STOCK TREND PREDICTION

Importing the libraries

In [232…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas_datareader import data as data
import yfinance as yf
from datetime import date
```

Yahoo!, Y!Finance, and Yahoo! finance are registered trademarks of Yahoo, Inc.

yfinance is not affiliated, endorsed, or vetted by Yahoo, Inc. It's an open-source tool that uses Yahoo's publicly available APIs, and is intended for research and educational purposes.

Here we will work only on AAPL stock.

In [233…

```
start = '2010-01-01'
end = date.today()

df = yf.download('AAPL', start, end)
df.tail()
```

```
[*********************100%***********************]  1 of 1 completed
```

Out[233]:

	Open	High	Low	Close	Adj Close	Volume
Date						
2023-05-08	172.479996	173.850006	172.110001	173.500000	173.260345	55962800
2023-05-09	173.050003	173.539993	171.600006	171.770004	171.532745	45326900
2023-05-10	173.020004	174.029999	171.899994	173.559998	173.320267	53724500
2023-05-11	173.850006	174.589996	172.169998	173.750000	173.510010	49514700
2023-05-12	173.619995	174.059998	171.000000	172.570007	172.570007	45497800

In [234…

```
df = df.reset_index()
df.head()
```

Out[234]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2010-01-04	7.622500	7.660714	7.585000	7.643214	6.496294	493729600
1	2010-01-05	7.664286	7.699643	7.616071	7.656429	6.507524	601904800
2	2010-01-06	7.656429	7.686786	7.526786	7.534643	6.404015	552160000
3	2010-01-07	7.562500	7.571429	7.466071	7.520714	6.392177	477131200
4	2010-01-08	7.510714	7.571429	7.466429	7.570714	6.434675	447610800

In [235…

```
df = df.drop(["Adj Close"], axis = 1)
df.head()
```

Out[235]:

	Date	Open	High	Low	Close	Volume
0	2010-01-04	7.622500	7.660714	7.585000	7.643214	493729600
1	2010-01-05	7.664286	7.699643	7.616071	7.656429	601904800
2	2010-01-06	7.656429	7.686786	7.526786	7.534643	552160000
3	2010-01-07	7.562500	7.571429	7.466071	7.520714	477131200
4	2010-01-08	7.510714	7.571429	7.466429	7.570714	447610800

Plotting a graph describling the closing price of AAPL stock from 2010 to 2023

Closing Price v/s Time

In [236…

```
plt.figure(figsize=(12,6))
plt.plot(df.Date,df.Close, label="Closing Price")
plt.title("Closing Price v/s Time")
plt.xlabel("Time")
plt.xlabel("Price")
plt.legend()
```

Out[236]: `<matplotlib.legend.Legend at 0x23d1024de40>`

In [237…

