```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pandas_datareader as data
```

Double-click (or enter) to edit

```
import yfinance as yf

start = '2010-01-01'
end = '2024-12-09'

# Download stock data for Apple (AAPL)
df = yf.download('AAPL', start=start, end=end)

# Display the first few rows of data
df
```

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

Price	Adj Close	Close	High	Low	Open	Volume
Ticker	AAPL	AAPL	AAPL	AAPL	AAPL	AAPL
Date						
2010-01-04	6.447411	7.643214	7.660714	7.585000	7.622500	493729600
2010-01-05	6.458560	7.656429	7.699643	7.616071	7.664286	601904800
2010-01-06	6.355827	7.534643	7.686786	7.526786	7.656429	552160000
2010-01-07	6.344078	7.520714	7.571429	7.466071	7.562500	477131200
2010-01-08	6.386255	7.570714	7.571429	7.466429	7.510714	447610800
...	...	...	...	...	...	...
2024-12-02	239.589996	239.589996	240.789993	237.160004	237.270004	48137100
2024-12-03	242.649994	242.649994	242.759995	238.899994	239.809998	38861000
2024-12-04	243.009995	243.009995	244.110001	241.250000	242.869995	44383900
2024-12-05	243.039993	243.039993	244.539993	242.130005	243.990005	40033900
2024-12-06	242.839996	242.839996	244.630005	242.080002	242.910004	36852100

3758 rows × 6 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.tail()
```

Price	Adj Close	Close	High	Low	Open	Volume
Ticker	AAPL	AAPL	AAPL	AAPL	AAPL	AAPL
Date						
2024-12-02	239.589996	239.589996	240.789993	237.160004	237.270004	48137100
2024-12-03	242.649994	242.649994	242.759995	238.899994	239.809998	38861000
2024-12-04	243.009995	243.009995	244.110001	241.250000	242.869995	44383900
2024-12-05	243.039993	243.039993	244.539993	242.130005	243.990005	40033900
2024-12-06	242.839996	242.839996	244.630005	242.080002	242.910004	36852100

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

Price	Date	Adj Close	Close	High	Low	Open	Volume
Ticker		AAPL	AAPL	AAPL	AAPL	AAPL	AAPL
0	2010-01-04	6.447413	7.643214	7.660714	7.585000	7.622500	493729600
1	2010-01-05	6.458558	7.656429	7.699643	7.616071	7.664286	601904800
2	2010-01-06	6.355827	7.534643	7.686786	7.526786	7.656429	552160000
3	2010-01-07	6.344078	7.520714	7.571429	7.466071	7.562500	477131200
4	2010-01-08	6.386254	7.570714	7.571429	7.466429	7.510714	447610800

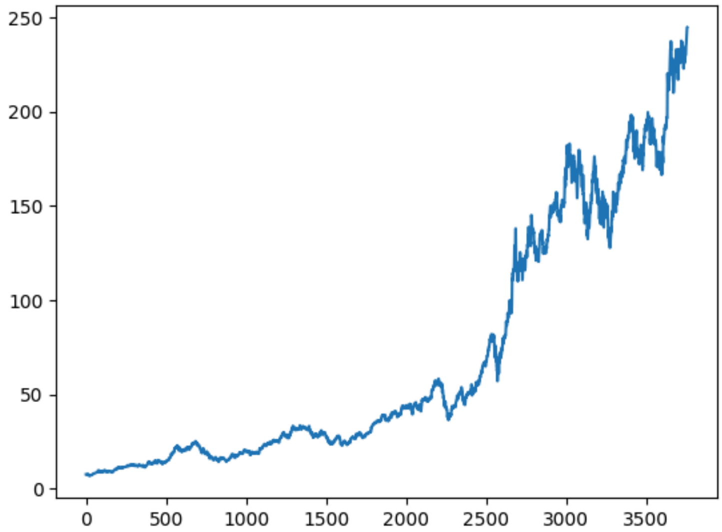
```
df=df.drop(['Date','Adj Close'],axis=1)
df.head()
```

