```
# **DSC 680 -PROJECT 3 - Stock Price Prediction**
```

DSC 680 -PROJECT 3 - Stock Price Prediction Avinash Alapati

The Goal of the project is to predict Stock Market Prices using RNN(LSTM) on Google Stock Price Data Set

LSTM Model along Sequentical are used to Predict Stock prices. The Model results visualization confirmed the Model is accurate.

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

```
/opt/anaconda3/lib/python3.8/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this
version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"
2024-03-03 17:52:00.960826: I
tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow
binary is optimized to use available CPU instructions in performance-
critical operations.
To enable the following instructions: AVX2 FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
```

```
train = pd.read_csv('Google_Stock_Price_Train.csv')
```

```
train.head()
```

```
       Date    Open    High     Low   Close      Volume
0  1/3/2012  325.25  332.83  324.97  663.59   7,380,500
1  1/4/2012  331.27  333.87  329.08  666.45   5,749,400
2  1/5/2012  329.83  330.75  326.89  657.21   6,590,300
3  1/6/2012  328.34  328.77  323.68  648.24   5,405,900
4  1/9/2012  322.04  322.29  309.46  620.76  11,688,800
```

```
training_set=train.iloc[:,1:2].values
```

```
training_set
```

```
array([[325.25],
       [331.27],
       [329.83],
       ...,
       [793.7 ],
```

```
       [783.33],
       [782.75]])

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

training_set_scaled.shape

(1258, 1)

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

#reshape it to (batche_size(#size of
inputs),timesteps,input_dimension)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

X_train.shape

(1198, 60, 1)

regressor=Sequential()

regressor.add(LSTM(units=100,return_sequences=True,input_shape=(X_trai
n.shape[1], 1)))
regressor.add(Dropout(0.4))

regressor.add(LSTM(units=100,return_sequences=True))
regressor.add(Dropout(0.6))

regressor.add(LSTM(units=100,return_sequences=True))
regressor.add(Dropout(0.6))

regressor.add(LSTM(units=100))
regressor.add(Dropout(0.4))

regressor.add(Dense(units=1))

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

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

Epoch 1/100
38/38 [==============================] - 9s 90ms/step - loss: 0.0418
Epoch 2/100
38/38 [==============================] - 3s 90ms/step - loss: 0.0095
Epoch 3/100
38/38 [==============================] - 3s 90ms/step - loss: 0.0071
Epoch 4/100
```

```
38/38 [==============================] - 4s 92ms/step - loss: 0.0065
Epoch 5/100
38/38 [==============================] - 3s 92ms/step - loss: 0.0063
Epoch 6/100
38/38 [==============================] - 4s 92ms/step - loss: 0.0059
Epoch 7/100
38/38 [==============================] - 4s 95ms/step - loss: 0.0063
Epoch 8/100
38/38 [==============================] - 4s 98ms/step - loss: 0.0062
Epoch 9/100
38/38 [==============================] - 4s 107ms/step - loss: 0.0060
Epoch 10/100
38/38 [==============================] - 4s 109ms/step - loss: 0.0052
Epoch 11/100
38/38 [==============================] - 4s 105ms/step - loss: 0.0057
Epoch 12/100
38/38 [==============================] - 4s 106ms/step - loss: 0.0050
Epoch 13/100