```
df
```

Out[237]:

	Date	Open	High	Low	Close	Volume
0	2010-01-04	7.622500	7.660714	7.585000	7.643214	493729600
1	2010-01-05	7.664286	7.699643	7.616071	7.656429	601904800
2	2010-01-06	7.656429	7.686786	7.526786	7.534643	552160000
3	2010-01-07	7.562500	7.571429	7.466071	7.520714	477131200
4	2010-01-08	7.510714	7.571429	7.466429	7.570714	447610800
...	...	...	...	...	...	...
3358	2023-05-08	172.479996	173.850006	172.110001	173.500000	55962800
3359	2023-05-09	173.050003	173.539993	171.600006	171.770004	45326900
3360	2023-05-10	173.020004	174.029999	171.899994	173.559998	53724500
3361	2023-05-11	173.850006	174.589996	172.169998	173.750000	49514700
3362	2023-05-12	173.619995	174.059998	171.000000	172.570007	45497800

3363 rows × 6 columns

Now we will plot a graph of Closing Price v/s Time with Moving Averages

What Is a Moving Average (MA)?

In finance, a moving average (MA) is a stock indicator commonly used in technical analysis. The reason for calculating the moving average of a stock is to help smooth out the price data by creating a constantly updated average price.

By calculating the moving average, the impacts of random, short-term fluctuations on the price of a stock over a specified time frame are mitigated. Simple moving averages (SMAs) use a simple arithmetic average of prices over some timespan, while exponential moving averages (EMAs) place greater weight on more recent prices than older ones over the time period.

Moving averages are calculated to identify the trend direction of a stock or to determine its support and resistance levels. It is a trend-following or lagging, indicator because it is based on past prices.

In this we are going to implement the Simple Moving Average (SMA)

Simple Moving Average

A simple moving average (SMA), is calculated by taking the arithmetic mean of a given set of values over a specified period. A set of numbers, or prices of stocks, are added together and then divided by the number of prices in the set. The formula for calculating the simple moving average of a security is as follows:

SMA = (A1+A2+A3+....+An)/n

where:

A=Average in period n

n=Number of time periods

