Srushti Ramesh Bhamre BC56

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.

In [1]:

```
import io
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import mean_absolute_error, r2_score
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
# Importing DataSet and take a look at Data
data = pd.read_csv('housing_data - housing_data.csv')
data
```

Out[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	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	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	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	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	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	NaN	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	NaN	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	NaN	2.5050	1	273	21.0	396.90	7.88	11.9

506 rows × 14 columns

In [3]:

```
# Handle null values by filling them with the mean of the respective columns
data.fillna(data.mean(), inplace=True)
```

In [4]:

```
data.isnull().sum()
```

Out[4]:

```
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
```

In [5]:
```python
data.describe()
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRA
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00
mean	3.611874	11.211934	11.083992	0.069959	0.554695	6.284634	68.518519	3.795043	9.549407	408.237154	18.45
std	8.545770	22.921051	6.699165	0.250233	0.115878	0.702617	27.439466	2.105710	8.707259	168.537116	2.16
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.60
25%	0.083235	0.000000	5.190000	0.000000	0.449000	5.885500	45.925000	2.100175	4.000000	279.000000	17.40
50%	0.290250	0.000000	9.900000	0.000000	0.538000	6.208500	74.450000	3.207450	5.000000	330.000000	19.05
75%	3.611874	11.211934	18.100000	0.000000	0.624000	6.623500	93.575000	5.188425	24.000000	666.000000	20.20
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.00

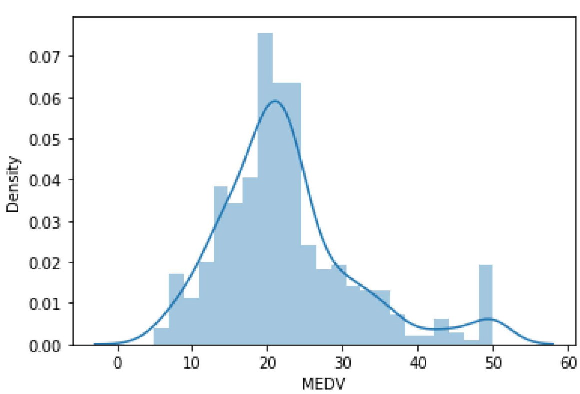
In [6]:
```python
data.info()
data.shape
```

```
<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    float64
 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(12), int64(2)
memory usage: 55.5 KB
```

Out[6]: (506, 14)

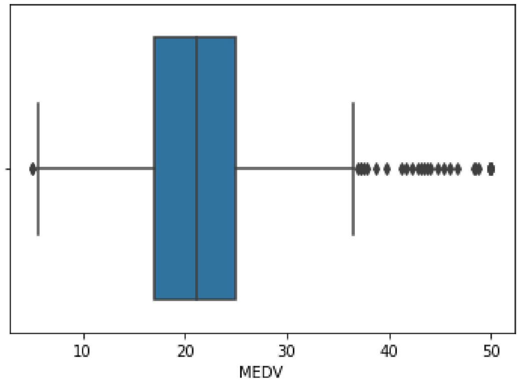
In [7]:
```python
import seaborn as sns
sns.distplot(data.MEDV)
```

Out[7]: <AxesSubplot:xlabel='MEDV', ylabel='Density'>



In [8]:
```python
sns.boxplot(data.MEDV)
```

Out[8]: <AxesSubplot:xlabel='MEDV'>



In [9]:
```python
correlation = data.corr()
correlation.loc['MEDV']
```

Out[9]:
```
CRIM      -0.379695
ZN         0.365943
INDUS     -0.478657
CHAS       0.179882
NOX       -0.427321
RM         0.695360
AGE       -0.380223
DIS        0.249929
RAD       -0.381626
TAX       -0.468536
PTRATIO   -0.507787
B          0.333461
LSTAT     -0.721975
MEDV       1.000000
Name: MEDV, dtype: float64
```

In [10]:

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

Out[10]: <AxesSubplot:>

In [11]:

```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 = data[col]
  y = data.MEDV
  plt.scatter(x, y, marker='o')
  plt.title("Variation in House prices")
  plt.xlabel(col)
  plt.ylabel('"House prices in $1000"')
```

In [12]:
```python
# Splitting the dependent feature and independent feature
#X = data[['LSTAT','RM','PTRATIO']]
X = data.iloc[:,:-1]
y= data.MEDV
```

In [13]:
```python
import numpy as np
from sklearn.model_selection import train_test_split