38/38 [==============================] - 4s 112ms/step - loss: 0.0048
Epoch 14/100
38/38 [==============================] - 4s 105ms/step - loss: 0.0051
Epoch 15/100
38/38 [==============================] - 4s 113ms/step - loss: 0.0047
Epoch 16/100
38/38 [==============================] - 4s 103ms/step - loss: 0.0050
Epoch 17/100
38/38 [==============================] - 4s 106ms/step - loss: 0.0046
Epoch 18/100
38/38 [==============================] - 4s 105ms/step - loss: 0.0042
Epoch 19/100
38/38 [==============================] - 4s 101ms/step - loss: 0.0045
Epoch 20/100
38/38 [==============================] - 4s 113ms/step - loss: 0.0042
Epoch 21/100
38/38 [==============================] - 4s 103ms/step - loss: 0.0046
Epoch 22/100
38/38 [==============================] - 4s 103ms/step - loss: 0.0043
Epoch 23/100
38/38 [==============================] - 4s 102ms/step - loss: 0.0041
Epoch 24/100
38/38 [==============================] - 4s 99ms/step - loss: 0.0039
Epoch 25/100
38/38 [==============================] - 4s 98ms/step - loss: 0.0041
Epoch 26/100
38/38 [==============================] - 4s 100ms/step - loss: 0.0044
Epoch 27/100
38/38 [==============================] - 4s 99ms/step - loss: 0.0036
Epoch 28/100
38/38 [==============================] - 4s 110ms/step - loss: 0.0039
```

```
Epoch 29/100
38/38 [==============================] - 5s 119ms/step - loss: 0.0035
Epoch 30/100
38/38 [==============================] - 4s 105ms/step - loss: 0.0035
Epoch 31/100
38/38 [==============================] - 4s 106ms/step - loss: 0.0035
Epoch 32/100
38/38 [==============================] - 4s 111ms/step - loss: 0.0034
Epoch 33/100
38/38 [==============================] - 4s 112ms/step - loss: 0.0036
Epoch 34/100
38/38 [==============================] - 4s 102ms/step - loss: 0.0042
Epoch 35/100
38/38 [==============================] - 4s 102ms/step - loss: 0.0034
Epoch 36/100
38/38 [==============================] - 4s 108ms/step - loss: 0.0034
Epoch 37/100
38/38 [==============================] - 5s 123ms/step - loss: 0.0034
Epoch 38/100
38/38 [==============================] - 5s 136ms/step - loss: 0.0034
Epoch 39/100
38/38 [==============================] - 5s 138ms/step - loss: 0.0036
Epoch 40/100
38/38 [==============================] - 6s 151ms/step - loss: 0.0033
Epoch 41/100
38/38 [==============================] - 5s 140ms/step - loss: 0.0031
Epoch 42/100
38/38 [==============================] - 5s 130ms/step - loss: 0.0030
Epoch 43/100
38/38 [==============================] - 5s 119ms/step - loss: 0.0031
Epoch 44/100
38/38 [==============================] - 4s 113ms/step - loss: 0.0032
Epoch 45/100
38/38 [==============================] - 4s 112ms/step - loss: 0.0028
Epoch 46/100
38/38 [==============================] - 4s 109ms/step - loss: 0.0030
Epoch 47/100
38/38 [==============================] - 4s 106ms/step - loss: 0.0025
Epoch 48/100
38/38 [==============================] - 4s 105ms/step - loss: 0.0038
Epoch 49/100
38/38 [==============================] - 4s 112ms/step - loss: 0.0028
Epoch 50/100
38/38 [==============================] - 5s 119ms/step - loss: 0.0031
Epoch 51/100
38/38 [==============================] - 5s 123ms/step - loss: 0.0029
Epoch 52/100
38/38 [==============================] - 5s 127ms/step - loss: 0.0027
Epoch 53/100
```

