DL Prac 1

April 17, 2024

1 Practical - 1

1.0.1 Problem Statement

Linear regression by using Deep Neural network: Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.

[1]:
```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

#Lets load the dataset and sample some
column_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',␣
↪'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'PRICE']
df = pd.read_csv('housing.csv', header=None, delimiter=r"\s+",␣
↪names=column_names)
```

[2]:
```python
df.head(5)
```

[2]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296.0
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242.0
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242.0
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222.0
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222.0

   PTRATIO       B  LSTAT  PRICE
0     15.3  396.90   4.98   24.0
1     17.8  396.90   9.14   21.6
2     17.8  392.83   4.03   34.7
3     18.7  394.63   2.94   33.4
4     18.7  396.90   5.33   36.2
```

[3]:
```python
# Dimension of the dataset
print(np.shape(df))
```

```
(506, 14)
```

[4]:
```python
# Let's summarize the data to see the distribution of data
print(df.describe())
```

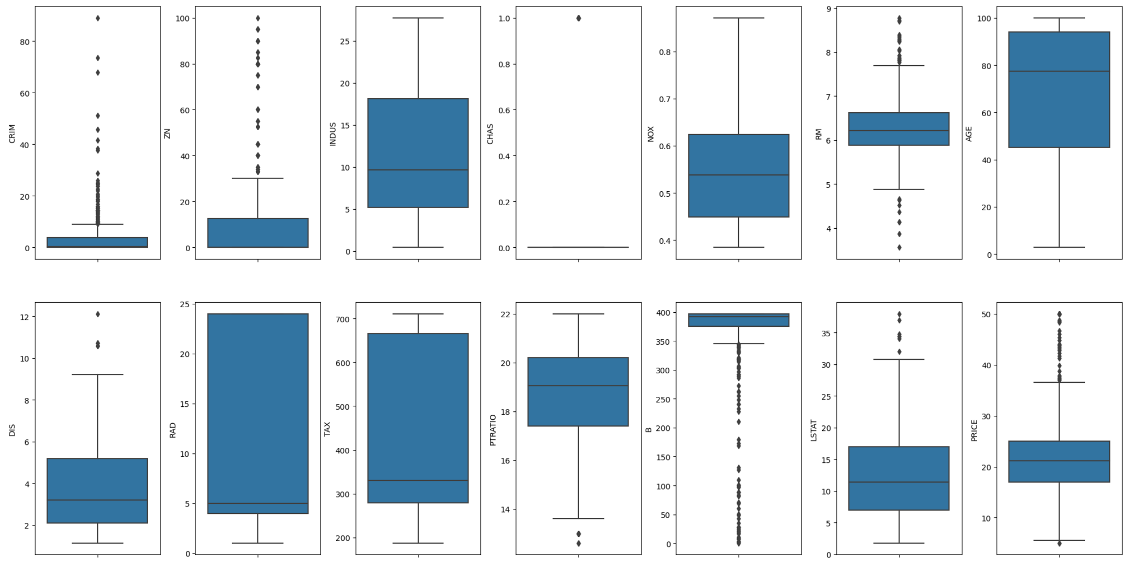
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000   
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634   
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617   
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000   
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500   
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500   
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500   
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000   

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000   
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032   
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864   
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000   
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500   
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000   
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000   
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000   

            LSTAT       PRICE  
count  506.000000  506.000000  
mean    12.653063   22.532806  
std      7.141062    9.197104  
min      1.730000    5.000000  
25%      6.950000   17.025000  
50%     11.360000   21.200000  
75%     16.955000   25.000000  
max     37.970000   50.000000  
```

[5]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

fig, axs = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
axs = axs.flatten()
for k,v in df.items():
    sns.boxplot(y=k, data=df, ax=axs[index])
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```



[6]:
```python
for k, v in df.items():
    q1 = v.quantile(0.25)
    q3 = v.quantile(0.75)
    irq = q3 - q1
    v_col = v[(v <= q1 - 1.5 * irq) | (v >= q3 + 1.5 * irq)]
    perc = np.shape(v_col)[0] * 100.0 / np.shape(df)[0]
    print("Column %s outliers = %.2f%%" % (k, perc))
```

```
Column CRIM outliers = 13.04%
Column ZN outliers = 13.44%
Column INDUS outliers = 0.00%
Column CHAS outliers = 100.00%
Column NOX outliers = 0.00%
Column RM outliers = 5.93%
Column AGE outliers = 0.00%
Column DIS outliers = 0.99%
Column RAD outliers = 0.00%
Column TAX outliers = 0.00%
Column PTRATIO outliers = 2.96%
Column B outliers = 15.22%
Column LSTAT outliers = 1.38%
Column PRICE outliers = 7.91%
```