```
<ipython-input-8-42a5f720bdfa>:1: PerformanceWarning: dropping on a non-lexsorted multi-index without a level parameter may impact p
  df=df.drop(['Date','Adj Close'],axis=1)
```

Price	Close	High	Low	Open	Volume
Ticker	AAPL	AAPL	AAPL	AAPL	AAPL
0	7.643214	7.660714	7.585000	7.622500	493729600
1	7.656429	7.699643	7.616071	7.664286	601904800
2	7.534643	7.686786	7.526786	7.656429	552160000
3	7.520714	7.571429	7.466071	7.562500	477131200
4	7.570714	7.571429	7.466429	7.510714	447610800

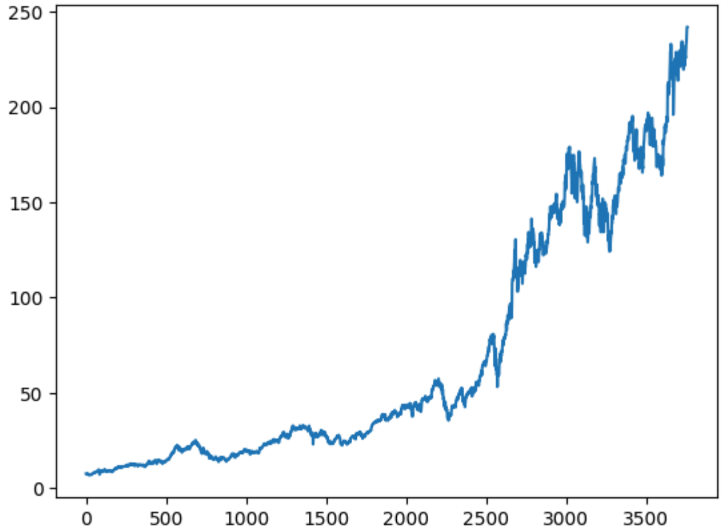
```
plt.plot(df.High)
```

```
[<matplotlib.lines.Line2D at 0x78588a4693c0>]
```




```
plt.plot(df.Low)
```

```
[<matplotlib.lines.Line2D at 0x78588a31bc40>]
```




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

Ticker	AAPL
0	NaN
1	NaN
2	NaN
3	NaN
4	NaN
...	...
3753	225.7750
3754	225.8961
3755	225.9822
3756	226.0644
3757	226.2040

3758 rows × 1 columns

```
ma200 = df.Close.rolling(200).mean()
ma200
```

Ticker	AAPL
0	NaN
1	NaN
2	NaN
3	NaN
4	NaN
...	...
3753	206.10510
3754	206.40680
3755	206.71405
3756	207.01765
3757	207.31000

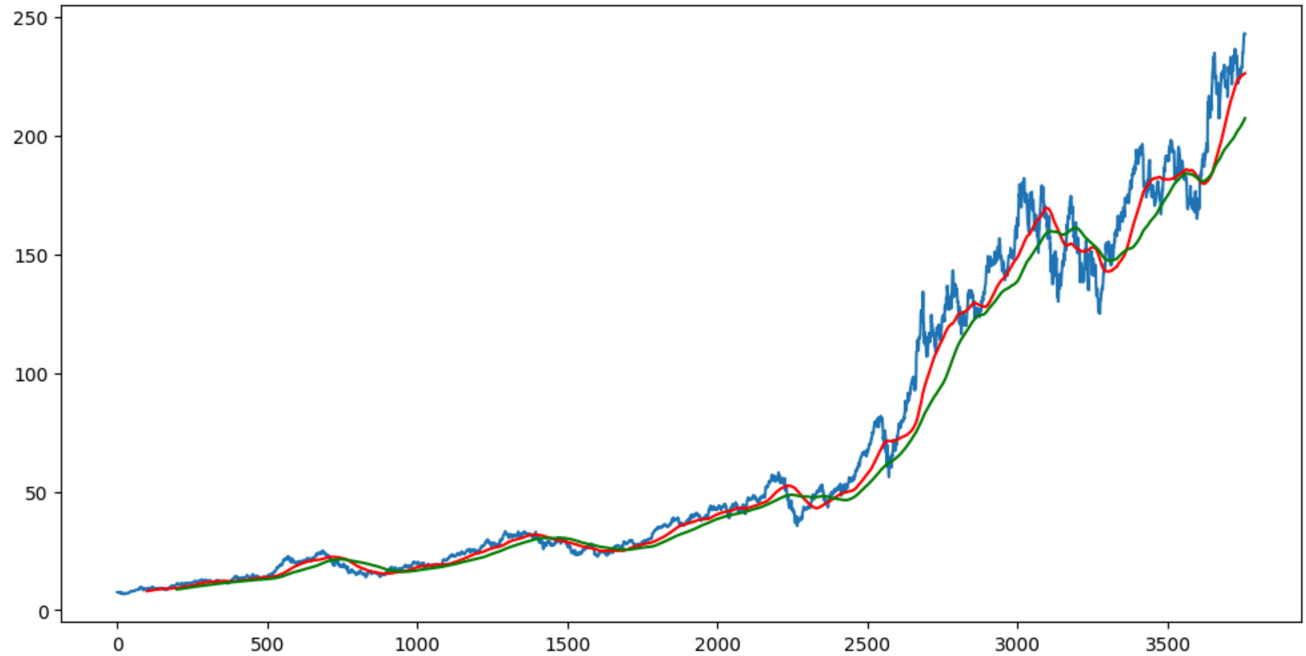
3758 rows × 1 columns

