STOCK PRICE PREDICTION

MODEL IMPLEMENTATION – PHASE 2

Submitted by:

Adhithya Sree Mohan	AM.EN. U4CSE20002
Akhilesh Rajesh	AM.EN. U4CSE20005
Tarun Raveesh	AM.EN. U4CSE20016
Golla Ajay Kumar	AM.EN. U4CSE20026
Karuturi Paavani	AM.EN. U4CSE20036

ABSTRACT

Informal description

The fluctuations in the stock market are erratic as it depends on supply versus demand. The goal is to employ machine learning to create a program to predict the future value of a stock based on current market indices by training on the previous patterns.

Formal description

• **Task (T):** To produce a forecast on the fluctuation in the economic value of stocks of a company.

• **Experience (E):** Aggregation of data on the prior performance of the particular stock which includes the following: book value, market capitalization, change of stock net price over the one month, margins, dividend yield, sales revenue turnover, etc.

• **Performance (P):** Prediction accuracy: the no stocks whose values are forecasted precisely (prediction rate) out of the total no. of stocks taken into consideration.

Assumptions

We presume an absence of occurrence of any volatile(unprecedented political) events that might cause an abrupt change.

1. INTRODUCTION

Motivation:

There's a podcast by the CEO of CRED, Kunal Shah, in which he anxiously states that merely 40 Million people of the 1.38 Billion population of our country invest in the Stock Market (i.e., 3% of the total population which is far behind countries like the US). So basically, the people of our own country do not invest in it due to various obtuse reasons. Hence, we decided to make a tool to predict stock prices using Machine Learning.

Machine Learning has various important applications in stock price prediction. We will be talking about predicting the returns on various stocks. Stock Price Prediction is considered one of the most complex tasks with many uncertainties as just the prediction can lead to a big amount of profits for the seller and the broker. Machine Learning is considered to be an efficient way to represent such processes as it predicts the market value close to the tangible value, hence increasing the accuracy. Two parts in which we will develop our project are:

1. We will learn how to predict Stock prices using the LSTM neural network.
2. Next, We will build a dashboard using Plotly dash for the Stock Analysis.

Benefits of Solution:

1. <u>Removes The Investment Bias</u>

When various investors plan their investments, it is not easy to follow behavioral bias as an investor. You often fall into the trap of choosing your favorite stocks instead of choosing a stock that has the potential for giving you better outcomes based on your analysis. If there we can use Machine Learning for Stock Prediction, you can get rid of this bias as it makes sure that you take decisions analytically instead of going on your gut feelings or investing in your preferred stocks.

2. <u>Minimizes Your Losses:</u>

Another advantage of Stock price prediction is that it minimizes your losses to a great extent. Before knowing how to predict, the investors often make mistakes of not doing their homework properly which means they often make the mistake of not using the correct prediction method.

3. Assures Consistency

One of the best advantages of Stock price prediction is the consistency you can achieve in the results. Since we all know that the stock market is highly volatile, there is no guarantee that even after making the prediction call using various strategies, you are going to be on the right path of trade for profits.

4. Gives a better idea about entry and exit points

Applying the correct stock price prediction methods helps you know better about your entry and exit points. So often the traders either enter or exit the market at inappropriate times, which means that they have failed to capitalize on the full potential of making profits.

Solution Use:

Due to a lack of awareness of the correct methods of prediction of Stock Prices, the Youth of our country as well as most of the population of our country are not that interested to invest in the stock market. So, having Stock Market Prediction with a good amount of efficiency can prove to be very helpful and can create more awareness among the population of our country.

2. DATASET FINALIZATION

1. Apple Stock Price Prediction

About:

This is a Dataset for Stock Prediction on Apple Inc.
This dataset starts from 1980 to 2021. It was collected from Yahoo Finance. You can perform Time Series Analysis and EDA on data.

Features:

There Are 7 Features in this dataset, and they are:

1. Date: Date
2. Open: It is the price at which the financial security opens in the market when trading begins.
3. High: The high is the highest price at which a stock is traded during a period.
4. Low: The low is the lowest price at which a stock is traded during a period.
5. Close: Closing price generally refers to the last price at which a stock trades during a regular trading session.
6. Adj Close: The adjusted closing price amends a stock’s closing price to reflect that stock’s value after accounting.
7. Volume: Volume measures the number of shares traded in a stock or contracts traded in futures or options.

Applications:

The Apple Dataset applications are extensively used in the fields of

1. Business
2. Investing
3. Electronics
4. Exploratory Data Analysis
5. Time Series Analysis
6. Statistical Analysis

2. Google Stock Price Prediction

About:

The art of forecasting stock prices has been a difficult task for many researchers and analysts. Investors are highly interested in the research area of stock price prediction. For a good and successful investment, many investors are keen on knowing the future situation of the stock market. Good and effective prediction systems for the stock market help traders, investors, and analysts by providing supportive information on the future direction of the stock market. In this work, we present a recurrent neural network (RNN) and Long Short-Term Memory (LSTM) approach to predict stock market indices.

Features:

There Are 6 Features in this dataset, and they are:

1. Date: Date
2. Open: It is the price at which the financial security opens in the market when trading begins.
3. High: The high is the highest price at which a stock is traded during a period.
4. Low: The low is the lowest price at which a stock is traded during a period.
5. Close: Closing price generally refers to the last price at which a stock trades during a regular trading session.
6. Volume: Volume measures the number of shares traded in a stock or contracts traded in futures or options.

Applications:

The Apple Dataset applications are extensively used in the fields of

1. Deep Learning
2. RNN

3. <u>Netflix Stock Price Prediction</u>

The Dataset contains data for 5 years i.e., from 5th Feb 2018 to 5th Feb 2022
The art of forecasting stock prices has been a difficult task for many researchers and analysts. Investors are highly interested in the research area of stock price prediction. For a good and successful investment, many investors are keen on knowing the future situation of the stock market. Good and effective prediction systems for the stock market help traders, investors, and analysts by providing supportive information on the future direction of the stock market.

Features:

There Are 7 Features in this dataset, and they are:

1. Date: Everyday price
2. Open: Price at which stock opened
3. High: Today's High
4. Low: Today's Low
5. Close: Close price adjusted for splits
6. Adj Close: Adjusted close price adjusted for splits and dividend and/or capital gain distributions.
7. Volume: Volume of stocks

Applications:

1. Business
2. Investing
3. Intermediate
4. Time Series Analysis
5. Python
6. LSTM

3. PYTHON PACKAGES USED

- **matplotlib.pyplot**
  Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

- **NumPy**
  NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

- **Pandas**
  Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. Pandas allows us to analyze big data and make conclusions based on statistical theories.

- **Seaborn**
  Seaborn is an open source, BSD-licensed Python library providing high level API for visualizing the data using Python programming language.

- **sklearn.model_selection**
  Split arrays or matrices into random train and test subsets. Quick utility that wraps input validation, `next(ShuffleSplit().split(X, y))`, and application to input data into a single call for splitting (and optionally subsampling) data into a one-liner.

- **sklearn.preprocessing**
  The `sklearn.preprocessing` package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.

- **sklearn.linear_model**
  Linear Regression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

- **sklearn.neighbors**
  sklearn.neighbors provides functionality for unsupervised and supervised neighbors-based learning methods. Unsupervised nearest neighbors is the foundation of many other learning methods, notably manifold learning and spectral clustering. Supervised neighbors-based learning comes in two flavors: classification for data with discrete labels, and regression for data with continuous labels.

- **sklearn.svm**
  Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

- **Sklearn.metrics**
  The sklearn.metrics implements several losses, scores and utility functions to measure classification performance. Some metrics might require probability estimates of the positive class, confidence values or binary decisions values.

1. APPLE DATASET

IMPORTING LIBRARIES

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import warnings
import string
import csv
import re

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from pandas.plotting import scatter_matrix
from collections import Counter
from sklearn import metrics

warnings.filterwarnings("ignore")
```

[1]

EXTRACTING DATA - Reading the CSV File:

```
Data = pd.read_csv("Apple.csv")
print(Data)
```

[2]

```
...
             Date        Open        High         Low       Close   Adj Close  \
0      1980-12-12    0.128348    0.128906    0.128348    0.128348    0.100178
1      1980-12-15    0.122210    0.122210    0.121652    0.121652    0.094952
2      1980-12-16    0.113281    0.113281    0.112723    0.112723    0.087983
3      1980-12-17    0.115513    0.116071    0.115513    0.115513    0.090160
4      1980-12-18    0.118862    0.119420    0.118862    0.118862    0.092774
...           ...         ...         ...         ...         ...         ...
10463  2022-06-13  132.869995  135.199997  131.440002  131.880005  131.880005
10464  2022-06-14  133.130005  133.889999  131.479996  132.759995  132.759995
10465  2022-06-15  134.289993  137.339996  132.160004  135.429993  135.429993
10466  2022-06-16  132.080002  132.389999  129.039993  130.059998  130.059998
10467  2022-06-17  130.070007  133.080002  129.809998  131.559998  131.559998

          Volume
0      469033600
1      175884800
2      105728000
3       86441600
4       73449600
...          ...
10463  122207100
10464   84784300
10465   91533000
10466  108123900
10467  134118500

[10468 rows x 7 columns]
```

DATA PREPROCESSING

```
# Finding size of dataset (i.e number of rows & columns of the dataset.)
print("Rows: ",Data.shape[0])
print("Columns: ",Data.shape[1])
```

[3]

```
Rows:  10468
Columns:  7
```

```
Data.head() # head() function by default showcases first five rows
```

[4]

	Date	Open	High	Low	Close	Adj Close	Volume
0	1980-12-12	0.128348	0.128906	0.128348	0.128348	0.100178	469033600
1	1980-12-15	0.122210	0.122210	0.121652	0.121652	0.094952	175884800
2	1980-12-16	0.113281	0.113281	0.112723	0.112723	0.087983	105728000
3	1980-12-17	0.115513	0.116071	0.115513	0.115513	0.090160	86441600
4	1980-12-18	0.118862	0.119420	0.118862	0.118862	0.092774	73449600

```
Data.shape
```

[5]

```
(10468, 7)
```

```
Data.describe()
```

[6]

	Open	High	Low	Close	Adj Close	Volume
count	10468.000000	10468.000000	10468.000000	10468.000000	10468.000000	1.046800e+04
mean	14.757987	14.921491	14.594484	14.763533	14.130431	3.308489e+08
std	31.914174	32.289158	31.543959	31.929489	31.637275	3.388418e+08
min	0.049665	0.049665	0.049107	0.049107	0.038329	0.000000e+00
25%	0.283482	0.289286	0.276786	0.283482	0.235462	1.237768e+08
50%	0.474107	0.482768	0.465960	0.475446	0.392373	2.181592e+08
75%	14.953303	15.057143	14.692589	14.901964	12.835269	4.105794e+08
max	182.630005	182.940002	179.119995	182.009995	181.511703	7.421641e+09

Different Data Types In Datasets:

```
Data.columns
```

[7]

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

```
Data.dtypes
```

[8]

```
Date          object
Open         float64
High         float64
Low          float64
Close        float64
Adj Close    float64
Volume         int64
dtype: object
```

```
Data.info()
```

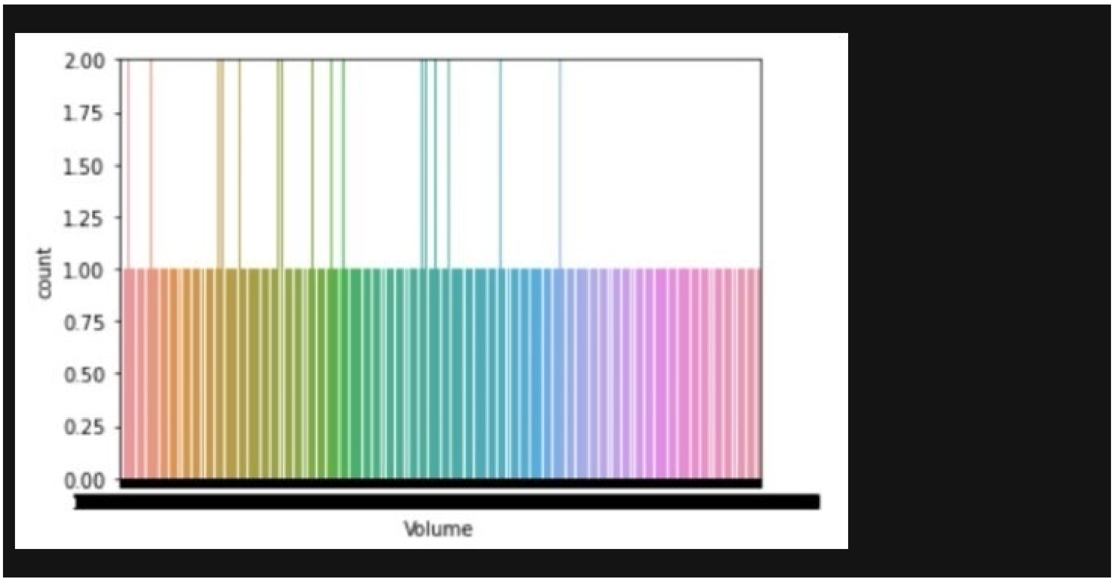
[9]

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10468 entries, 0 to 10467
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       10468 non-null  object
 1   Open       10468 non-null  float64
 2   High       10468 non-null  float64
 3   Low        10468 non-null  float64
 4   Close      10468 non-null  float64
 5   Adj Close  10468 non-null  float64
 6   Volume     10468 non-null  int64
dtypes: float64(5), int64(1), object(1)
memory usage: 572.6+ KB
```

```
# Finding Number of samples under target variable
print(f"Number of samples under target value: \n{Data['Volume'].value_counts()}")
sns.countplot(Data.Volume).set_ylim(0, 2)
plt.show()
```

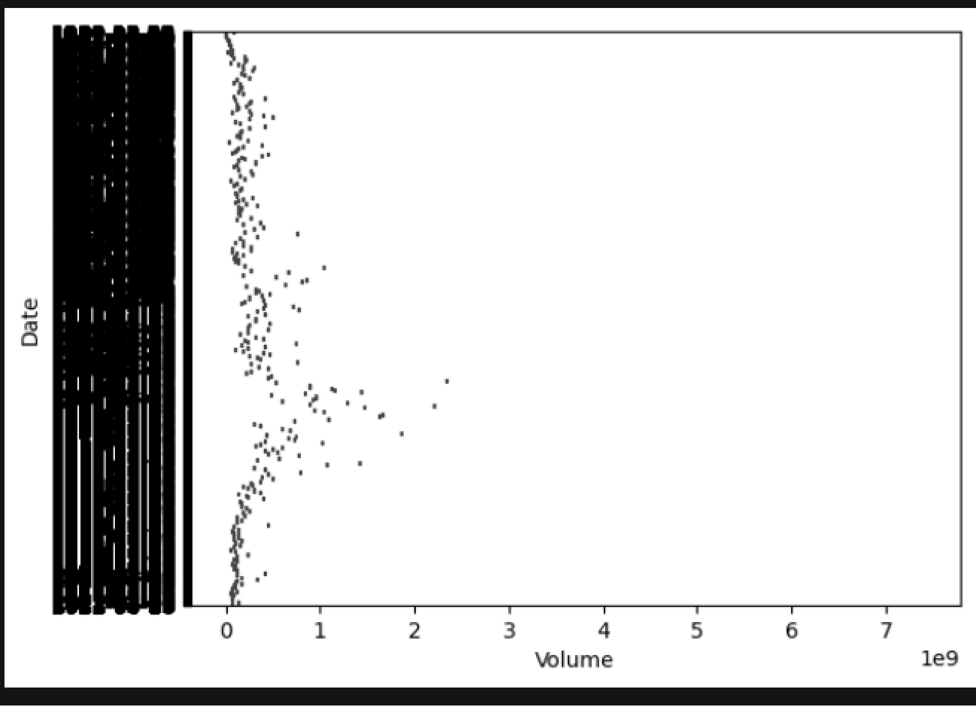
[10]

```
Number of samples under target value:
246400000    7
239680000    6
244160000    6
302400000    5
255360000    5
            ..
260915200    1
244652800    1
209171200    1
119470400    1
134118500    1
Name: Volume, Length: 9905, dtype: int64
```




```python
sns.boxplot(x = "Volume", y = "Date", data = Data)
plt.show()
```

