```
import warnings
warnings.filterwarnings('ignore')
# Importing required libraries
import pandas as p
import numpy as n
import matplotlib.pyplot as plt


data = p.read_csv("/content/GOOG.csv")
data
```

| | symbol | date | close | high | low | open | volume | adjClose | adjHigh | adjLow | adjOpen | adjVolume |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | GOOG | 2016-06-14 00:00:00+00:00 | 718.27 | 722.470 | 713.1200 | 716.48 | 1306065 | 718.27 | 722.470 | 713.1200 | 716.48 | 1306065 |
| 1 | GOOG | 2016-06-15 00:00:00+00:00 | 718.92 | 722.980 | 717.3100 | 719.00 | 1214517 | 718.92 | 722.980 | 717.3100 | 719.00 | 1214517 |
| 2 | GOOG | 2016-06-16 00:00:00+00:00 | 710.36 | 716.650 | 703.2600 | 714.91 | 1982471 | 710.36 | 716.650 | 703.2600 | 714.91 | 1982471 |
| 3 | GOOG | 2016-06-17 00:00:00+00:00 | 691.72 | 708.820 | 688.4515 | 708.65 | 3402357 | 691.72 | 708.820 | 688.4515 | 708.65 | 3402357 |
| 4 | GOOG | 2016-06-20 00:00:00+00:00 | 693.71 | 702.480 | 693.4100 | 698.77 | 2082538 | 693.71 | 702.480 | 693.4100 | 698.77 | 2082538 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1253 | GOOG | 2021-06-07 00:00:00+00:00 | 2466.09 | 2468.000 | 2441.0725 | 2451.32 | 1192453 | 2466.09 | 2468.000 | 2441.0725 | 2451.32 | 1192453 |
| 1254 | GOOG | 2021-06-08 00:00:00+00:00 | 2482.85 | 2494.495 | 2468.2400 | 2479.90 | 1253253 | 2482.85 | 2494.495 | 2468.2400 | 2479.90 | 1253253 |
| 1255 | GOOG | 2021-06-09 00:00:00+00:00 | 2491.40 | 2505.000 | 2487.3300 | 2499.50 | 1006337 | 2491.40 | 2505.000 | 2487.3300 | 2499.50 | 1006337 |
| 1256 | GOOG | 2021-06-10 00:00:00+00:00 | 2521.60 | 2523.260 | 2494.0000 | 2494.01 | 1561733 | 2521.60 | 2523.260 | 2494.0000 | 2494.01 | 1561733 |
| 1257 | GOOG | 2021-06-11 00:00:00+00:00 | 2513.93 | 2526.990 | 2498.2900 | 2524.92 | 1262309 | 2513.93 | 2526.990 | 2498.2900 | 2524.92 | 1262309 |

1258 rows × 14 columns

```
data.head()
```

| | symbol | date | close | high | low | open | volume | adjClose | adjHigh | adjLow | adjOpen | adjVolume | divCash |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |
| 1 | GOOG | 2016-06-15 00:00:00+00:00 | 718.92 | 722.98 | 717.3100 | 719.00 | 1214517 | 718.92 | 722.98 | 717.3100 | 719.00 | 1214517 | 0.0 |
| 2 | GOOG | 2016-06-16 00:00:00+00:00 | 710.36 | 716.65 | 703.2600 | 714.91 | 1982471 | 710.36 | 716.65 | 703.2600 | 714.91 | 1982471 | 0.0 |
| 3 | GOOG | 2016-06-17 00:00:00+00:00 | 691.72 | 708.82 | 688.4515 | 708.65 | 3402357 | 691.72 | 708.82 | 688.4515 | 708.65 | 3402357 | 0.0 |
| 4 | GOOG | 2016-06-20 00:00:00+00:00 | 693.71 | 702.48 | 693.4100 | 698.77 | 2082538 | 693.71 | 702.48 | 693.4100 | 698.77 | 2082538 | 0.0 |

```
data.tail()
```

