Deep Learning

Roll No. :- 19121028

Assignment No: 5

Title of the Assignment:

> Use the Google stock prices dataset and design a time series analysis and prediction system using RNN

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import datetime
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from sklearn.metrics import r2_score
# /content/drive/MyDrive/ML Datasets/Google_Stock_Price_Test.csv
```

```
data = pd.read_csv('/content/drive/MyDrive/ML_dataset/Copy of Google_Stock_Price_Train.csv',thousands=',')
print(data.head(10))
data.shape
```

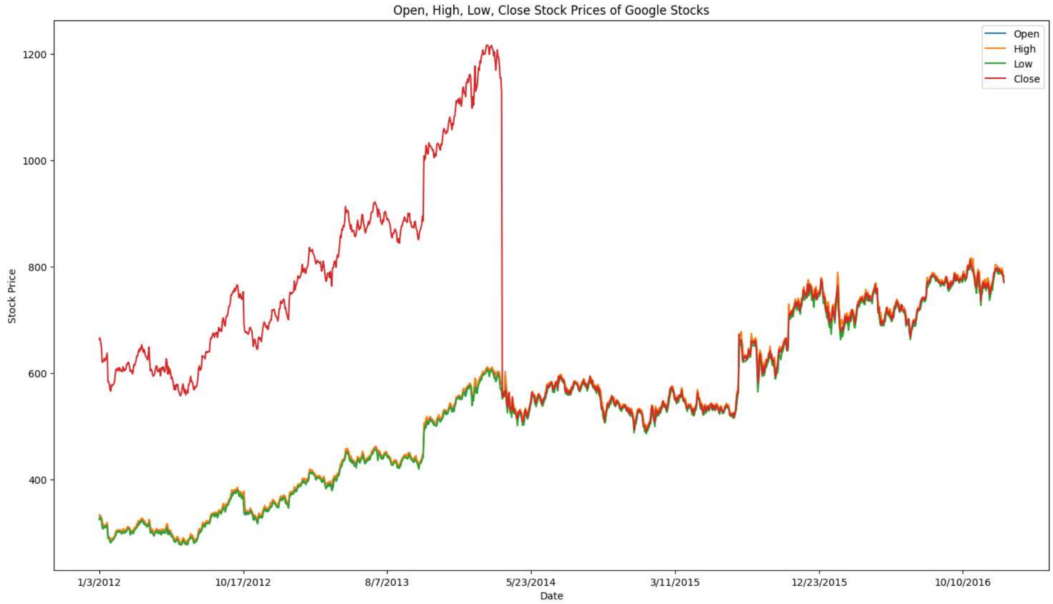
```
        Date    Open    High     Low   Close    Volume
0   1/3/2012  325.25  332.83  324.97  663.59   7380500
1   1/4/2012  331.27  333.87  329.08  666.45   5749400
2   1/5/2012  329.83  330.75  326.89  657.21   6590300
3   1/6/2012  328.34  328.77  323.68  648.24   5405900
4   1/9/2012  322.04  322.29  309.46  620.76  11688800
5  1/10/2012  313.70  315.72  307.30  621.43   8824000
6  1/11/2012  310.59  313.52  309.40  624.25   4817800
7  1/12/2012  314.43  315.26  312.08  627.92   3764400
8  1/13/2012  311.96  312.30  309.37  623.28   4631800
9  1/17/2012  314.81  314.81  311.67  626.86   3832800
(1258, 6)
```

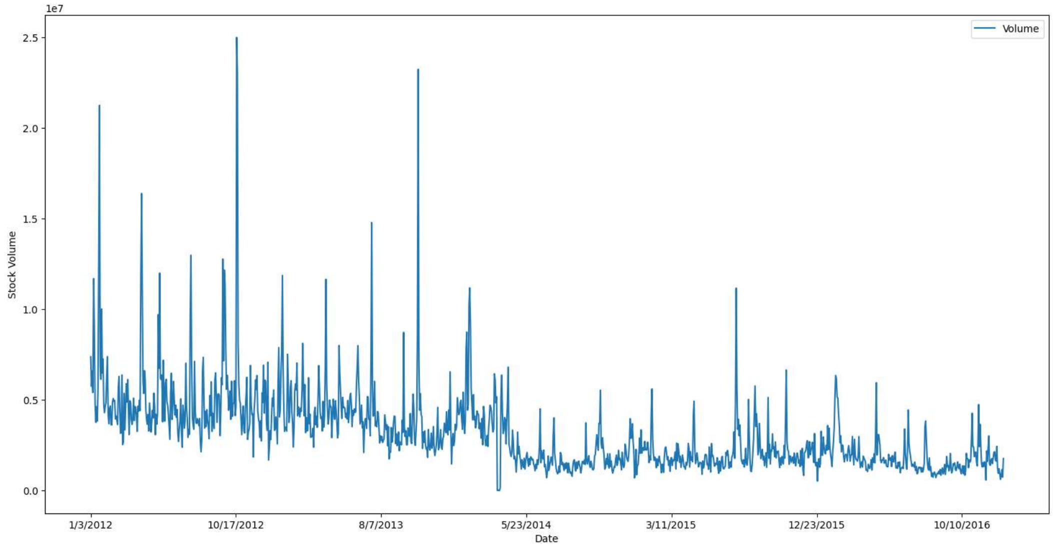
```
from google.colab import drive
drive.mount('/content/drive')
```

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

