CM 4310 – Artificial Neural Networks & Evolutionary Computing

Assignment 02

Name : Lakshita S.U.

Index no : 174085J

This use case has been used to elaborate the techniques of data cleansing, data transformation, feature coding, feature scaling and standardization, dimension reduction with Principle Component Analysis (PCA), and Recurrent Neural Network(RNN) using a Google_Stock_Price_Train Dataset.

First, we need to import following python libraries.

Then, we should load the dataset and get an idea about the dataset using values. And also, we can get idea using their information.

In [1]:
```
#import python libraries
import numpy as np
from numpy import asarray
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import scipy.stats as stats
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
```

Read the data

In [2]:
```
data_set = pd.read_csv('D:Google_Stock_Price.csv',thousands=',')
data_set.head()
```

	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

In [3]:
```
data_set.shape
```

```
(1258, 6)
```

```
In [4]: data_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    1258 non-null   object
 1   Open    1258 non-null   float64
 2   High    1258 non-null   float64
 3   Low     1258 non-null   float64
 4   Close   1258 non-null   float64
 5   Volume  1258 non-null   int64
dtypes: float64(4), int64(1), object(1)
memory usage: 59.1+ KB
```

1. Preprocessing

In this stage, we should identify missing values, remove outliers, avoid null values etc. First checked if there are null values in any column.

Handling missing values

```
In [5]: # check if there are null values
data_set.isnull().any()
```

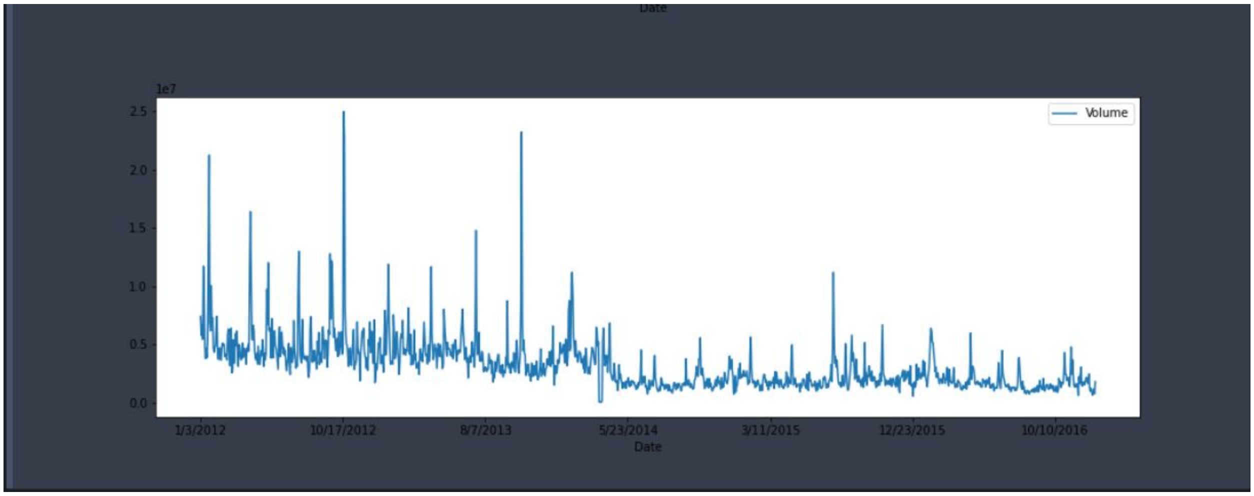
```
Date      False
Open      False
High      False
Low       False
Close     False
Volume    False
dtype: bool
```

There are not any null values of the dataset.

After, I draw two charts. First one is plotted "Open","Close","Low","High" values against date. Second one id “Volume” against date.

```
In [6]: data_set.plot(x="Date", y=["Open","Close","Low","High"], kind="line",figsize=(15,5))
data_set.plot(x="Date", y=["Volume"], kind="line",figsize=(15,5))
```

```
<AxesSubplot:xlabel='Date'>
```



In first graph, "Close" value is greater than "High" value before 2014. But "Close" value should be less than "High" value. Therefore, model cannot train with "Close" values. Then "Close" column should be dropped.

```
In [7]: # remove the 'close' column
        data_set=data_set.drop(['Close'],axis=1)
```

