We are gonna use AAPL stock for this prediction

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

```
In [2]: df=pd.read_csv('AAPL.csv')
        df.head()
```

Out[2]:

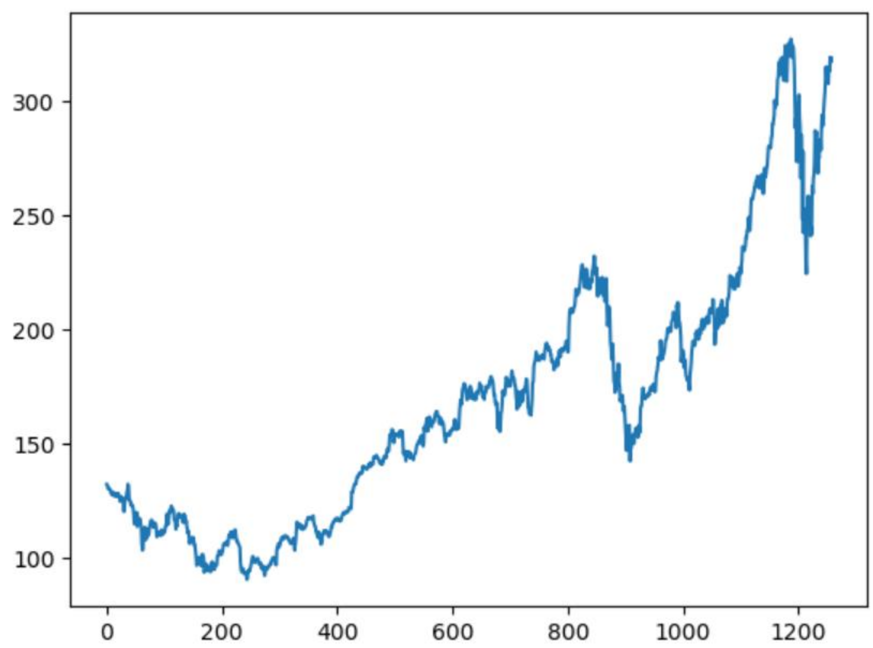
	Unnamed: 0	symbol	date	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	adjVolume	divCash	splitFa
0	0	AAPL	2015-05-27 00:00:00+00:00	132.045	132.260	130.05	130.34	45833246	121.682558	121.880685	119.844118	120.111360	45833246	0.0	
1	1	AAPL	2015-05-28 00:00:00+00:00	131.780	131.950	131.10	131.86	30733309	121.438354	121.595013	120.811718	121.512076	30733309	0.0	
2	2	AAPL	2015-05-29 00:00:00+00:00	130.280	131.450	129.90	131.23	50884452	120.056069	121.134251	119.705890	120.931516	50884452	0.0	
3	3	AAPL	2015-06-01 00:00:00+00:00	130.535	131.390	130.05	131.20	32112797	120.291057	121.078960	119.844118	120.903870	32112797	0.0	
4	4	AAPL	2015-06-02 00:00:00+00:00	129.960	130.655	129.32	129.86	33667627	119.761181	120.401640	119.171406	119.669029	33667627	0.0	

```
In [3]: df.tail()
```

Importing basic libraries and checking the dataset

```
In [4]: plt.plot(df.close)
```

```
Out[4]: [<matplotlib.lines.Line2D at 0x1adbc23acd0>]
```



Plotting closing price as only this would be needed

Now in general conditions we use 100 days avg and 200 days avg to calculate if stock goes up or down

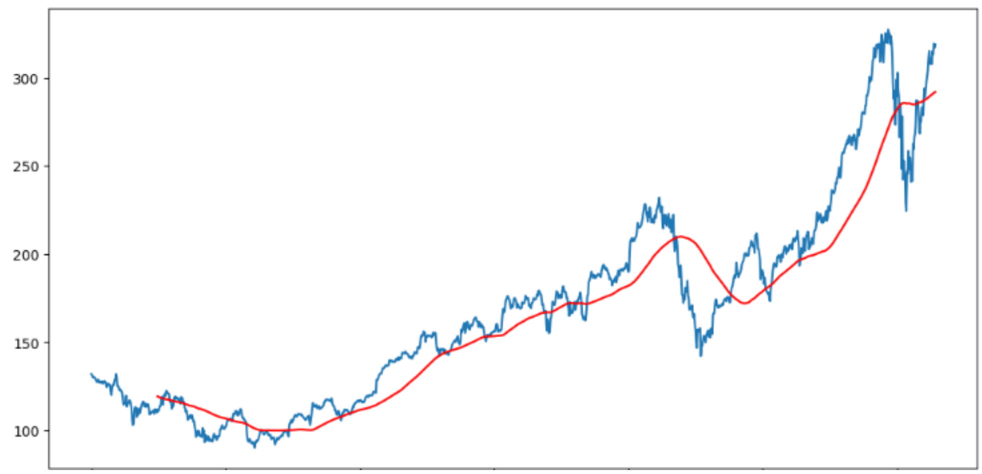
If 100 days>200 days avg then stock up else down

```
In [5]: ma100=df.close.rolling(100).mean()
        ma100
```

```
Out[5]: 0            NaN
        1            NaN
        2            NaN
        3            NaN
        4            NaN
                   ...
        1253    290.6798
        1254    290.9685
        1255    291.2617
        1256    291.5322
        1257    291.8059
        Name: close, Length: 1258, dtype: float64
```

```
In [6]: plt.figure(figsize=(12,6))
        plt.plot(df.close)
        plt.plot(ma100,'r')
```

```
Out[6]: [<matplotlib.lines.Line2D at 0x1adbc87aa50>]
```



