STOCK PRICE PREDICTION USING APPLIED DATA SCIENCE

Phase 5 submission document

Project Title: Stock Price Prediction

ABSTRACT

In the past decades, there is an increasing interest in predicting markets among economists, policymakers, academics and market makers. The objective of the proposed work is to study and improve the supervised learning algorithms to predict the stock price. Stock Market Analysis of stocks using data mining will be useful for new investors to invest in stock market based on the various factors considered by the software. Stock market includes daily activities like Sensex calculation, exchange of shares. The exchange provides an efficient and transparent market for trading in equity, debt instruments and derivatives. Our aim is to create software that analyses previous stock data of certain companies, with help of certain parameters that affect stock value. We are going to implement these values in data mining algorithms and we will be able to decide which algorithm gives the best result. This will also help us to determine the values that stock will have in near future. We will determine the patterns in data with help of machine learning algorithms.

OVERVIEW

In recent times stock market predictions is gaining more attention, maybe due to the fact that if the trend of the market is successfully predicted the investors may be better guided. The profits gained by investing and trading in the stock market greatly depends on the predictability. If there is a system that can consistently predict the direction of the dynamic stock market will enable the users of the system to make informed decisions. Moreover, the predicted trends of the market will help there regulators of the market in taking corrective measures.

AIM AND OBJECTIVE

The aim of the project is to examine a number of different forecasting techniques to predict future stock returns based on past returns and numerical news indicators to construct a portfolio of multiple stocks in order to diversify the risk. We do this by applying supervised learning methods for stock price forecasting by interpreting the seemingly chaotic market data.

STOCK PRICE PREDICTION

Stock price prediction is the process of using historical data and various analytical methods to forecast the future price movements of individual stocks or the overall stock market. The goal is to make informed predictions about whether a stock's price will go up, down, or remain stable, which is valuable for investment and trading decisions. This involves the use of statistical models, machine learning algorithms, and financial analysis to make these forecasts.

A stock market, equity market or share market is the aggregation of buyers and sellers (a loose network of economic transactions, not a physical facility or discrete entity) of stocks (also called shares), which represent ownership claims on businesses; these may include securities listed on a public stock exchange as well as those only traded privately. Examples of the latter include shares of private companies which are sold to investors through equity crowd funding platforms. Stock exchanges list shares of common equity as well as other security types, e.g. corporate bonds and convertible bonds.

Stock price prediction is one of the most widely studied problem, attracting researchers from many fields. The volatile nature of the stock market makes it difficult to apply simple time-series or regression techniques. Financial institutions and active traders have created various proprietary models to beat the market for themselves or their clients, but rarely did anyone achieve consistently higher than the average returns on investment. The challenge of stock market price forecasting is so appealing because an improvement of just a few points of percentage can increase the profit by millions of dollars. This paper discusses the application of Support Vector Machines and Linear Regression in detail along with the pros and cons of the given methods. The paper introduces the parameters and variables which can be used to recognize the patterns in stock prices which can be helpful in future stock prediction and how boosting can be integrated with various other machine learning algorithms to improve the accuracy of our prediction systems.

MOTIVATION

Stock price prediction is a classic and important problem. With a successful model for stock prediction, we can gain insight about market behaviour over time, spotting trends that would otherwise not have been noticed. With the increasingly computational power of the computer, machine learning will be an efficient method to solve this problem. Thus, our motivation is to design a public service incorporating historical data and users predictions to make a stronger model that will benefit everyone.

Here's a list of tools and software commonly used in the process:

In the process of stock price prediction using data science, a wide range of tools and software can be employed to collect, analyze, model, and visualize data. Here is a comprehensive list of tools and software commonly used for stock price prediction:

1. Programming Language: -

1. Python
2. R

Python is the most popular language for machine learning due toits extensive libraries and frameworks.

2. Data Analysis and Manipulation Libraries:

1. Pandas
2. NumPy
3. SciPy

3. Integrated Development Environment (IDE): -

1. Jupyter Notebook
2. RStudio
3. Visual Studio Code

Choose an IDE for coding and running machine learning experiments. Some popular options include Jupyter Notebook, GoogleColab, or traditional IDEs like PyCharm.

4. Machine Learning Libraries: -

1. Scikit-Learn (Python)
2. TensorFlow and Keras (Python)
3. PyTorch (Python)
4. XGBoost and LightGBM (Python)
5. Caret (R)

You'll need various machine learning libraries, including: scikit-learn for building and evaluating machine learning models-TensorFlow or PyTorch for deep learning, if needed. - XGBoost, LightGBM, or CatBoost for gradient boosting models.

5. Data Visualization Tools: -

1. Matplotlib
2. Seaborn
3. Plotly
4. Tableau
5. Power BI

Tools like Matplotlib, Seaborn, or Plotly are essential for data exploration and visualization.

6. Data Preprocessing Tools: - Libraries like pandas help with data cleaning, manipulation, and preprocessing.

DESIGN THINKING AND PRESENT IN FORM OF DOCUMENT

1.Empathize:

- Understand the needs and pain points of stock market participants.
- Conducted surveys, interviews, and focus groups with investors, traders, and financial analysts to gather insights.
- Identified user frustrations, such as the need for accessible and understandable predictions.

2.Define:

- Revised the problem statement to focus on creating user-friendly stock price prediction tools that provide clear insights.
- Clearly articulate the problem statement, such as "How might we predict stock prices more accurately and transparently using machine learning?"
- Identify the key goals and success criteria for the project, such as increasing prediction accuracy, reducing bias, or improving user trust in the valuation process.

3.Ideate:

- Brainstorm creative solutions and data sources that can enhance the accuracy and transparency of house price predictions.
- Encourage interdisciplinary collaboration to generate a wide range of ideas, including the use of alternative data, new algorithms, or improved visualization techniques.

4.Prototype:

- Create mockups of redesigned stock price prediction tools.
- Developed mockups of user interfaces with improved data visualization.
- Experimented with different machine learning models for prediction improvement.

5.Test:

- Gather feedback from users on the prototype tools.
- Conducted usability testing sessions with investors and traders to gather their feedback on the redesigned tools.
- Collected suggestions for further improvements.

6.Implement:

- Develop a production-ready machine learning solution for predicting stock prices, integrating the best-performing algorithms and data sources.
- Implement transparency measures, such as model interpretability tools, to ensure users understand how predictions are generated.

7.Evaluate:

- Continuously monitor the performance of the machine learning model after implementation to ensure it remains accurate and relevant in a changing real stock market.
- Gather feedback and insights from users to identify areas for improvement.

8.Iterate:

- Apply an iterative approach to refine the machine learning model based on ongoing feedback and changing user needs.
- Continuously seek ways to enhance prediction accuracy, transparency, and user satisfaction.

9.Scale and Deploy:

- Once the machine learning model has been optimized and validated, deploy it at scale to serve a broader audience, such as investors, and stock market owners.

10.Educate and Train:

- Provide training and educational resources to help users understand how the machine learning model works, what factors it considers, and its limitations.

DESIGN INTO INNOVATION

1. Data Collection:

Data collection is a crucial step in the process of stock price prediction. To create accurate and reliable predictive models, you need to gather historical and real-time data related to the stocks you want to predict.

Data Sources:

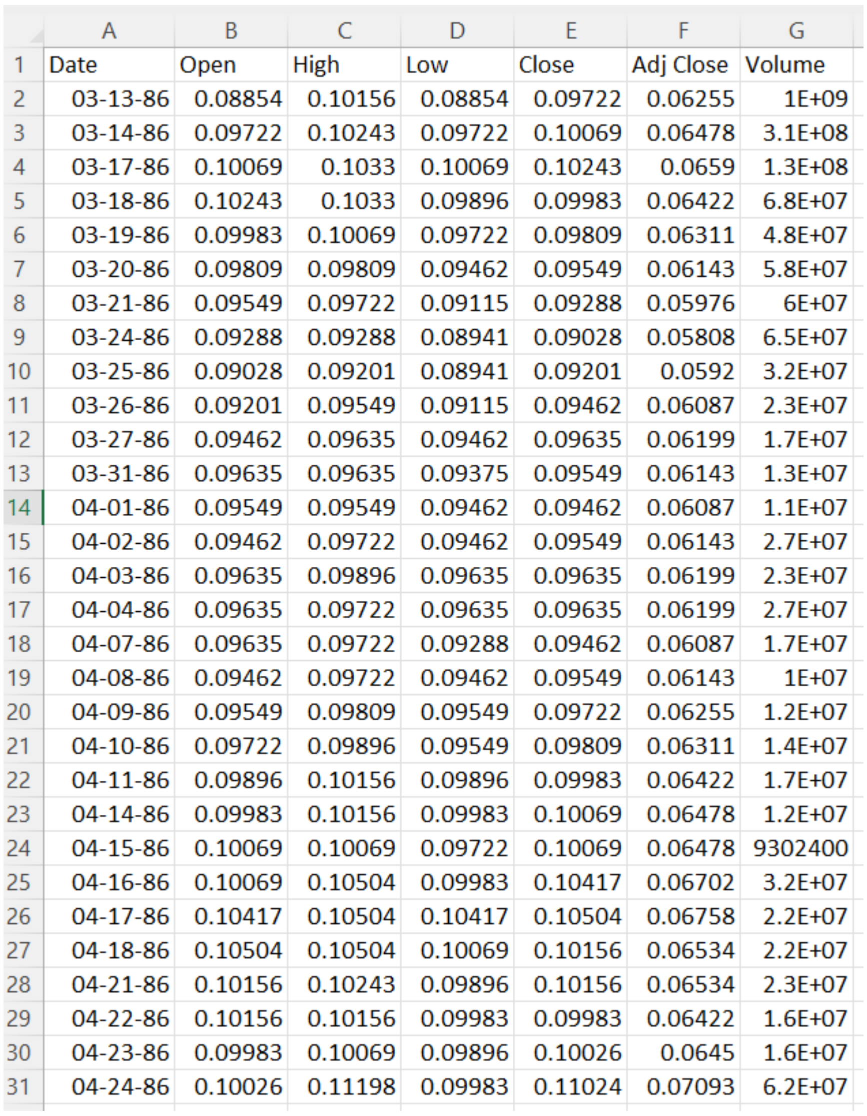
Collect historical price data, including open, high, low, and closing prices, as well as trading volume. You can obtain this data from financial databases, stock exchanges, or financial news websites.

Given data set:

Dataset Link:

https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset_

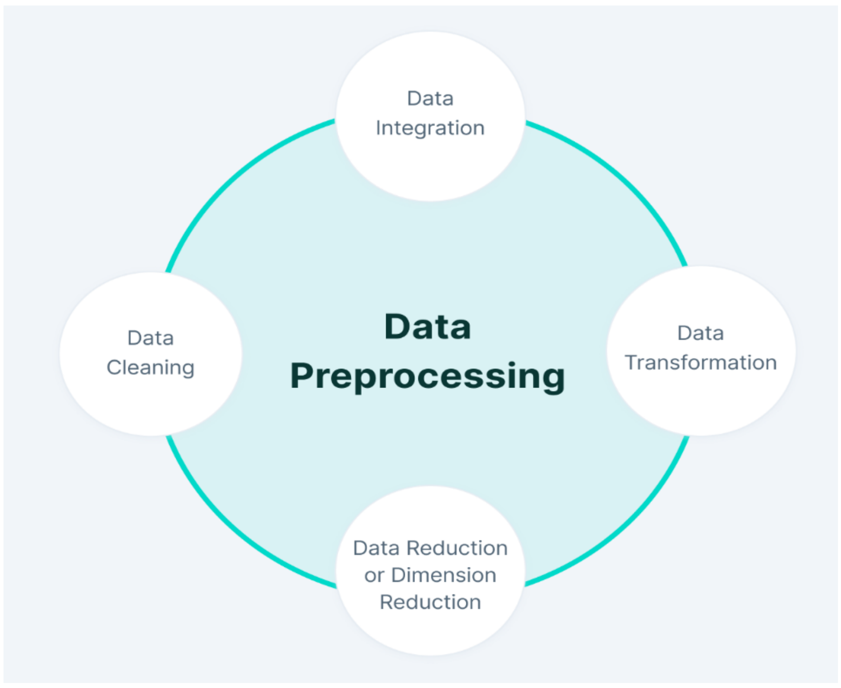
First Step of the project is to collect the data from the given website.



	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Adj Close	Volume
2	03-13-86	0.08854	0.10156	0.08854	0.09722	0.06255	1E+09
3	03-14-86	0.09722	0.10243	0.09722	0.10069	0.06478	3.1E+08
4	03-17-86	0.10069	0.1033	0.10069	0.10243	0.0659	1.3E+08
5	03-18-86	0.10243	0.1033	0.09896	0.09983	0.06422	6.8E+07
6	03-19-86	0.09983	0.10069	0.09722	0.09809	0.06311	4.8E+07
7	03-20-86	0.09809	0.09809	0.09462	0.09549	0.06143	5.8E+07
8	03-21-86	0.09549	0.09722	0.09115	0.09288	0.05976	6E+07
9	03-24-86	0.09288	0.09288	0.08941	0.09028	0.05808	6.5E+07
10	03-25-86	0.09028	0.09201	0.08941	0.09201	0.0592	3.2E+07
11	03-26-86	0.09201	0.09549	0.09115	0.09462	0.06087	2.3E+07
12	03-27-86	0.09462	0.09635	0.09462	0.09635	0.06199	1.7E+07
13	03-31-86	0.09635	0.09635	0.09375	0.09549	0.06143	1.3E+07
14	04-01-86	0.09549	0.09549	0.09462	0.09462	0.06087	1.1E+07
15	04-02-86	0.09462	0.09722	0.09462	0.09549	0.06143	2.7E+07
16	04-03-86	0.09635	0.09896	0.09635	0.09635	0.06199	2.3E+07
17	04-04-86	0.09635	0.09722	0.09635	0.09635	0.06199	2.7E+07
18	04-07-86	0.09635	0.09722	0.09288	0.09462	0.06087	1.7E+07
19	04-08-86	0.09462	0.09722	0.09462	0.09549	0.06143	1E+07
20	04-09-86	0.09549	0.09809	0.09549	0.09722	0.06255	1.2E+07
21	04-10-86	0.09722	0.09896	0.09549	0.09809	0.06311	1.4E+07
22	04-11-86	0.09896	0.10156	0.09896	0.09983	0.06422	1.7E+07
23	04-14-86	0.09983	0.10156	0.09983	0.10069	0.06478	1.2E+07
24	04-15-86	0.10069	0.10069	0.09722	0.10069	0.06478	9302400
25	04-16-86	0.10069	0.10504	0.09983	0.10417	0.06702	3.2E+07
26	04-17-86	0.10417	0.10504	0.10417	0.10504	0.06758	2.2E+07
27	04-18-86	0.10504	0.10504	0.10069	0.10156	0.06534	2.2E+07
28	04-21-86	0.10156	0.10243	0.09896	0.10156	0.06534	2.3E+07
29	04-22-86	0.10156	0.10156	0.09983	0.09983	0.06422	1.6E+07
30	04-23-86	0.09983	0.10069	0.09896	0.10026	0.0645	1.6E+07
31	04-24-86	0.10026	0.11198	0.09983	0.11024	0.07093	6.2E+07

2.Data Preprocessing:

- Data preprocessing is a crucial step in the data analysis and machine learning pipeline.
- It involves cleaning, transforming, and organizing raw data to make it suitable for modeling.
- Proper data preprocessing is essential for ensuring that the data is of high quality, consistent, and relevant for the analysis or prediction task.
- Clean the data by handling missing values, outliers, and encoding categorical variables. Standardize or normalize numerical features as necessary.



Program code:

```
 #import libraries

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler
```

Load the Dataset:

Load your dataset into a Pandas DataFrame. You can typically find stock price datasets in CSV format, but you can adapt this code to other formats as needed.

Program code:

```
# Load the dataset into a pandas DataFrame

df = pd. read_csv('MSFT1.CSV')

pd. read()
```

3. Exploratory Data Analysis (EDA):

Performing Exploratory Data Analysis (EDA) is a crucial step in understanding the characteristics and patterns present in our dataset. In the context of stock price prediction, EDA helps in identifying trends, detecting outliers, and gaining insights that can inform the choice of features and modeling strategies.

Program code:

```
# Check for missing values

print(df.isnull().sum())

# Explore statistics

print(df.describe())

# Visualize the data (e.g., histograms, scatter plots, etc.)
```

4. Feature Engineering:

Feature engineering is the process of creating new features or modifying existing ones in a dataset to improve the performance of machine learning models.

It involves transforming raw data into a format that is better suited to reveal patterns and relationships, making it easier for models to learn and make predictions.

In the context of stock price prediction, feature engineering plays a crucial role in enhancing the information available to the model and capturing relevant patterns in financial time series data.

5. Split the Data:

Split the dataset into training and testing sets. This helps you evaluate the model's performance later.

```
X = df. drop ('price', axis=1) # Features

y = df['price'] # Target variable

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

random_state=42)
```

6. Feature Scaling:

Apply feature scaling to normalize your data, ensuring that all features have similar scales. Standardization (scaling to mean=0 and std=1) is a common choice.

Program code:

```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

7. Feature Engineering:

Create new features or transform existing ones to extract more valuable information.

8. Model Selection:

Choose the appropriate machine learning model for the task. Common models for regression problems like house price prediction include Linear Regression, Decision Trees, Random Forest, Gradient Boosting, and Neural Networks.

9. Training:

Split the dataset into training and testing sets to evaluate the model's performance. Consider techniques like cross-validation to prevent overfitting.

10. Hyperparameter Tuning:

Optimize the model's hyperparameters to improve its predictive accuracy. Techniques like grid search or random search can help with this.

11. Evaluation Metrics:

Select appropriate evaluation metrics for regression tasks, such as Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE). Choose the metric that aligns with the specific objectives of your project.

12. Regularization:

Apply regularization techniques like L1 (Lasso) or L2 (Ridge)regularization to prevent overfitting.

13. Feature Selection:

Use techniques like feature importance scores or recursive feature elimination to identify the most relevant features for the prediction.

14. Interpretability:

Ensure that the model's predictions are interpretable and explainable. This is especially important for real estate applications where stakeholders want to understand the factors affecting predictions.

15. Deployment:

Develop a user-friendly interface or API for end-users to input property details and receive price predictions.

16. Continuous Improvement:

Implement a feedback loop for continuous model improvement based on user feedback and new data.

17. Ethical Considerations:

Be mindful of potential biases in the data and model. Ensure fairness and transparency in your predictions.

18. Monitoring and Maintenance:

Regularly monitor the model's performance in the real world and update it as needed.

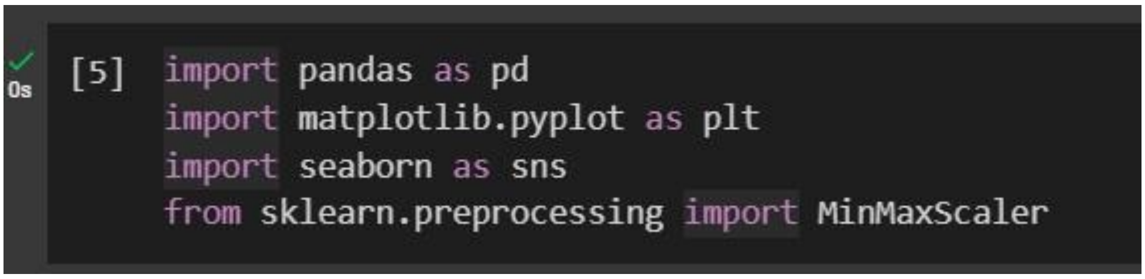
19. Innovation:

Consider innovative approaches such as using satellite imagery root data for real-time property condition monitoring or integrating natural language processing for textual property descriptions.

Stock price prediction

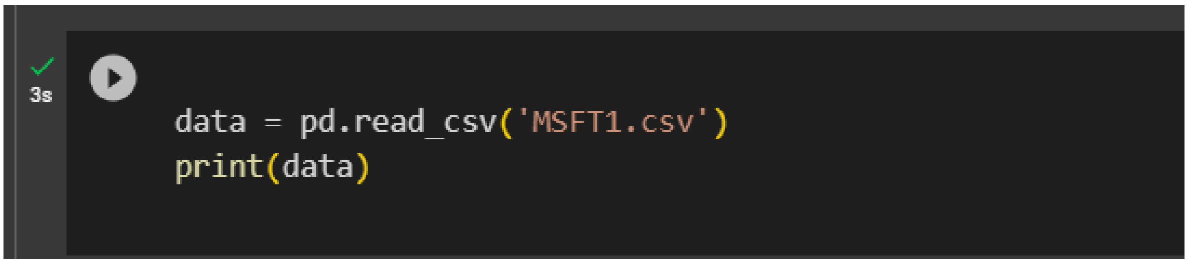
BUILD LOADING AND PREPROCESSING THE DATASET

Import libraries

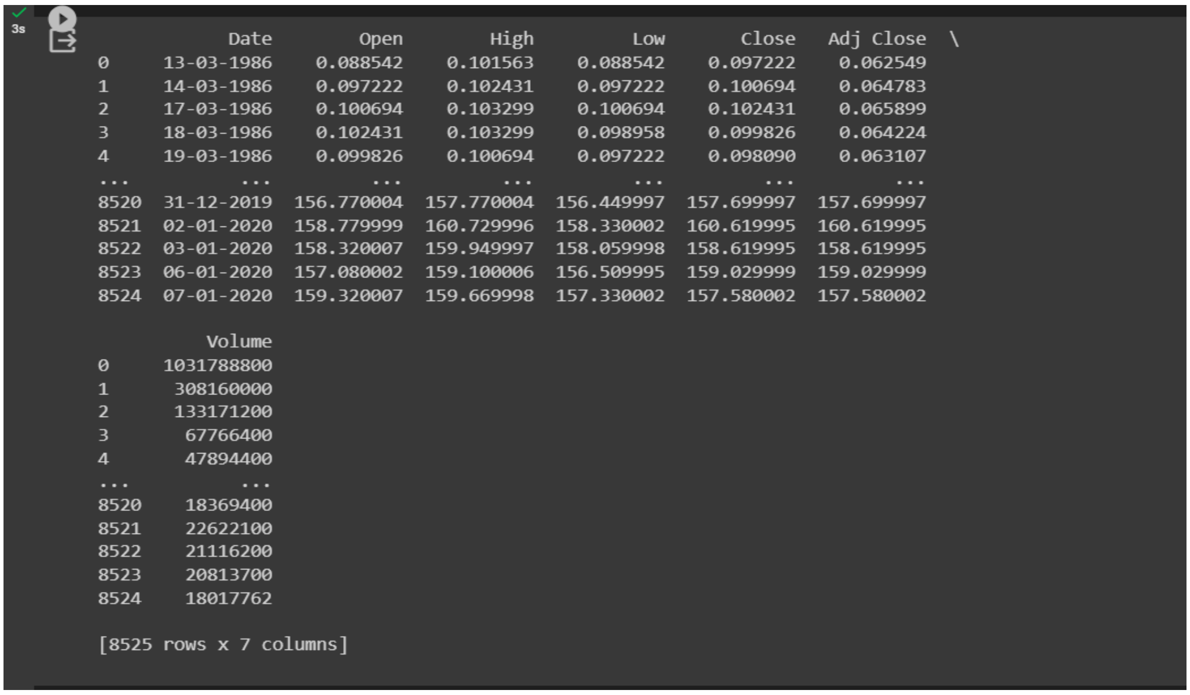


```
[5] import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.preprocessing import MinMaxScaler
```

Loading the dataset

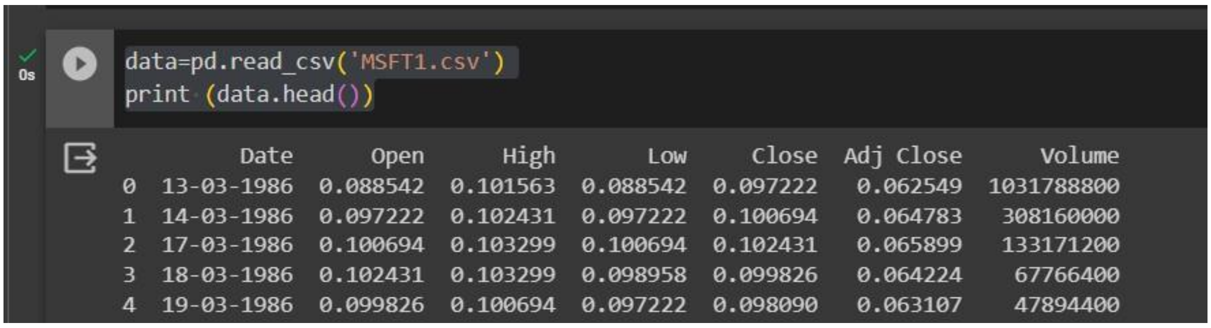


```
data = pd.read_csv('MSFT1.csv')
print(data)
```



```
            Date        Open        High         Low       Close   Adj Close  \
0     13-03-1986    0.088542    0.101563    0.088542    0.097222    0.062549
1     14-03-1986    0.097222    0.102431    0.097222    0.100694    0.064783
2     17-03-1986    0.100694    0.103299    0.100694    0.102431    0.065899
3     18-03-1986    0.102431    0.103299    0.098958    0.099826    0.064224
4     19-03-1986    0.099826    0.100694    0.097222    0.098090    0.063107
...          ...         ...         ...         ...         ...         ...
8520  31-12-2019  156.770004  157.770004  156.449997  157.699997  157.699997
8521  02-01-2020  158.779999  160.729996  158.330002  160.619995  160.619995
8522  03-01-2020  158.320007  159.949997  158.059998  158.619995  158.619995
8523  06-01-2020  157.080002  159.100006  156.509995  159.029999  159.029999
8524  07-01-2020  159.320007  159.669998  157.330002  157.580002  157.580002

          Volume
0     1031788800
1      308160000
2      133171200
3       67766400
4       47894400
...          ...
8520    18369400
8521    22622100
8522    21116200
8523    20813700
8524    18017762

[8525 rows x 7 columns]
```

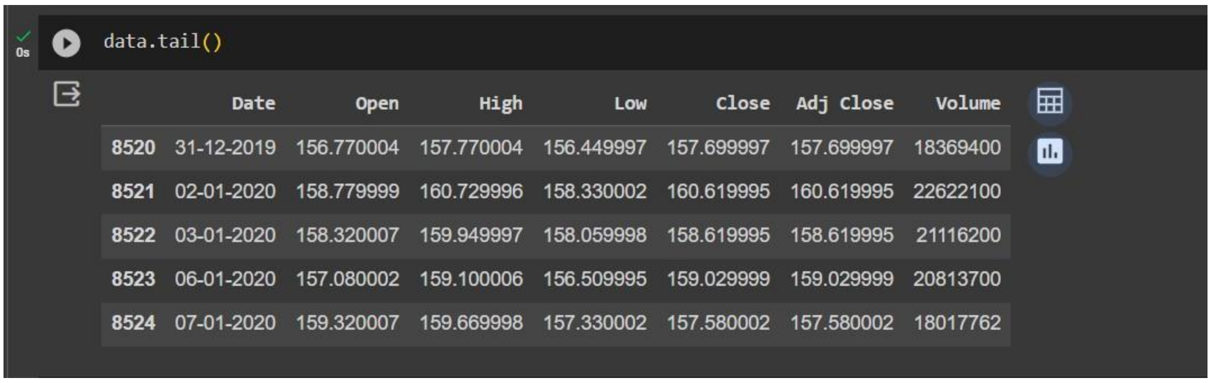
Print first five rows