# Assuming you have data stored in some variables X and y
# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Now you can proceed with the code you provided
# Importing necessary libraries
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

# Scaling the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

mean = X_train.mean(axis=0)
std = X_train.std(axis=0)
X_train = (X_train - mean) / std
X_test = (X_test - mean) / std
#Linear Regression

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
#Fitting the model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
regressor.fit(X_train,y_train)
```

Out[13]:
```
▾ LinearRegression
LinearRegression()
```

In [14]:
```python
#Prediction on the test dataset
y_pred = regressor.predict(X_test)
# 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)
```

```
5.001766890194158
```

In [15]:
```python
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2)
```

```
0.6588520195508162
```

In [16]:

```python
import keras
from keras.layers import Dense
from keras.models import Sequential
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Assuming X_train and X_test are defined and initialized previously
# Assuming y_train is also defined and initialized

# Scaling the dataset
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Creating the neural network model
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))

# Compiling the model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

# Visualizing the model architecture
keras.utils.plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=True)

# Assuming you have defined your training data X_train and y_train
history = model.fit(X_train, y_train, epochs=100, validation_split=0.05)

# Plotting the training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

```

You must install pydot (`pip install pydot`) and install graphviz (see instructions at https://graphviz.gitlab.io/download/) (https://graphviz.gitlab.io/download/)) for plot_model to work.

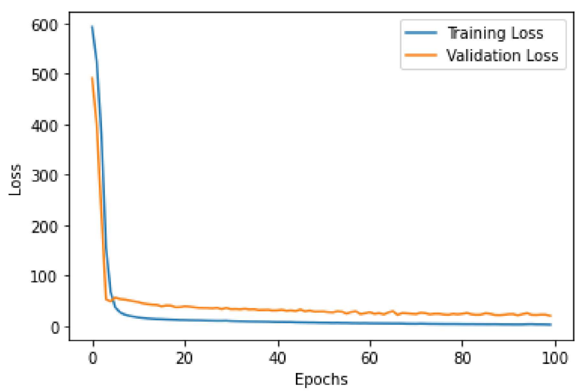
```
Epoch 1/100
12/12 [==============================] - 2s 44ms/step - loss: 593.3365 - mae: 22.5031 - val_loss: 492.1255 - val_mae: 20.7200
Epoch 2/100
12/12 [==============================] - 0s 9ms/step - loss: 525.0057 - mae: 20.9747 - val_loss: 399.8764 - val_mae: 18.5050
Epoch 3/100
12/12 [==============================] - 0s 8ms/step - loss: 378.6923 - mae: 17.3772 - val_loss: 222.7657 - val_mae: 13.2538
Epoch 4/100
12/12 [==============================] - 0s 10ms/step - loss: 155.3056 - mae: 10.3306 - val_loss: 52.9837 - val_mae: 4.9532
Epoch 5/100
12/12 [==============================] - 0s 9ms/step - loss: 67.5092 - mae: 6.4644 - val_loss: 48.6647 - val_mae: 4.4646
Epoch 6/100
12/12 [==============================] - 0s 9ms/step - loss: 37.0765 - mae: 4.5780 - val_loss: 56.6724 - val_mae: 4.6085
Epoch 7/100
12/12 [==============================] - 0s 9ms/step - loss: 27.4871 - mae: 3.8358 - val_loss: 53.4959 - val_mae: 4.6053
Epoch 8/100
12/12 [==============================] - 0s 9ms/step - loss: 22.7617 - mae: 3.5466 - val_loss: 52.5010 - val_mae: 4.6640
Epoch 9/100
12/12 [==============================] - 0s 9ms/step - loss: 20.1987 - mae: 3.2901 - val_loss: 50.7301 - val_mae: 4.5293
Epoch 10/100
12/12 [==============================] - 0s 10ms/step - loss: 18.5831 - mae: 3.1421 - val_loss: 49.0089 - val_mae: 4.3896
Epoch 11/100
12/12 [==============================] - 0s 12ms/step - loss: 16.9307 - mae: 2.9703 - val_loss: 47.4259 - val_mae: 4.2535
Epoch 12/100
12/12 [==============================] - 0s 9ms/step - loss: 15.8584 - mae: 2.8793 - val_loss: 44.7945 - val_mae: 4.1555
Epoch 13/100
12/12 [==============================] - 0s 10ms/step - loss: 14.9110 - mae: 2.7769 - val_loss: 43.3636 - val_mae: 4.0691
Epoch 14/100
12/12 [==============================] - 0s 10ms/step - loss: 14.2704 - mae: 2.7415 - val_loss: 42.3949 - val_mae: 4.0346
Epoch 15/100
12/12 [==============================] - 0s 9ms/step - loss: 13.6585 - mae: 2.6691 - val_loss: 41.9805 - val_mae: 3.9858
Epoch 16/100
12/12 [==============================] - 0s 7ms/step - loss: 13.3851 - mae: 2.6387 - val_loss: 39.0053 - val_mae: 3.8408
Epoch 17/100
12/12 [==============================] - 0s 11ms/step - loss: 12.9651 - mae: 2.6520 - val_loss: 40.7240 - val_mae: 3.9047
Epoch 18/100
12/12 [==============================] - 0s 11ms/step - loss: 12.5664 - mae: 2.5557 - val_loss: 40.6434 - val_mae: 3.9262
Epoch 19/100
12/12 [==============================] - 0s 10ms/step - loss: 12.2979 - mae: 2.5747 - val_loss: 37.2956 - val_mae: 3.7524
Epoch 20/100
12/12 [==============================] - 0s 7ms/step - loss: 11.8418 - mae: 2.4917 - val_loss: 37.6266 - val_mae: 3.7707
Epoch 21/100
12/12 [==============================] - 0s 10ms/step - loss: 11.7158 - mae: 2.5176 - val_loss: 39.0760 - val_mae: 3.8018
Epoch 22/100
12/12 [==============================] - 0s 9ms/step - loss: 11.4431 - mae: 2.4458 - val_loss: 38.5427 - val_mae: 3.6912
Epoch 23/100
12/12 [==============================] - 0s 10ms/step - loss: 11.2924 - mae: 2.5149 - val_loss: 37.3225 - val_mae: 3.6865
Epoch 24/100
12/12 [==============================] - 0s 9ms/step - loss: 11.1942 - mae: 2.4082 - val_loss: 36.1898 - val_mae: 3.6670
Epoch 25/100
12/12 [==============================] - 0s 10ms/step - loss: 10.9826 - mae: 2.4827 - val_loss: 35.9758 - va
```