```
plt.figure(figsize=(12,6))
plt.plot(df.High)
plt.plot(df.Low)
```

```
[<matplotlib.lines.Line2D at 0x79ecdc3b1930>]
```


```
plt.figure(figsize=(12,6))
plt.plot(df.Close)
plt.plot(ma100,'r')
plt.plot(ma200,'g')
```

```
[<matplotlib.lines.Line2D at 0x7858f771e4d0>]
```




```
df.shape
```

```
(3758, 6)
```

```
# Split data for high and low price predictions
data_training_high = pd.DataFrame(df['High'][0:int(len(df)*0.70)])
data_testing_high = pd.DataFrame(df['High'][int(len(df)*0.70):])

data_training_low = pd.DataFrame(df['Low'][0:int(len(df)*0.70)])
data_testing_low = pd.DataFrame(df['Low'][int(len(df)*0.70):])


print(data_training_high.shape)
print(data_testing_high.shape)
print(data_training_low.shape)
print(data_testing_low.shape)
```

```
(2630, 1)
(1128, 1)
```

```
(2630, 1)
(1128, 1)
```

```
#scale data for high
from sklearn.preprocessing import MinMaxScaler
scaler_high = MinMaxScaler(feature_range=(0, 1))
data_training_high_array = scaler_high.fit_transform(data_training_high)
```

```
data_training_high_array
```

```
array([[0.00808782],
       [0.00856435],
       [0.00840697],
       ...,
       [0.98864647],
       [0.97867001],
       [0.97218228]])
```

```
#scale data for low
from sklearn.preprocessing import MinMaxScaler
scaler_low = MinMaxScaler(feature_range=(0, 1))
data_training_low_array = scaler_low.fit_transform(data_training_low)
data_training_low_array
```

```
array([[0.00991319],
       [0.0103029 ],
       [0.00918303],
       ...,
       [0.96673062],
       [0.96277965],
       [0.95763711]])
```

```
x_train_high = []
y_train_high = []
for i in range(100, data_training_high_array.shape[0]):
    x_train_high.append(data_training_high_array[i-100:i])
    y_train_high.append(data_training_high_array[i, 0])

x_train_high, y_train_high = np.array(x_train_high), np.array(y_train_high)
x_train_high
y_train_high
```

```
array([0.02530832, 0.02771718, 0.03057633, ..., 0.98864647, 0.97867001,
       0.97218228])
```

```
# Prepare training data for low price prediction
x_train_low, y_train_low = [], []
for i in range(100, data_training_low_array.shape[0]):
    x_train_low.append(data_training_low_array[i-100:i])
    y_train_low.append(data_training_low_array[i, 0])

