Data Science @Bharat-Intern

Done By Harsha Vardhan

```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM, Bidirectional
```

```
df = pd.read_csv('Google_Stock.csv') # importing the data
df.head(10)
```

	symbol	date	close	high	low	open	volume	adjClose	adjHigh
0	GOOG	2016-06-14 00:00:00+00:00	718.27	722.47	713.1200	716.48	1306065	718.27	722.47
1	GOOG	2016-06-15 00:00:00+00:00	718.92	722.98	717.3100	719.00	1214517	718.92	722.98
2	GOOG	2016-06-16 00:00:00+00:00	710.36	716.65	703.2600	714.91	1982471	710.36	716.65
3	GOOG	2016-06-17 00:00:00+00:00	691.72	708.82	688.4515	708.65	3402357	691.72	708.82
4	GOOG	2016-06-20 00:00:00+00:00	693.71	702.48	693.4100	698.77	2082538	693.71	702.48
5	GOOG	2016-06-21 00:00:00+00:00	695.94	702.77	692.0100	698.40	1465634	695.94	702.77

```
print("Shape of data:",df.shape)
```

```
Shape of data: (1258, 14)
```

```
df.describe()
```

	close	high	low	open	volume	adjClose	adjHigh	ad
count	1258.000000	1258.000000	1258.000000	1258.000000	1.258000e+03	1258.000000	1258.000000	1258.00
mean	1216.317067	1227.430934	1204.176430	1215.260779	1.601590e+06	1216.317067	1227.430936	1204.17
std	383.333358	387.570872	378.777094	382.446995	6.960172e+05	383.333358	387.570873	378.77
min	668.260000	672.300000	663.284000	671.000000	3.467530e+05	668.260000	672.300000	663.28
25%	960.802500	968.757500	952.182500	959.005000	1.173522e+06	960.802500	968.757500	952.18
50%	1132.460000	1143.935000	1117.915000	1131.150000	1.412588e+06	1132.460000	1143.935000	1117.91
75%	1360.595000	1374.345000	1348.557500	1361.075000	1.812156e+06	1360.595000	1374.345000	1348.55
max	2521.600000	2526.990000	2498.290000	2524.920000	6.207027e+06	2521.600000	2526.990000	2498.29

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   symbol       1258 non-null   object 
 1   date         1258 non-null   object 
 2   close        1258 non-null   float64
 3   high         1258 non-null   float64
 4   low          1258 non-null   float64
 5   open         1258 non-null   float64
 6   volume       1258 non-null   int64
 7   adjClose     1258 non-null   float64
 8   adjHigh      1258 non-null   float64
 9   adjLow       1258 non-null   float64
 10  adjOpen      1258 non-null   float64
 11  adjVolume    1258 non-null   int64
 12  divCash      1258 non-null   float64
 13  splitFactor  1258 non-null   float64
dtypes: float64(10), int64(2), object(2)
memory usage: 137.7+ KB
```

```
df.isnull().sum()
```

```
symbol         0
date           0
close          0
high           0
low            0
open           0
volume         0
adjClose       0
adjHigh        0
adjLow         0
adjOpen        0
adjVolume      0
divCash        0
splitFactor    0
dtype: int64
```

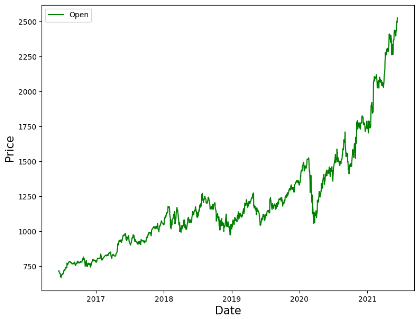
```
df = df[['date','open','close']]
df['date'] = pd.to_datetime(df['date'].apply(lambda x: x.split()[0]))
df.set_index('date',drop=True,inplace=True)
df.head(10)
```

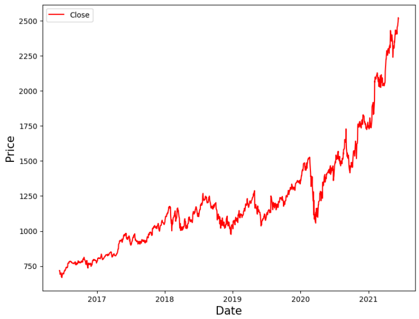
	open	close
date		
2016-06-14	716.48	718.27
2016-06-15	719.00	718.92
2016-06-16	714.91	710.36
2016-06-17	708.65	691.72
2016-06-20	698.77	693.71
2016-06-21	698.40	695.94
2016-06-22	699.06	697.46
2016-06-23	697.45	701.87
2016-06-24	675.17	675.22
2016-06-27	671.00	668.26