In [238…

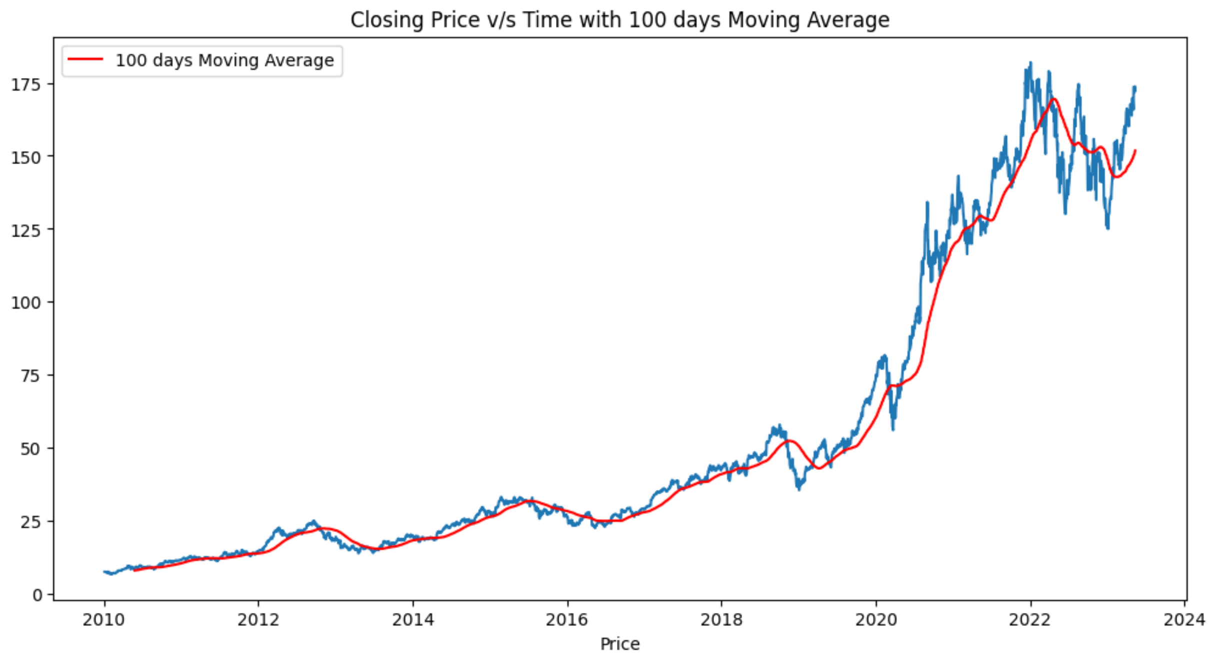
```
# TAking a 100 day moving average.
ma100 = df.Close.rolling(100).mean()
ma100
```

Out[238]:
```
0              NaN
1              NaN
2              NaN
3              NaN
4              NaN
           ...
3358    150.419900
3359    150.682900
3360    150.986399
3361    151.358899
3362    151.739500
Name: Close, Length: 3363, dtype: float64
```

Plotting a graph of Closing Price v/s Time with 100 Days Moving Averages

In [239…
```
plt.figure(figsize = (12,6))
plt.plot(df.Date,df.Close)
plt.plot(df.Date,ma100, color="r", label="100 days Moving Average")
plt.title("Closing Price v/s Time with 100 days Moving Average")
plt.xlabel("Time")
plt.xlabel("Price")
plt.legend()
```

Out[239]: <matplotlib.legend.Legend at 0x23d1c3c8af0>



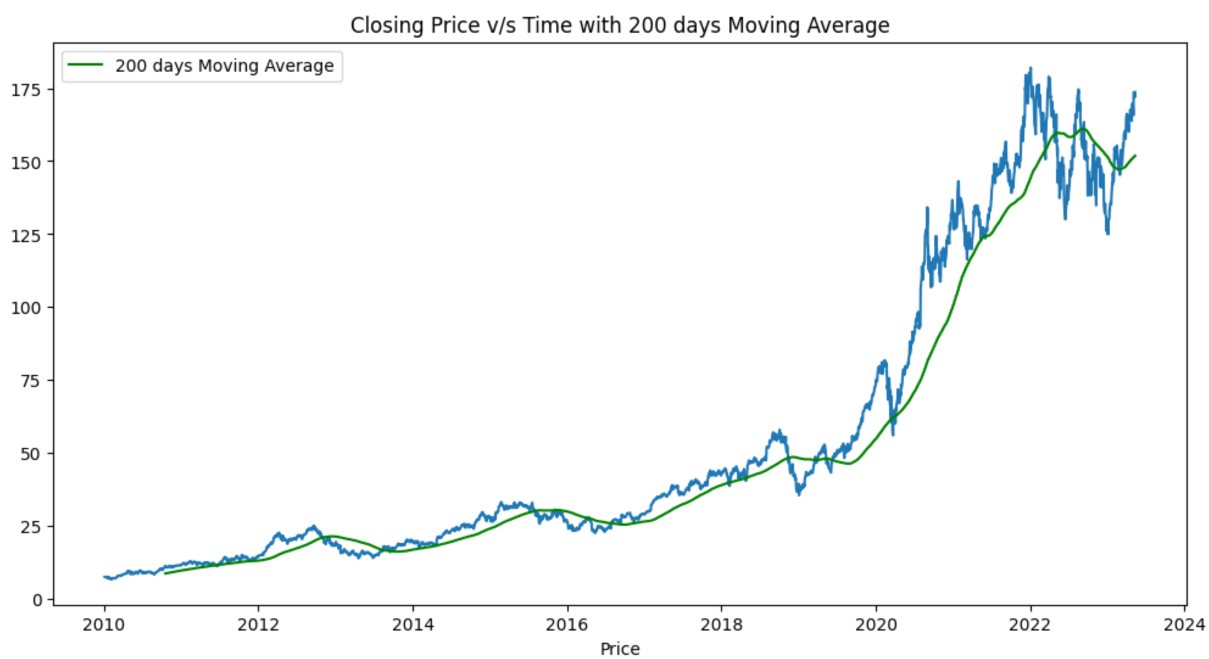
Plotting a graph of Closing Price v/s Time with 200 Days Moving Averages

In [240…
```
#Taking a 200 days moving Average
ma200 = df.Close.rolling(200).mean()
ma200
```

Out[240]:
```
0              NaN
1              NaN
2              NaN
3              NaN
4              NaN
           ...
3358    151.462949
3359    151.551350
3360    151.654400
3361    151.765149
3362    151.844050
Name: Close, Length: 3363, dtype: float64
```

In [241…
```
plt.figure(figsize = (12,6))
plt.plot(df.Date,df.Close)
plt.plot(df.Date,ma200, color="g", label="200 days Moving Average")
plt.title("Closing Price v/s Time with 200 days Moving Average")
plt.xlabel("Time")
plt.xlabel("Price")
plt.legend()
```

Out[241]:
```
<matplotlib.legend.Legend at 0x23d13377c70>
```



Plotting and Analysing graph of Closing Price v/s Time with 100 Days and 200 Days Moving Averages

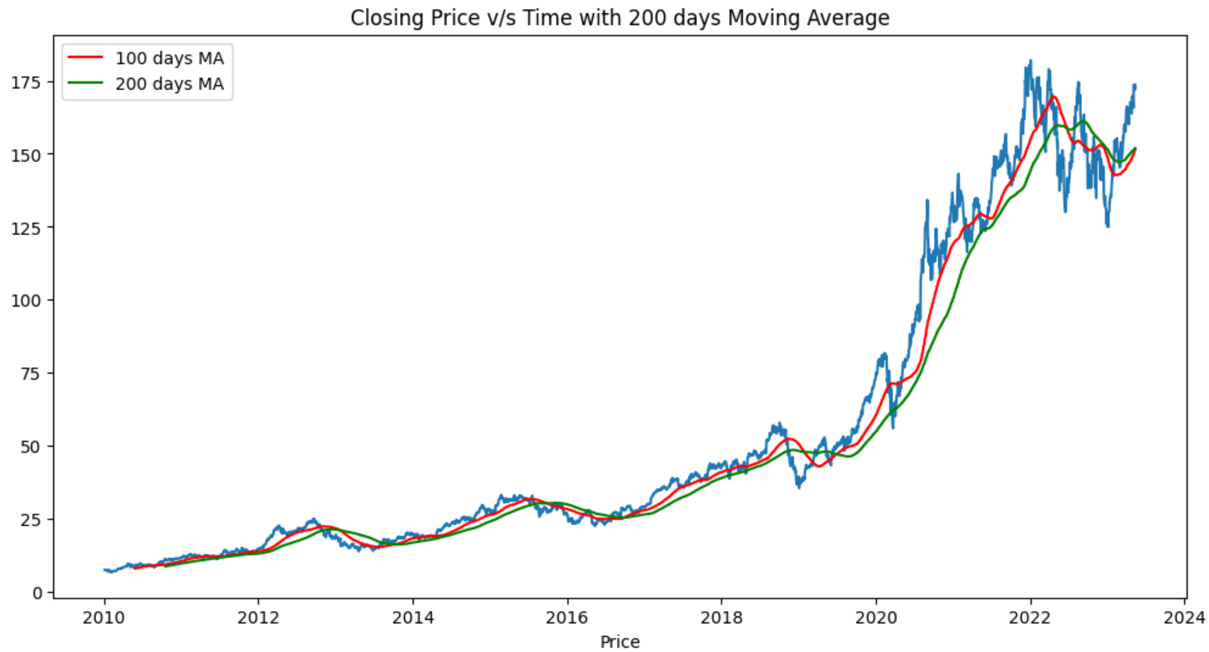
When the 100day SMA (red line) is above the 200days SMA(green line) then a up-trend can be seen and if the reverse occurs then a down-trend occurs

Many experts also use Exponential moving average for the analysis of the trend of Stocks.

In [242…
```
plt.figure(figsize = (12,6))
plt.plot(df.Date, df.Close)
plt.plot(df.Date, ma100, color="r", label="100 days MA")
plt.plot(df.Date, ma200, color="g", label="200 days MA")
```