```
data=pd.read_csv('MSFT1.csv')
print (data.head())
```

```
         Date      Open      High       Low     Close  Adj Close      Volume
0  13-03-1986  0.088542  0.101563  0.088542  0.097222   0.062549  1031788800
1  14-03-1986  0.097222  0.102431  0.097222  0.100694   0.064783   308160000
2  17-03-1986  0.100694  0.103299  0.100694  0.102431   0.065899   133171200
3  18-03-1986  0.102431  0.103299  0.098958  0.099826   0.064224    67766400
4  19-03-1986  0.099826  0.100694  0.097222  0.098090   0.063107    47894400
```

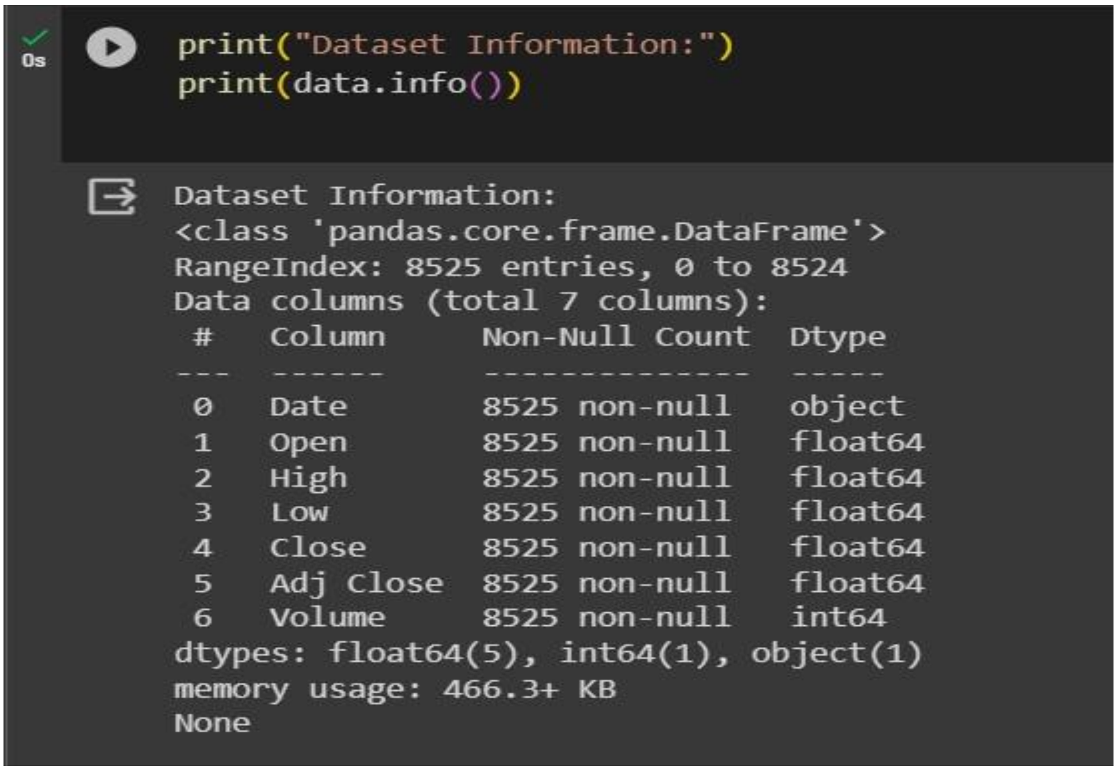
Print last five rows



```
data.tail()
```

```
            Date        Open        High         Low       Close   Adj Close     Volume
8520  31-12-2019  156.770004  157.770004  156.449997  157.699997  157.699997   18369400
8521  02-01-2020  158.779999  160.729996  158.330002  160.619995  160.619995   22622100
8522  03-01-2020  158.320007  159.949997  158.059998  158.619995  158.619995   21116200
8523  06-01-2020  157.080002  159.100006  156.509995  159.029999  159.029999   20813700
8524  07-01-2020  159.320007  159.669998  157.330002  157.580002  157.580002   18017762
```

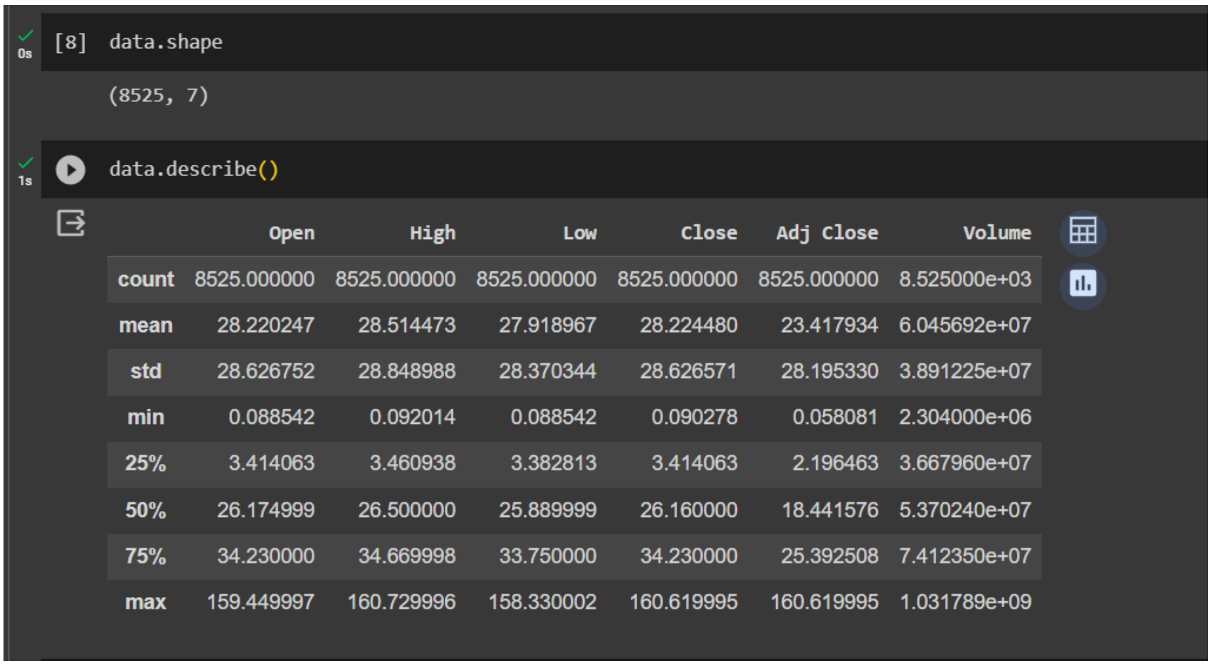
Print Dataset Information



```
print("Dataset Information:")
print(data.info())
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8525 entries, 0 to 8524
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       8525 non-null   object
 1   Open       8525 non-null   float64
 2   High       8525 non-null   float64
 3   Low        8525 non-null   float64
 4   Close      8525 non-null   float64
 5   Adj Close  8525 non-null   float64
 6   Volume     8525 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 466.3+ KB
None
```

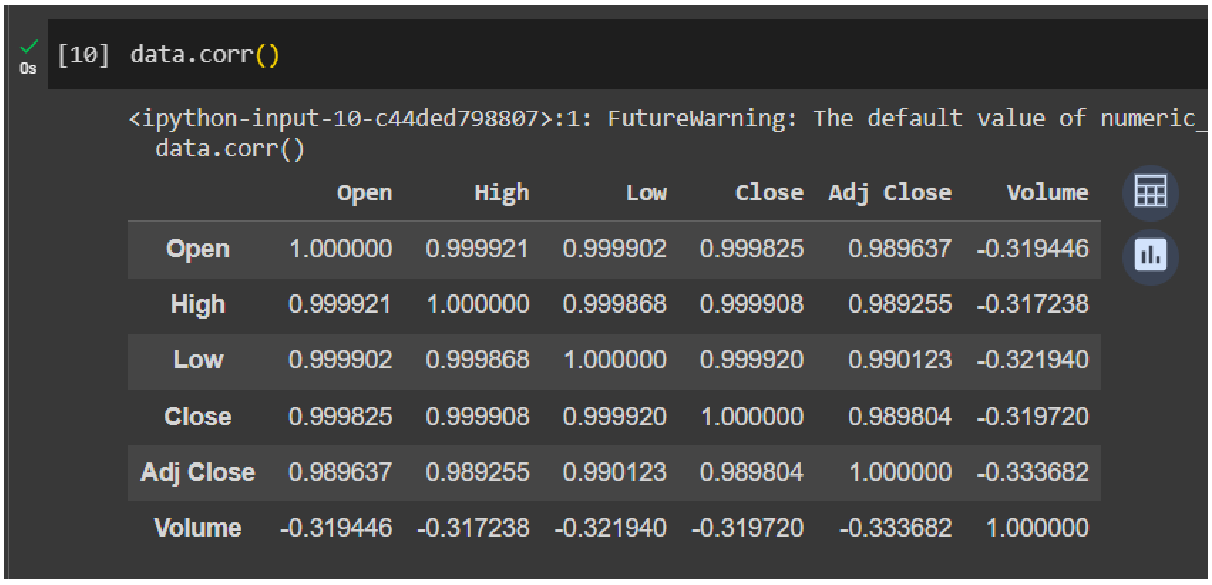
Print data shape and describe about dataset



```
[8]  data.shape

     (8525, 7)
```

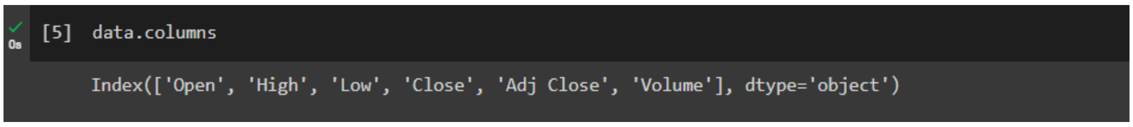
```
data.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	8525.000000	8525.000000	8525.000000	8525.000000	8525.000000	8.525000e+03
mean	28.220247	28.514473	27.918967	28.224480	23.417934	6.045692e+07
std	28.626752	28.848988	28.370344	28.626571	28.195330	3.891225e+07
min	0.088542	0.092014	0.088542	0.090278	0.058081	2.304000e+06
25%	3.414063	3.460938	3.382813	3.414063	2.196463	3.667960e+07
50%	26.174999	26.500000	25.889999	26.160000	18.441576	5.370240e+07
75%	34.230000	34.669998	33.750000	34.230000	25.392508	7.412350e+07
max	159.449997	160.729996	158.330002	160.619995	160.619995	1.031789e+09

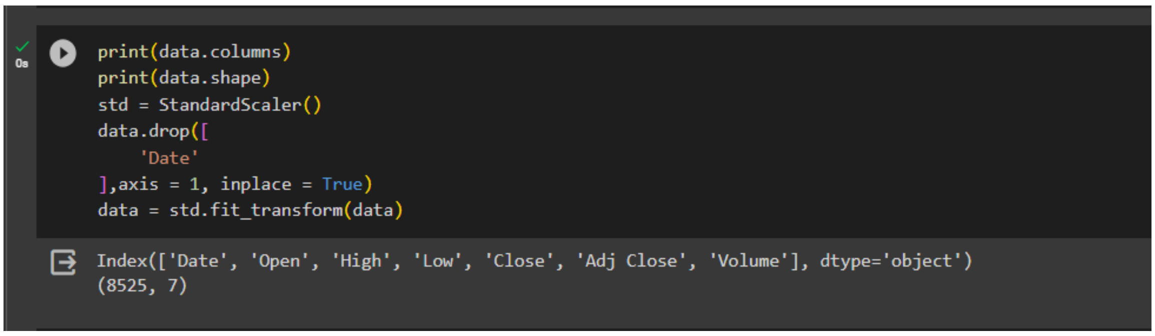


```
[10]  data.corr()

      <ipython-input-10-c44ded798807>:1: FutureWarning: The default value of numeric_
        data.corr()
```

	Open	High	Low	Close	Adj Close	Volume
Open	1.000000	0.999921	0.999902	0.999825	0.989637	-0.319446
High	0.999921	1.000000	0.999868	0.999908	0.989255	-0.317238
Low	0.999902	0.999868	1.000000	0.999920	0.990123	-0.321940
Close	0.999825	0.999908	0.999920	1.000000	0.989804	-0.319720
Adj Close	0.989637	0.989255	0.990123	0.989804	1.000000	-0.333682
Volume	-0.319446	-0.317238	-0.321940	-0.319720	-0.333682	1.000000

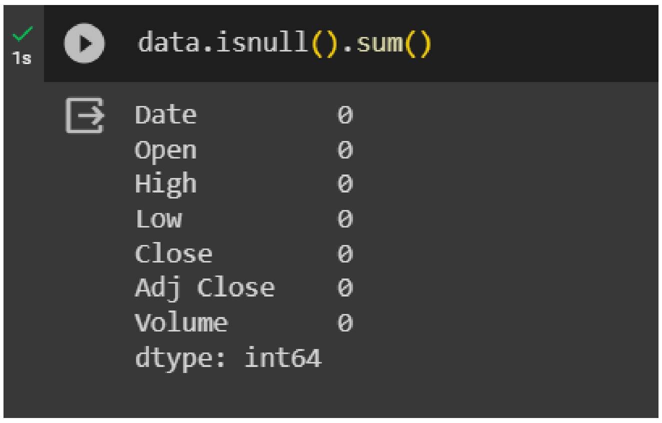


```
[5] data.columns

    Index(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
```



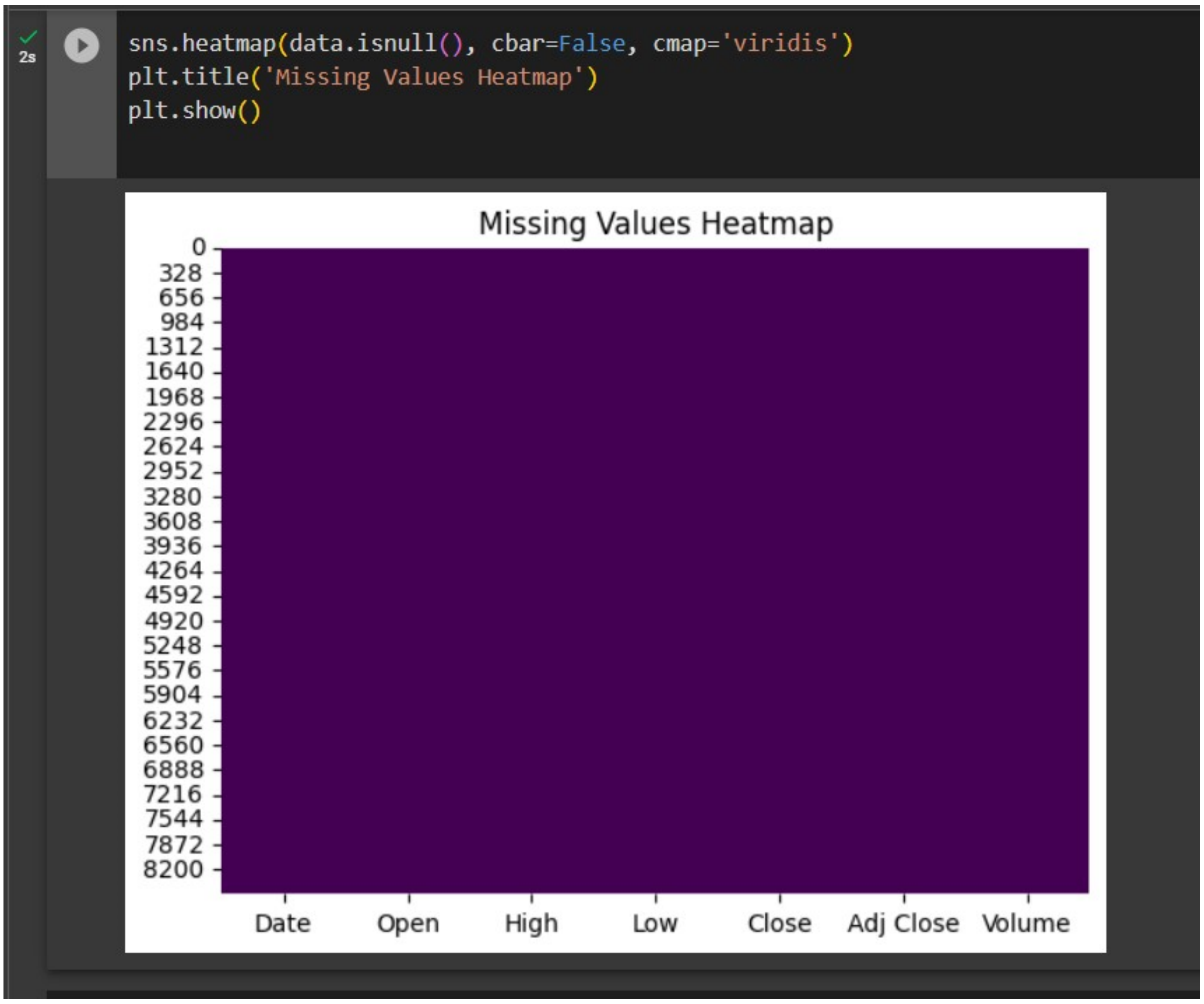
```
print(data.columns)
print(data.shape)
std = StandardScaler()
data.drop([
    'Date'
],axis = 1, inplace = True)
data = std.fit_transform(data)

Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
(8525, 7)
```

Handling missing data



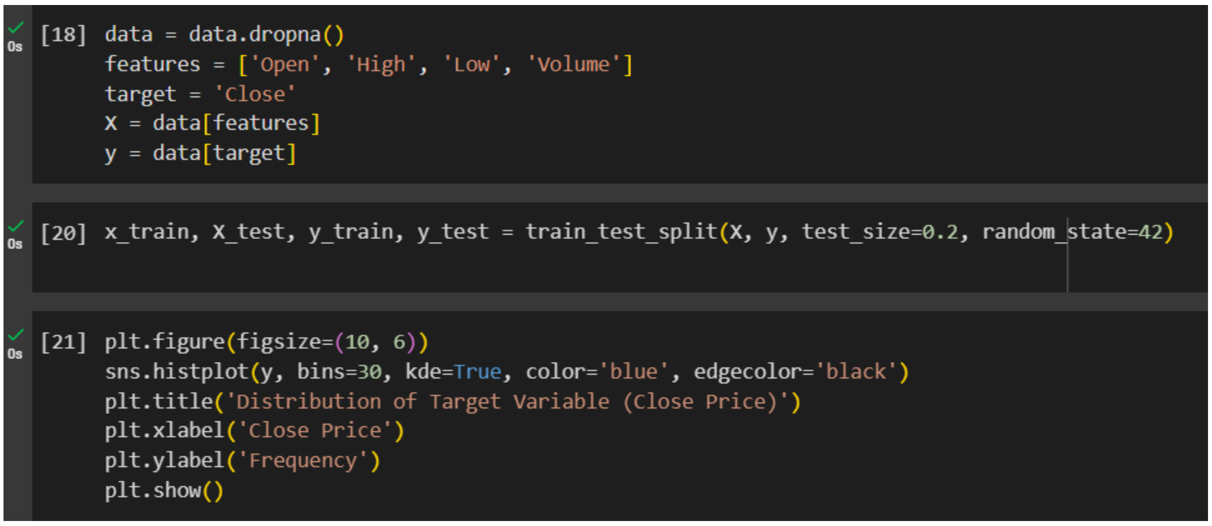
```
data.isnull().sum()

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

Visualize the missing values

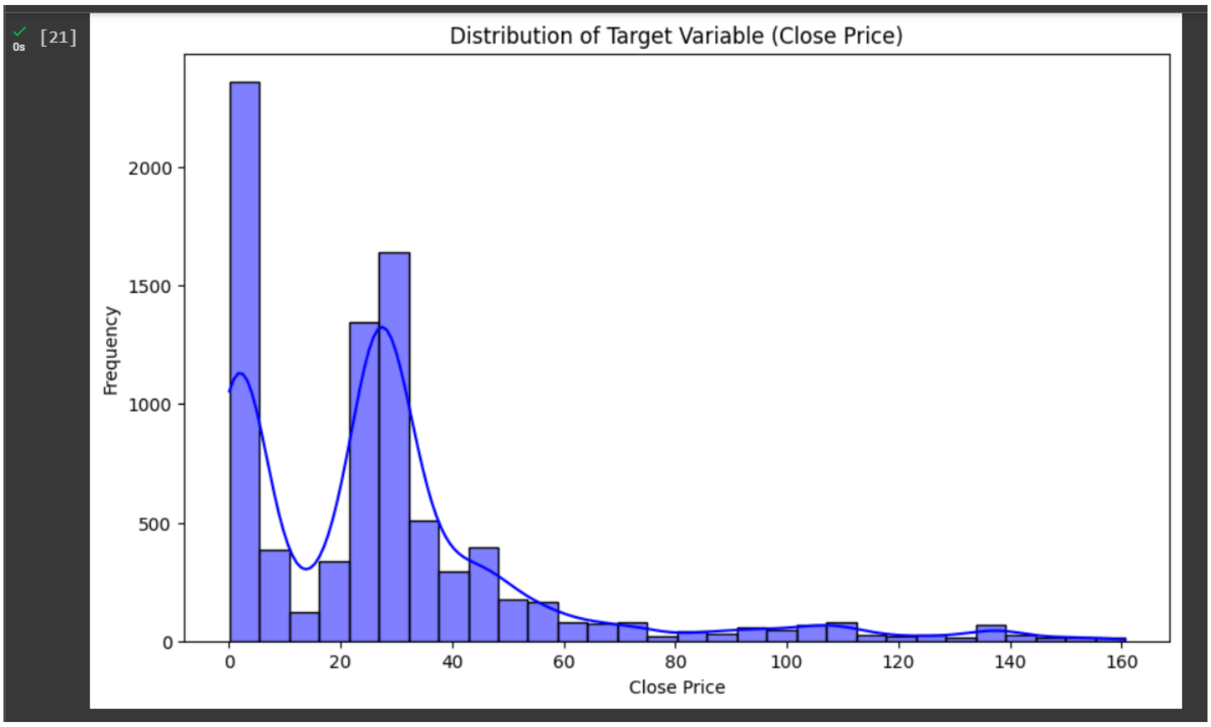


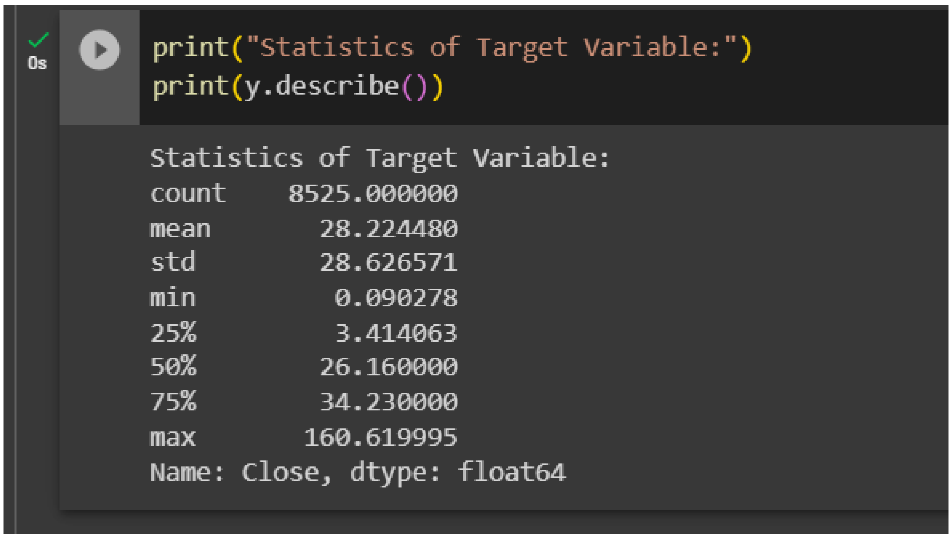
```
sns.heatmap(data.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Values Heatmap')
plt.show()
```

Visualize target variable



```
[18] data = data.dropna()
     features = ['Open', 'High', 'Low', 'Volume']
     target = 'Close'
     X = data[features]
     y = data[target]

[20] x_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

[21] plt.figure(figsize=(10, 6))
     sns.histplot(y, bins=30, kde=True, color='blue', edgecolor='black')
     plt.title('Distribution of Target Variable (Close Price)')
     plt.xlabel('Close Price')
     plt.ylabel('Frequency')
     plt.show()
```

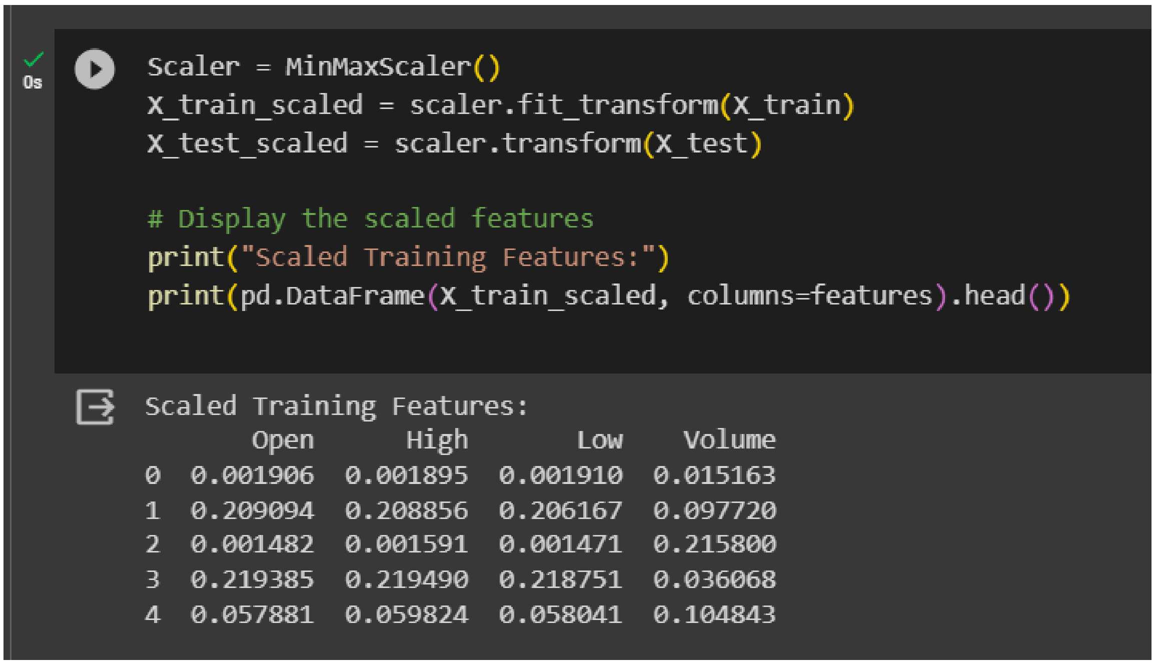