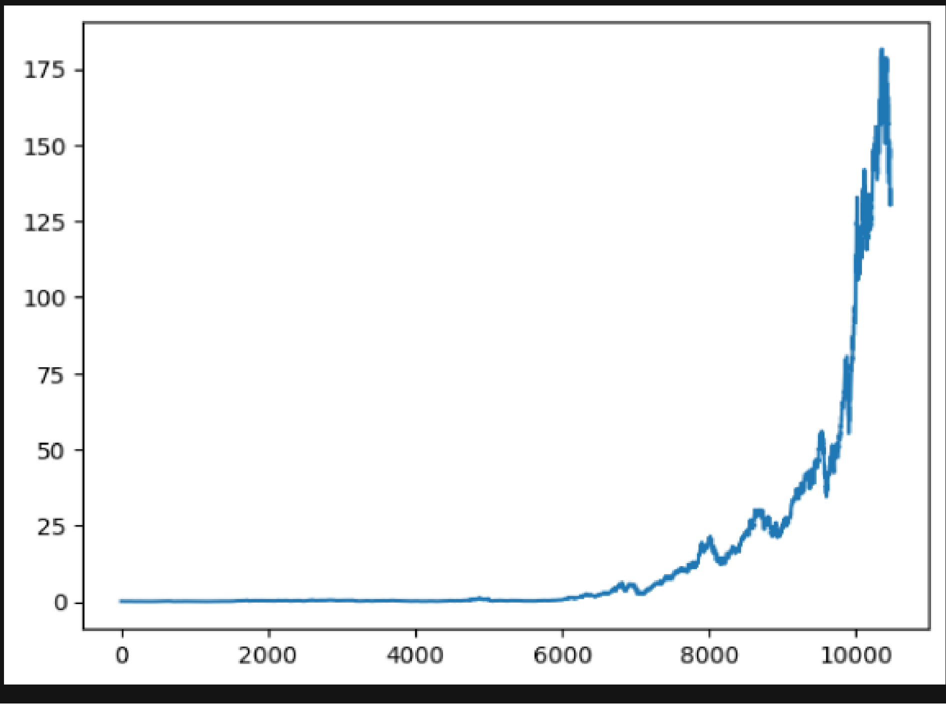
[11]



The Adjusted Close Value is the final output value that will be forecasted using the Machine Learning model.

```
[12]
# Plot the True Adj Close Value
Data['Adj Close'].plot()
```

```
<AxesSubplot: >
```



DATA CLEANING

```
[13]
Data.isnull().values.any()  # Checking whether we have any missing values in dataset
```

```
False
```

```
[14]
Data.isnull().sum()
```

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

There were no missing values in the datasets. So, there was no replacement and missing values.

DATA STANDARDIZATION

[15]
```
Data.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	1980-12-12	0.128348	0.128906	0.128348	0.128348	0.100178	469033600
1	1980-12-15	0.122210	0.122210	0.121652	0.121652	0.094952	175884800
2	1980-12-16	0.113281	0.113281	0.112723	0.112723	0.087983	105728000
3	1980-12-17	0.115513	0.116071	0.115513	0.115513	0.090160	86441600
4	1980-12-18	0.118862	0.119420	0.118862	0.118862	0.092774	73449600

[16]
```
Data['Volume'][:5]
```

```
0    469033600
1    175884800
2    105728000
3     86441600
4     73449600
Name: Volume, dtype: int64
```

[17]
```
scalar = StandardScaler(copy=True, with_mean=True, with_std=True)
Data["Volume"] = scalar.fit_transform(Data["Volume"].values.reshape(-1,1))
print ("After Standardisation: ")
Data.head()
```

After Standardisation:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1980-12-12	0.128348	0.128906	0.128348	0.128348	0.100178	0.407834
1	1980-12-15	0.122210	0.122210	0.121652	0.121652	0.094952	-0.457356
2	1980-12-16	0.113281	0.113281	0.112723	0.112723	0.087983	-0.664415
3	1980-12-17	0.115513	0.116071	0.115513	0.115513	0.090160	-0.721336
4	1980-12-18	0.118862	0.119420	0.118862	0.118862	0.092774	-0.759681

DATA NORMALIZATION

[18]
```
norm = MinMaxScaler()
Data["Volume"] = norm.fit_transform(Data["Volume"].values.reshape(-1,1))
print ("After Normalisation: ")
Data.head()
```

After Normalisation:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1980-12-12	0.128348	0.128906	0.128348	0.128348	0.100178	0.063198
1	1980-12-15	0.122210	0.122210	0.121652	0.121652	0.094952	0.023699
2	1980-12-16	0.113281	0.113281	0.112723	0.112723	0.087983	0.014246
3	1980-12-17	0.115513	0.116071	0.115513	0.115513	0.090160	0.011647
4	1980-12-18	0.118862	0.119420	0.118862	0.118862	0.092774	0.009897

Making data available for various ML models through normalization.

Discretization

```
[19]
Data['Adj Close'].unique()
```

```
array([1.00178000e-01, 9.49520000e-02, 8.79830000e-02, ...,
       1.35429993e+02, 1.30059998e+02, 1.31559998e+02])
```

```
[20]
print(Data['Adj Close'].max())
print(Data['Adj Close'].min())
```

```
181.511703
0.038329
```

```
[21]
Data['bin_of_Adj Close'] = pd.cut(Data['Adj Close'], [200,300,400,500,600,700],
                                  labels=['200-300', '300-400', '400-500', '500-600', '600-700'])
```

```
[22]
Data.groupby([Data["bin_of_Adj Close"]]).count()
```

bin_of_Adj Close	Date	Open	High	Low	Close	Adj Close	Volume
200-300	0	0	0	0	0	0	0
300-400	0	0	0	0	0	0	0
400-500	0	0	0	0	0	0	0
500-600	0	0	0	0	0	0	0
600-700	0	0	0	0	0	0	0

```
[23]
for column in Data.columns:
    print("------  " + column + "  ------")
    print(Data[column].value_counts())
```

```
Output exceeds the size limit. Open the full output data in a text editor
------  Date  ------
1980-12-12    1
2008-08-28    1
2008-08-04    1
2008-08-05    1
2008-08-06    1
```

```
           ..
1994-10-03    1
1994-10-04    1
1994-10-05    1
1994-10-06    1
2022-06-17    1
Name: Date, Length: 10468, dtype: int64
------  Open  ------
0.354911      38
0.401786      37
0.366071      36
0.397321      34
0.357143      34
              ..
3.477500       1
3.462500       1
3.563571       1
3.557143       1
130.070007     1
...
400-500    0
500-600    0
600-700    0
Name: bin_of_Adj Close, dtype: int64
```

Making the values group-wise and making continuous values discrete.

DATA SUMMARIZATION

```
[24] print(Data.shape)
```

```
(10468, 8)
```

```
[25] Data.describe()
```

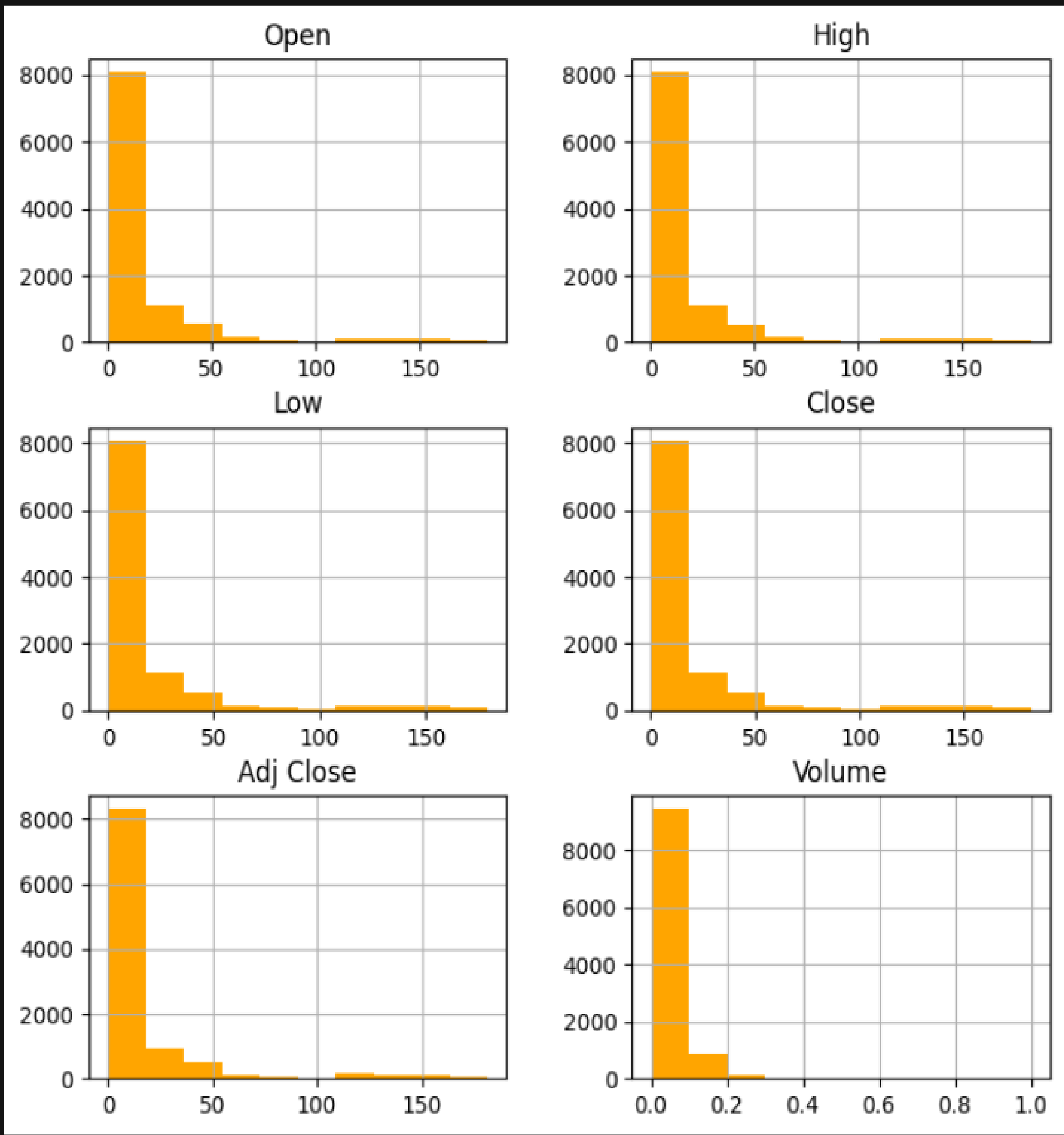
	Open	High	Low	Close	Adj Close	Volume
count	10468.000000	10468.000000	10468.000000	10468.000000	10468.000000	10468.000000
mean	14.757987	14.921491	14.594484	14.763533	14.130431	0.044579
std	31.914174	32.289158	31.543959	31.929489	31.637275	0.045656
min	0.049665	0.049665	0.049107	0.049107	0.038329	0.000000
25%	0.283482	0.289286	0.276786	0.283482	0.235462	0.016678
50%	0.474107	0.482768	0.465960	0.475446	0.392373	0.029395
75%	14.953303	15.057143	14.692589	14.901964	12.835269	0.055322
max	182.630005	182.940002	179.119995	182.009995	181.511703	1.000000

DATA VISUALIZATION

Histogram

```
Data.hist(color = "orange", figsize = (8,8))
plt.show()
```

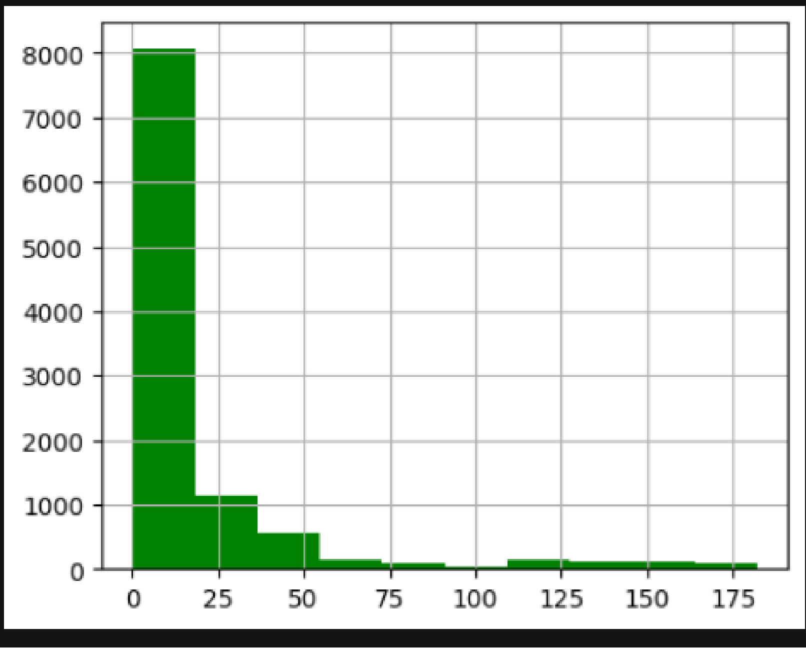
[26]



```
Data['Close'].hist(color = "green", figsize = (5,4))
```

[27]

<AxesSubplot: >

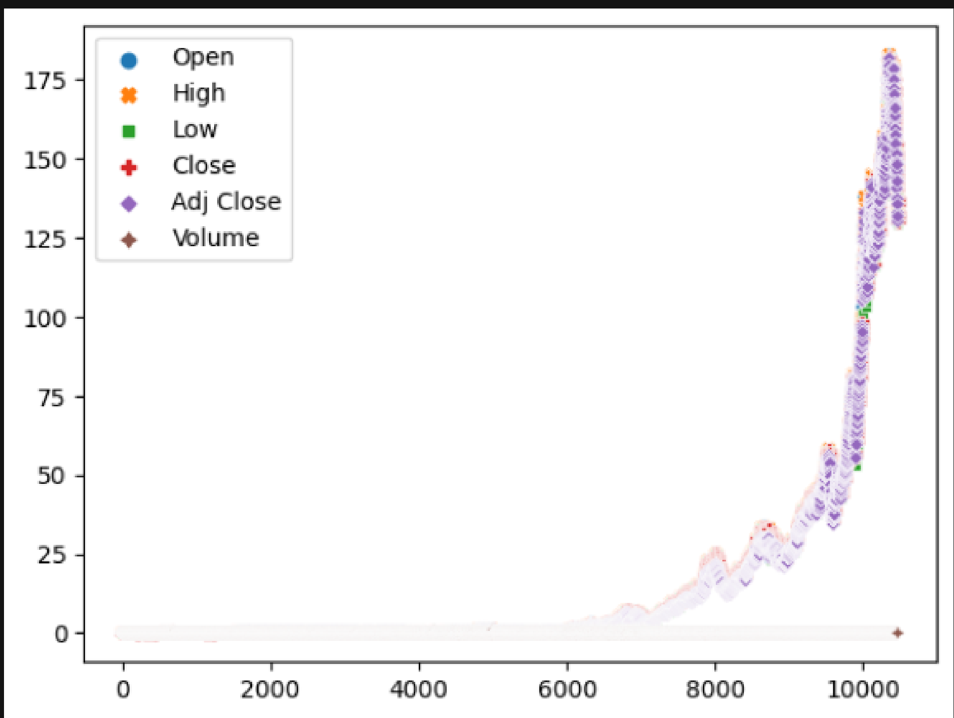


<u>Scatter Plot</u>