```
ax1 = data.plot(x="Date", y=["Open", "High", "Low", "Close"],  figsize=(18,10),title='Open, High, Low, Close Stock Prices of Google Stock
ax1.set_ylabel("Stock Price")

ax2 = data.plot(x="Date", y=["Volume"],  figsize=(18,9))
ax2.set_ylabel("Stock Volume")
```

```
Text(0, 0.5, 'Stock Volume')
```

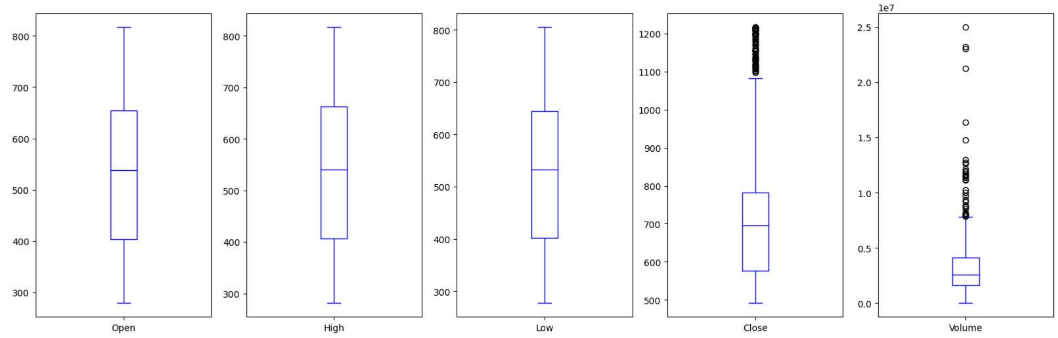





```
# Getting a summary of missing values for each field/attribute
print(data.isnull().sum())
```

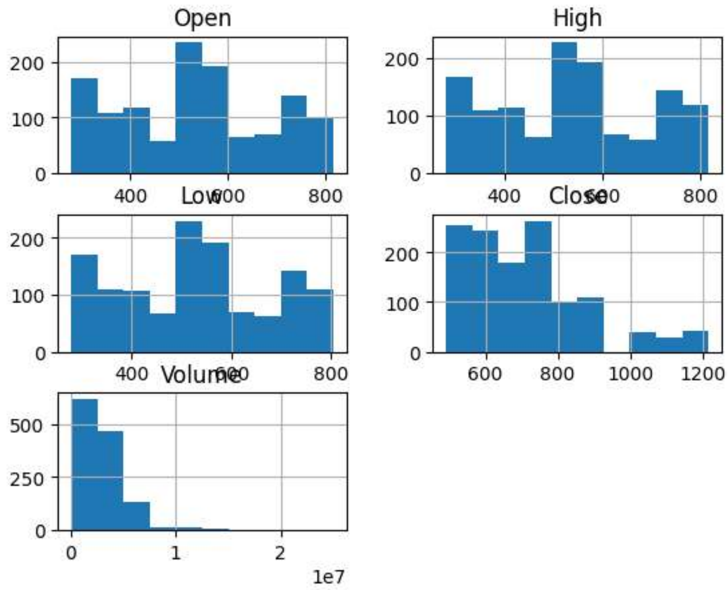
```
Date      0
Open      0
High      0
Low       0
Close     0
Volume    0
dtype: int64
```

```
data[['Open','High','Low','Close','Volume']].plot(kind= 'box' ,layout=(1,5),subplots=True, sharex=False, sharey=False, figsize=(20,6),col
plt.show()
```




```
data.hist()
```

```
array([[<Axes: title={'center': 'Open'}>,
        <Axes: title={'center': 'High'}>],
