```
TASK 1 : STOCK PREDICTION
PURPOSE : TO PREDICT THE STOCK PRICE OF A COMPANY USING LSTM.
ABOUT DATASET
Google Stock Prediction
This dataset contains historical data of Googles stock prices and related attributes.
It consists of 14 columns and a smaller subset of 1257 rows. Each column represents a specific attribute,
and each row contains the corresponding values for that attribute.

The columns in the dataset are as follows:

Symbol: The name of the company, which is GOOG in this case.
Date: The year and date of the stock data.
Close: The closing price of Google stock on a particular day.
High: The highest value reached by Google stock on the given day.
Low: The lowest value reached by Google stock on the given day.
Open: The opening value of Google stock on the given day.
Volume: The trading volume of Google stock on the given day, i.e., the number of shares traded.
adjClose: The adjusted closing price of Google stock, considering factors such as dividends and stock splits.
adjHigh: The adjusted highest value reached by Google stock on the given day.
adjLow: The adjusted lowest value reached by Google stock on the given day.
adjOpen: The adjusted opening value of Google stock on the given day.
adjVolume: The adjusted trading volume of Google stock on the given day, accounting for factors such as stock splits.
divCash: The amount of cash dividend paid out to shareholders on the given day.
splitFactor: The split factor, if any, applied to Google stock on the given day. A split factor of 1 indicates no split.

The dataset is available at Kaggle : https://www.kaggle.com/datasets/shreenidhihipparagi/google-stock-prediction
```

```
STEPS INVOLVED :

1. IMPORTING LIBRARIES AND DATA TO BE USED
2. GATHERING INSIGHTS
3. DATA PRE-PROCESSING
4. CREATING LSTM MODEL
5. VISUALIZING ACTUAL VS PREDICTED DATA
6. PREDICTING UPCOMING 15 DAYS
```

```
STEP 1 : IMPORTING LIBRARIES AND DATA TO BE USED
```

```
#importing libraries to be used
import numpy as np # for linear algebra
import pandas as pd # data preprocessing
import matplotlib.pyplot as plt # data visualization library
import seaborn as sns # data visualization library
%matplotlib inline
import warnings
warnings.filterwarnings('ignore') # ignore warnings

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

```
df = pd.read_csv('GOOG.csv') # data_importing
df.head(10) # fetching first 10 rows of dataset
```

	symbol	date	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	adjVolume	divCash	splitFac
0	GOOG	2016-06-14 00:00:00+00:00	718.27	722.47	713.1200	716.48	1306065	718.27	722.47	713.1200	716.48	1306065	0.0	

```
STEP 2 : GATHERING INSIGHTS
```

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

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

```
# statistical description of data
df.describe()
```

	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	adjVolum
count	1258.000000	1258.000000	1258.000000	1258.000000	1.258000e+03	1258.000000	1258.000000	1258.000000	1258.000000	1.258000e+(
mean	1216.317067	1227.430934	1204.176430	1215.260779	1.601590e+06	1216.317067	1227.430936	1204.176436	1215.260779	1.601590e+(
std	383.333358	387.570872	378.777094	382.446995	6.960172e+05	383.333358	387.570873	378.777099	382.446995	6.960172e+(
min	668.260000	672.300000	663.284000	671.000000	3.467530e+05	668.260000	672.300000	663.284000	671.000000	3.467530e+(
25%	960.802500	968.757500	952.182500	959.005000	1.173522e+06	960.802500	968.757500	952.182500	959.005000	1.173522e+(
50%	1132.460000	1143.935000	1117.915000	1131.150000	1.412588e+06	1132.460000	1143.935000	1117.915000	1131.150000	1.412588e+(
75%	1360.595000	1374.345000	1348.557500	1361.075000	1.812156e+06	1360.595000	1374.345000	1348.557500	1361.075000	1.812156e+(
max	2521.600000	2526.990000	2498.290000	2524.920000	6.207027e+06	2521.600000	2526.990000	2498.290000	2524.920000	6.207027e+(

```
# 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
```

```
# checking null values
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
```

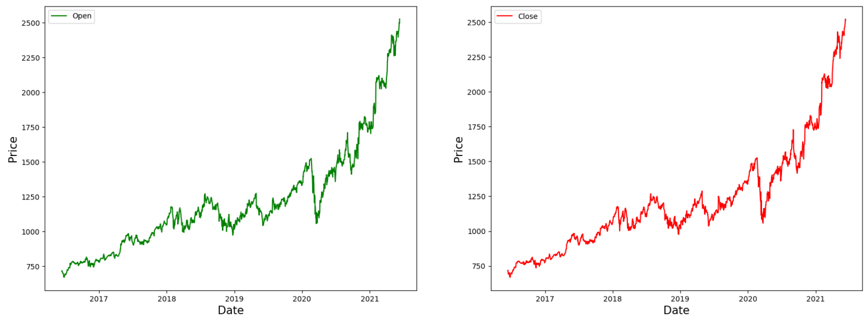
```
df = df[['date','open','close']] # Extracting required columns
df['date'] = pd.to_datetime(df['date'].apply(lambda x: x.split()[0])) # converting object dtype of date column to datetime dtype
df.set_index('date',drop=True,inplace=True) # Setting date column as index
df.head(10)
```

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