```
sns.scatterplot(Data)
```

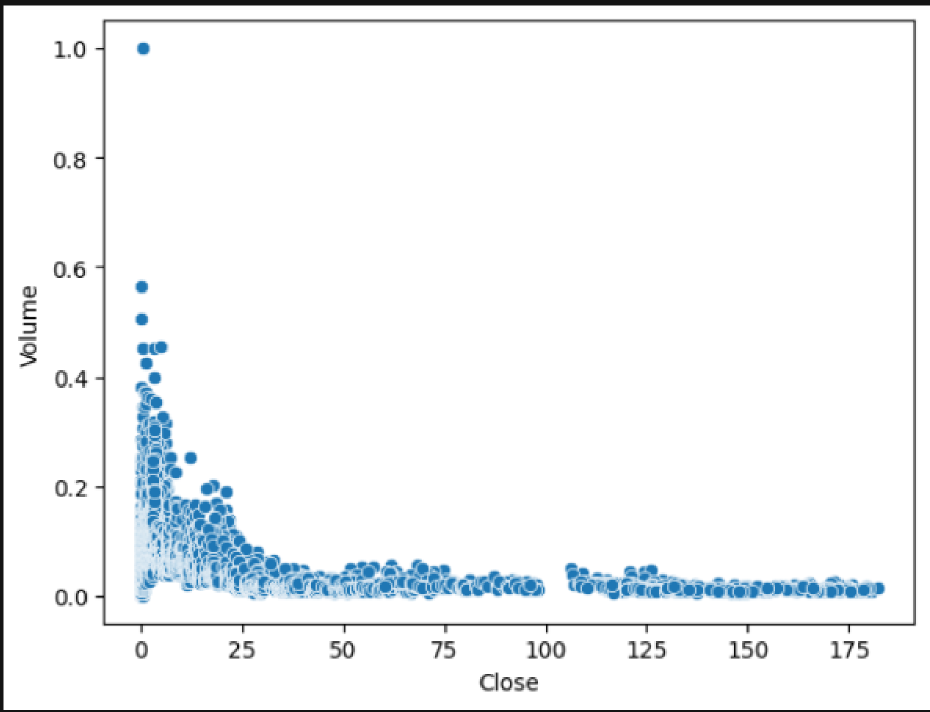
[28]

<AxesSubplot: >



```python
sns.scatterplot(x = Data['Close'], y = Data['Volume'])
```

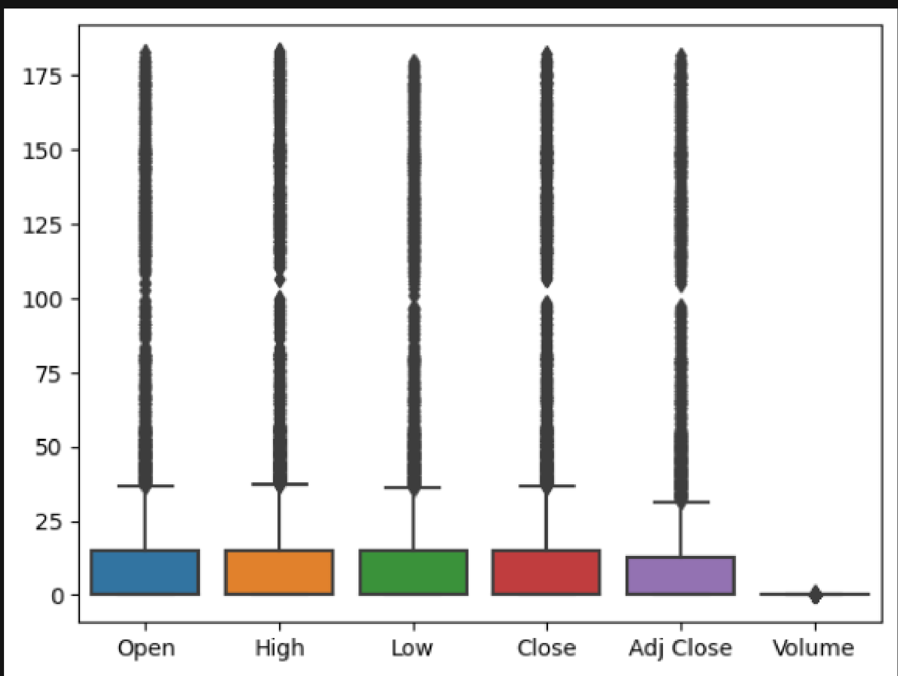
```
<AxesSubplot: xlabel='Close', ylabel='Volume'>
```



Box Plot

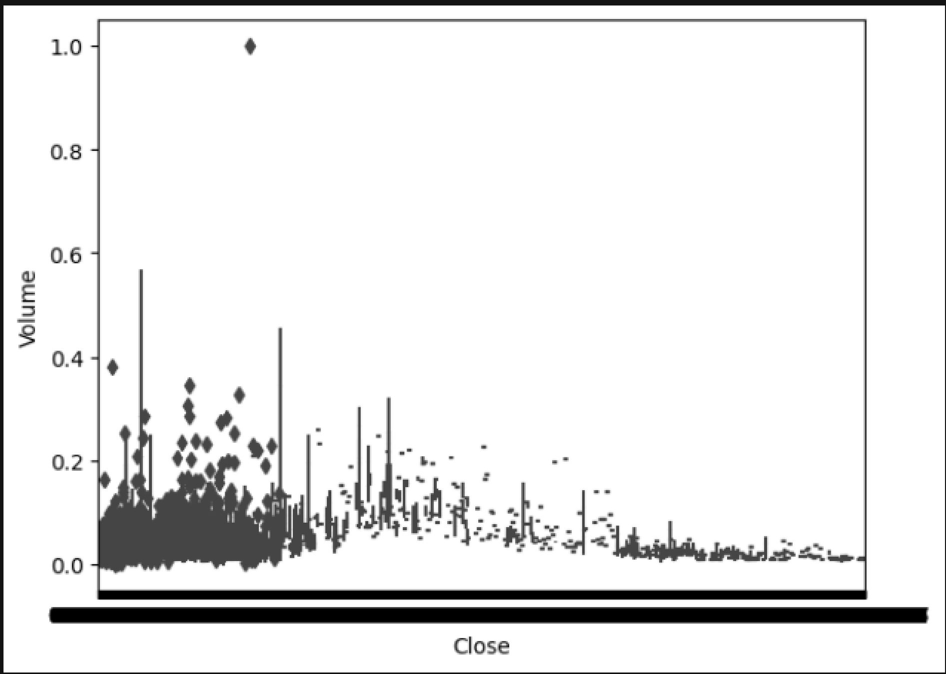
```python
sns.boxplot(Data)
```

```
<AxesSubplot: >
```



```python
sns.boxplot(x = "Close", y = "Volume", data = Data)
```

```
<AxesSubplot: xlabel='Close', ylabel='Volume'>
```



These are the various visualizations of data. Now we can use this data and apply it to various models.

PHASE 2

DATA MODELLING

Split your data into training, validation, and testing.

```python
X = Data[["Open", "High", "Low", "Volume"]]
y = Data["Close"]

# Import the train_test_split function
from sklearn.model_selection import train_test_split

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Split the training set further into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2)

print("Train Size: " + str(X_train.shape[0]))
print("Test Size: " + str(X_test.shape[0]))
print("Validation Size: " + str(X_val.shape[0]))
```

```
Train Size: 6699
Test Size: 2094
Validation Size: 1675
```

1. LINEAR REGRESSION (LR)

- Linear regression is a statistical method used to model the relationship between a dependent variable (in this case, stock price) and one or more independent variables (in this case, potential factors that may influence stock price). By fitting a linear regression model to historical data, we can attempt to predict future stock prices based on the relationship between the dependent and independent variables.

Model Implementation

```
from sklearn.linear_model import LinearRegression

# Create a linear regression model
lr_model = LinearRegression()

# Fit the model to the data
lr_model.fit(X_train, y_train)

# Use the model to make predictions
y_pred_lr = lr_model.predict(X_test)

lr_score_test = lr_model.score(X_test, y_test)
lr_score_train = lr_model.score(X_train, y_train)

# Print the evaluation score
print('Test Data Acurracy:', lr_score_test)
print('Train Data Acurracy:', lr_score_train)
```

[33]

```
Test Data Acurracy: 0.9999342182418407
Train Data Acurracy: 0.9999353893860727
```

Accuracy

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

kf = KFold(n_splits = 5)
scores = cross_val_score(lr_model, X, y, cv = kf)
print("Average Accuracy Using KFold:", scores.mean())
```

[34]

```
Average Accuracy Using KFold: 0.9994320648119583
```

Mean Absolute Error

```
from sklearn.metrics import mean_absolute_error
lr_mae = mean_absolute_error(y_test, y_pred_lr)
print('Mean Absolute Error:', lr_mae)
```

[35]

```
Mean Absolute Error: 0.07816850986439684
```

2. KTH NEAREST NEIGHBOUR (KNN)

- KNN, or k-nearest neighbors, is a machine learning algorithm that can be used for a variety of purposes, including stock price prediction. In the context of stock price prediction, the KNN algorithm would take historical data on the prices of a particular stock as well as other relevant factors (such as the overall performance of the stock market, the performance of competing stocks, and macroeconomic indicators) and use this data to make predictions about future stock prices.

Model Implementation

```
from sklearn.neighbors import KNeighborsRegressor

# Create a KNN model
knn_model = KNeighborsRegressor(n_neighbors = 5)

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

# Use the model to make predictions
y_pred_knn = knn_model.predict(X_test)

knn_score_test = knn_model.score(X_test, y_test)
knn_score_train = knn_model.score(X_train, y_train)

# Print the evaluation score
print('Test Data Acurracy:', knn_score_test)
print('Train Data Acurracy:', knn_score_train)
```
[36]

```
Test Data Acurracy: 0.9998921194020267
Train Data Acurracy: 0.9999064522089086
```

Accuracy

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

kf = KFold(n_splits = 5)
scores = cross_val_score(knn_model, X, y, cv = kf)
print("Average Accuracy Using KFold:", scores.mean())
```
[37]

```
Average Accuracy Using KFold: 0.5429704778350681
```

Mean Absolute Error

```
from sklearn.metrics import mean_absolute_error
knn_mae = mean_absolute_error(y_test, y_pred_knn)
print('Mean Absolute Error:', knn_mae)
```
[38]

```
Mean Absolute Error: 0.10891471929321878
```

3. Support Vector Machine (SVM)

- Support vector machines (SVMs) are a type of supervised learning algorithm that can be used for classification or regression tasks. In the context of stock price prediction, an SVM could be used to classify whether the price of a stock will go up or down based on historical data. The SVM algorithm works by finding the best line or hyperplane that separates the data into different classes, allowing it to make predictions on new data based on this line. While SVMs are not the most commonly used algorithm for stock price prediction, they can be effective in certain cases.

Model Implementation

```
from sklearn.svm import SVR

# Create a SVM regression model
svm_model = SVR(kernel="rbf", C=1.0, epsilon = 0.1)

# Fit the model to the data
svm_model.fit(X_train, y_train)

# Use the model to make predictions
y_pred_svm = svm_model.predict(X_test)

svm_score_test = svm_model.score(X_test, y_test)
svm_score_train = svm_model.score(X_train, y_train)

# Print the evaluation score
print('Test Data Acurracy:',svm_score_test)
print('Train Data Acurracy:',svm_score_train)
```
[39]

```
Test Data Acurracy: 0.9971173335450411
Train Data Acurracy: 0.9962495093460568
```

Accuracy

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

kf = KFold(n_splits = 5)
scores = cross_val_score(svm_model, X, y, cv = kf)
print("Average Accuracy Using KFold:", scores.mean())
```
[40]

```
Average Accuracy Using KFold: 0.27237926891317005
```

Mean Absolute Error

```
from sklearn.metrics import mean_absolute_error
svm_mae = mean_absolute_error(y_test, y_pred_svm)
print('Mean Absolute Error:', svm_mae)
```
[41]

```
Mean Absolute Error: 0.32294731263018944
```

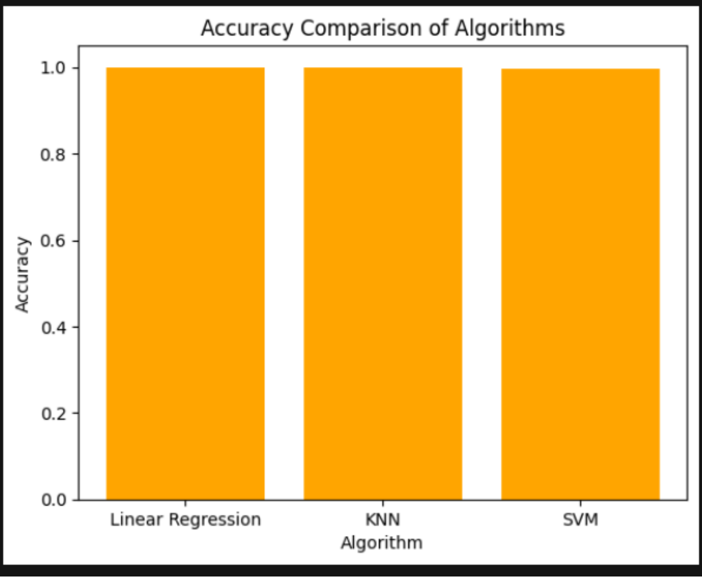
Comparison Plots

Accuracy vs. Algorithm Plot

```
plt.bar(["Linear Regression", "KNN", "SVM"], [lr_score_test, knn_score_test, svm_score_test], color = "orange")
plt.xlabel("Algorithm")
plt.ylabel("Accuracy")
plt.title("Accuracy Comparison of Algorithms")
plt.show()
```

[42]

...

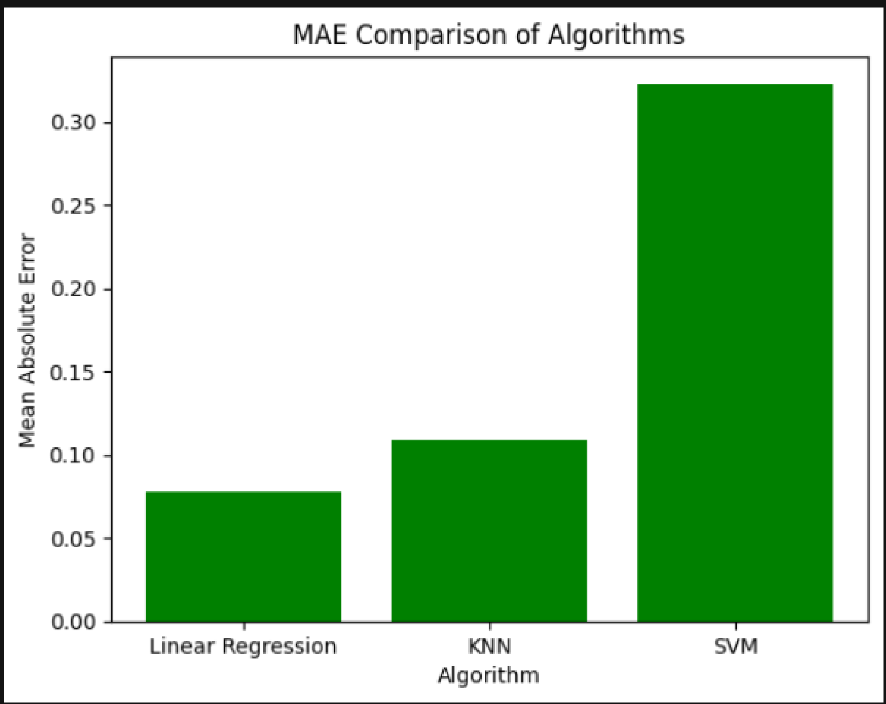


Error vs. Algorithm Plot

```
plt.bar(["Linear Regression", "KNN", "SVM"], [lr_mae, knn_mae, svm_mae], color = "green")
plt.xlabel("Algorithm")
plt.ylabel("Mean Absolute Error")
plt.title("MAE Comparison of Algorithms")
plt.show()
```

[43]

...



