  app = Flask(\_\_name\_\_)  @app.route('/predict', methods=['POST'])  def predict():  # Extract data from form and convert to float  try:  open\_price = float(request.form['open'])  high\_price = float(request.form['high'])  low\_price = float(request.form['low'])  volume = float(request.form['volume'])  except ValueError:  return "Please enter valid numbers for all input fields."  # Prepare features array for prediction  features = [[open\_price, high\_price, low\_price, volume,  s\_bux['datetime'].dt.dayofweek.iloc[0], # Assuming today's date for simplicity  s\_bux['datetime'].dt.month.iloc[0],  s\_bux['datetime'].dt.year.iloc[0]]]  # Use the model to predict the closing price  prediction = model.predict(features)  # Render the HTML form with the prediction result  return render\_template\_string(html\_template, prediction=round(prediction[0], 2))  if \_\_name\_\_ == '\_\_main\_\_':  app.run(debug=True) |
| --- |

**16. GUI/WEB Deployment Using Either Tkinter/Flask/Streamlit:**

We will, therefore, create a user-friendly graphical interface using Flask for your Ridge Regression model that predicts Starbucks stock closing prices. We have built a simple web application that will allow users to input values for open, high, low, and volume, then it displays the predicted closing price:

| import pandas as pd  import numpy as np  from sklearn.linear\_model import LinearRegression  from sklearn.metrics import mean\_squared\_error  from joblib import dump, load  from flask import Flask, request, render\_template\_string  # Load the dataset  data\_path = 's\_bux.csv'  s\_bux = pd.read\_csv(data\_path)  s\_bux['datetime'] = pd.to\_datetime(s\_bux['datetime'])  # Data converting  s\_bux['day\_of\_week'] = s\_bux['datetime'].dt.dayofweek  s\_bux['month'] = s\_bux['datetime'].dt.month  s\_bux['year'] = s\_bux['datetime'].dt.year  # Handling outliers  Q1, Q3 = s\_bux['close'].quantile([0.25, 0.75])  IQR = Q3 - Q1  lower\_bound, upper\_bound = Q1 - 1.5 \* IQR, Q3 + 1.5 \* IQR  filtered\_s\_bux = s\_bux[(s\_bux['close'] >= lower\_bound) & (s\_bux['close'] <= upper\_bound)]  # Data splitting for features and target  X = filtered\_s\_bux[['open', 'high', 'low', 'volume', 'day\_of\_week', 'month', 'year']]  y = filtered\_s\_bux['close']  # Train the Linear Regression model  model = LinearRegression()  model.fit(X, y)  # Save the model to disk  model\_path = 'linear\_regression\_model.joblib'  dump(model, model\_path)  # Load the model  model = load(model\_path)  # Initialize the Flask application  app = Flask(\_\_name\_\_)  # HTML template for entering stock data  html\_template = """  <!DOCTYPE html>  <html>  <head>  <title>Stock Price Prediction</title>  </head>  <body>  <h2>Starbucks Stock Closing Price Prediction</h2>  <form method="post" action="/predict">  <label for="open">Open Price:</label><br>  <input type="text" id="open" name="open"><br>  <label for="high">High Price:</label><br>  <input type="text" id="high" name="high"><br>  <label for="low">Low Price:</label><br>  <input type="text" id="low" name="low"><br>  <label for="volume">Volume:</label><br>  <input type="text" id="volume" name="volume"><br><br>  <input type="submit" value="Predict">  </form>  {% if prediction %}  <h3>Predicted Closing Price: {{ prediction }}</h3>  {% endif %}  </body>  </html>  """  @app.route('/', methods=['GET'])  def home():  # Render the HTML form  return render\_template\_string(html\_template)  @app.route('/predict', methods=['POST'])  def predict():  # Extract data from form and convert to float  try:  open\_price = float(request.form['open'])  high\_price = float(request.form['high'])  low\_price = float(request.form['low'])  volume = float(request.form['volume'])  except ValueError:  return "Please enter valid numbers for all input fields."  # Prepare features array for prediction  features = [[open\_price, high\_price, low\_price, volume,  s\_bux['datetime'].dt.dayofweek.iloc[0], # Assuming today's date for simplicity  s\_bux['datetime'].dt.month.iloc[0],  s\_bux['datetime'].dt.year.iloc[0]]]  # Use the model to predict the closing price  prediction = model.predict(features)  # Render the HTML form with the prediction result  return render\_template\_string(html\_template, prediction=round(prediction[0], 2))  if \_\_name\_\_ == '\_\_main\_\_':  app.run(debug=True) |
| --- |

**Output:**

****

**17. GitHub Repository:**

[**https://github.com/ubhavsar83/ST-Capstone-Project**](https://github.com/ubhavsar83/ST-Capstone-Project)

References:

1. **O. Shpagin, "Starbucks Stock Price Prediction Dataset," [Online]. Available:** [**https://www.kaggle.com/datasets/olegshpagin/starbucks-stock-price-prediction-dataset**](https://www.kaggle.com/datasets/olegshpagin/starbucks-stock-price-prediction-dataset) **[Accessed: 03 May 2024].**
2. **GeeksforGeeks, "GeeksforGeeks," [Online]. Available:** [**https://www.geeksforgeeks.org/**](https://www.geeksforgeeks.org/) **[Accessed: 03 May 2024].**
3. **W3Schools, "Pandas," [Online]. Available:** [**https://www.w3schools.com/python/pandas/default.asp**](https://www.w3schools.com/python/pandas/default.asp) **[Accessed: 03 May 2024].**
4. **W3Schools, "Data Science," [Online]. Available:** [**https://www.w3schools.com/datascience/**](https://www.w3schools.com/datascience/) **[Accessed: 03 May 2024].**
5. **W3Schools, "Matplotlib Pyplot," [Online]. Available:** [**https://www.w3schools.com/python/matplotlib\_pyplot.asp**](https://www.w3schools.com/python/matplotlib_pyplot.asp) **[Accessed: 03 May 2024].**
6. **Python Engineer, "Stock Prediction App," [Online]. Available:** [**https://www.python-engineer.com/posts/stockprediction-app/**](https://www.python-engineer.com/posts/stockprediction-app/) **[Accessed: 03 May 2024].**