```
plt.title("Closing Price v/s Time with 200 days Moving Average")
plt.xlabel("Time")
plt.xlabel("Price")
plt.legend()
```

Out[242]: <matplotlib.legend.Legend at 0x23d1c34d0c0>



Splitting data into training and testing

In [243…

```
#splitting data into training and testing

data_training = pd.DataFrame(df["Close"][0:int(len(df)*0.70)])
data_testing = pd.DataFrame(df["Close"][int(len(df)*0.70):int(len(df))])

print(data_training.shape)
print(data_testing.shape)
```

```
(2354, 1)
(1009, 1)
```

In [244…

```
data_training
```

Out[244]:

	Close
0	7.643214
1	7.656429
2	7.534643
3	7.520714
4	7.570714
...	...
2349	52.119999
2350	50.715000
2351	50.724998
2352	50.180000
2353	49.294998

2354 rows × 1 columns

In [245…

```
data_testing.head()
```

Out[245]:

	Close
2354	46.430000
2355	47.165001
2356	47.730000
2357	47.520000
2358	47.250000

In [246…

```
#Scalling the data between 0 and 1

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0,1))
```

In [247…

```
data_training_array = scaler.fit_transform(data_training)
data_training_array
```

Out[247]:

```
array([[0.01533047],
       [0.01558878],
       [0.01320823],
       ...,
       [0.85745296],
       [0.84679984],
       [0.82950064]])
```

In [248…
```
data_training_array.shape
```

Out[248]:
```
(2354, 1)
```

Traing Data

In [249…
```
x_train = []
y_train = []

for i in range(100,data_training_array.shape[0]):
    x_train.append(data_training_array[i-100:i])
    y_train.append(data_training_array[i,0])

x_train, y_train =np.array(x_train), np.array(y_train)
```

In [250…
```
x_train
```

Out[250]:
```
array([[[0.01533047],
        [0.01558878],
        [0.01320823],
        ...,
        [0.03819355],
        [0.03711847],
        [0.03634356]],