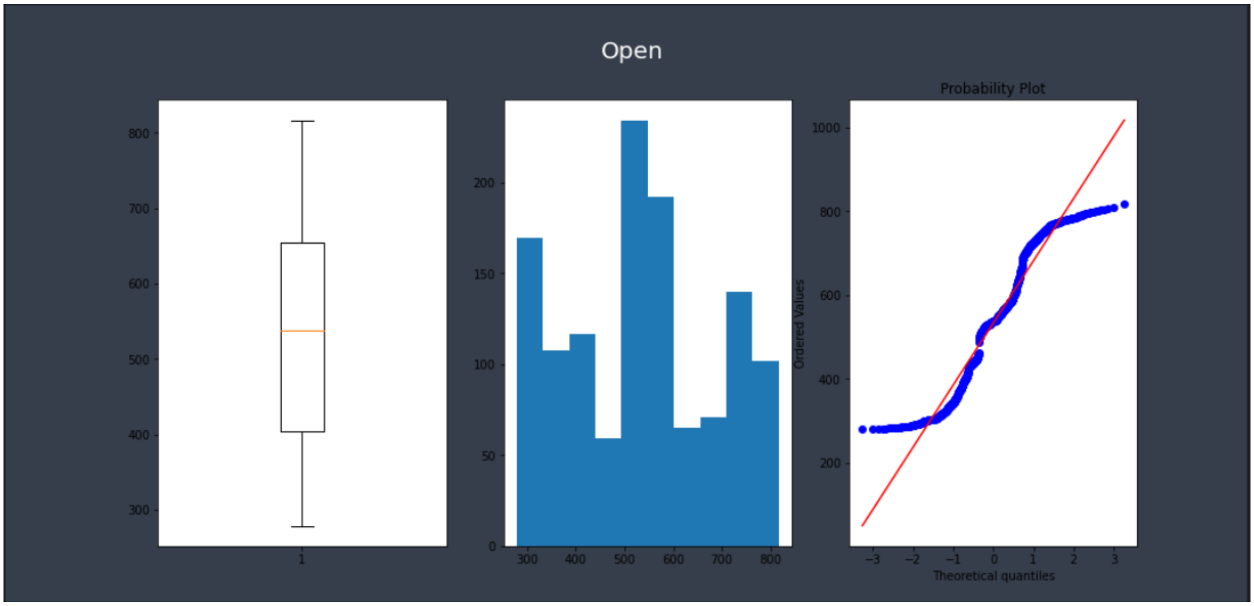
1.1. Handling outliers

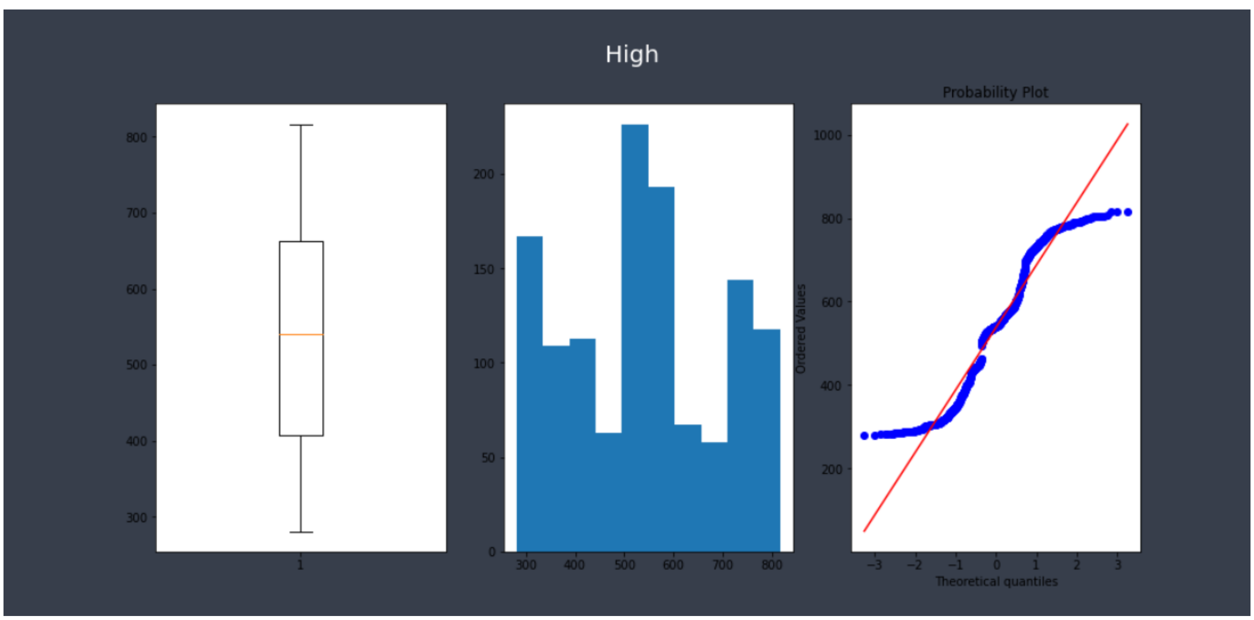
Outliers can be identified by using boxplots. Therefore, each feature can be represented using boxplots.

