Practical No 01

Linear regression by using Deep Neural network: Implement Boston housing price predictionproblem by Linear regression using Deep Neural network. Use Boston House price predictiondataset.

```
## Importing required Library
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split,KFold,cross_val_score
from sklearn.preprocessing import StandardScaler,MinMaxScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Conv2D,MaxPooling2D,Flatten
import warnings
warnings.filterwarnings('ignore')
```

```
#Reading Dataset
```

```
housing_data = pd.read_csv('BostonHousing.csv')
```

```
# Finding Top 5 Result
```

```
housing_data.head()
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90

```
# Analysing Dataset
```

```
housing_data.describe()
```

	crim	zn	indus	chas	nox	rm	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.57
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.14
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.90
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.02
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.50
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.07
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.00

```
#Finding any Null Values
```

```
housing_data.isnull().sum()
```

```
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
medv       0
dtype: int64
```

```
#Finding Duplicate Values
```

```
housing_data.duplicated().sum()
```

```
0
```

```
#DataSet Distributing Traning and testing
```

```
X = housing_data.drop(columns = ['medv'])
y = housing_data.medv
sc = StandardScaler()
X = sc.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state = 5)
```

```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

```
((354, 13), (152, 13), (354,), (152,))
```

```
#Model Training
```

```
model = Sequential()
model.add(Dense(128, input_shape=(13, ), activation='relu', name='dense_1'))
model.add(Dense(64, activation='relu', name='dense_2'))
model.add(Dense(1, activation='linear', name='dense_output'))
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 128)               1792

 dense_2 (Dense)             (None, 64)                8256

 dense_output (Dense)        (None, 1)                 65

=================================================================
Total params: 10,113
Trainable params: 10,113
Non-trainable params: 0
_________________________________________________________________
```

```
#Model Fitting with 200 Epochs
```

```
history = model.fit(X_train, y_train, epochs=200, validation_split=0.2)
```

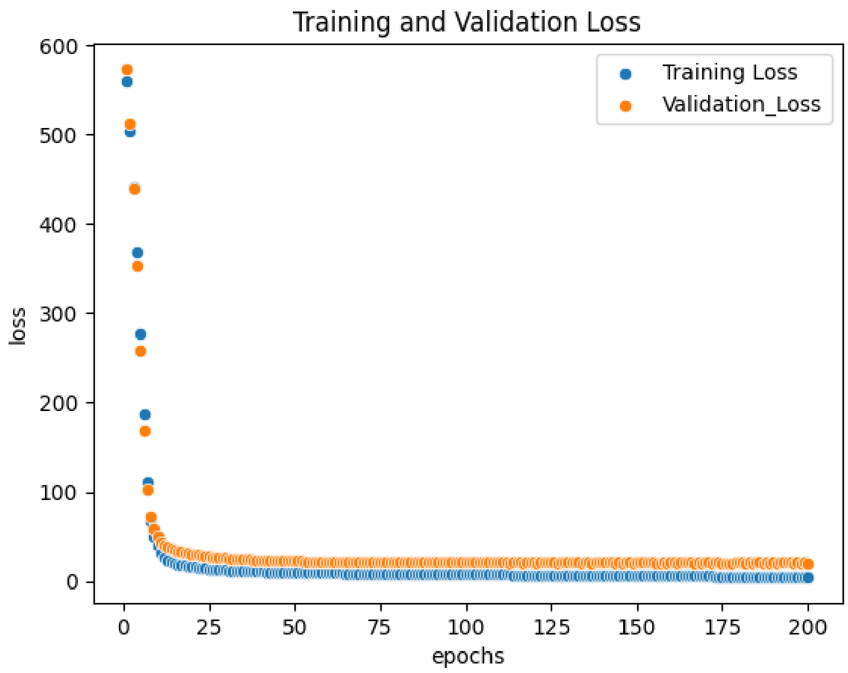
```
Epoch 175/200
9/9 [==============================] - 0s 11ms/step - loss: 5.1111 - mae: 1.5756 - val_loss: 20.3476 -
Epoch 176/200
9/9 [==============================] - 0s 12ms/step - loss: 5.0803 - mae: 1.5806 - val_loss: 20.4396 -
Epoch 177/200
9/9 [==============================] - 0s 9ms/step - loss: 5.1509 - mae: 1.5663 - val_loss: 20.3926 - v
Epoch 178/200
9/9 [==============================] - 0s 11ms/step - loss: 5.0448 - mae: 1.5821 - val_loss: 20.3617 -
Epoch 179/200
9/9 [==============================] - 0s 9ms/step - loss: 5.0559 - mae: 1.6029 - val_loss: 20.5059 - v
Epoch 180/200
9/9 [==============================] - 0s 9ms/step - loss: 5.0407 - mae: 1.5555 - val_loss: 20.4755 - v
Epoch 181/200
9/9 [==============================] - 0s 8ms/step - loss: 5.0905 - mae: 1.5621 - val_loss: 20.9358 - v
Epoch 182/200
9/9 [==============================] - 0s 10ms/step - loss: 5.0026 - mae: 1.5719 - val_loss: 20.3094 -
Epoch 183/200
9/9 [==============================] - 0s 9ms/step - loss: 4.8622 - mae: 1.5599 - val_loss: 20.7895 - v
Epoch 184/200
9/9 [==============================] - 0s 8ms/step - loss: 4.8883 - mae: 1.5285 - val_loss: 20.2752 - v
Epoch 185/200
9/9 [==============================] - 0s 10ms/step - loss: 5.0436 - mae: 1.5816 - val_loss: 20.4656 -
Epoch 186/200
9/9 [==============================] - 0s 10ms/step - loss: 4.9210 - mae: 1.5656 - val_loss: 20.5648 -
Epoch 187/200
9/9 [==============================] - 0s 9ms/step - loss: 4.8744 - mae: 1.5169 - val_loss: 20.1752 - v
Epoch 188/200
9/9 [==============================] - 0s 11ms/step - loss: 4.7451 - mae: 1.5279 - val_loss: 20.8158 -
Epoch 189/200
9/9 [==============================] - 0s 14ms/step - loss: 4.7845 - mae: 1.5242 - val_loss: 20.3473 -
Epoch 190/200
9/9 [==============================] - 0s 13ms/step - loss: 4.8478 - mae: 1.5418 - val_loss: 20.7425 -
Epoch 191/200
9/9 [==============================] - 0s 9ms/step - loss: 4.8055 - mae: 1.5250 - val_loss: 20.2817 - v
Epoch 192/200
9/9 [==============================] - 0s 9ms/step - loss: 4.6039 - mae: 1.4931 - val_loss: 20.6244 - v
Epoch 193/200
9/9 [==============================] - 0s 11ms/step - loss: 4.6854 - mae: 1.5392 - val_loss: 20.5111 -
Epoch 194/200
9/9 [==============================] - 0s 8ms/step - loss: 4.6542 - mae: 1.5006 - val_loss: 20.5895 - v
Epoch 195/200
9/9 [==============================] - 0s 6ms/step - loss: 4.6124 - mae: 1.5000 - val_loss: 20.5493 - v
Epoch 196/200
9/9 [==============================] - 0s 6ms/step - loss: 4.6775 - mae: 1.5165 - val_loss: 20.4457 - v
Epoch 197/200
9/9 [==============================] - 0s 8ms/step - loss: 4.6487 - mae: 1.5125 - val_loss: 20.5668 - v
Epoch 198/200
9/9 [==============================] - 0s 8ms/step - loss: 4.6026 - mae: 1.4859 - val_loss: 20.3844 - v
Epoch 199/200
9/9 [==============================] - 0s 8ms/step - loss: 4.5009 - mae: 1.4756 - val_loss: 20.7519 - v
Epoch 200/200
9/9 [==============================] - 0s 8ms/step - loss: 4.4873 - mae: 1.4943 - val_loss: 20.4057 - v
```