| | symbol | date | close | high | low | open | volume | adjClose | adjHigh | adjLow | adjOpen | adjVolume |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1253 | GOOG | 2021-06-07 00:00:00+00:00 | 2466.09 | 2468.000 | 2441.0725 | 2451.32 | 1192453 | 2466.09 | 2468.000 | 2441.0725 | 2451.32 | 1192453 |
| 1254 | GOOG | 2021-06-08 00:00:00+00:00 | 2482.85 | 2494.495 | 2468.2400 | 2479.90 | 1253253 | 2482.85 | 2494.495 | 2468.2400 | 2479.90 | 1253253 |
| 1255 | GOOG | 2021-06-09 00:00:00+00:00 | 2491.40 | 2505.000 | 2487.3300 | 2499.50 | 1006337 | 2491.40 | 2505.000 | 2487.3300 | 2499.50 | 1006337 |
| 1256 | GOOG | 2021-06-10 00:00:00+00:00 | 2521.60 | 2523.260 | 2494.0000 | 2494.01 | 1561733 | 2521.60 | 2523.260 | 2494.0000 | 2494.01 | 1561733 |
| 1257 | GOOG | 2021-06-11 00:00:00+00:00 | 2513.93 | 2526.990 | 2498.2900 | 2524.92 | 1262309 | 2513.93 | 2526.990 | 2498.2900 | 2524.92 | 1262309 |

```
data.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
```

```
data.describe()
```

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

```
#Required columns
data = data[['date','open','close']]
#Selecting only date
data['date'] = p.to_datetime(data['date'].apply(lambda x: x.split()[0]))
data.set_index('date',drop=True,inplace=True)


fg, ax =plt.subplots(1,2,figsize=(25,7))

ax[0].plot(data['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(data['close'],label='Close',color='red')
ax[1].set_xlabel('Date',size=15)
ax[1].set_ylabel('Price',size=15)
ax[1].legend()

fg.show()
```


**Data Pre-Processing**

```
from sklearn.preprocessing import MinMaxScaler
MMS = MinMaxScaler()
data[data.columns] = MMS.fit_transform(data)


data.shape
```

```
(1258, 2)
```

```
train_size = round(len(data)*0.8)
train_size
```

```
1006
```

```
train_data = data[:train_size]
test_data  = data[train_size:]

print(train_data.shape, test_data.shape)
```

```
(1006, 2) (252, 2)
```

### 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)):
    sequences.append(dataset.iloc[start_idx:stop_idx])
    labels.append(dataset.iloc[stop_idx])
    start_idx += 1
  return (n.array(sequences),n.array(labels))


train_seq, train_label = create_sequence(train_data)
test_seq, test_label = create_sequence(test_data)


print(train_seq.shape, train_label.shape, test_seq.shape, test_label.shape)
```

```
(956, 50, 2) (956, 2) (202, 50, 2) (202, 2)
```

### Creating LSTM model

### Importing the libraries

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

### Building the model

```
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_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_2 (LSTM)               (None, 50, 50)            10600