```
# 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('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()
```




```
STEP 3 : DATA PRE-PROCESSING
```

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

open close

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

```
944
```

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

train_data.shape, test_data.shape
```

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

2016-06-23 0.014267 0.018135

```
# 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))
```

```
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))
```

STEP 4 : CREATING LSTM MODEL

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

```
# fitting the model by iterating the dataset over 100 times(100 epochs)
model.fit(train_seq, train_label, epochs=100,validation_data=(test_seq, test_label), verbose=1)
```

```
=================] - 1s 51ms/step - loss: 1.4478e-04 - mean_absolute_error: 0.0086 - val_loss: 0.0013 - val_mean_absolute_error:

=================] - 1s 50ms/step - loss: 1.4071e-04 - mean_absolute_error: 0.0086 - val_loss: 0.0013 - val_mean_absolute_error:

=================] - 1s 48ms/step - loss: 1.2989e-04 - mean_absolute_error: 0.0082 - val_loss: 0.0016 - val_mean_absolute_error:

=================] - 1s 49ms/step - loss: 1.4151e-04 - mean_absolute_error: 0.0086 - val_loss: 0.0015 - val_mean_absolute_error:

=================] - 1s 49ms/step - loss: 1.4548e-04 - mean_absolute_error: 0.0089 - val_loss: 0.0033 - val_mean_absolute_error:

=================] - 1s 49ms/step - loss: 1.5965e-04 - mean_absolute_error: 0.0091 - val_loss: 0.0033 - val_mean_absolute_error:

=================] - 2s 56ms/step - loss: 1.4431e-04 - mean_absolute_error: 0.0088 - val_loss: 0.0018 - val_mean_absolute_error:

=================] - 2s 88ms/step - loss: 1.4037e-04 - mean_absolute_error: 0.0086 - val_loss: 0.0014 - val_mean_absolute_error:

=================] - 1s 52ms/step - loss: 1.5790e-04 - mean_absolute_error: 0.0090 - val_loss: 0.0010 - val_mean_absolute_error:

=================] - 1s 49ms/step - loss: 1.4527e-04 - mean_absolute_error: 0.0087 - val_loss: 0.0016 - val_mean_absolute_error:

=================] - 1s 49ms/step - loss: 1.3490e-04 - mean_absolute_error: 0.0083 - val_loss: 0.0018 - val_mean_absolute_error:

=================] - 1s 49ms/step - loss: 1.3471e-04 - mean_absolute_error: 0.0083 - val_loss: 0.0011 - val_mean_absolute_error:

=================] - 1s 51ms/step - loss: 1.3137e-04 - mean_absolute_error: 0.0082 - val_loss: 0.0016 - val_mean_absolute_error:

=================] - 1s 50ms/step - loss: 1.6265e-04 - mean_absolute_error: 0.0094 - val_loss: 0.0023 - val_mean_absolute_error:

=================] - 1s 49ms/step - loss: 1.4543e-04 - mean_absolute_error: 0.0087 - val_loss: 0.0019 - val_mean_absolute_error:

=================] - 2s 82ms/step - loss: 1.2375e-04 - mean_absolute_error: 0.0081 - val_loss: 0.0022 - val_mean_absolute_error:

=================] - 2s 57ms/step - loss: 1.2814e-04 - mean_absolute_error: 0.0081 - val_loss: 0.0028 - val_mean_absolute_error:

=================] - 1s 48ms/step - loss: 1.2366e-04 - mean_absolute_error: 0.0079 - val_loss: 0.0011 - val_mean_absolute_error:

=================] - 1s 50ms/step - loss: 1.2398e-04 - mean_absolute_error: 0.0080 - val_loss: 0.0012 - val_mean_absolute_error:

=================] - 1s 49ms/step - loss: 1.2106e-04 - mean_absolute_error: 0.0079 - val_loss: 0.0013 - val_mean_absolute_error:

=================] - 1s 48ms/step - loss: 1.2974e-04 - mean_absolute_error: 0.0083 - val_loss: 0.0011 - val_mean_absolute_error:

=================] - 1s 50ms/step - loss: 1.2799e-04 - mean_absolute_error: 0.0082 - val_loss: 6.6583e-04 - val_mean_absolute_err

=================] - 1s 50ms/step - loss: 1.3045e-04 - mean_absolute_error: 0.0084 - val_loss: 0.0011 - val_mean_absolute_error:

=================] - 2s 70ms/step - loss: 1.2283e-04 - mean_absolute_error: 0.0080 - val_loss: 7.2310e-04 - val_mean_absolute_err
.History at 0x78bb28903280>
```

