Project Name: Stock Price Predictor v2

Goal: To predict the price of a Stock using the Date

Dataset: Kaggle Stock Price Trend Prediction Dataset
(https://www.kaggle.com/datasets/aumashe/stock-ew)

About the Project: Stock Market is a widespread trading platform open to all interested parties across the world, and is thus an incredible source of earning, and also a occupation for many
Trading can both cause profits and losses on a large scale, so it is necessary to understand the trends well to succeed in making a profit out of it, but it is obviously a challenging task for perform for a singular person.
But its a different story if one utilizes technology to provide assistance.

To facilitate this, we design a Machine Learning Model which will be provided adequate time-series data regarding various stocks, and will be used by us to provide assistance in recognising the market trends from based on the current scenario, provided as input.
This will help the user to make a profit with a lot lesser effort and higher efficiency.

Section 1: Collecting the data

First of all, we import the necessary libraries, and then proceed to import the concerned dataset

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

In [2]:
```
data = pd.read_csv(r"C:\Users\goura\Desktop\Data Science\Datasets\Stock Price -
data_copy = data
data
```

Out[2]:

	Date	Open	High	Low	Close	Adj_Close	Volume
0	2000/3/27	3.812500	4.156250	3.812500	4.125000	4.125000	3675600
1	2000/3/28	4.125000	4.125000	4.000000	4.015625	4.015625	1077600
2	2000/3/29	4.000000	4.031250	3.953125	4.000000	4.000000	437200
3	2000/3/30	4.000000	4.000000	3.843750	3.843750	3.843750	1883600
4	2000/3/31	3.734375	3.734375	3.390625	3.390625	3.390625	7931600
...	...	...	...	...	...	...	...
4387	2017/9/1	113.790001	114.099998	112.790001	113.309998	113.309998	950000
4388	2017/9/5	112.519997	113.529999	111.160004	111.870003	111.870003	1805200
4389	2017/9/6	112.029999	112.489998	110.250000	112.230003	112.230003	2136700
4390	2017/9/7	112.459999	112.900002	112.000000	112.339996	112.339996	1251600
4391	2017/9/8	112.300003	114.790001	112.010002	113.190002	113.190002	1611700

4392 rows × 7 columns

In [3]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4392 entries, 0 to 4391
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       4392 non-null   object
 1   Open       4392 non-null   float64
 2   High       4392 non-null   float64
 3   Low        4392 non-null   float64
 4   Close      4392 non-null   float64
 5   Adj_Close  4392 non-null   float64
 6   Volume     4392 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 240.3+ KB
```

In [4]:

