Stock_Price_Prediction_Using_LSTM

IMPORTING LIBRARIES AND DATA TO BE USED

In [2]:
```
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
```

In [3]:
```
df = pd.read_csv('Task_1_Stocks_dataset.csv') # data_importing
df.head(10)
```

Out[3]:

	symbol	date	close	high	low	open	volume	adjClose	adjHigh	ad
0	GOOG	2016-06-14 00:00:00+00:00	718.27	722.47	713.1200	716.48	1306065	718.27	722.47	713.
1	GOOG	2016-06-15 00:00:00+00:00	718.92	722.98	717.3100	719.00	1214517	718.92	722.98	717.
2	GOOG	2016-06-16 00:00:00+00:00	710.36	716.65	703.2600	714.91	1982471	710.36	716.65	703.
3	GOOG	2016-06-17 00:00:00+00:00	691.72	708.82	688.4515	708.65	3402357	691.72	708.82	688.
4	GOOG	2016-06-20 00:00:00+00:00	693.71	702.48	693.4100	698.77	2082538	693.71	702.48	693.
5	GOOG	2016-06-21 00:00:00+00:00	695.94	702.77	692.0100	698.40	1465634	695.94	702.77	692.
6	GOOG	2016-06-22 00:00:00+00:00	697.46	700.86	693.0819	699.06	1184318	697.46	700.86	693.
7	GOOG	2016-06-23 00:00:00+00:00	701.87	701.95	687.0000	697.45	2171415	701.87	701.95	687.
8	GOOG	2016-06-24 00:00:00+00:00	675.22	689.40	673.4500	675.17	4449022	675.22	689.40	673.
9	GOOG	2016-06-27 00:00:00+00:00	668.26	672.30	663.2840	671.00	2641085	668.26	672.30	663.

GATHERING INSIGHTS

In [4]:
```
print("Shape of data:",df.shape)
```

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

In [5]:
```
# statistical description of data
df.describe()
```

Out[5]:

	close	high	low	open	volume	adjClose	a
count	1258.000000	1258.000000	1258.000000	1258.000000	1.258000e+03	1258.000000	1258.(
mean	1216.317067	1227.430934	1204.176430	1215.260779	1.601590e+06	1216.317067	1227.4
std	383.333358	387.570872	378.777094	382.446995	6.960172e+05	383.333358	387.5
min	668.260000	672.300000	663.284000	671.000000	3.467530e+05	668.260000	672.3
25%	960.802500	968.757500	952.182500	959.005000	1.173522e+06	960.802500	968.7
50%	1132.460000	1143.935000	1117.915000	1131.150000	1.412588e+06	1132.460000	1143.9
75%	1360.595000	1374.345000	1348.557500	1361.075000	1.812156e+06	1360.595000	1374.3
max	2521.600000	2526.990000	2498.290000	2524.920000	6.207027e+06	2521.600000	2526.9

In [6]:
```
# summary of data
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
```

In [7]:
```
# checking null values
df.isnull().sum()
```

Out[7]:
```
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
```

There are no null values in the dataset

In [8]:
```
df = df[['date','open','close']] # Extracting required columns
df['date'] = pd.to_datetime(df['date'].apply(lambda x: x.split()[0])) # conv
df.set_index('date',drop=True,inplace=True) # Setting date column as index
df.head(10)
```

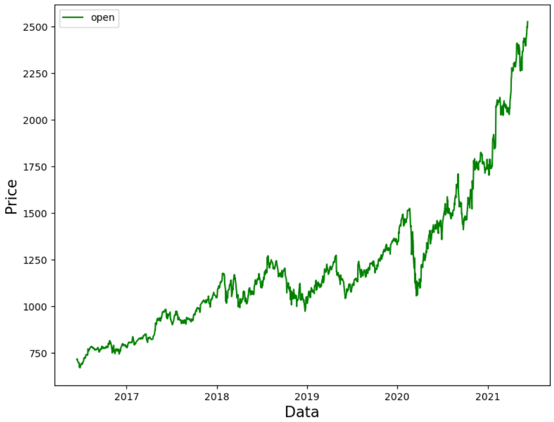
Out[8]:

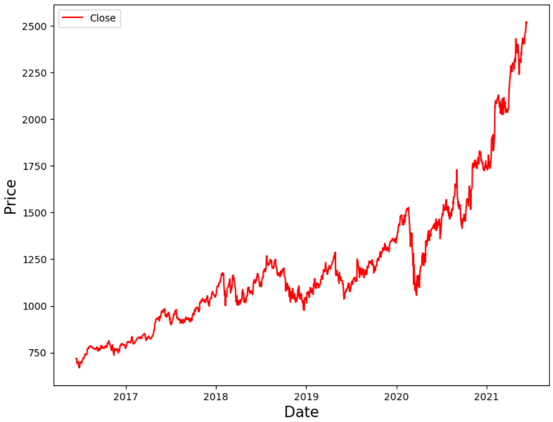
date	open	close
2016-06-14	716.48	718.27
2016-06-15	719.00	718.92
2016-06-16	714.91	710.36
2016-06-17	708.65	691.72
2016-06-20	698.77	693.71
2016-06-21	698.40	695.94
2016-06-22	699.06	697.46
2016-06-23	697.45	701.87
2016-06-24	675.17	675.22
2016-06-27	671.00	668.26

In [9]:
```
# plotting open and closing price on date index
fig, ax = plt.subplots(1,2, figsize=(20,7))
ax[0].plot(df['open'], label = 'open', color = 'green')
ax[0].set_xlabel('Data', 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()
```





DATA PRE-PROCESSING

In [10]:
```
# normalizing all the values of all columns using MinMaxScaler
MMS = MinMaxScaler()
df[df.columns] = MMS.fit_transform(df)
df.head(10)
```

Out[10]:

date	open	close
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

In [11]:
```
# splitting the data into training and test set
training_size = round(len(df) * 0.75) # Selecting 75 % for training and 25 %
training_size
```

Out[11]: 944

In [12]:
```
train_data = df[:training_size]
test_data = df[training_size:]
train_data.shape, test_data.shape
```

Out[12]: ((944, 2), (314, 2))

In [13]:
```
# Function to create sequence of data for training and testing

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))
```

In [14]:
```
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
```

Out[14]: ((894, 50, 2), (894, 2), (264, 50, 2), (264, 2))

CREATING LSTM MODEL

In [15]:

```
# imported Sequential from keras.models
model = Sequential()
# importing Dense, Dropout, LSTM, Bidirectional from keras.layers
model.add(LSTM(units=50, return_sequences = True, input_shape = (train_seq.s

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

model.add(Dense(2))

model.compile(loss = 'mean_squared_error', optimizer = 'adam', metrics = ['m

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: 30902 (120.71 KB)
Trainable params: 30902 (120.71 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [16]:

```
# fitting the model by iterating the dataset over 100 times(100 epochs)
model.fit(train_seq, train_label, epochs = 100, validation_data = (test_seq,
```

```
Epoch 1/100
28/28 [==============================] - 7s 68ms/step - loss: 0.0083 - m
ean_absolute_error: 0.0639 - val_loss: 0.0256 - val_mean_absolute_error:
0.1344
Epoch 2/100
28/28 [==============================] - 1s 34ms/step - loss: 9.1967e-04
- mean_absolute_error: 0.0239 - val_loss: 0.0091 - val_mean_absolute_err
or: 0.0753
Epoch 3/100
28/28 [==============================] - 1s 37ms/step - loss: 4.5380e-04
- mean_absolute_error: 0.0158 - val_loss: 0.0063 - val_mean_absolute_err
or: 0.0613
Epoch 4/100
28/28 [==============================] - 1s 40ms/step - loss: 4.3286e-04
- mean_absolute_error: 0.0152 - val_loss: 0.0072 - val_mean_absolute_err
or: 0.0661
Epoch 5/100
28/28 [==============================] - 1s 33ms/step - loss: 4.2960e-04
- mean_absolute_error: 0.0152 - val_loss: 0.0067 - val_mean_absolute_err
```

In [18]:
```
# predicting the values after running the model
test_predicted = model.predict(test_seq)
test_predicted[:5]
```

