

#Phapale Apeksha Roll NO: 4231 Div:B

DL Practical NO.1

import pandas as pd

import numpy as np

df = pd.read_csv(r'C:\Users\apeksha\Downloads\DL\pact_1\Boston_Housing_Data.csv')

print(df)

print("Shape of X:")

X = df.drop("medv", axis=1).values

print(X.shape)

print("Shape of y:")

y = df["medv"].values

print(y.shape)

Set the random seed for reproducibility

np.random.seed(42)

Split the data into training and testing sets

test_size = 0.2 # Percentage of data to be used for testing

num_test_samples = int(test_size * len(df))

Shuffle the indices of the data randomly

shuffled_indices = np.random.permutation(len(df))

Split the indices into training and test sets

test_indices = shuffled_indices[num_test_samples:]

train_indices = shuffled_indices[:num_test_samples]

Split the features and target variable based on the indices

X_train = X[train_indices]

X_test = X[test_indices]

y_train = y[train_indices]

y_test = y[test_indices]

print("Shape of X_train:", X_train.shape)

print("Shape of X_test:", X_test.shape)

print("Shape of y_train:", y_train.shape)

print("Shape of y_test:", y_test.shape)

#Data preprocessing

```
mean=X_train.mean(axis=0)
std=X_train.std(axis=0)
X_train=(X_train-mean)/std
X_test=(X_test-mean)/std
print(X_train[0])
print(y_train[0])
```

#building model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model=Sequential()
model.add(Dense(128,activation='relu',input_shape=(X_train[0].shape)))
model.add(Dense(64,activation='relu'))
model.add(Dense(1,activation='linear'))
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
print(model.summary())
```

#training our model

```
model.fit(X_train, y_train, epochs=100, batch_size=1, verbose=1,validation_data=(X_test, y_test))
print(X_test[8])
```

#Testing our Model

```
test_input=[[-0.395866 , -0.49954771 , 0.26756998, -0.27771867, -1.01383089 ,-0.0421327,
-0.84072005 , 0.32578231, -0.50700864, -0.03630072, 0.17125245 , 0.29430897,
-0.49453011]]
print("Actual Output :",y_test[8])
print("Predicted Output :",model.predict(test_input))
```

#model evaluation

```
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)
from sklearn.metrics import r2_score
y_dl=model.predict(X_test)
r2=r2_score(y_test,y_dl)
print('R2 Score :',r2)
```

Output:

```
crim  zn  indus  chas  nox ... tax  ptratio    b  lstat  medv
0  0.00632  18.0  2.31   0  0.538 ... 296   15.3  396.90  4.98  24.0
1  0.02731  0.0  7.07   0  0.469 ... 242   17.8  396.90  9.14  21.6
2  0.02729  0.0  7.07   0  0.469 ... 242   17.8  392.83  4.03  34.7
3  0.03237  0.0  2.18   0  0.458 ... 222   18.7  394.63  2.94  33.4
4  0.06905  0.0  2.18   0  0.458 ... 222   18.7  396.90  5.33  36.2
..  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
501 0.06263  0.0 11.93   0  0.573 ... 273   21.0  391.99  9.67  22.4
502 0.04527  0.0 11.93   0  0.573 ... 273   21.0  396.90  9.08  20.6
503 0.06076  0.0 11.93   0  0.573 ... 273   21.0  396.90  5.64  23.9
504 0.10959  0.0 11.93   0  0.573 ... 273   21.0  393.45  6.48  22.0
505 0.04741  0.0 11.93   0  0.573 ... 273   21.0  396.90  7.88  11.9
```

[506 rows x 14 columns]

Shape of X:

(506, 13)

Shape of y:

(506,)

Shape of X_train: (405, 13)

Shape of X_test: (101, 13)

Shape of y_train: (405,)

Shape of y_test: (101,)