```
l_mae: 3.6580
Epoch 26/100
12/12 [==============================] - 0s 6ms/step - loss: 10.6356 - mae: 2.3610 - val_loss: 35.8442 - val
_mae: 3.6747
Epoch 27/100
12/12 [==============================] - 0s 8ms/step - loss: 10.5184 - mae: 2.4071 - val_loss: 35.2805 - val
_mae: 3.5572
Epoch 28/100
12/12 [==============================] - 0s 8ms/step - loss: 10.3016 - mae: 2.3593 - val_loss: 36.2390 - val
_mae: 3.6871
Epoch 29/100
12/12 [==============================] - 0s 9ms/step - loss: 10.3144 - mae: 2.3381 - val_loss: 33.6862 - val
_mae: 3.6116
Epoch 30/100
12/12 [==============================] - 0s 7ms/step - loss: 10.5037 - mae: 2.4386 - val_loss: 36.0907 - val
_mae: 3.6182
Epoch 31/100
12/12 [==============================] - 0s 10ms/step - loss: 9.8025 - mae: 2.3110 - val_loss: 33.4866 - val
_mae: 3.6011
Epoch 32/100
12/12 [==============================] - 0s 10ms/step - loss: 9.5271 - mae: 2.2879 - val_loss: 33.7885 - val
_mae: 3.5193
Epoch 33/100
12/12 [==============================] - 0s 9ms/step - loss: 9.3183 - mae: 2.2621 - val_loss: 33.1790 - val_
mae: 3.5449
Epoch 34/100
12/12 [==============================] - 0s 9ms/step - loss: 9.1131 - mae: 2.2614 - val_loss: 34.3521 - val_
mae: 3.6106
Epoch 35/100
12/12 [==============================] - 0s 9ms/step - loss: 8.8850 - mae: 2.2105 - val_loss: 32.9580 - val_
mae: 3.5442
Epoch 36/100
12/12 [==============================] - 0s 9ms/step - loss: 8.7797 - mae: 2.2195 - val_loss: 33.3213 - val_
mae: 3.5653
Epoch 37/100
12/12 [==============================] - 0s 10ms/step - loss: 8.6799 - mae: 2.2248 - val_loss: 31.7044 - val
_mae: 3.4381
Epoch 38/100
12/12 [==============================] - 0s 9ms/step - loss: 8.5635 - mae: 2.1858 - val_loss: 31.8526 - val_
mae: 3.5358
Epoch 39/100
12/12 [==============================] - 0s 10ms/step - loss: 8.4396 - mae: 2.1538 - val_loss: 32.1248 - val
_mae: 3.5076
Epoch 40/100
12/12 [==============================] - 0s 11ms/step - loss: 8.1019 - mae: 2.1395 - val_loss: 30.8459 - val
_mae: 3.4163
Epoch 41/100
12/12 [==============================] - 0s 11ms/step - loss: 7.9169 - mae: 2.0986 - val_loss: 31.0522 - val
_mae: 3.4657
Epoch 42/100
12/12 [==============================] - 0s 11ms/step - loss: 7.8935 - mae: 2.1001 - val_loss: 32.1896 - val
_mae: 3.5412
Epoch 43/100
12/12 [==============================] - 0s 11ms/step - loss: 7.7575 - mae: 2.1354 - val_loss: 30.1514 - val
_mae: 3.3762
Epoch 44/100
12/12 [==============================] - 0s 8ms/step - loss: 7.7905 - mae: 2.1055 - val_loss: 31.0291 - val_
mae: 3.4593
Epoch 45/100
12/12 [==============================] - 0s 9ms/step - loss: 7.3119 - mae: 2.0280 - val_loss: 29.6046 - val_
mae: 3.4678
Epoch 46/100
12/12 [==============================] - 0s 8ms/step - loss: 7.1641 - mae: 2.0162 - val_loss: 32.8597 - val_
mae: 3.5630
Epoch 47/100
12/12 [==============================] - 0s 8ms/step - loss: 7.0373 - mae: 1.9754 - val_loss: 29.0646 - val_
mae: 3.3674
Epoch 48/100
12/12 [==============================] - 0s 10ms/step - loss: 6.8431 - mae: 1.9862 - val_loss: 30.8615 - val
_mae: 3.4752
Epoch 49/100
12/12 [==============================] - 0s 8ms/step - loss: 6.8121 - mae: 1.9738 - val_loss: 28.5740 - val_
mae: 3.3669
Epoch 50/100
12/12 [==============================] - 0s 9ms/step - loss: 6.5312 - mae: 1.9086 - val_loss: 28.7225 - val_
mae: 3.3653
```