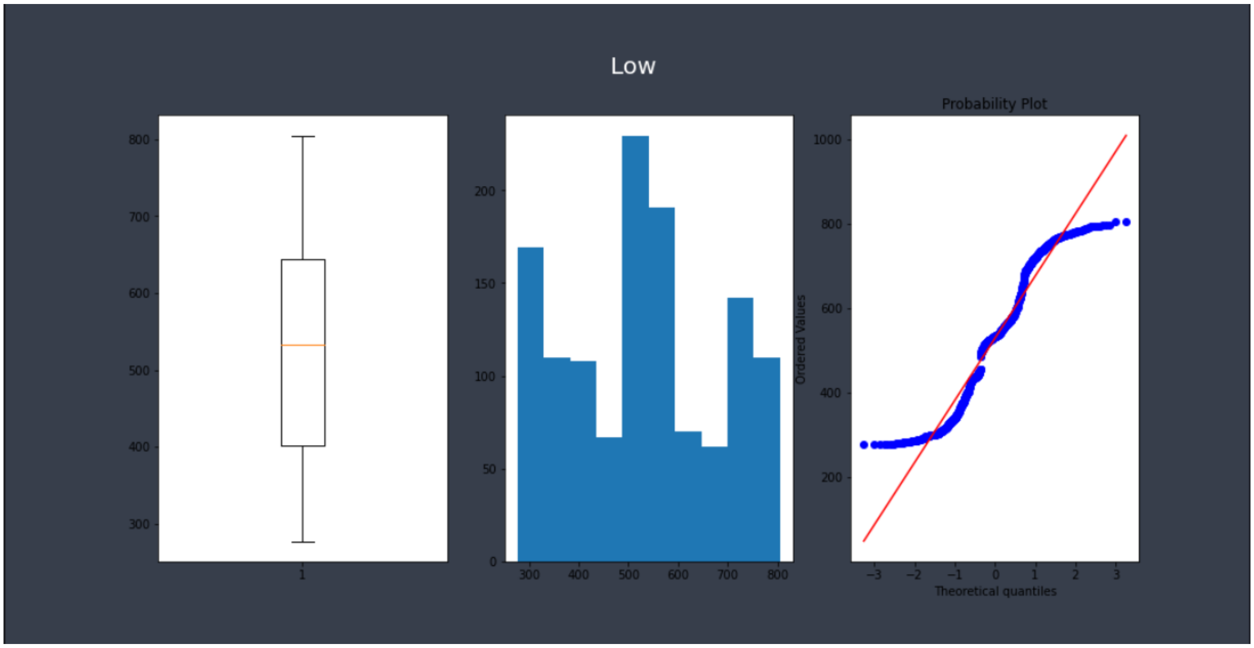
```
In [8]: data_set.boxplot()

<AxesSubplot:>
```