2. GOOGLE DATASET

IMPORTING LIBRARIES

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import warnings
import string
import csv
import re

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from pandas.plotting import scatter_matrix
from collections import Counter
from sklearn import metrics

warnings.filterwarnings("ignore")
```
[1]

EXTRACTING DATA - Reading the CSV File:

```
Data = pd.read_csv("Google.csv")
print(Data)
```
[2]

```
            Date         Open         High          Low        Close  \
0     2004-08-19    50.050049    52.082081    48.028027    50.220219
1     2004-08-20    50.555557    54.594597    50.300301    54.209209
2     2004-08-23    55.430431    56.796799    54.579578    54.754753
3     2004-08-24    55.675674    55.855858    51.836838    52.487488
4     2004-08-25    52.532532    54.054054    51.991993    53.053055
...          ...          ...          ...          ...          ...
4426  2022-03-18  2668.489990  2724.879883  2645.169922  2722.510010
4427  2022-03-21  2723.270020  2741.000000  2681.850098  2722.030029
4428  2022-03-22  2722.030029  2821.000000  2722.030029  2797.360107
4429  2022-03-23  2774.050049  2791.770020  2756.699951  2765.510010
4430  2022-03-24  2784.000000  2832.379883  2755.010010  2831.439941

        Adj Close    Volume
0       50.220219  44659096
1       54.209209  22834343
2       54.754753  18256126
3       52.487488  15247337
4       53.053055   9188602
...           ...       ...
4426  2722.510010   2223100
4427  2722.030029   1341600
4428  2797.360107   1774800
4429  2765.510010   1257700
4430  2831.439941   1317900

[4431 rows x 7 columns]
```

DATA PREPROCESSING

```
# Finding size of dataset (i.e number of rows & columns of the dataset.)
print("Rows: ",Data.shape[0])
print("Columns: ",Data.shape[1])
```

[3]

```
Rows:  4431
Columns:  7
```

```
Data.head() # head() function by default showcases first five rows
```

[4]

	Date	Open	High	Low	Close	Adj Close	Volume
0	2004-08-19	50.050049	52.082081	48.028027	50.220219	50.220219	44659096
1	2004-08-20	50.555557	54.594597	50.300301	54.209209	54.209209	22834343
2	2004-08-23	55.430431	56.796799	54.579578	54.754753	54.754753	18256126
3	2004-08-24	55.675674	55.855858	51.836838	52.487488	52.487488	15247337
4	2004-08-25	52.532532	54.054054	51.991993	53.053055	53.053055	9188602

```
Data.shape
```

[5]

```
(4431, 7)
```

```
Data.describe()
```

[6]

	Open	High	Low	Close	Adj Close	Volume
count	4431.000000	4431.000000	4431.000000	4431.000000	4431.000000	4.431000e+03
mean	693.087345	699.735595	686.078751	693.097367	693.097367	6.444992e+06
std	645.118799	651.331215	638.579488	645.187806	645.187806	7.690351e+06
min	49.644646	50.920921	48.028027	50.055054	50.055054	4.656000e+05
25%	248.558563	250.853355	245.813309	248.415916	248.415916	1.695600e+06
50%	434.924927	437.887878	432.687683	435.330322	435.330322	3.778418e+06
75%	1007.364990	1020.649994	997.274994	1007.790008	1007.790008	8.002390e+06
max	3025.000000	3030.929932	2977.979980	2996.770020	2996.770020	8.215117e+07

Different Data Types In Datasets:

```
Data.columns
```

[7]

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

```
[8]
Data.dtypes
```

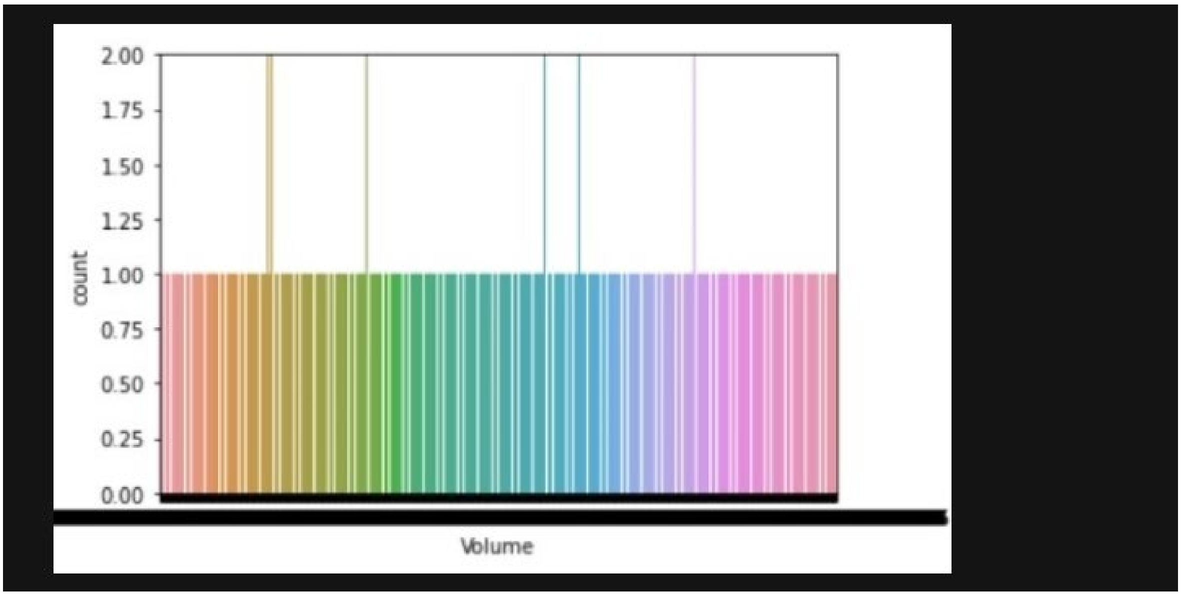
```
Date          object
Open         float64
High         float64
Low          float64
Close        float64
Adj Close    float64
Volume         int64
dtype: object
```

```
[9]
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4431 entries, 0 to 4430
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       4431 non-null   object 
 1   Open       4431 non-null   float64
 2   High       4431 non-null   float64
 3   Low        4431 non-null   float64
 4   Close      4431 non-null   float64
 5   Adj Close  4431 non-null   float64
 6   Volume     4431 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 242.4+ KB
```

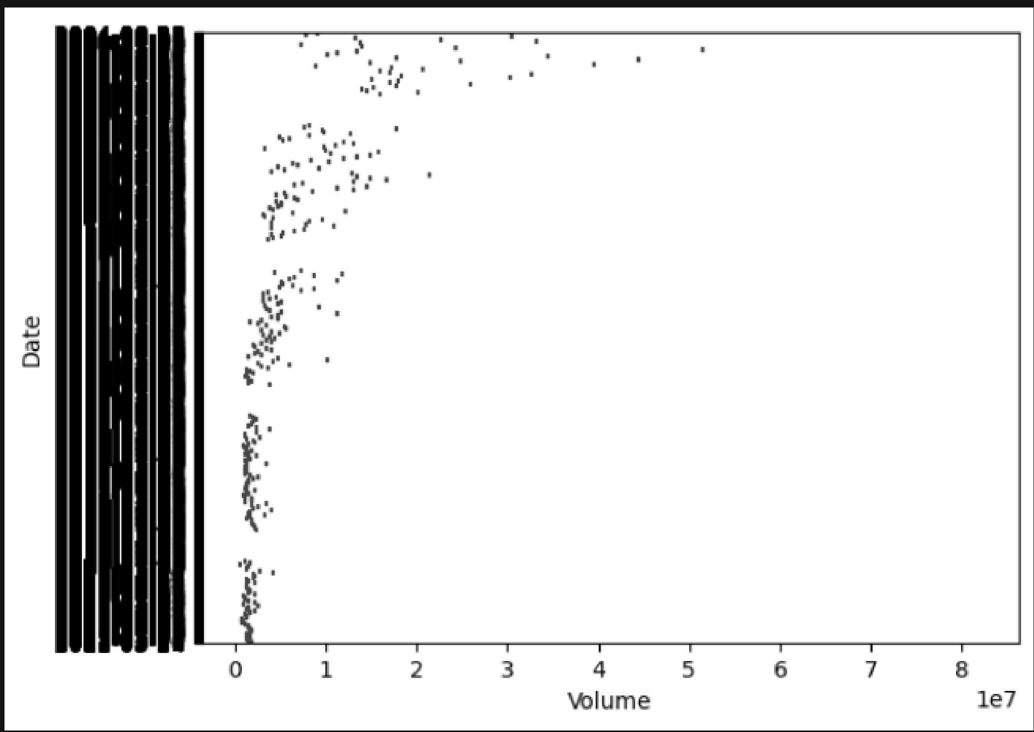
```
[10]
# Finding Number of samples under target variable
print(f"Number of samples under target value: \n{Data['Volume'].value_counts()}")
sns.countplot(Data.Volume).set_ylim(0, 2)
plt.show()
```

```
Number of samples under target value:
1660500    3
1529700    3
3346850    2
9680310    2
3865531    2
          ..
5612582    1
5783810    1
5328266    1
5076518    1
1317900    1
Name: Volume, Length: 4317, dtype: int64
```




```
sns.boxplot(x = "Volume", y = "Date", data = Data)
plt.show()
```

[11]

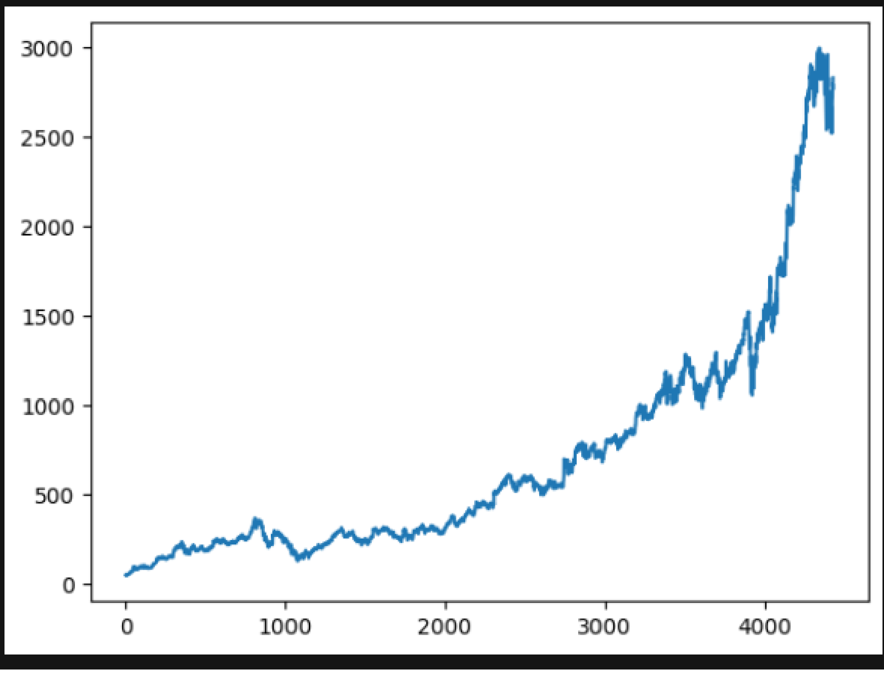


The Adjusted Close Value is the final output value that will be forecasted using the Machine Learning model.

```
# Plot the True Close Value
Data['Close'].plot()
```

[12]

```
<AxesSubplot: >
```



DATA CLEANING

```
Data.isnull().values.any()  # Checking whether we have any missing values in dataset
```

[13]

```
False
```

```
Data.isnull().sum()
```

[14]

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

There were no missing values in the datasets. So, there was no replacement and missing values.

DATA STANDARDIZATION

```
[15] Data.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2004-08-19	50.050049	52.082081	48.028027	50.220219	50.220219	44659096
1	2004-08-20	50.555557	54.594597	50.300301	54.209209	54.209209	22834343
2	2004-08-23	55.430431	56.796799	54.579578	54.754753	54.754753	18256126
3	2004-08-24	55.675674	55.855858	51.836838	52.487488	52.487488	15247337
4	2004-08-25	52.532532	54.054054	51.991993	53.053055	53.053055	9188602

```
[16] Data['Volume'][:5]
```

```
0    44659096
1    22834343
2    18256126
3    15247337
4     9188602
Name: Volume, dtype: int64
```

```
scalar = StandardScaler(copy=True, with_mean=True, with_std=True)
Data["Volume"] = scalar.fit_transform(Data["Volume"].values.reshape(-1,1))
print ("After Standardisation: ")
Data.head()
[17]
```

```
After Standardisation:
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2004-08-19	50.050049	52.082081	48.028027	50.220219	50.220219	4.969659
1	2004-08-20	50.555557	54.594597	50.300301	54.209209	54.209209	2.131398
2	2004-08-23	55.430431	56.796799	54.579578	54.754753	54.754753	1.536011
3	2004-08-24	55.675674	55.855858	51.836838	52.487488	52.487488	1.144725
4	2004-08-25	52.532532	54.054054	51.991993	53.053055	53.053055	0.356800

DATA NORMALIZATION

```
norm = MinMaxScaler()
Data["Volume"] = norm.fit_transform(Data["Volume"].values.reshape(-1,1))
print ("After Normalisation: ")
Data.head()
[18]
```

```
After Normalisation:
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2004-08-19	50.050049	52.082081	48.028027	50.220219	50.220219	0.541020
1	2004-08-20	50.555557	54.594597	50.300301	54.209209	54.209209	0.273840
2	2004-08-23	55.430431	56.796799	54.579578	54.754753	54.754753	0.217793
3	2004-08-24	55.675674	55.855858	51.836838	52.487488	52.487488	0.180959
4	2004-08-25	52.532532	54.054054	51.991993	53.053055	53.053055	0.106788

Making data available for various ML models through normalization.