[7]:
```python
df = df[~(df['PRICE'] >= 35.0)]
print(np.shape(df))
```

```
(458, 14)
```

[8]:
```
#Looking at the data with names and target variable
df.head()
```

[8]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296.0
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242.0
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242.0
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222.0
5  0.02985   0.0   2.18     0  0.458  6.430  58.7  6.0622    3  222.0

   PTRATIO       B  LSTAT  PRICE
0     15.3  396.90   4.98   24.0
1     17.8  396.90   9.14   21.6
2     17.8  392.83   4.03   34.7
3     18.7  394.63   2.94   33.4
5     18.7  394.12   5.21   28.7
```

[9]:
```
#Shape of the data
print(df.shape)
```

```
(458, 14)
```

[10]:
```
#Checking the null values in the dataset
df.isnull().sum()
```

[10]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
PRICE      0
dtype: int64
```

No null values in the dataset, no missing value treatement needed

[11]:
```
#Checking the statistics of the data
df.describe()
```

[11]:
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  458.000000  458.000000  458.000000  458.000000  458.000000  458.000000
```

```
mean     3.880713   10.180131   11.588166    0.058952    0.558875    6.156945
std      8.973996   21.950057    6.756057    0.235792    0.117724    0.563489
min      0.006320    0.000000    0.740000    0.000000    0.385000    3.561000
25%      0.084020    0.000000    5.860000    0.000000    0.453000    5.871250
50%      0.256510    0.000000    9.900000    0.000000    0.538000    6.152000
75%      4.082653    0.000000   18.100000    0.000000    0.624000    6.481750
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  458.000000  458.000000  458.000000  458.000000  458.000000  458.000000
mean    69.170524    3.807797    9.842795  417.893013   18.676201  353.521965
std     28.008853    2.125004    8.884462  168.736868    2.027875   95.363794
min      2.900000    1.137000    1.000000  187.000000   12.600000    0.320000
25%     45.725000    2.100175    4.000000  287.000000   17.600000  373.105000
50%     78.400000    3.199200    5.000000  345.000000   19.200000  391.880000
75%     94.300000    5.214600   24.000000  666.000000   20.200000  396.397500
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT       PRICE
count  458.000000  458.000000
mean    13.490699   20.320087
std      6.967358    6.185151
min      1.980000    5.000000
25%      7.927500   16.200000
50%     12.370000   20.400000
75%     17.302500   23.800000
max     37.970000   34.900000
```

This is sometimes very useful, for example if you look at the CRIM the max is 88.97 and 75% of the value is below 3.677083 and mean is 3.613524 so it means the max values is actually an outlier or there are outliers present in the column

[12]:
```
df.info()
```

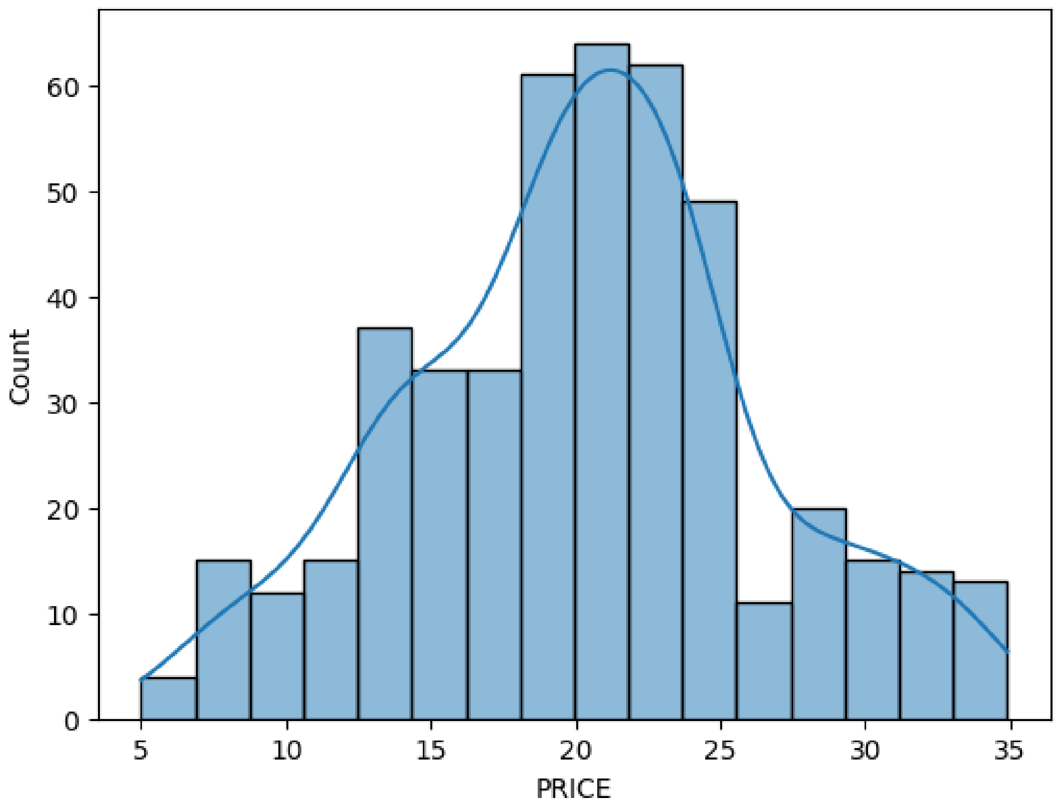
```
<class 'pandas.core.frame.DataFrame'>
Index: 458 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     458 non-null    float64
 1   ZN       458 non-null    float64
 2   INDUS    458 non-null    float64
 3   CHAS     458 non-null    int64
 4   NOX      458 non-null    float64
 5   RM       458 non-null    float64
 6   AGE      458 non-null    float64
 7   DIS      458 non-null    float64
 8   RAD      458 non-null    int64
```

