Assignment No 1: 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.

```
import numpy as np
import pandas as pd
df = pd.read_csv("1_boston_housing.csv")
df.head()
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   crim     506 non-null    float64
 1   zn       506 non-null    float64
 2   indus    506 non-null    float64
 3   chas     506 non-null    int64  
 4   nox      506 non-null    float64
 5   rm       506 non-null    float64
 6   age      506 non-null    float64
 7   dis      506 non-null    float64
 8   rad      506 non-null    int64  
 9   tax      506 non-null    int64  
 10  ptratio  506 non-null    float64
 11  b        506 non-null    float64
 12  lstat    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

```
from sklearn.model_selection import train_test_split

X = df.loc[:, df.columns != 'MEDV']
y = df.loc[:, df.columns == 'MEDV']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=123)
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Assuming you have already split your data into training and testing sets (X_train, X_test, y_train, y_test)

# Linear Regression model
regressor = LinearRegression()

# Fitting the model
regressor.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
# Predictions on the test set
y_pred = regressor.predict(X_test)

# Calculating mean squared error and mean absolute error
mse_lr = mean_squared_error(y_test, y_pred)
mae_lr = mean_absolute_error(y_test, y_pred)

print('Mean squared error on test data: ', mse_lr)
print('Mean absolute error on test data: ', mae_lr)
```

```
Mean squared error on test data:  28.405854810508146
Mean absolute error on test data:  3.6913626771162664
```

```
from sklearn.preprocessing import StandardScaler
mms = StandardScaler()
mms.fit(X_train)
X_train = mms.transform(X_train)
X_test = mms.transform(X_test)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

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(32, activation='relu', name='dense_3'))
model.add(Dense(16, activation='relu', name='dense_4'))
model.add(Dense(8, activation='relu', name='dense_5'))
model.add(Dense(1, activation='relu', name='dense_output'))

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

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 128)               1792

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 32)                2080

 dense_4 (Dense)             (None, 16)                528

 dense_5 (Dense)             (None, 8)                 136

 dense_output (Dense)        (None, 1)                 9

=================================================================
Total params: 12801 (50.00 KB)
Trainable params: 12801 (50.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
history = model.fit(X_train, y_train, epochs=110, validation_split=0.05, verbose = 1)
```

```
Epoch 1/110
11/11 [==============================] - 1s 25ms/step - loss: 593.3658 - mae: 22.5302 - val_loss: 615.3805 - val_mae: 22.9291
Epoch 2/110
11/11 [==============================] - 0s 6ms/step - loss: 559.3401 - mae: 21.7613 - val_loss: 558.8005 - val_mae: 21.7347
Epoch 3/110
11/11 [==============================] - 0s 8ms/step - loss: 478.7791 - mae: 19.8663 - val_loss: 433.7032 - val_mae: 18.8656
Epoch 4/110
11/11 [==============================] - 0s 8ms/step - loss: 315.5183 - mae: 15.4785 - val_loss: 200.4464 - val_mae: 12.1302
Epoch 5/110
11/11 [==============================] - 0s 6ms/step - loss: 109.7954 - mae: 8.2282 - val_loss: 43.5370 - val_mae: 4.9452
Epoch 6/110
11/11 [==============================] - 0s 8ms/step - loss: 61.0049 - mae: 6.1297 - val_loss: 26.6826 - val_mae: 4.2462
Epoch 7/110
11/11 [==============================] - 0s 8ms/step - loss: 33.9542 - mae: 4.2219 - val_loss: 23.3544 - val_mae: 3.8470
Epoch 8/110
11/11 [==============================] - 0s 6ms/step - loss: 26.5743 - mae: 3.6450 - val_loss: 13.6337 - val_mae: 2.8756
Epoch 9/110
11/11 [==============================] - 0s 6ms/step - loss: 22.4963 - mae: 3.5466 - val_loss: 12.4883 - val_mae: 2.6129
Epoch 10/110
11/11 [==============================] - 0s 6ms/step - loss: 21.2775 - mae: 3.4195 - val_loss: 12.2912 - val_mae: 2.4545
Epoch 11/110
11/11 [==============================] - 0s 6ms/step - loss: 19.3669 - mae: 3.2213 - val_loss: 12.3605 - val_mae: 2.5805
Epoch 12/110
11/11 [==============================] - 0s 6ms/step - loss: 18.2684 - mae: 3.1172 - val_loss: 11.8856 - val_mae: 2.5983
Epoch 13/110
11/11 [==============================] - 0s 7ms/step - loss: 17.4588 - mae: 3.0280 - val_loss: 11.5354 - val_mae: 2.5655
Epoch 14/110
11/11 [==============================] - 0s 8ms/step - loss: 16.4371 - mae: 2.9754 - val_loss: 11.1261 - val_mae: 2.5948
Epoch 15/110
11/11 [==============================] - 0s 9ms/step - loss: 15.6947 - mae: 2.8708 - val_loss: 10.8160 - val_mae: 2.5181
Epoch 16/110
11/11 [==============================] - 0s 7ms/step - loss: 15.0158 - mae: 2.8691 - val_loss: 10.0800 - val_mae: 2.4065
Epoch 17/110
11/11 [==============================] - 0s 6ms/step - loss: 14.4069 - mae: 2.7958 - val_loss: 9.9425 - val_mae: 2.4324
Epoch 18/110
11/11 [==============================] - 0s 8ms/step - loss: 13.6935 - mae: 2.6957 - val_loss: 9.5205 - val_mae: 2.4581
```