```
[-0.395866 -0.49954771 0.26756998 -0.27771867 -1.01383089 -0.0421327
-0.84072005 0.32578231 -0.50700864 -0.03630072 0.17125245 0.29430897
-0.49453011]
```

21.4

Model: "sequential_10"

Layer (type)	Output Shape	Param #
dense_28 (Dense)	(None, 128)	1792
dense_29 (Dense)	(None, 64)	8256
dense_30 (Dense)	(None, 1)	65
Total params: 10,113		
Trainable params: 10,113		

Non-trainable params: 0

None

Epoch 1/100

405/405 [=====] - 1s 2ms/step - loss: 131.9694 - mae: 7.6868 - val_loss: 23.2772 - val_mae: 3.1643

Epoch 2/100

405/405 [=====] - 1s 1ms/step - loss: 20.3213 - mae: 3.0870 - val_loss: 18.0007 - val_mae: 2.8652

Epoch 3/100

405/405 [=====] - 1s 1ms/step - loss: 16.0820 - mae: 2.8830 - val_loss: 15.3380 - val_mae: 2.5316

Epoch 4/100

405/405 [=====] - 1s 1ms/step - loss: 14.1997 - mae: 2.6933 - val_loss: 13.7441 - val_mae: 2.4574

Epoch 5/100

405/405 [=====] - 1s 1ms/step - loss: 13.6563 - mae: 2.6144 - val_loss: 13.6630 - val_mae: 2.3891

Epoch 6/100

405/405 [=====] - 1s 1ms/step - loss: 11.9183 - mae: 2.4655 - val_loss: 12.9116 - val_mae: 2.3726

Epoch 7/100

405/405 [=====] - 1s 1ms/step - loss: 12.1821 - mae: 2.5067 - val_loss: 13.0036 - val_mae: 2.4194

Epoch 8/100

405/405 [=====] - 1s 1ms/step - loss: 11.1841 - mae: 2.4717 - val_loss: 14.4589 - val_mae: 2.6005

Epoch 9/100

405/405 [=====] - 1s 1ms/step - loss: 10.7131 - mae: 2.3939 - val_loss: 11.9493 - val_mae: 2.2703

Epoch 10/100

405/405 [=====] - 1s 1ms/step - loss: 9.7527 - mae: 2.3094 - val_loss: 12.7310 - val_mae: 2.4561

Epoch 11/100

405/405 [=====] - 1s 1ms/step - loss: 9.6041 - mae: 2.3094 - val_loss: 13.2768 - val_mae: 2.5256

Epoch 12/100

405/405 [=====] - 1s 1ms/step - loss: 9.2582 - mae: 2.3332 - val_loss: 13.4592 - val_mae: 2.4178

Epoch 13/100

405/405 [=====] - 1s 1ms/step - loss: 9.2426 - mae: 2.2480 - val_loss: 12.5691 - val_mae: 2.4292

Epoch 14/100

405/405 [=====] - 1s 1ms/step - loss: 8.3951 - mae: 2.1531 - val_loss: 11.9299 - val_mae: 2.2402

Epoch 15/100

405/405 [=====] - 1s 2ms/step - loss: 8.2520 - mae: 2.1126 - val_loss: 13.0701 - val_mae: 2.4060

Epoch 16/100

405/405 [=====] - 1s 1ms/step - loss: 8.0935 - mae: 2.1551 - val_loss: 12.3728 - val_mae: 2.3272

Epoch 17/100

405/405 [=====] - 1s 1ms/step - loss: 7.5124 - mae: 2.0854 - val_loss: 12.7399 - val_mae: 2.3531

Epoch 18/100

405/405 [=====] - 1s 1ms/step - loss: 7.5126 - mae: 2.0528 - val_loss: 13.3957 - val_mae: 2.2906

Epoch 19/100

405/405 [=====] - 1s 1ms/step - loss: 6.9238 - mae: 2.0088 - val_loss: 13.4070 - val_mae: 2.4391

Epoch 20/100

405/405 [=====] - 1s 1ms/step - loss: 6.8276 - mae: 2.0280 - val_loss: 12.3925 - val_mae: 2.3334

Epoch 21/100

405/405 [=====] - 1s 1ms/step - loss: 6.7970 - mae: 1.9785 - val_loss: 13.7717 - val_mae: 2.5497

Epoch 22/100

405/405 [=====] - 1s 1ms/step - loss: 6.6257 - mae: 1.9875 - val_loss: 12.3755 - val_mae: 2.3525

Epoch 23/100

405/405 [=====] - 1s 1ms/step - loss: 6.1608 - mae: 1.9382 - val_loss: 11.6934 - val_mae: 2.2718

Epoch 24/100