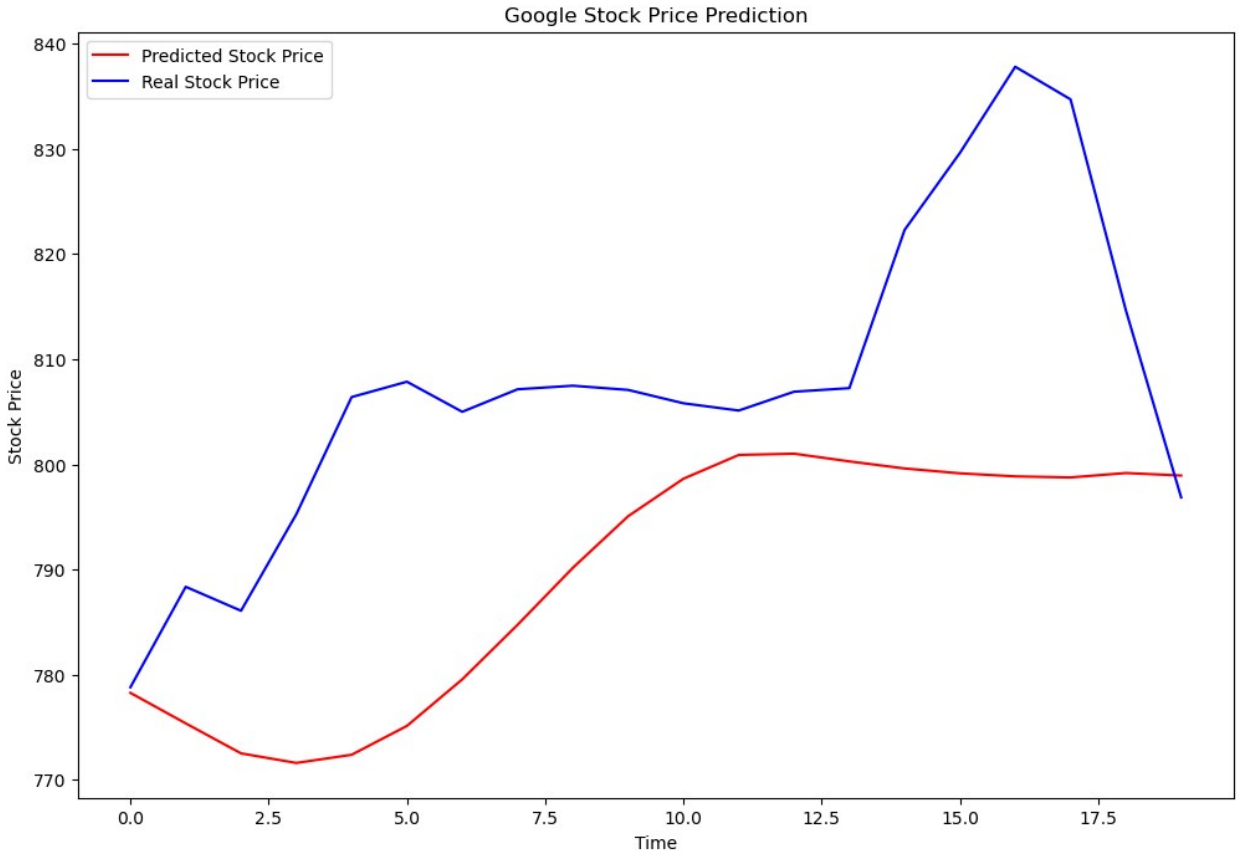
```
38/38 [==============================] - 5s 138ms/step - loss: 0.0026
Epoch 54/100
38/38 [==============================] - 5s 136ms/step - loss: 0.0028
Epoch 55/100
38/38 [==============================] - 5s 138ms/step - loss: 0.0028
Epoch 56/100
38/38 [==============================] - 5s 125ms/step - loss: 0.0031
Epoch 57/100
38/38 [==============================] - 4s 117ms/step - loss: 0.0032
Epoch 58/100
38/38 [==============================] - 4s 107ms/step - loss: 0.0025
Epoch 59/100
38/38 [==============================] - 4s 117ms/step - loss: 0.0029
Epoch 60/100
38/38 [==============================] - 4s 105ms/step - loss: 0.0030
Epoch 61/100
38/38 [==============================] - 4s 109ms/step - loss: 0.0024
Epoch 62/100
38/38 [==============================] - 4s 107ms/step - loss: 0.0027
Epoch 63/100
38/38 [==============================] - 4s 111ms/step - loss: 0.0026
Epoch 64/100
38/38 [==============================] - 4s 103ms/step - loss: 0.0028
Epoch 65/100
38/38 [==============================] - 4s 114ms/step - loss: 0.0026
Epoch 66/100
38/38 [==============================] - 4s 110ms/step - loss: 0.0027
Epoch 67/100
38/38 [==============================] - 4s 118ms/step - loss: 0.0024
Epoch 68/100
38/38 [==============================] - 5s 142ms/step - loss: 0.0028
Epoch 69/100
38/38 [==============================] - 6s 153ms/step - loss: 0.0026
Epoch 70/100
38/38 [==============================] - 6s 158ms/step - loss: 0.0026
Epoch 71/100
38/38 [==============================] - 6s 145ms/step - loss: 0.0024
Epoch 72/100
38/38 [==============================] - 6s 159ms/step - loss: 0.0024
Epoch 73/100
38/38 [==============================] - 6s 153ms/step - loss: 0.0024
Epoch 74/100
38/38 [==============================] - 5s 143ms/step - loss: 0.0025
Epoch 75/100
38/38 [==============================] - 4s 117ms/step - loss: 0.0026
Epoch 76/100
38/38 [==============================] - 4s 108ms/step - loss: 0.0024
Epoch 77/100
38/38 [==============================] - 4s 103ms/step - loss: 0.0022
```

```
Epoch 78/100
38/38 [==============================] - 4s 109ms/step - loss: 0.0024
Epoch 79/100
38/38 [==============================] - 4s 102ms/step - loss: 0.0021
Epoch 80/100
38/38 [==============================] - 4s 107ms/step - loss: 0.0026
Epoch 81/100
38/38 [==============================] - 4s 102ms/step - loss: 0.0030
Epoch 82/100
38/38 [==============================] - 4s 112ms/step - loss: 0.0020
Epoch 83/100
38/38 [==============================] - 4s 118ms/step - loss: 0.0022
Epoch 84/100