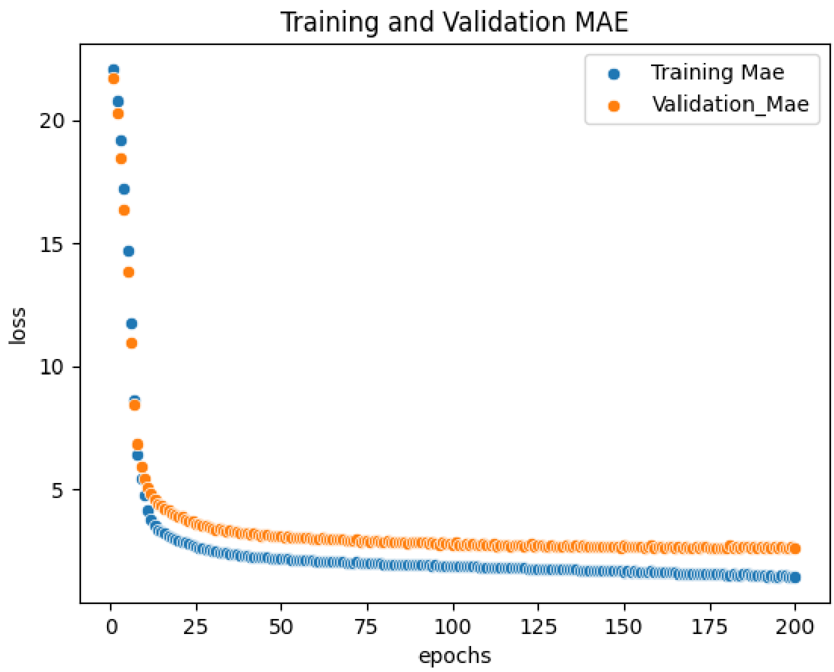
```
print(len(history.history['mae']))
```

```
200
```

```
#Visualizing Training and Validation Loss
```

```
sns.scatterplot(y = history.history['loss'],x = range(1,200+1))
sns.scatterplot(y = history.history['val_loss'],x = range(1,200+1))
plt.title('Training and Validation Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend(['Training Loss','Validation_Loss'])
plt.show()
sns.scatterplot(y = history.history['mae'],x = range(1,200+1))
sns.scatterplot(y = history.history['val_mae'],x = range(1,200+1))
plt.title('Training and Validation MAE')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend(['Training Mae','Validation_Mae'])
plt.show()
```






```
1 #Evaluating Error
```

```
1 mse_nn, mae_nn = model.evaluate(X_test, y_test)
2 print('Mean squared error on test data: ', mse_nn)
3 print('Mean absolute error on test data: ', mae_nn)
```

```
5/5 [==============================] - 0s 3ms/step - loss: 14.2418 - mae: 2.3073
Mean squared error on test data:  14.24176025390625
Mean absolute error on test data:  2.3072948455810547
```

```
1
```

✓ 0s completed at 1:30 PM