x_train_low, y_train_low = np.array(x_train_low), np.array(y_train_low)
```

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

# Model for High Prices
model_high = Sequential()
model_high.add(LSTM(units=50, activation='relu', return_sequences=True, input_shape=(x_train_high.shape[1], 1)))
model_high.add(Dropout(0.2))
model_high.add(LSTM(units=60, activation='relu', return_sequences=True))
model_high.add(Dropout(0.3))
model_high.add(LSTM(units=80, activation='relu', return_sequences=True))
model_high.add(Dropout(0.4))
model_high.add(LSTM(units=120, activation='relu'))
model_high.add(Dropout(0.5))
model_high.add(Dense(units=1))

model_high.compile(optimizer='adam', loss='mean_squared_error')
model_high.fit(x_train_high, y_train_high, epochs=50, batch_size=32)

# Model for Low Prices
model_low = Sequential()
model_low.add(LSTM(units=50, activation='relu', return_sequences=True, input_shape=(x_train_low.shape[1], 1)))
model_low.add(Dropout(0.2))
model_low.add(LSTM(units=60, activation='relu', return_sequences=True))
model_low.add(Dropout(0.3))
```

```
model_low.add(LSTM(units=80, activation='relu', return_sequences=True))
model_low.add(Dropout(0.4))
model_low.add(LSTM(units=120, activation='relu'))
model_low.add(Dropout(0.5))
model_low.add(Dense(units=1))

model_low.compile(optimizer='adam', loss='mean_squared_error')
model_low.fit(x_train_low, y_train_low, epochs=50, batch_size=32)
```

```
Epoch 23/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - loss: 0.0017
Epoch 24/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 5s 44ms/step - loss: 0.0017
Epoch 25/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 4s 47ms/step - loss: 0.0018
Epoch 26/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - loss: 0.0016
Epoch 27/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 5s 43ms/step - loss: 0.0016
Epoch 28/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 4s 48ms/step - loss: 0.0016
Epoch 29/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 4s 44ms/step - loss: 0.0015
Epoch 30/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - loss: 0.0015
Epoch 31/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 6s 48ms/step - loss: 0.0014
Epoch 32/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - loss: 0.0015
Epoch 33/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 5s 43ms/step - loss: 0.0014
Epoch 34/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 4s 48ms/step - loss: 0.0016
Epoch 35/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - loss: 0.0015
Epoch 36/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - loss: 0.0014
Epoch 37/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 4s 44ms/step - loss: 0.0028
Epoch 38/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 5s 43ms/step - loss: 0.0016
Epoch 39/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 5s 43ms/step - loss: 0.0017
Epoch 40/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - loss: 0.0014
Epoch 41/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 5s 43ms/step - loss: 0.0014
Epoch 42/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 4s 44ms/step - loss: 0.0013
Epoch 43/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 4s 47ms/step - loss: 0.0017
Epoch 44/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - loss: 0.0015
Epoch 45/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - loss: 0.0019
Epoch 46/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - loss: 0.0013
Epoch 47/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - loss: 0.0012
Epoch 48/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 5s 44ms/step - loss: 0.0014
Epoch 49/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - loss: 0.0017
Epoch 50/50
80/80 ━━━━━━━━━━━━━━━━━━━━ 5s 44ms/step - loss: 0.0013
<keras.src.callbacks.history.History at 0x79ec005c5b10>
```

```
model_high.save('model_high.h5')
model_low.save('model_low.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

```
from google.colab import files

# Download the saved models
files.download('model_high.h5')
files.download('model_low.h5')
```

```
# Reset the index and drop the current index (Date will be removed from the index)
data_testing_high.reset_index(drop=True, inplace=True)
# Drop the 'Date' column
print(data_testing_high.head())
data_testing_low.reset_index(drop=True, inplace=True)
print(data_testing_low.head())
```

```
Ticker       AAPL
0       88.300003
1       88.849998
2       88.362503
3       89.139999
4       89.864998
Ticker       AAPL
0       86.180000
1       87.772499
2       87.305000
3       86.287498
4       87.787498
```

```
 #to predict the next value we need the value of the previous 100 days
 past_100_days_high = data_training_high.tail(100)
 past_100_days_low = data_training_low.tail(100)
```

```
final_df_high = pd.concat([past_100_days_high, data_testing_high], ignore_index=True)
final_df_low = pd.concat([past_100_days_low, data_testing_low], ignore_index=True)
```

```
input_data_high = scaler_high.fit_transform(final_df_high)
input_data_low = scaler_low.fit_transform(final_df_low)
input_data_high
input_data_low
```

```
array([[0.13631251],
       [0.13878634],
       [0.1220648 ],
       ...,
       [0.99534334],
       [1.        ],
       [0.9997354 ]])
```

```
input_data_high.shape
```

```
(1228, 1)
```

```
x_test_high = []
y_test_high = []

for i in range(100, input_data_high.shape[0]):
    x_test_high.append(input_data_high[i-100:i])
    y_test_high.append(input_data_high[i, 0])


x_test_low = []
y_test_low = []

for i in range(100, input_data_low.shape[0]):
    x_test_low.append(input_data_low[i-100:i])
    y_test_low.append(input_data_low[i, 0])
```

```
x_test_high, y_test_high = np.array(x_test_high), np.array(y_test_high)
```

```
print(x_test_high.shape)
print(y_test_high.shape)
```

```
(1128, 100, 1)
(1128,)
```

```
x_test_low, y_test_low = np.array(x_test_low), np.array(y_test_low)
print(x_test_low.shape)
print(y_test_low.shape)
```