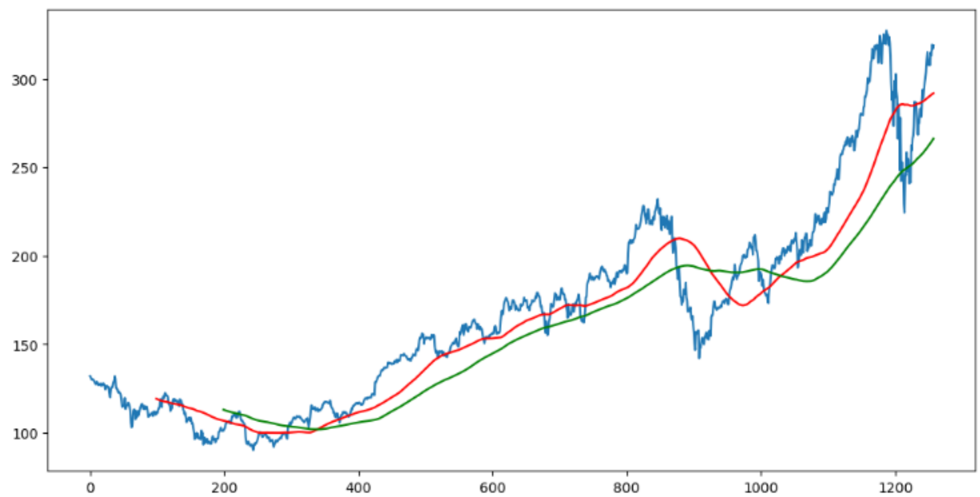
Calculated 100 days avg and plotted a graph

In [7]:
```
ma200=df.close.rolling(200).mean()
ma200
```

Out[7]:
```
0              NaN
1              NaN
2              NaN
3              NaN
4              NaN
           ...
1253    263.742025
1254    264.287625
1255    264.917075
1256    265.516325
1257    266.115575
Name: close, Length: 1258, dtype: float64
```

In [8]:
```
plt.figure(figsize=(12,6))
plt.plot(df.close)
plt.plot(ma100,'r')
plt.plot(ma200,'g')
```

Out[8]: [<matplotlib.lines.Line2D at 0x1adbc930bd0>]



Calculated 200 days avg as well

In [10]:
```
#Splitting Data into Training and Testing
```

In [11]:
```
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))])
```

In [12]:
```
print(data_training.shape)
```

```
(880, 1)
```

In [13]:
```
print(data_testing.shape)
```

```
(378, 1)
```

In [14]:
```
data_training.head()
```

Out[14]:

	close
0	132.045
1	131.780
2	130.280
3	130.535
4	129.960

In [15]:
```
data_testing.head()
```

Out[15]:

	close
880	176.98
881	176.78
882	172.29
883	174.62
884	174.24

Splitting the data(70 perc training and 30 perc testing)

In [16]:
```
#Scaling Down The Data
```

In [17]:
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
```

In [18]:
```
data_training_array=scaler.fit_transform(data_training)
```

In [19]:
```
data_training_array
```

Out[19]:
```
array([[0.29425669],
       [0.29238693],
       [0.28180343],
       [0.28360262],
       [0.27954561],
       [0.28067452],
       [0.27531221],
       [0.27030269],
       [0.26430537],
       [0.26162422],
       [0.27192549],
       [0.26987935],
       [0.2598603 ],
       [0.25809638],
       [0.26289424],
       [0.26077753],
       [0.26486982],
       [0.25583857],
       [0.26296479],
```

In [20]:
```
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])
```

Scaled the data in between 0 to 1

Also made x train and y train which will be needed for prediction. we need the before 100 days for avg that is why there is a difference of 100 days

```
In [23]: x_train,y_train=np.array(x_train),np.array(y_train)
```

```
In [24]: #Making the ML Model
```

```
In [25]: from keras.layers import Dense,Dropout,LSTM
         from keras.models import Sequential
```

```
In [28]: x_train.shape
Out[28]: (780, 100, 1)
```

```
In [34]: model=Sequential()
         model.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
         model.add(LSTM(50,return_sequences=True))
         model.add(LSTM(50))
         model.add(Dense(1))
         model.compile(loss='mean_squared_error',optimizer='adam')
         model.summary()

Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_9 (LSTM)               (None, 100, 50)           10400

 lstm_10 (LSTM)              (None, 100, 50)           20200

 lstm_11 (LSTM)              (None, 50)                20200

 dense_2 (Dense)             (None, 1)                 51

=================================================================
Total params: 50851 (198.64 KB)
Trainable params: 50851 (198.64 KB)
Non-trainable params: 0 (0.00 Byte)
```

Made the ML Model with these specifications