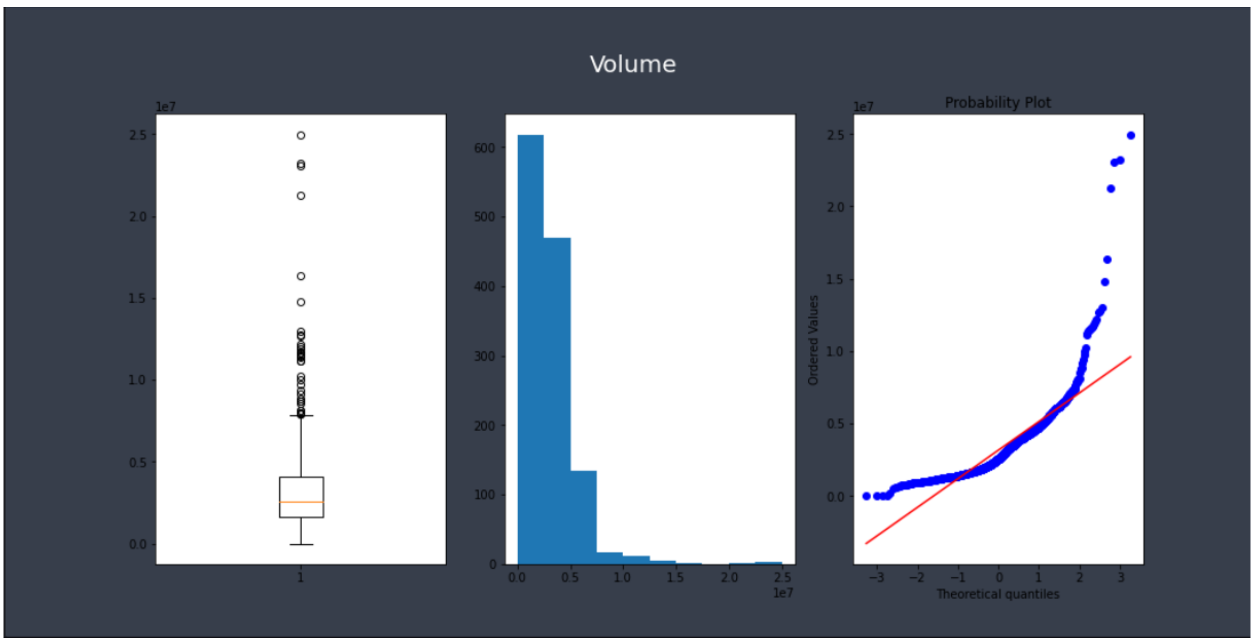
Histograms, Q-Q plots and boxplots are represented all numerical data as follows.

```python
In [9]: cols = data_set.select_dtypes(include=np.number).columns

        for col in cols:
            fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15,7))
            fig.suptitle(col.capitalize(), fontsize=20, color='White')
            axes[0].boxplot(data_set[col])
            axes[1].hist(data_set[col])
            stats.probplot(data_set[col], dist="norm", plot=axes[2])
            plt.show()
```









By using above graphs, we can confirm that there are not any outliers of the dataset.

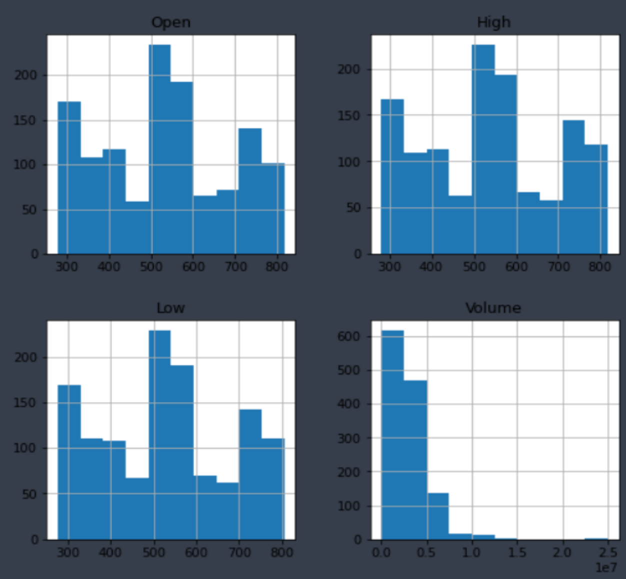
1.2. Data transformation

First, plotted histograms for all numerical columns.

Data transformation

```
In [10]: data_set.hist(figsize = (8,8))
```

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



We can see that no need to apply transformations for dataset.

Non numerical columns are avoided and numerical data separated to standardation and scaling.

```
In [11]: numerical_df = data_set.drop('Date', 1)
         numerical_df.head()
```