 dropout_1 (Dropout)         (None, 50, 50)            0

 lstm_3 (LSTM)               (None, 50)                20200

 dense_1 (Dense)             (None, 2)                 102

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

### Running with 80 Epcohs

```
model.fit(train_seq, train_label, epochs=80,validation_data=(test_seq, test_label), verbose=1)
```

```
30/30 [==============================] - 1s 48ms/step - loss: 1.7867e-04 - mean_absolute_error: 0.0097 - val_loss: 0.0040 - val_m
Epoch 56/80
30/30 [==============================] - 1s 46ms/step - loss: 1.6729e-04 - mean_absolute_error: 0.0095 - val_loss: 0.0041 - val_m
Epoch 57/80
30/30 [==============================] - 1s 46ms/step - loss: 1.6513e-04 - mean_absolute_error: 0.0093 - val_loss: 0.0081 - val_m
Epoch 58/80
30/30 [==============================] - 2s 69ms/step - loss: 1.6453e-04 - mean_absolute_error: 0.0092 - val_loss: 0.0056 - val_m
Epoch 59/80
30/30 [==============================] - 2s 64ms/step - loss: 2.0450e-04 - mean_absolute_error: 0.0106 - val_loss: 0.0028 - val_m
Epoch 60/80
30/30 [==============================] - 1s 46ms/step - loss: 1.7807e-04 - mean_absolute_error: 0.0097 - val_loss: 0.0022 - val_m
Epoch 61/80
30/30 [==============================] - 1s 49ms/step - loss: 1.5869e-04 - mean_absolute_error: 0.0091 - val_loss: 0.0052 - val_m
Epoch 62/80
30/30 [==============================] - 1s 49ms/step - loss: 1.6487e-04 - mean_absolute_error: 0.0092 - val_loss: 0.0027 - val_m
Epoch 63/80
30/30 [==============================] - 1s 49ms/step - loss: 1.6994e-04 - mean_absolute_error: 0.0095 - val_loss: 0.0052 - val_m
Epoch 64/80
30/30 [==============================] - 1s 49ms/step - loss: 1.5746e-04 - mean_absolute_error: 0.0090 - val_loss: 0.0054 - val_m
Epoch 65/80
30/30 [==============================] - 1s 47ms/step - loss: 1.5238e-04 - mean_absolute_error: 0.0089 - val_loss: 0.0033 - val_m
Epoch 66/80
30/30 [==============================] - 2s 64ms/step - loss: 1.4174e-04 - mean_absolute_error: 0.0085 - val_loss: 0.0033 - val_m
Epoch 67/80
30/30 [==============================] - 2s 67ms/step - loss: 1.6695e-04 - mean_absolute_error: 0.0095 - val_loss: 0.0030 - val_m
Epoch 68/80
30/30 [==============================] - 1s 49ms/step - loss: 1.4484e-04 - mean_absolute_error: 0.0087 - val_loss: 0.0036 - val_m
Epoch 69/80
30/30 [==============================] - 1s 47ms/step - loss: 1.4488e-04 - mean_absolute_error: 0.0087 - val_loss: 0.0048 - val_m
Epoch 70/80
30/30 [==============================] - 1s 47ms/step - loss: 1.3814e-04 - mean_absolute_error: 0.0086 - val_loss: 0.0018 - val_m
Epoch 71/80
30/30 [==============================] - 1s 47ms/step - loss: 1.6009e-04 - mean_absolute_error: 0.0092 - val_loss: 0.0049 - val_m
Epoch 72/80
30/30 [==============================] - 1s 45ms/step - loss: 1.5752e-04 - mean_absolute_error: 0.0091 - val_loss: 0.0030 - val_m
Epoch 73/80
30/30 [==============================] - 1s 46ms/step - loss: 1.4828e-04 - mean_absolute_error: 0.0088 - val_loss: 0.0045 - val_m
Epoch 74/80
30/30 [==============================] - 2s 55ms/step - loss: 1.4206e-04 - mean_absolute_error: 0.0086 - val_loss: 0.0071 - val_m
Epoch 75/80
30/30 [==============================] - 2s 78ms/step - loss: 1.3556e-04 - mean_absolute_error: 0.0085 - val_loss: 0.0028 - val_m
Epoch 76/80
30/30 [==============================] - 1s 46ms/step - loss: 1.3279e-04 - mean_absolute_error: 0.0083 - val_loss: 0.0029 - val_m
Epoch 77/80
30/30 [==============================] - 1s 49ms/step - loss: 1.3425e-04 - mean_absolute_error: 0.0082 - val_loss: 0.0037 - val_m
Epoch 78/80
30/30 [==============================] - 1s 49ms/step - loss: 1.2834e-04 - mean_absolute_error: 0.0081 - val_loss: 0.0028 - val_m
Epoch 79/80
30/30 [==============================] - 1s 47ms/step - loss: 1.3716e-04 - mean_absolute_error: 0.0083 - val_loss: 0.0028 - val_m
Epoch 80/80
30/30 [==============================] - 1s 47ms/step - loss: 1.2542e-04 - mean_absolute_error: 0.0081 - val_loss: 0.0052 - val_m
<keras.src.callbacks.History at 0x7f682b8f1ff0>
```