```
 9   TAX       458 non-null    float64
 10  PTRATIO   458 non-null    float64
 11  B         458 non-null    float64
 12  LSTAT     458 non-null    float64
 13  PRICE     458 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 53.7 KB
```

Visualisation

[13]:
```python
#checking the distribution of the target variable
import seaborn as sns
sns.histplot(df.PRICE , kde = True)
```

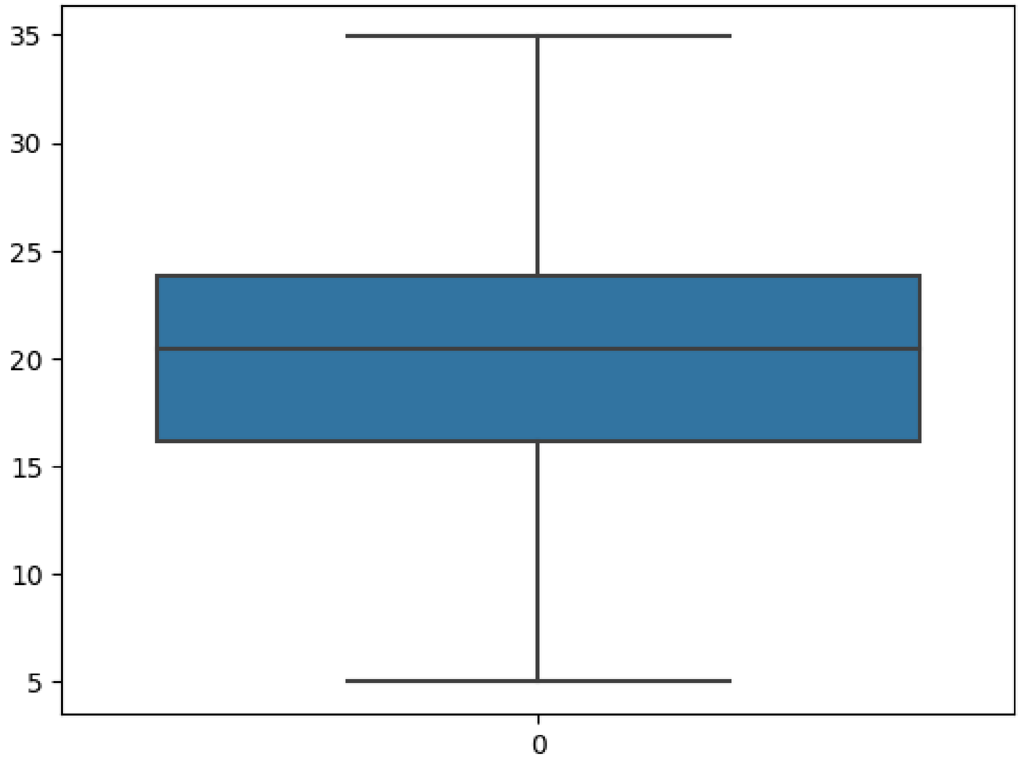
[13]:
```
<Axes: xlabel='PRICE', ylabel='Count'>
```



The distribution seems normal, has not be the data normal we would have perform log transformation or took to square root of the data to make the data normal. Normal distribution is need for the machine learning for better predictiblity of the model

[14]:
```python
#Distribution using box plot
sns.boxplot(df.PRICE)
```

[14]: <Axes: >



Checking the correlation of the independent feature with the dependent feature

Correlation is a statistical technique that can show whether and how strongly pairs of variables are related.An intelligent correlation analysis can lead to a greater understanding of your data

[15]:
```
#checking Correlation of the data
correlation = df.corr()
correlation.loc['PRICE']
```

[15]:
```
CRIM      -0.509111
ZN         0.432791
INDUS     -0.598380
CHAS       0.098362
NOX       -0.584249
RM         0.540151
AGE       -0.571890
DIS        0.461164
RAD       -0.515860
TAX       -0.587285
PTRATIO   -0.471471
B          0.404020
```