```
data.describe()
```

Out[4]:

	Open	High	Low	Close	Adj_Close	Volume
count	4392.000000	4392.000000	4392.000000	4392.000000	4392.000000	4.392000e+03
mean	30.562539	30.893618	30.238833	30.572580	30.572580	1.884027e+06
std	29.914758	30.210974	29.615761	29.905778	29.905778	1.621609e+06
min	3.296875	3.390625	3.000000	3.250000	3.250000	1.904000e+05
25%	8.718125	8.803125	8.625000	8.712500	8.712500	1.088800e+06
50%	14.766250	14.981250	14.662500	14.767500	14.767500	1.539300e+06
75%	42.546248	43.051249	42.086249	42.539999	42.539999	2.188900e+06
max	121.080002	121.750000	120.169998	121.360001	121.360001	4.641260e+07

Section 2: Data Manipulation pt.1

Since the datatype of 'Date' is showing up to be 'object', we should change it to 'datetype' for convenience

In [5]:
```
data['Date'] = pd.to_datetime(data['Date'])
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4392 entries, 0 to 4391
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       4392 non-null   datetime64[ns]
 1   Open       4392 non-null   float64
 2   High       4392 non-null   float64
 3   Low        4392 non-null   float64
 4   Close      4392 non-null   float64
 5   Adj_Close  4392 non-null   float64
 6   Volume     4392 non-null   int64
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 240.3 KB
```

To think about it, the entire date wont be helpful in our prediction.
Each set of information broken down to simpler pieces is much more worth.

In [6]:
```
data['Year'] = data['Date'].dt.year
data['Month'] = data['Date'].dt.month
data['Day'] = data['Date'].dt.day
```

In [7]:
```
columns = [ i for i in data.columns if i!="Date"]
data = data[columns]
data
```

Out[7]:

	Open	High	Low	Close	Adj_Close	Volume	Year	Mon
0	3.812500	4.156250	3.812500	4.125000	4.125000	3675600	2000	
1	4.125000	4.125000	4.000000	4.015625	4.015625	1077600	2000	
2	4.000000	4.031250	3.953125	4.000000	4.000000	437200	2000	
3	4.000000	4.000000	3.843750	3.843750	3.843750	1883600	2000	
4	3.734375	3.734375	3.390625	3.390625	3.390625	7931600	2000	
...	...	...	...	...	...	...	...	
4387	113.790001	114.099998	112.790001	113.309998	113.309998	950000	2017	
4388	112.519997	113.529999	111.160004	111.870003	111.870003	1805200	2017	
4389	112.029999	112.489998	110.250000	112.230003	112.230003	2136700	2017	
4390	112.459999	112.900002	112.000000	112.339996	112.339996	1251600	2017	
4391	112.300003	114.790001	112.010002	113.190002	113.190002	1611700	2017	

4392 rows × 9 columns

The initial setup of our data has been completed.

Section 3: Problem Formulation

Since our goal to predict the closing price of the stock on any given day, there may be variable amount of imformation available surrounding the stock.
For example, if we want to predict the cost of the stock on a day 10 days from now, or maybe 10 months/years, We wont have the data on attributes like

- Open
- High
- Low
- Adj_Close
- Volume

Lets us have a look which of these have an impact on our target attribute, using corr_matrix

In [8]:
```
corr_matrix = data.corr()
corr_matrix
```

Out[8]:

	Open	High	Low	Close	Adj_Close	Volume	Year	
Open	1.000000	0.999907	0.999899	0.999806	0.999806	0.048770	0.866331	-
High	0.999907	1.000000	0.999874	0.999909	0.999909	0.051444	0.866572	-
Low	0.999899	0.999874	1.000000	0.999912	0.999912	0.045101	0.866187	-
Close	0.999806	0.999909	0.999912	1.000000	1.000000	0.047917	0.866529	-
Adj_Close	0.999806	0.999909	0.999912	1.000000	1.000000	0.047917	0.866529	-
Volume	0.048770	0.051444	0.045101	0.047917	0.047917	1.000000	0.133137	-
Year	0.866331	0.866572	0.866187	0.866529	0.866529	0.133137	1.000000	-
Month	-0.011538	-0.011593	-0.011621	-0.011749	-0.011749	-0.046754	-0.063439	
Day	0.000012	-0.000202	-0.000230	-0.000629	-0.000629	0.023683	-0.006832	-

As it can be seen, Volume doesn't correlate much with any other parameters, so it can be dropped.Same goes for Day
And, Adj_Close is practically the same as Close, which is also supported by the correlation coefficient of 1. Hence, we drop it too.

In [9]:
```
columns.remove('Volume')
columns.remove('Adj_Close')
columns.remove('Day')
```

In [10]:
```
data = data[columns]
data
```

Out[10]:

	Open	High	Low	Close	Year	Month
0	3.812500	4.156250	3.812500	4.125000	2000	3
1	4.125000	4.125000	4.000000	4.015625	2000	3
2	4.000000	4.031250	3.953125	4.000000	2000	3
3	4.000000	4.000000	3.843750	3.843750	2000	3