**Test Predicted**

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

```
7/7 [==============================] - 1s 12ms/step
array([[0.47375992, 0.47437757],
       [0.47986326, 0.4804546 ],
       [0.48490945, 0.48543447],
       [0.50077134, 0.50026995],
       [0.51046824, 0.5098664 ]], dtype=float32)
```

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

```
array([[1549.313 , 1547.443 ],
       [1560.628 , 1558.7057],
       [1569.9833, 1567.9352],
       [1599.39  , 1595.4303],
       [1617.3673, 1613.2158]], dtype=float32)
```

**Visualizing predicted and actual data**

```
# Merging actual and predicted data for better visualization
gs_slic_data = pd.concat([data.iloc[-202:].copy(),p.DataFrame(test_inverse_predicted,columns=['open_predicted','close_predicted'],index=d
```

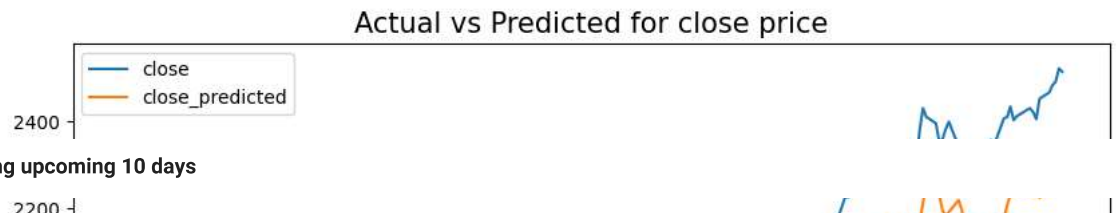
```
gs_slic_data[['open','close']] = MMS.inverse_transform(gs_slic_data[['open','close']]) # Inverse scaling
```

```
gs_slic_data.head()
```

| date | open | close | open_predicted | close_predicted |
|---|---|---|---|---|
| **2020-08-24** | 1593.98 | 1588.20 | 1549.312988 | 1547.442993 |
| **2020-08-25** | 1582.07 | 1608.22 | 1560.628052 | 1558.705688 |
| **2020-08-26** | 1608.00 | 1652.38 | 1569.983276 | 1567.935181 |
| **2020-08-27** | 1653.68 | 1634.33 | 1599.390015 | 1595.430298 |
| **2020-08-28** | 1633.49 | 1644.41 | 1617.367310 | 1613.215820 |

```
gs_slic_data[['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()
```


```
gs_slic_data[['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**

```
# Creating a dataframe and adding 10 days to existing index
gs_slic_data = gs_slic_data.append(p.DataFrame(columns=gs_slic_data.columns,index=p.date_range(start=gs_slic_data.index[-1], periods=11,
```

```
gs_slic_data['2021-06-09    ':'2021-06-16']
```

| | open | close | open_predicted | close_predicted |
|---|---|---|---|---|
| **2021-06-09** | 2499.50 | 2491.40 | 2193.927734 | 2292.554443 |
| **2021-06-10** | 2494.01 | 2521.60 | 2203.161377 | 2304.351074 |
| **2021-06-11** | 2524.92 | 2513.93 | 2213.244873 | 2316.301025 |
| **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=gs_slic_data.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 27ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 25ms/step
```

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

```
fg,ax=plt.subplots(figsize=(10,5))
ax.plot(gs_slic_data.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()
fg.show()
```


```
fg,ax=plt.subplots(figsize=(10,5))
ax.plot(gs_slic_data.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()
fg.show()
```