```
LSTAT      -0.780531
PRICE       1.000000
Name: PRICE, dtype: float64
```

[16]:
```python
# plotting the heatmap
import matplotlib.pyplot as plt
fig,axes = plt.subplots(figsize=(15,12))
sns.heatmap(correlation,square = True,annot = True)
```

[16]: <Axes: >

By looking at the correlation plot LSAT is negatively correlated with -0.75 and RM is positively correlated to the price and PTRATIO is correlated negatively with -0.51

[17]:
```python
# Checking the scatter plot with the most correlated features
plt.figure(figsize = (20,5))
features = ['LSTAT','RM','PTRATIO']
```

```
for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = df[col]
    y = df.PRICE
    plt.scatter(x, y, marker='o')
    plt.title("Variation in House prices")
    plt.xlabel(col)
    plt.ylabel('"House prices in $1000"')
```

Splitting the dependent feature and independent feature

[18]:
```
#X = data[['LSTAT','RM','PTRATIO']]
X = df.iloc[:,:-1]
y= df.PRICE
```

Splitting the data for Model Validation

[19]:
```
# Splitting the data into train and test for building the model
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2,␣
 ↪random_state = 4)
```

Building the Model

[20]:
```
#Linear Regression
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
```

[21]:
```
#Fitting the model
regressor.fit(X_train,y_train)
```

[21]: LinearRegression()

Model Evaluation

[22]:
```
#Prediction on the test dataset
y_pred = regressor.predict(X_test)
```

[23]:
```
# Predicting RMSE the Test set results
from sklearn.metrics import mean_squared_error
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
print(rmse)
```

```
3.061315764852644
```

[24]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2)
```

```
0.7443761652148535
```

Neural Networks

[25]:
```
#Scaling the dataset
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[26]:
```
#Creating the neural network model
import keras
from keras.layers import Dense, Activation,Dropout
from keras.models import Sequential

model = Sequential()

model.add(Dense(128,activation  = 'relu',input_dim =13))
model.add(Dense(64,activation  = 'relu'))
model.add(Dense(32,activation  = 'relu'))
model.add(Dense(16,activation  = 'relu'))
model.add(Dense(1))
model.compile(optimizer = 'adam',loss = 'mean_squared_error')
```

[27]:
```
model.fit(X_train, y_train, epochs = 100)
```

```
Epoch 1/100
12/12 [==============================] - 1s 2ms/step - loss: 425.8531
Epoch 2/100
12/12 [==============================] - 0s 2ms/step - loss: 348.8212
Epoch 3/100
12/12 [==============================] - 0s 2ms/step - loss: 208.7996
Epoch 4/100
12/12 [==============================] - 0s 2ms/step - loss: 68.4106
Epoch 5/100
12/12 [==============================] - 0s 2ms/step - loss: 44.3092
Epoch 6/100
12/12 [==============================] - 0s 2ms/step - loss: 27.0487
```