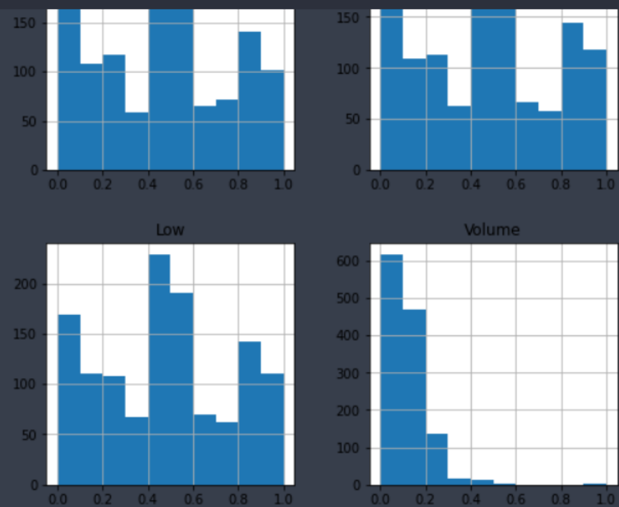
	Open	High	Low	Volume
0	325.25	332.83	324.97	7380500
1	331.27	333.87	329.08	5749400
2	329.83	330.75	326.89	6590300
3	328.34	328.77	323.68	5405900
4	322.04	322.29	309.46	11688800

1.3. Scaling and standardization

Min max scaler used to scale the data.

Scaling and standardization

```
In [12]:
scaler = MinMaxScaler()
new_df = pd.DataFrame(scaler.fit_transform(numerical_df) ,columns = numerical_df.columns)
new_df.hist(figsize = (8,8))
```

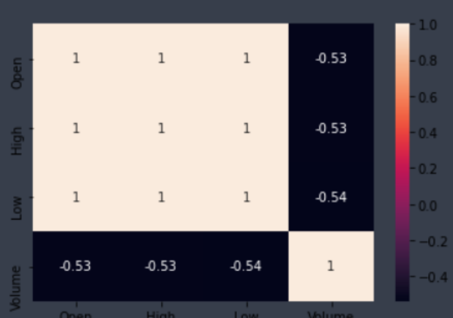


2. Feature Engineering

In feature engineering, the relationships between the features and the optimal feature set to train the ML model are manually decided. To identify the significant features a 'Correlation Matrix' can be constructed. By analyzing it, it can be determined how the features are related to each other or to the target variable.

```
In [13]:
sns.heatmap(new_df.corr(), annot=True)
new_df.corr()
```

	Open	High	Low	Volume
Open	1.000000	0.999692	0.999498	-0.530506
High	0.999692	1.000000	0.999480	-0.526936
Low	0.999498	0.999480	1.000000	-0.538273
Volume	-0.530506	-0.526936	-0.538273	1.000000



“Open”, “High”, “Low” are highly correlated. Therefore, we can use only one column of these 3 variables. “Open” and “Low” variables are dropped.

```
In [14]: # enough getting 1 other and volume field
         new_df=new_df.drop(['Open','Low'],axis=1)
         new_df.shape

(1258, 2)
```

3. Recurrent Neural Network

Recurrent Neural Network are used to evaluate data patterns that change with time serious. This dataset is related with Date. Therefore, RNN model is created for evaluate volume.

First, split a univariate sequence into samples. In this case Multivariate is used to split sequence as follows.

```
In [15]: # split a univariate sequence into samples
         def split_seq_multi(sequence, n_past, n_future):

             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)
```

Window size should be specified and data should be split into samples as follows.

```
In [16]: # specify the window size
         n_steps = 73
         # split into samples
         X, y = split_seq_multi(new_df.to_numpy(), n_steps,1)

In [17]: X.shape

(1185, 73, 2)
```

After, dataset split into train and test. Train_test_split is used to slit data.

```
In [18]:
# 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_train.shape,"     y_train shape:",y_train.shape)
print("X_test shape:",X_test.shape,"      y_test shape:", y_test.shape)

X_train shape: (948, 73, 2)      y_train shape: (948, 1)
X_test shape: (237, 73, 2)       y_test shape: (237, 1)
```

RNN model defined as follows.

```
In [19]:
# define RNN model
model = Sequential()
model.add(LSTM(100, activation='tanh', 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 summery represents no.of parameters and output shape of data of each layer.

```
In [20]:
model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 100)               41200
_________________________________________________________________
dense (Dense)                (None, 50)                5050
_________________________________________________________________
dense_1 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_2 (Dense)              (None, 30)                1530
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 31
=================================================================
Total params: 50,361
Trainable params: 50,361
Non-trainable params: 0
_________________________________________________________________
```

After we can compile and fit the RNN model. I used no.of epochs as 100.