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

```
9/9 [==============================] - 1s 23ms/step
array([[0.407208  , 0.41572613],
       [0.4070375 , 0.4155994 ],
       [0.402454  , 0.41112638],
       [0.4048601 , 0.41352707],
       [0.40856364, 0.41741836]], dtype=float32)
```

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

```
array([[1425.931 , 1438.742 ],
       [1425.615 , 1438.5071],
       [1417.1174, 1430.217 ],
       [1421.5782, 1434.6663],
       [1428.4442, 1441.8782]], dtype=float32)
```

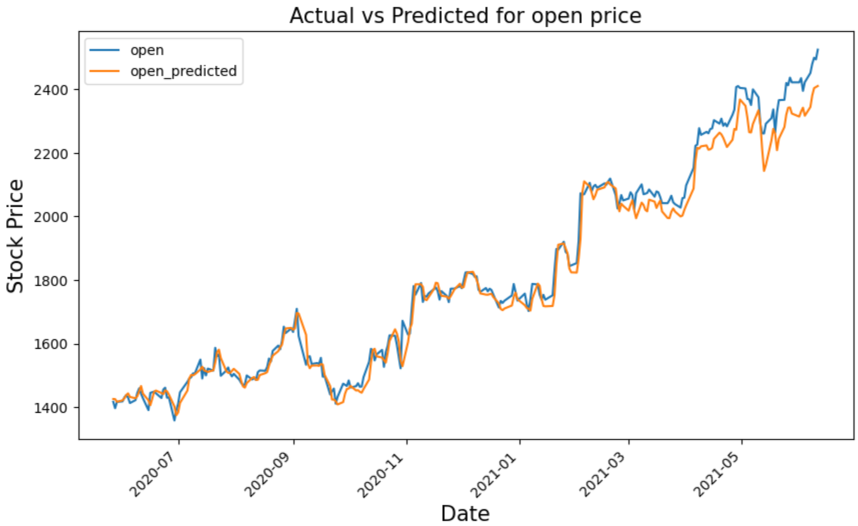
STEP 5 : VISUALIZING ACTUAL VS PREDICTED DATA

```
# Merging actual and predicted data for better visualization
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)
```

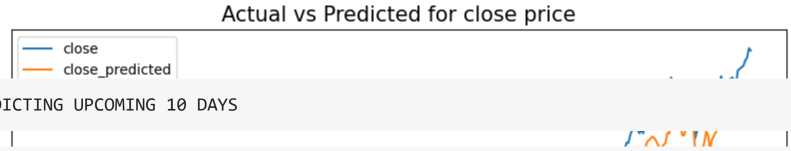
```
# Inversing normalization/scaling
df_merge[['open','close']] = MMS.inverse_transform(df_merge[['open','close']])
df_merge.head()
```

date	open	close	open_predicted	close_predicted
2020-05-27	1417.25	1417.84	1425.931030	1438.741943
2020-05-28	1396.86	1416.73	1425.614990	1438.507080
2020-05-29	1416.94	1428.92	1417.117432	1430.217041
2020-06-01	1418.39	1431.82	1421.578247	1434.666260

```
# 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()
```




```
# 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()
```




```
STEP 6. PREDICTING UPCOMING 10 DAYS
```

```
# 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.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	2403.844482	2373.338623
2021-06-10	2494.01	2521.60	2407.136719	2376.198975
2021-06-11	2524.92	2513.93	2410.606934	2379.103760
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

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

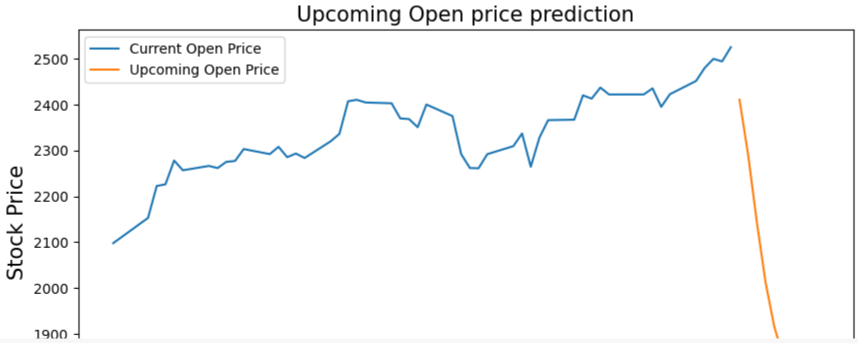
```
curr_seq = test_seq[-1:]

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 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 33ms/step
```

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

```
# 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 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()
```




```
# 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 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()
```