```
#Step 1 – Importing required libraries.

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

```
#Step 2 – Reading our training data.

dataset_train = pd.read_csv('/content/Google_Stock_Price_Train.csv')
training_set = dataset_train.iloc[:, 1:2].values

sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)

X_train = []
y_train = []

for i in range(60, 1258):
    X_train.append(training_set_scaled[i-60:i, 0])
    y_train.append(training_set_scaled[i, 0])
```

Here we are reading the input train data. We are just taking the first column here which is the 'open' column. Then we are initializing the MinMaxScaler to scale our data between 0 and 1. After that, we initialized two arrays X_train and y_train. The first entry in the X_train would be an array of the first 60 open stock prices and the first entry in the y_train will be the 61st value of open stock price. It means that we want our model to predict the 61st value of stock price when we provide it with the previous 60 values. In this way, we keep on building our X_train and y_train.

Step 3 – Getting our training data in shape.

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

Here we are converting our lists to arrays. In the 2nd step, we are adding a dummy dimension in the end because Keras models require the data in this format only.

Step 4 – Creating the Stock Price Prediction model.

```
regressor = Sequential()

regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

regressor.add(Dense(units = 1))

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

print(regressor.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 60, 50)            10400

 dropout_4 (Dropout)         (None, 60, 50)            0

 lstm_5 (LSTM)               (None, 60, 50)            20200

 dropout_5 (Dropout)         (None, 60, 50)            0

 lstm_6 (LSTM)               (None, 60, 50)            20200

 dropout_6 (Dropout)         (None, 60, 50)            0

 lstm_7 (LSTM)               (None, 50)                20200

 dropout_7 (Dropout)         (None, 50)                0

 dense_1 (Dense)             (None, 1)                 51

=================================================================
Total params: 71051 (277.54 KB)
Trainable params: 71051 (277.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Our Sequential model consists of 4 LSTM layers with 50 units each. After these LSTM layers, we have a Dense layer. We are using Adam optimizer here and the loss we are using is Mean Squared Error.

Step 5 – Training the Stock Price Prediction model.

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

```
38/38 [==============================] - 4s 94ms/step - loss: 0.0021
Epoch 67/100
38/38 [==============================] - 4s 119ms/step - loss: 0.0018
Epoch 68/100
38/38 [==============================] - 4s 106ms/step - loss: 0.0019
Epoch 69/100
38/38 [==============================] - 4s 96ms/step - loss: 0.0020
Epoch 70/100
38/38 [==============================] - 4s 96ms/step - loss: 0.0016
Epoch 71/100
38/38 [==============================] - 5s 128ms/step - loss: 0.0019
Epoch 72/100
38/38 [==============================] - 4s 95ms/step - loss: 0.0020
Epoch 73/100
38/38 [==============================] - 4s 94ms/step - loss: 0.0020
Epoch 74/100
38/38 [==============================] - 5s 129ms/step - loss: 0.0018
Epoch 75/100
38/38 [==============================] - 4s 97ms/step - loss: 0.0019
Epoch 76/100
38/38 [==============================] - 4s 95ms/step - loss: 0.0018
Epoch 77/100
38/38 [==============================] - 5s 123ms/step - loss: 0.0017
Epoch 78/100
38/38 [==============================] - 4s 103ms/step - loss: 0.0020
Epoch 79/100
38/38 [==============================] - 4s 95ms/step - loss: 0.0018
Epoch 80/100
38/38 [==============================] - 4s 98ms/step - loss: 0.0016
Epoch 81/100
38/38 [==============================] - 5s 127ms/step - loss: 0.0017
Epoch 82/100
38/38 [==============================] - 4s 96ms/step - loss: 0.0017
Epoch 83/100
38/38 [==============================] - 4s 94ms/step - loss: 0.0015
Epoch 84/100
38/38 [==============================] - 5s 127ms/step - loss: 0.0017
Epoch 85/100
38/38 [==============================] - 4s 95ms/step - loss: 0.0018
Epoch 86/100
38/38 [==============================] - 4s 95ms/step - loss: 0.0017
Epoch 87/100
38/38 [==============================] - 5s 122ms/step - loss: 0.0015
Epoch 88/100
38/38 [==============================] - 4s 97ms/step - loss: 0.0017
Epoch 89/100
38/38 [==============================] - 4s 96ms/step - loss: 0.0016
Epoch 90/100
38/38 [==============================] - 4s 108ms/step - loss: 0.0016
Epoch 91/100
38/38 [==============================] - 4s 116ms/step - loss: 0.0016
Epoch 92/100
38/38 [==============================] - 4s 95ms/step - loss: 0.0016
Epoch 93/100
38/38 [==============================] - 4s 96ms/step - loss: 0.0015
```