```
fig, ax =plt.subplots(1,2,figsize=(20,7))
ax[0].plot(df['open'],label='Open',color='green')
ax[0].set_xlabel('Date',size=15)
ax[0].set_ylabel('Price',size=15)
ax[0].legend()

ax[1].plot(df['close'],label='Close',color='red')
ax[1].set_xlabel('Date',size=15)
ax[1].set_ylabel('Price',size=15)
ax[1].legend()

fig.show()
```





```
MMS = MinMaxScaler()
df[df.columns] = MMS.fit_transform(df)
df.head(10)
```

	open	close
date		
2016-06-14	0.024532	0.026984
2016-06-15	0.025891	0.027334
2016-06-16	0.023685	0.022716
2016-06-17	0.020308	0.012658
2016-06-20	0.014979	0.013732
2016-06-21	0.014779	0.014935
2016-06-22	0.015135	0.015755
2016-06-23	0.014267	0.018135
2016-06-24	0.002249	0.003755
2016-06-27	0.000000	0.000000

```
training_size = round(len(df) * 0.75)
training_size
```

```
944
```

```
train_data = df[:training_size]
test_data  = df[training_size:]

train_data.shape, test_data.shape
```

```
((944, 2), (314, 2))
```

```
def create_sequence(dataset):
  sequences = []
  labels = []

  start_idx = 0

  for stop_idx in range(50,len(dataset)): # Selecting 50 rows at a time
    sequences.append(dataset.iloc[start_idx:stop_idx])
    labels.append(dataset.iloc[stop_idx])
    start_idx += 1
  return (np.array(sequences),np.array(labels))
```

```
train_seq, train_label = create_sequence(train_data)
test_seq, test_label = create_sequence(test_data)
train_seq.shape, train_label.shape, test_seq.shape, test_label.shape
```

```
((894, 50, 2), (894, 2), (264, 50, 2), (264, 2))
```

```
model = Sequential()

model.add(LSTM(units=50, return_sequences=True, input_shape = (train_seq.shape[1], train_seq.shape[2])))

model.add(Dropout(0.1))
model.add(LSTM(units=50))

model.add(Dense(2))
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_absolute_error'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50, 50)            10600

 dropout (Dropout)           (None, 50, 50)            0

 lstm_1 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 2)                 102

=================================================================
Total params: 30,902
Trainable params: 30,902
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(train_seq, train_label, epochs=100,validation_data=(test_seq, test_label), verbose=1)
```