```
print("Statistics of Target Variable:")
print(y.describe())
```

```
Statistics of Target Variable:
count    8525.000000
mean       28.224480
std        28.626571
min         0.090278
25%         3.414063
50%        26.160000
75%        34.230000
max       160.619995
Name: Close, dtype: float64
```

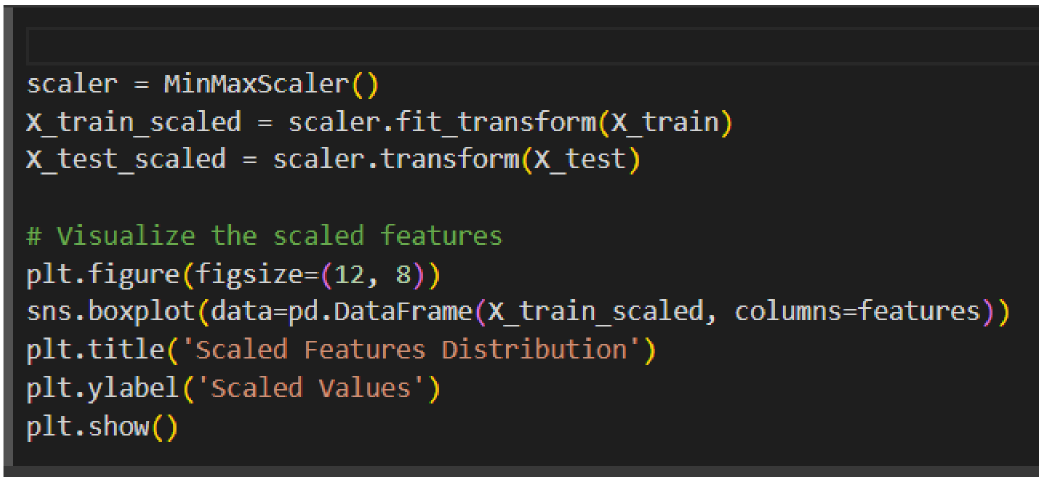
Normalize or scaler features



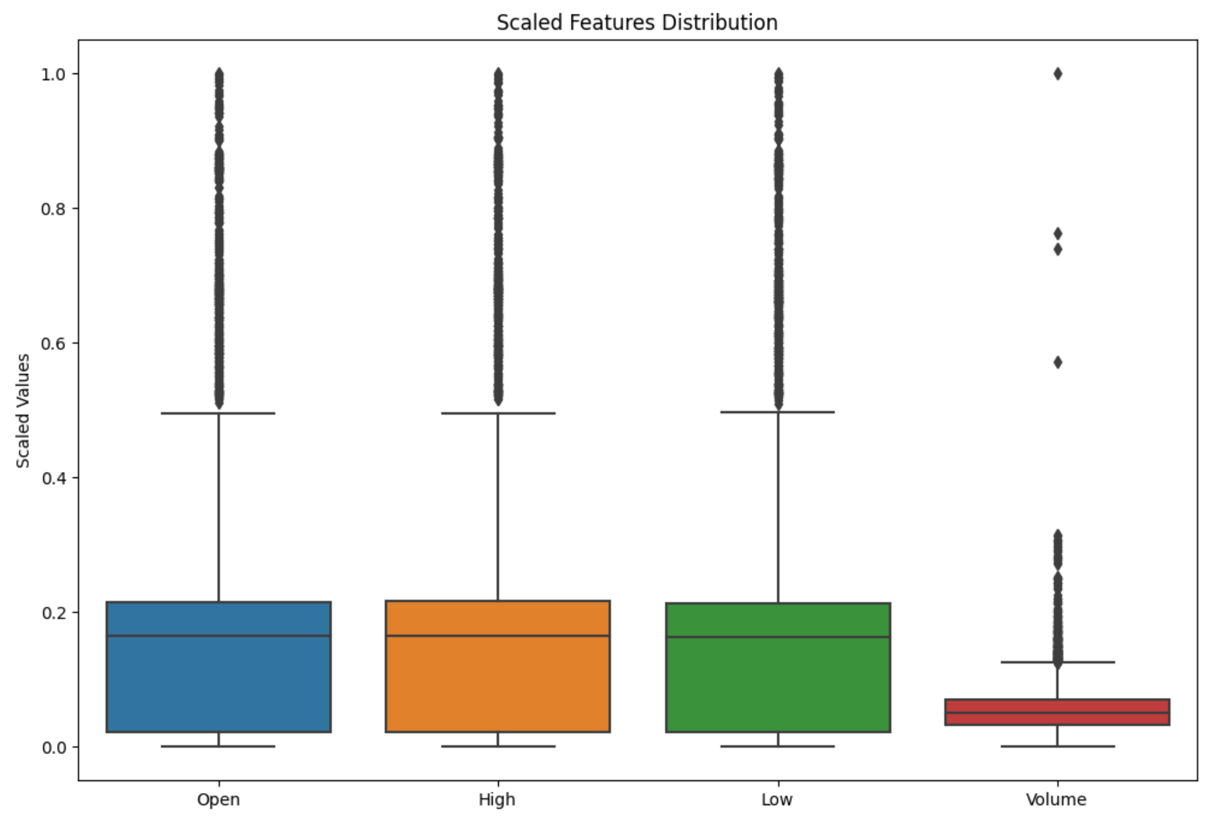
```
Scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Display the scaled features
print("Scaled Training Features:")
print(pd.DataFrame(X_train_scaled, columns=features).head())
```

```
Scaled Training Features:
       Open      High       Low    Volume
0  0.001906  0.001895  0.001910  0.015163
1  0.209094  0.208856  0.206167  0.097720
2  0.001482  0.001591  0.001471  0.215800
3  0.219385  0.219490  0.218751  0.036068
4  0.057881  0.059824  0.058041  0.104843
```

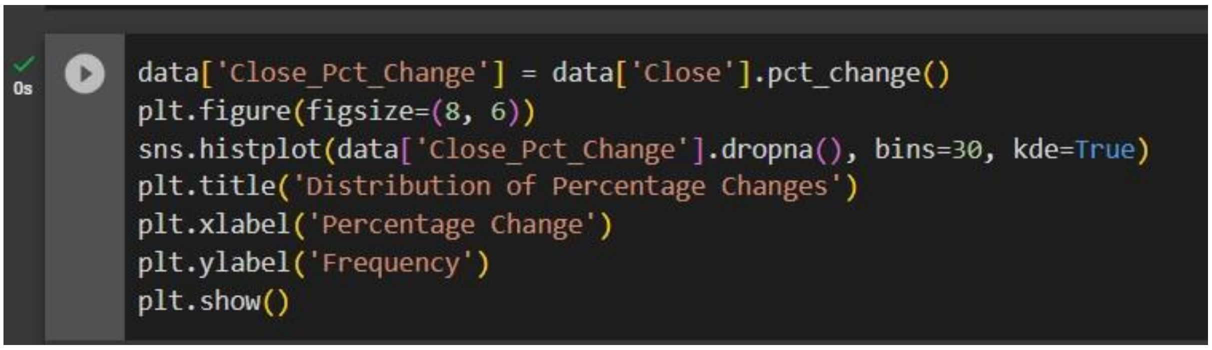
Visualize the scaler features



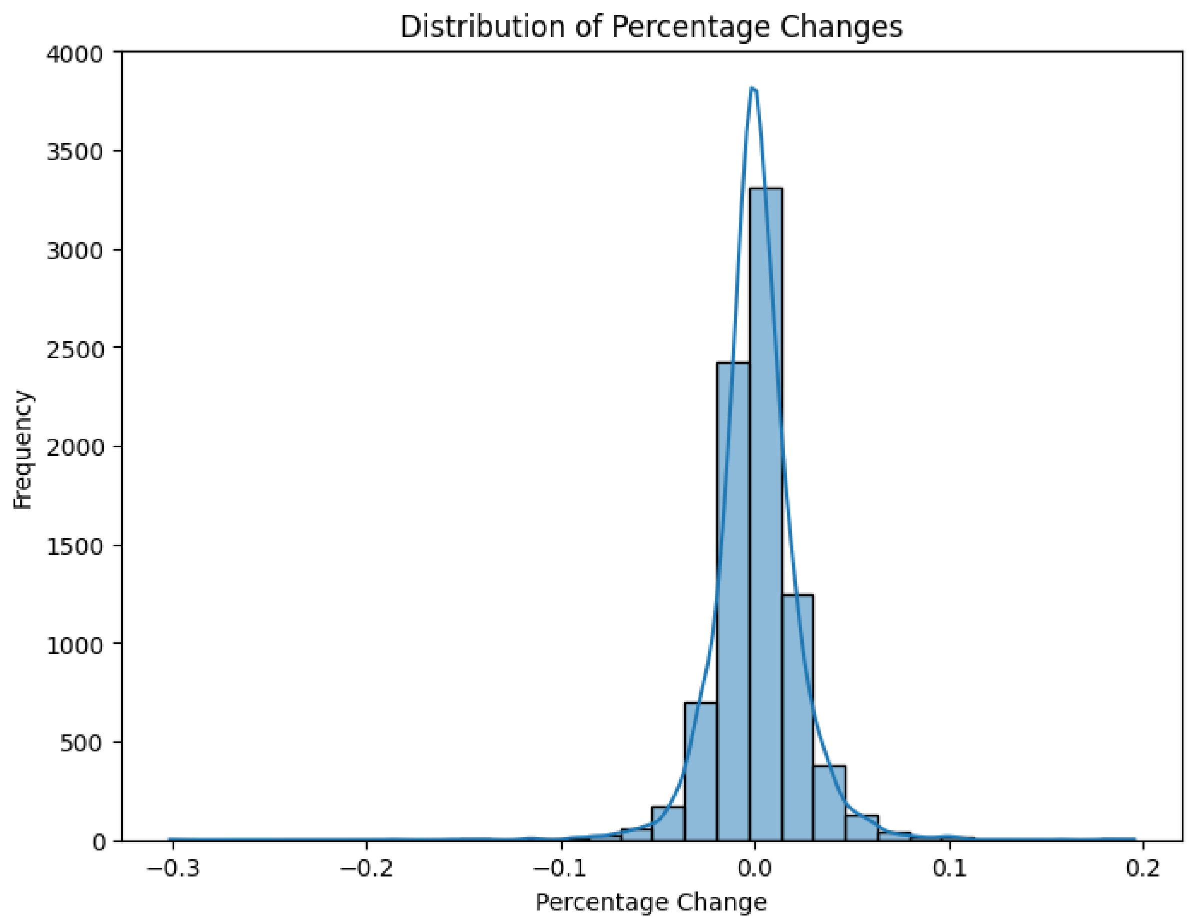
```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Visualize the scaled features
plt.figure(figsize=(12, 8))
sns.boxplot(data=pd.DataFrame(X_train_scaled, columns=features))
plt.title('Scaled Features Distribution')
plt.ylabel('Scaled Values')
plt.show()
```



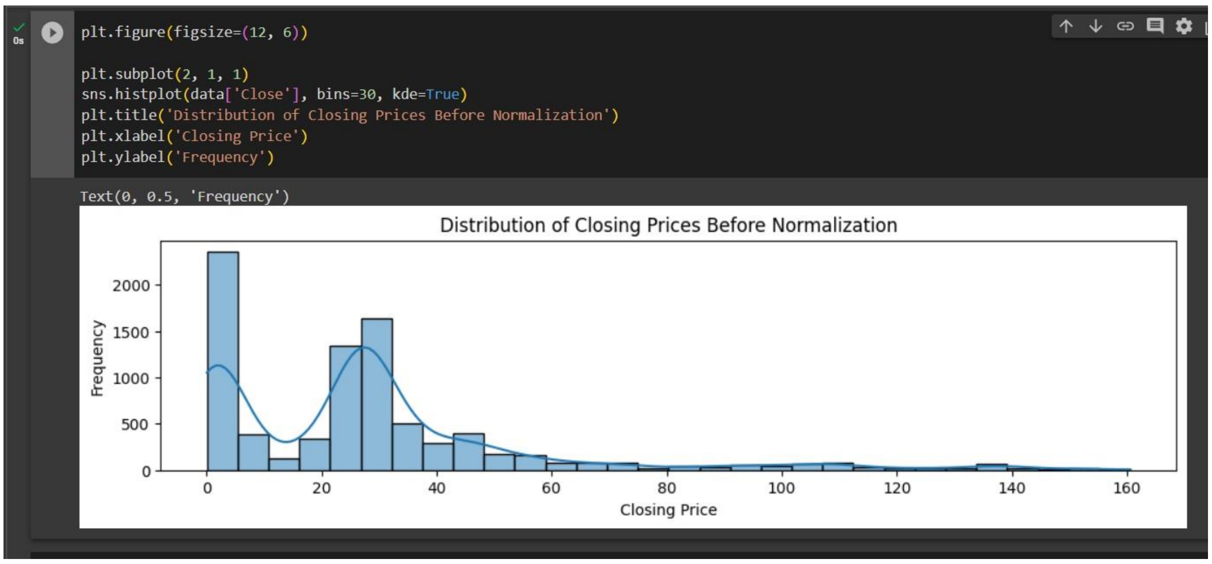
Visualize the distribution of percentage changes



```
data['Close_Pct_Change'] = data['Close'].pct_change()
plt.figure(figsize=(8, 6))
sns.histplot(data['Close_Pct_Change'].dropna(), bins=30, kde=True)
plt.title('Distribution of Percentage Changes')
plt.xlabel('Percentage Change')
plt.ylabel('Frequency')
plt.show()
```

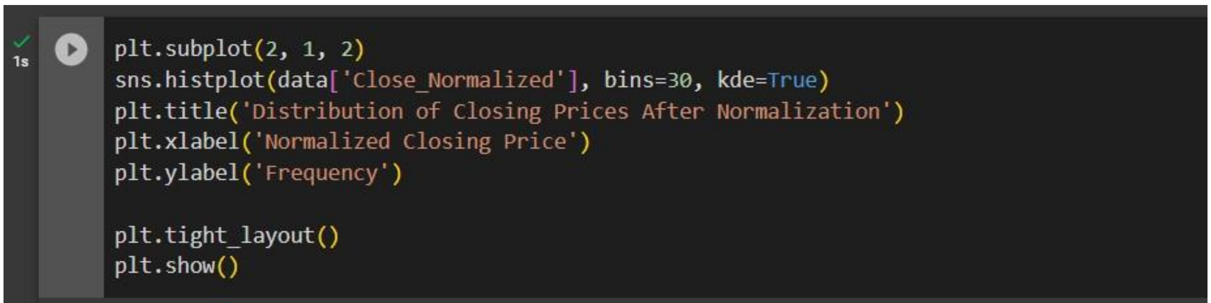


Normalization

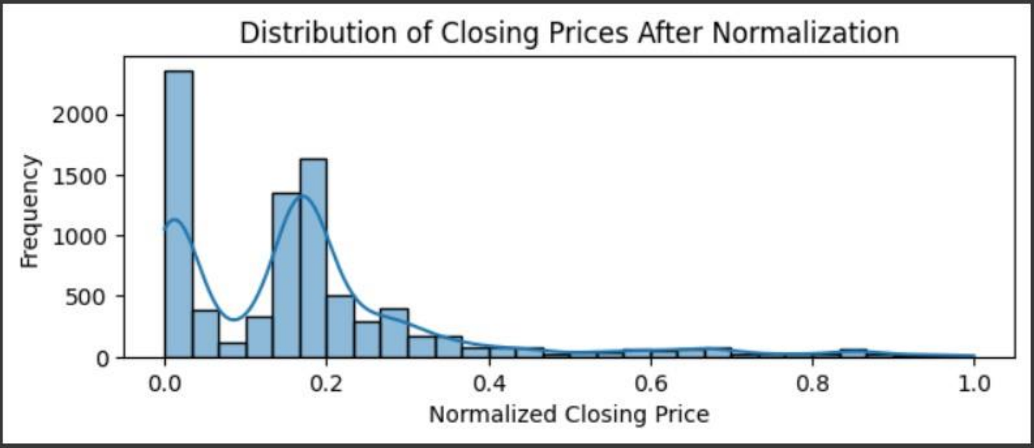


```
plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)
sns.histplot(data['Close'], bins=30, kde=True)
plt.title('Distribution of Closing Prices Before Normalization')
plt.xlabel('Closing Price')
plt.ylabel('Frequency')
```

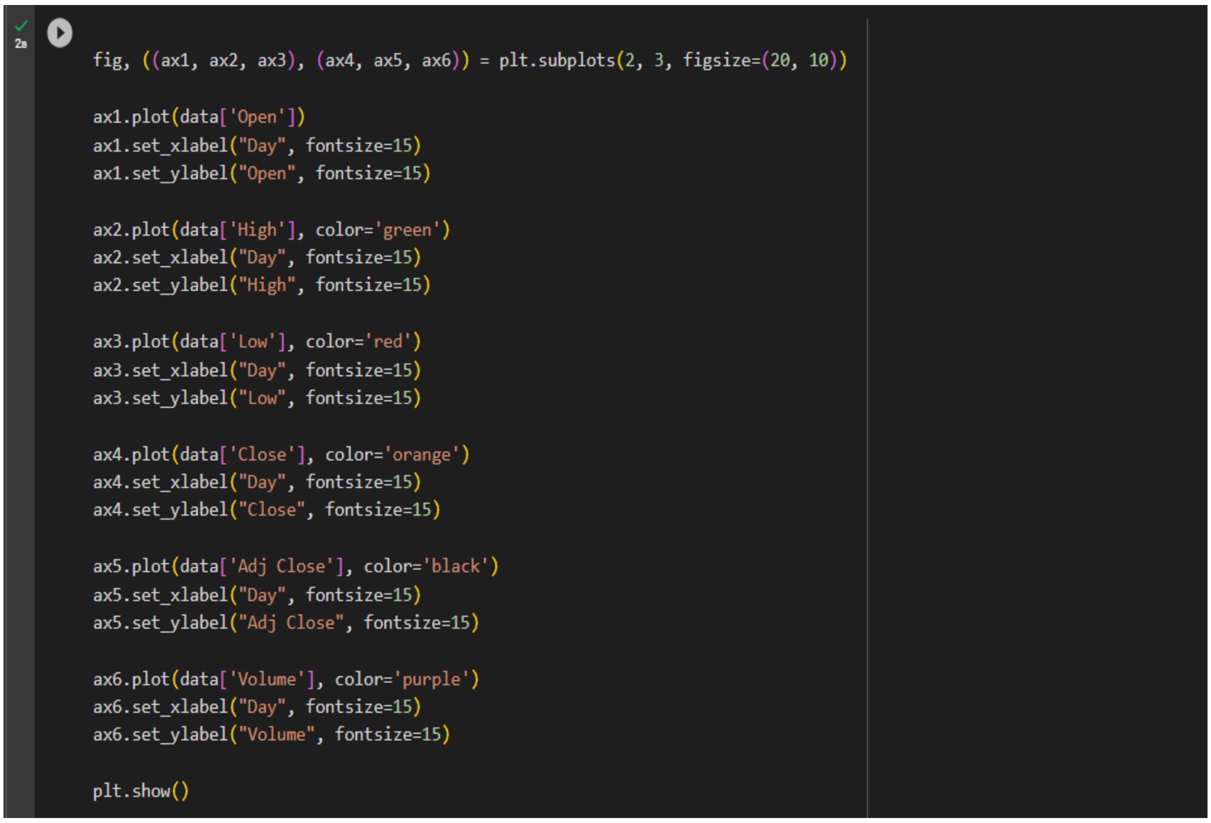
```
Text(0, 0.5, 'Frequency')
```



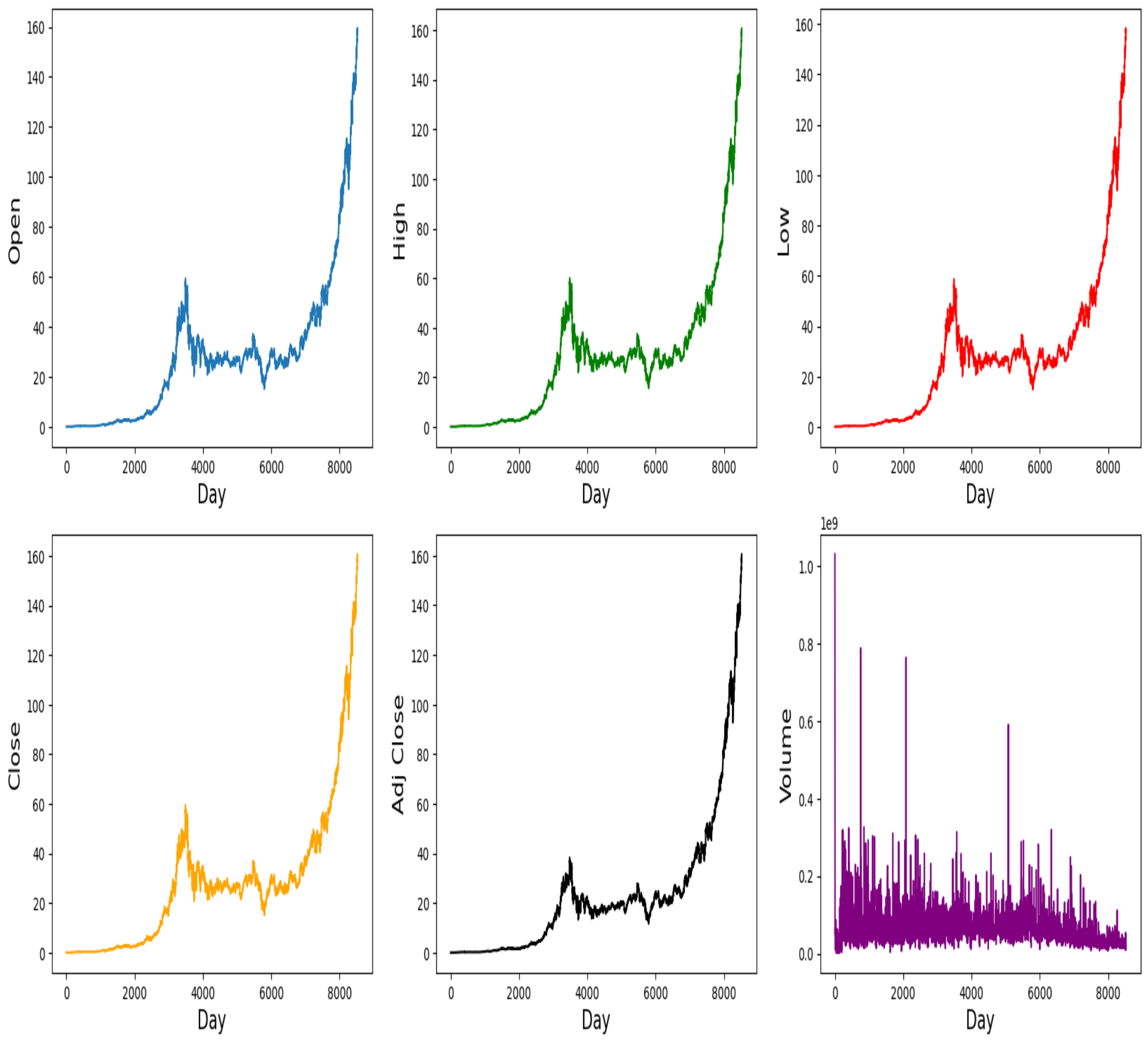
```
plt.subplot(2, 1, 2)
sns.histplot(data['Close_Normalized'], bins=30, kde=True)
plt.title('Distribution of Closing Prices After Normalization')
plt.xlabel('Normalized Closing Price')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```

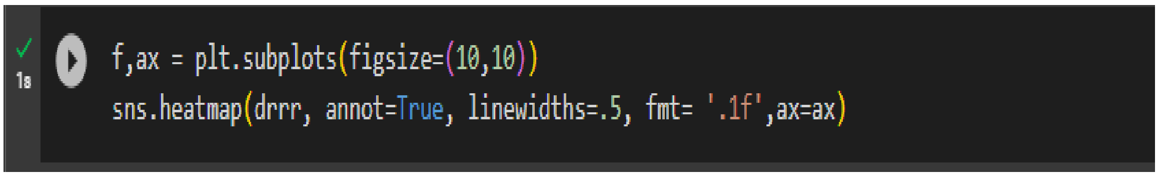


Data Visualization

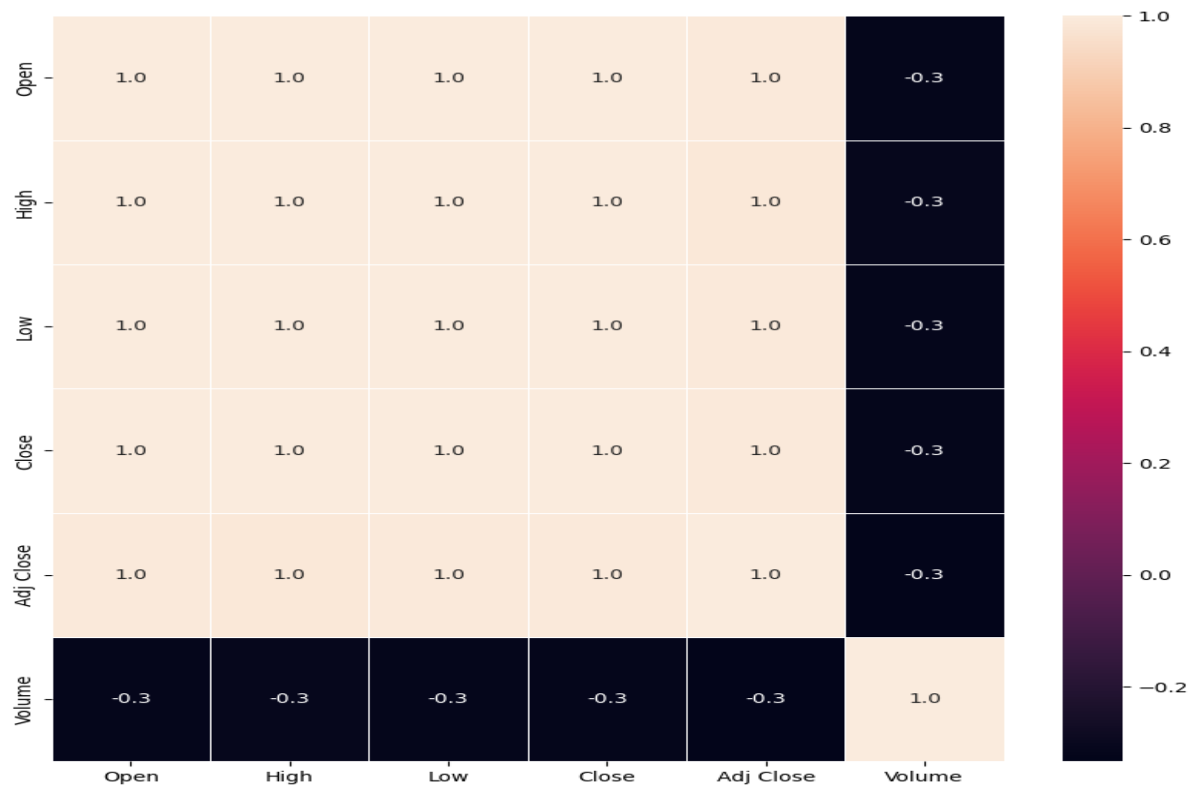


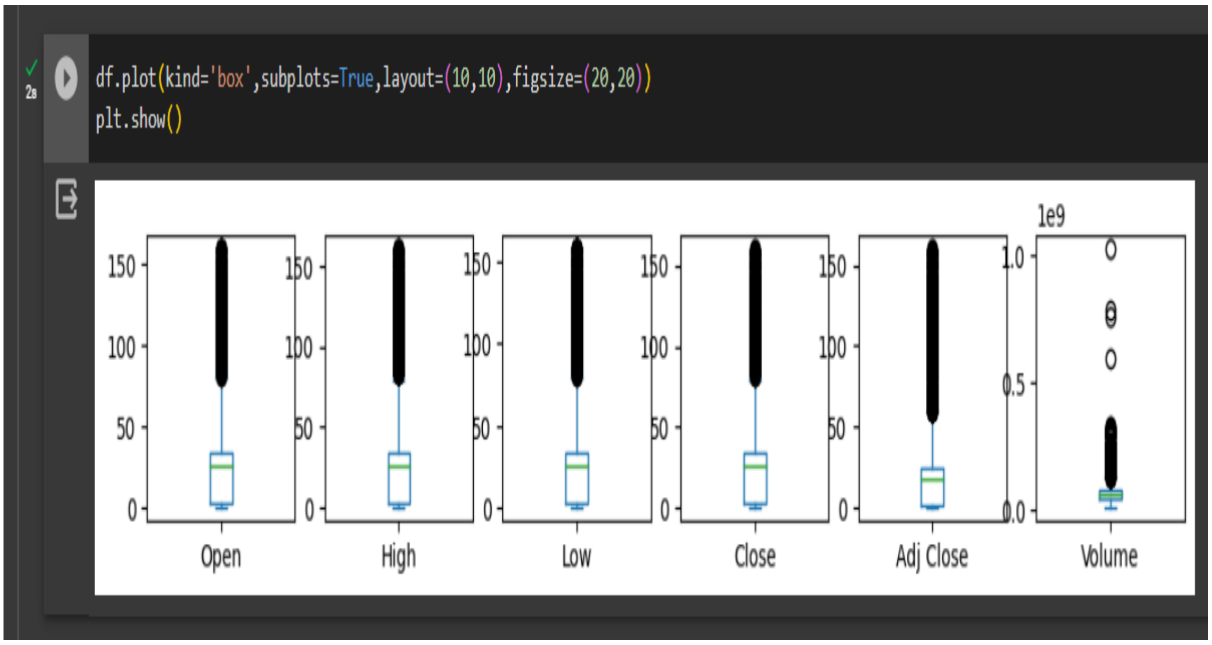
```
fig, ((ax1, ax2, ax3), (ax4, ax5, ax6)) = plt.subplots(2, 3, figsize=(20, 10))

ax1.plot(data['Open'])
ax1.set_xlabel("Day", fontsize=15)
ax1.set_ylabel("Open", fontsize=15)

ax2.plot(data['High'], color='green')
ax2.set_xlabel("Day", fontsize=15)
ax2.set_ylabel("High", fontsize=15)

ax3.plot(data['Low'], color='red')
ax3.set_xlabel("Day", fontsize=15)
ax3.set_ylabel("Low", fontsize=15)

ax4.plot(data['Close'], color='orange')
ax4.set_xlabel("Day", fontsize=15)
ax4.set_ylabel("Close", fontsize=15)

ax5.plot(data['Adj Close'], color='black')
ax5.set_xlabel("Day", fontsize=15)
ax5.set_ylabel("Adj Close", fontsize=15)

ax6.plot(data['Volume'], color='purple')
ax6.set_xlabel("Day", fontsize=15)
ax6.set_ylabel("Volume", fontsize=15)

plt.show()
```