       [[0.01558878],
        [0.01320823],
        [0.01293595],
        ...,
        [0.03711847],
        [0.03634356],
        [0.04279409]],

       [[0.01320823],
        [0.01293595],
        [0.01391331],
        ...,
        [0.03634356],
        [0.04279409],
        [0.04525843]],

       ...,

       [[0.69228031],
        [0.70132078],
        [0.67459016],
        ...,
        [0.90070087],
        [0.88472112],
        [0.85725752]],

       [[0.70132078],
        [0.67459016],
        [0.66706457],
        ...,
        [0.88472112],
        [0.85725752],
        [0.85745296]],

       [[0.67459016],
        [0.66706457],
        [0.67747341],
        ...,
        [0.85725752],
        [0.85745296],
        [0.84679984]]])
```

In [251…
```
y_train
```

Out[251]:
```
array([0.04279409, 0.04525843, 0.04801596, ..., 0.85745296, 0.84679984,
       0.82950064])
```

In [252…
```
x_train.shape
```
Out[252]:
```
(2254, 100, 1)
```
In [253…
```
y_train.shape
```
Out[253]:
```
(2254,)
```

Making the ML Model with Sequential and LSTM from tensorflow.keras

In [254…
```
# ML Model

from tensorflow.keras import Sequential
from keras.layers import Dense, Dropout, LSTM
```
In [255…
```
model = Sequential()
model.add(LSTM(units=50, activation = 'relu', return_sequences=True,
               input_shape = (x_train.shape[1],1)))
model.add(Dropout(0.2))



model.add(LSTM(units=60, activation = 'relu', return_sequences=True))
model.add(Dropout(0.3))



model.add(LSTM(units=80, activation = 'relu', return_sequences=True))
model.add(Dropout(0.4))



model.add(LSTM(units=120, activation = 'relu'))
model.add(Dropout(0.5))


model.add(Dense(units=1))
```
In [256…
```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 lstm_12 (LSTM)              (None, 100, 50)           10400     
                                                                 
 dropout_12 (Dropout)        (None, 100, 50)           0         
                                                                 
 lstm_13 (LSTM)              (None, 100, 60)           26640     
                                                                 
 dropout_13 (Dropout)        (None, 100, 60)           0         
                                                                 
 lstm_14 (LSTM)              (None, 100, 80)           45120     
                                                                 
 dropout_14 (Dropout)        (None, 100, 80)           0         
                                                                 
 lstm_15 (LSTM)              (None, 120)               96480     
                                                                 
 dropout_15 (Dropout)        (None, 120)               0         
                                                                 
 dense_3 (Dense)             (None, 1)                 121       
                                                                 
=================================================================
Total params: 178,761
Trainable params: 178,761
Non-trainable params: 0
_________________________________________________________________
```

Training the model with our train data that we have fetched from yfinance. We will use 'adam' Optimizer and 'mean_squared_error' or MSE for determining loss factor of the model. Taking Epochs = 15

In [257…

```
model.compile(optimizer='adam', loss = 'mean_squared_error')
model.fit(x_train,y_train,epochs = 15)
```

```
Epoch 1/15
71/71 [==============================] - 35s 346ms/step - loss: 0.0294
Epoch 2/15
71/71 [==============================] - 27s 380ms/step - loss: 0.0070
Epoch 3/15
71/71 [==============================] - 26s 368ms/step - loss: 0.0059
Epoch 4/15
71/71 [==============================] - 26s 372ms/step - loss: 0.0051
Epoch 5/15
71/71 [==============================] - 26s 372ms/step - loss: 0.0052
Epoch 6/15
71/71 [==============================] - 26s 371ms/step - loss: 0.0052
Epoch 7/15
71/71 [==============================] - 27s 383ms/step - loss: 0.0045
Epoch 8/15
71/71 [==============================] - 26s 370ms/step - loss: 0.0042
Epoch 9/15
71/71 [==============================] - 26s 372ms/step - loss: 0.0038
Epoch 10/15
71/71 [==============================] - 26s 370ms/step - loss: 0.0039
Epoch 11/15
71/71 [==============================] - 26s 367ms/step - loss: 0.0038
Epoch 12/15
71/71 [==============================] - 27s 382ms/step - loss: 0.0031
Epoch 13/15
71/71 [==============================] - 26s 372ms/step - loss: 0.0032
Epoch 14/15
71/71 [==============================] - 26s 372ms/step - loss: 0.0031
Epoch 15/15
71/71 [==============================] - 25s 346ms/step - loss: 0.0030
```

