```
Name: Sanket Shelke
Roll No: COBB079
Subject: DL
Lab Assignment: 4
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files 2 files

- **Google_Stock_Price_Test.csv**(text/csv) - 7978 bytes, last modified: 4/7/2024 - 100% done
- **Google_Stock_Price_Train.csv**(text/csv) - 77979 bytes, last modified: 4/7/2024 - 100% done

```
Saving Google_Stock_Price_Test.csv to Google_Stock_Price_Test.csv
Saving Google_Stock_Price_Train.csv to Google_Stock_Price_Train.csv
```

```
dataset_train = pd.read_csv('Google_Stock_Price_Train.csv')
dataset_train.head()
```

	Date	Open	High	Low	Close	Volume
0	01/03/2012	325.25	332.83	324.97	663.59	7,380,500
1	01/04/2012	331.27	333.87	329.08	666.45	5,749,400
2	01/05/2012	329.83	330.75	326.89	657.21	6,590,300
3	01/06/2012	328.34	328.77	323.68	648.24	5,405,900
4	01/09/2012	322.04	322.29	309.46	620.76	11,688,800

Next steps: Generate code with dataset_train View recommended plots

```
#keras only takes numpy array
training_set = dataset_train.iloc[:, 1: 2].values
training_set.shape
```

```
(1509, 1)
```

```
sc = MinMaxScaler(feature_range = (0, 1))
#fit: get min/max of train data
training_set_scaled = sc.fit_transform(training_set)
```

```
#60 timesteps and 1 output
X_train = []
y_train = []
for i in range(60, len(training_set_scaled)):
    X_train.append(training_set_scaled[i-60: i, 0])
    y_train.append(training_set_scaled[i, 0])