```
Epoch 51/100
12/12 [==============================] - 0s 9ms/step - loss: 6.4348 - mae: 1.9185 - val_loss: 28.8222 - val_
mae: 3.3936
Epoch 52/100
12/12 [==============================] - 0s 10ms/step - loss: 6.3505 - mae: 1.9192 - val_loss: 27.5490 - val
_mae: 3.3383
Epoch 53/100
12/12 [==============================] - 0s 9ms/step - loss: 6.0513 - mae: 1.8575 - val_loss: 27.0754 - val_
mae: 3.3404
Epoch 54/100
12/12 [==============================] - 0s 8ms/step - loss: 6.2330 - mae: 1.8712 - val_loss: 29.2607 - val_
mae: 3.4202
Epoch 55/100
12/12 [==============================] - 0s 9ms/step - loss: 5.9660 - mae: 1.8637 - val_loss: 28.4836 - val_
mae: 3.4191
Epoch 56/100
12/12 [==============================] - 0s 8ms/step - loss: 5.9944 - mae: 1.8457 - val_loss: 24.6683 - val_
mae: 3.2299
Epoch 57/100
12/12 [==============================] - 0s 9ms/step - loss: 5.6571 - mae: 1.8017 - val_loss: 27.9240 - val_
mae: 3.4076
Epoch 58/100
12/12 [==============================] - 0s 10ms/step - loss: 5.6048 - mae: 1.8299 - val_loss: 29.2053 - val
_mae: 3.4728
Epoch 59/100
12/12 [==============================] - 0s 9ms/step - loss: 5.4385 - mae: 1.7720 - val_loss: 23.6976 - val_
mae: 3.1610
Epoch 60/100
12/12 [==============================] - 0s 9ms/step - loss: 5.5935 - mae: 1.8099 - val_loss: 25.4217 - val_
mae: 3.3188
Epoch 61/100
```