4	3.734375	3.734375	3.390625	3.390625	2000	3
...	...	...	...	...	...	...
4387	113.790001	114.099998	112.790001	113.309998	2017	9
4388	112.519997	113.529999	111.160004	111.870003	2017	9
4389	112.029999	112.489998	110.250000	112.230003	2017	9
4390	112.459999	112.900002	112.000000	112.339996	2017	9
4391	112.300003	114.790001	112.010002	113.190002	2017	9

4392 rows × 6 columns

Now, lets have a look at the data that will be provided.

- For Next day
  - Open
  - Year
  - Month
- For Any day beyond tomorrow
  - Year
  - Month

So, what we have to do it prepare for multiple models to predict unavailable data, and use those to fit into the final model which will predict the Closing Price for us.

Here is the list of models

- Open Price Predictor
- High Price Predictor
- Low Price Predictor
- Close Price Predictor

We can use the same model multiple times, by fitting different data each time, but that may create confusion.
So, we stick to creating a model for each. Which also means, we will have to create multiple sub-datasets from the original.

Let's proceed with the current approach.

Section 4: Data Manipulation pt.2

Lets start with creating the appropriate sub-datasets

In [11]:
```
open_columns = ['Year','Month','Open']
high_columns = ['Year','Month','Open','High']
low_columns = ['Year','Month','Open','High','Low']
close_columns = columns
```

In [12]:
```
open_data = data[open_columns]
close_data=data[close_columns]
high_data = data[high_columns]
low_data = data[low_columns]
close_data
```

Out[12]:

	Open	High	Low	Close	Year	Month
0	3.812500	4.156250	3.812500	4.125000	2000	3
1	4.125000	4.125000	4.000000	4.015625	2000	3
2	4.000000	4.031250	3.953125	4.000000	2000	3
3	4.000000	4.000000	3.843750	3.843750	2000	3
4	3.734375	3.734375	3.390625	3.390625	2000	3
...	...	...	...	...	...	...
4387	113.790001	114.099998	112.790001	113.309998	2017	9
4388	112.519997	113.529999	111.160004	111.870003	2017	9
4389	112.029999	112.489998	110.250000	112.230003	2017	9
4390	112.459999	112.900002	112.000000	112.339996	2017	9
4391	112.300003	114.790001	112.010002	113.190002	2017	9

4392 rows × 6 columns

We have prepared the datasets required for each model.

We have another thing to consider.
Which arrangement would provide us with more accurate results?

- fitting lower level predictions (open, high etc) to make high level prediction (close)
- predicting Close with the features available directly.

Lets first try out the naive (first) approach

In [13]:
```
open_features = ['Year','Month']
open_label = ['Open']
```

In [14]:
```
from sklearn.model_selection import train_test_split
open_train_set, open_test_set = train_test_split(open_data, test_size=0.2, randc
open_X_train = open_train_set[open_features]
open_X_test = open_test_set[open_features]
open_Y_train = open_train_set[open_label]
open_Y_test = open_test_set[open_label]
```

In [15]:
```
open_X_test
```

Out[15]:

	Year	Month
2744	2011	2
2306	2009	5
1166	2004	11
98	2000	8
2403	2009	10
...	...	...
3196	2012	12
1358	2005	8
2881	2011	9
4072	2016	6
1630	2006	9

879 rows × 2 columns

In [16]:
```
open_Y_train = open_Y_train.values.ravel()
open_Y_test = open_Y_test.values.ravel()
```

In [17]:
```
high_features = ['Year','Month','Open']
high_label = ['High']
```

In [18]:
```
high_train_set, high_test_set = train_test_split(high_data, test_size=0.2, randc
high_X_train = high_train_set[high_features]
high_X_test = high_test_set[high_features]
high_Y_train = high_train_set[high_label]
high_Y_test = high_test_set[high_label]
```

In [19]:
```
high_Y_train = high_Y_train.values.ravel()
high_Y_test = high_Y_test.values.ravel()
```

In [20]:
```
low_features = ['Year','Month','Open','High']
low_label = ['Low']
```

In [21]:
```
low_train_set, low_test_set = train_test_split(low_data, test_size=0.2, random_s
low_X_train = low_train_set[low_features]
low_X_test = low_test_set[low_features]
low_Y_train = low_train_set[low_label]
low_Y_test = low_test_set[low_label]
```

In [22]:
```
low_Y_train = low_Y_train.values.ravel()
low_Y_test = low_Y_test.values.ravel()
```

In [23]:
```
close_features = ['Year','Month','Open','High','Low']
close_label = ['Close']
```

In [24]:
```
close_train_set, close_test_set = train_test_split(close_data, test_size=0.2, ra
close_X_train = close_train_set[close_features]
```