X_train, y_train = np.array(X_train), np.array(y_train)
```

```
X_train.shape
```

```
(1449, 60)
```

```
y_train.shape
```

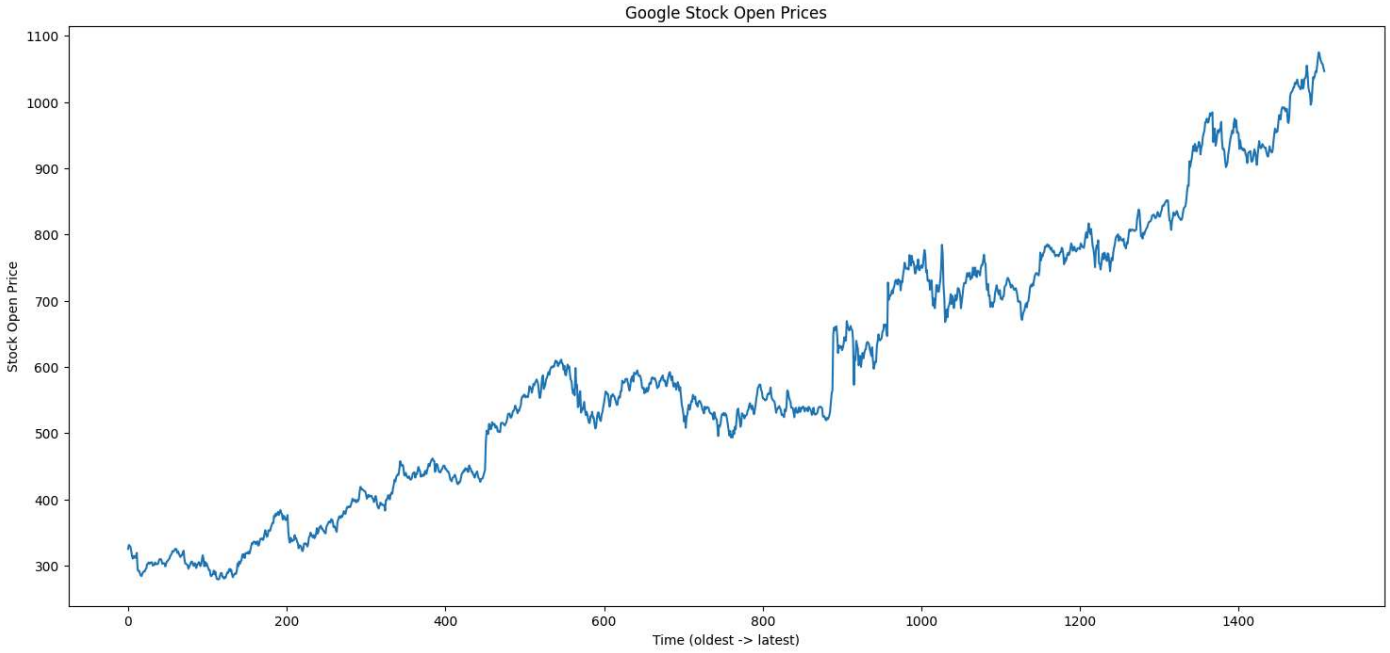
```
(1449,)
```

```
X_train = np.reshape(X_train, newshape = (X_train.shape[0], X_train.shape[1], 1))
```

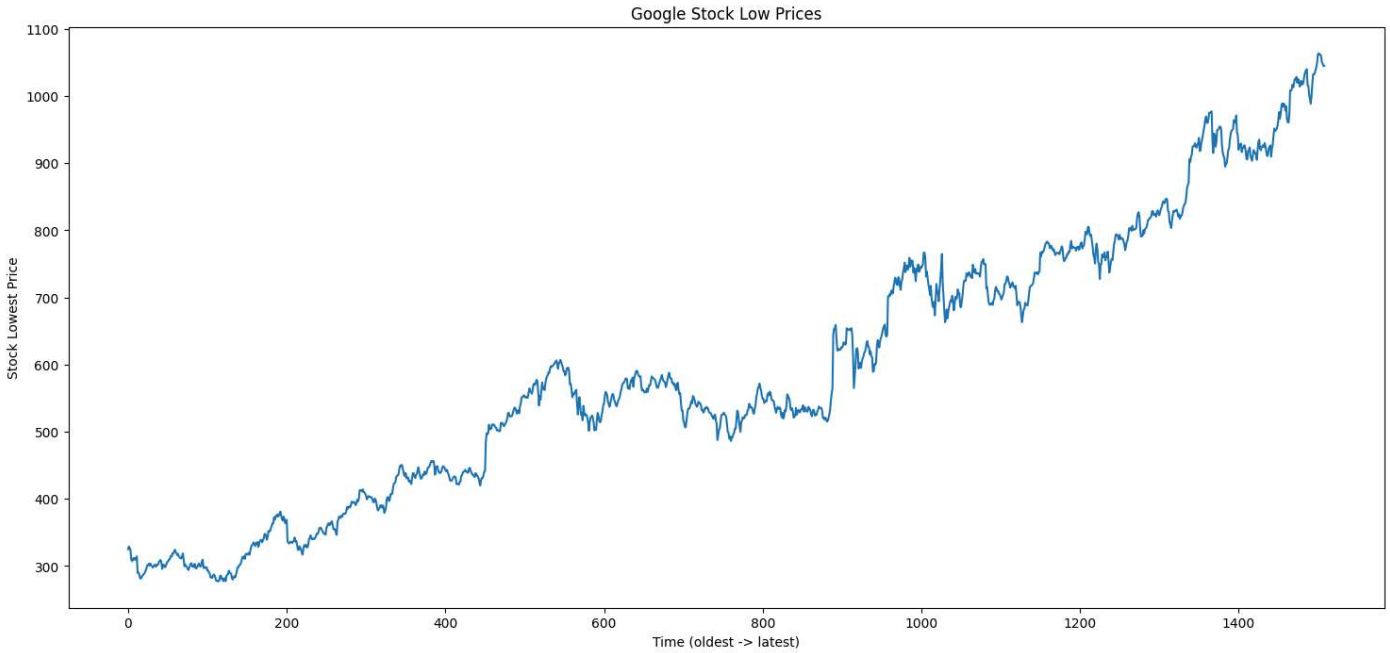
```
X_train.shape
```

```
(1449, 60, 1)
```

```
plt.figure(figsize=(18, 8))
plt.plot(dataset_train['Open'])
plt.title("Google Stock Open Prices")
plt.xlabel("Time (oldest -> latest)")
plt.ylabel("Stock Open Price")
plt.show()
```




```
plt.figure(figsize=(18, 8))
plt.plot(dataset_train['Low'])
plt.title("Google Stock Low Prices")
plt.xlabel("Time (oldest -> latest)")
plt.ylabel("Stock Lowest Price")
plt.show()
```




```
regressor = Sequential()
#add 1st lstm layer
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(rate = 0.2))

##add 2nd lstm layer: 50 neurons
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(rate = 0.2))

##add 3rd lstm layer
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(rate = 0.2))

##add 4th lstm layer
regressor.add(LSTM(units = 50, return_sequences = False))
regressor.add(Dropout(rate = 0.2))

##add output layer
regressor.add(Dense(units = 1))


regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')