```
Epoch 1/100
28/28 [==============================] - 7s 98ms/step - loss: 0.0077 - mean_absolute_error: 0.0604 - val_loss: 0.0124 - val_mean_
Epoch 2/100
28/28 [==============================] - 2s 86ms/step - loss: 7.7463e-04 - mean_absolute_error: 0.0219 - val_loss: 0.0062 - val_r
Epoch 3/100
28/28 [==============================] - 2s 76ms/step - loss: 4.7846e-04 - mean_absolute_error: 0.0162 - val_loss: 0.0042 - val_r
Epoch 4/100
28/28 [==============================] - 2s 63ms/step - loss: 4.9564e-04 - mean_absolute_error: 0.0162 - val_loss: 0.0041 - val_r
Epoch 5/100
28/28 [==============================] - 2s 62ms/step - loss: 4.3763e-04 - mean_absolute_error: 0.0155 - val_loss: 0.0034 - val_r
Epoch 6/100
28/28 [==============================] - 2s 62ms/step - loss: 4.6218e-04 - mean_absolute_error: 0.0154 - val_loss: 0.0054 - val_r
Epoch 7/100
28/28 [==============================] - 2s 61ms/step - loss: 4.2639e-04 - mean_absolute_error: 0.0156 - val_loss: 0.0038 - val_r
Epoch 8/100
28/28 [==============================] - 2s 62ms/step - loss: 4.0365e-04 - mean_absolute_error: 0.0146 - val_loss: 0.0061 - val_r
Epoch 9/100
28/28 [==============================] - 3s 97ms/step - loss: 4.1644e-04 - mean_absolute_error: 0.0148 - val_loss: 0.0034 - val_r
Epoch 10/100
28/28 [==============================] - 2s 66ms/step - loss: 4.0317e-04 - mean_absolute_error: 0.0148 - val_loss: 0.0030 - val_r
Epoch 11/100
28/28 [==============================] - 2s 62ms/step - loss: 3.8071e-04 - mean_absolute_error: 0.0143 - val_loss: 0.0046 - val_r
Epoch 12/100
28/28 [==============================] - 2s 62ms/step - loss: 3.7372e-04 - mean_absolute_error: 0.0141 - val_loss: 0.0037 - val_r
Epoch 13/100
28/28 [==============================] - 2s 62ms/step - loss: 3.6237e-04 - mean_absolute_error: 0.0140 - val_loss: 0.0047 - val_r
Epoch 14/100
28/28 [==============================] - 2s 62ms/step - loss: 3.4868e-04 - mean_absolute_error: 0.0137 - val_loss: 0.0051 - val_r
Epoch 15/100
28/28 [==============================] - 2s 67ms/step - loss: 3.5568e-04 - mean_absolute_error: 0.0138 - val_loss: 0.0037 - val_r
Epoch 16/100
28/28 [==============================] - 3s 100ms/step - loss: 3.5125e-04 - mean_absolute_error: 0.0137 - val_loss: 0.0052 - val_
Epoch 17/100
28/28 [==============================] - 2s 62ms/step - loss: 3.1965e-04 - mean_absolute_error: 0.0131 - val_loss: 0.0043 - val_r
Epoch 18/100
28/28 [==============================] - 2s 61ms/step - loss: 3.3675e-04 - mean_absolute_error: 0.0133 - val_loss: 0.0060 - val_r
Epoch 19/100
28/28 [==============================] - 2s 61ms/step - loss: 3.0591e-04 - mean_absolute_error: 0.0129 - val_loss: 0.0062 - val_r
Epoch 20/100
28/28 [==============================] - 2s 62ms/step - loss: 2.9171e-04 - mean_absolute_error: 0.0126 - val_loss: 0.0064 - val_r
Epoch 21/100
28/28 [==============================] - 2s 61ms/step - loss: 2.9481e-04 - mean_absolute_error: 0.0126 - val_loss: 0.0072 - val_r
Epoch 22/100
28/28 [==============================] - 2s 76ms/step - loss: 3.0077e-04 - mean_absolute_error: 0.0128 - val_loss: 0.0077 - val_r
Epoch 23/100
28/28 [==============================] - 2s 89ms/step - loss: 2.9011e-04 - mean_absolute_error: 0.0126 - val_loss: 0.0049 - val_r
Epoch 24/100
28/28 [==============================] - 2s 62ms/step - loss: 3.0030e-04 - mean_absolute_error: 0.0128 - val_loss: 0.0067 - val_r
Epoch 25/100
28/28 [==============================] - 2s 61ms/step - loss: 2.7004e-04 - mean_absolute_error: 0.0122 - val_loss: 0.0049 - val_r
Epoch 26/100
28/28 [==============================] - 2s 62ms/step - loss: 2.9431e-04 - mean_absolute_error: 0.0126 - val_loss: 0.0034 - val_r
Epoch 27/100
28/28 [==============================] - 2s 62ms/step - loss: 3.6854e-04 - mean_absolute_error: 0.0145 - val_loss: 0.0077 - val_r
Epoch 28/100
28/28 [==============================] - 2s 62ms/step - loss: 3.3013e-04 - mean_absolute_error: 0.0134 - val_loss: 0.0035 - val_r
Epoch 29/100
```

```
test_predicted = model.predict(test_seq)
test_predicted[:5]
```

```
9/9 [==============================] - 2s 30ms/step
array([[0.41044712, 0.4126559 ],
       [0.4094599 , 0.4115072 ],
       [0.40534687, 0.40727264],
       [0.4103477 , 0.412193  ],
       [0.41390288, 0.41581792]], dtype=float32)
```

```
test_inverse_predicted = MMS.inverse_transform(test_predicted)
test_inverse_predicted[:5]
```

