```
DL 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
```

```
dataset_train = pd.read_csv('/content/sample_data/Google_Stock_Price_Test.csv')
```

```
dataset_train.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_train` View recommended plots

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

```
training_set.shape
```

```
(125, 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
```

```
(65, 60)
```

```
y_train.shape
```

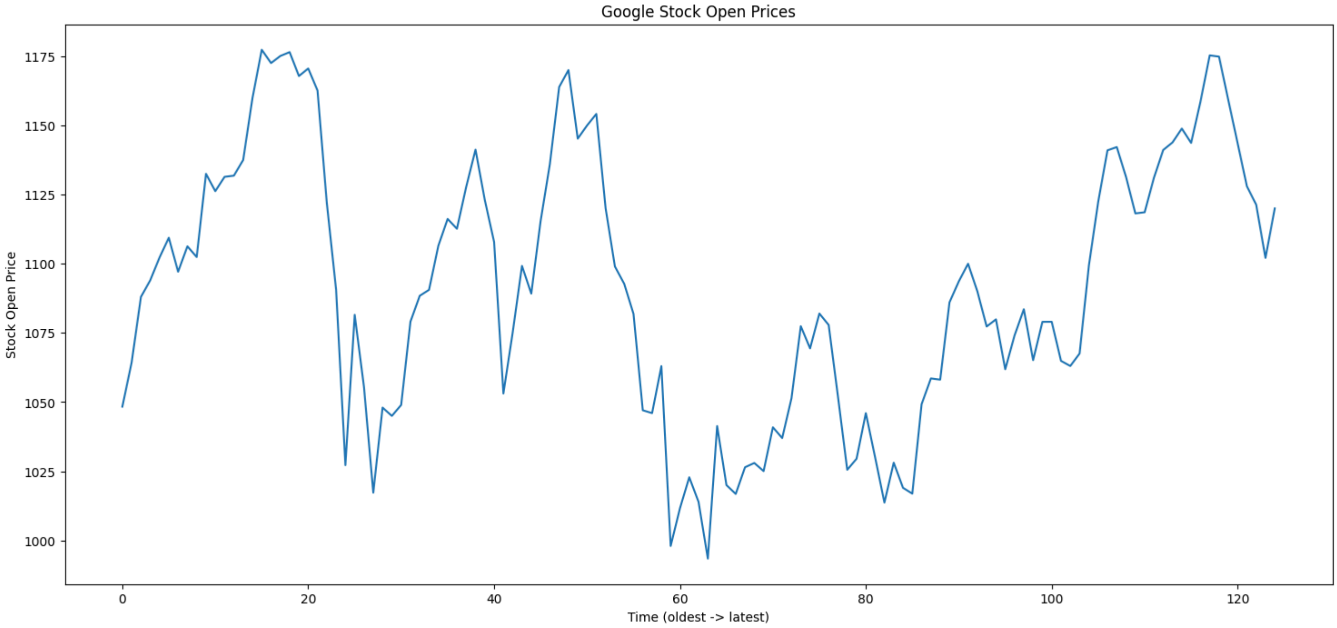
```
(65,)
```

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

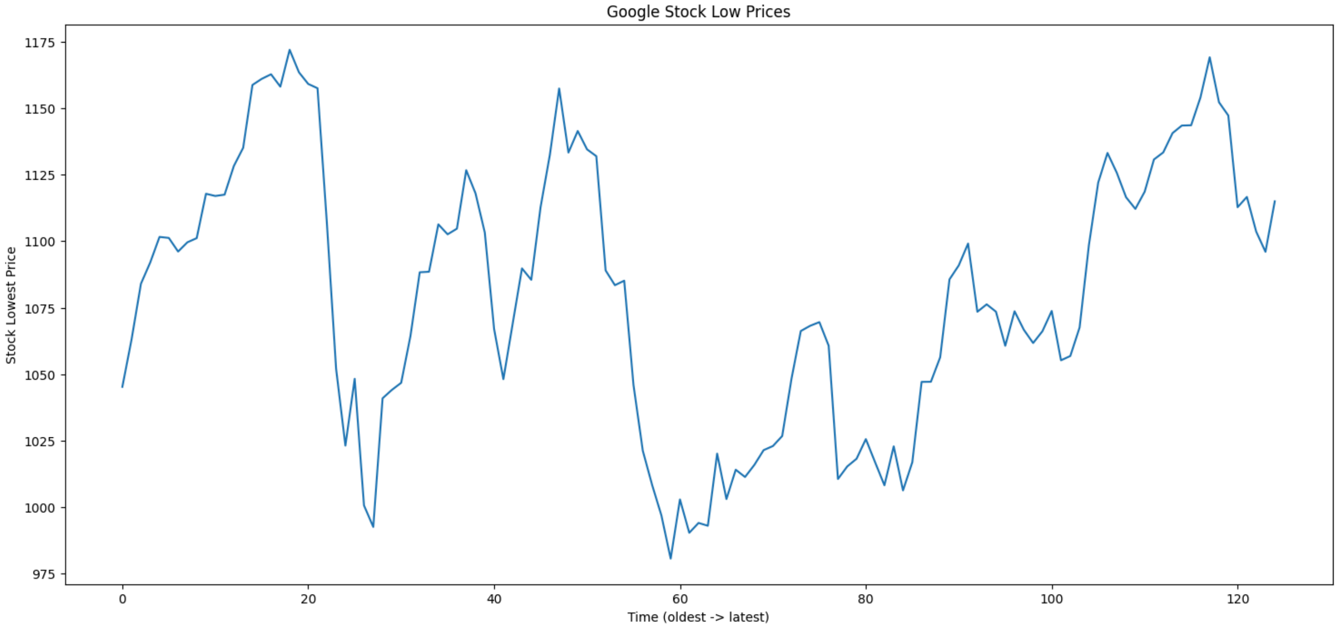
```
X_train.shape
```