```
Out[257]: <keras.callbacks.History at 0x23d1e4fe500>
```

Saving the model

```
In [278…  model.save("stock_trend_prediction_model_main")
```

```
WARNING:absl:Found untraced functions such as _update_step_xla while saving (showing 1 of 1). These functions will not be directly callable after loading.
INFO:tensorflow:Assets written to: stock_trend_prediction_model_main\assets
INFO:tensorflow:Assets written to: stock_trend_prediction_model_main\assets
```

Testing the ML Model

```
In [279…  #testing the ML Model

data_testing.head()
```

Out[279]:

	Close
2354	46.430000
2355	47.165001
2356	47.730000
2357	47.520000
2358	47.250000

In [280…

```
past_100_days = data_training.tail(100)
past_100_days
```

Out[280]:

	Close
2254	40.985001
2255	41.517502
2256	40.222500
2257	39.207500
2258	37.682499
...	...
2349	52.119999
2350	50.715000
2351	50.724998
2352	50.180000
2353	49.294998

100 rows × 1 columns

In [281…

```
final_df = past_100_days.append(data_testing, ignore_index= True)
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_6844\3501726630.py:1: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  final_df = past_100_days.append(data_testing, ignore_index= True)
```

In [282…

```
final_df
```

Out[282]:

	Close
0	40.985001
1	41.517502
2	40.222500
3	39.207500
4	37.682499
...	...
1104	173.500000
1105	171.770004
1106	173.559998
1107	173.750000
1108	172.570007

1109 rows × 1 columns

In [283…

```
input_data = scaler.fit_transform(final_df)
input_data
```

Out[283]:

```
array([[0.03712555],
       [0.0407613 ],
       [0.03191943],
       ...,
       [0.94230607],
       [0.94360335],
       [0.93554673]])
```

In [284…

```
input_data.shape
```

Out[284]:

```
(1109, 1)
```

In [285…

```
x_test = []
y_test = []

for i in range (100,input_data.shape[0]):
    x_test.append(input_data[i-100:i])
    y_test.append(input_data[i,0])
```

In [286…

```
x_test, y_test = np.array(x_test), np.array(y_test)
print(x_test.shape)
print(y_test.shape)
```

```
(1009, 100, 1)
(1009,)
```

In [287…

```
x_test
```

Out[287]:
```
array([[[0.03712555],
        [0.0407613 ],
        [0.03191943],
        ...,
        [0.1036272 ],
        [0.09990612],
        [0.0938636 ]],

       [[0.0407613 ],
        [0.03191943],
        [0.02498933],
        ...,
        [0.09990612],
        [0.0938636 ],
        [0.0743023 ]],

       [[0.03191943],
        [0.02498933],
        [0.0145771 ],
        ...,
        [0.0938636 ],
        [0.0743023 ],
        [0.07932065]],

       ...,

       [[0.73508585],
        [0.68927202],
        [0.67568489],
        ...,
        [0.94237441],
        [0.94189642],
        [0.93008456]],

       [[0.68927202],
        [0.67568489],
        [0.66107364],
        ...,
        [0.94189642],
        [0.93008456],
        [0.94230607]],

       [[0.67568489],
        [0.66107364],
        [0.66059576],
        ...,
        [0.93008456],
        [0.94230607],
        [0.94360335]]])
```

Our ML MOdel Making Predictions

In [288…
```
#Making Predictions

y_predicted = model.predict(x_test)
```

```
32/32 [==============================] - 2s 70ms/step
```

In [289…

```
y_predicted
```

Out[289]:

```
array([[0.11604273],
       [0.1163878 ],
       [0.11662881],
       ...,
       [0.86071455],
       [0.86329484],
       [0.8662408 ]], dtype=float32)
```

In [290…

```
y_predicted.shape
```

Out[290]:

```
(1009, 1)
```

In [291…

```
y_test
```

Out[291]:

```
array([0.0743023 , 0.07932065, 0.08317828, ..., 0.94230607, 0.94360335,
       0.93554673])
```

In [292…