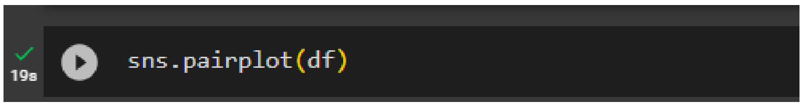


```
f,ax = plt.subplots(figsize=(10,10))
sns.heatmap(drrr, annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```

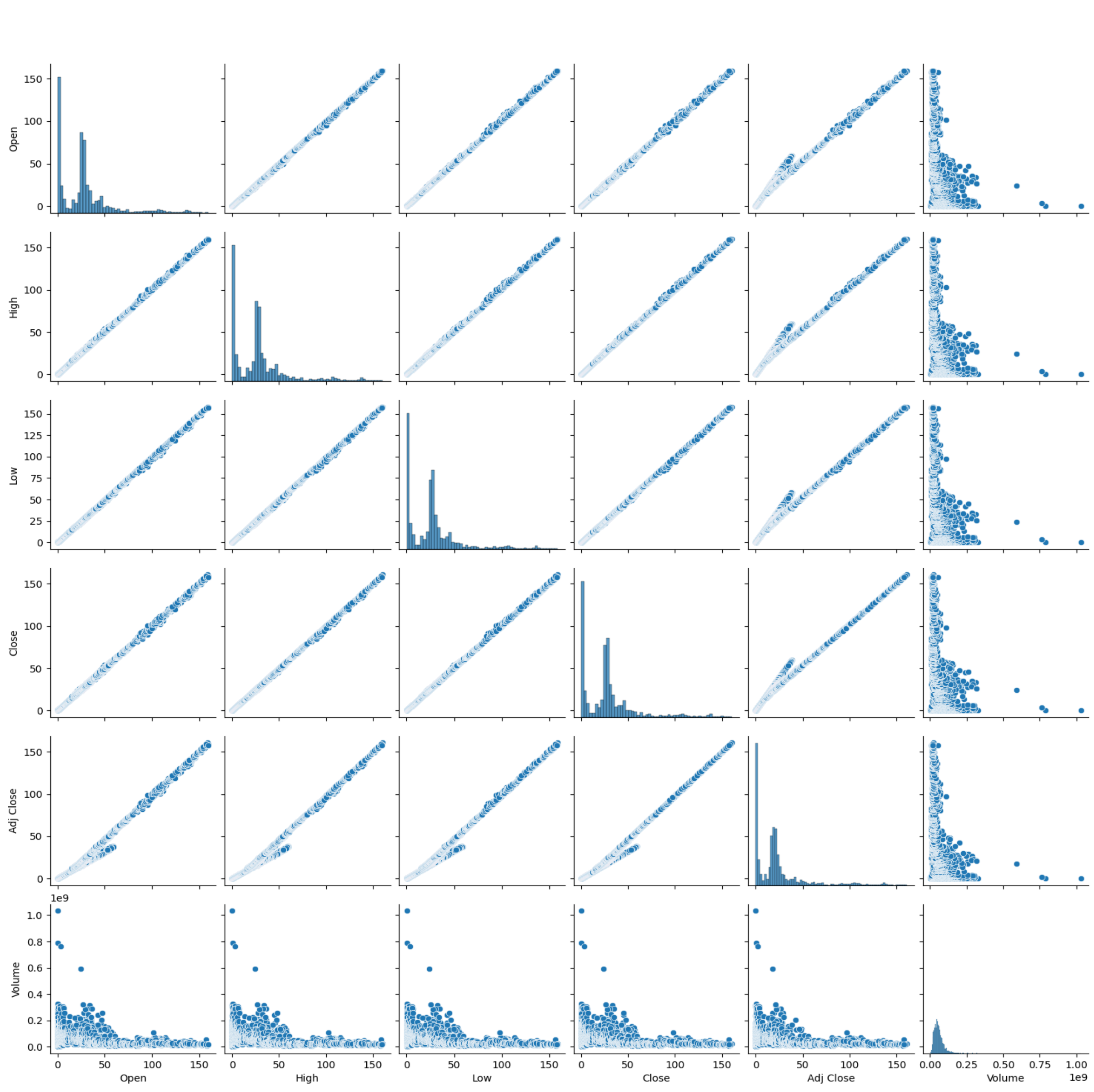


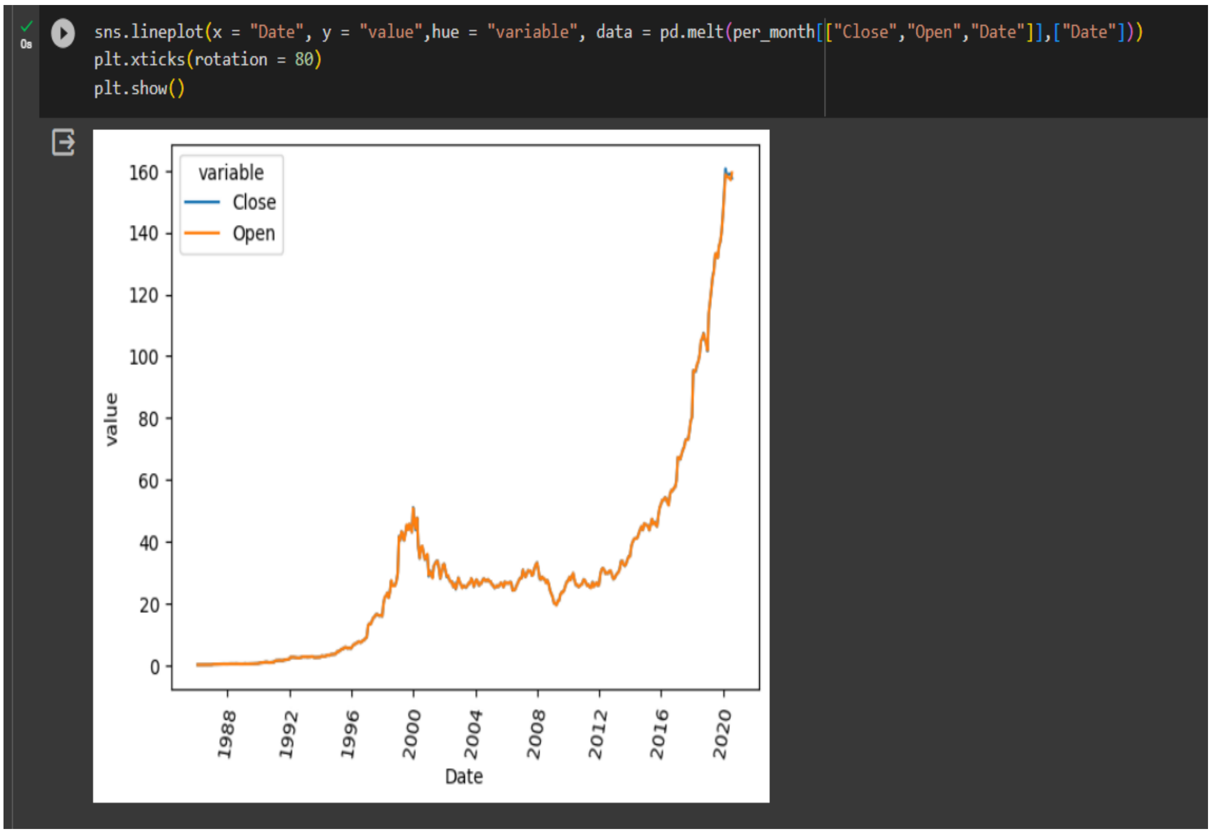


```
df.plot(kind='box',subplots=True,layout=(10,10),figsize=(20,20))
plt.show()
```



```
sns.pairplot(df)
```





```
sns.lineplot(x = "Date", y = "value",hue = "variable", data = pd.melt(per_month[["Close","Open","Date"]],["Date"]))
plt.xticks(rotation = 80)
plt.show()
```

PERFORMING DIFFERENT ACTIVITIES LIKE FEATURE ENGINEERING, MODEL TRAINING, EVALUATION

Feature selection:

1. Identify the target variable:

This is the variable that you want to predict, such as house price.

2. Explore the data.

This will help you to understand the relationships between the different features and the target variable. You can use data visualization and correlation analysis to identify features that are highly correlated with the target variable.

3. Remove redundant features.

If two features are highly correlated with each other, then you can remove one of the features, as they are likely to contain redundant information.

4. Remove irrelevant features.

If a feature is not correlated with the target variable, then you can remove it, as it is unlikely to be useful for prediction.

Model Training

Choose a machine learning algorithm.

There are a number of different machine learning algorithms that can be used for stock price prediction, such as linear regression, LSTM, KNN, ridge regression, lasso regression, decision trees, and random forests are Covered above.

Model evaluation:

Model evaluation is the process of assessing the performance of a machine learning model on unseen data. This is important to ensure that the model will generalize well to new data.

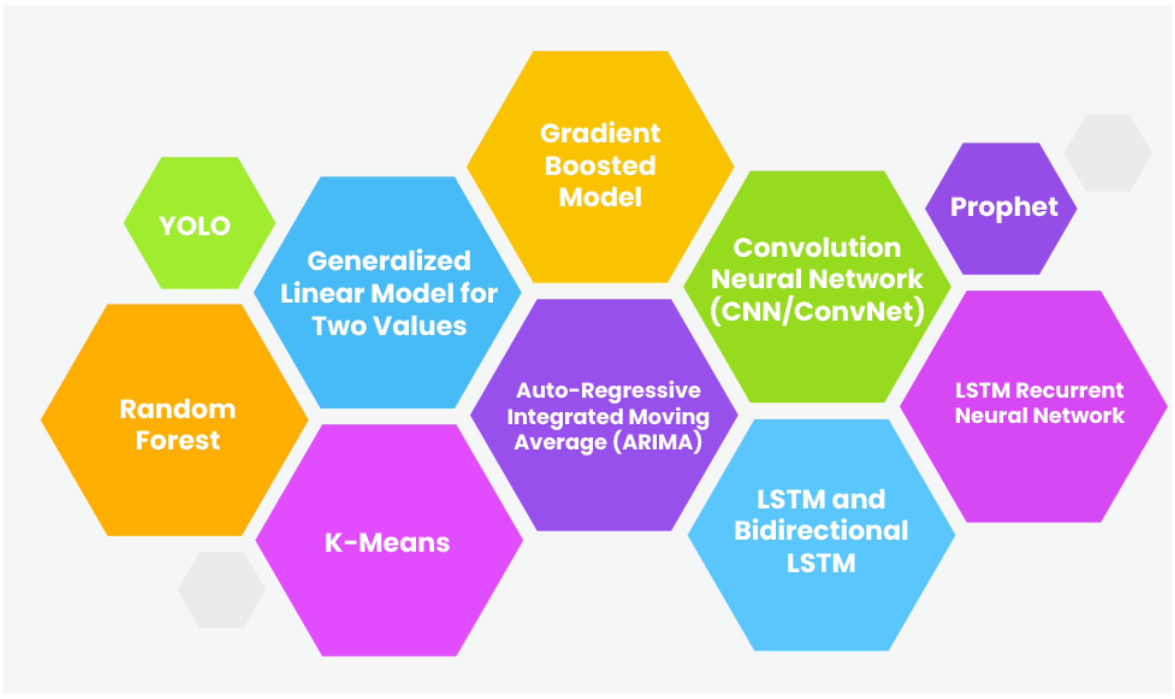
There are a number of different metrics that can be used to evaluate the performance of a stock price prediction model. Some of the most common measures are:

Root mean squared error (RMSE): This metric is the square root of the MSE.

Mean absolute error (MAE): This metric measures the average absolute difference between the predicted and actual house prices.

R-squared: This metric measures how well the model explains the variation in the actual house prices.

PREDICTIVE ANALTYICS TECHNIQUES


YOLO
Gradient Boosted Model
Prophet
Generalized Linear Model for Two Values
Convolution Neural Network (CNN/ConvNet)
Random Forest
Auto-Regressive Integrated Moving Average (ARIMA)
LSTM Recurrent Neural Network
K-Means
LSTM and Bidirectional LSTM

Some of the techniques are:

Overall, predictive analytics algorithms can be separated into two groups: machine learning and deep learning.

- **Machine learning** involves structural data that we see in a table. Algorithms for this comprise both linear and nonlinear varieties. Linear algorithms train more quickly, while nonlinear are better optimized for the problems they are likely to face (which are often nonlinear).
- **Deep learning** is a subset of machine learning that is more popular to deal with audio, video, text, and images.
- With machine learning predictive modelling, there are several different algorithms that can be applied.

- **Below are some of the most common algorithms that are being used to power the predictive analytics models described above.**

1. LSTM

- LSTMs are a type of Recurrent Neural Network (RNN) that can learn and memorize long-term dependencies.

- LSTM is designed to handle sequential data with long-term dependencies.

- It includes memory cells that can store and retrieve information over extended sequences.

- LSTMs are effective at capturing patterns and relationships in sequential data.
- They can be used for various tasks, including text generation, sentiment.
- Random Forest, XGBoost, and LSTM (Long Short-Term Memory) are three distinct machine learning models, each with its own characteristics and use cases.
- They can be applied to a wide range of problems, including classification, regression, and time series forecasting.

2. Random Forest:

- Random Forest is an ensemble of decision trees.
- It combines the predictions of multiple decision trees to produce a more accurate and stable prediction.
- It is capable of handling both categorical and numerical features.
- Random Forest provides feature importance scores, which can help identify the most important features in the dataset.

3. XGBoost (Extreme Gradient Boosting):

- XGBoost is a highly efficient and scalable implementation of gradient boosting machines.
- It is known for its superior performance on a wide range of machine learning tasks.
- XGBoost allows for custom loss functions and optimization objectives.
- It can handle missing data and is robust to outliers.
- It offers feature selection through importance scores.

4. Generalized Linear Model

- The generalized linear model is a complex extension of the general linear model. It takes the latter model's comparison of the effects of multiple variables on continuous variables. After that, it draws from various distributions to find the "best fit" model.
- The most important advantage of this predictive model is that it trains very quickly. Also, it helps to deal with the categorical predictors as it is simple to interpret. A generalized linear model helps understand how the predictors will affect future outcomes and resist overfitting.

5. Prophet

- The Prophet algorithm is generally used in forecast models and time series models. This predictive analytics algorithm was initially developed by Facebook and is used internally by the company for forecasting.
- The Prophet algorithm is excellent for capacity planning by automatically allocating the resources and setting appropriate sales goals.

- Manual forecasting of data requires hours of labour work with highly professional analysts to draw out accurate outputs. With inconsistent performance levels and inflexibility of other forecasting algorithms, the prophet algorithm is a valuable alternative.

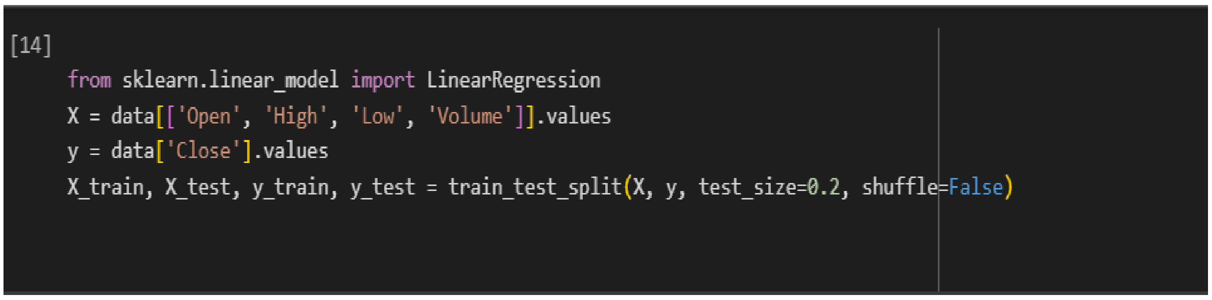
6. Convolution Neural Network (CNN/ConvNet)

- Convolution neural networks(CNN) is artificial neural network that performs feature detection in image data.
- They are based on the convolution operation, transforming the input image into a matrix where rows and columns correspond to different image planes and differentiate one object.
- The architecture of the CNN model is inspired by the visual cortex of the human brain. As a result, it is quite similar to the pattern of neurons connected in the human brain. Individual neurons of the model respond to stimuli only to specific regions of the visual field known as the Receptive Field.

TECHNIQUES

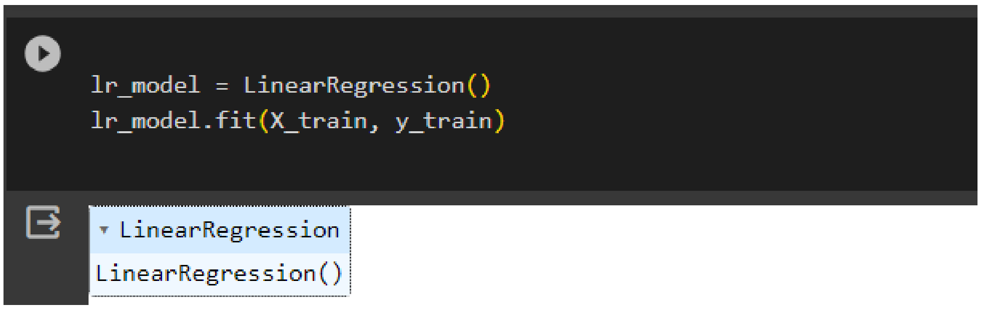
1. Linear Regression

Data Preparation



```
[14]
    from sklearn.linear_model import LinearRegression
    X = data[['Open', 'High', 'Low', 'Volume']].values
    y = data['Close'].values
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

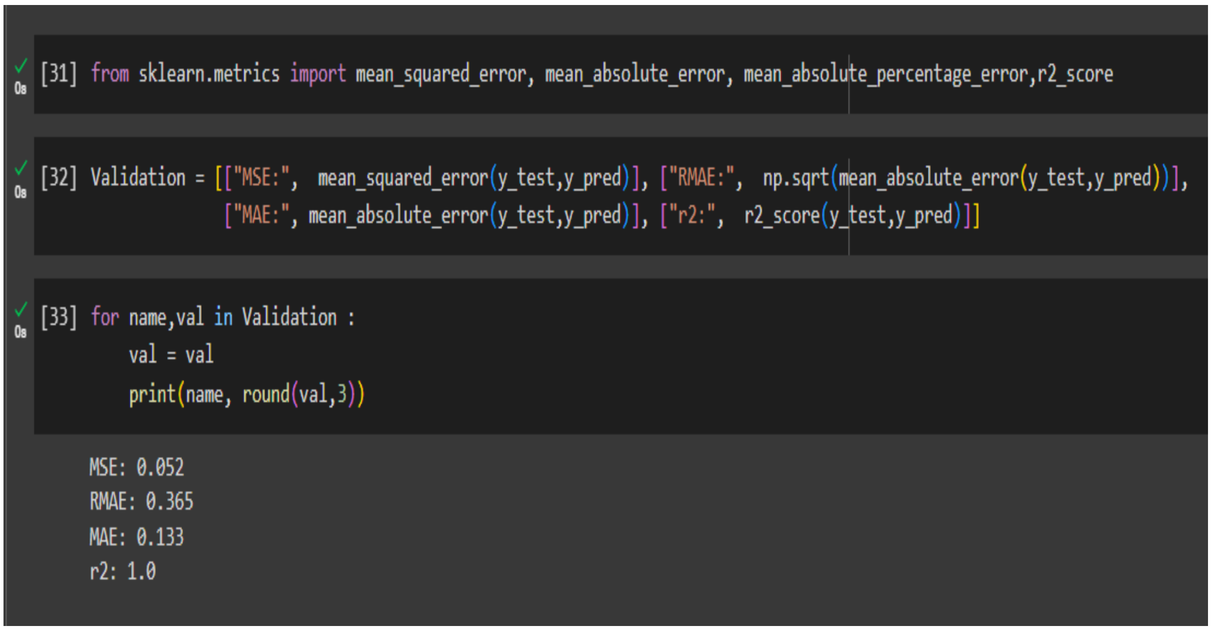
Model training



```
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
```

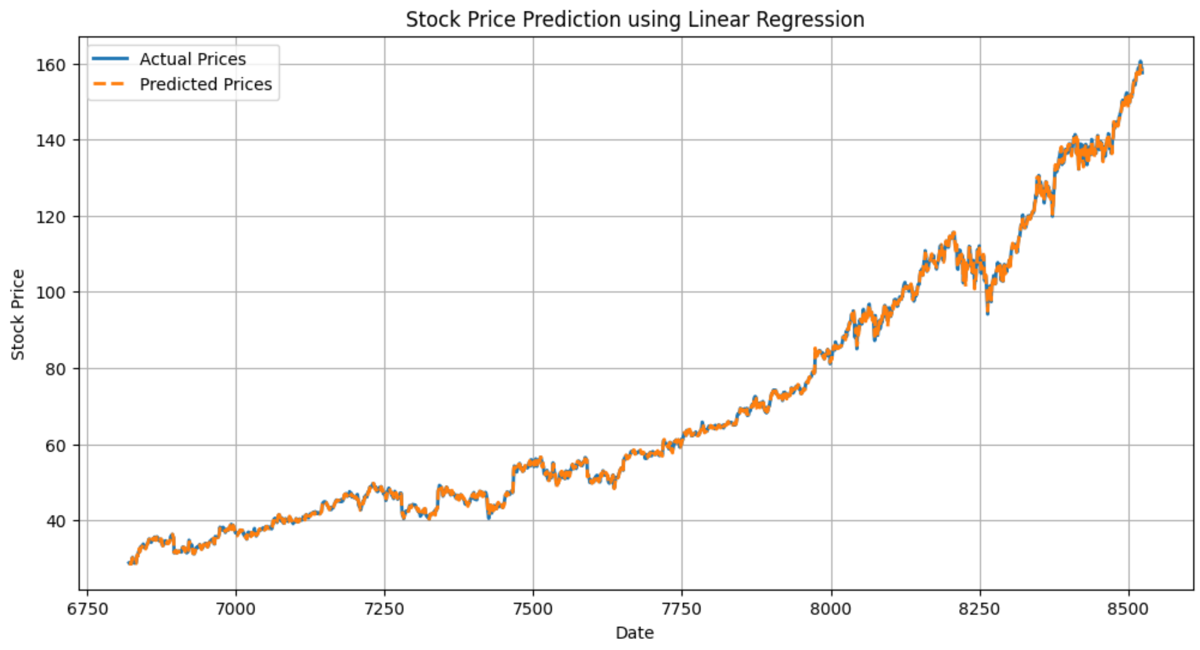
```
▾ LinearRegression
LinearRegression()
```

Model Evaluation



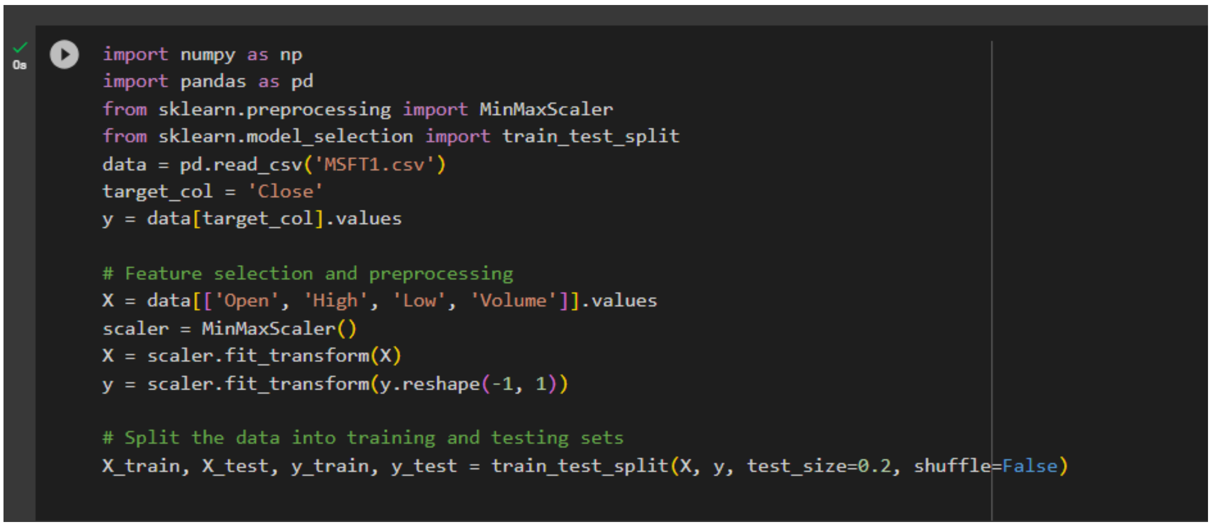
```
[31] from sklearn.metrics import mean_squared_error, mean_absolute_error, mean_absolute_percentage_error,r2_score

[32] Validation = [["MSE:",  mean_squared_error(y_test,y_pred)], ["RMAE:",  np.sqrt(mean_absolute_error(y_test,y_pred))],
                 ["MAE:", mean_absolute_error(y_test,y_pred)], ["r2:",  r2_score(y_test,y_pred)]]

[33] for name,val in Validation :
         val = val
         print(name, round(val,3))

MSE: 0.052
RMAE: 0.365
MAE: 0.133
r2: 1.0
```

Visualization of Linear regression



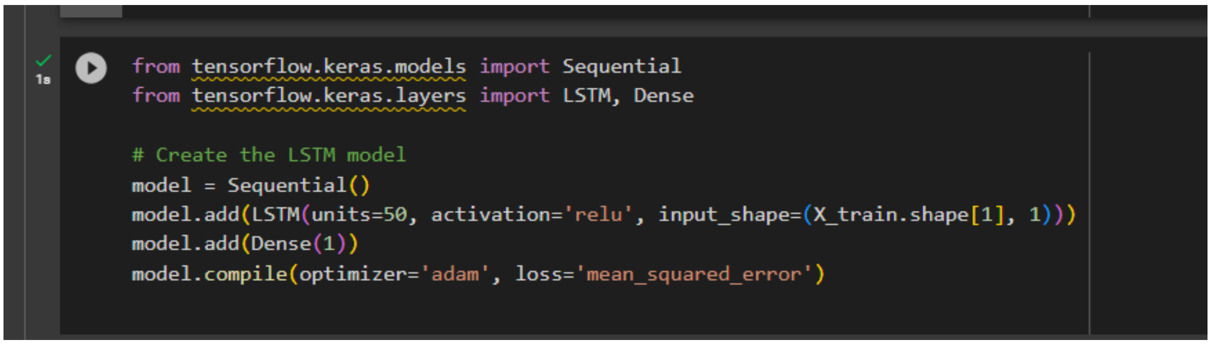
2.LSTM

Data preparation



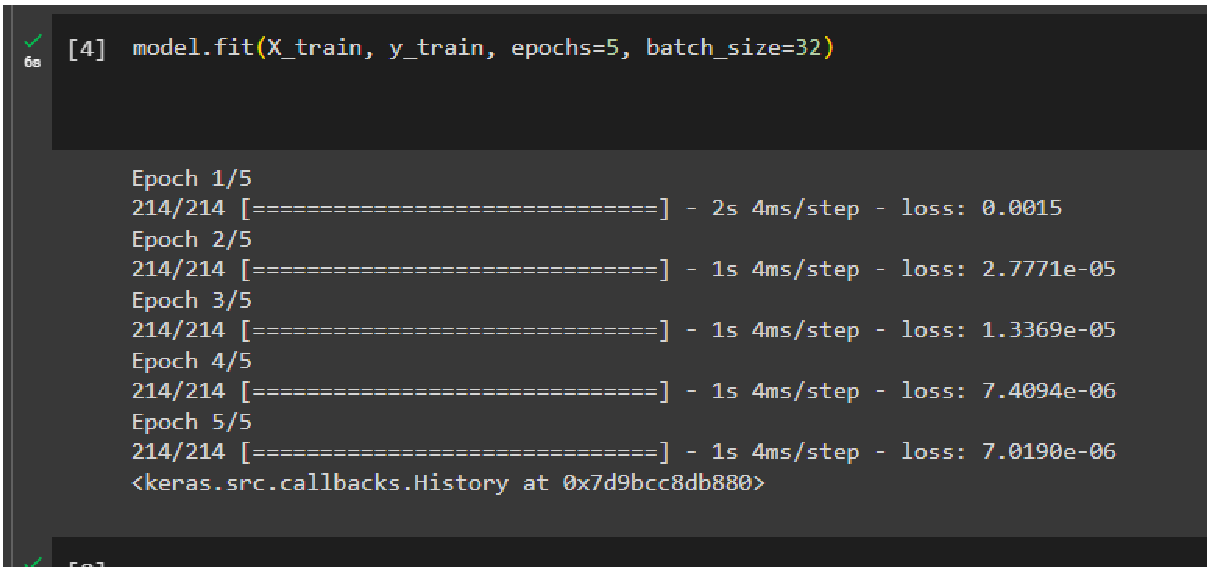
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
data = pd.read_csv('MSFT1.csv')
target_col = 'Close'
y = data[target_col].values

