In [1]:

```python
'''TASK 01 - STOCK PREDICTION'''
'''SELECTED COMPANY - GOOGLE'''


# Importing libraries for data analysis and visualization
import numpy as np # For linear algebra operations
import pandas as pd # For data preprocessing and manipulation
import matplotlib.pyplot as plt # For data visualization
import seaborn as sns # For enhanced data visualization
%matplotlib inline

# Ignore warnings for cleaner output
import warnings
warnings.filterwarnings('ignore')

# Importing libraries for machine learning and deep learning
from sklearn.preprocessing import MinMaxScaler # For data normaliza
from keras.models import Sequential # For creating a sequential neu
from keras.layers import Dense, Dropout, LSTM, Bidirectional # For
```

In [2]:

```python
# Data importing: Reading the CSV file into a DataFrame
df = pd.read_csv('G_dataset.csv')

# Fetching the first 10 rows of the dataset for quick inspection
df.head(10)
```

Out[2]:

	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
6	GOOG	2016-06-22 00:00:00+00:00	697.46	700.86	693.0819	699.06	1184318	697.46	700.86
7	GOOG	2016-06-23 00:00:00+00:00	701.87	701.95	687.0000	697.45	2171415	701.87	701.95
8	GOOG	2016-06-24 00:00:00+00:00	675.22	689.40	673.4500	675.17	4449022	675.22	689.40
9	GOOG	2016-06-27 00:00:00+00:00	668.26	672.30	663.2840	671.00	2641085	668.26	672.30

In [3]:

```python
# Printing the shape of the DataFrame (number of rows and columns)
print("Shape of data:", df.shape)
```

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

In [4]:

```python
# Computing the statistical description of the DataFrame
df.describe()
```

Out[4]:

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

In [5]:

```python
# 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 [6]:

```python
# checking null values
df.isnull().sum()
```

Out[6]:

```
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
```

In [7]:

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

Out[7]:

	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

In [8]:

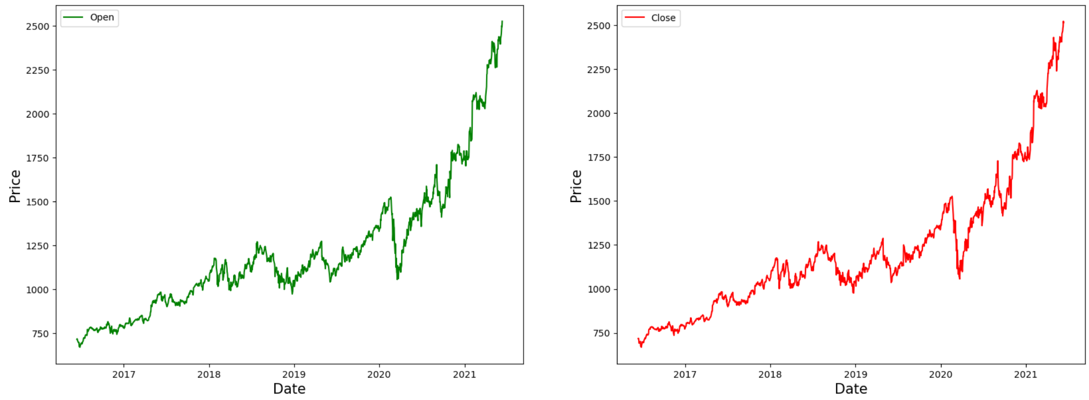
```python
import matplotlib.pyplot as plt

# Creating a figure with two subplots side by side
fig, ax = plt.subplots(1, 2, figsize=(20, 7))

# Plotting the open prices
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()

# Plotting the closing prices
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()

# Displaying the plots
plt.show()
```



In [9]:

```
from sklearn.preprocessing import MinMaxScaler

# Creating a MinMaxScaler object
MMS = MinMaxScaler()

# Applying Min-Max Scaling to normalize all values in the DataFrame
df[df.columns] = MMS.fit_transform(df)

# Displaying the first 10 rows of the normalized DataFrame
df.head(10)
```

Out[9]:

	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

In [10]:

```
# splitting the data into training and test set
training_size = round(len(df) * 0.75) # Selecting 75 % for training
training_size
```

Out[10]: 944

In [11]:

```
# Assuming 'training_size' has been defined before this code snippe
# and represents the number of rows to be used for training the mod

# Slicing the DataFrame 'df' to create 'train_data' containing the
train_data = df[:training_size]

# Slicing the DataFrame 'df' to create 'test_data' containing the r
test_data = df[training_size:]

# Printing the shapes of the newly created 'train_data' and 'test_d
print(train_data.shape, test_data.shape)
```

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

In [12]:

```python
# 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
    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 [13]:

```python
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.shap
```

Out[13]: ((894, 50, 2), (894, 2), (264, 50, 2), (264, 2))

In [14]:

```
# Importing the required modules from Keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM

# Creating a Sequential model
model = Sequential()

# Adding an LSTM layer with 50 units, return_sequences=True is used
# input_shape represents the shape of input sequences in the format
model.add(LSTM(units=50, return_sequences=True, input_shape=(train_

# Adding a Dropout layer to avoid overfitting (10% of the neurons w

model.add(Dropout(0.1))

# Adding another LSTM layer with 50 units (return_sequences=False b
model.add(LSTM(units=50))

# Adding a Dense layer with 2 neurons (output layer)
model.add(Dense(2))

# Compiling the model with mean squared error loss and Adam optimiz
model.compile(loss='mean_squared_error', optimizer='adam', metrics=

# Displaying the summary of the model architecture
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]:
```python
# fitting the model by iterating the dataset over 100 times(100 epo
model.fit(train_seq, train_label, epochs=100,validation_data=(test_
```

```
Epoch 1/100
28/28 [==============================] - 5s 78ms/step - loss: 0.0124
- mean_absolute_error: 0.0791 - val_loss: 0.0436 - val_mean_absolute
_error: 0.1875
Epoch 2/100
28/28 [==============================] - 2s 54ms/step - loss: 0.0011
- mean_absolute_error: 0.0264 - val_loss: 0.0073 - val_mean_absolute
_error: 0.0672
Epoch 3/100
28/28 [==============================] - 1s 50ms/step - loss: 5.1846
e-04 - mean_absolute_error: 0.0170 - val_loss: 0.0033 - val_mean_abs
olute_error: 0.0447
Epoch 4/100
28/28 [==============================] - 1s 51ms/step - loss: 4.3151
e-04 - mean_absolute_error: 0.0150 - val_loss: 0.0032 - val_mean_abs
olute_error: 0.0436
Epoch 5/100
28/28 [==============================] - 2s 55ms/step - loss: 4.3078
e-04 - mean_absolute_error: 0.0151 - val_loss: 0.0029 - val_mean_abs
```

In [20]:
```python
# predicting the values after running the model
test_predicted = model.predict(test_seq)
test_predicted[:5]
```