```
array([[1431.9362, 1433.0518],
       [1430.1058, 1430.9227],
       [1422.4807, 1423.0747],
       [1431.7518, 1432.1938],
       [1438.3428, 1438.912 ]], dtype=float32)
```

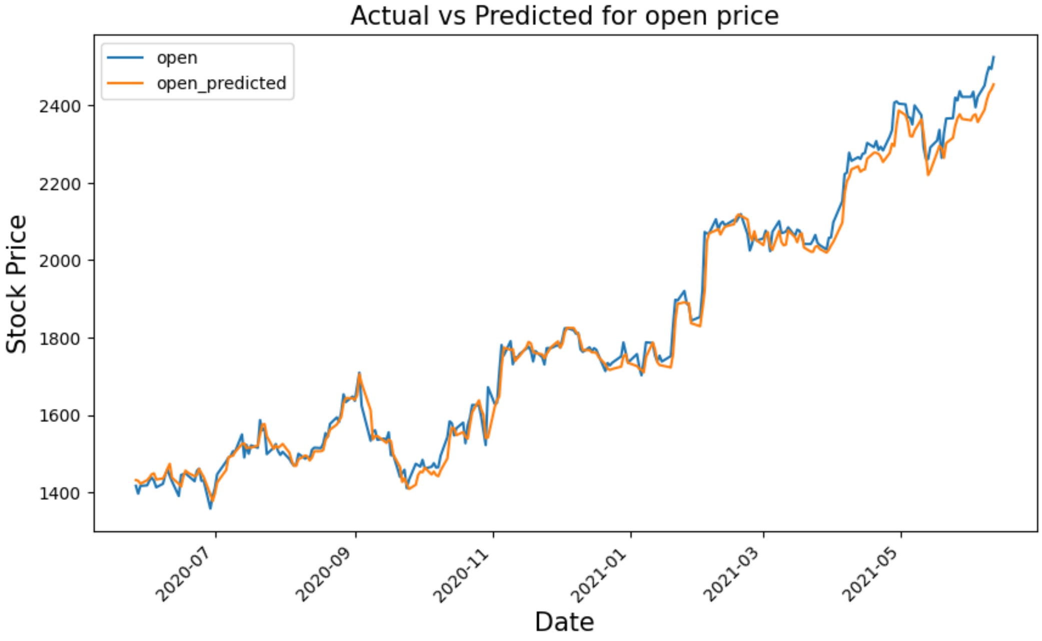
Comparing Predicted Data

```
df_merge = pd.concat([df.iloc[-264:].copy(),
                          pd.DataFrame(test_inverse_predicted,columns=['open_predicted','close_predicted'],
                                       index=df.iloc[-264:].index)], axis=1)
```

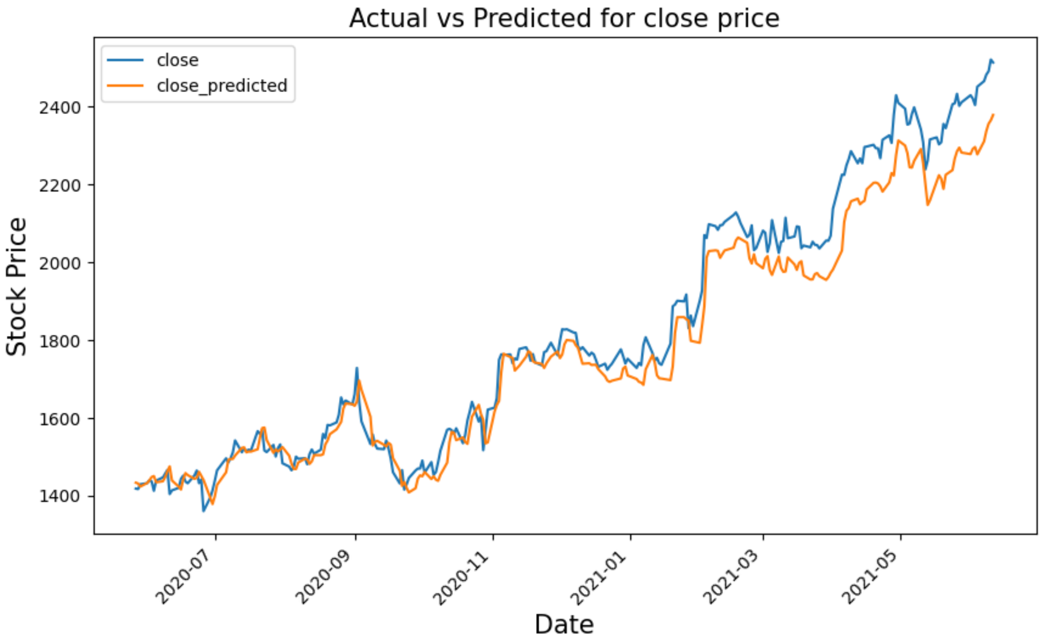
```
df_merge[['open','close']] = MMS.inverse_transform(df_merge[['open','close']])
df_merge.head()
```

	open	close	open_predicted	close_predicted
date				
2020-05-27	1417.25	1417.84	1431.936157	1433.051758
2020-05-28	1396.86	1416.73	1430.105835	1430.922729
2020-05-29	1416.94	1428.92	1422.480713	1423.074707
2020-06-01	1418.39	1431.82	1431.751831	1432.193848
2020-06-02	1430.55	1439.22	1438.342773	1438.911987