# Feature selection and preprocessing
X = data[['Open', 'High', 'Low', 'Volume']].values
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
y = scaler.fit_transform(y.reshape(-1, 1))

# Split 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)
```

LSTM Model Creation



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Create the LSTM model
model = Sequential()
model.add(LSTM(units=50, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')
```

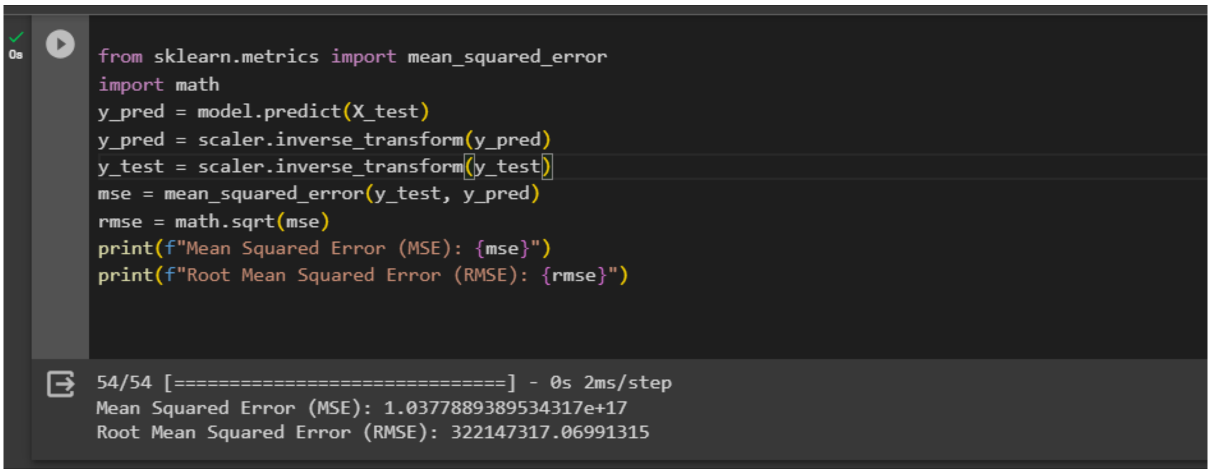
Model training



```
[4] model.fit(X_train, y_train, epochs=5, batch_size=32)
```

```
Epoch 1/5
214/214 [==============================] - 2s 4ms/step - loss: 0.0015
Epoch 2/5
214/214 [==============================] - 1s 4ms/step - loss: 2.7771e-05
Epoch 3/5
214/214 [==============================] - 1s 4ms/step - loss: 1.3369e-05
Epoch 4/5
214/214 [==============================] - 1s 4ms/step - loss: 7.4094e-06
Epoch 5/5
214/214 [==============================] - 1s 4ms/step - loss: 7.0190e-06
<keras.src.callbacks.History at 0x7d9bcc8db880>
```

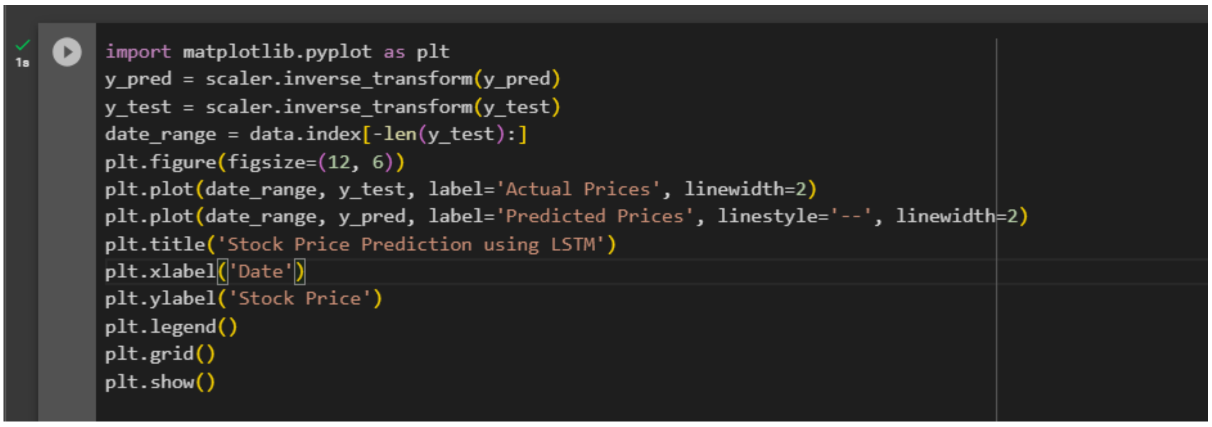
Model Evaluation



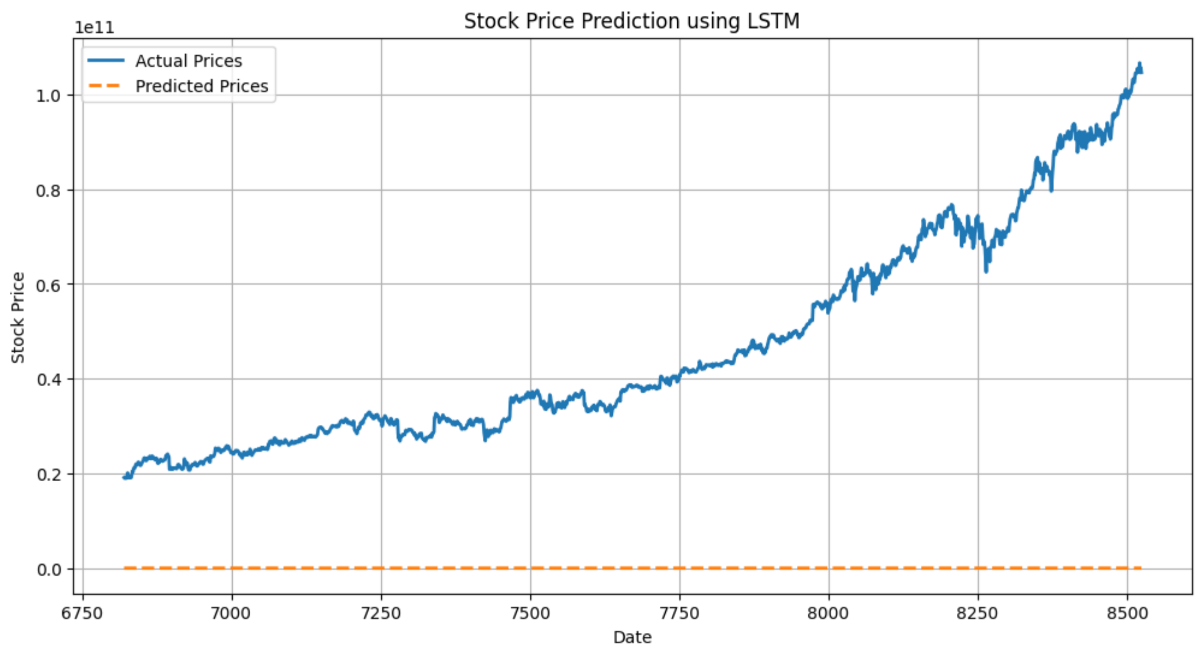
```
from sklearn.metrics import mean_squared_error
import math
y_pred = model.predict(X_test)
y_pred = scaler.inverse_transform(y_pred)
y_test = scaler.inverse_transform(y_test)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

```
54/54 [==============================] - 0s 2ms/step
Mean Squared Error (MSE): 1.0377889389534317e+17
Root Mean Squared Error (RMSE): 322147317.06991315
```

Visualization of LSTM Model

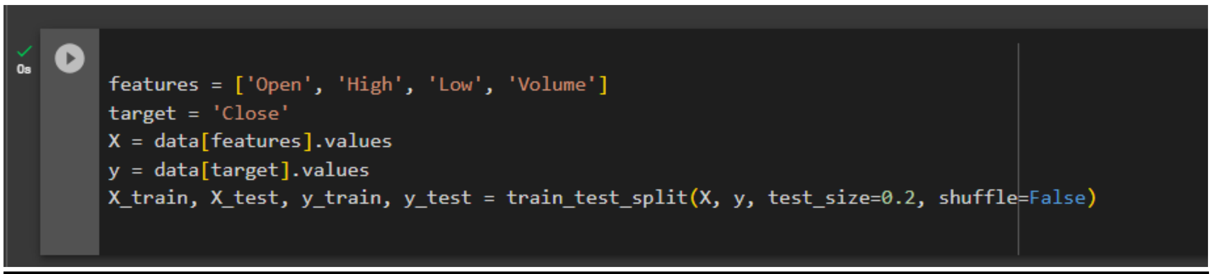


```
import matplotlib.pyplot as plt
y_pred = scaler.inverse_transform(y_pred)
y_test = scaler.inverse_transform(y_test)
date_range = data.index[-len(y_test):]
plt.figure(figsize=(12, 6))
plt.plot(date_range, y_test, label='Actual Prices', linewidth=2)
plt.plot(date_range, y_pred, label='Predicted Prices', linestyle='--', linewidth=2)
plt.title('Stock Price Prediction using LSTM')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.grid()
plt.show()
```



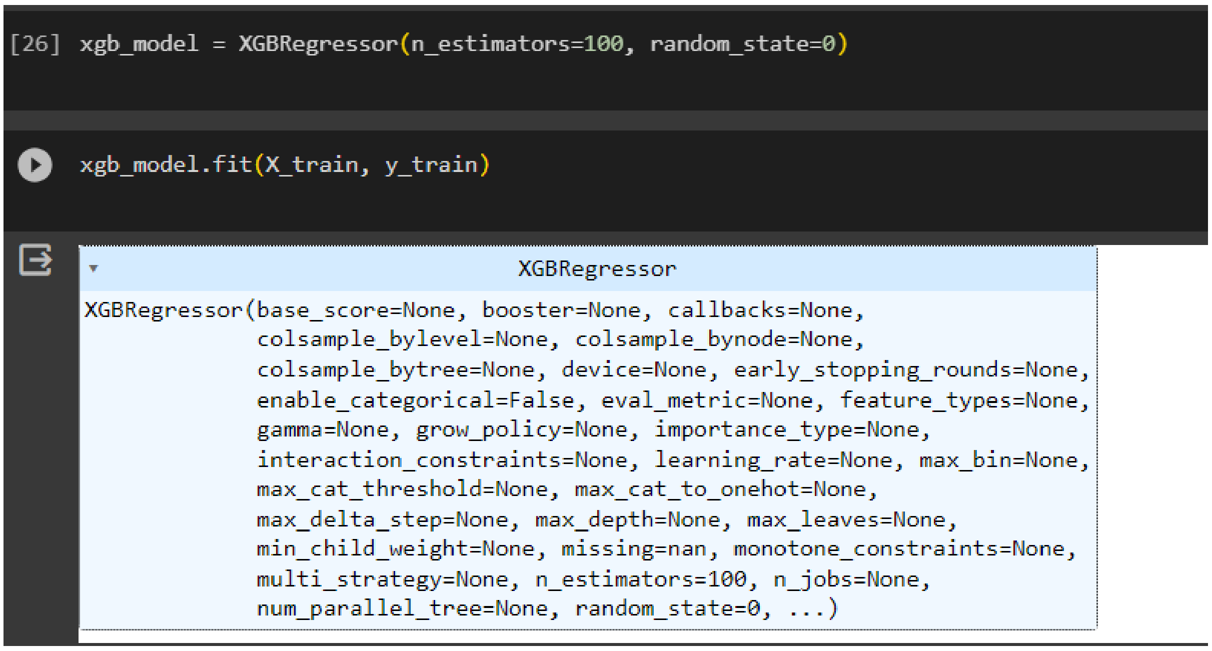
3.XGBoost

Data Preparation



```
features = ['Open', 'High', 'Low', 'Volume']
target = 'Close'
X = data[features].values
y = data[target].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

XGBoost Model Building and Training

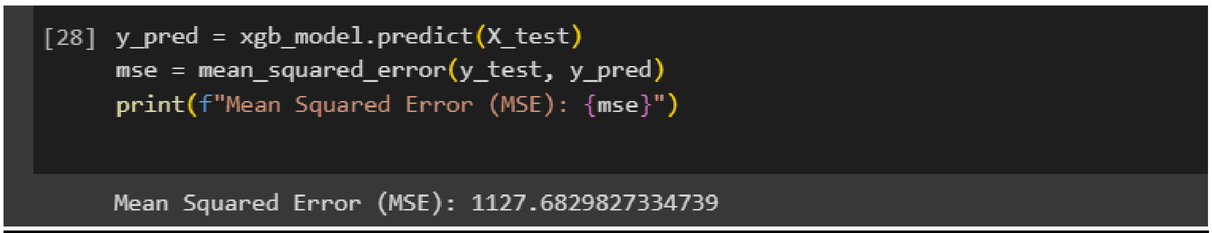


```
[26] xgb_model = XGBRegressor(n_estimators=100, random_state=0)
```

```
xgb_model.fit(X_train, y_train)
```

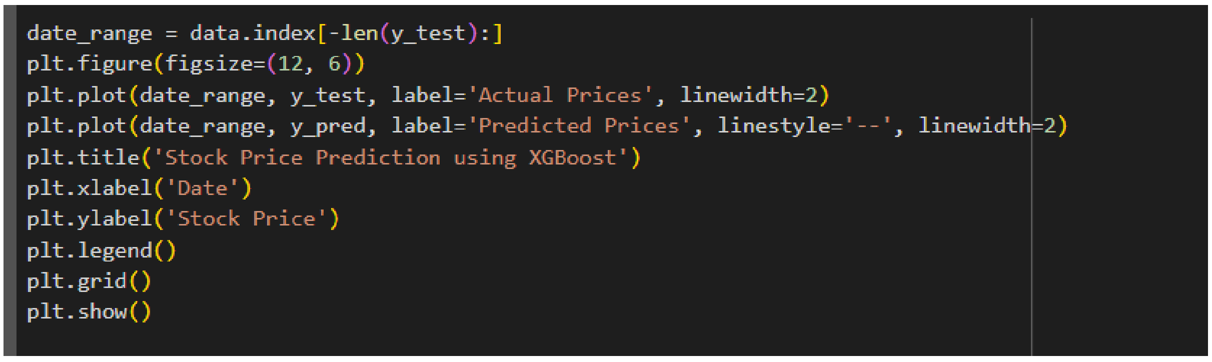
```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=100, n_jobs=None,
             num_parallel_tree=None, random_state=0, ...)
```

Model Evaluation

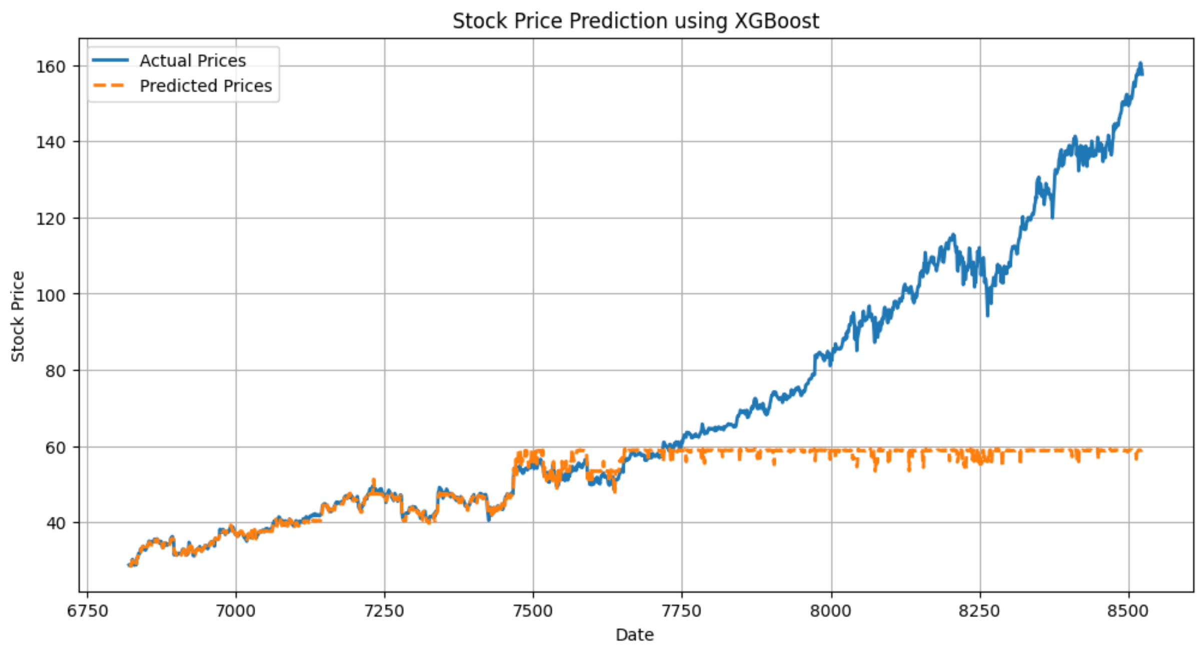


```
[28] y_pred = xgb_model.predict(X_test)
     mse = mean_squared_error(y_test, y_pred)
     print(f"Mean Squared Error (MSE): {mse}")

     Mean Squared Error (MSE): 1127.6829827334739
```

Visualization of XGBoost

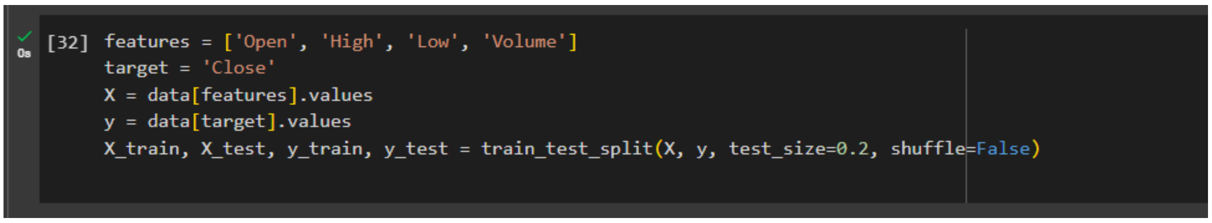


```
date_range = data.index[-len(y_test):]
plt.figure(figsize=(12, 6))
plt.plot(date_range, y_test, label='Actual Prices', linewidth=2)
plt.plot(date_range, y_pred, label='Predicted Prices', linestyle='--', linewidth=2)
plt.title('Stock Price Prediction using XGBoost')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.grid()
plt.show()
```



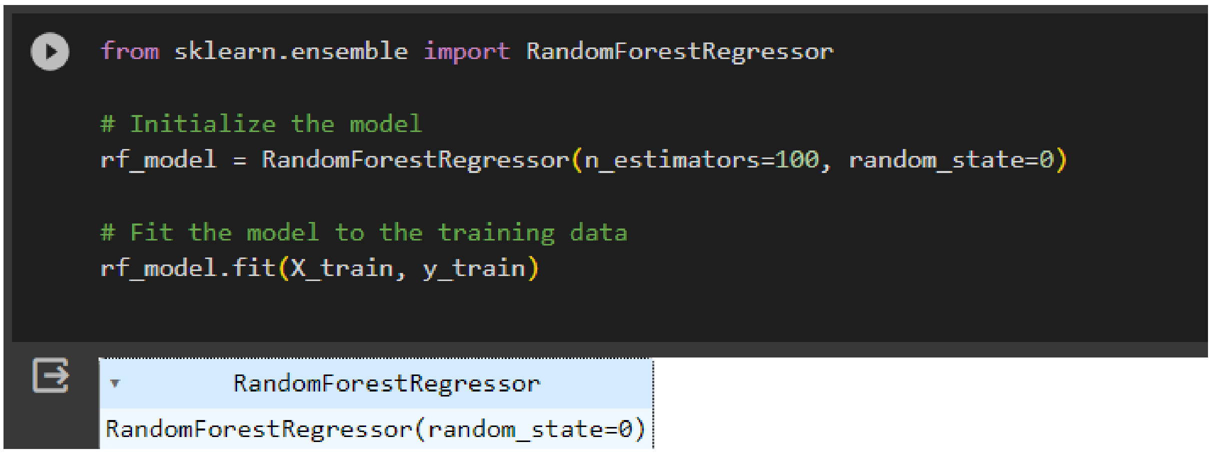
4. Random Forest

Data Preparation



```
[32] features = ['Open', 'High', 'Low', 'Volume']
     target = 'Close'
     X = data[features].values
     y = data[target].values
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

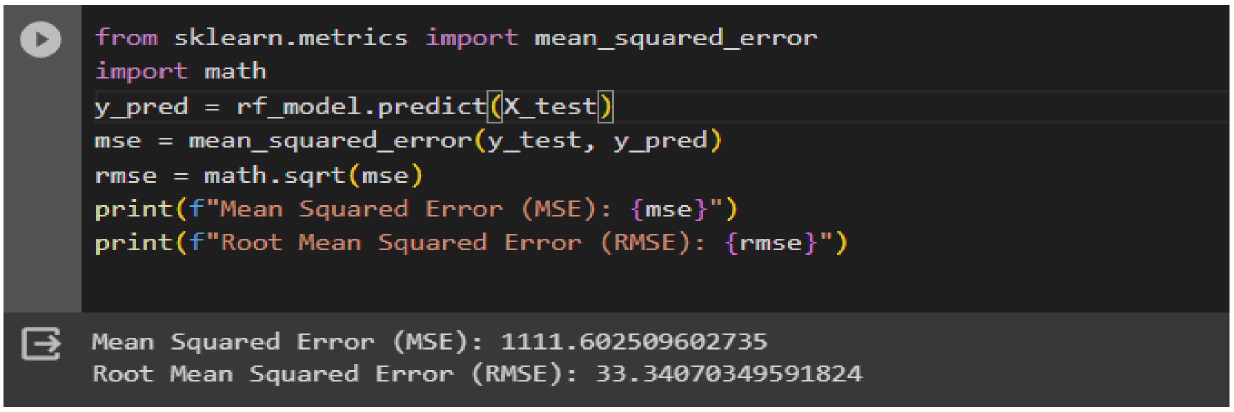
Model Training



```
from sklearn.ensemble import RandomForestRegressor

# Initialize the model
rf_model = RandomForestRegressor(n_estimators=100, random_state=0)

# Fit the model to the training data
rf_model.fit(X_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=0)
```

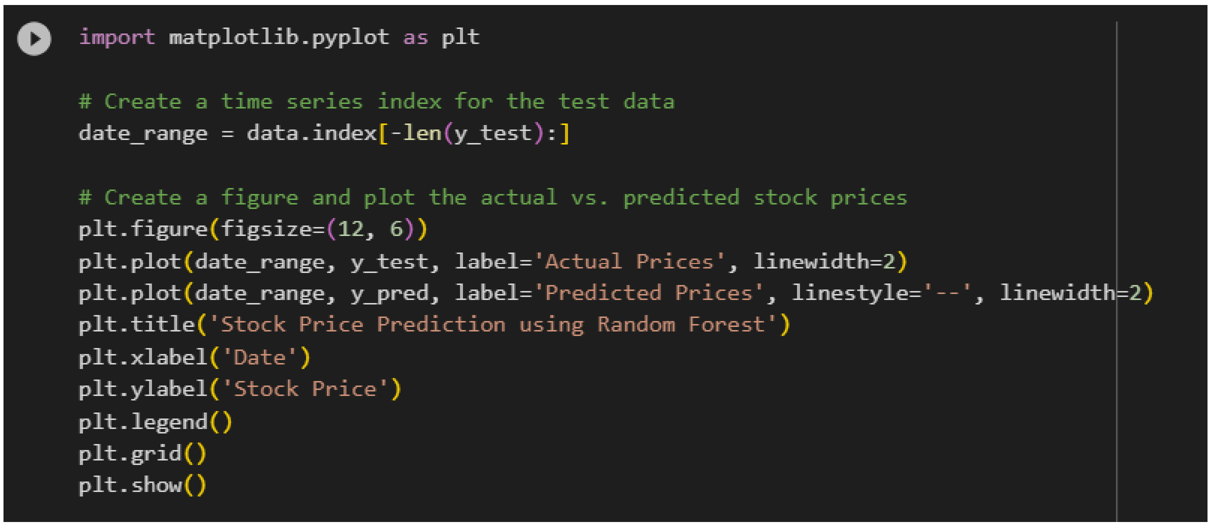
Model Evaluation



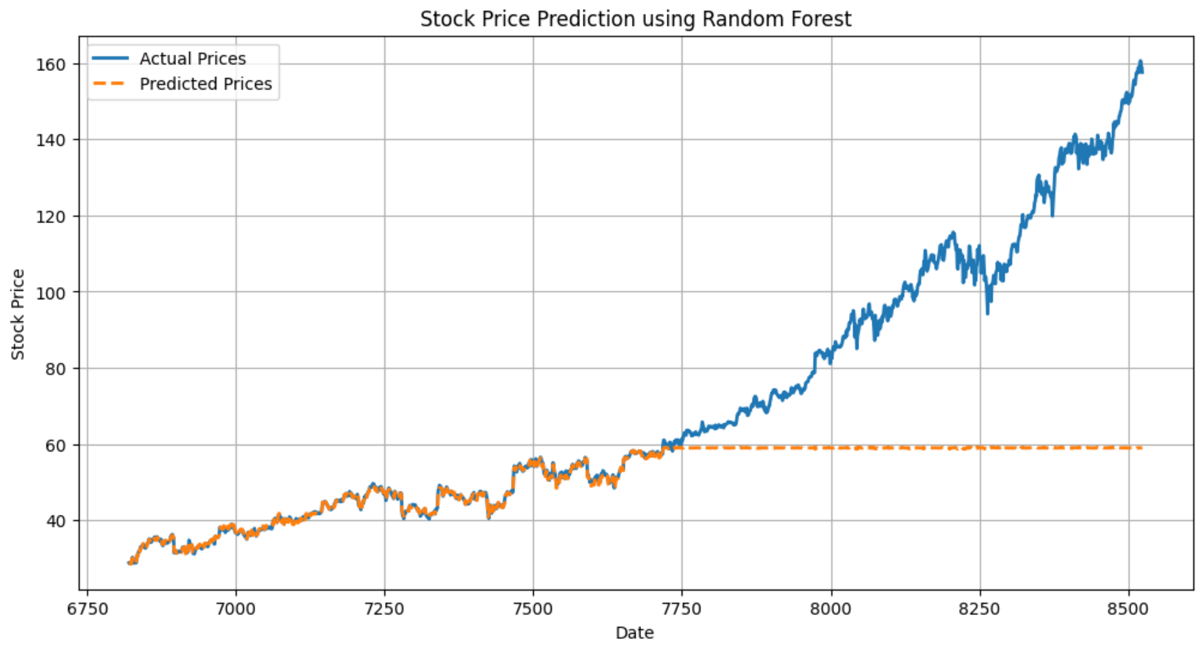
```
from sklearn.metrics import mean_squared_error
import math
y_pred = rf_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

```
Mean Squared Error (MSE): 1111.602509602735
Root Mean Squared Error (RMSE): 33.34070349591824
```