```
Epoch 7/100
12/12 [==============================] - 0s 3ms/step - loss: 19.2680
Epoch 8/100
12/12 [==============================] - 0s 2ms/step - loss: 14.8583
Epoch 9/100
12/12 [==============================] - 0s 2ms/step - loss: 12.5224
Epoch 10/100
12/12 [==============================] - 0s 2ms/step - loss: 11.3755
Epoch 11/100
12/12 [==============================] - 0s 1ms/step - loss: 10.3956
Epoch 12/100
12/12 [==============================] - 0s 1ms/step - loss: 9.6852
Epoch 13/100
12/12 [==============================] - 0s 1ms/step - loss: 9.2933
Epoch 14/100
12/12 [==============================] - 0s 1ms/step - loss: 8.8832
Epoch 15/100
12/12 [==============================] - 0s 1ms/step - loss: 8.5272
Epoch 16/100
12/12 [==============================] - 0s 1ms/step - loss: 8.1917
Epoch 17/100
12/12 [==============================] - 0s 1ms/step - loss: 7.9201
Epoch 18/100
12/12 [==============================] - 0s 2ms/step - loss: 7.6381
Epoch 19/100
12/12 [==============================] - 0s 2ms/step - loss: 7.3685
Epoch 20/100
12/12 [==============================] - 0s 2ms/step - loss: 7.3165
Epoch 21/100
12/12 [==============================] - 0s 2ms/step - loss: 7.0080
Epoch 22/100
12/12 [==============================] - 0s 2ms/step - loss: 6.8680
Epoch 23/100
12/12 [==============================] - 0s 2ms/step - loss: 6.7146
Epoch 24/100
12/12 [==============================] - 0s 2ms/step - loss: 6.5021
Epoch 25/100
12/12 [==============================] - 0s 2ms/step - loss: 6.3534
Epoch 26/100
12/12 [==============================] - 0s 2ms/step - loss: 6.2923
Epoch 27/100
12/12 [==============================] - 0s 2ms/step - loss: 6.1283
Epoch 28/100
12/12 [==============================] - 0s 2ms/step - loss: 5.9678
Epoch 29/100
12/12 [==============================] - 0s 1ms/step - loss: 5.9013
Epoch 30/100
12/12 [==============================] - 0s 2ms/step - loss: 5.7838
```

```
Epoch 31/100
12/12 [==============================] - 0s 2ms/step - loss: 5.6921
Epoch 32/100
12/12 [==============================] - 0s 1ms/step - loss: 5.6592
Epoch 33/100
12/12 [==============================] - 0s 2ms/step - loss: 5.5430
Epoch 34/100
12/12 [==============================] - 0s 2ms/step - loss: 5.3618
Epoch 35/100
12/12 [==============================] - 0s 2ms/step - loss: 5.3939
Epoch 36/100
12/12 [==============================] - 0s 2ms/step - loss: 5.2997
Epoch 37/100
12/12 [==============================] - 0s 2ms/step - loss: 5.2949
Epoch 38/100
12/12 [==============================] - 0s 2ms/step - loss: 5.1599
Epoch 39/100
12/12 [==============================] - 0s 2ms/step - loss: 4.9120
Epoch 40/100
12/12 [==============================] - 0s 1ms/step - loss: 4.8994
Epoch 41/100
12/12 [==============================] - 0s 2ms/step - loss: 4.9959
Epoch 42/100
12/12 [==============================] - 0s 2ms/step - loss: 4.7916
Epoch 43/100
12/12 [==============================] - 0s 2ms/step - loss: 4.6412
Epoch 44/100
12/12 [==============================] - 0s 2ms/step - loss: 4.6663
Epoch 45/100
12/12 [==============================] - 0s 2ms/step - loss: 4.6062
Epoch 46/100
12/12 [==============================] - 0s 2ms/step - loss: 4.4449
Epoch 47/100
12/12 [==============================] - 0s 2ms/step - loss: 4.4023
Epoch 48/100
12/12 [==============================] - 0s 2ms/step - loss: 4.2623
Epoch 49/100
12/12 [==============================] - 0s 2ms/step - loss: 4.1582
Epoch 50/100
12/12 [==============================] - 0s 2ms/step - loss: 4.1524
Epoch 51/100
12/12 [==============================] - 0s 2ms/step - loss: 4.0109
Epoch 52/100
12/12 [==============================] - 0s 2ms/step - loss: 4.0082
Epoch 53/100
12/12 [==============================] - 0s 2ms/step - loss: 3.8892
Epoch 54/100
12/12 [==============================] - 0s 2ms/step - loss: 3.9190
```