```
_________________________________________________________________

In [35]: model.fit(x_train,y_train,epochs=50)

Epoch 1/50
25/25 [==============================] - 14s 174ms/step - loss: 0.0377
```

Ran 50 epochs

```
In [37]: data_testing.head()
Out[37]:
```

	close
880	176.98
881	176.78
882	172.29
883	174.62
884	174.24

```
In [38]: past_100_days=data_training.tail(100)
         final_df=past_100_days.append(data_testing,ignore_index=True)

C:\Users\mites\AppData\Local\Temp\ipykernel_15024\674358354.py:2: FutureWarning: The frame.append method is deprecated and will
be removed from pandas in a future version. Use pandas.concat instead.
  final_df=past_100_days.append(data_testing,ignore_index=True)
```

```
In [39]: final_df.head()
Out[39]:
```

	close
0	185.11
1	187.18
2	183.92
3	185.40
4	187.97

```
In [40]: inp_data=scaler.fit_transform(final_df)
         inp_data
```

Scaled the testing data

```
Out[41]: (478, 1)
```

```
In [43]: x_test=[]
         y_test=[]
         for i in range(100,inp_data.shape[0]):
             x_test.append(inp_data[i-100:i])
             y_test.append(inp_data[i,0])
```

```
In [44]: x_test,y_test=np.array(x_test),np.array(y_test)
         print(x_test.shape)
         print(y_test.shape)

         (378, 100, 1)
         (378,)
```

```
In [45]: #Making Predictions
```

```
In [46]: y_predicted=model.predict(x_test)

         12/12 [==============================] - 3s 65ms/step
```

```
In [47]: y_predicted.shape

Out[47]: (378, 1)
```

```
In [48]: y_test

Out[48]: array([0.18804389, 0.18696287, 0.16269391, 0.17528782, 0.17323388,
                0.20944814, 0.20193503, 0.19669207, 0.23041998, 0.18647641,
                0.17582833, 0.14215448, 0.14815415, 0.14291119, 0.1454516 ,
```

Same procedure and ran the model

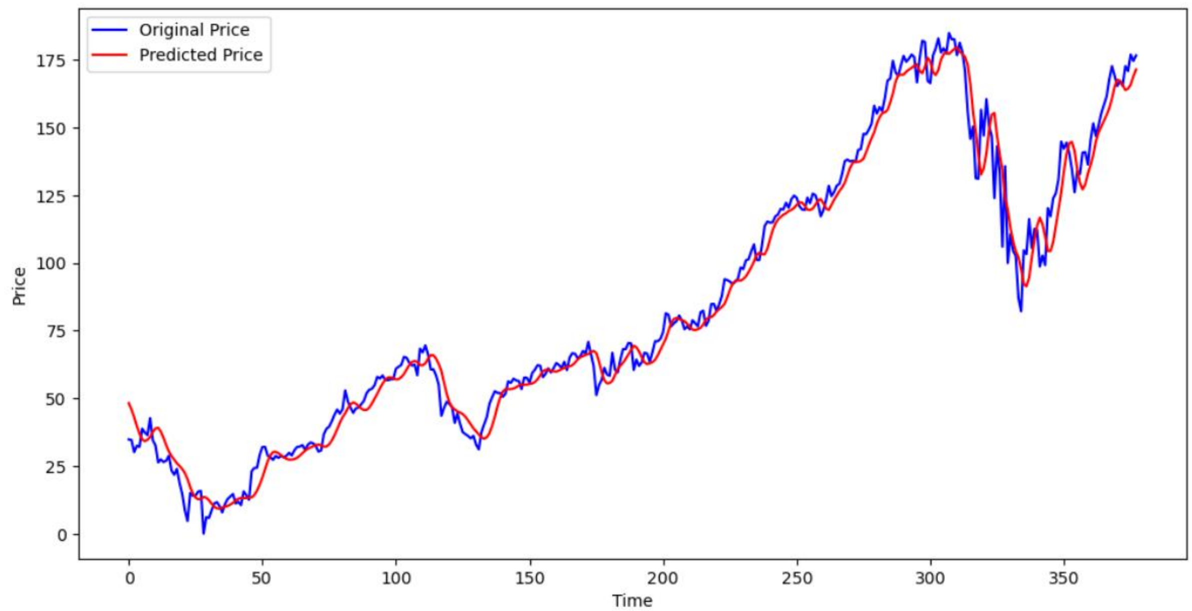
```
In [50]: scaler.scale_

Out[50]: array([0.00540511])
```

```
In [51]: scale_factor=1/0.00540511
         y_predicted=y_predicted*scale_factor
         y_test=y_test*scale_factor
```

```
In [52]: plt.figure(figsize=(12,6))
         plt.plot(y_test,'b',label='Original Price')
         plt.plot(y_predicted,'r',label='Predicted Price')
         plt.xlabel('Time')
         plt.ylabel('Price')
         plt.legend()
         plt.show()
```

Now we will compare predicted values and actual values of testing data

```
plt.show()
```



Final Prediction