       [<Axes: title={'center': 'Low'}>,
        <Axes: title={'center': 'Close'}>],
       [<Axes: title={'center': 'Volume'}>, <Axes: >]], dtype=object)
```



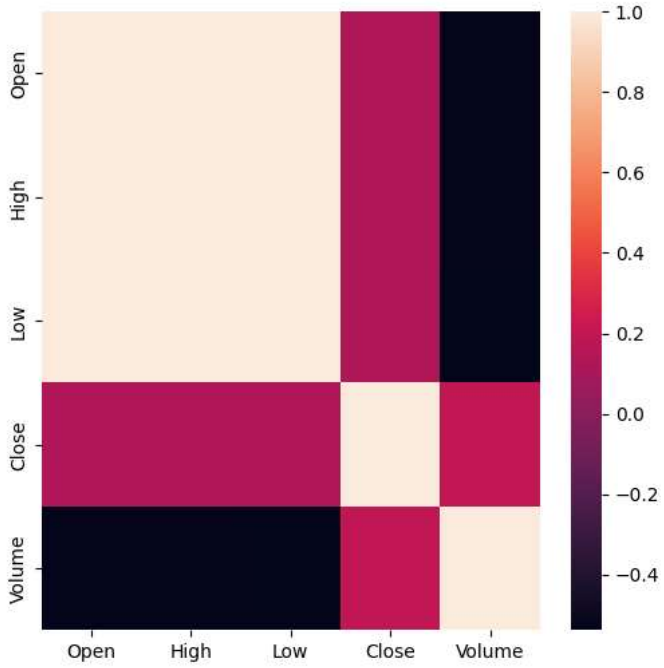

```
scaler = MinMaxScaler()
data_without_date = data[['Open','High','Low','Close','Volume']]
data_scaled = pd.DataFrame(scaler.fit_transform(data_without_date))
```

```
data_scaled.hist()
```

```
array([[<Axes: title={'center': '0'}>, <Axes: title={'center': '1'}>],
       [<Axes: title={'center': '2'}>, <Axes: title={'center': '3'}>],
       [<Axes: title={'center': '4'}>, <Axes: >]], dtype=object)
```

```
plt.figure(figsize=(6,6))
sns.heatmap(data.corr())
```

```
<ipython-input-13-373595b0d5cd>:2: FutureWarning: The default value of numeric_only in DataFrame.corr i
  sns.heatmap(data.corr())
<Axes: >
```