```
Epoch 19/110
11/11 [==============================] - 0s 6ms/step - loss: 13.1832 - mae: 2.6794 - val_loss: 9.1794 - val_mae: 2.4019
Epoch 20/110
11/11 [==============================] - 0s 6ms/step - loss: 12.8753 - mae: 2.5996 - val_loss: 9.0113 - val_mae: 2.3928
Epoch 21/110
11/11 [==============================] - 0s 6ms/step - loss: 12.5647 - mae: 2.6135 - val_loss: 8.6027 - val_mae: 2.2958
Epoch 22/110
11/11 [==============================] - 0s 7ms/step - loss: 12.2744 - mae: 2.5206 - val_loss: 8.6211 - val_mae: 2.3033
Epoch 23/110
11/11 [==============================] - 0s 5ms/step - loss: 11.5889 - mae: 2.5146 - val_loss: 8.8627 - val_mae: 2.3396
Epoch 24/110
11/11 [==============================] - 0s 8ms/step - loss: 11.3273 - mae: 2.4649 - val_loss: 8.4371 - val_mae: 2.2638
Epoch 25/110
11/11 [==============================] - 0s 6ms/step - loss: 11.1090 - mae: 2.4066 - val_loss: 8.6908 - val_mae: 2.3081
Epoch 26/110
11/11 [==============================] - 0s 6ms/step - loss: 10.8522 - mae: 2.4206 - val_loss: 8.1354 - val_mae: 2.2012
Epoch 27/110
11/11 [==============================] - 0s 6ms/step - loss: 10.5311 - mae: 2.3327 - val_loss: 8.0582 - val_mae: 2.1655
Epoch 28/110
11/11 [==============================] - 0s 6ms/step - loss: 10.3425 - mae: 2.3330 - val_loss: 8.5572 - val_mae: 2.1804
Epoch 29/110
```

```
mse_nn, mae_nn = model.evaluate(X_test, y_test)

print('Mean squared error on test data: ', mse_nn)
print('Mean absolute error on test data: ', mae_nn)
```

```
5/5 [==============================] - 0s 3ms/step - loss: 19.3763 - mae: 2.8848
Mean squared error on test data:  19.376283645629883
Mean absolute error on test data:  2.884821653366089
```

```
import sklearn
new_data = [[11.5779, 0,18.1, 0, 0.7, 5.036, 97, 1.77, 3, 666, 20.2, 396.9, 25.68]]
new_data = sklearn.preprocessing.StandardScaler().fit_transform((new_data))
prediction = model.predict(new_data)
print("Predicted house price:", prediction)#9.7  ==394
```

```
1/1 [==============================] - 0s 80ms/step
Predicted house price: [[10.922903]]
```