Visualization of Random Forest

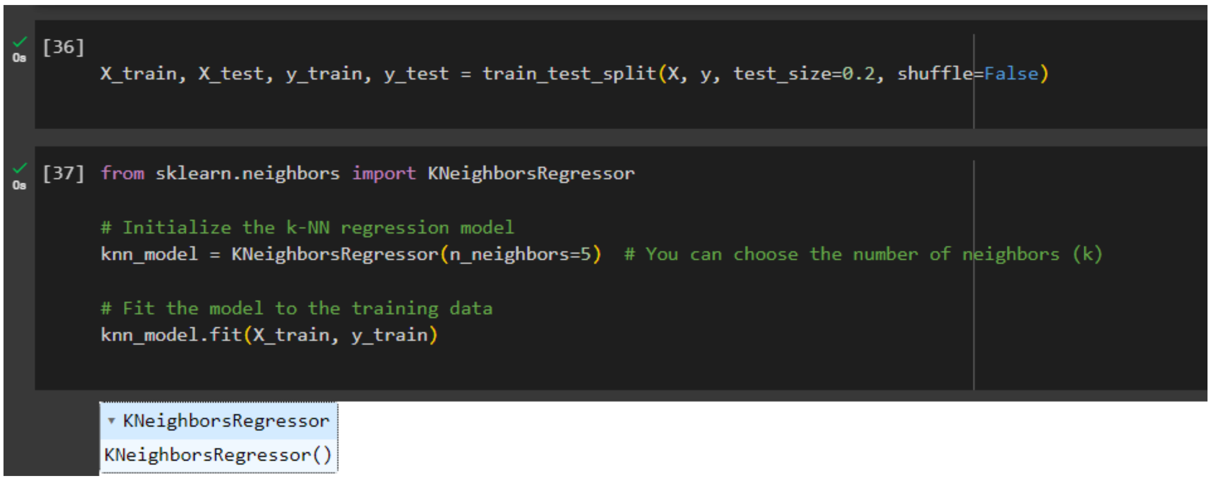


```
import matplotlib.pyplot as plt

# Create a time series index for the test data
date_range = data.index[-len(y_test):]

# Create a figure and plot the actual vs. predicted stock prices
plt.figure(figsize=(12, 6))
plt.plot(date_range, y_test, label='Actual Prices', linewidth=2)
plt.plot(date_range, y_pred, label='Predicted Prices', linestyle='--', linewidth=2)
plt.title('Stock Price Prediction using Random Forest')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.grid()
plt.show()
```



5. KNN

Model Training

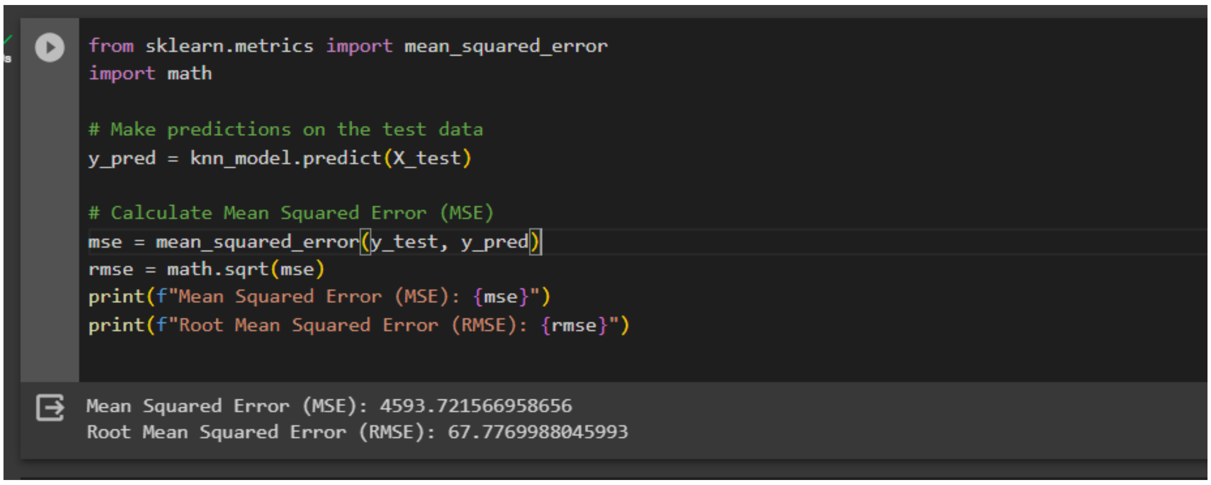


```
[36]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

```
[37] from sklearn.neighbors import KNeighborsRegressor

     # Initialize the k-NN regression model
     knn_model = KNeighborsRegressor(n_neighbors=5)  # You can choose the number of neighbors (k)

     # Fit the model to the training data
     knn_model.fit(X_train, y_train)
```

```
▾ KNeighborsRegressor
KNeighborsRegressor()
```

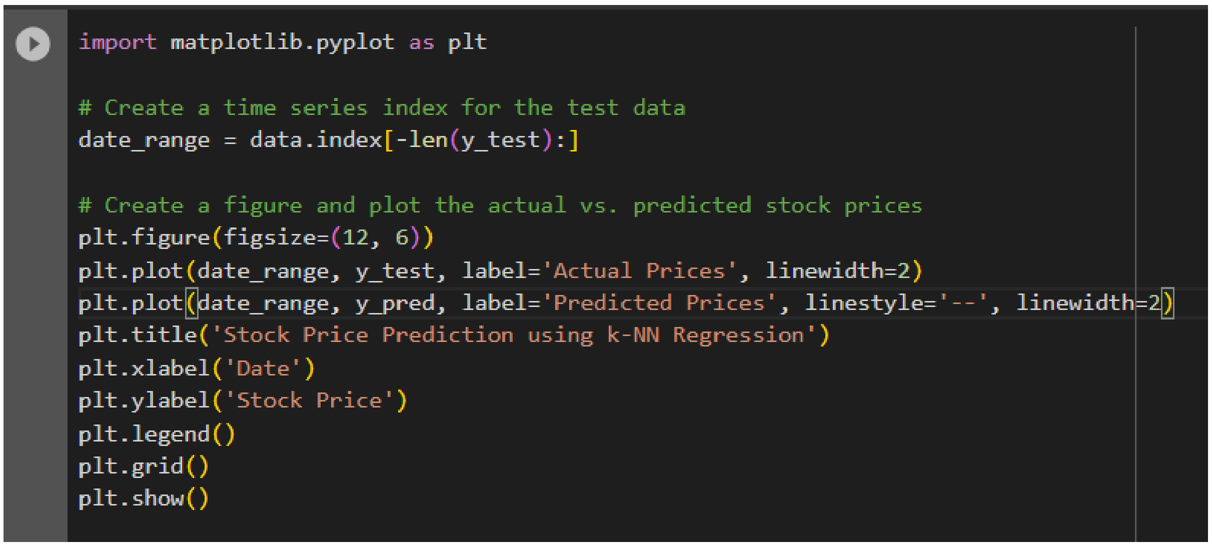
Model Evaluation



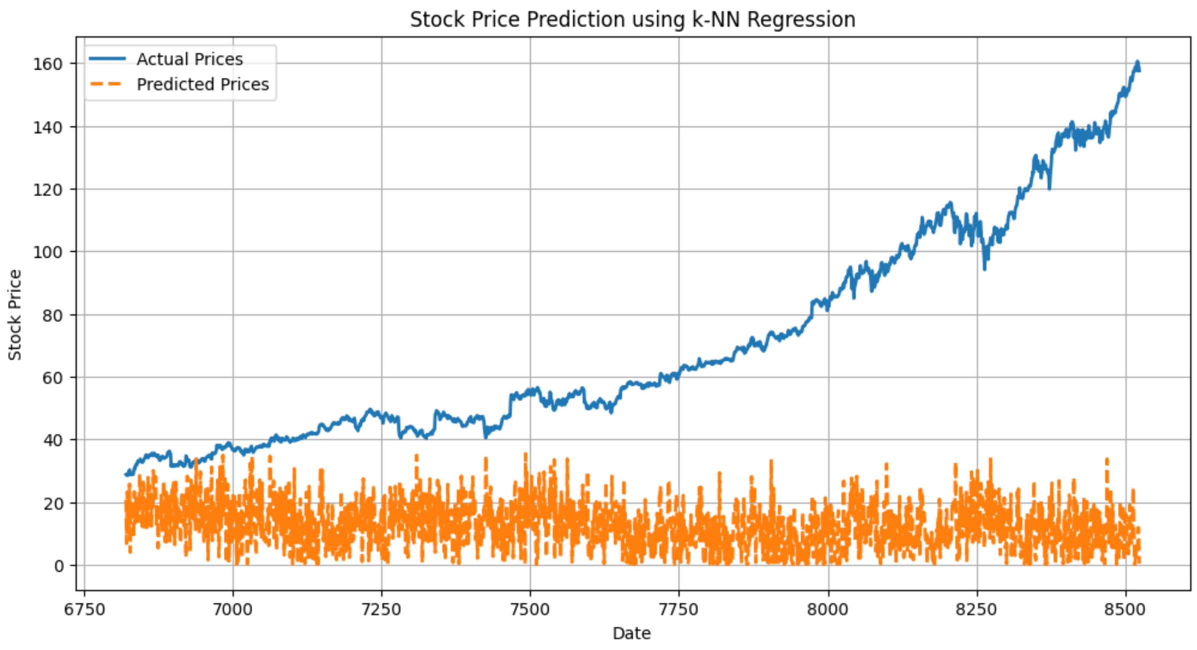
```
from sklearn.metrics import mean_squared_error
import math

# Make predictions on the test data
y_pred = knn_model.predict(X_test)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

```
Mean Squared Error (MSE): 4593.721566958656
Root Mean Squared Error (RMSE): 67.7769988045993
```

Visualization of KNN

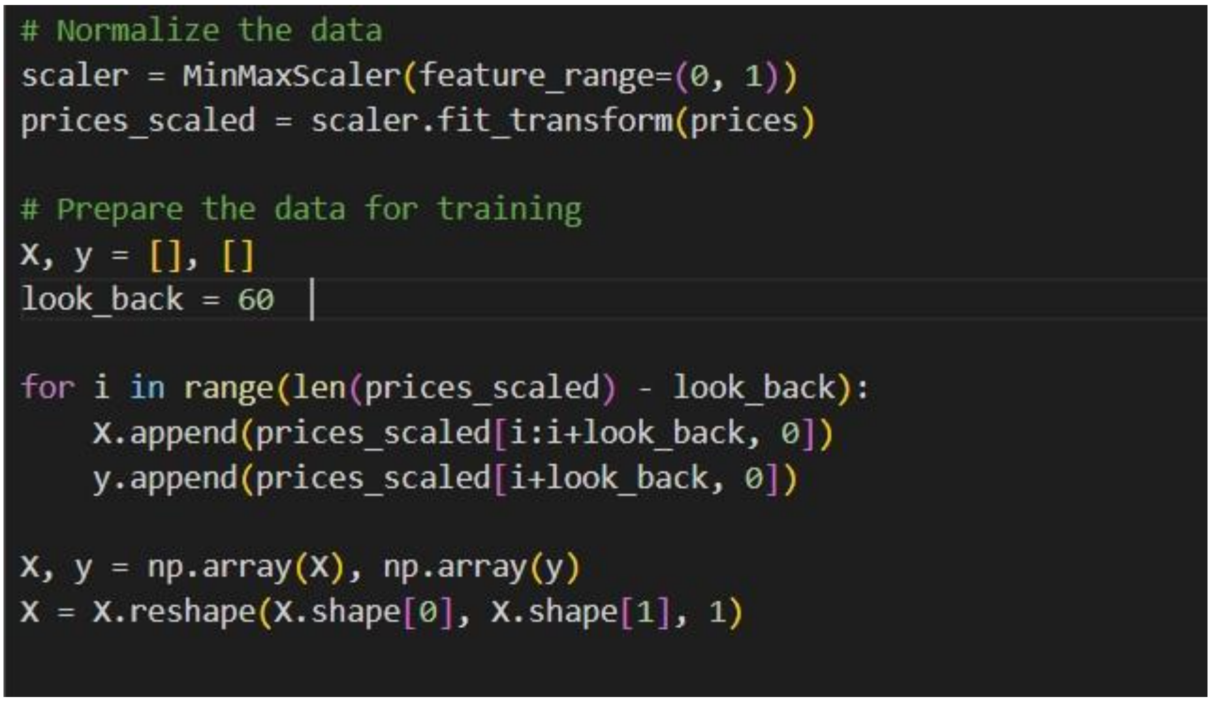


```
import matplotlib.pyplot as plt

# Create a time series index for the test data
date_range = data.index[-len(y_test):]

# Create a figure and plot the actual vs. predicted stock prices
plt.figure(figsize=(12, 6))
plt.plot(date_range, y_test, label='Actual Prices', linewidth=2)
plt.plot(date_range, y_pred, label='Predicted Prices', linestyle='--', linewidth=2)
plt.title('Stock Price Prediction using k-NN Regression')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.grid()
plt.show()
```

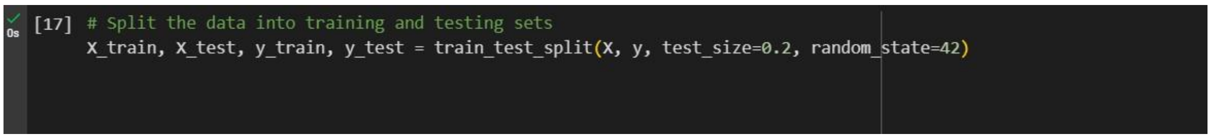


7. CNN

Model Training

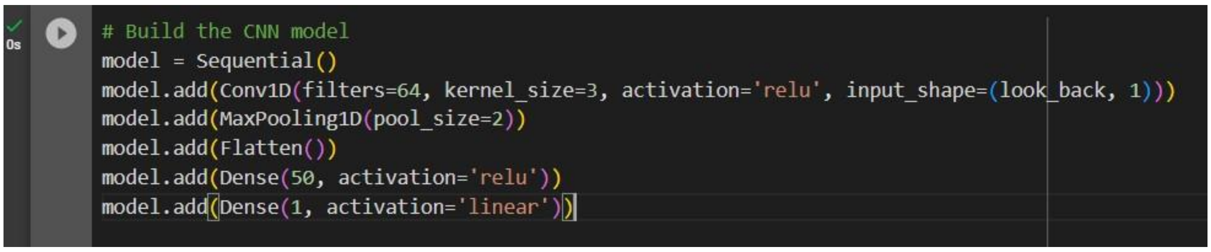


```
# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
prices_scaled = scaler.fit_transform(prices)

# Prepare the data for training
X, y = [], []
look_back = 60

for i in range(len(prices_scaled) - look_back):
    X.append(prices_scaled[i:i+look_back, 0])
    y.append(prices_scaled[i+look_back, 0])

X, y = np.array(X), np.array(y)
X = X.reshape(X.shape[0], X.shape[1], 1)
```



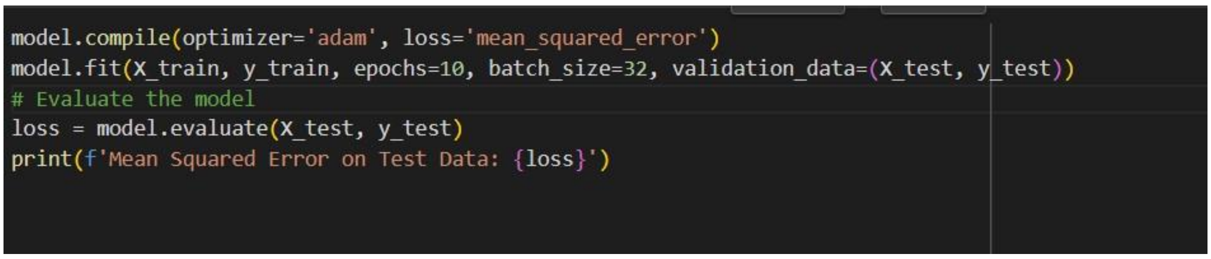
```
[17] # Split the data into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Build CNN Model



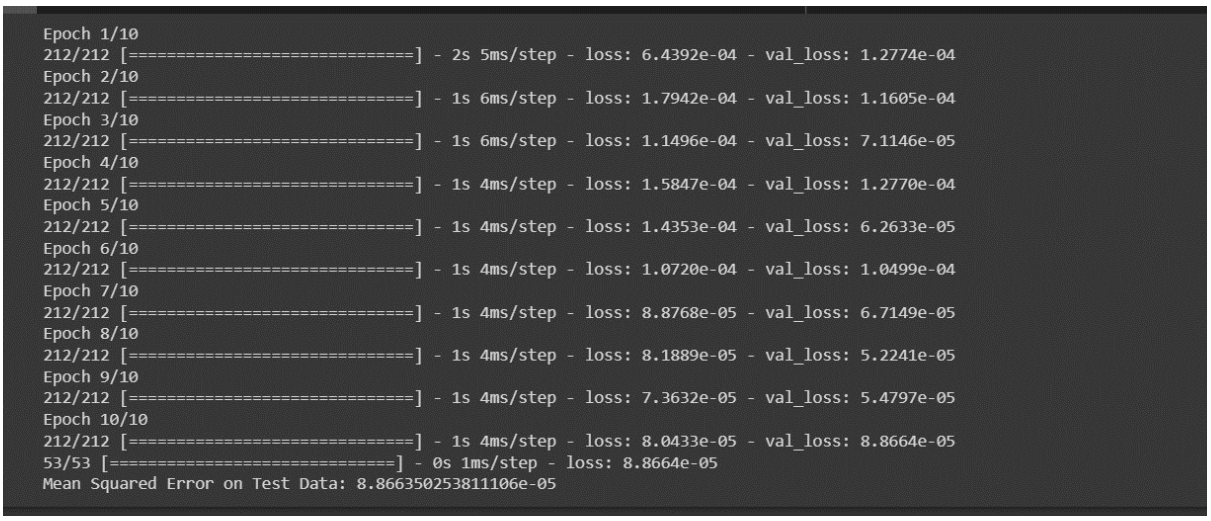
```
# Build the CNN model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(look_back, 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(1, activation='linear'))
```

Evaluate the Model



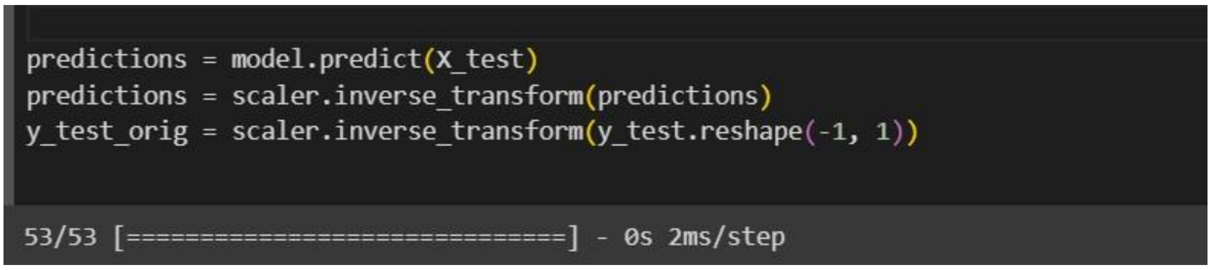
```
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
# Evaluate the model
loss = model.evaluate(X_test, y_test)
print(f'Mean Squared Error on Test Data: {loss}')
```

FIND EPOCH AND MSE:



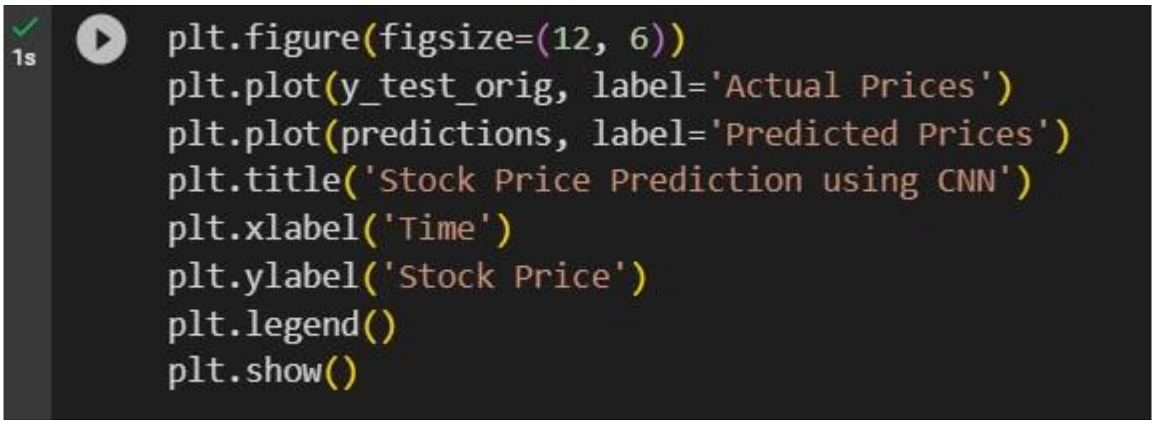
```
Epoch 1/10
212/212 [==============================] - 2s 5ms/step - loss: 6.4392e-04 - val_loss: 1.2774e-04
Epoch 2/10
212/212 [==============================] - 1s 6ms/step - loss: 1.7942e-04 - val_loss: 1.1605e-04
Epoch 3/10
212/212 [==============================] - 1s 6ms/step - loss: 1.1496e-04 - val_loss: 7.1146e-05
Epoch 4/10
212/212 [==============================] - 1s 4ms/step - loss: 1.5847e-04 - val_loss: 1.2770e-04
Epoch 5/10
212/212 [==============================] - 1s 4ms/step - loss: 1.4353e-04 - val_loss: 6.2633e-05
Epoch 6/10
212/212 [==============================] - 1s 4ms/step - loss: 1.0720e-04 - val_loss: 1.0499e-04
Epoch 7/10
212/212 [==============================] - 1s 4ms/step - loss: 8.8768e-05 - val_loss: 6.7149e-05
Epoch 8/10
212/212 [==============================] - 1s 4ms/step - loss: 8.1889e-05 - val_loss: 5.2241e-05
Epoch 9/10
212/212 [==============================] - 1s 4ms/step - loss: 7.3632e-05 - val_loss: 5.4797e-05
Epoch 10/10
212/212 [==============================] - 1s 4ms/step - loss: 8.0433e-05 - val_loss: 8.8664e-05
53/53 [==============================] - 0s 1ms/step - loss: 8.8664e-05
Mean Squared Error on Test Data: 8.866350253811106e-05
```

Predict the model



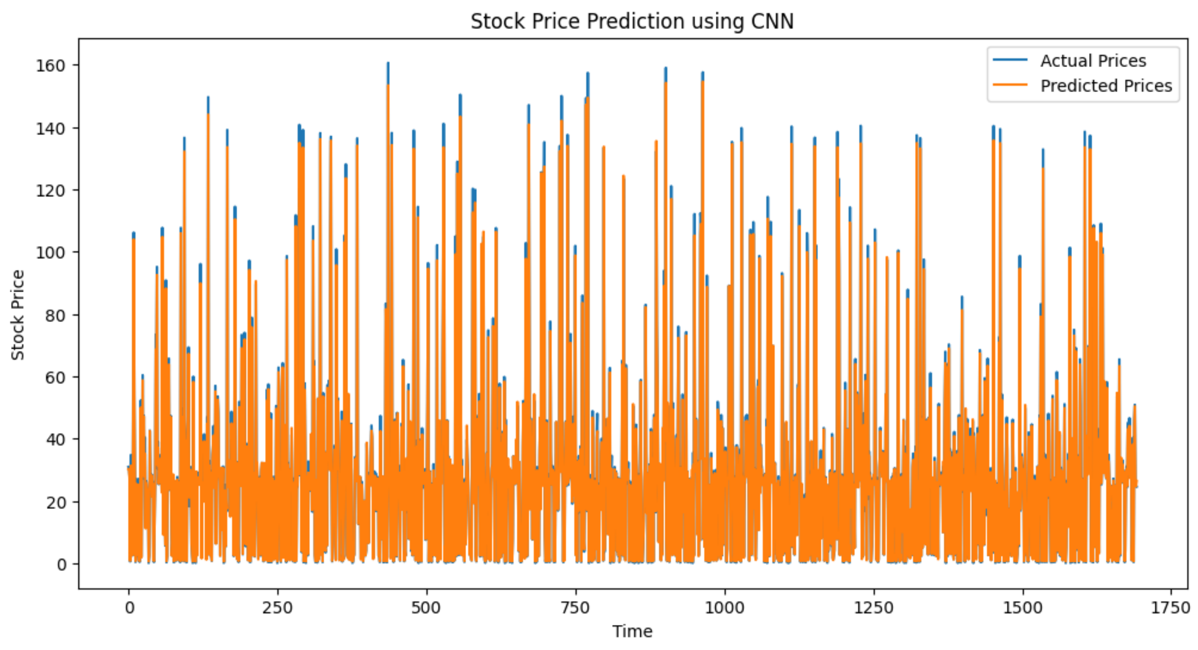
```
predictions = model.predict(X_test)
predictions = scaler.inverse_transform(predictions)
y_test_orig = scaler.inverse_transform(y_test.reshape(-1, 1))

53/53 [==============================] - 0s 2ms/step
```

Visualize the model

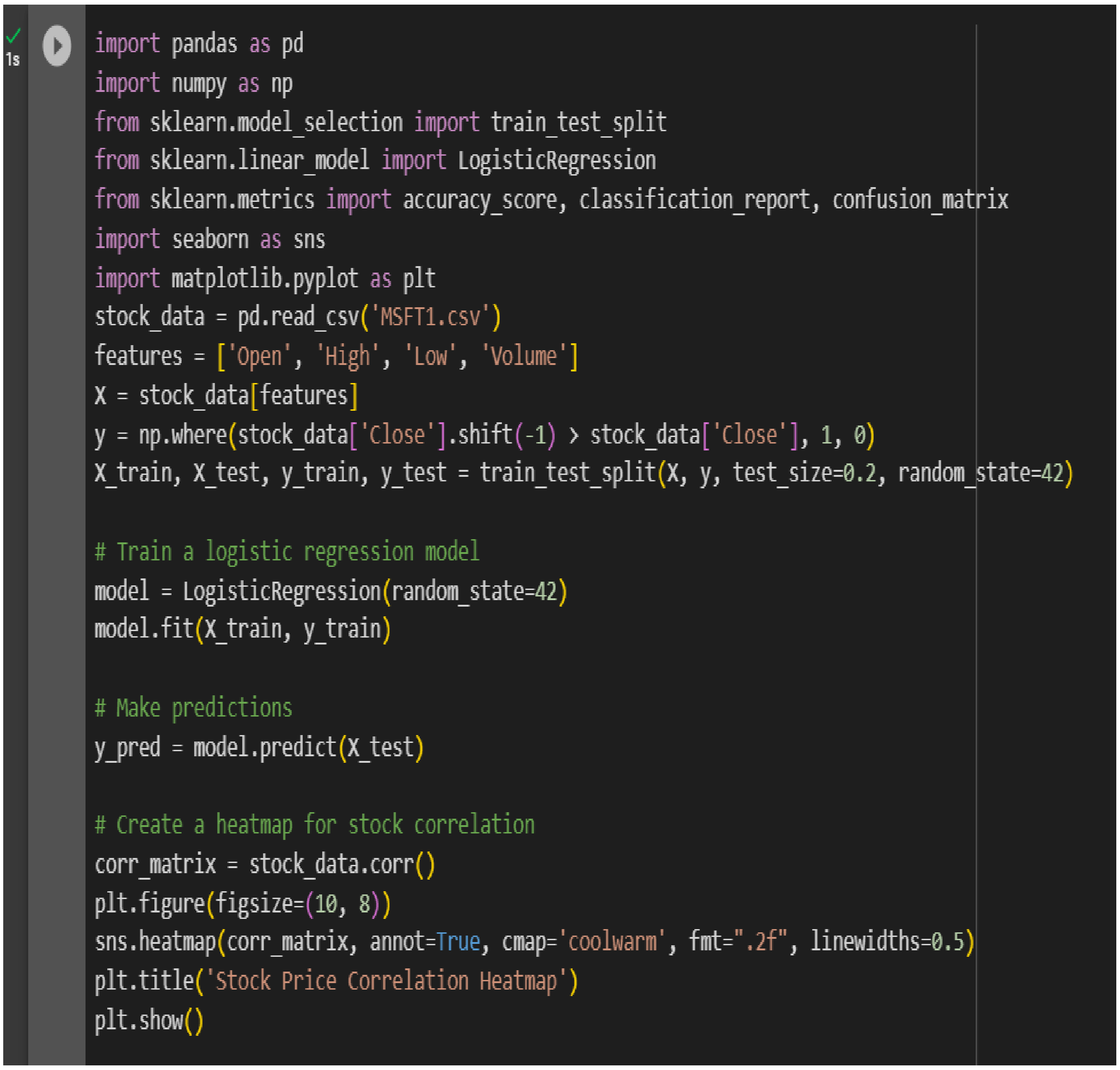