405/405 [=====] - 1s 1ms/step - loss: 6.0773 - mae: 1.9229 - val_loss: 13.6853 - val_mae: 2.4924

Epoch 25/100

405/405 [=====] - 1s 1ms/step - loss: 6.2665 - mae: 1.9089 - val_loss: 12.2459 - val_mae: 2.4337

Epoch 26/100

405/405 [=====] - 1s 1ms/step - loss: 5.9866 - mae: 1.8763 - val_loss: 11.4314 - val_mae: 2.2453

Epoch 27/100

405/405 [=====] - 1s 1ms/step - loss: 5.9049 - mae: 1.8171 - val_loss: 11.2346 - val_mae: 2.1282

Epoch 28/100

405/405 [=====] - 1s 1ms/step - loss: 6.1705 - mae: 1.8980 - val_loss: 10.8513 - val_mae: 2.1821

Epoch 29/100

405/405 [=====] - 1s 1ms/step - loss: 5.3383 - mae: 1.7492 - val_loss: 12.6256 - val_mae: 2.3391

Epoch 30/100

405/405 [=====] - 1s 1ms/step - loss: 5.5110 - mae: 1.8034 - val_loss: 12.6019 - val_mae: 2.5013

Epoch 31/100

405/405 [=====] - 1s 1ms/step - loss: 5.6895 - mae: 1.7772 - val_loss: 10.7513 - val_mae: 2.1376

Epoch 32/100

405/405 [=====] - 1s 1ms/step - loss: 5.3270 - mae: 1.7627 - val_loss: 11.8629 - val_mae: 2.2704

Epoch 33/100

405/405 [=====] - 1s 1ms/step - loss: 5.4483 - mae: 1.7627 - val_loss: 11.2606 - val_mae: 2.1566

Epoch 34/100

405/405 [=====] - 1s 1ms/step - loss: 5.0249 - mae: 1.7200 - val_loss: 10.9003 - val_mae: 2.1400

Epoch 35/100

405/405 [=====] - 1s 1ms/step - loss: 4.9463 - mae: 1.6451 - val_loss: 10.4056 - val_mae: 2.0512

Epoch 36/100

405/405 [=====] - 1s 2ms/step - loss: 4.5831 - mae: 1.6284 - val_loss: 10.2521 - val_mae: 2.1242

Epoch 37/100

405/405 [=====] - 1s 2ms/step - loss: 4.4603 - mae: 1.6002 - val_loss: 10.7677 - val_mae: 2.1620

Epoch 38/100

405/405 [=====] - 1s 2ms/step - loss: 4.4700 - mae: 1.6421 - val_loss: 10.7208 - val_mae: 2.1212

Epoch 39/100

405/405 [=====] - 1s 1ms/step - loss: 4.3356 - mae: 1.5848 - val_loss: 13.5537 - val_mae: 2.6014

Epoch 40/100

405/405 [=====] - 1s 1ms/step - loss: 4.5014 - mae: 1.6023 - val_loss: 10.7915 - val_mae: 2.0444

Epoch 41/100

405/405 [=====] - 1s 1ms/step - loss: 4.7389 - mae: 1.6668 - val_loss: 11.2216 - val_mae: 2.2934

Epoch 42/100

405/405 [=====] - 1s 1ms/step - loss: 4.3529 - mae: 1.5532 - val_loss: 13.6586 - val_mae: 2.5355

Epoch 43/100

405/405 [=====] - 1s 1ms/step - loss: 4.7267 - mae: 1.6801 - val_loss: 10.0980 - val_mae: 2.0130

Epoch 44/100

405/405 [=====] - 1s 1ms/step - loss: 4.0055 - mae: 1.5279 - val_loss: 11.0452 - val_mae: 2.1923

Epoch 45/100

405/405 [=====] - 1s 1ms/step - loss: 4.8441 - mae: 1.6669 - val_loss: 10.5468 - val_mae: 1.9689

Epoch 46/100

405/405 [=====] - 1s 1ms/step - loss: 4.2151 - mae: 1.5774 - val_loss: 10.9348 - val_mae: 2.2068

Epoch 47/100

405/405 [=====] - 1s 1ms/step - loss: 4.2922 - mae: 1.5638 - val_loss: 11.4779 - val_mae: 2.2920

Epoch 48/100

405/405 [=====] - 1s 1ms/step - loss: 3.6515 - mae: 1.4587 - val_loss: 10.5481 - val_mae: 2.1221

Epoch 49/100

405/405 [=====] - 1s 1ms/step - loss: 3.3489 - mae: 1.3714 - val_loss: 11.4757 - val_mae: 2.1562

Epoch 50/100

405/405 [=====] - 1s 1ms/step - loss: 4.0256 - mae: 1.5022 - val_loss: 11.6842 - val_mae: 2.1417

Epoch 51/100