Discretization

```
Data['Close'].unique()
```

[19]

```
array([  50.220219,   54.209209,   54.754753, ..., 2797.360107,
       2765.51001 , 2831.439941])
```

```
print(Data['Close'].max())
print(Data['Close'].min())
```

[20]

```
2996.77002
50.055054
```

```
Data['bin_of_Close'] = pd.cut(Data['Close'], [200,300,400,500,600,700],
                              labels=['200-300', '300-400', '400-500', '500-600', '600-700'])
```

[21]

```
Data.groupby([Data["bin_of_Close"]]).count()
```

[22]

bin_of_Close	Date	Open	High	Low	Close	Adj Close	Volume
200-300	1126	1126	1126	1126	1126	1126	1126
300-400	438	438	438	438	438	438	438
400-500	155	155	155	155	155	155	155
500-600	408	408	408	408	408	408	408
600-700	101	101	101	101	101	101	101

```
for column in Data.columns:
    print("------  " + column + "  ------")
    print(Data[column].value_counts())
```

[23]

```
Output exceeds the size limit. Open the full output data in a text editor
------  Date  ------
2004-08-19    1
2016-05-11    1
2016-05-19    1
2016-05-18    1
2016-05-17    1
             ..
2010-07-09    1
2010-07-12    1
2010-07-13    1
2010-07-14    1
2022-03-24    1
Name: Date, Length: 4431, dtype: int64
```

```
------  Open  ------
295.795807     4
307.807800     3
263.523529     3
230.230225     3
281.781769     3
              ..
244.744751     1
242.082077     1
244.509506     1
245.465469     1
2784.000000    1
...
500-600     408
400-500     155
600-700     101
Name: bin_of_Close, dtype: int64
```

Making the values group-wise and making continuous values discrete.

DATA SUMMARIZATION

```
[24]
print(Data.shape)
```

```
(4431, 8)
```

```
[25]
Data.describe()
```

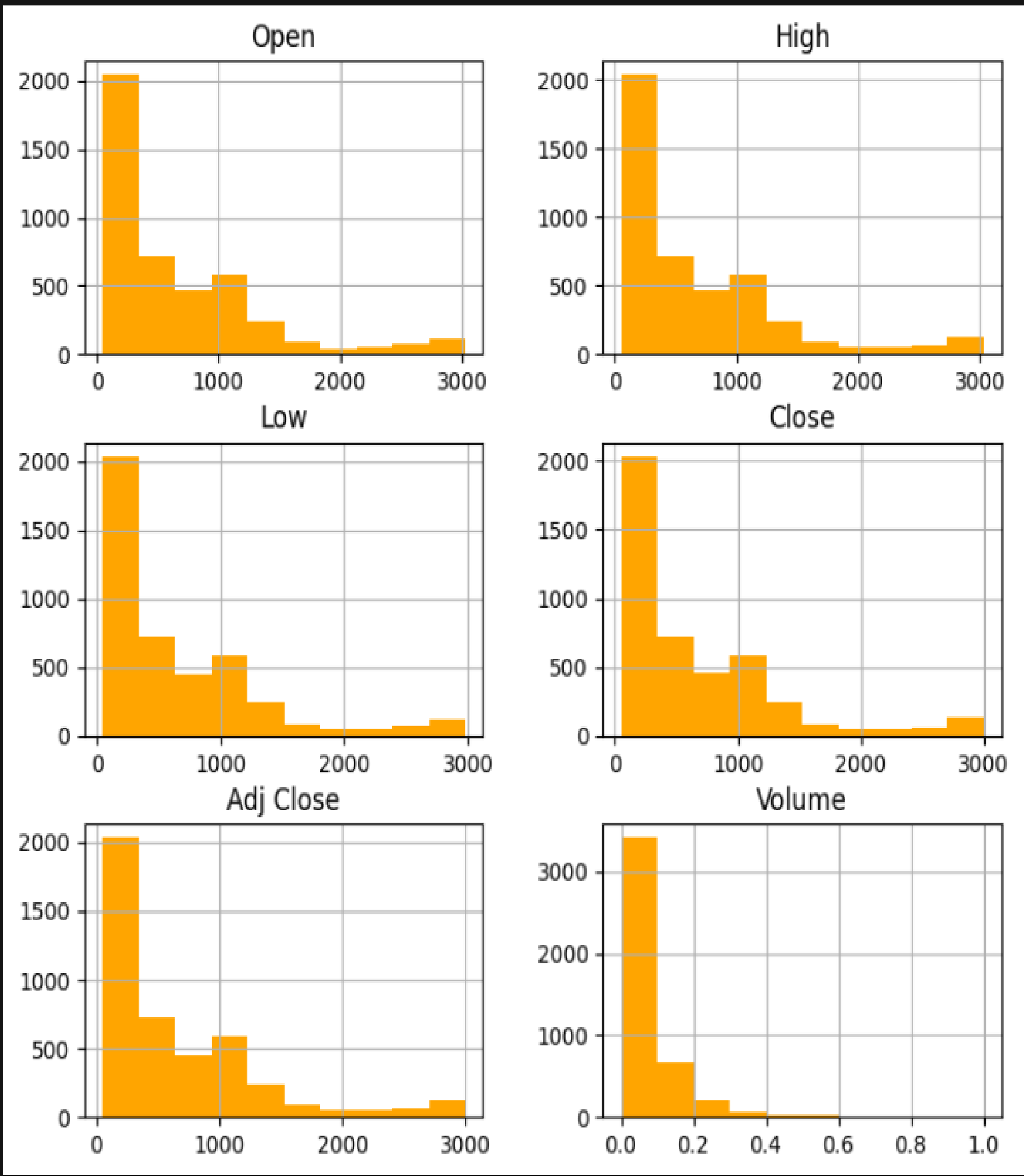
	Open	High	Low	Close	Adj Close	Volume
count	4431.000000	4431.000000	4431.000000	4431.000000	4431.000000	4431.000000
mean	693.087345	699.735595	686.078751	693.097367	693.097367	0.073200
std	645.118799	651.331215	638.579488	645.187806	645.187806	0.094146
min	49.644646	50.920921	48.028027	50.055054	50.055054	0.000000
25%	248.558563	250.853355	245.813309	248.415916	248.415916	0.015058
50%	434.924927	437.887878	432.687683	435.330322	435.330322	0.040556
75%	1007.364990	1020.649994	997.274994	1007.790008	1007.790008	0.092266
max	3025.000000	3030.929932	2977.979980	2996.770020	2996.770020	1.000000

DATA VISUALIZATION

Histogram

```
Data.hist(color = "orange", figsize = (8,8))
plt.show()
```

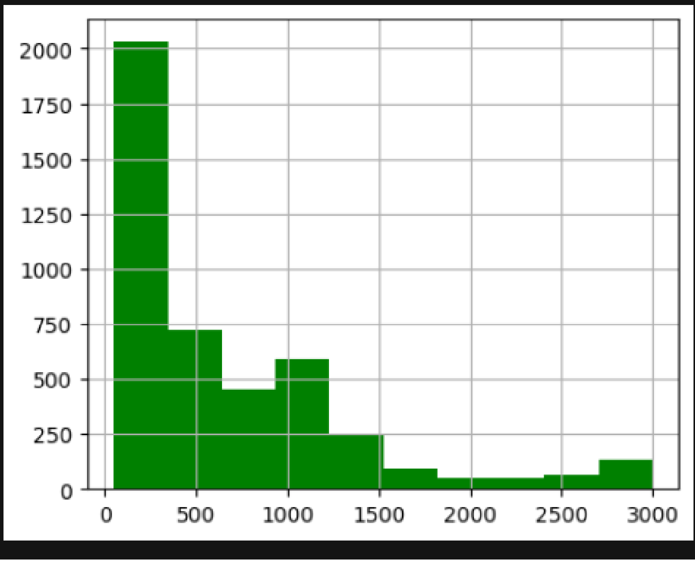
[26]



```python
Data['Close'].hist(color = "green", figsize = (5,4))
```

[27]

```
<AxesSubplot: >
```

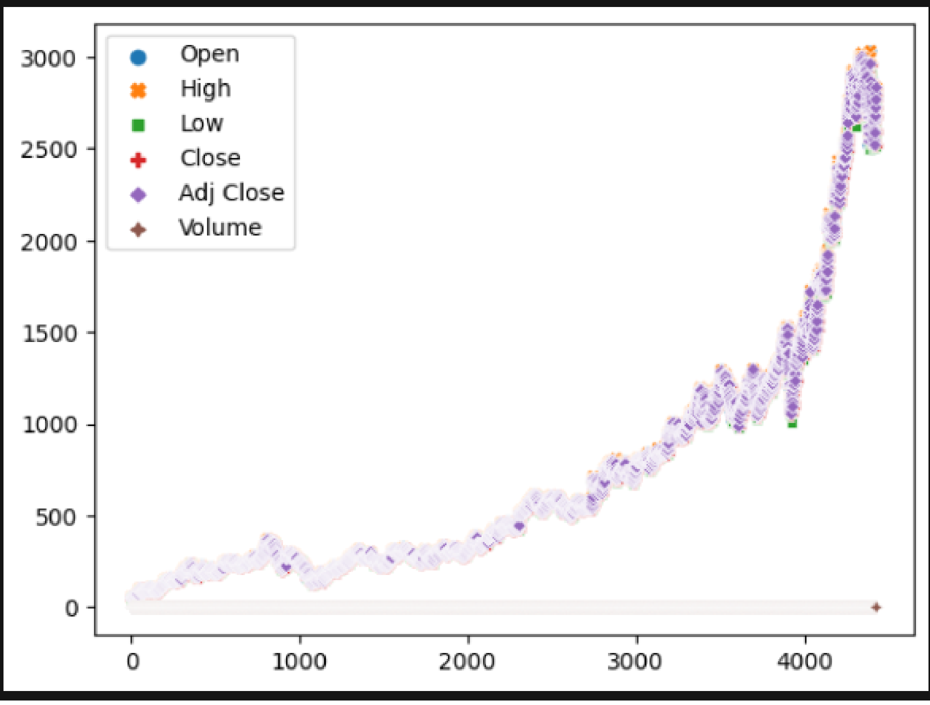


Scatter Plot

```python
sns.scatterplot(Data)
```

[28]

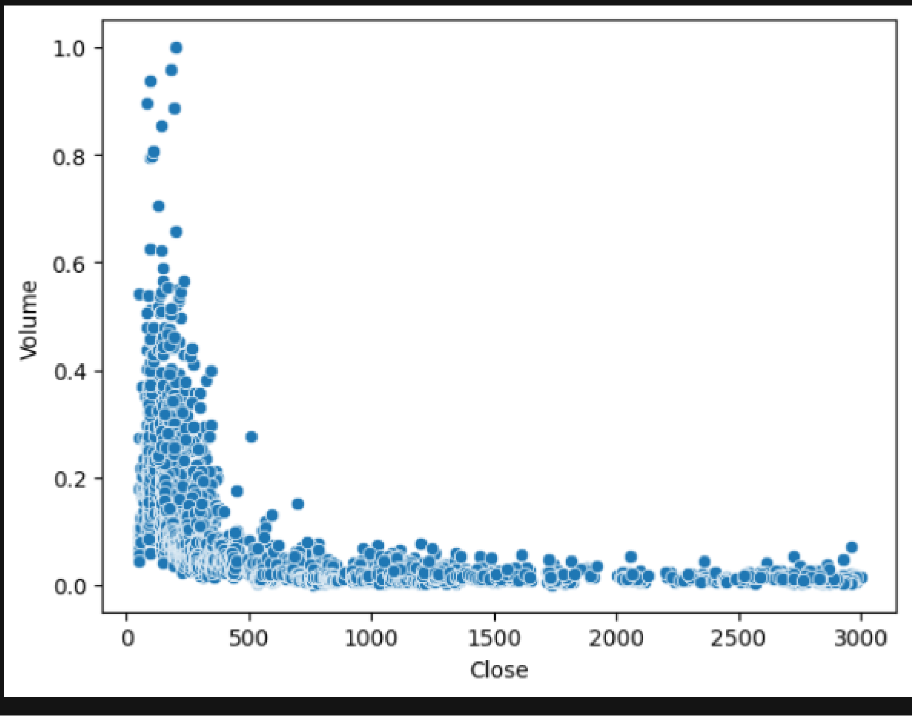
```
<AxesSubplot: >
```



```
sns.scatterplot(x = Data['Close'], y = Data['Volume'])
```

[29]

```
<AxesSubplot: xlabel='Close', ylabel='Volume'>
```

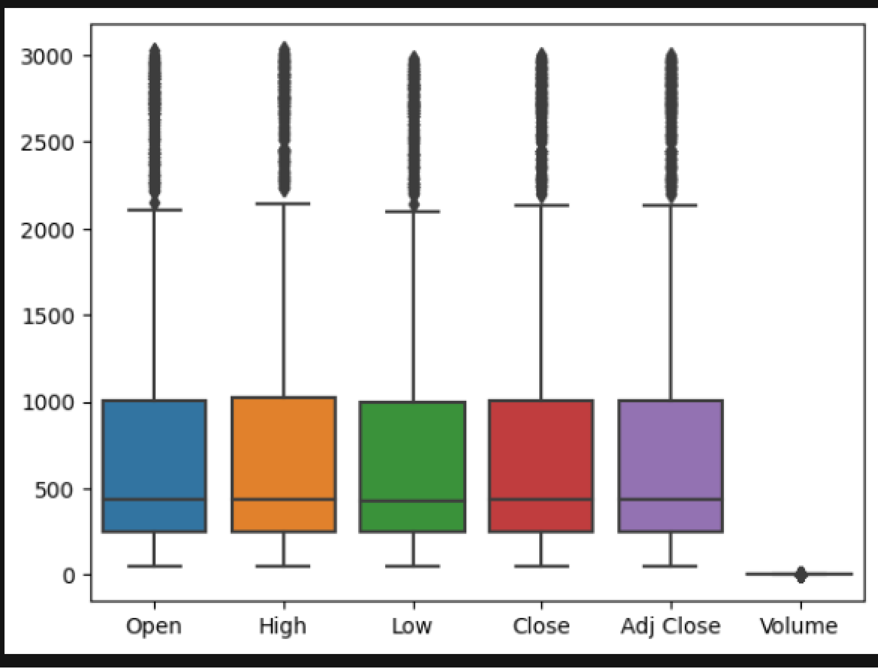


Box Plot

```
sns.boxplot(Data)
```

[30]

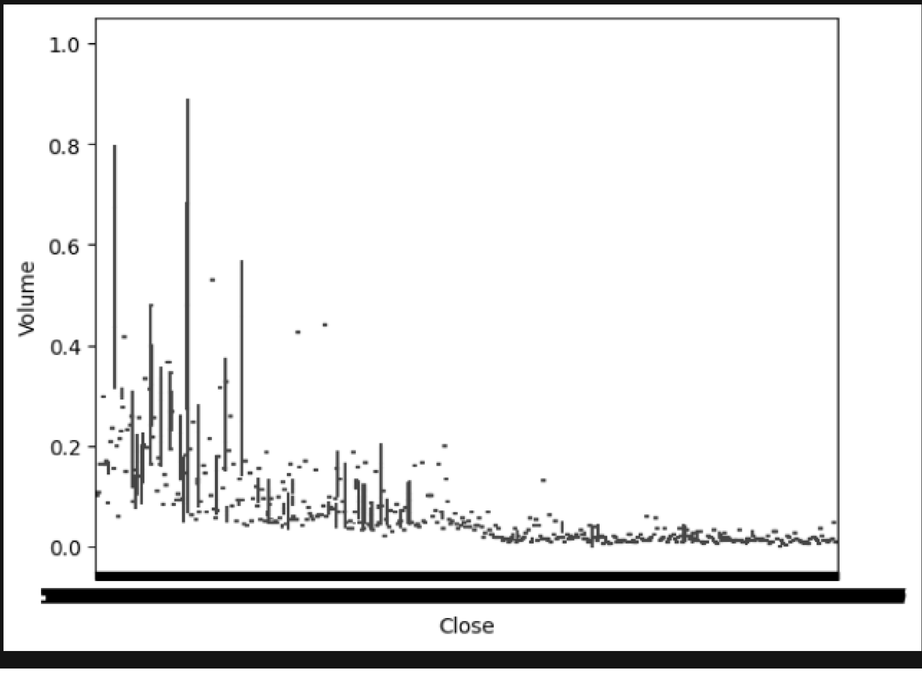
```
<AxesSubplot: >
```



```
sns.boxplot(x = "Close", y = "Volume", data = Data)
```

[31]

```
<AxesSubplot: xlabel='Close', ylabel='Volume'>
```



These are the various visualizations of data. Now we can use this data and apply it to various models.