```
plt.figure(figsize=(12, 6))
plt.plot(y_test_orig, label='Actual Prices')
plt.plot(predictions, label='Predicted Prices')
plt.title('Stock Price Prediction using CNN')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

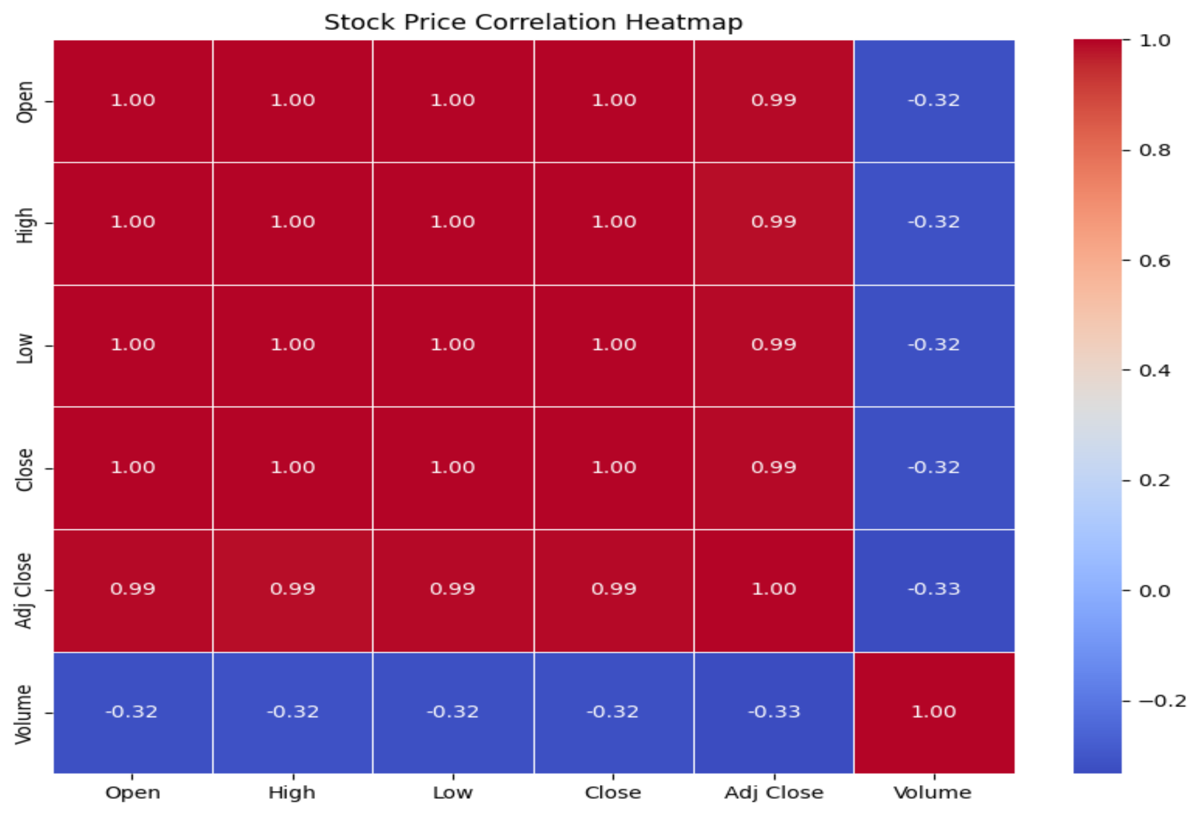
O/P:



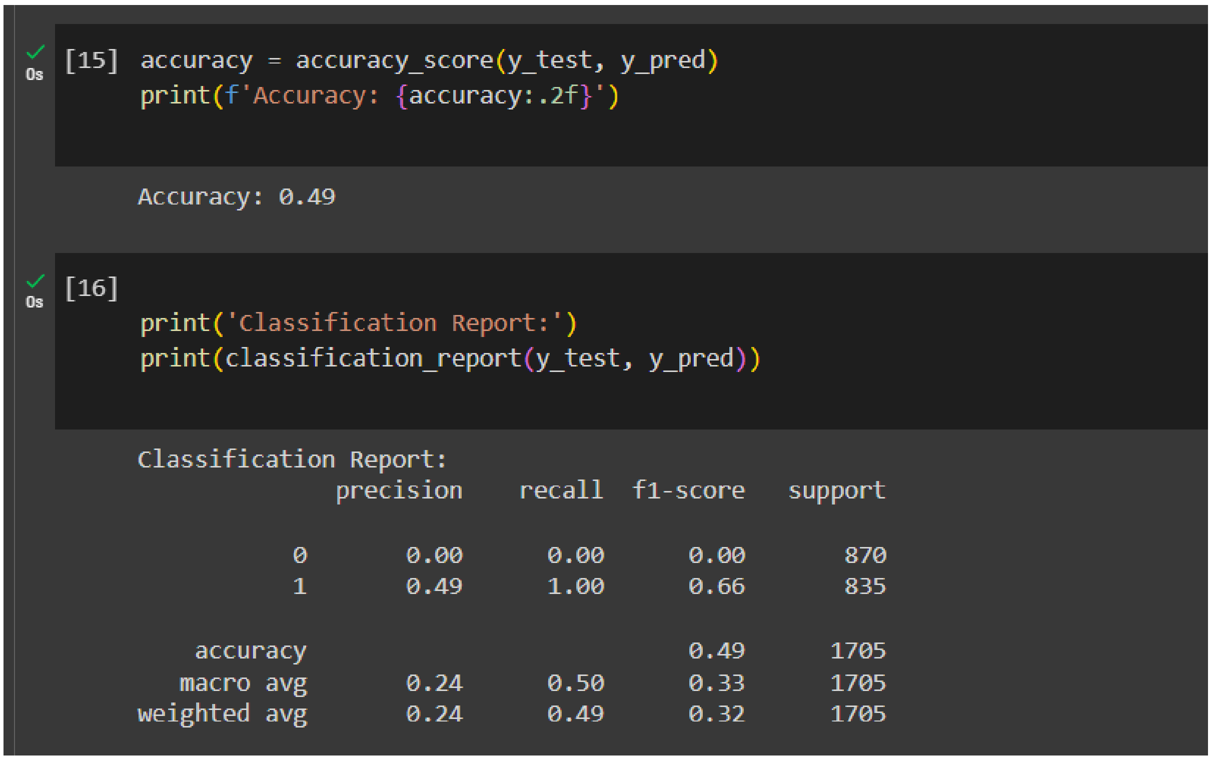
6. Logistic regression



```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
stock_data = pd.read_csv('MSFT1.csv')
features = ['Open', 'High', 'Low', 'Volume']
X = stock_data[features]
y = np.where(stock_data['Close'].shift(-1) > stock_data['Close'], 1, 0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a logistic regression model
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Create a heatmap for stock correlation
corr_matrix = stock_data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Stock Price Correlation Heatmap')
plt.show()
```



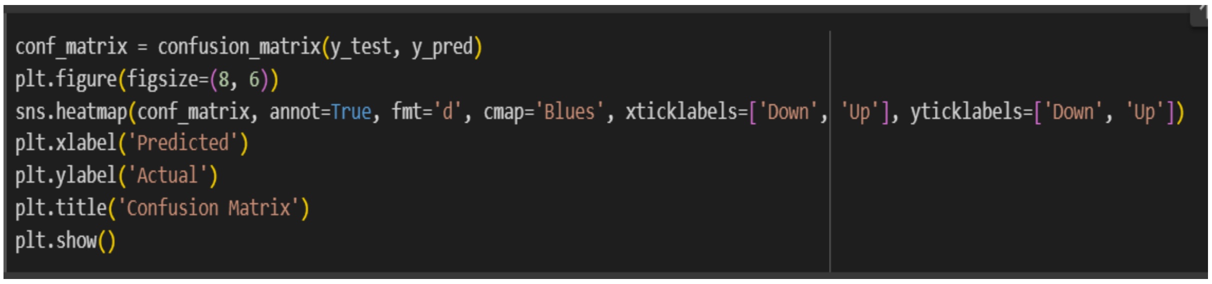
Print accuracy



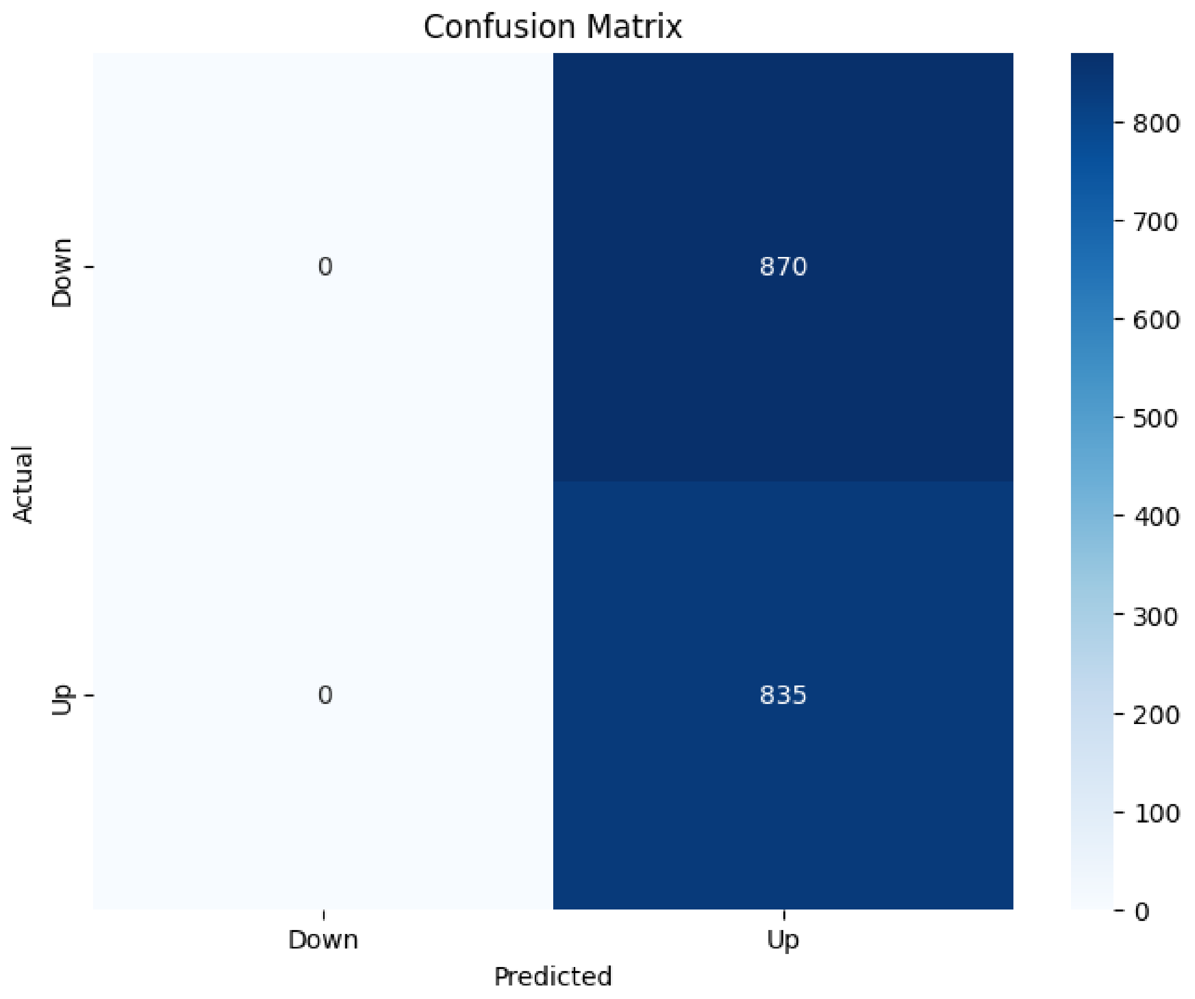
```
[15] accuracy = accuracy_score(y_test, y_pred)
     print(f'Accuracy: {accuracy:.2f}')

     Accuracy: 0.49
```

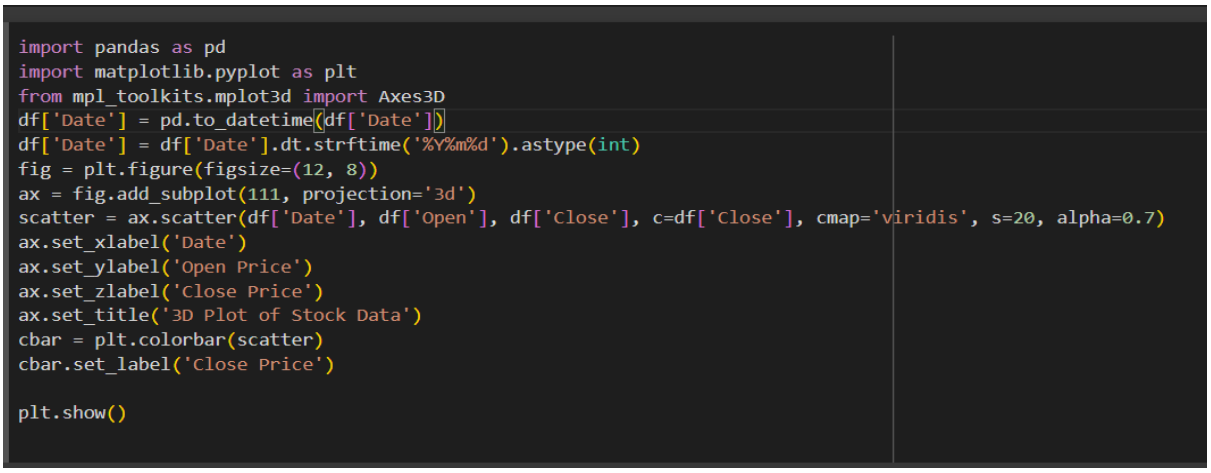
```
[16]
     print('Classification Report:')
     print(classification_report(y_test, y_pred))

     Classification Report:
                   precision    recall  f1-score   support

                0       0.00      0.00      0.00       870
                1       0.49      1.00      0.66       835

         accuracy                           0.49      1705
        macro avg       0.24      0.50      0.33      1705
     weighted avg       0.24      0.49      0.32      1705
```

Visualize the confusion matrix

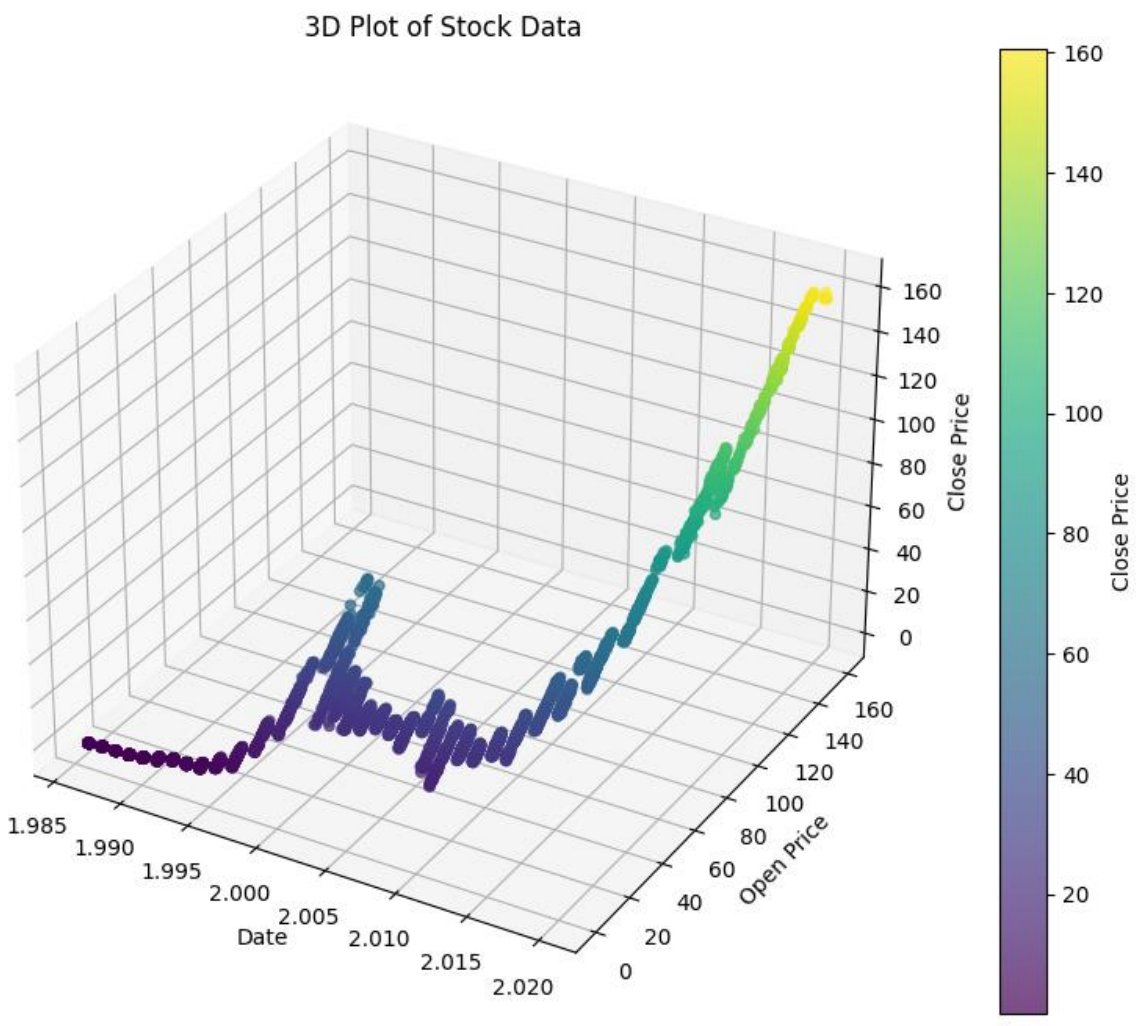


```
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Down', 'Up'], yticklabels=['Down', 'Up'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

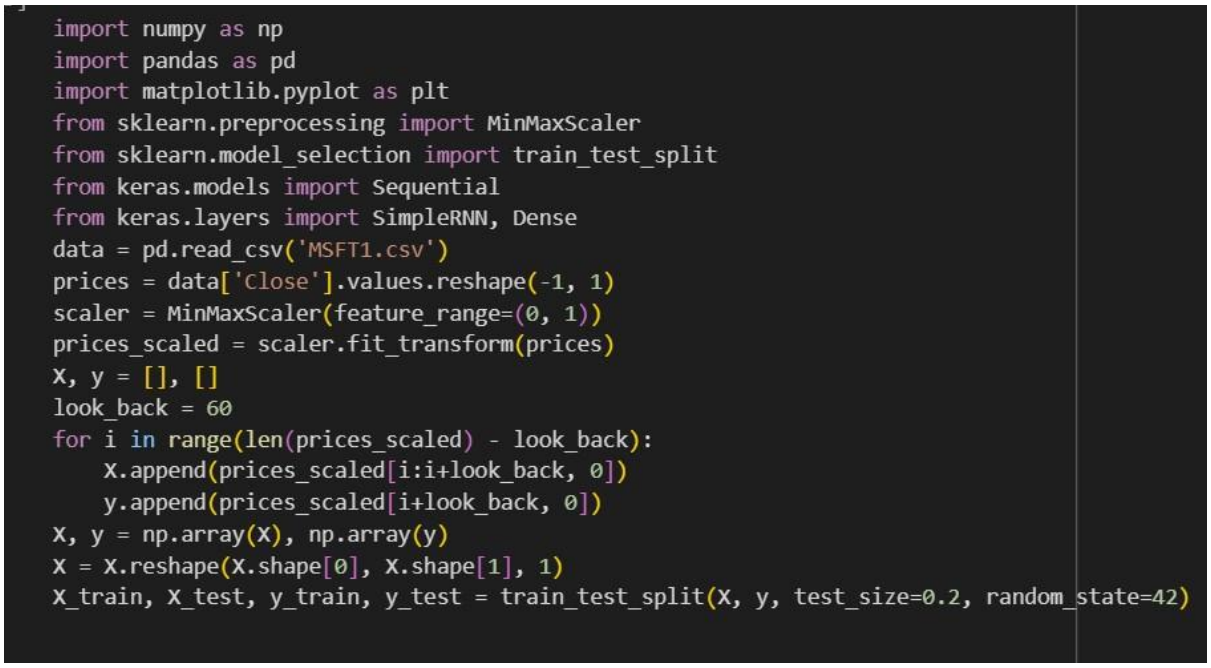




```python
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
df['Date'] = pd.to_datetime(df['Date'])
df['Date'] = df['Date'].dt.strftime('%Y%m%d').astype(int)
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(df['Date'], df['Open'], df['Close'], c=df['Close'], cmap='viridis', s=20, alpha=0.7)
ax.set_xlabel('Date')
ax.set_ylabel('Open Price')
ax.set_zlabel('Close Price')
ax.set_title('3D Plot of Stock Data')
cbar = plt.colorbar(scatter)
cbar.set_label('Close Price')

plt.show()
```

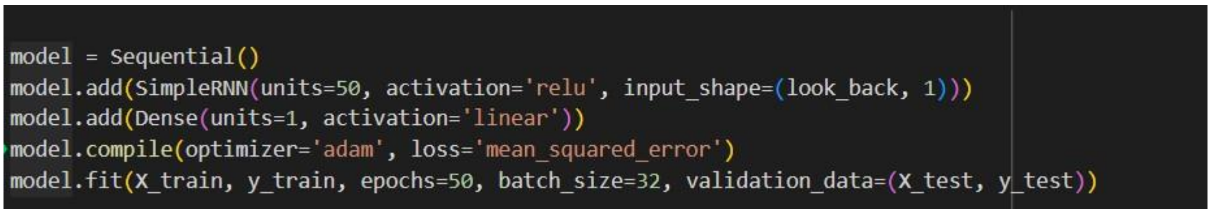


7. RNN



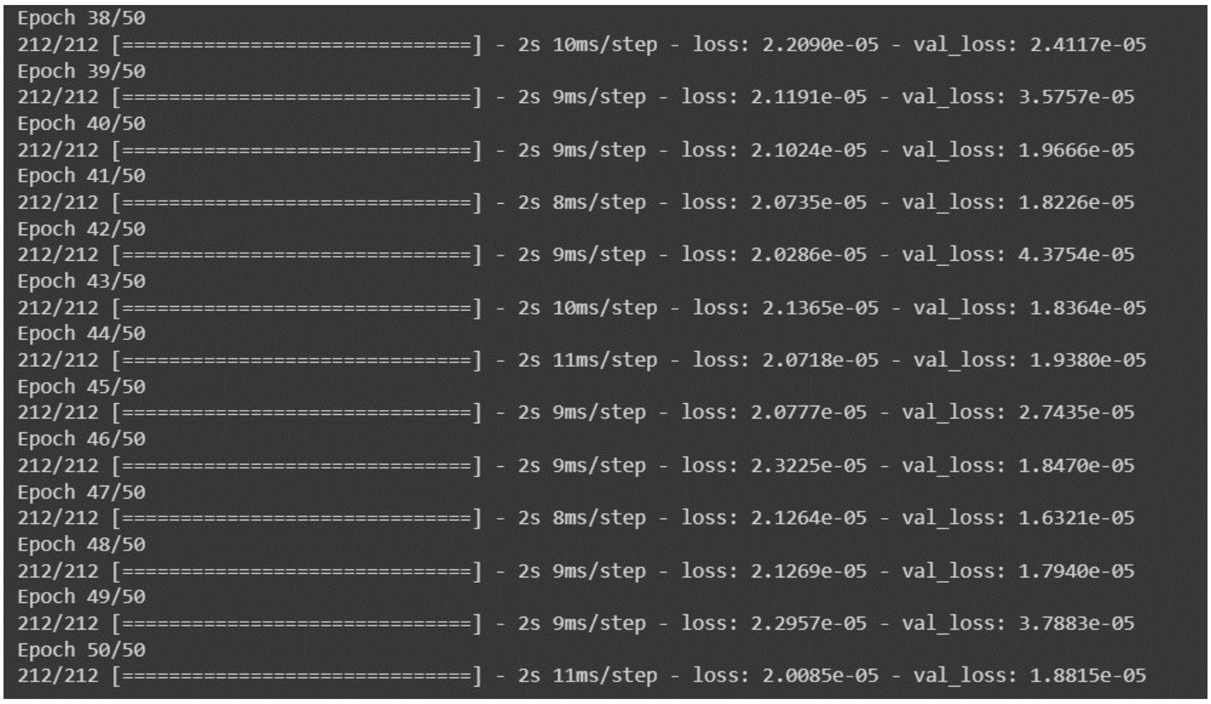
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import SimpleRNN, Dense
data = pd.read_csv('MSFT1.csv')
prices = data['Close'].values.reshape(-1, 1)
scaler = MinMaxScaler(feature_range=(0, 1))
prices_scaled = scaler.fit_transform(prices)
X, y = [], []
look_back = 60
for i in range(len(prices_scaled) - look_back):
    X.append(prices_scaled[i:i+look_back, 0])
    y.append(prices_scaled[i+look_back, 0])
X, y = np.array(X), np.array(y)
X = X.reshape(X.shape[0], X.shape[1], 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

BUILD THE RNN MODEL:



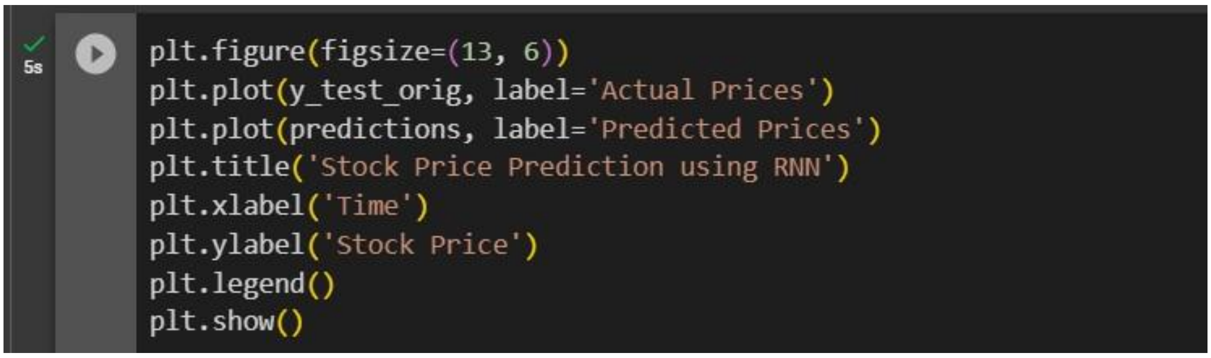
```
model = Sequential()
model.add(SimpleRNN(units=50, activation='relu', input_shape=(look_back, 1)))
model.add(Dense(units=1, activation='linear'))
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))
```

Find Epoch



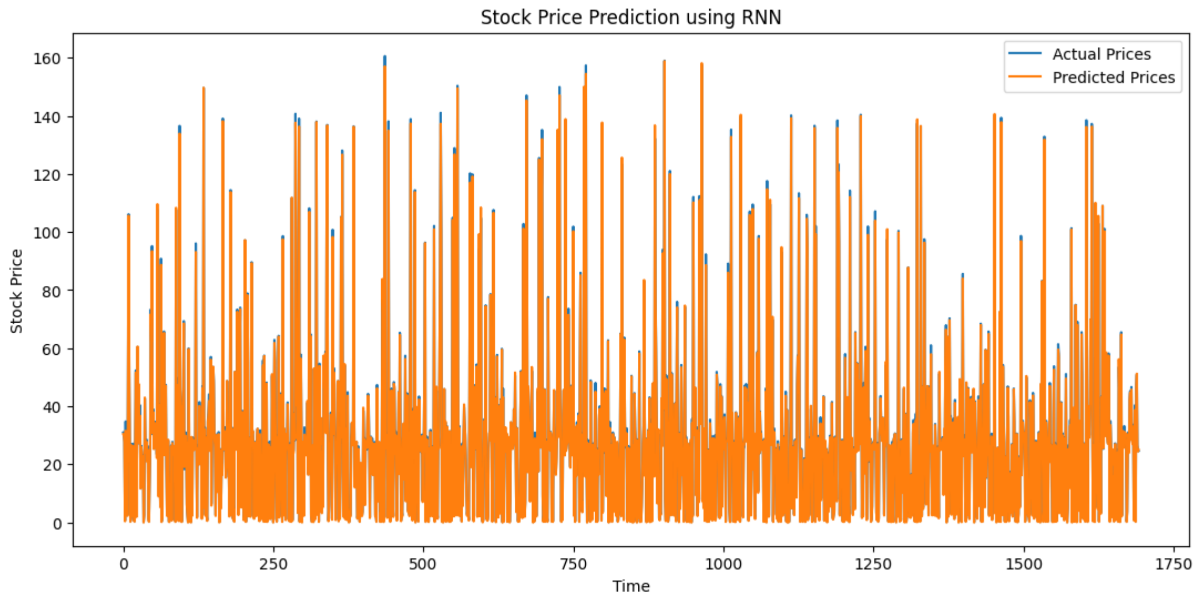
```
Epoch 38/50
212/212 [==============================] - 2s 10ms/step - loss: 2.2090e-05 - val_loss: 2.4117e-05
Epoch 39/50
212/212 [==============================] - 2s 9ms/step - loss: 2.1191e-05 - val_loss: 3.5757e-05
Epoch 40/50
212/212 [==============================] - 2s 9ms/step - loss: 2.1024e-05 - val_loss: 1.9666e-05
Epoch 41/50
212/212 [==============================] - 2s 8ms/step - loss: 2.0735e-05 - val_loss: 1.8226e-05
Epoch 42/50
212/212 [==============================] - 2s 9ms/step - loss: 2.0286e-05 - val_loss: 4.3754e-05
Epoch 43/50
212/212 [==============================] - 2s 10ms/step - loss: 2.1365e-05 - val_loss: 1.8364e-05
Epoch 44/50
212/212 [==============================] - 2s 11ms/step - loss: 2.0718e-05 - val_loss: 1.9380e-05
Epoch 45/50
212/212 [==============================] - 2s 9ms/step - loss: 2.0777e-05 - val_loss: 2.7435e-05
Epoch 46/50
212/212 [==============================] - 2s 9ms/step - loss: 2.3225e-05 - val_loss: 1.8470e-05
Epoch 47/50
212/212 [==============================] - 2s 8ms/step - loss: 2.1264e-05 - val_loss: 1.6321e-05
Epoch 48/50
212/212 [==============================] - 2s 9ms/step - loss: 2.1269e-05 - val_loss: 1.7940e-05
Epoch 49/50
212/212 [==============================] - 2s 9ms/step - loss: 2.2957e-05 - val_loss: 3.7883e-05
Epoch 50/50
212/212 [==============================] - 2s 11ms/step - loss: 2.0085e-05 - val_loss: 1.8815e-05
```

Visualize the result



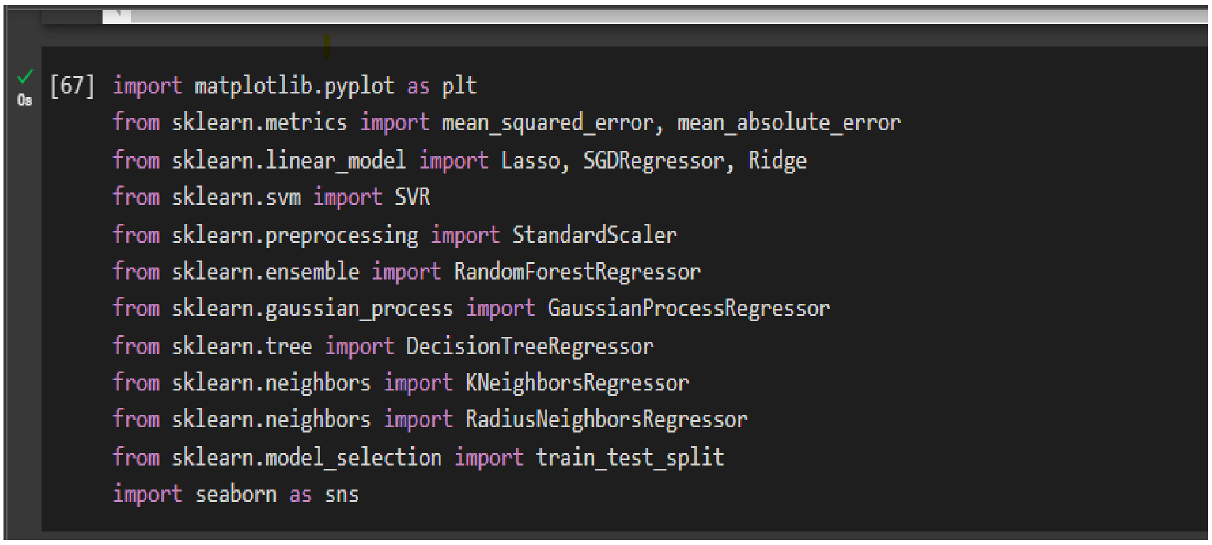
```
5s
plt.figure(figsize=(13, 6))
plt.plot(y_test_orig, label='Actual Prices')
plt.plot(predictions, label='Predicted Prices')
plt.title('Stock Price Prediction using RNN')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