405/405 [=====] - 1s 1ms/step - loss: 3.3893 - mae: 1.3968 - val_loss: 10.7627 - val_mae: 2.1765

Epoch 52/100

405/405 [=====] - 1s 1ms/step - loss: 3.8956 - mae: 1.4359 - val_loss: 10.8663 - val_mae: 2.2566

Epoch 53/100

405/405 [=====] - 1s 1ms/step - loss: 3.7817 - mae: 1.4491 - val_loss: 12.3847 - val_mae: 2.4174

Epoch 54/100

405/405 [=====] - 1s 1ms/step - loss: 3.6936 - mae: 1.4608 - val_loss: 10.8442 - val_mae: 2.0744

Epoch 55/100

405/405 [=====] - 1s 1ms/step - loss: 3.8502 - mae: 1.4222 - val_loss: 11.3007 - val_mae: 2.2329

Epoch 56/100

405/405 [=====] - 1s 1ms/step - loss: 3.7814 - mae: 1.4599 - val_loss: 11.2870 - val_mae: 2.3081

Epoch 57/100

405/405 [=====] - 1s 1ms/step - loss: 3.3710 - mae: 1.4046 - val_loss: 10.9141 - val_mae: 2.0836

Epoch 58/100

405/405 [=====] - 1s 1ms/step - loss: 3.6419 - mae: 1.4324 - val_loss: 11.0506 - val_mae: 2.3588

Epoch 59/100

405/405 [=====] - 1s 1ms/step - loss: 3.4211 - mae: 1.3690 - val_loss: 10.1135 - val_mae: 2.0520

Epoch 60/100

405/405 [=====] - 1s 1ms/step - loss: 3.4310 - mae: 1.4082 - val_loss: 11.8353 - val_mae: 2.1743

Epoch 61/100

405/405 [=====] - 1s 1ms/step - loss: 3.3805 - mae: 1.3818 - val_loss: 12.6161 - val_mae: 2.4017

Epoch 62/100

405/405 [=====] - 1s 1ms/step - loss: 3.2174 - mae: 1.3312 - val_loss: 10.0526 - val_mae: 2.2038

Epoch 63/100

405/405 [=====] - 1s 1ms/step - loss: 3.0071 - mae: 1.3008 - val_loss: 10.4478 - val_mae: 2.1736

Epoch 64/100

405/405 [=====] - 1s 1ms/step - loss: 3.0988 - mae: 1.3416 - val_loss: 10.3429 - val_mae: 2.0281

Epoch 65/100

405/405 [=====] - 1s 1ms/step - loss: 3.4175 - mae: 1.3802 - val_loss: 12.4171 - val_mae: 2.4823

Epoch 66/100

405/405 [=====] - 1s 1ms/step - loss: 3.3607 - mae: 1.3539 - val_loss: 10.6548 - val_mae: 2.1394

Epoch 67/100

405/405 [=====] - 1s 1ms/step - loss: 3.0866 - mae: 1.3118 - val_loss: 11.4328 - val_mae: 2.2975

Epoch 68/100

405/405 [=====] - 1s 1ms/step - loss: 3.0616 - mae: 1.2770 - val_loss: 13.8675 - val_mae: 2.4276

Epoch 69/100

405/405 [=====] - 1s 1ms/step - loss: 3.0182 - mae: 1.2975 - val_loss: 11.7308 - val_mae: 2.1988

Epoch 70/100

405/405 [=====] - 1s 1ms/step - loss: 3.0972 - mae: 1.3418 - val_loss: 10.3175 - val_mae: 2.1612

Epoch 71/100

405/405 [=====] - 1s 1ms/step - loss: 2.9455 - mae: 1.2727 - val_loss: 10.4521 - val_mae: 2.1601

Epoch 72/100

405/405 [=====] - 1s 1ms/step - loss: 2.7348 - mae: 1.2413 - val_loss: 9.8982 - val_mae: 2.0186

Epoch 73/100

405/405 [=====] - 1s 1ms/step - loss: 3.1025 - mae: 1.3526 - val_loss: 10.3144 - val_mae: 2.1408

Epoch 74/100

405/405 [=====] - 1s 1ms/step - loss: 2.9083 - mae: 1.2865 - val_loss: 11.1949 - val_mae: 2.2077

Epoch 75/100

405/405 [=====] - 1s 1ms/step - loss: 2.9698 - mae: 1.2642 - val_loss: 11.1746 - val_mae: 2.3780

Epoch 76/100

405/405 [=====] - 1s 1ms/step - loss: 2.6385 - mae: 1.2211 - val_loss: 11.3594 - val_mae: 2.2316

Epoch 77/100

405/405 [=====] - 1s 1ms/step - loss: 2.4980 - mae: 1.1867 - val_loss: 10.8848 - val_mae: 2.2254

Epoch 78/100