```
12/12 [==============================] - 0s 9ms/step - loss: 5.3113 - mae: 1.7609 - val_loss: 27.5410 - val_
mae: 3.4120
Epoch 62/100
12/12 [==============================] - 0s 8ms/step - loss: 5.2426 - mae: 1.7450 - val_loss: 24.2072 - val_
mae: 3.1952
Epoch 63/100
12/12 [==============================] - 0s 8ms/step - loss: 5.2041 - mae: 1.7553 - val_loss: 25.6998 - val_
mae: 3.3223
Epoch 64/100
12/12 [==============================] - 0s 9ms/step - loss: 5.0989 - mae: 1.7075 - val_loss: 23.0396 - val_
mae: 3.1827
Epoch 65/100
12/12 [==============================] - 0s 9ms/step - loss: 5.1073 - mae: 1.7285 - val_loss: 26.8679 - val_
mae: 3.3379
Epoch 66/100
12/12 [==============================] - 0s 8ms/step - loss: 4.9866 - mae: 1.7151 - val_loss: 29.6811 - val_
mae: 3.4808
Epoch 67/100
12/12 [==============================] - 0s 8ms/step - loss: 5.0088 - mae: 1.7180 - val_loss: 22.0234 - val_
mae: 3.0447
Epoch 68/100
12/12 [==============================] - 0s 9ms/step - loss: 4.8893 - mae: 1.6920 - val_loss: 25.9152 - val_
mae: 3.2718
Epoch 69/100
12/12 [==============================] - 0s 9ms/step - loss: 4.5496 - mae: 1.6437 - val_loss: 25.2454 - val_
mae: 3.2645
Epoch 70/100
12/12 [==============================] - 0s 7ms/step - loss: 4.4986 - mae: 1.6102 - val_loss: 24.4274 - val_
mae: 3.2512
Epoch 71/100
12/12 [==============================] - 0s 7ms/step - loss: 4.3851 - mae: 1.5780 - val_loss: 23.7263 - val_
mae: 3.1603
Epoch 72/100
12/12 [==============================] - 0s 9ms/step - loss: 4.7020 - mae: 1.6812 - val_loss: 26.5144 - val_
mae: 3.3664
Epoch 73/100
12/12 [==============================] - 0s 10ms/step - loss: 4.4619 - mae: 1.6281 - val_loss: 25.6483 - val
_mae: 3.2771
Epoch 74/100
12/12 [==============================] - 0s 8ms/step - loss: 4.2446 - mae: 1.5621 - val_loss: 23.1437 - val_
mae: 3.1621
Epoch 75/100
12/12 [==============================] - 0s 7ms/step - loss: 4.1941 - mae: 1.5483 - val_loss: 24.4475 - val_
mae: 3.2213
Epoch 76/100
12/12 [==============================] - 0s 8ms/step - loss: 4.1007 - mae: 1.5233 - val_loss: 24.4011 - val_
mae: 3.2016
Epoch 77/100
12/12 [==============================] - 0s 7ms/step - loss: 3.9885 - mae: 1.5270 - val_loss: 22.9175 - val_
mae: 3.1584
Epoch 78/100
12/12 [==============================] - 0s 9ms/step - loss: 3.9726 - mae: 1.4958 - val_loss: 22.3622 - val_
mae: 3.0758
Epoch 79/100
12/12 [==============================] - 0s 8ms/step - loss: 3.9568 - mae: 1.5301 - val_loss: 24.2059 - val_
mae: 3.2229
Epoch 80/100
12/12 [==============================] - 0s 9ms/step - loss: 3.9104 - mae: 1.4846 - val_loss: 23.2432 - val_
mae: 3.2169
Epoch 81/100
12/12 [==============================] - 0s 7ms/step - loss: 3.7377 - mae: 1.4833 - val_loss: 24.3587 - val_
mae: 3.2552
Epoch 82/100
12/12 [==============================] - 0s 8ms/step - loss: 3.6685 - mae: 1.4626 - val_loss: 26.0521 - val_
mae: 3.2781
Epoch 83/100
12/12 [==============================] - 0s 7ms/step - loss: 3.7127 - mae: 1.4596 - val_loss: 23.1975 - val_
mae: 3.1065
Epoch 84/100
12/12 [==============================] - 0s 8ms/step - loss: 3.6200 - mae: 1.4455 - val_loss: 22.2413 - val_
mae: 3.0606
Epoch 85/100
12/12 [==============================] - 0s 7ms/step - loss: 3.5957 - mae: 1.4313 - val_loss: 22.5648 - val_
mae: 3.1206
Epoch 86/100
12/12 [==============================] - 0s 9ms/step - loss: 3.5110 - mae: 1.3960 - val_loss: 25.4949 - val_
```