```
In [21]: # compile the model
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
# fit the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=2, validation_data=(X_test, y_test)
```

```
Epoch 90/100
30/30 - 2s - loss: 0.0035 - mae: 0.0344 - val_loss: 0.0019 - val_mae: 0.0275
Epoch 91/100
30/30 - 2s - loss: 0.0033 - mae: 0.0326 - val_loss: 0.0019 - val_mae: 0.0283
Epoch 92/100


30/30 - 2s - loss: 0.0032 - mae: 0.0320 - val_loss: 0.0023 - val_mae: 0.0289
Epoch 93/100
30/30 - 2s - loss: 0.0033 - mae: 0.0337 - val_loss: 0.0020 - val_mae: 0.0276
Epoch 94/100
30/30 - 2s - loss: 0.0034 - mae: 0.0335 - val_loss: 0.0019 - val_mae: 0.0274
Epoch 95/100
30/30 - 2s - loss: 0.0034 - mae: 0.0331 - val_loss: 0.0020 - val_mae: 0.0273
Epoch 96/100
30/30 - 2s - loss: 0.0034 - mae: 0.0343 - val_loss: 0.0020 - val_mae: 0.0295
Epoch 97/100
30/30 - 2s - loss: 0.0032 - mae: 0.0326 - val_loss: 0.0023 - val_mae: 0.0306
Epoch 98/100
30/30 - 2s - loss: 0.0035 - mae: 0.0335 - val_loss: 0.0018 - val_mae: 0.0277
Epoch 99/100
30/30 - 2s - loss: 0.0034 - mae: 0.0333 - val_loss: 0.0018 - val_mae: 0.0270
Epoch 100/100
30/30 - 2s - loss: 0.0033 - mae: 0.0323 - val_loss: 0.0019 - val_mae: 0.0284
```

In every epoch, data loss is happened. Evaluating the model, I changed the no.of epochs and no.of steps.

```
In [22]: # 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: 0.002, RMSE: 0.043, MAE: 0.028
```

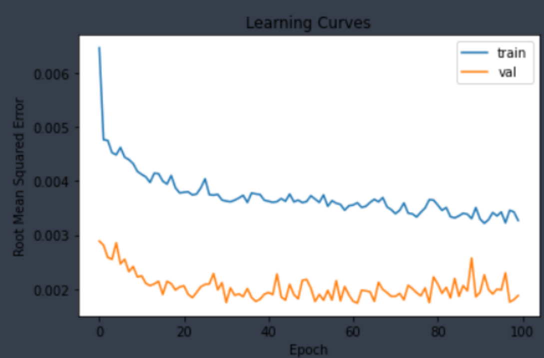
Before evaluate the accuracy of the model, model should evaluate the values using testing dataset.

```
In [23]: # predicting y_test values
print(X_test.shape)
y_pred = model.predict(X_test)
```

```
(237, 73, 2)
```

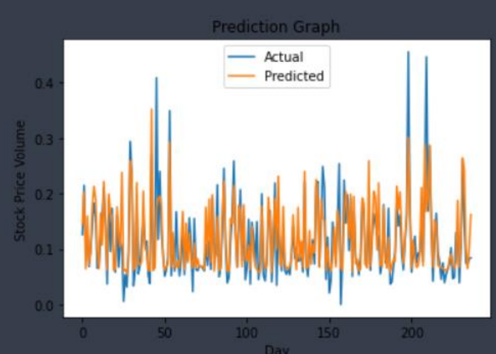
Model accuracy can be represented using graphs. Learning curve is represented the loss values of evaluated and actual values.

```
In [24]: # plot learning curves
         plt.title('Learning Curves')
         plt.xlabel('Epoch')
         plt.ylabel('Root Mean Squared Error')
         plt.plot(history.history['loss'], label='train')
         plt.plot(history.history['val_loss'], label='val')
         plt.legend()
         plt.tight_layout()
         plt.show()
```



Actual values and model predicted values can be compared by using graph as follows.

```
In [25]: plt.plot(y_test)
         plt.plot(y_pred)
         plt.xlabel('Day')
         plt.ylabel('Stock Price Volume')
         plt.title('Prediction Graph')
         plt.legend(['Actual','Predicted'],loc = 'upper center')
         plt.show()
```



Model can be evaluated by using R^2 value as follows.

```
In [26]: y_pred = model.predict(X_test)
         print('R2 Score: ', r2_score(y_test, y_pred))

R2 Score:  0.6250364919203248
```

This model has 62.5% accuracy.