405/405 [=====] - 1s 1ms/step - loss: 2.5195 - mae: 1.1846 - val_loss: 10.7358 - val_mae: 2.1499

Epoch 79/100

405/405 [=====] - 1s 1ms/step - loss: 2.9154 - mae: 1.2707 - val_loss: 12.6326 - val_mae: 2.3690

Epoch 80/100

405/405 [=====] - 1s 1ms/step - loss: 2.5097 - mae: 1.1776 - val_loss: 11.4425 - val_mae: 2.3022

Epoch 81/100

405/405 [=====] - 1s 1ms/step - loss: 2.6592 - mae: 1.2059 - val_loss: 11.3584 - val_mae: 2.2487

Epoch 82/100

405/405 [=====] - 1s 1ms/step - loss: 2.6194 - mae: 1.2351 - val_loss: 10.3772 - val_mae: 2.1942

Epoch 83/100

405/405 [=====] - 1s 1ms/step - loss: 2.5899 - mae: 1.2102 - val_loss: 9.9613 - val_mae: 1.9894

Epoch 84/100

405/405 [=====] - 1s 1ms/step - loss: 2.9611 - mae: 1.2933 - val_loss: 11.4531 - val_mae: 2.2179

Epoch 85/100

405/405 [=====] - 1s 1ms/step - loss: 2.3555 - mae: 1.1617 - val_loss: 11.4880 - val_mae: 2.1919

Epoch 86/100

405/405 [=====] - 1s 1ms/step - loss: 3.0024 - mae: 1.2981 - val_loss: 11.3684 - val_mae: 2.2743

Epoch 87/100

405/405 [=====] - 1s 1ms/step - loss: 2.7357 - mae: 1.2221 - val_loss: 11.1201 - val_mae: 2.2172

Epoch 88/100

405/405 [=====] - 1s 1ms/step - loss: 2.4083 - mae: 1.1566 - val_loss: 11.0137 - val_mae: 2.2381

Epoch 89/100

405/405 [=====] - 1s 1ms/step - loss: 2.2354 - mae: 1.1169 - val_loss: 9.8410 - val_mae: 2.1405

Epoch 90/100

405/405 [=====] - 1s 1ms/step - loss: 2.6332 - mae: 1.2512 - val_loss: 10.4604 - val_mae: 2.0903

Epoch 91/100

405/405 [=====] - 1s 1ms/step - loss: 2.2639 - mae: 1.1138 - val_loss: 10.7505 - val_mae: 2.0973

Epoch 92/100

405/405 [=====] - 1s 1ms/step - loss: 2.7683 - mae: 1.2271 - val_loss: 9.6680 - val_mae: 2.0408

Epoch 93/100

405/405 [=====] - 1s 1ms/step - loss: 2.2436 - mae: 1.1091 - val_loss: 12.4192 - val_mae: 2.3987

Epoch 94/100

405/405 [=====] - 1s 1ms/step - loss: 2.2074 - mae: 1.0994 - val_loss: 10.4986 - val_mae: 2.2612

Epoch 95/100

405/405 [=====] - 1s 1ms/step - loss: 2.7602 - mae: 1.2375 - val_loss: 9.7729 - val_mae: 2.0761

Epoch 96/100

405/405 [=====] - 1s 1ms/step - loss: 2.5559 - mae: 1.1601 - val_loss: 9.4099 - val_mae: 1.9973

Epoch 97/100

405/405 [=====] - 1s 1ms/step - loss: 2.0664 - mae: 1.0761 - val_loss: 9.8013 - val_mae: 2.0054

Epoch 98/100

405/405 [=====] - 1s 1ms/step - loss: 2.3195 - mae: 1.1356 - val_loss: 10.8694 - val_mae: 2.1843

Epoch 99/100

405/405 [=====] - 1s 1ms/step - loss: 2.3759 - mae: 1.1641 - val_loss: 10.6165 - val_mae: 2.1379

Epoch 100/100

405/405 [=====] - 1s 1ms/step - loss: 2.4255 - mae: 1.1388 - val_loss: 10.1201 - val_mae: 2.1865

[0.0494602 -0.49954771 1.03376078 -0.27771867 -0.20575443 -0.12267537
0.79433239 -0.33420999 1.70971379 1.58047402 0.84593377 0.42676661
0.05938496]

Actual Output : 19.6

1/1 [=====] - 0s 60ms/step

Predicted Output : [[22.452795]]

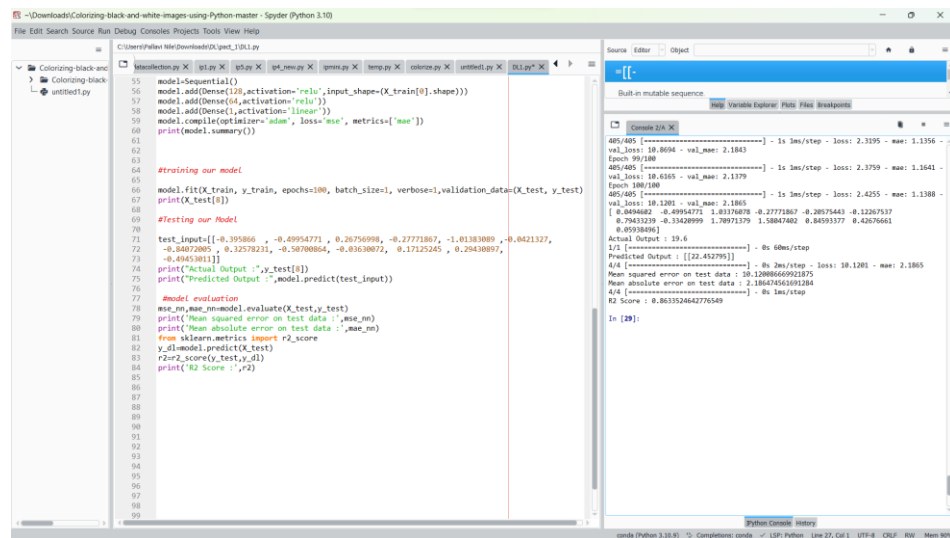
4/4 [=====] - 0s 2ms/step - loss: 10.1201 - mae: 2.1865

Mean squared error on test data : 10.120086669921875

Mean absolute error on test data : 2.186474561691284

4/4 [=====] - 0s 1ms/step

R2 Score : 0.8633524642776549