```
9/9 [==============================] - 1s 15ms/step
```

Out[20]:
```
array([[0.40083   , 0.40118378],
       [0.40117568, 0.40138644],
       [0.39814386, 0.39816135],
       [0.40093723, 0.4010728 ],
       [0.40463322, 0.40483487]], dtype=float32)
```

In [21]:
```python
# Inversing normalization/scaling on predicted data
test_inverse_predicted = MMS.inverse_transform(test_predicted)
test_inverse_predicted[:5]
```

Out[21]:
```
array([[1414.1067, 1411.79  ],
       [1414.7477, 1412.1655],
       [1409.1268, 1406.1884],
       [1414.3055, 1411.5844],
       [1421.1576, 1418.5566]], dtype=float32)
```

In [22]:
```python
# Merging actual and predicted data for better visualization
df_merge = pd.concat([df.iloc[-264:].copy(),
                          pd.DataFrame(test_inverse_predicted,colum
                                       index=df.iloc[-264:].index)]
```

In [23]:

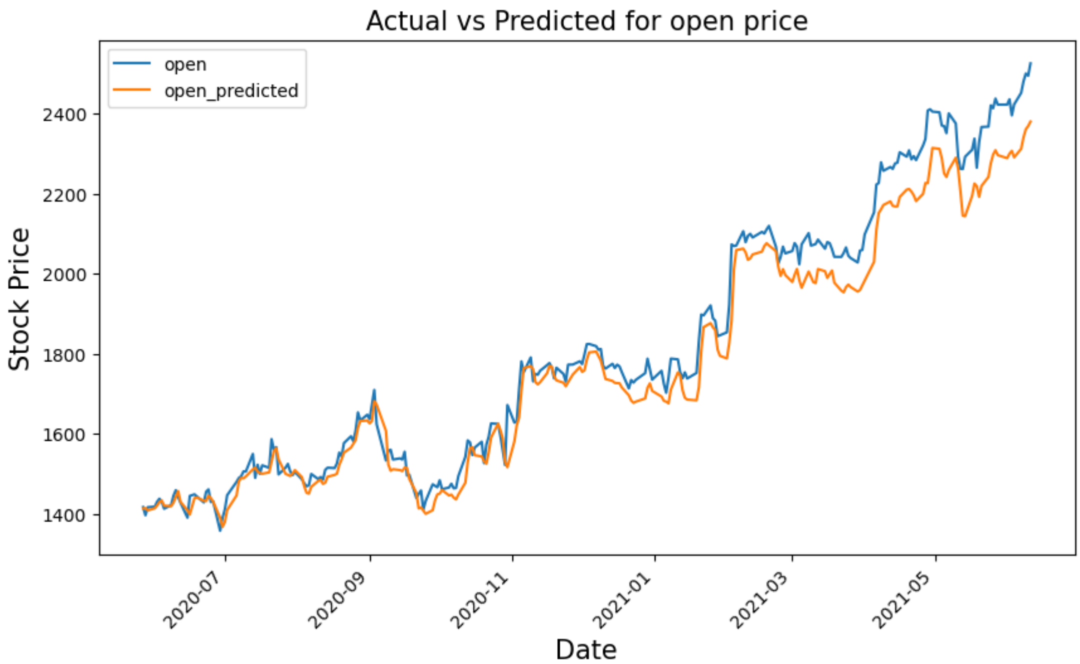
```python
# Inversing normalization/scaling
df_merge[['open','close']] = MMS.inverse_transform(df_merge[['open'
df_merge.head()
```

Out[23]:

	open	close	open_predicted	close_predicted
date				
2020-05-27	1417.25	1417.84	1414.106689	1411.790039
2020-05-28	1396.86	1416.73	1414.747681	1412.165527
2020-05-29	1416.94	1428.92	1409.126831	1406.188354
2020-06-01	1418.39	1431.82	1414.305542	1411.584351
2020-06-02	1430.55	1439.22	1421.157593	1418.556641

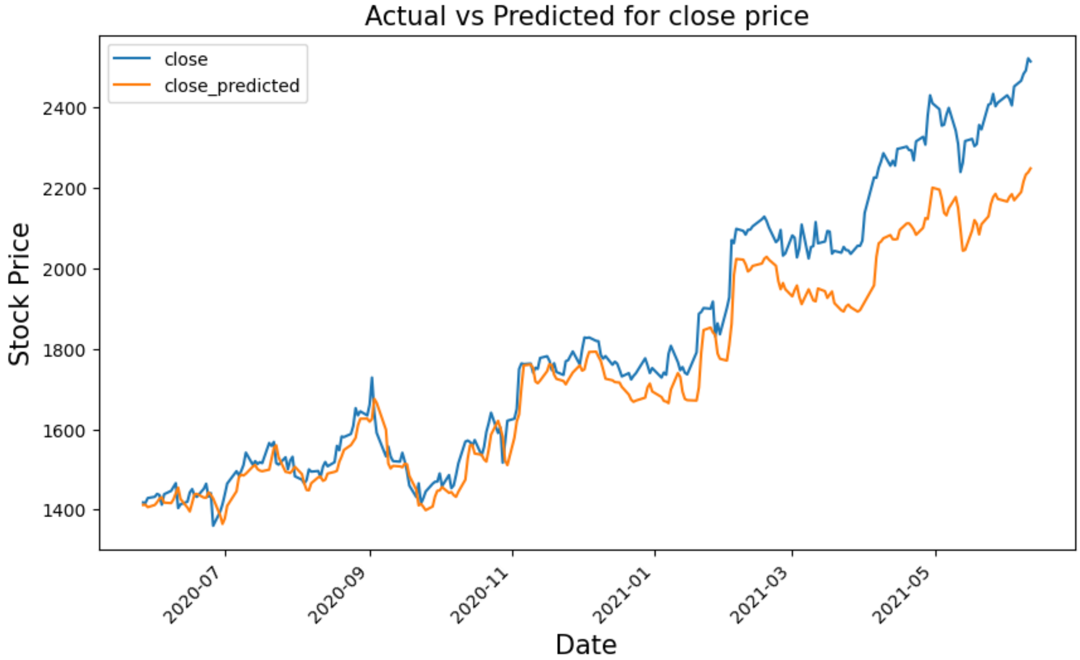
In [24]:

```python
# 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 [25]:

```python
# plotting the actual close and predicted close prices on date inde
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()
```



In [26]:

```python
# 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=d
df_merge['2021-06-09':'2021-06-16']
```

Out[26]:

	open	close	open_predicted	close_predicted
2021-06-09	2499.50	2491.40	2360.455811	2232.811279
2021-06-10	2494.01	2521.60	2368.316650	2238.388428
2021-06-11	2524.92	2513.93	2379.575439	2248.059326
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 [27]:

```python
# creating a DataFrame and filling values of open and close column
upcoming_prediction = pd.DataFrame(columns=['open','close'],index=d
upcoming_prediction.index=pd.to_datetime(upcoming_prediction.index)
```

In [28]:

```
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)
```

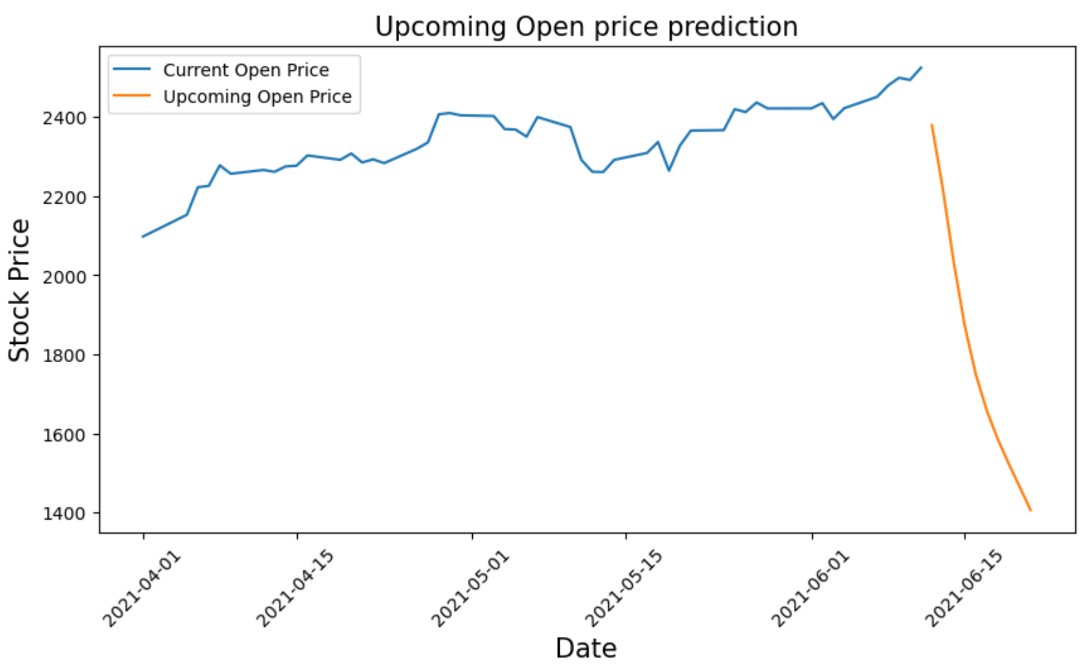
```
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
```

In [30]:

```
# inversing Normalization/scaling
upcoming_prediction[['open','close']] = MMS.inverse_transform(upcom
```

In [31]:

```
# 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 Pric
ax.plot(upcoming_prediction.loc['2021-04-01':,'open'],label='Upcomi
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 [32]:

```
# 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 Pr
ax.plot(upcoming_prediction.loc['2021-04-01':,'close'],label='Upcom
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()

```