```
data_scaled=data_scaled.drop([0,2,3], axis=1)
data_scaled
```

	1	4
0	0.096401	0.295258
1	0.098344	0.229936
2	0.092517	0.263612
3	0.088819	0.216179
4	0.076718	0.467797
...	...	...
1253	0.955292	0.024650
1254	0.964853	0.031286
1255	0.958074	0.045891
1256	0.942574	0.029491
1257	0.936691	0.070569

1258 rows × 2 columns

```
def split_seq_multivariate(sequence, n_past, n_future):

    '''
    n_past ==> no of past observations
    n_future ==> no of future observations
    '''
    x, y = [], []
    for window_start in range(len(sequence)):
        past_end = window_start + n_past
        future_end = past_end + n_future
        if future_end > len(sequence):
            break
        # slicing the past and future parts of the window
        past   = sequence[window_start:past_end, :]
        future = sequence[past_end:future_end, -1]
        x.append(past)
```

```
        y.append(future)

    return np.array(x), np.array(y)
```

```
# specify the window size
n_steps = 60

data_scaled = data_scaled.to_numpy()
data_scaled.shape
```

```
(1258, 2)
```

```
# split into samples
X, y = split_seq_multivariate(data_scaled, n_steps,1)


# X is in the shape of [samples, timesteps, features]
print(X.shape)
print(y.shape)

# make y to the shape of [samples]
y=y[:,0]
y.shape
```

```
(1198, 60, 2)
(1198, 1)
(1198,)
```

```
# split into train/test
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=50)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(958, 60, 2) (240, 60, 2) (958,) (240,)
```

```
# further dividing the training set into train and validation data
X_train, X_val, y_train, y_val = train_test_split(X_train,y_train,test_size=0.2,random_state=30)

print(X_train.shape, X_val.shape, y_train.shape, y_val.shape)
```

```
(766, 60, 2) (192, 60, 2) (766,) (192,)
```

```
# define RNN model
model = Sequential()
model.add(LSTM(612, input_shape=(n_steps,2)))
model.add(Dense(50, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(1))
```

```
model.summary()
```

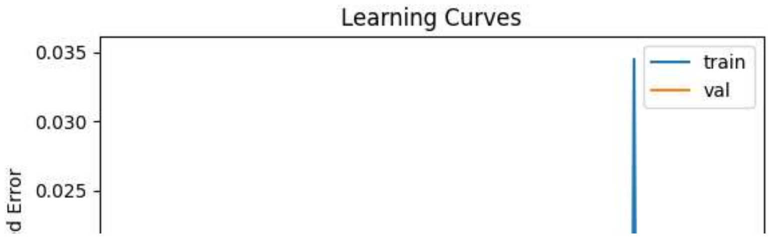
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 612)               1505520

 dense (Dense)               (None, 50)                30650

 dense_1 (Dense)             (None, 50)                2550

 dense_2 (Dense)             (None, 30)                1530

 dense_3 (Dense)             (None, 1)                 31

=================================================================
Total params: 1,540,281
Trainable params: 1,540,281
Non-trainable params: 0
_________________________________________________________________
```

```
# compile the model
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

```
# fit the model
history = model.fit(X_train, y_train, epochs=250, batch_size=32, verbose=2, validation_data=(X_val, y_val))  # has used
```