```
118 model.compile(optimizer='adam', loss='mse', metrics=['mae'])
119 print(model.summary())
120
121 #training our model
122 model.fit(X_train, y_train, epochs=100, batch_size=1, verbose=1, validation_data=(X_test, y_test))
123 print(X_test[8])
124
125 #testing our Model
126 test_input=[-0.395866, -0.49954771, 0.2075099, -0.27771867, -1.01303089, -0.8421327,
127 -0.8407205, 0.32578231, -0.5070064, -0.83630872, 0.17125245, 0.29430807,
128 -0.40453011]
129 print("Actual Output :",y_test[8])
130 print("Predicted Output :",model.predict(test_input))
131
132 #model evaluation
133 mse,m,mae,mn=model.evaluate(X_test,y_test)
134 print("Mean squared error on test data :",mse,mn)
135 print("Mean absolute error on test data :",mae,mn)
136 from sklearn.metrics import r2_score
137 y_d1=model.predict(X_test)
138 r2=r2_score(y_test,y_d1)
139 print('R2 Score :',r2)
```

405/405 [=====] - 1s 1ms/step - loss: 2.3195 - mae: 1.1356 - val_loss: 10.8694 - val_mae: 2.1843
Epoch 99/100
405/405 [=====] - 1s 1ms/step - loss: 2.3759 - mae: 1.1641 - val_loss: 10.6165 - val_mae: 2.1379
Epoch 100/100
405/405 [=====] - 1s 1ms/step - loss: 2.4255 - mae: 1.1388 - val_loss: 10.1201 - val_mae: 2.1865
[0.0494602 -0.49954771 1.03376078 -0.27771867 -0.20575443 -0.12267537
0.79433239 -0.33420999 1.70971379 1.58047402 0.84593377 0.42676661
0.05938496]
Actual Output : 19.6
1/1 [=====] - 0s 60ms/step
Predicted Output : [[22.452795]]
4/4 [=====] - 0s 2ms/step - loss: 10.1201 - mae: 2.1865
Mean squared error on test data : 10.120086669921875
Mean absolute error on test data : 2.186474561691284
4/4 [=====] - 0s 1ms/step
R2 Score : 0.8633524642776549