```
close_X_test = close_test_set[close_features]
close_Y_train = close_train_set[close_label]
close_Y_test = close_test_set[close_label]
```

In [25]:
```
close_Y_train = close_Y_train.values.ravel()
close_Y_test = close_Y_test.values.ravel()
```

We have created a lot of sub-datasets (16), lets review them.

For each prediction "name"

- name_X_train
- name_Y_train
- name_X_test
- name_Y_test

Section 5: Model Creation

Lets start by defining the models

We will use the **RandomForestRegressor** Ensemble model for our models, and use **MeanAbsoluteError** for measuring the errors.

In [26]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [27]:
```
open_model = RandomForestRegressor()
open_model.fit(open_X_train,open_Y_train)
open_preds = open_model.predict(open_X_test)
```

In [28]:
```
mae = np.mean((abs(open_preds - open_Y_test)))
print("MAE", mae)
```

```
MAE 0.7121476437083224
```

In [29]:
```
temp_df=open_test_set
temp_df
```

Out[29]:

	Year	Month	Open
2744	2011	2	43.544998
2306	2009	5	15.962500
1166	2004	11	9.000000
98	2000	8	5.640625
2403	2009	10	17.400000
...	...	...	...
3196	2012	12	45.529999
1358	2005	8	10.775000
2881	2011	9	37.110001
4072	2016	6	102.080002
1630	2006	9	11.675000

879 rows × 3 columns

In [30]:
```
def accuracy (A,B):
    return (1 - (A/B))*100
```

In [31]:
```
accuracy(mae*879,(temp_df['Open']).sum())
```

Out[31]: 97.68396040602518

As we can see, the accuracy of our random forest regressor is 97% even for the test data. Which means, this model is ready to be used.

Lets move on to the creation of our "High" determining model using similar steps.

In [32]:
```
high_model = RandomForestRegressor()
high_model.fit(high_X_train,high_Y_train)
high_preds = high_model.predict(high_X_test)
```

In [33]:
```
mae = np.mean((abs(high_preds - high_Y_test)))
print("MAE", mae)
```

```
MAE 0.2432261782555503
```

In [34]:
```
temp_df=high_test_set
temp_df
```

Out[34]:

	Year	Month	Open	High
2744	2011	2	43.544998	43.715000
2306	2009	5	15.962500	16.000000
1166	2004	11	9.000000	9.092500
98	2000	8	5.640625	5.718750
2403	2009	10	17.400000	17.549999
...	...	...	...	...
3196	2012	12	45.529999	46.665001
1358	2005	8	10.775000	10.800000
2881	2011	9	37.110001	37.465000
4072	2016	6	102.080002	102.190002
1630	2006	9	11.675000	11.710000

879 rows × 4 columns

In [35]:
```
accuracy(mae*879,(temp_df['High']).sum())
```

Out[35]: 99.21719602657724

This too has a great accuracy! Lets move on to the next, the model for predicting "Low" attribute.

In [36]:
```
low_model = RandomForestRegressor()
low_model.fit(low_X_train,low_Y_train)
low_preds = low_model.predict(low_X_test)
```

In [37]:
```
mae = np.mean((abs(low_preds - low_Y_test)))
print("MAE", mae)
```

```
MAE 0.20874616533926563
```

In [38]:
```
temp_df=low_test_set
temp_df
```

Out[38]:

	Year	Month	Open	High	Low
2744	2011	2	43.544998	43.715000	42.090000
2306	2009	5	15.962500	16.000000	15.612500
1166	2004	11	9.000000	9.092500	8.955000
98	2000	8	5.640625	5.718750	5.578125
2403	2009	10	17.400000	17.549999	17.342501
...	...	...	...	...	...
3196	2012	12	45.529999	46.665001	45.445000
1358	2005	8	10.775000	10.800000	10.615000
2881	2011	9	37.110001	37.465000	36.514999
4072	2016	6	102.080002	102.190002	100.389999
1630	2006	9	11.675000	11.710000	11.632500

879 rows × 5 columns

In [39]:
```
accuracy(mae*879,(temp_df['Low']).sum())
```

Out[39]:
```
99.31396764944137
```

We're going good till now.
Proceeding to the final step of this sequence, the "Close" attribute predictor

In [40]:
```
close_model = RandomForestRegressor()
close_model.fit(close_X_train,close_Y_train)
close_preds = close_model.predict(close_X_test)
```

In [41]:
```
mae = np.mean((abs(close_preds - close_Y_test)))
print("MAE", mae)
```