```
df_merge[['open','open_predicted']].plot(figsize=(10,6))
plt.xticks(rotation=45)
plt.xlabel('Date',size=15)
plt.ylabel('Stock Price',size=15)
plt.title('Actual vs Predicted for open price',size=15)
plt.show()
```



```
df_merge[['close','close_predicted']].plot(figsize=(10,6))
plt.xticks(rotation=45)
plt.xlabel('Date',size=15)
plt.ylabel('Stock Price',size=15)
plt.title('Actual vs Predicted for close price',size=15)
plt.show()
```



Prediction For Next 10 Days

```
df_merge = df_merge.append(pd.DataFrame(columns=df_merge.columns,
                                        index=pd.date_range(start=df_merge.index[-1], periods=11, freq='D', closed='right')))
df_merge['2021-06-09':'2021-06-16']
```

	open	close	open_predicted	close_predicted
2021-06-09	2499.50	2491.40	2432.127930	2356.553467
2021-06-10	2494.01	2521.60	2440.968262	2365.456299
2021-06-11	2524.92	2513.93	2454.351807	2379.111816
2021-06-12	NaN	NaN	NaN	NaN
2021-06-13	NaN	NaN	NaN	NaN
2021-06-14	NaN	NaN	NaN	NaN
2021-06-15	NaN	NaN	NaN	NaN
2021-06-16	NaN	NaN	NaN	NaN

```
upcoming_prediction = pd.DataFrame(columns=['open','close'],index=df_merge.index)
upcoming_prediction.index=pd.to_datetime(upcoming_prediction.index)
```

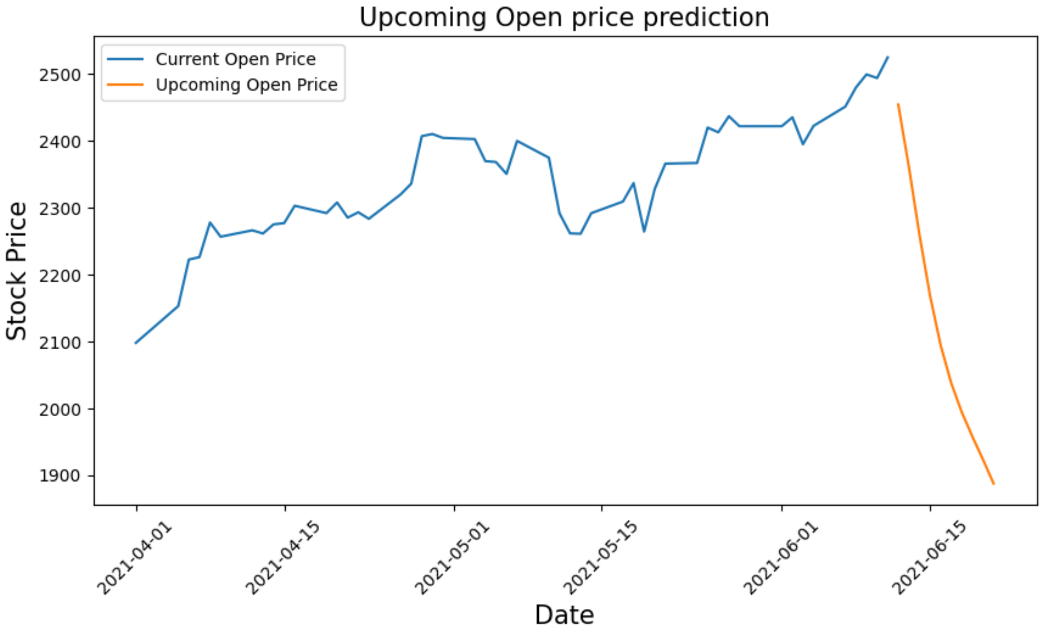
```
curr_seq = test_seq[-1:]