```
~/Downloads/Colorizing-black-and-white-images-using-Python-master - Spyder (Python 3.10)
File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Pallavi Nile\Downloads\DL\pact_1\DL1.py

# Shuffle the indices of the data randomly
shuffled_indices = np.random.permutation(len(df))

# Split the indices into training and test sets
test_indices = shuffled_indices[num_test_samples:]
train_indices = shuffled_indices[:num_test_samples]

# Split the features and target variable based on the indices
X_train = X[train_indices]
X_test = X[test_indices]
y_train = y[train_indices]
y_test = y[test_indices]

# Data preprocessing
mean_X_train = X_train.mean(axis=0)
std_X_train = X_train.std(axis=0)
X_train = (X_train - mean_X_train) / std_X_train
mean_X_test = X_test.mean(axis=0)
std_X_test = X_test.std(axis=0)
X_test = (X_test - mean_X_test) / std_X_test

# Building model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(X_train[0].shape)))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
print(model.summary())

# Training our model
model.fit(X_train, y_train, epochs=100, batch_size=1, verbose=1, validation_data=(X_test, y_test))
```

Console Output:

```
Epoch 1/100
485/485 [-----] - 1s 2ms/step - loss: 131.9694 - mae: 7.6868 - val_loss: 23.2772 - val_mae: 3.1643
Epoch 2/100
485/485 [-----] - 1s 1ms/step - loss: 20.3213 - mae: 3.0670 - val_loss: 18.0007 - val_mae: 2.8652
Epoch 3/100
485/485 [-----] - 1s 1ms/step - loss: 16.0820 - mae: 2.8830 - val_loss: 15.3380 - val_mae: 2.5316
Epoch 4/100
485/485 [-----] - 1s 1ms/step - loss: 14.1997 - mae: 2.6935 - val_loss: 13.7442 - val_mae: 2.4574
Epoch 5/100
485/485 [-----] - 1s 1ms/step - loss: 13.6563 - mae: 2.6144 - val_loss: 13.6630 - val_mae: 2.3891
Epoch 6/100
485/485 [-----] - 1s 1ms/step - loss: 11.9183 - mae: 2.4655 - val_loss: 12.9316 - val_mae: 2.3729
Epoch 7/100
485/485 [-----] - 1s 1ms/step - loss: 12.1821 - mae: 2.5067 - val_loss: 13.0030 - val_mae: 2.4284
Epoch 8/100
485/485 [-----] - 1s 1ms/step - loss: 11.1841 - mae: 2.4717 - val_loss: 14.4389 - val_mae: 2.6895
Epoch 9/100
485/485 [-----] - 1s 1ms/step - loss: 10.7131 - mae: 2.3939 - val_loss: 11.9493 - val_mae: 2.2783
Epoch 10/100
485/485 [-----] - 1s 1ms/step - loss: 9.7527 - mae: 2.3094 - val_loss: 12.7318 - val_mae: 2.4541
Epoch 11/100
485/485 [-----] - 1s 1ms/step - loss: 9.6841 - mae: 2.3094 - val_loss: 13.2768 - val_mae: 2.5256
Epoch 12/100
```