```
(65, 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)
```

```
3/3 [==============================] - 0s 93ms/step - loss: 0.0345
Epoch 73/100
3/3 [==============================] - 0s 84ms/step - loss: 0.0367
Epoch 74/100
3/3 [==============================] - 0s 84ms/step - loss: 0.0352
Epoch 75/100
3/3 [==============================] - 0s 86ms/step - loss: 0.0372
Epoch 76/100
3/3 [==============================] - 0s 85ms/step - loss: 0.0419
Epoch 77/100
3/3 [==============================] - 0s 87ms/step - loss: 0.0375
Epoch 78/100
3/3 [==============================] - 0s 83ms/step - loss: 0.0361
Epoch 79/100
3/3 [==============================] - 0s 91ms/step - loss: 0.0411
Epoch 80/100
3/3 [==============================] - 0s 83ms/step - loss: 0.0255
Epoch 81/100
3/3 [==============================] - 0s 85ms/step - loss: 0.0311
Epoch 82/100
3/3 [==============================] - 0s 82ms/step - loss: 0.0346
Epoch 83/100
3/3 [==============================] - 0s 84ms/step - loss: 0.0372
Epoch 84/100
3/3 [==============================] - 0s 84ms/step - loss: 0.0308
Epoch 85/100
3/3 [==============================] - 0s 86ms/step - loss: 0.0293
Epoch 86/100
3/3 [==============================] - 0s 85ms/step - loss: 0.0302
Epoch 87/100
3/3 [==============================] - 0s 84ms/step - loss: 0.0265
Epoch 88/100
3/3 [==============================] - 0s 82ms/step - loss: 0.0273
Epoch 89/100
3/3 [==============================] - 0s 81ms/step - loss: 0.0220
Epoch 90/100
3/3 [==============================] - 0s 91ms/step - loss: 0.0259
Epoch 91/100
3/3 [==============================] - 0s 84ms/step - loss: 0.0275
Epoch 92/100
3/3 [==============================] - 0s 85ms/step - loss: 0.0276
Epoch 93/100
3/3 [==============================] - 0s 119ms/step - loss: 0.0444
Epoch 94/100
3/3 [==============================] - 0s 139ms/step - loss: 0.0250
Epoch 95/100
3/3 [==============================] - 0s 137ms/step - loss: 0.0303
Epoch 96/100
3/3 [==============================] - 0s 138ms/step - loss: 0.0343
Epoch 97/100
3/3 [==============================] - 0s 139ms/step - loss: 0.0349
Epoch 98/100
3/3 [==============================] - 0s 145ms/step - loss: 0.0374
Epoch 99/100
3/3 [==============================] - 0s 92ms/step - loss: 0.0317
Epoch 100/100
3/3 [==============================] - 0s 84ms/step - loss: 0.0260
<keras.src.callbacks.History at 0x7a9f859be770>
```

```
dataset_test = pd.read_csv('/content/sample_data/Google_Stock_Price_Test.csv')
dataset_test.head()
#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)
```

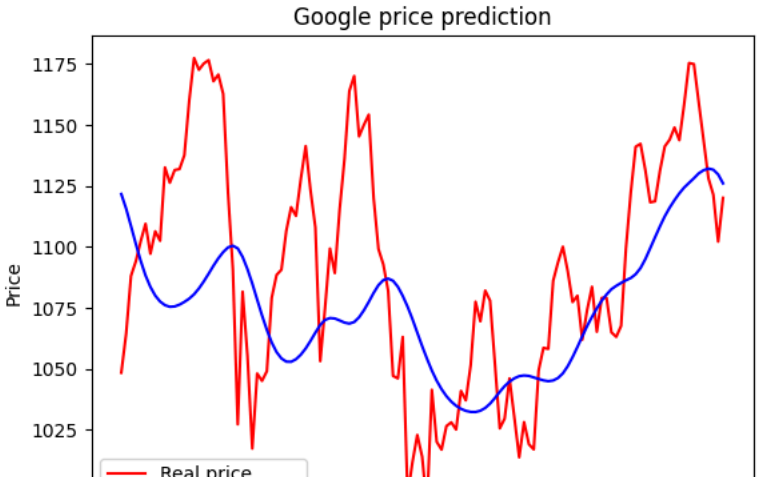
```
predicted_stock_price = regressor.predict(X_test)

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

```
4/4 [==============================] - 2s 30ms/step
```

```
##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()
```


Google price prediction
Price
1175
1150
1125
1100
1075
1050
1025
Real price