```
9/9 [==============================] - 0s 12ms/step
```

Out[18]:
```
array([[0.4047748 , 0.40282473],
       [0.4052958 , 0.40340117],
       [0.40167013, 0.39990655],
       [0.404325  , 0.40235925],
       [0.40851966, 0.4063637 ]], dtype=float32)
```

In [20]:
```
# Inversing normalization/scaling on predicted data
test_inverse_predicted = MMS.inverse_transform(test_predicted)
test_inverse_predicted[:5]
```

Out[20]:
```
array([[1421.42  , 1414.8312],
       [1422.386 , 1415.8995],
       [1415.6642, 1409.4227],
       [1420.5862, 1413.9685],
       [1428.3627, 1421.3901]], dtype=float32)
```

VISUALIZING ACTUAL VS PREDICTED DATA

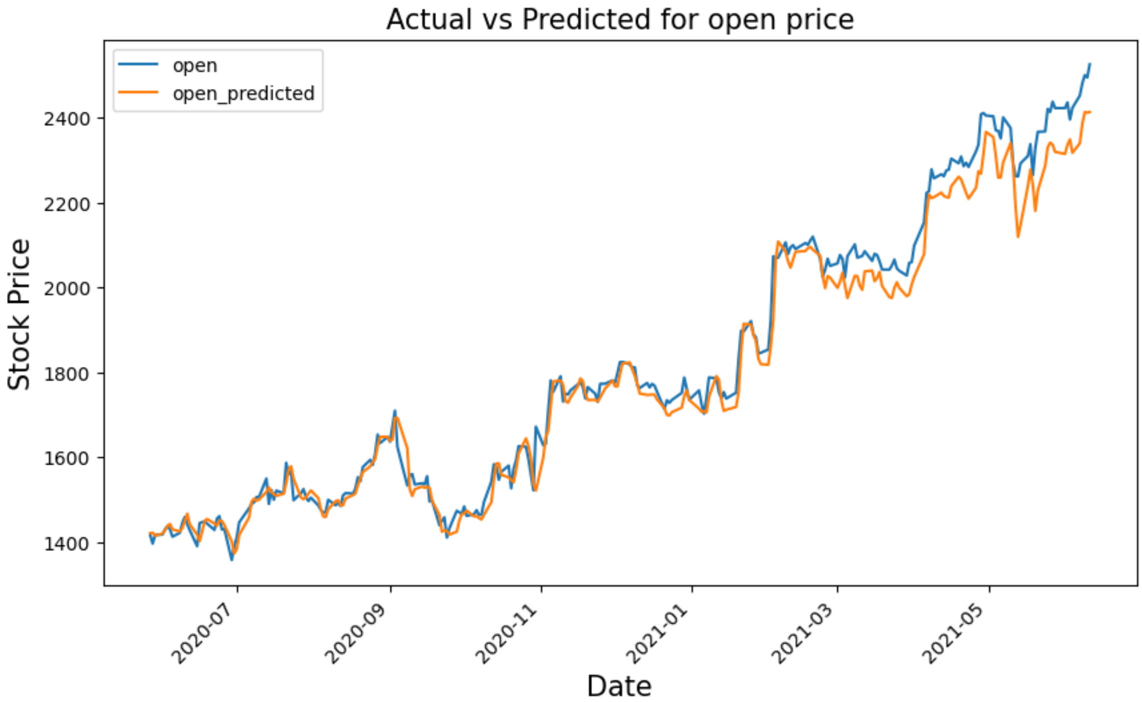
In [26]:
```
# Merging actual and predicted data for better visualization
df_merge = pd.concat([df.iloc[-264:].copy(),
                          pd.DataFrame(test_inverse_predicted,columns=['oper
                                        index=df.iloc[-264:].index)], axis=1)
```

In [27]:
```
# Inversing normalization/scaling
df_merge[['open','close']] = MMS.inverse_transform(df_merge[['open','close']]
df_merge.head()
```

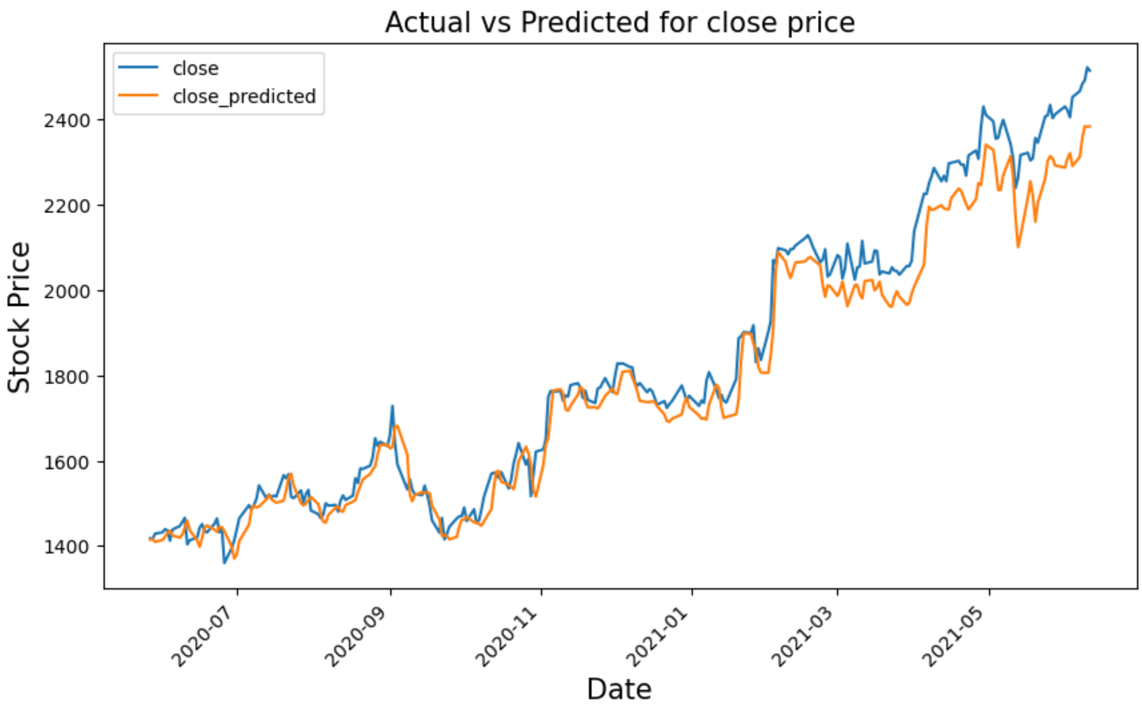
Out[27]:

	open	close	open_predicted	close_predicted
date				
2020-05-27	1417.25	1417.84	1421.420044	1414.831177
2020-05-28	1396.86	1416.73	1422.385986	1415.899536
2020-05-29	1416.94	1428.92	1415.664185	1409.422729
2020-06-01	1418.39	1431.82	1420.586182	1413.968506
2020-06-02	1430.55	1439.22	1428.362671	1421.390137

In [28]:
```
# plotting the actual open and predicted open prices on date index
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()
```



In [29]:
```
# plotting the actual close and predicted close prices on date index
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()
```



PREDICTING UPCOMING 10 DAYS

In [30]:
```
# Creating a dataframe and adding 10 days to existing index