```
Epoch 55/100
12/12 [==============================] - 0s 2ms/step - loss: 3.8922
Epoch 56/100
12/12 [==============================] - 0s 2ms/step - loss: 3.7423
Epoch 57/100
12/12 [==============================] - 0s 2ms/step - loss: 3.7351
Epoch 58/100
12/12 [==============================] - 0s 2ms/step - loss: 3.8098
Epoch 59/100
12/12 [==============================] - 0s 1ms/step - loss: 3.6451
Epoch 60/100
12/12 [==============================] - 0s 2ms/step - loss: 3.8622
Epoch 61/100
12/12 [==============================] - 0s 2ms/step - loss: 3.5084
Epoch 62/100
12/12 [==============================] - 0s 2ms/step - loss: 3.4727
Epoch 63/100
12/12 [==============================] - 0s 2ms/step - loss: 3.5024
Epoch 64/100
12/12 [==============================] - 0s 3ms/step - loss: 3.4538
Epoch 65/100
12/12 [==============================] - 0s 2ms/step - loss: 3.3805
Epoch 66/100
12/12 [==============================] - 0s 1ms/step - loss: 3.2964
Epoch 67/100
12/12 [==============================] - 0s 2ms/step - loss: 3.4344
Epoch 68/100
12/12 [==============================] - 0s 2ms/step - loss: 3.3076
Epoch 69/100
12/12 [==============================] - 0s 2ms/step - loss: 3.2100
Epoch 70/100
12/12 [==============================] - 0s 2ms/step - loss: 3.2939
Epoch 71/100
12/12 [==============================] - 0s 2ms/step - loss: 3.1752
Epoch 72/100
12/12 [==============================] - 0s 1ms/step - loss: 3.1794
Epoch 73/100
12/12 [==============================] - 0s 2ms/step - loss: 3.1633
Epoch 74/100
12/12 [==============================] - 0s 2ms/step - loss: 3.0420
Epoch 75/100
12/12 [==============================] - 0s 1ms/step - loss: 2.9316
Epoch 76/100
12/12 [==============================] - 0s 1ms/step - loss: 2.8973
Epoch 77/100
12/12 [==============================] - 0s 2ms/step - loss: 2.9141
Epoch 78/100
12/12 [==============================] - 0s 2ms/step - loss: 2.8537
```

```
Epoch 79/100
12/12 [==============================] - 0s 2ms/step - loss: 3.0466
Epoch 80/100
12/12 [==============================] - 0s 2ms/step - loss: 3.0060
Epoch 81/100
12/12 [==============================] - 0s 2ms/step - loss: 3.1338
Epoch 82/100
12/12 [==============================] - 0s 2ms/step - loss: 2.8544
Epoch 83/100
12/12 [==============================] - 0s 2ms/step - loss: 2.7994
Epoch 84/100
12/12 [==============================] - 0s 2ms/step - loss: 2.7240
Epoch 85/100
12/12 [==============================] - 0s 1ms/step - loss: 3.0241
Epoch 86/100
12/12 [==============================] - 0s 2ms/step - loss: 2.8542
Epoch 87/100
12/12 [==============================] - 0s 2ms/step - loss: 2.8366
Epoch 88/100
12/12 [==============================] - 0s 2ms/step - loss: 2.6345
Epoch 89/100
12/12 [==============================] - 0s 2ms/step - loss: 2.6547
Epoch 90/100
12/12 [==============================] - 0s 2ms/step - loss: 2.6072
Epoch 91/100
12/12 [==============================] - 0s 2ms/step - loss: 2.7143
Epoch 92/100
12/12 [==============================] - 0s 2ms/step - loss: 2.8281
Epoch 93/100
12/12 [==============================] - 0s 2ms/step - loss: 2.6895
Epoch 94/100
12/12 [==============================] - 0s 2ms/step - loss: 2.5827
Epoch 95/100
12/12 [==============================] - 0s 1ms/step - loss: 2.4230
Epoch 96/100
12/12 [==============================] - 0s 2ms/step - loss: 2.5258
Epoch 97/100
12/12 [==============================] - 0s 2ms/step - loss: 2.3642
Epoch 98/100
12/12 [==============================] - 0s 2ms/step - loss: 2.5008
Epoch 99/100
12/12 [==============================] - 0s 1ms/step - loss: 2.4174
Epoch 100/100
12/12 [==============================] - 0s 2ms/step - loss: 2.3897
```

```
[27]: <keras.callbacks.History at 0x27c8215be50>
```

Evaluation of the model

[28]:
```python
y_pred = model.predict(X_test)
```

```
3/3 [==============================] - 0s 2ms/step
```

[29]:
```python
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2)
```

```
0.824490637896401
```

[30]:
```python
# Predicting RMSE the Test set results
from sklearn.metrics import mean_squared_error
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
print(rmse)
```

```
2.5366327719812016
```