<u>PHASE 2</u>

DATA MODELLING

Split your data into training, validation, and testing.

```
X = Data[["Open", "High", "Low", "Volume"]]
y = Data["Close"]

# Import the train_test_split function
from sklearn.model_selection import train_test_split

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Split the training set further into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2)

print("Train Size: " + str(X_train.shape[0]))
print("Test Size: " + str(X_test.shape[0]))
print("Validation Size: " + str(X_val.shape[0]))
```

[32]

```
Train Size: 2835
Test Size: 887
Validation Size: 709
```

1. Linear Regression (LR)

- Linear regression is a statistical method used to model the relationship between a dependent variable (in this case, stock price) and one or more independent variables (in this case, potential factors that may influence stock price). By fitting a linear regression model to historical data, we can attempt to predict future stock prices based on the relationship between the dependent and independent variables.

Model Implementation

```
from sklearn.linear_model import LinearRegression

# Create a linear regression model
lr_model = LinearRegression()

# Fit the model to the data
lr_model.fit(X_train, y_train)

# Use the model to make predictions
y_pred_lr = lr_model.predict(X_test)

lr_score_test = lr_model.score(X_test, y_test)
lr_score_train = lr_model.score(X_train, y_train)

# Print the evaluation score
print('Test Data Acurracy:', lr_score_test)
print('Train Data Acurracy:', lr_score_train)
```

[33]

```
Test Data Acurracy: 0.9999198372766136
Train Data Acurracy: 0.9999163514197196
```

Accuracy

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

kf = KFold(n_splits = 5)
scores = cross_val_score(lr_model, X, y, cv = kf)
print("Average Accuracy Using KFold:", scores.mean())
```

[34]

```
Average Accuracy Using KFold: 0.9993725733985496
```

Mean Absolute Error

```
from sklearn.metrics import mean_absolute_error
lr_mae = mean_absolute_error(y_test, y_pred_lr)
print('Mean Absolute Error:', lr_mae)
```

[35]

```
Mean Absolute Error: 3.275319889397112
```

2. KTH NEAREST NEIGHBOUR (KNN)

- KNN, or k-nearest neighbors, is a machine learning algorithm that can be used for a variety of purposes, including stock price prediction. In the context of stock price prediction, the KNN algorithm would take historical data on the prices of a particular stock as well as other relevant factors (such as the overall performance of the stock market, the performance of competing stocks, and macroeconomic indicators) and use this data to make predictions about future stock prices.

Model Implementation

```
from sklearn.neighbors import KNeighborsRegressor

# Create a KNN model
knn_model = KNeighborsRegressor(n_neighbors = 5)

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

# Use the model to make predictions
y_pred_knn = knn_model.predict(X_test)

knn_score_test = knn_model.score(X_test, y_test)
knn_score_train = knn_model.score(X_train, y_train)

# Print the evaluation score
print('Test Data Acurracy:', knn_score_test)
print('Train Data Acurracy:', knn_score_train)
```
[36]

```
Test Data Acurracy: 0.9998394047132592
Train Data Acurracy: 0.9998913266551764
```

Accuracy

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

kf = KFold(n_splits = 5)
scores = cross_val_score(knn_model, X, y, cv = kf)
print("Average Accuracy Using KFold:", scores.mean())
```
[37]

```
Average Accuracy Using KFold: 0.6064639898689311
```

Mean Absolute Error

```
from sklearn.metrics import mean_absolute_error
knn_mae = mean_absolute_error(y_test, y_pred_knn)
print('Mean Absolute Error:', knn_mae)
```
[38]

```
Mean Absolute Error: 4.5004368489289766
```

3. Support Vector Machine (SVM)

- Support vector machines (SVMs) are a type of supervised learning algorithm that can be used for classification or regression tasks. In the context of stock price prediction, an SVM could be used to classify whether the price of a stock will go up or down based on historical data. The SVM algorithm works by finding the best line or hyperplane that separates the data into different classes, allowing it to make predictions on new data based on this line. While SVMs are not the most commonly used algorithm for stock price prediction, they can be effective in certain cases.

Model Implementation

```
from sklearn.svm import SVR

# Create a SVM regression model
svm_model = SVR(kernel="rbf", C=1.0, epsilon=0.1)

# Fit the model to the data
svm_model.fit(X_train, y_train)

# Use the model to make predictions
y_pred_svm = svm_model.predict(X_test)

svm_score_test = svm_model.score(X_test, y_test)
svm_score_train = svm_model.score(X_train, y_train)

# Print the evaluation score
print('Test Data Acurracy:',svm_score_test)
print('Train Data Acurracy:',svm_score_train)
```
[39]

```
Test Data Acurracy: 0.5416163467746646
Train Data Acurracy: 0.5567320044427395
```

Accuracy

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

kf = KFold(n_splits = 5)
scores = cross_val_score(svm_model, X, y, cv = kf)
print("Average Accuracy Using KFold:", scores.mean())
```
[40]

```
Average Accuracy Using KFold: -0.45819663739951577
```

Mean Absolute Error

```
from sklearn.metrics import mean_absolute_error
svm_mae = mean_absolute_error(y_test, y_pred_svm)
print('Mean Absolute Error:', svm_mae)
```
[41]

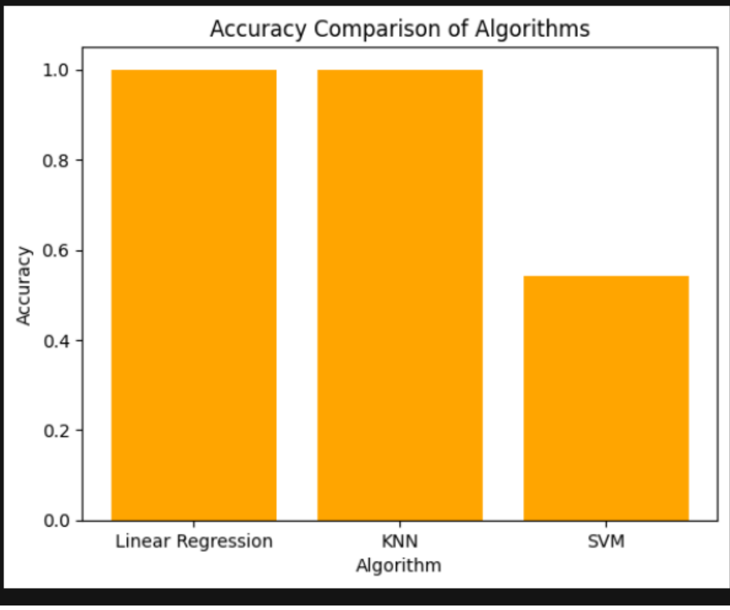
```
Mean Absolute Error: 167.91231161025675
```

Comparison Plots

Accuracy vs. Algorithm Plot

```
plt.bar(["Linear Regression", "KNN", "SVM"], [lr_score_test, knn_score_test, svm_score_test], color = "orange")
plt.xlabel("Algorithm")
plt.ylabel("Accuracy")
plt.title("Accuracy Comparison of Algorithms")
plt.show()
```

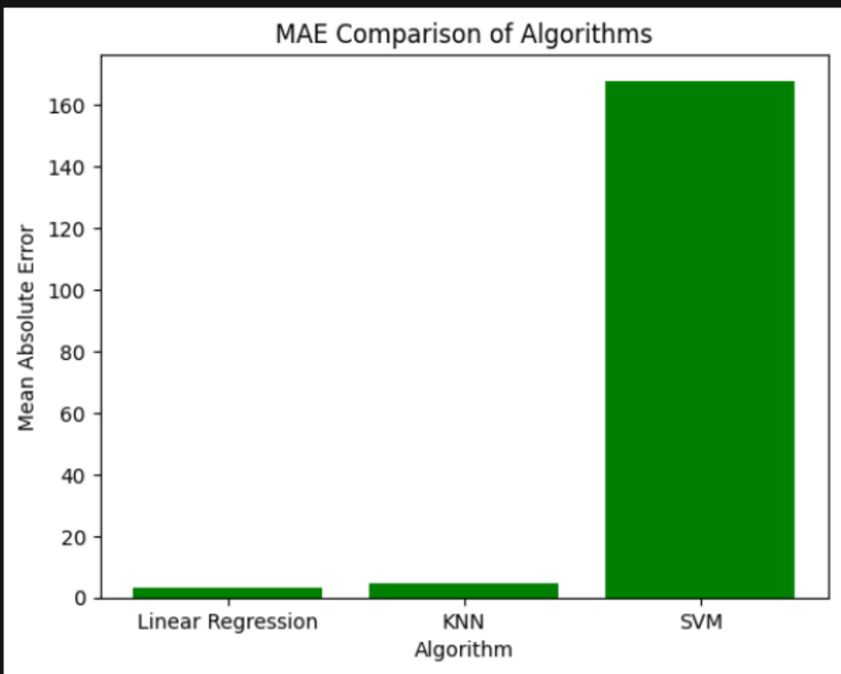
[42]



Error vs. Algorithm Plot

```
plt.bar(["Linear Regression", "KNN", "SVM"], [lr_mae, knn_mae, svm_mae], color = "green")
plt.xlabel("Algorithm")
plt.ylabel("Mean Absolute Error")
plt.title("MAE Comparison of Algorithms")
plt.show()
```

[43]



3. NETFLIX DATASET

IMPORTING LIBRARIES

```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import warnings
import string
import csv
import re

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from pandas.plotting import scatter_matrix
from collections import Counter
from sklearn import metrics

warnings.filterwarnings("ignore")
```
[1]

EXTRACTING DATA - Reading the CSV File:

```python
Data = pd.read_csv("Netflix.csv")
print(Data)
```
[2]

```
            Date        Open        High         Low       Close   Adj Close  \
0     2018-02-05  262.000000  267.899994  250.029999  254.259995  254.259995
1     2018-02-06  247.699997  266.700012  245.000000  265.720001  265.720001
2     2018-02-07  266.579987  272.450012  264.329987  264.559998  264.559998
3     2018-02-08  267.079987  267.619995  250.000000  250.100006  250.100006
4     2018-02-09  253.850006  255.800003  236.110001  249.470001  249.470001
...          ...         ...         ...         ...         ...         ...
1004  2022-01-31  401.970001  427.700012  398.200012  427.140015  427.140015
1005  2022-02-01  432.959991  458.480011  425.540009  457.130005  457.130005
1006  2022-02-02  448.250000  451.980011  426.480011  429.480011  429.480011
1007  2022-02-03  421.440002  429.260010  404.279999  405.600006  405.600006
1008  2022-02-04  407.309998  412.769989  396.640015  410.170013  410.170013

        Volume
0     11896100
1     12595800
2      8981500
3      9306700
4     16906900
...        ...
1004  20047500
1005  22542300
1006  14346000
1007   9905200
1008   7782400

[1009 rows x 7 columns]
```

DATA PREPROCESSING

```
[3]
# Finding size of dataset (i.e number of rows & columns of the dataset.)
print("Rows: ",Data.shape[0])
print("Columns: ",Data.shape[1])
```

```
Rows:  1009
Columns:  7
```

```
[4]
Data.head() # head() function by default showcases first five rows
```

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

```
[5]
Data.shape
```

```
(1009, 7)
```

```
[6]
Data.describe()
```

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

Different Data Types In Datasets:

```
[7]
Data.columns
```

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

```python
Data.dtypes
```

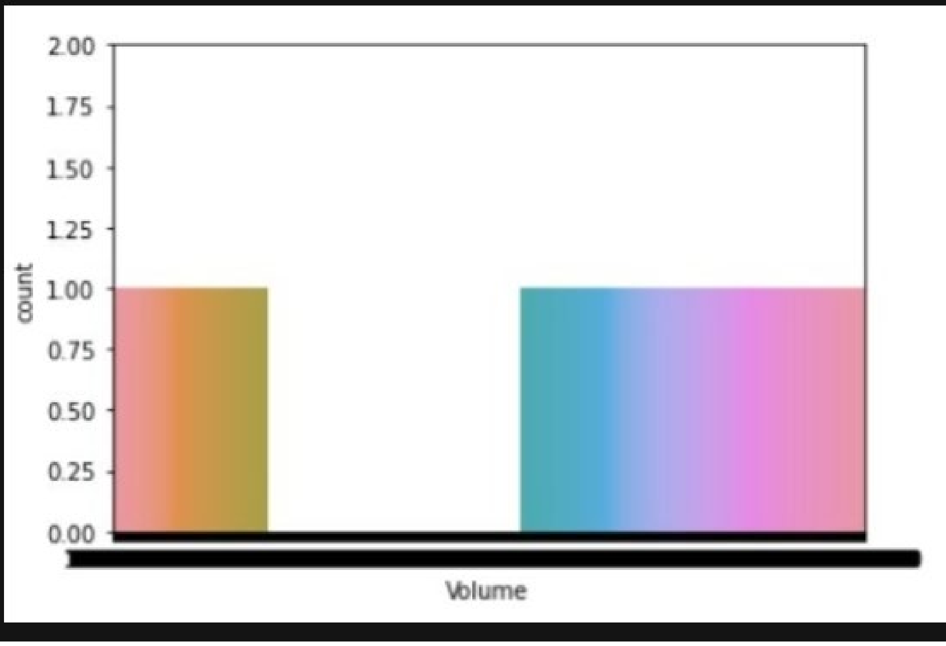
```
[8]
...
Date          object
Open         float64
High         float64
Low          float64
Close        float64
Adj Close    float64
Volume         int64
dtype: object
```

```python
Data.info()
```

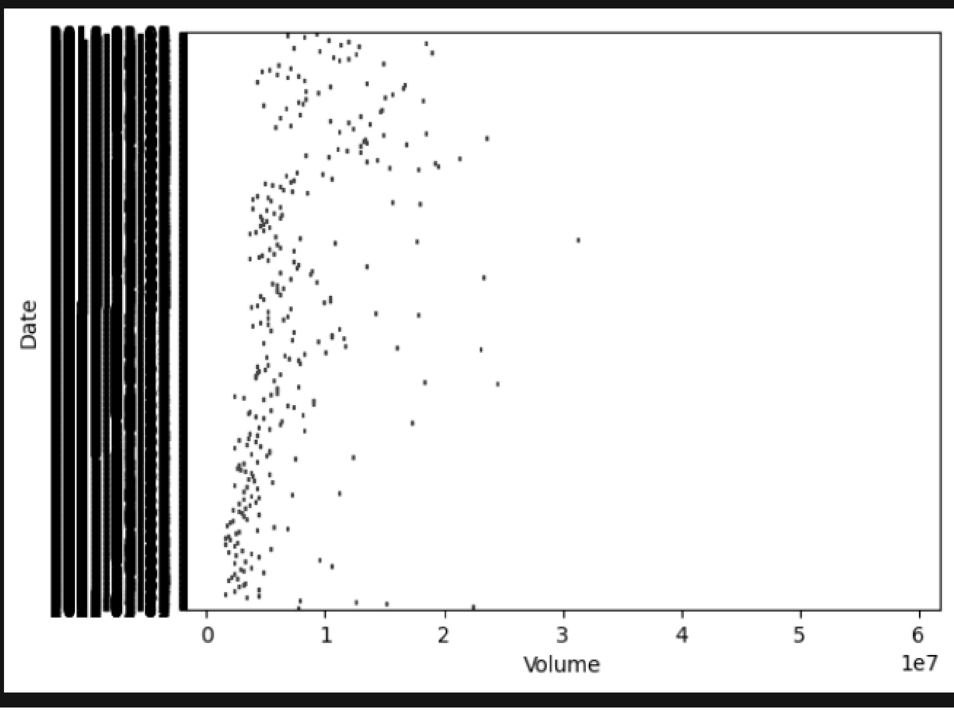
```
[9]
...
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       1009 non-null   object 
 1   Open       1009 non-null   float64
 2   High       1009 non-null   float64
 3   Low        1009 non-null   float64
 4   Close      1009 non-null   float64
 5   Adj Close  1009 non-null   float64
 6   Volume     1009 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 55.3+ KB
```