```
(1128, 100, 1)
(1128,)
```

```
y_high_predictions = model_high.predict(x_test_high)
y_low_predictions = model_low.predict(x_test_low)
```

```
36/36 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step
36/36 ━━━━━━━━━━━━━━━━━━━━ 3s 59ms/step
```

```
y_high_predictions.shape
y_low_predictions.shape
```

```
(1128, 1)
```

Start coding or generate with AI.

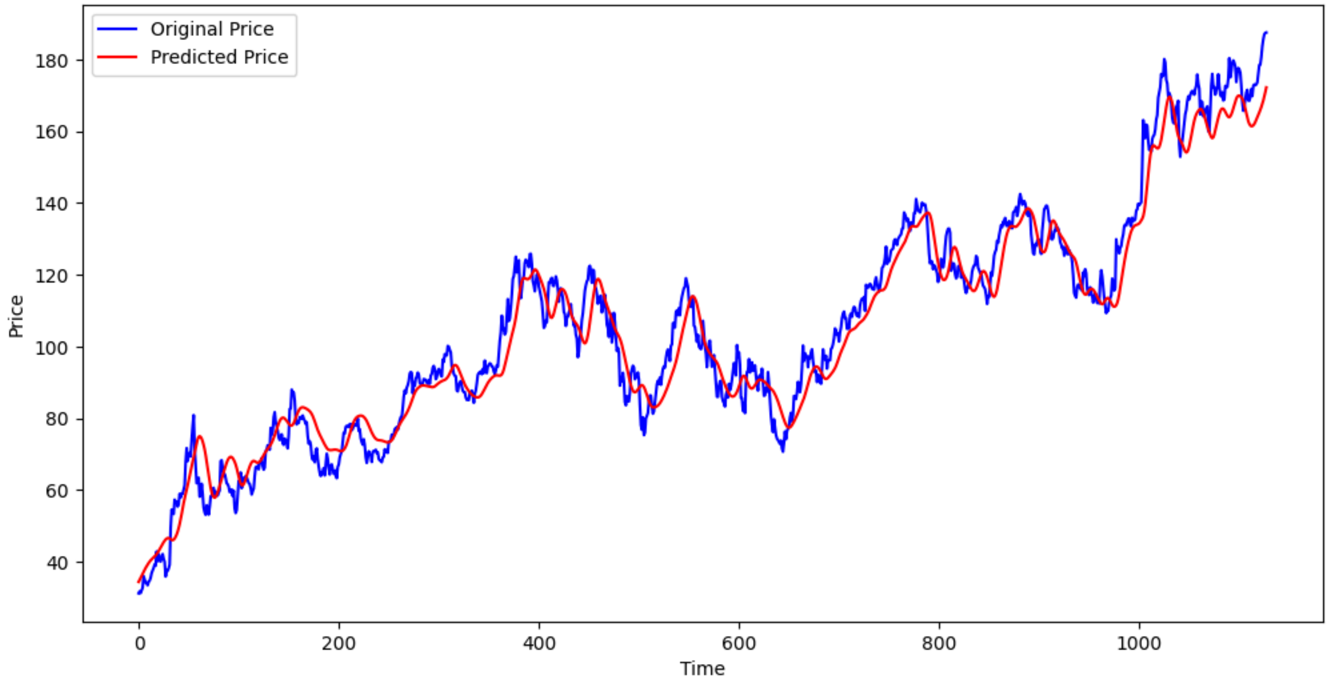
```
scaler2 = scaler_high.scale_
```

```
scaler3= scaler_low.scale_
```

```
scale_factor_high = 1/scaler2[0]
scale_factor_low = 1/scaler3[0]
y_predicted_high = y_high_predictions * scale_factor_high
y_predicted_low = y_low_predictions * scale_factor_low
y_test_high = y_test_high * scale_factor_high
y_test_low = y_test_low * scale_factor_low
```

```
plt.figure(figsize=(12,6))
plt.plot(y_test_high,'b',label='Original Price')
plt.plot(y_predicted_high,'r',label='Predicted Price')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt
```

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>
```




```
plt.figure(figsize=(12,6))
plt.plot(y_test_low,'b',label='Original Price')
plt.plot(y_predicted_low,'r',label='Predicted Price')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt
```

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>
```