Training our model for 100 epochs. The batch size which we have taken is 32. You can also experiment with these values.

Step 6 – Reading the test data.

```
dataset_test = pd.read_csv('/content/Google_Stock_Price_Test.csv')
real_stock_price = dataset_test.iloc[:, 1:2].values
```

Now let's read the test data to perform our predictions.

Step 7 -Getting the Stock Price Predictions on test data.

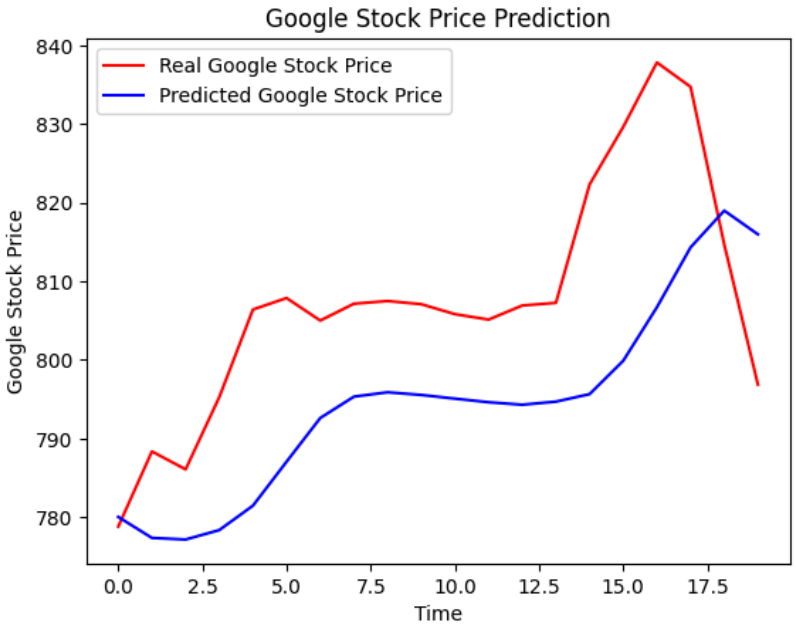
```
# Getting the predicted stock price of 2017
dataset_total = pd.concat((dataset_train['Open'], dataset_test['Open']), axis = 0)
inputs = dataset_total[len(dataset_total) - len(dataset_test) - 60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []
for i in range(60, 80):
    X_test.append(inputs[i-60:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

```
1/1 [==============================] - 1s 1s/step
```

Concatenating the test 'Open' column with the train 'Open' column row-wise. The step we did above was just to take the last 60 values from the train data and also add that to the test data. Then we are reshaping it to have just one column and as many rows. Scaling it using MinMaxScaler. Then we are creating the test data as we did for the train data. And finally, we are making predictions.

NOTE – What we are doing above is we are first taking the last 60 open values from the train and making predictions from it for the 61st value. Then what we will do is we will drop the 0th open value and now our input array will be 1st open value to the 61st open value (60 values) and we will predict the 62nd open value and like this, we will keep on predicting the next value and take that for predicting next values.

```
# Visualising the results
plt.plot(real_stock_price, color = 'red', label = 'Real Google Stock Price')
plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted Google Stock Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Google Stock Price')
plt.legend()
plt.show()
```



Here we are simply plotting our predictions in the blue curve. The red curve represents its true value, which means what it should be exactly. We can see that our model is not that perfect but still it is capable of catching the spikes (where the curve starts to go up).