```python
# Finding Number of samples under target variable
print(f"Number of samples under target value: \n{Data['Volume'].value_counts()}")
sns.countplot(Data.Volume).set_ylim(0, 2)
plt.show()
```

```
[10]
...
Number of samples under target value:
6717700    2
5439200    2
3732200    2
6997900    2
4408200    1
          ..
5019000    1
5358200    1
5428500    1
5667200    1
7782400    1
Name: Volume, Length: 1005, dtype: int64
```




```python
sns.boxplot(x = "Volume", y = "Date", data = Data)
plt.show()
```

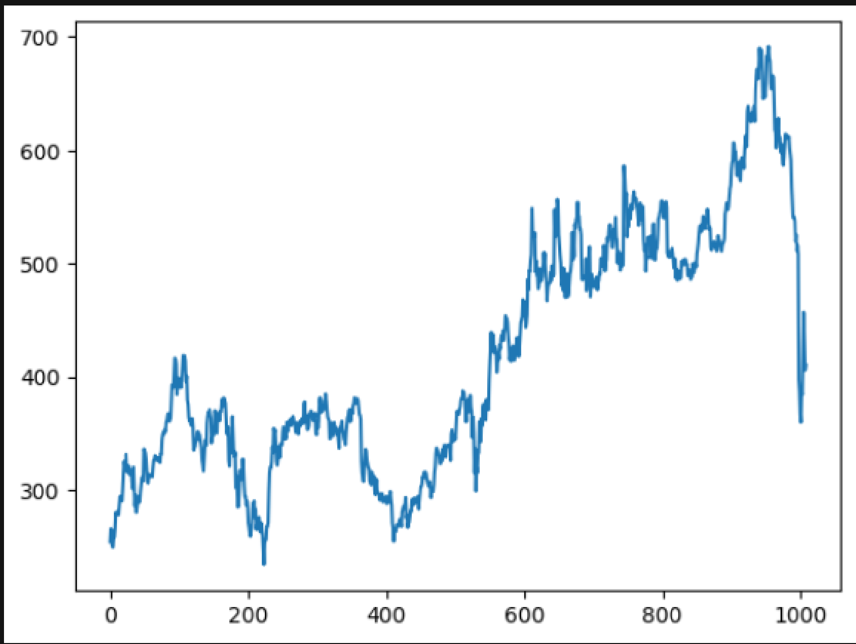


The Adjusted Close Value is the final output value that will be forecasted using the Machine Learning model.

```
# Plot the True Adj Close Value
Data['Adj Close'].plot()
```

[12]

```
<AxesSubplot: >
```



DATA CLEANING

```
Data.isnull().values.any()  # Checking whether we have any missing values in dataset
```

[13]

```
False
```

```
Data.isnull().sum()
```

[14]

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

There were no missing values in the datasets. So, there was no replacement and missing values.

DATA STANDARDIZATION

```
Data.head()
```

[15]

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

```
Data['Volume'][:5]
```

[16]

```
0    11896100
1    12595800
2     8981500
3     9306700
4    16906900
Name: Volume, dtype: int64
```

```
scalar = StandardScaler(copy=True, with_mean=True, with_std=True)
Data["Volume"] = scalar.fit_transform(Data["Volume"].values.reshape(-1,1))
print ("After Standardisation: ")
Data.head()
```

[17]

After Standardisation:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	0.791791
1	2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	0.919875
2	2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	0.258257
3	2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	0.317787
4	2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	1.709045

DATA NORMALIZATION

```
norm = MinMaxScaler()
Data["Volume"] = norm.fit_transform(Data["Volume"].values.reshape(-1,1))
print ("After Normalisation: ")
Data.head()
```

[18]

After Normalisation:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	0.186150
1	2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	0.198264
2	2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	0.135690
3	2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	0.141320
4	2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	0.272902

Making data available for various ML models through normalization.

Discretization

```
Data['Adj Close'].unique()
```

[19]

```
Output exceeds the size limit. Open the full output data in a text editor
array([254.259995, 265.720001, 264.559998, 250.100006, 249.470001,
       257.950012, 258.269989, 266.      , 280.269989, 278.519989,
       278.549988, 281.040009, 278.140015, 285.929993, 294.160004,
       290.609985, 291.380005, 290.390015, 301.049988, 315.      ,
       325.220001, 321.160004, 317.      , 331.440002, 321.299988,
...
       591.150024, 567.52002 , 553.289978, 541.059998, 539.849976,
       540.840027, 537.219971, 519.200012, 525.690002, 510.799988,
       515.859985, 508.25    , 397.5     , 387.149994, 366.420013,
       359.700012, 386.700012, 384.359985, 427.140015, 457.130005,
       429.480011, 405.600006, 410.170013])
```

```
print(Data['Adj Close'].max())
print(Data['Adj Close'].min())
```

[20]

```
691.690002
233.880005
```

```
Data['bin_of_Adj Close'] = pd.cut(Data['Adj Close'], [200,300,400,500,600,700],
                                  labels = ['200-300', '300-400', '400-500', '500-600', '600-700'])
```

[21]

```
Data.groupby([Data["bin_of_Adj Close"]]).count()
```

[22]

bin_of_Adj Close	Date	Open	High	Low	Close	Adj Close	Volume
200-300	136	136	136	136	136	136	136
300-400	410	410	410	410	410	410	410
400-500	169	169	169	169	169	169	169
500-600	231	231	231	231	231	231	231
600-700	63	63	63	63	63	63	63

```
for column in Data.columns:
    print("------  " + column + "  ------")
    print(Data[column].value_counts())
```

[23]

```
Output exceeds the size limit. Open the full output data in a text editor
------  Date  ------
2018-02-05    1
2020-10-14    1
2020-09-25    1
2020-09-28    1
2020-09-29    1
```

```
              ..
2019-06-14    1
2019-06-17    1
2019-06-18    1
2019-06-19    1
2022-02-04    1
Name: Date, Length: 1009, dtype: int64
------  Open  ------
365.000000    4
359.000000    3
355.000000    3
295.000000    3
425.000000    2
              ..
378.290009    1
378.190002    1
379.059998    1
382.769989    1
407.309998    1
...
400-500    169
200-300    136
600-700     63
Name: bin_of_Adj Close, dtype: int64
```

Making the values group-wise and making continuous values discrete.

DATA SUMMARIZATION

```
[24]
print(Data.shape)
```

```
(1009, 8)
```

```
[25]
Data.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	1009.000000	1009.000000	1009.000000	1009.000000	1009.000000	1009.000000
mean	419.059673	425.320703	412.374044	419.000733	419.000733	0.111265
std	108.537532	109.262960	107.555867	108.289999	108.289999	0.094624
min	233.919998	250.649994	231.229996	233.880005	233.880005	0.000000
25%	331.489990	336.299988	326.000000	331.619995	331.619995	0.051037
50%	377.769989	383.010010	370.880005	378.670013	378.670013	0.082938
75%	509.130005	515.630005	502.529999	509.079987	509.079987	0.141592
max	692.349976	700.989990	686.090027	691.690002	691.690002	1.000000

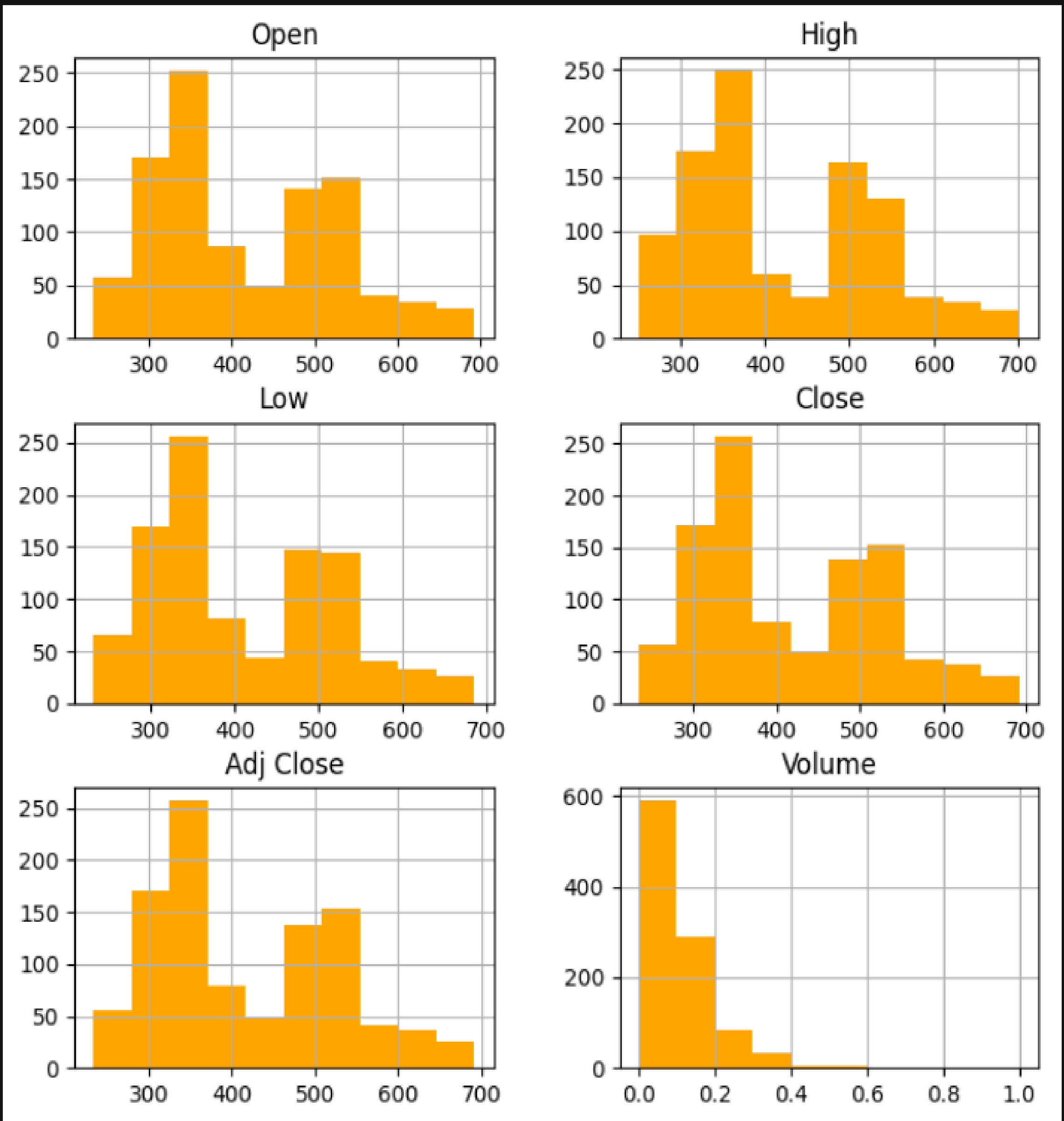
DATA VISUALIZATION

Histogram

```
Data.hist(color = "orange", figsize = (8,8))
plt.show()
```

[26]

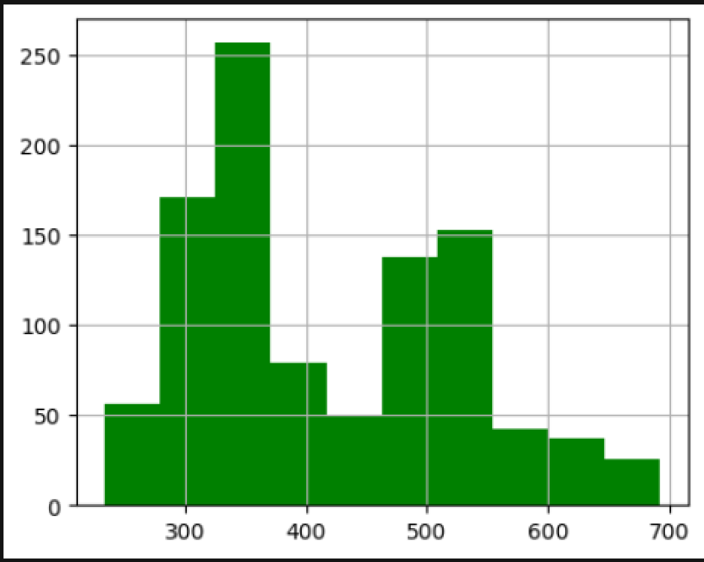
...



```
Data['Close'].hist(color = "green", figsize = (5,4))
```

[27]

```
<AxesSubplot: >
```

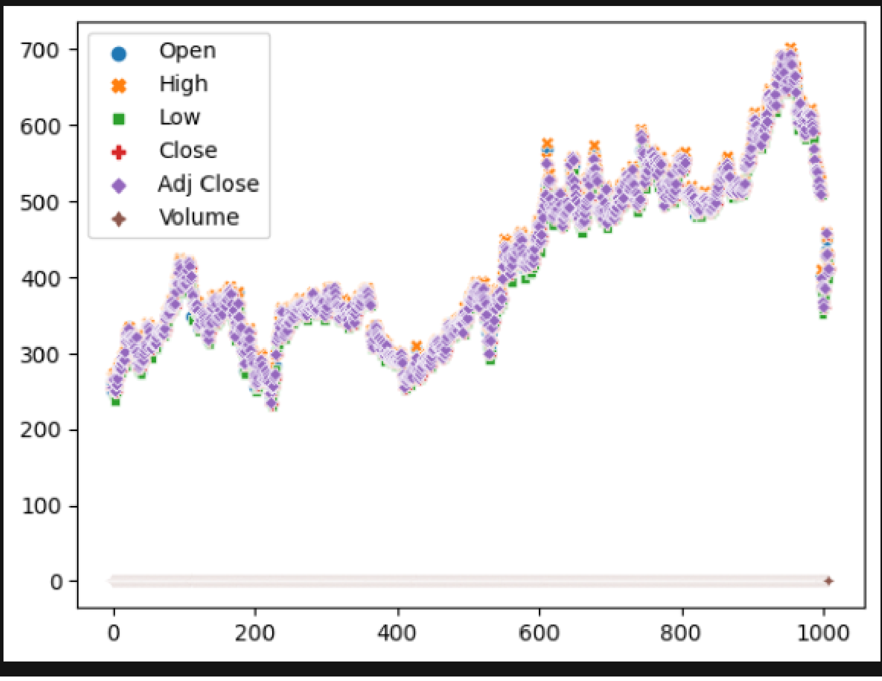


Scatter Plot

```
sns.scatterplot(Data)
```

[28]