for i in range(-10,0):
  up_pred = model.predict(curr_seq)
  upcoming_prediction.iloc[i] = up_pred
  curr_seq = np.append(curr_seq[0][1:],up_pred,axis=0)
  curr_seq = curr_seq.reshape(test_seq[-1:].shape)
```

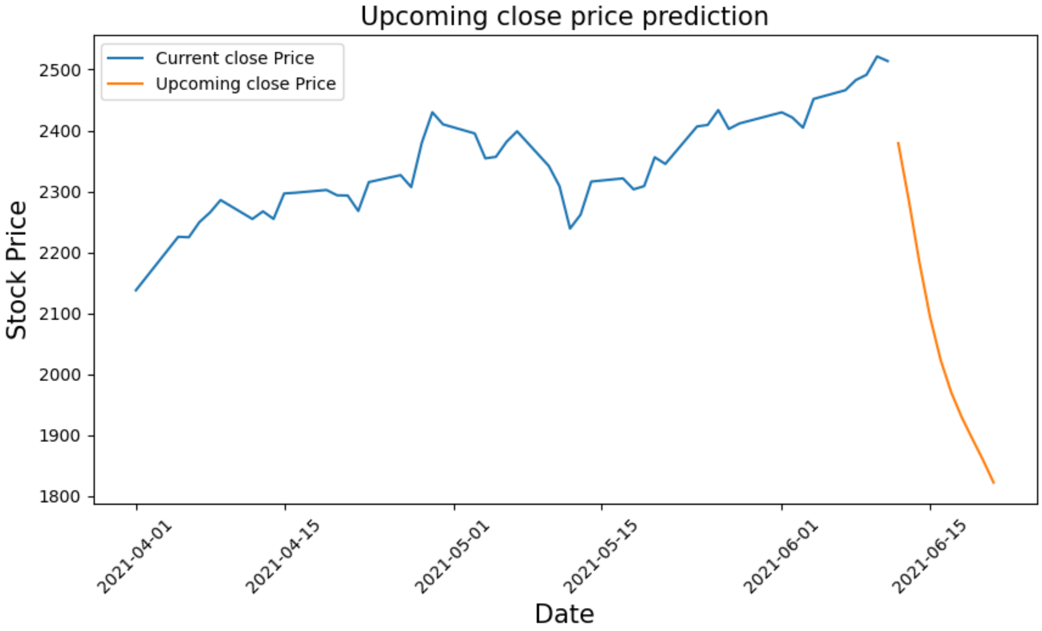
```
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 52ms/step
```

```
upcoming_prediction[['open','close']] = MMS.inverse_transform(upcoming_prediction[['open','close']])
```

```
fig,ax=plt.subplots(figsize=(10,5))
ax.plot(df_merge.loc['2021-04-01':,'open'],label='Current Open Price')
ax.plot(upcoming_prediction.loc['2021-04-01':,'open'],label='Upcoming Open Price')
plt.setp(ax.xaxis.get_majorticklabels(), rotation=45)
ax.set_xlabel('Date',size=15)
ax.set_ylabel('Stock Price',size=15)
ax.set_title('Upcoming Open price prediction',size=15)
ax.legend()
fig.show()
```



```
fig,ax=plt.subplots(figsize=(10,5))
ax.plot(df_merge.loc['2021-04-01':,'close'],label='Current close Price')
ax.plot(upcoming_prediction.loc['2021-04-01':,'close'],label='Upcoming close Price')
plt.setp(ax.xaxis.get_majorticklabels(), rotation=45)
ax.set_xlabel('Date',size=15)
ax.set_ylabel('Stock Price',size=15)
ax.set_title('Upcoming close price prediction',size=15)
ax.legend()
fig.show()
```



● ×