df_merge = df_merge.append(pd.DataFrame(columns=df_merge.columns,
                                         index=pd.date_range(start=df_merge.i
df_merge['2021-06-09':'2021-06-16']
```

Out[30]:

	open	close	open_predicted	close_predicted
2021-06-09	2499.50	2491.40	2412.476074	2383.626465
2021-06-10	2494.01	2521.60	2411.255127	2382.576660
2021-06-11	2524.92	2513.93	2412.763184	2383.162354
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

In [31]:
```
# creating a DataFrame and filling values of open and close column
upcoming_prediction = pd.DataFrame(columns=['open','close'],index=df_merge.i
upcoming_prediction.index=pd.to_datetime(upcoming_prediction.index)
```

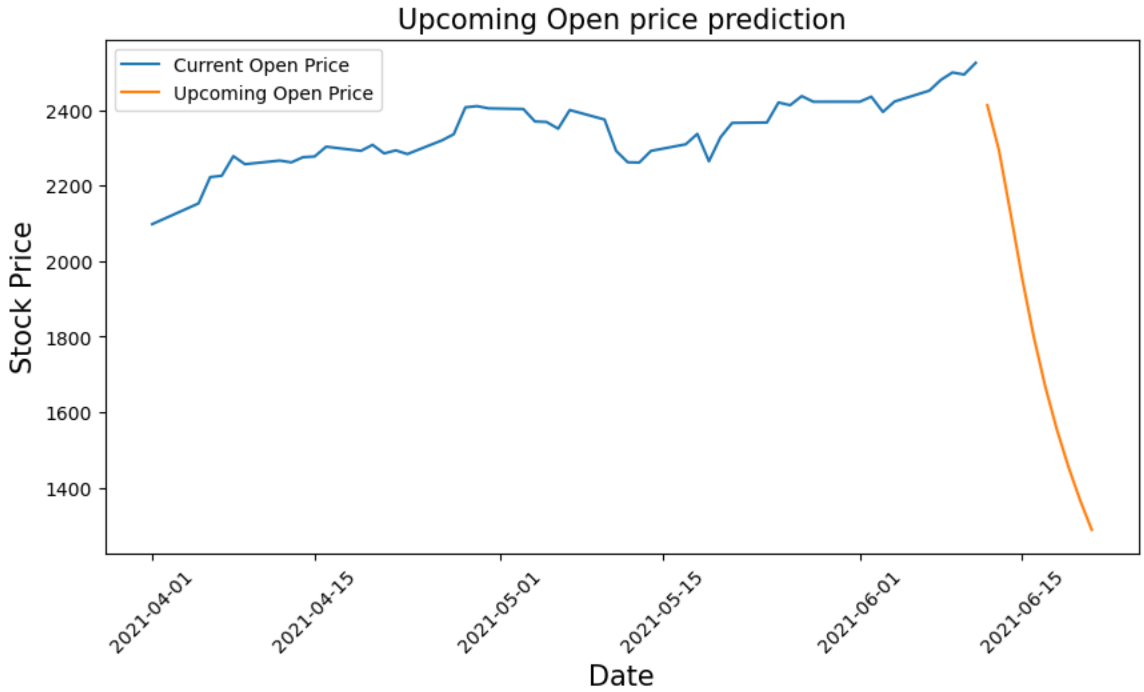
In [32]:
```
curr_seq = test_seq[-1:]