O/P:


Stock Price Prediction using RNN
Actual Prices
Predicted Prices
Stock Price
Time
0
20
40
60
80
100
120
140
160
0
250
500
750
1000
1250
1500
1750

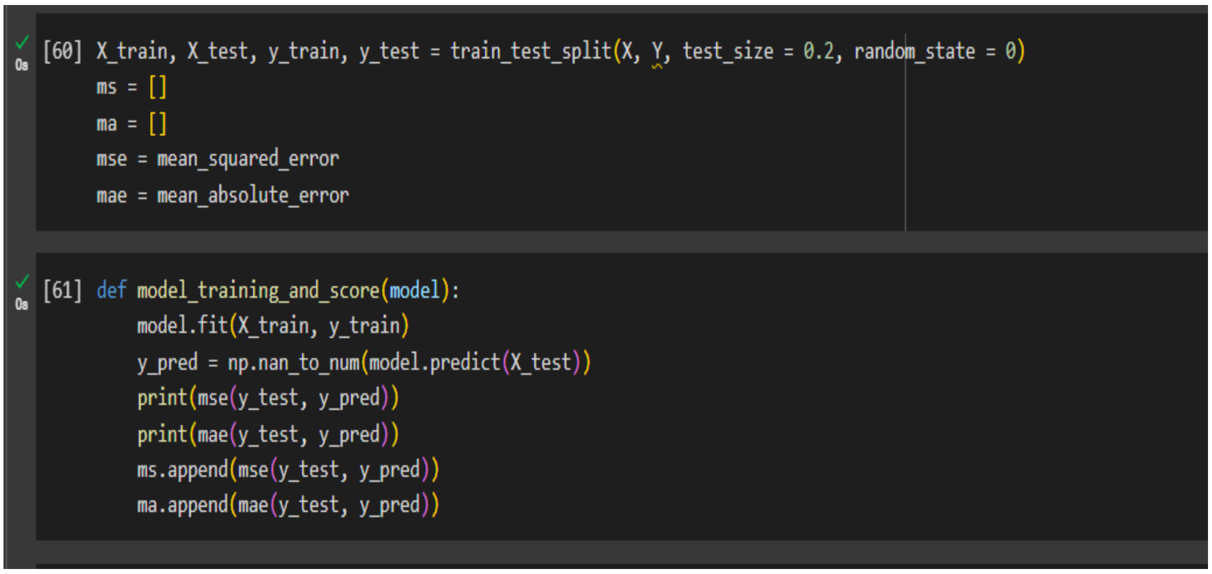
Some Common Techniques :

Import libraries



```
[67] import matplotlib.pyplot as plt
     from sklearn.metrics import mean_squared_error, mean_absolute_error
     from sklearn.linear_model import Lasso, SGDRegressor, Ridge
     from sklearn.svm import SVR
     from sklearn.preprocessing import StandardScaler
     from sklearn.ensemble import RandomForestRegressor
     from sklearn.gaussian_process import GaussianProcessRegressor
     from sklearn.tree import DecisionTreeRegressor
     from sklearn.neighbors import KNeighborsRegressor
     from sklearn.neighbors import RadiusNeighborsRegressor
     from sklearn.model_selection import train_test_split
     import seaborn as sns
```

Model Training

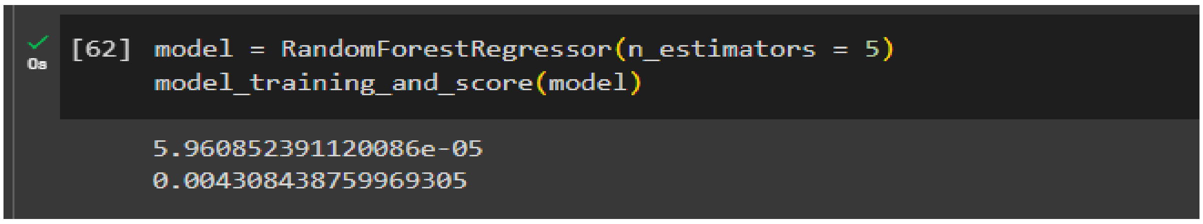


```
[60] X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)
     ms = []
     ma = []
     mse = mean_squared_error
     mae = mean_absolute_error
```

```
[61] def model_training_and_score(model):
         model.fit(X_train, y_train)
         y_pred = np.nan_to_num(model.predict(X_test))
         print(mse(y_test, y_pred))
         print(mae(y_test, y_pred))
         ms.append(mse(y_test, y_pred))
         ma.append(mae(y_test, y_pred))
```

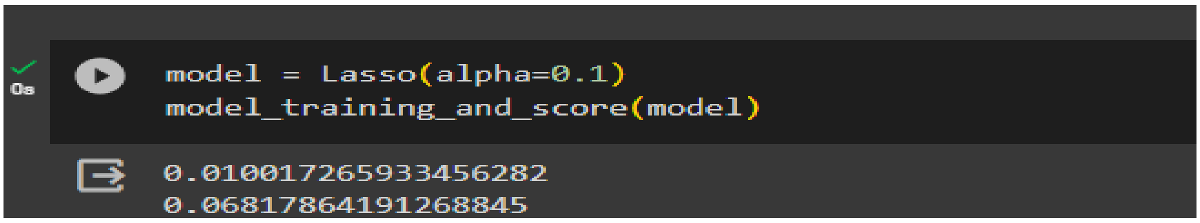
Model Evaluation:

1.Random Forest



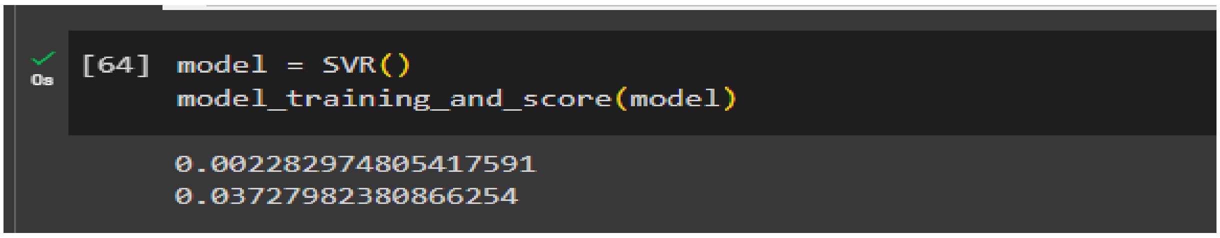
```
[62] model = RandomForestRegressor(n_estimators = 5)
     model_training_and_score(model)

     5.960852391120086e-05
     0.004308438759969305
```

2. Lasso Regression



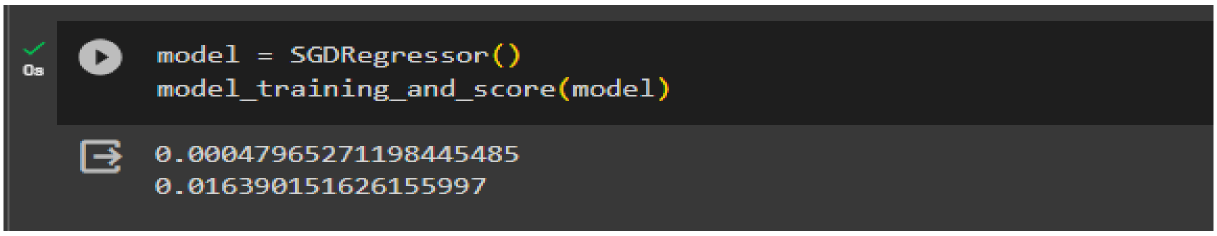
```
model = Lasso(alpha=0.1)
model_training_and_score(model)

0.010017265933456282
0.06817864191268845
```

3. Support Vector Regressor



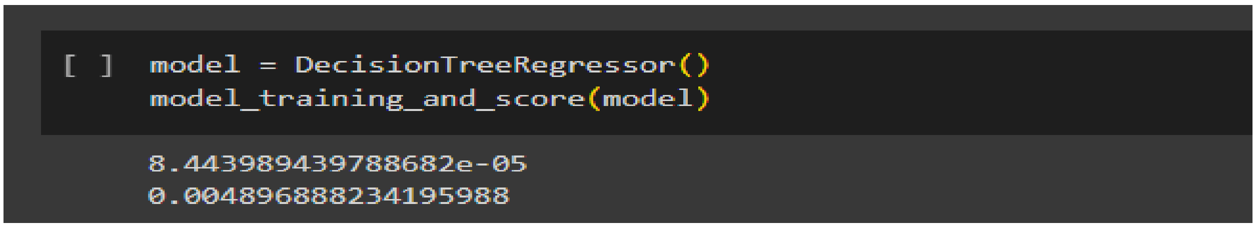
```
[64] model = SVR()
     model_training_and_score(model)

     0.0022829748054 17591
     0.03727982380866254
```

4. Stochastic Gradient Descent



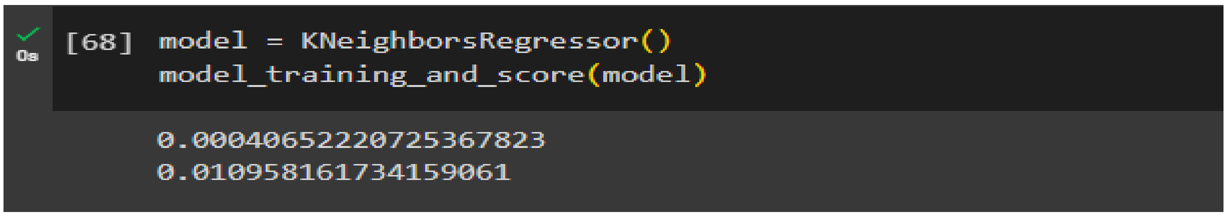
```
model = SGDRegressor()
model_training_and_score(model)

0.0004796527119844 5485
0.016390151626155997
```

5. Decision Tree Regressor



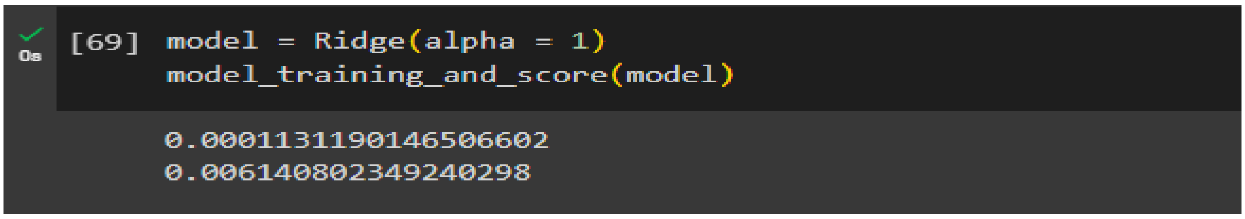
```
[ ]  model = DecisionTreeRegressor()
     model_training_and_score(model)

     8.443989439788682e-05
     0.004896888234195988
```

6. K Neighbors Regressor



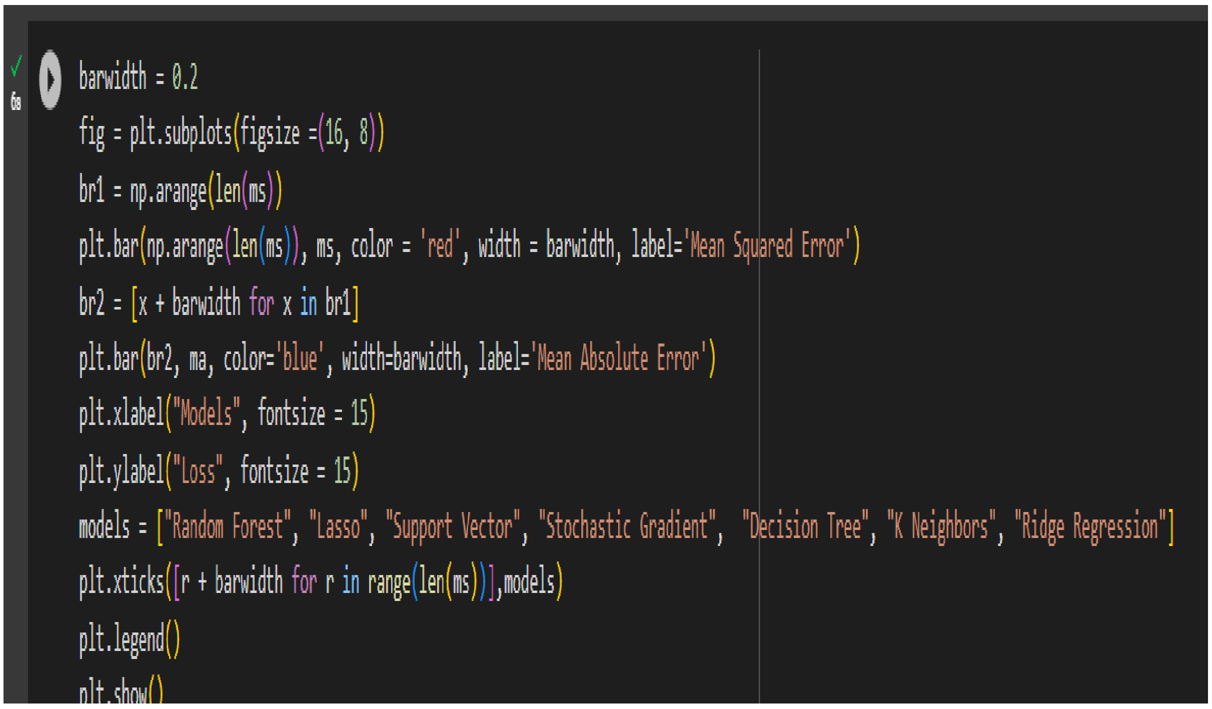
```
0s  [68]  model = KNeighborsRegressor()
          model_training_and_score(model)

          0.00040652220725367823
          0.010958161734159061
```

7. Ridge Regression

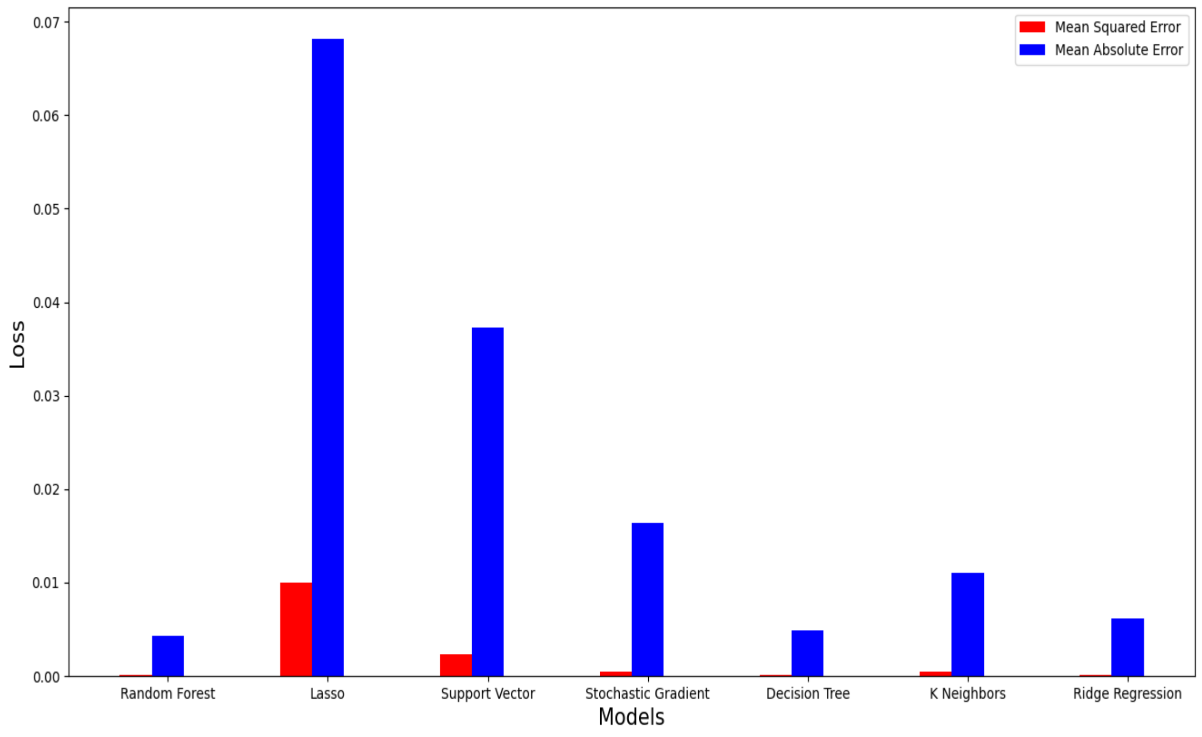


```
0s  [69]  model = Ridge(alpha = 1)
          model_training_and_score(model)

          0.00011311901465066 02
          0.006140802349240298
```

Plotting Losses of different models



```
barwidth = 0.2
fig = plt.subplots(figsize =(16, 8))
br1 = np.arange(len(ms))
plt.bar(np.arange(len(ms)), ms, color = 'red', width = barwidth, label='Mean Squared Error')
br2 = [x + barwidth for x in br1]
plt.bar(br2, ma, color='blue', width=barwidth, label='Mean Absolute Error')
plt.xlabel("Models", fontsize = 15)
plt.ylabel("Loss", fontsize = 15)
models = ["Random Forest", "Lasso", "Support Vector", "Stochastic Gradient",  "Decision Tree", "K Neighbors", "Ridge Regression"]
plt.xticks([r + barwidth for r in range(len(ms))],models)
plt.legend()
plt.show()
```



ADVANTAGES OF STOCK PRICE PREDICTION

Here are some potential advantages of stock price prediction:

1. **Informed Decision-Making:**

   Stock price predictions provide investors and traders with information to make more informed decisions regarding buying, holding, or selling stocks.

2. **Risk Management:**

   Predictions can help assess and manage risks associated with stock investments, allowing for the development of strategies to mitigate potential losses.

3. **Timing Opportunities:**

   Predictions can assist in timing investments by identifying potential entry and exit points, optimizing returns.

4. **Portfolio Diversification:**

   Predictions can guide the diversification of investment portfolios, reducing exposure to single stocks or sectors and spreading risk.

5. **Profit Potential:**

   Accurate predictions can lead to profitable trading or investment opportunities, enabling individuals to capitalize on market movements.

6. **Long-Term Planning:**

   Stock price predictions can be valuable for long-term financial planning, such as retirement or education savings, by helping individuals make informed investment choices.

7.Quantitative Analysis:

Predictive models allow for quantitative analysis of factors influencing stock prices, providing a structured and data-driven approach to investment.

8.Hedging Strategies:

Predictions can inform hedging strategies, allowing investors to protect their investments from adverse market movements.

9. Algorithmic Trading:

Predictions can be used to develop algorithmic trading strategies, automating trades based on prediction signals.

10. Back testing and Strategy Evaluation:

Predictive models can be back tested to evaluate the historical performance of trading or investment strategies, helping investors refine and optimize their approaches.

11.Research and Analysis:

Stock price prediction research contributes to a better understanding of market dynamics and factors influencing stock prices.

12. Real-Time Monitoring:

With real-time data and analytics, investors can continuously monitor and adapt their strategies in response to changing market conditions.

13. Reduced Emotional Bias:

Data-driven predictions can help reduce emotional decision-making, leading to more rational and objective investment choices.

14. Asset Allocation:

Predictions can inform decisions about the allocation of assets within a portfolio, optimizing the balance between different types of investments.

15. Machine Learning and AI:

Advanced techniques like machine learning and artificial intelligence can improve prediction accuracy and adapt to changing market conditions.

16. Sector and Industry Insights:

Stock price predictions can provide insights into the performance of specific sectors or industries, aiding in sector-specific investment decisions.

DISADVANTAGES OF STOCK PRICE PREDICTION

Stock price prediction, while valuable, comes with its share of disadvantages and challenges. Some of the disadvantages of stock price prediction include:

1. Uncertainty:

Stock markets are influenced by numerous unpredictable factors, making it difficult to create accurate predictions. Even the most sophisticated models can't eliminate the inherent uncertainty of the market.

2. Market Noise:

Stock prices can be affected by noise in the market, such as rumors, speculative trading, or temporary shocks, which can lead to unpredictable short-term fluctuations.

3.Overfitting:

Complex prediction models may overfit historical data, meaning they perform well in the past but fail to generalize to new data. This can lead to poor future performance.

4. Data Quality:

Accurate predictions depend on high-quality, reliable data. Inaccurate or incomplete data can lead to incorrect predictions.

5. Data Lag:

Market data is not always available in real-time, leading to delays in decision-making and potentially missing out on opportunities.

6. Model Assumptions:

Many prediction models rely on certain assumptions about market behavior, which may not always hold true in changing market conditions.

7. Market Dynamics:

Stock markets can exhibit non-linear and chaotic behavior, making predictions challenging, especially during extreme market events.

8. Black Swan Events:

Unpredictable, rare events, such as financial crises or natural disasters, can have a significant impact on stock prices and are challenging to predict.

9. Emotional Factors:

10. Human emotions and market sentiment can have a profound impact on stock prices, and these factors are often difficult to quantify and predict.

10. Regulatory Changes:

Changes in financial regulations or government policies can impact stock prices and are often difficult to anticipate.

11. Short-Term vs. Long-Term Predictions:

While some models may excel at short-term predictions, they may not be suitable for long-term forecasting, and vice versa.

12. High-Frequency Trading:

High-frequency traders can execute thousands of trades per second, making it challenging for traditional prediction models to keep up.

13. Market Manipulation:

Unethical practices or market manipulation can distort stock prices and render predictions less reliable.

14. Lack of Information:

Not all information relevant to stock prices is publicly available, and proprietary data sources may not be accessible.

15. Technological Challenges:

Real-time prediction requires robust technology infrastructure, low-latency data feeds, and advanced hardware, which can be costly and complex to implement.

16. Model Complexity:

Sophisticated models can be difficult to understand and interpret, making it challenging for investors and traders to have confidence in their predictions.

17. Past Performance vs. Future Results:

Past stock performance does not guarantee future results, so relying solely on historical data can be misleading.

18. Psychological Stress:

Frequent monitoring of stock price predictions and trading can lead to psychological stress and anxiety.

It's important to recognize these disadvantages when using stock price predictions and to approach them with a balanced perspective, taking into consideration the inherent risks and uncertainties in the stock market. Diversification, risk management, and ongoing evaluation of prediction models are essential for mitigating these disadvantages.

CONCLUSION:

By measuring the accuracy of the different algorithms, we found that the most suitable algorithm for predicting the market price of a stock based on various data points from the historical data. The algorithm will be a great asset for brokers and investors for investing money in the stock market since it is trained on a huge collection of historical data and has been chosen after being tested on a sample data. The project demonstrates the machine learning model to predict the stock value with more accuracy as compared to previously implemented machine learning models.

Future scope of this project will involve adding more parameters and factors like the financial ratios, multiple instances, etc. The more the parameters are considered more will be the accuracy. The algorithms can also be applied for analysing the contents of public comments and thus determine patterns/relationships between the customer and the corporate employee.

In conclusion, while data science techniques can provide valuable insights into stock price movements, the inherent complexity and uncertainty of financial markets mean that predictions should be used as one of several tools in the decision-making process. Additionally, ethical considerations, transparency, and a deep understanding of financial markets are essential when applying data

science to stock price prediction.