```
y_test = pd.DataFrame(y_test)
y_test
```

Out[292]:

	0
0	0.074302
1	0.079321
2	0.083178
3	0.081744
4	0.079901
...	...
1004	0.941896
1005	0.930085
1006	0.942306
1007	0.943603
1008	0.935547

1009 rows × 1 columns

In [293…

```
y_predicted = scaler.inverse_transform(y_predicted)
y_test = scaler.inverse_transform(y_test)
```

In [294…

```
y_predicted
```

Out[294]:
```
array([[ 52.543407],
       [ 52.593945],
       [ 52.629246],
       ...,
       [161.6099  ],
       [161.98781 ],
       [162.41928 ]], dtype=float32)
```

In [295…
```
y_test
```

Out[295]:
```
array([[ 46.43000031],
       [ 47.16500092],
       [ 47.72999954],
       ...,
       [173.55999756],
       [173.75      ],
       [172.57000732]])
```

Compairing the Original Prices of the Stock and the Predictions made by our ML Model

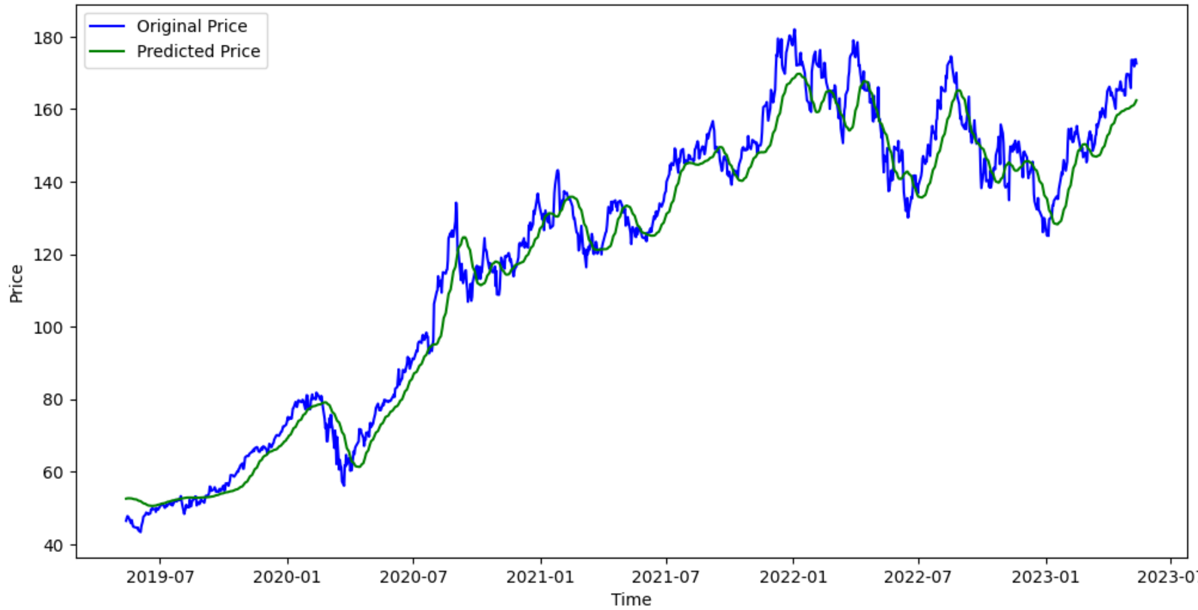
Original Price : Blue

Predictions : Green

In [296…
```
# x-axis date fpr the graph plotted below
date = df.loc[ int(len(df)*0.70):int(len(df)),"Date"]
print(date)
```

```
2354   2019-05-13
2355   2019-05-14
2356   2019-05-15
2357   2019-05-16
2358   2019-05-17
          ...
3358   2023-05-08
3359   2023-05-09
3360   2023-05-10
3361   2023-05-11
3362   2023-05-12
Name: Date, Length: 1009, dtype: datetime64[ns]
```

In [297…
```
plt.figure(figsize=(12,6))
plt.plot(date,y_test,'blue',label='Original Price')
plt.plot(date,y_predicted,'green',label='Predicted Price')
plt.xlabel("Time")
plt.ylabel("Price")
plt.legend()
plt.show()
```



THANK YOU !!

Project by Mr. Arghadip Biswas