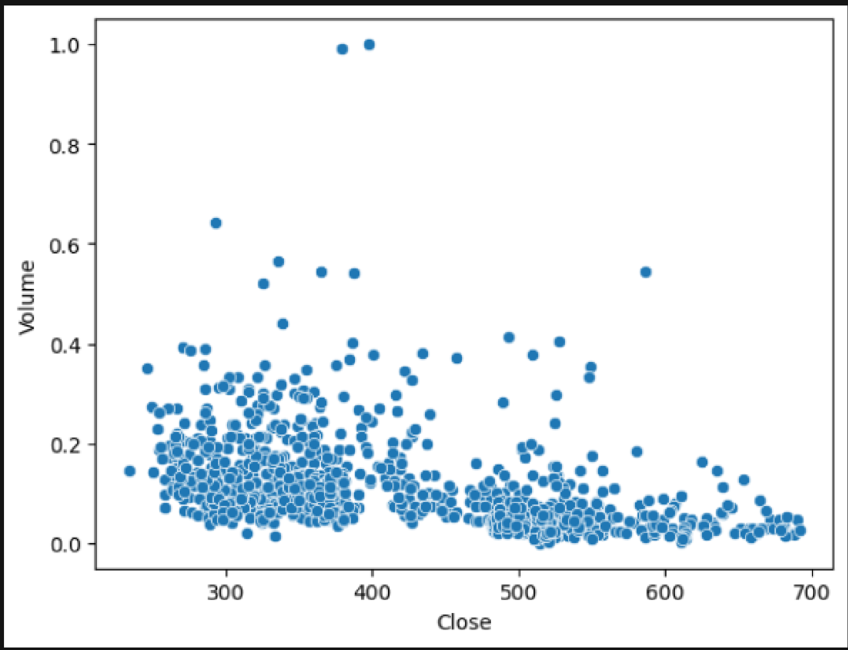
```
<AxesSubplot: >
```



```
sns.scatterplot(x = Data['Close'], y = Data['Volume'])
```

[29]

```
<AxesSubplot: xlabel='Close', ylabel='Volume'>
```

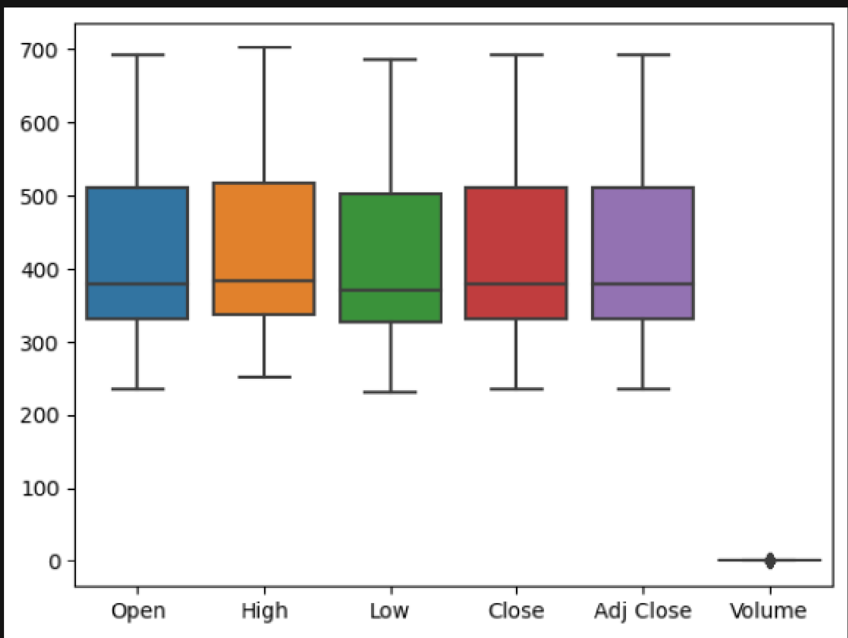


Box Plot

```
sns.boxplot(Data)
```

[30]

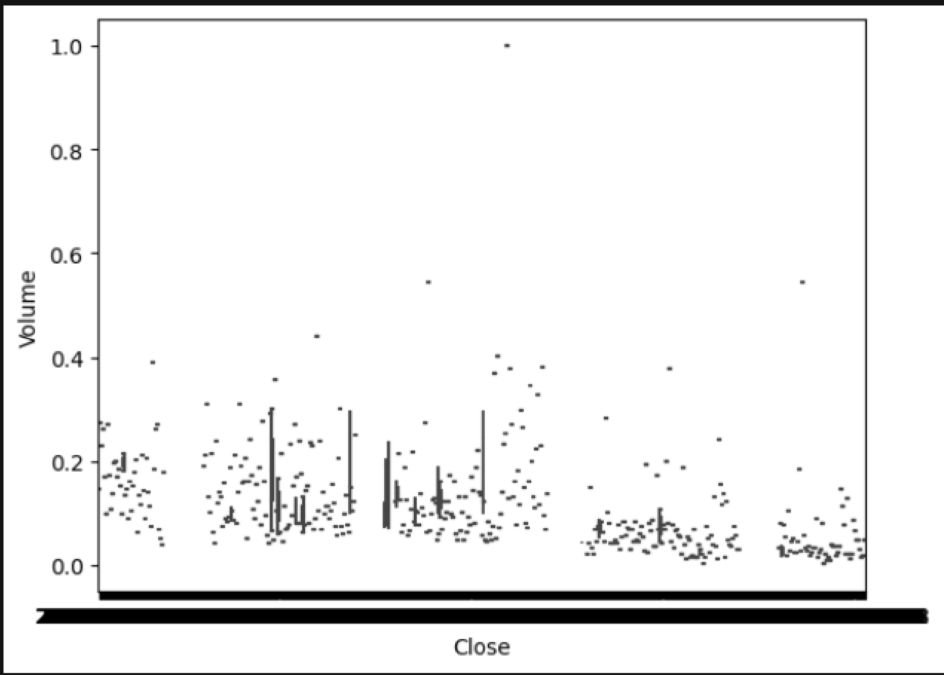
```
<AxesSubplot: >
```



```
sns.boxplot(x = 'Close', y = 'Volume', data=Data)
```

[31]

```
<AxesSubplot: xlabel='Close', ylabel='Volume'>
```



These are the various visualizations of data. Now we can use this data and apply it to various models.

<u>PHASE 2</u>

DATA MODELLING

Split your data into training, validation, and testing.

```
X = Data[["Open", "High", "Low", "Volume"]]
y = Data["Close"]

# Import the train_test_split function
from sklearn.model_selection import train_test_split

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Split the training set further into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2)

print("Train Size: " + str(X_train.shape[0]))
print("Test Size: " + str(X_test.shape[0]))
print("Validation Size: " + str(X_val.shape[0]))
```

[32]

```
Train Size: 645
Test Size: 202
Validation Size: 162
```

1. LINEAR REGRESSION (LR)

- Linear regression is a statistical method used to model the relationship between a dependent variable (in this case, stock price) and one or more independent variables (in this case, potential factors that may influence stock price). By fitting a linear regression model to historical data, we can attempt to predict future stock prices based on the relationship between the dependent and independent variables.

Model Implementation

```
from sklearn.linear_model import LinearRegression

# Create a linear regression model
lr_model = LinearRegression()

# Fit the model to the data
lr_model.fit(X_train, y_train)

# Use the model to make predictions
y_pred_lr = lr_model.predict(X_test)

lr_score_test = lr_model.score(X_test, y_test)
lr_score_train = lr_model.score(X_train, y_train)

# Print the evaluation score
print('Test Data Acurracy:', lr_score_test)
print('Train Data Acurracy:', lr_score_train)
```

[33]

```
Test Data Acurracy: 0.9988210726468153
Train Data Acurracy: 0.9987263351602502
```

Accuracy

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

kf = KFold(n_splits = 5)
scores = cross_val_score(lr_model, X, y, cv = kf)
print("Average Accuracy Using KFold:", scores.mean())
```

[34]

```
Average Accuracy Using KFold: 0.9893231234396737
```

Mean Absolute Error

```
from sklearn.metrics import mean_absolute_error
lr_mae = mean_absolute_error(y_test, y_pred_lr)
print('Mean Absolute Error:', lr_mae)
```

[35]

```
Mean Absolute Error: 2.8847294631532154
```

2. KTH NEAREST NEIGHBOUR (KNN)

- KNN, or k-nearest neighbors, is a machine learning algorithm that can be used for a variety of purposes, including stock price prediction. In the context of stock price prediction, the KNN algorithm would take historical data on the prices of a particular stock as well as other relevant factors (such as the overall performance of the stock market, the performance of competing stocks, and macroeconomic indicators) and use this data to make predictions about future stock prices.

Model Implementation

```
from sklearn.neighbors import KNeighborsRegressor

# Create a KNN model
knn_model = KNeighborsRegressor(n_neighbors = 5)

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

# Use the model to make predictions
y_pred_knn = knn_model.predict(X_test)

knn_score_test = knn_model.score(X_test, y_test)
knn_score_train = knn_model.score(X_train, y_train)

# Print the evaluation score
print('Test Data Acurracy:', knn_score_test)
print('Train Data Acurracy:', knn_score_train)
```

[36]

```
Test Data Acurracy: 0.9978202403332014
Train Data Acurracy: 0.9984142837324946
```

Accuracy

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

kf = KFold(n_splits = 5)
scores = cross_val_score(knn_model, X, y, cv = kf)
print("Average Accuracy Using KFold:", scores.mean())
```

[37]

```
Average Accuracy Using KFold: 0.9008601196897315
```

Mean Absolute Error

```
from sklearn.metrics import mean_absolute_error
knn_mae = mean_absolute_error(y_test, y_pred_knn)
print('Mean Absolute Error:', knn_mae)
```

[38]

```
Mean Absolute Error: 3.9163471653465396
```

3. Support Vector Machine (SVM)

- Support vector machines (SVMs) are a type of supervised learning algorithm that can be used for classification or regression tasks. In the context of stock price prediction, an SVM could be used to classify whether the price of a stock will go up or down based on historical data. The SVM algorithm works by finding the best line or hyperplane that separates the data into different classes, allowing it to make predictions on new data based on this line. While SVMs are not the most commonly used algorithm for stock price prediction, they can be effective in certain cases.

Model Implementation

```python
from sklearn.svm import SVR

# Create a SVM regression model
svm_model = SVR(kernel="rbf", C=1.0, epsilon=0.1)

# Fit the model to the data
svm_model.fit(X_train, y_train)

# Use the model to make predictions
y_pred_svm = svm_model.predict(X_test)

svm_score_test = svm_model.score(X_test, y_test)
svm_score_train = svm_model.score(X_train, y_train)

# Print the evaluation score
print('Test Data Acurracy:',svm_score_test)
print('Train Data Acurracy:',svm_score_train)
```

[39]

```
Test Data Acurracy: 0.9141561209362162
Train Data Acurracy: 0.9232107816365819
```

Accuracy

```python
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

kf = KFold(n_splits = 5)
scores = cross_val_score(svm_model, X, y, cv = kf)
print("Average Accuracy Using KFold:", scores.mean())
```

[40]

```
Average Accuracy Using KFold: 0.5246008800923047
```

Mean Absolute Error

```python
from sklearn.metrics import mean_absolute_error
svm_mae = mean_absolute_error(y_test, y_pred_svm)
print('Mean Absolute Error:', svm_mae)
```

[41]

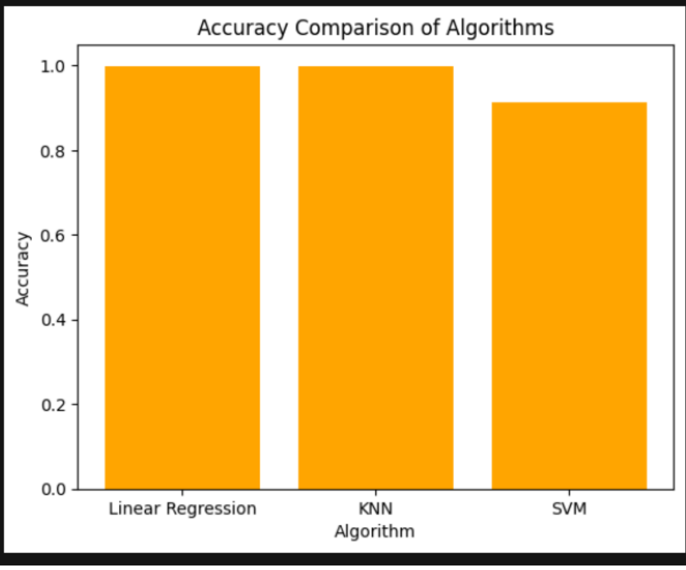
```
Mean Absolute Error: 18.903773568622483
```

COMPARISON PLOTS

Accuracy vs. Algorithm Plot

```
plt.bar(["Linear Regression", "KNN", "SVM"], [lr_score_test, knn_score_test, svm_score_test], color = "orange")
plt.xlabel("Algorithm")
plt.ylabel("Accuracy")
plt.title("Accuracy Comparison of Algorithms")
plt.show()
```

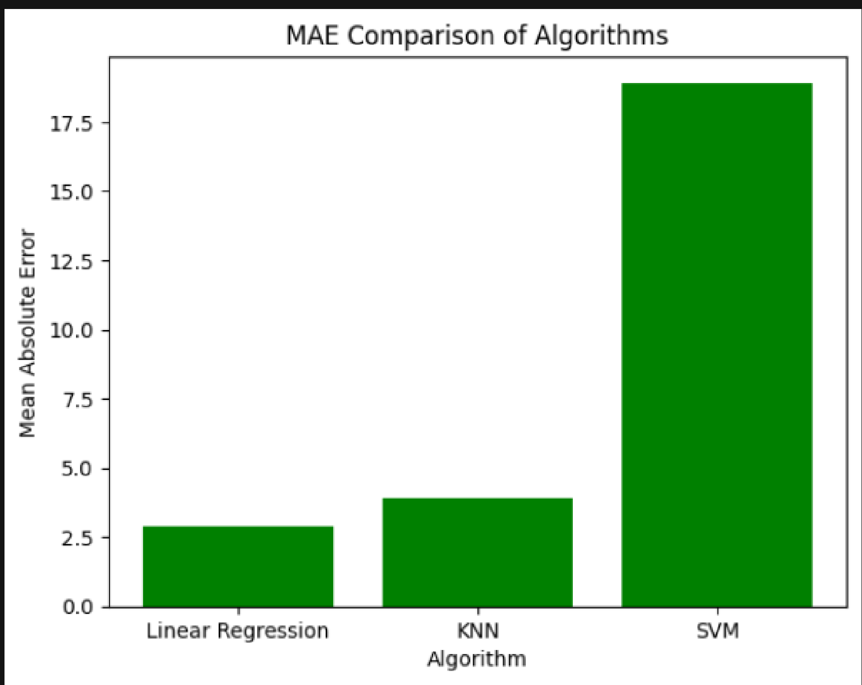
[42]



Error vs. Algorithm Plot

```
plt.bar(["Linear Regression", "KNN", "SVM"], [lr_mae, knn_mae, svm_mae], color = "green")
plt.xlabel("Algorithm")
plt.ylabel("Mean Absolute Error")
plt.title("MAE Comparison of Algorithms")
plt.show()
```

[43]



CONCLUSION

So, for all Three Datasets, we can see that the Linear Regression Algorithm performs with better accuracy followed by KNN and lastly the SVM algorithm. Although, this might not be the case every time with all sorts of Datasets. The appropriate machine learning model for a given task depends on a variety of factors, including the size and quality of the available data, the nature of the prediction task, and the desired level of accuracy. In some cases, linear regression may be a good choice for stock price prediction, while in others, KNN or SVM may be more appropriate. It is important to carefully evaluate the strengths and weaknesses of each model and select the one that is best suited to the specific prediction task at hand.

Furthermore, this was our ML Project namely **Stock Price Prediction** through which we aim to incentivize more people to know and utilize the benefits of the stock market with ease and eventually promote investing as a concept for the masses.

*****