```
mae: 3.2908
Epoch 87/100
12/12 [==============================] - 0s 7ms/step - loss: 3.4958 - mae: 1.4208 - val_loss: 24.3938 - val_
mae: 3.1356
Epoch 88/100
12/12 [==============================] - 0s 8ms/step - loss: 3.5582 - mae: 1.4469 - val_loss: 21.8915 - val_
mae: 3.0376
Epoch 89/100
12/12 [==============================] - 0s 8ms/step - loss: 3.4020 - mae: 1.3666 - val_loss: 21.4599 - val_
mae: 2.9952
Epoch 90/100
12/12 [==============================] - 0s 8ms/step - loss: 3.2326 - mae: 1.3574 - val_loss: 22.2129 - val_
mae: 3.1246
Epoch 91/100
12/12 [==============================] - 0s 7ms/step - loss: 3.2493 - mae: 1.3628 - val_loss: 23.5151 - val_
mae: 3.1295
Epoch 92/100
12/12 [==============================] - 0s 7ms/step - loss: 3.1388 - mae: 1.3269 - val_loss: 23.6589 - val_
mae: 3.0611
Epoch 93/100
12/12 [==============================] - 0s 7ms/step - loss: 3.1233 - mae: 1.3342 - val_loss: 21.1233 - val_
mae: 2.9461
Epoch 94/100
12/12 [==============================] - 0s 8ms/step - loss: 3.1661 - mae: 1.3309 - val_loss: 23.6154 - val_
mae: 3.1297
Epoch 95/100
12/12 [==============================] - 0s 8ms/step - loss: 3.5468 - mae: 1.4289 - val_loss: 25.6488 - val_
mae: 3.1956
Epoch 96/100
12/12 [==============================] - 0s 9ms/step - loss: 3.7170 - mae: 1.4744 - val_loss: 22.1357 - val_
mae: 3.1080
Epoch 97/100
12/12 [==============================] - 0s 8ms/step - loss: 3.2998 - mae: 1.3654 - val_loss: 21.8552 - val_
mae: 3.0664
Epoch 98/100
12/12 [==============================] - 0s 7ms/step - loss: 3.2627 - mae: 1.3816 - val_loss: 22.6286 - val_
mae: 3.0797
Epoch 99/100
12/12 [==============================] - 0s 9ms/step - loss: 3.0908 - mae: 1.3260 - val_loss: 22.4770 - val_
mae: 3.0708
Epoch 100/100
12/12 [==============================] - 0s 9ms/step - loss: 2.8196 - mae: 1.2830 - val_loss: 20.1691 - val_
mae: 2.8952
```



In [17]:

```
#Evaluation of the model
y_pred = model.predict(X_test)
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)
```

```
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 6ms/step - loss: 11.8925 - mae: 2.2676
Mean squared error on test data:  11.892463684082031
Mean absolute error on test data:  2.2675862312316895
```

In [18]:

```
#Comparison with traditional approaches
#First let's try with a simple algorithm, the Linear Regression:
from sklearn.metrics import mean_absolute_error
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)
mse_lr = mean_squared_error(y_test, y_pred_lr)
mae_lr = mean_absolute_error(y_test, y_pred_lr)
print('Mean squared error on test data: ', mse_lr)
print('Mean absolute error on test data: ', mae_lr)
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2)
```

```
Mean squared error on test data:  25.01767202384286
Mean absolute error on test data:  3.1499233573458034
0.8378310222849991
```

In [19]:

```
# 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)
```

```
3.448545294335442
```

In [20]:

```
# Make predictions on new data
import sklearn
new_data = scaler.transform([[0.1, 10.0, 5.0, 0, 0.4, 6.0, 50, 6.0, 1, 400, 20, 300, 10]])  # Scaling new
prediction = model.predict(new_data)
print("Predicted house price:", prediction)
```

```
1/1 [==============================] - 0s 246ms/step
Predicted house price: [[14.101335]]
```