```
MAE 0.19499594218240401
```

In [42]:
```
temp_df=close_test_set
temp_df
```

Out[42]:

	Open	High	Low	Close	Year	Month
2744	43.544998	43.715000	42.090000	42.570000	2011	2
2306	15.962500	16.000000	15.612500	15.960000	2009	5
1166	9.000000	9.092500	8.955000	8.975000	2004	11
98	5.640625	5.718750	5.578125	5.703125	2000	8
2403	17.400000	17.549999	17.342501	17.549999	2009	10
...	...	...	...	...	...	...
3196	45.529999	46.665001	45.445000	46.040001	2012	12
1358	10.775000	10.800000	10.615000	10.780000	2005	8
2881	37.110001	37.465000	36.514999	36.540001	2011	9
4072	102.080002	102.190002	100.389999	101.529999	2016	6
1630	11.675000	11.710000	11.632500	11.675000	2006	9

879 rows × 6 columns

In [43]:
```
accuracy(mae*879,(temp_df['Close']).sum())
```

Out[43]: 99.36614927550356

Great. All four of our model's have been extremely accurate. We can follow the chain of

- **Step 1**: Take date as input
- **Step 2**: Predict the open price using model 1,
- **Step 3**: Predict the high price using model 2, using date and prediction from model 1.
- **Step 4**: Predict the low price using model 3, using date, prediction from model 1,2.
- **Step 5**: Predict the close price using model 4, using date, prediction from model 1,2,3.

Theres an alternate method too, hinted a while ago.
That would be,

Directly predicting the Close price using the date as input (and also the opening price in some cases)

In [44]:
```
dd_model = RandomForestRegressor()
dd_model.fit(open_X_train,close_Y_train)
preds = dd_model.predict(open_X_test)
```

In [45]:
```
mae = np.mean((abs(preds - close_Y_test)))
print("MAE", mae)
```

```
MAE 0.6851918142441464
```

In [46]:
```
temp_df=close_test_set.copy()
temp_df
```

Out[46]:

	Open	High	Low	Close	Year	Month
2744	43.544998	43.715000	42.090000	42.570000	2011	2
2306	15.962500	16.000000	15.612500	15.960000	2009	5
1166	9.000000	9.092500	8.955000	8.975000	2004	11
98	5.640625	5.718750	5.578125	5.703125	2000	8
2403	17.400000	17.549999	17.342501	17.549999	2009	10
...	...	...	...	...	...	...
3196	45.529999	46.665001	45.445000	46.040001	2012	12
1358	10.775000	10.800000	10.615000	10.780000	2005	8
2881	37.110001	37.465000	36.514999	36.540001	2011	9
4072	102.080002	102.190002	100.389999	101.529999	2016	6
1630	11.675000	11.710000	11.632500	11.675000	2006	9

879 rows × 6 columns

In [47]:

```
accuracy(mae*879,(temp_df['Close']).sum())
```

Out[47]: 97.77272632949756

As we see, the direct model can provide us with **97.77%** accuracy.

Even this model provides a great accuracy! Now we have to check if we can combine the previously created 4 models and surpass the accuracy of this model.

Section 6: Model Combination

First of all, our "Open" model would take in the date and predict the opening price. Available Attributes: Year, Month

In [48]:

```
cdf = open_X_test.copy()
```

In [49]:

```
open_preds = open_model.predict(open_X_test)
```

In [50]:

```
cdf.loc[:, 'Open'] = open_preds
cdf
```

Out[50]:

	Year	Month	Open
2744	2011	2	43.617302
2306	2009	5	15.761983
1166	2004	11	8.968740
98	2000	8	5.700087
2403	2009	10	18.127387
...	...	...	...
3196	2012	12	45.466649
1358	2005	8	10.977857
2881	2011	9	36.705408
4072	2016	6	99.454314
1630	2006	9	11.654070

879 rows × 3 columns

Next, our "High" model would do the same, while utilizing the prediction of "Open" by the last model.

In [51]:

```
high_preds = high_model.predict(cdf)
```

In [52]:

```
cdf.loc[:, 'High'] = high_preds
cdf
```

Out[52]:

	Year	Month	Open	High
2744	2011	2	43.617302	43.907250
2306	2009	5	15.761983	15.969225
1166	2004	11	8.968740	9.059812
98	2000	8	5.700087	5.730042
2403	2009	10	18.127387	18.345724
...	...	...	...	...
3196	2012	12	45.466649	46.365351
1358	2005	8	10.977857	11.042475
2881	2011	9	36.705408	37.401450
4072	2016	6	99.454314	99.910102
1630	2006	9	11.654070	11.739525

879 rows × 4 columns

Next, our "Low" model would follow the same process

In [53]:
```
low_preds = low_model.predict(cdf)
```

In [54]:
```
cdf.loc[:, 'Low'] = low_preds
cdf
```

Out[54]:

	Year	Month	Open	High	Low
2744	2011	2	43.617302	43.907250	43.232350
2306	2009	5	15.761983	15.969225	15.595075
1166	2004	11	8.968740	9.059812	8.951275
98	2000	8	5.700087	5.730042	5.645651
2403	2009	10	18.127387	18.345724	17.767575
...	...	...	...	...	...
3196	2012	12	45.466649	46.365351	45.320199
1358	2005	8	10.977857	11.042475	10.911500
2881	2011	9	36.705408	37.401450	36.587400
4072	2016	6	99.454314	99.910102	98.308099
1630	2006	9	11.654070	11.739525	11.556975

879 rows × 5 columns

Next, our "Close" model would perform the final predictive step, while using all the information predicted by the previous models.

In [55]:
```
close_preds = close_model.predict(cdf)
```

In [56]:
```
cdf.loc[:, 'Close'] = close_preds
cdf
```

Out[56]:

	Year	Month	Open	High	Low	Close
2744	2011	2	43.617302	43.907250	43.232350	43.544800
2306	2009	5	15.761983	15.969225	15.595075	15.875625
1166	2004	11	8.968740	9.059812	8.951275	9.015675
98	2000	8	5.700087	5.730042	5.645651	5.712500
2403	2009	10	18.127387	18.345724	17.767575	18.074574
...	...	...	...	...	...	...
3196	2012	12	45.466649	46.365351	45.320199	45.849200
1358	2005	8	10.977857	11.042475	10.911500	11.015275
2881	2011	9	36.705408	37.401450	36.587400	37.225150
4072	2016	6	99.454314	99.910102	98.308099	98.712300
1630	2006	9	11.654070	11.739525	11.556975	11.661850

879 rows × 6 columns

Now that we have the predictions from the combination as we desires, lets test these values against the actual values.

In [57]:
```
mae = np.mean((abs(close_preds - close_Y_test)))
print("MAE", mae)
```