```
Epoch 222/250
24/24 - 0s - loss: 0.0031 - mae: 0.0323 - val_loss: 0.0030 - val_mae: 0.0342 - 367ms/epoch - 15ms/step
Epoch 223/250
24/24 - 0s - loss: 0.0029 - mae: 0.0305 - val_loss: 0.0047 - val_mae: 0.0399 - 384ms/epoch - 16ms/step
Epoch 224/250
24/24 - 0s - loss: 0.0029 - mae: 0.0314 - val_loss: 0.0036 - val_mae: 0.0367 - 404ms/epoch - 17ms/step
Epoch 225/250
24/24 - 0s - loss: 0.0026 - mae: 0.0293 - val_loss: 0.0044 - val_mae: 0.0379 - 381ms/epoch - 16ms/step
Epoch 226/250
24/24 - 0s - loss: 0.0027 - mae: 0.0303 - val_loss: 0.0033 - val_mae: 0.0359 - 341ms/epoch - 14ms/step
Epoch 227/250
24/24 - 0s - loss: 0.0026 - mae: 0.0289 - val_loss: 0.0036 - val_mae: 0.0380 - 329ms/epoch - 14ms/step
Epoch 228/250
24/24 - 0s - loss: 0.0030 - mae: 0.0318 - val_loss: 0.0032 - val_mae: 0.0371 - 366ms/epoch - 15ms/step
Epoch 229/250
24/24 - 0s - loss: 0.0070 - mae: 0.0454 - val_loss: 0.0048 - val_mae: 0.0528 - 337ms/epoch - 14ms/step
Epoch 230/250
24/24 - 0s - loss: 0.0049 - mae: 0.0384 - val_loss: 0.0027 - val_mae: 0.0324 - 370ms/epoch - 15ms/step
Epoch 231/250
24/24 - 0s - loss: 0.0047 - mae: 0.0364 - val_loss: 0.0024 - val_mae: 0.0338 - 343ms/epoch - 14ms/step
Epoch 232/250
24/24 - 0s - loss: 0.0039 - mae: 0.0349 - val_loss: 0.0028 - val_mae: 0.0338 - 338ms/epoch - 14ms/step
Epoch 233/250
24/24 - 0s - loss: 0.0039 - mae: 0.0332 - val_loss: 0.0027 - val_mae: 0.0337 - 358ms/epoch - 15ms/step
Epoch 234/250
24/24 - 0s - loss: 0.0036 - mae: 0.0324 - val_loss: 0.0030 - val_mae: 0.0349 - 360ms/epoch - 15ms/step
Epoch 235/250
24/24 - 0s - loss: 0.0035 - mae: 0.0322 - val_loss: 0.0032 - val_mae: 0.0350 - 331ms/epoch - 14ms/step
Epoch 236/250
24/24 - 0s - loss: 0.0034 - mae: 0.0323 - val_loss: 0.0032 - val_mae: 0.0351 - 360ms/epoch - 15ms/step
Epoch 237/250
24/24 - 0s - loss: 0.0034 - mae: 0.0321 - val_loss: 0.0031 - val_mae: 0.0345 - 362ms/epoch - 15ms/step
Epoch 238/250
24/24 - 0s - loss: 0.0029 - mae: 0.0304 - val_loss: 0.0036 - val_mae: 0.0363 - 329ms/epoch - 14ms/step
Epoch 239/250
24/24 - 0s - loss: 0.0027 - mae: 0.0308 - val_loss: 0.0040 - val_mae: 0.0374 - 335ms/epoch - 14ms/step
Epoch 240/250
24/24 - 0s - loss: 0.0028 - mae: 0.0302 - val_loss: 0.0039 - val_mae: 0.0371 - 358ms/epoch - 15ms/step
Epoch 241/250
24/24 - 0s - loss: 0.0027 - mae: 0.0299 - val_loss: 0.0038 - val_mae: 0.0370 - 360ms/epoch - 15ms/step
Epoch 242/250
24/24 - 0s - loss: 0.0024 - mae: 0.0287 - val_loss: 0.0048 - val_mae: 0.0406 - 329ms/epoch - 14ms/step
Epoch 243/250
24/24 - 0s - loss: 0.0023 - mae: 0.0286 - val_loss: 0.0037 - val_mae: 0.0360 - 339ms/epoch - 14ms/step
Epoch 244/250
24/24 - 0s - loss: 0.0022 - mae: 0.0291 - val_loss: 0.0043 - val_mae: 0.0380 - 326ms/epoch - 14ms/step
Epoch 245/250
24/24 - 0s - loss: 0.0024 - mae: 0.0291 - val_loss: 0.0050 - val_mae: 0.0384 - 355ms/epoch - 15ms/step
Epoch 246/250
24/24 - 0s - loss: 0.0023 - mae: 0.0286 - val_loss: 0.0042 - val_mae: 0.0367 - 329ms/epoch - 14ms/step
Epoch 247/250
24/24 - 0s - loss: 0.0022 - mae: 0.0299 - val_loss: 0.0058 - val_mae: 0.0402 - 357ms/epoch - 15ms/step
Epoch 248/250
24/24 - 0s - loss: 0.0020 - mae: 0.0279 - val_loss: 0.0052 - val_mae: 0.0393 - 331ms/epoch - 14ms/step
Epoch 249/250
24/24 - 0s - loss: 0.0020 - mae: 0.0277 - val_loss: 0.0053 - val_mae: 0.0418 - 357ms/epoch - 15ms/step
Epoch 250/250
24/24 - 0s - loss: 0.0018 - mae: 0.0273 - val_loss: 0.0058 - val_mae: 0.0416 - 324ms/epoch - 14ms/step
```