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)
```

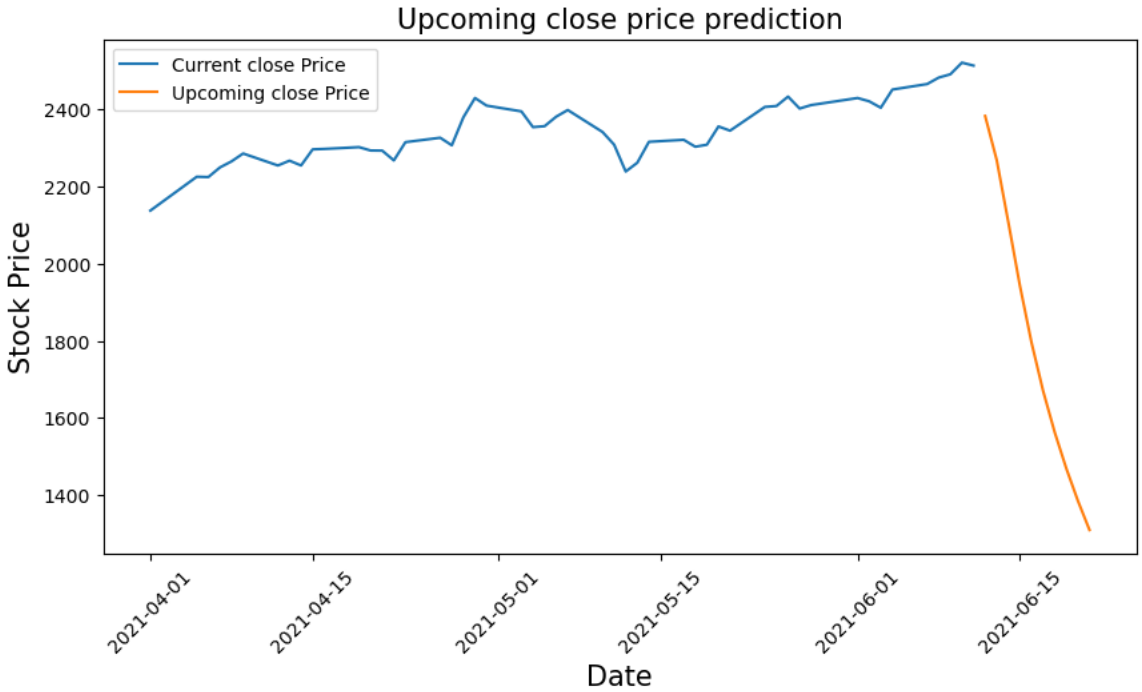
```
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 24ms/step
```

In [33]:
```
# inversing Normalization/scaling
upcoming_prediction[['open','close']] = MMS.inverse_transform(upcoming_predi
```

In [34]:
```
# plotting Upcoming Open price on date index
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 P
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()
```



In [35]:
```
# plotting Upcoming Close price on date index
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
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()
```



THANK YOU!

GitHub: [https://github.com/anujtiwari21?tab=repositories (https://github.com/anujtiwari21?tab=repositories)](https://github.com/anujtiwari21?tab=repositories)