regressor.fit(x = X_train, y = y_train, batch_size = 32, epochs = 100)
```

```
46/46 [==============================] - 1s 15ms/step - loss: 0.0010
Epoch 81/100
46/46 [==============================] - 1s 15ms/step - loss: 8.7082e-04
Epoch 82/100
46/46 [==============================] - 1s 22ms/step - loss: 9.4907e-04
Epoch 83/100
46/46 [==============================] - 1s 23ms/step - loss: 0.0011
Epoch 84/100
46/46 [==============================] - 1s 22ms/step - loss: 9.4199e-04
Epoch 85/100
46/46 [==============================] - 1s 16ms/step - loss: 0.0010
Epoch 86/100
46/46 [==============================] - 1s 16ms/step - loss: 9.0285e-04
Epoch 87/100
46/46 [==============================] - 1s 15ms/step - loss: 0.0011
Epoch 88/100
46/46 [==============================] - 1s 15ms/step - loss: 9.3580e-04
Epoch 89/100
46/46 [==============================] - 1s 19ms/step - loss: 8.9859e-04
Epoch 90/100
46/46 [==============================] - 1s 20ms/step - loss: 9.5481e-04
Epoch 91/100
46/46 [==============================] - 1s 21ms/step - loss: 0.0010
Epoch 92/100
46/46 [==============================] - 1s 21ms/step - loss: 9.6291e-04
Epoch 93/100
46/46 [==============================] - 1s 15ms/step - loss: 9.2810e-04
Epoch 94/100
46/46 [==============================] - 1s 14ms/step - loss: 9.5693e-04
Epoch 95/100
46/46 [==============================] - 1s 15ms/step - loss: 9.2776e-04
Epoch 96/100
46/46 [==============================] - 1s 15ms/step - loss: 8.5062e-04
Epoch 97/100
46/46 [==============================] - 1s 15ms/step - loss: 8.2771e-04
Epoch 98/100
46/46 [==============================] - 1s 15ms/step - loss: 8.3726e-04
Epoch 99/100
46/46 [==============================] - 1s 15ms/step - loss: 8.7885e-04
Epoch 100/100
46/46 [==============================] - 1s 14ms/step - loss: 7.9860e-04
<keras.src.callbacks.History at 0x7dc2c0aca170>
```

```
dataset_test = pd.read_csv('Google_Stock_Price_Test.csv')
dataset_test.head()
```

	Date	Open	High	Low	Close	Volume
0	02/01/2018	1048.339966	1066.939941	1045.229980	1065.000000	1237600
1	03/01/2018	1064.310059	1086.290039	1063.209961	1082.479980	1430200
2	04/01/2018	1088.000000	1093.569946	1084.001953	1086.400024	1004600
3	05/01/2018	1094.000000	1104.250000	1092.000000	1102.229980	1279100
4	08/01/2018	1102.229980	1111.270020	1101.619995	1106.939941	1047600

Next steps: Generate code with dataset_test View recommended plots

```
#keras only takes numpy array
real_stock_price = dataset_test.iloc[:, 1: 2].values
real_stock_price.shape
```

```
(125, 1)
```

```
#vertical concat use 0, horizontal uses 1
dataset_total = pd.concat((dataset_train['Open'], dataset_test['Open']),
                          axis = 0)
#use .values to make numpy array
inputs = dataset_total[len(dataset_total) - len(dataset_test) - 60:].values


#reshape data to only have 1 col
inputs = inputs.reshape(-1, 1)

#scale input
inputs = sc.transform(inputs)
len(inputs)
```

```
185
```

```
X_test = []
for i in range(60, len(inputs)):
    X_test.append(inputs[i-60:i, 0])
X_test = np.array(X_test)
#add dimension of indicator
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

```
X_test.shape
```

```
(125, 60, 1)
```

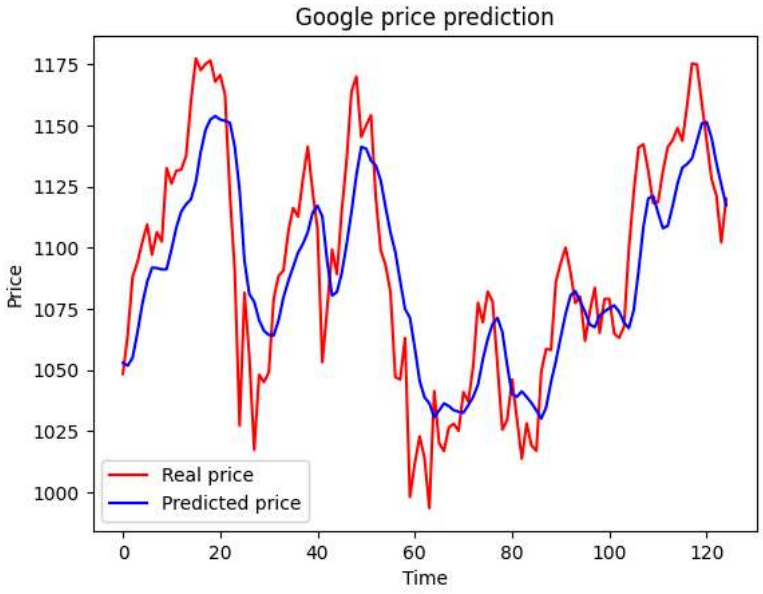
```
predicted_stock_price = regressor.predict(X_test)
```

```
4/4 [==============================] - 1s 9ms/step
```

```
#inverse the scaled value
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

```
#visualize the prediction and real price
plt.plot(real_stock_price, color = 'red', label = 'Real price')
plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted price')

plt.title('Google price prediction')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.show()
```



Start coding or generate with AI.