```
MAE 0.6962357372582482
```

In [58]:
```
close_test_set
```

Out[58]:

	Open	High	Low	Close	Year	Month
2744	43.544998	43.715000	42.090000	42.570000	2011	2
2306	15.962500	16.000000	15.612500	15.960000	2009	5
1166	9.000000	9.092500	8.955000	8.975000	2004	11
98	5.640625	5.718750	5.578125	5.703125	2000	8
2403	17.400000	17.549999	17.342501	17.549999	2009	10
...	...	...	...	...	...	...
3196	45.529999	46.665001	45.445000	46.040001	2012	12
1358	10.775000	10.800000	10.615000	10.780000	2005	8
2881	37.110001	37.465000	36.514999	36.540001	2011	9
4072	102.080002	102.190002	100.389999	101.529999	2016	6
1630	11.675000	11.710000	11.632500	11.675000	2006	9

879 rows × 6 columns

In [59]:
```
accuracy(mae*879,sum(close_Y_test))
```

Out[59]: 97.73682712808123

We have acheived an accuracy of **97.74%** by using the model combination.
While this model does have a good accuracy, we couldn't overtake the accuracy of the direct model.
The accuracy of both approaches is nearly the same. (**97.77%** & **97.74%**)

But, by using the naive appraoch, we have one key insight.
The accuracy of predicting Open Price using the date has an accuracy of 97.68

In [60]:
```
corr_matrix
```

Out[60]:

	Open	High	Low	Close	Adj_Close	Volume	Year	
Open	1.000000	0.999907	0.999899	0.999806	0.999806	0.048770	0.866331	-
High	0.999907	1.000000	0.999874	0.999909	0.999909	0.051444	0.866572	-
Low	0.999899	0.999874	1.000000	0.999912	0.999912	0.045101	0.866187	-
Close	0.999806	0.999909	0.999912	1.000000	1.000000	0.047917	0.866529	-
Adj_Close	0.999806	0.999909	0.999912	1.000000	1.000000	0.047917	0.866529	-
Volume	0.048770	0.051444	0.045101	0.047917	0.047917	1.000000	0.133137	-
Year	0.866331	0.866572	0.866187	0.866529	0.866529	0.133137	1.000000	-
Month	-0.011538	-0.011593	-0.011621	-0.011749	-0.011749	-0.046754	-0.063439	
Day	0.000012	-0.000202	-0.000230	-0.000629	-0.000629	0.023683	-0.006832	-

This may be the cause behind our error, cause according to the corr_matrix, the "Open" and "Close" price are highly correlated.
So, an error in prediction of "Open" will definietly affect the "Close" prediction.

Possible Approaches to Counter this:

- Using Different Model
- Using Combination of Models to finalize prediction
- Predicting a different attribute (since "High" has the highest correlation with "Year", predicting "High" first may lead to a better result)

We won't dwelve into these methods as for this project.
Rather, we try to improvise using predictions of our already established models.

Section 8: Prediction Finalization

In [61]:
```
preds1 = dd_model.predict(open_X_test)
preds2 = close_preds
```

In [62]:
```
sample_pred = [ ( 0.9777*i + 0.9774*j )/(0.9777 + 0.9774) for i,j in zip(preds1,
```

In [63]:
```
mae = np.mean((abs(sample_pred - close_Y_test)))
print("MAE", mae)
```

```
MAE 0.6866236883764479
```

In [64]:
```
accuracy(mae*879,sum(close_Y_test))
```

Out[64]:
```
97.76807190209773
```

As we can see, combining the predictions of doesn't particularly lead to improvement of highest possible accuracy.
Now,

- Combining both predictions doesnt lead to significant improvements.
- The model-combination is more prone to introducing noise to the prediction, than the direct model.
- The direct model has a higher accuracy even if consuming lesser data and time.

Even if having multiple predictions acts as a recitifier in many cases, using two predictions here means using 5 models instead of 1.
This increases the time consumption, which is unjustified considering the minimal advantages.

Thus, we should proceed with the use of only the direct model for Close Price.

Section 9: User-Interactive Space

We start by training our model on the entire dataset, so that it has more data to recognise relationships.
This will enhance the quality and versatility of the model

In [65]:
```
dd_model.fit(data[open_features],data['Close'])
```

Out[65]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [66]:
```
ques = [ i for i in open_features ]
c=0
while True:
    if c==0:
        print("Do you want to make any predictions?")
    elif c>0:
        print("Do you want to make any more predictions?")
    print("Enter 1 if Yes, else 0")
    a = int(input("Choice:"))
    if a!=0 and a!=1:
        print("Invalid Choice")
    elif a==0:
        print("Choice is No (0)")
        print("Exit Program")
```

```python
        if c>0:
            print("Thank you for using our services.")
        break
    elif a==1:
        print("Choice is Yes (1)")
        lst = []
        for i in ques:
            lst.append(float(input(f"Enter value of {i}: ")))
        df = pd.DataFrame([lst], columns=ques)
        pred = dd_model.predict(df)
        print("Reported Parameters: ",lst)
        print("The Closing Price is predicted to be:",pred)
        c+=1
```

```
Do you want to make any predictions?
Enter 1 if Yes, else 0
Choice is No (0)
Exit Program
```

Section 10: Conclusion

This marks the end of our project.

This program assists us in predicting the Closing Price of the stock based on the Date.
We acheived this via creating a RandomForestRegressor Model

We noticed that the maximum accuracy of predicting the Closing Price goes up to **97.8%**, for both training and testing data, thus we can conclude the model to be highly accurate for predicting new/unknown data.
Therefore, the Program is ready to be used.