```
~/Downloads/Colorizing-black-and-white-images-using-Python-master - Spyder (Python 3.10)
File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Pallavi Nile\Downloads\DL\pact_1\DL1.py

# While Pallavi Nile Roll NO: 4217 Div:B
# DL Practical NO.1

import pandas as pd
import numpy as np

df = pd.read_csv('C:/Users/Pallavi Nile/Downloads/DL/pact_1/Hoston.csv')
print(df)

# Print the shape of X
X = df.drop('medv', axis=1).values
print(X.shape)

# Print the shape of y
y = df['medv'].values
print(y.shape)

# Set the random seed for reproducibility
np.random.seed(42)

# Split the data into training and testing sets
test_size = 0.2 # Percentage of data to be used for testing
num_test_samples = int(test_size * len(df))

# Shuffle the indices of the data randomly
shuffled_indices = np.random.permutation(len(df))

# Split the indices into training and test sets
test_indices = shuffled_indices[num_test_samples:]
train_indices = shuffled_indices[:num_test_samples]

# Split the features and target variable based on the indices
X_train = X[train_indices]
X_test = X[test_indices]
y_train = y[train_indices]
y_test = y[test_indices]

# Data preprocessing
mean_X_train = X_train.mean(axis=0)
```

Console Output:

```
In [28]: runfile('C:/Users/Pallavi Nile/Downloads/DL/pact_1/DL1.py', wdir='C:/Users/Pallavi Nile/Downloads/DL/pact_1')
Out[28]:
[[0.00632 18.0 2.31 0.538 ... 296 15.3 396.90 4.98 24.0
0.00718 0.0 7.07 0.469 ... 242 17.8 396.90 5.16 21.6
0.02729 0.0 7.07 0.469 ... 242 17.8 392.83 4.03 34.7
0.00217 0.0 2.18 0.458 ... 222 18.7 394.63 2.94 31.4
0.00908 0.0 2.18 0.458 ... 222 18.7 396.90 5.33 36.2
...
501 0.06263 0.0 11.93 0.573 ... 273 21.0 391.99 9.67 22.4
502 0.04527 0.0 11.93 0.573 ... 273 21.0 395.90 9.00 20.6
503 0.06076 0.0 11.93 0.573 ... 273 21.0 396.90 1.64 23.9
504 0.10959 0.0 11.93 0.573 ... 273 21.0 393.45 6.48 22.0
505 0.04741 0.0 11.93 0.573 ... 273 21.0 396.90 7.88 11.9]]

[506 rows x 14 columns]
Shape of X: (506, 13)
Shape of y: (506,)
Shape of X_train: (405, 13)
Shape of X_test: (101, 13)
Shape of y_train: (405,)
Shape of y_test: (101,)
[0.395866 -0.49954771 0.26756998 -0.27771867 -1.81383089 -0.8421327
-0.58873085 0.32578231 -0.50708064 -0.83630072 0.17125245 0.29430897
-0.49453011]
21.4
Model: "sequential_18"
Layer (type) Output Shape Param #
-----
dense_28 (Dense) (None, 128) 1792
dense_29 (Dense) (None, 64) 8256
dense_30 (Dense) (None, 1) 65
Total params: 10,113
Trainable params: 10,113
Non-trainable params: 0
```