```
import pandas as pd
import numpy as np
```

```
df=pd.read_csv('BostonHousing.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.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
```

```
X = pd.DataFrame(np.c_[df['lstat'], df['rm']], columns = ['lstat','rm'])
Y = df['medv']
```

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x_scaled=scaler.fit_transform(X)
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(x_scaled, Y, test_size = 0.20, random_state=12)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(404, 2)
(102, 2)
(404,)
(102,)
```

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

lin_model = LinearRegression()
lin_model.fit(X_train, Y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
from sklearn.metrics import mean_squared_error, r2_score

# Your code for model training and prediction goes here

# model evaluation for training set
y_train_predict = lin_model.predict(X_train)
rmse_train = np.sqrt(mean_squared_error(Y_train, y_train_predict))
r2_train = r2_score(Y_train, y_train_predict)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse_train))
print('R2 score is {}'.format(r2_train))
print("\n")

# model evaluation for testing set
y_test_predict = lin_model.predict(X_test)
rmse_test = np.sqrt(mean_squared_error(Y_test, y_test_predict))
r2_test = r2_score(Y_test, y_test_predict)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse_test))
print('R2 score is {}'.format(r2_test))
```

```
The model performance for training set
--------------------------------------
RMSE is 5.5501562067052665
R2 score is 0.6376199854440077


The model performance for testing set
--------------------------------------
RMSE is 5.431992943356162
R2 score is 0.6388882580693107
```

```
from sklearn.neural_network import MLPRegressor
```

```
mlp=MLPRegressor(hidden_layer_sizes=(64,16))
```

```
mlp.fit(X_train,Y_train)
```

```
MLPRegressor(hidden_layer_sizes=(64, 16))
```

```
y_pred1=mlp.predict(X_test)
```

```
r2_score(Y_test,y_pred1)
```

```
0.7924576325461047
```

```
Start coding or generate with AI.
```

```
### Sequential model###
```

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

```
model=Sequential()
```

```
# define the deep neural network model
from tensorflow.keras.layers import Dense # dense means fully connected layer
model.add(Dense(64,activation='relu',input_shape=(X_train.shape[1],)))
```

```
model.add(Dense(64,activation='relu'))
```

```
model.add(Dense(1))
```

```
#compile model
# optimizer tries to adjust the weights
from sklearn.metrics import mean_squared_error
from tensorflow.keras.optimizers import Adam
model.compile(optimizer=Adam(),loss='mean_squared_error')


#train the model
#verbose will print all the info of the model when it is training
tm=model.fit(X_train,Y_train,epochs=100,batch_size=32,validation_split=0.20,verbose=1)
```

```
Epoch 1/100
11/11 [==============================] - 2s 35ms/step - loss: 589.4464 - val_loss: 543.7251
Epoch 2/100
11/11 [==============================] - 0s 13ms/step - loss: 567.1978 - val_loss: 522.3540
Epoch 3/100
11/11 [==============================] - 0s 10ms/step - loss: 541.7750 - val_loss: 496.5316
Epoch 4/100
11/11 [==============================] - 0s 8ms/step - loss: 510.9503 - val_loss: 463.2098
Epoch 5/100
11/11 [==============================] - 0s 8ms/step - loss: 469.2240 - val_loss: 420.7393
Epoch 6/100
11/11 [==============================] - 0s 5ms/step - loss: 417.7153 - val_loss: 367.1132
Epoch 7/100
11/11 [==============================] - 0s 7ms/step - loss: 353.7452 - val_loss: 305.3683
Epoch 8/100
11/11 [==============================] - 0s 5ms/step - loss: 281.5373 - val_loss: 238.5807
Epoch 9/100
11/11 [==============================] - 0s 7ms/step - loss: 210.1311 - val_loss: 176.9558
Epoch 10/100
11/11 [==============================] - 0s 5ms/step - loss: 146.3537 - val_loss: 123.2198
Epoch 11/100
11/11 [==============================] - 0s 5ms/step - loss: 94.1605 - val_loss: 85.3173
Epoch 12/100
11/11 [==============================] - 0s 7ms/step - loss: 62.0638 - val_loss: 64.3899
Epoch 13/100
11/11 [==============================] - 0s 5ms/step - loss: 49.5232 - val_loss: 54.1167
Epoch 14/100
11/11 [==============================] - 0s 6ms/step - loss: 43.9593 - val_loss: 48.4843
Epoch 15/100
11/11 [==============================] - 0s 5ms/step - loss: 41.1079 - val_loss: 44.5764
Epoch 16/100
11/11 [==============================] - 0s 7ms/step - loss: 38.8711 - val_loss: 41.6747
Epoch 17/100
11/11 [==============================] - 0s 7ms/step - loss: 37.1791 - val_loss: 39.7498
Epoch 18/100
11/11 [==============================] - 0s 5ms/step - loss: 35.5611 - val_loss: 38.1785
Epoch 19/100
11/11 [==============================] - 0s 7ms/step - loss: 34.4165 - val_loss: 36.9250
Epoch 20/100
11/11 [==============================] - 0s 6ms/step - loss: 33.4523 - val_loss: 35.6283
Epoch 21/100
11/11 [==============================] - 0s 5ms/step - loss: 32.5593 - val_loss: 34.4903
Epoch 22/100
11/11 [==============================] - 0s 7ms/step - loss: 31.7490 - val_loss: 33.4304
Epoch 23/100
11/11 [==============================] - 0s 7ms/step - loss: 31.0709 - val_loss: 32.7053
Epoch 24/100
11/11 [==============================] - 0s 5ms/step - loss: 30.3946 - val_loss: 31.8115
Epoch 25/100
11/11 [==============================] - 0s 6ms/step - loss: 29.8108 - val_loss: 31.0981
Epoch 26/100
11/11 [==============================] - 0s 7ms/step - loss: 29.2015 - val_loss: 30.2778
Epoch 27/100
11/11 [==============================] - 0s 7ms/step - loss: 28.7507 - val_loss: 29.5658
Epoch 28/100
11/11 [==============================] - 0s 6ms/step - loss: 28.2830 - val_loss: 28.9484
Epoch 29/100
11/11 [==============================] - 0s 5ms/step - loss: 27.8280 - val_loss: 28.4684
```

```
y_pred=model.predict(X_test)
```

```
4/4 [==============================] - 0s 5ms/step
```

```
mse=mean_squared_error(Y_test,y_pred)
print(mse)
```

```
16.530726416537878
```

```
r2_score(Y_test,y_pred)
```

```
0.797691022824729
```

Start coding or generate with AI.