```
from matplotlib import pyplot
# plot learning curves
pyplot.title('Learning Curves')
pyplot.xlabel('Epoch')
pyplot.ylabel('Root Mean Squared Error')
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='val')
pyplot.legend()
pyplot.show()
```




```
# evaluate the model
mse, mae = model.evaluate(X_test, y_test, verbose=0)
print('MSE: %.3f, RMSE: %.3f, MAE: %.3f' % (mse, np.sqrt(mse), mae))
```

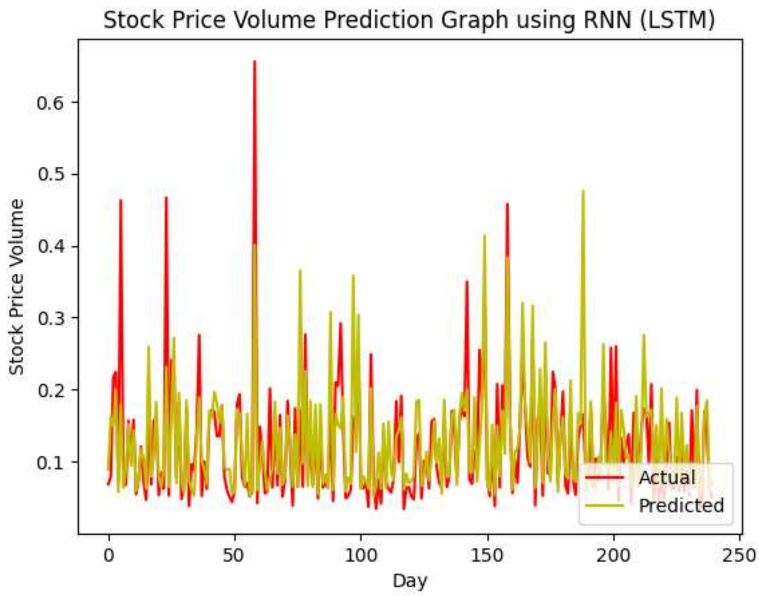
    MSE: 0.003, RMSE: 0.057, MAE: 0.035

```
# predicting y_test values
print(X_test.shape)
predicted_values = model.predict(X_test)
print(predicted_values.shape)
# print(predicted_values)
```

    (240, 60, 2)
    8/8 [==============================] - 1s 6ms/step
    (240, 1)

```
plt.plot(y_test,c = 'r')
plt.plot(predicted_values,c = 'y')
plt.xlabel('Day')
plt.ylabel('Stock Price Volume')
plt.title('Stock Price Volume Prediction Graph using RNN (LSTM)')
plt.legend(['Actual','Predicted'],loc = 'lower right')
plt.figure(figsize=(10,6))
```

    <Figure size 1000x600 with 0 Axes>



    <Figure size 1000x600 with 0 Axes>

```
# evaluating using R squared
R_square = r2_score(y_test, predicted_values)

print(R_square)
```

    0.5211996317839598