38/38 [==============================] - 5s 129ms/step - loss: 0.0023
Epoch 85/100
38/38 [==============================] - 5s 133ms/step - loss: 0.0023
Epoch 86/100
38/38 [==============================] - 5s 136ms/step - loss: 0.0025
Epoch 87/100
38/38 [==============================] - 5s 137ms/step - loss: 0.0022
Epoch 88/100
38/38 [==============================] - 6s 154ms/step - loss: 0.0019
Epoch 89/100
38/38 [==============================] - 5s 126ms/step - loss: 0.0021
Epoch 90/100
38/38 [==============================] - 5s 119ms/step - loss: 0.0021
Epoch 91/100
38/38 [==============================] - 4s 116ms/step - loss: 0.0020
Epoch 92/100
38/38 [==============================] - 4s 108ms/step - loss: 0.0024
Epoch 93/100
38/38 [==============================] - 4s 101ms/step - loss: 0.0021
Epoch 94/100
38/38 [==============================] - 4s 102ms/step - loss: 0.0022
Epoch 95/100
38/38 [==============================] - 4s 112ms/step - loss: 0.0020
Epoch 96/100
38/38 [==============================] - 5s 130ms/step - loss: 0.0019
Epoch 97/100
38/38 [==============================] - 4s 116ms/step - loss: 0.0019
Epoch 98/100
38/38 [==============================] - 5s 124ms/step - loss: 0.0022
Epoch 99/100
38/38 [==============================] - 5s 129ms/step - loss: 0.0026

Epoch 100/100
38/38 [==============================] - 5s 135ms/step - loss: 0.0021

<keras.src.callbacks.History at 0x7f8872783d00>
```

```
test = pd.read_csv('Google_Stock_Price_Test.csv')
real_stock_price = test.iloc[:, 1:2].values
total_data=pd.concat([train['Open'],test['Open']],axis=0)

inputs=total_data[len(train)-len(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.shape
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 997ms/step
```

```
#Visualization
plt.rcParams["figure.figsize"] = (12,8)
plt.plot(predicted_stock_price,color='red',label='Predicted Stock
Price')
plt.plot(real_stock_price,color='blue',label='Real Stock Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```




```
Conclusion

Classic RNNs have short memory, and were neither popular nor powerful
for this exact reason. But a recent major improvement in Recurrent
Neural Networks gave rise to the popularity of LSTMs (Long Short Term
Memory RNNs) which has completely changed the playing field.

Predicted stock price for Google's trending Stock data

    Successfully predicted Google stock price by using last 5 year's
data of Google stock price.
    Use of RNN(LSTM) was implemented alongside with Keras framework
producing some good results.

Technology Used: Deep Learning , RNN , Machine Learning , LSTM
```