dl1

April 1, 2025

[2]:
```python
import io
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```python
%matplotlib inline
```

[4]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, r2_score
```

[5]:
```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

[6]:
```python
import warnings
warnings.filterwarnings('ignore')
```

[7]:
```python
data=pd.read_csv('housing_data.csv')
```

[8]:
```python
data
```

[8]:
```
        CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  \
0    0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900    1  296
1    0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622    3  222
..       …     …      …     …      …      …     …       …   …    …
501  0.06263   0.0  11.93   0.0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93   0.0  0.573  6.120  76.7  2.2875    1  273
503  0.06076   0.0  11.93   0.0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93   0.0  0.573  6.794  89.3  2.3889    1  273
505  0.04741   0.0  11.93   0.0  0.573  6.030   NaN  2.5050    1  273

     PTRATIO       B  LSTAT  MEDV
```

```
0      15.3  396.90   4.98  24.0
1      17.8  396.90   9.14  21.6
2      17.8  392.83   4.03  34.7
3      18.7  394.63   2.94  33.4
4      18.7  396.90    NaN  36.2
..      …      …     …    …
501    21.0  391.99    NaN  22.4
502    21.0  396.90   9.08  20.6
503    21.0  396.90   5.64  23.9
504    21.0  393.45   6.48  22.0
505    21.0  396.90   7.88  11.9

[506 rows x 14 columns]
```

[9]:
```python
data.isnull().sum()
```

[9]:
```
CRIM       20
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
MEDV        0
dtype: int64
```

[10]:
```python
# Handle null values by filling them with the mean of the respective columns
data.fillna(data.mean(), inplace=True)
```

[11]:
```python
data.isnull().sum()
```

[11]:
```
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
```

[12]:
```
data.describe()
```

[12]:

	CRIM	ZN	INDUS	CHAS	NOX	RM
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.611874	11.211934	11.083992	0.069959	0.554695	6.284634
std	8.545770	22.921051	6.699165	0.250233	0.115878	0.702617
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000
25%	0.083235	0.000000	5.190000	0.000000	0.449000	5.885500
50%	0.290250	0.000000	9.900000	0.000000	0.538000	6.208500
75%	3.611874	11.211934	18.100000	0.000000	0.624000	6.623500
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000

	AGE	DIS	RAD	TAX	PTRATIO	B
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	68.518519	3.795043	9.549407	408.237154	18.455534	356.674032
std	27.439466	2.105710	8.707259	168.537116	2.164946	91.294864
min	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000
25%	45.925000	2.100175	4.000000	279.000000	17.400000	375.377500
50%	74.450000	3.207450	5.000000	330.000000	19.050000	391.440000
75%	93.575000	5.188425	24.000000	666.000000	20.200000	396.225000
max	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000

	LSTAT	MEDV
count	506.000000	506.000000
mean	12.715432	22.532806
std	7.012739	9.197104
min	1.730000	5.000000
25%	7.230000	17.025000
50%	11.995000	21.200000
75%	16.570000	25.000000
max	37.970000	50.000000

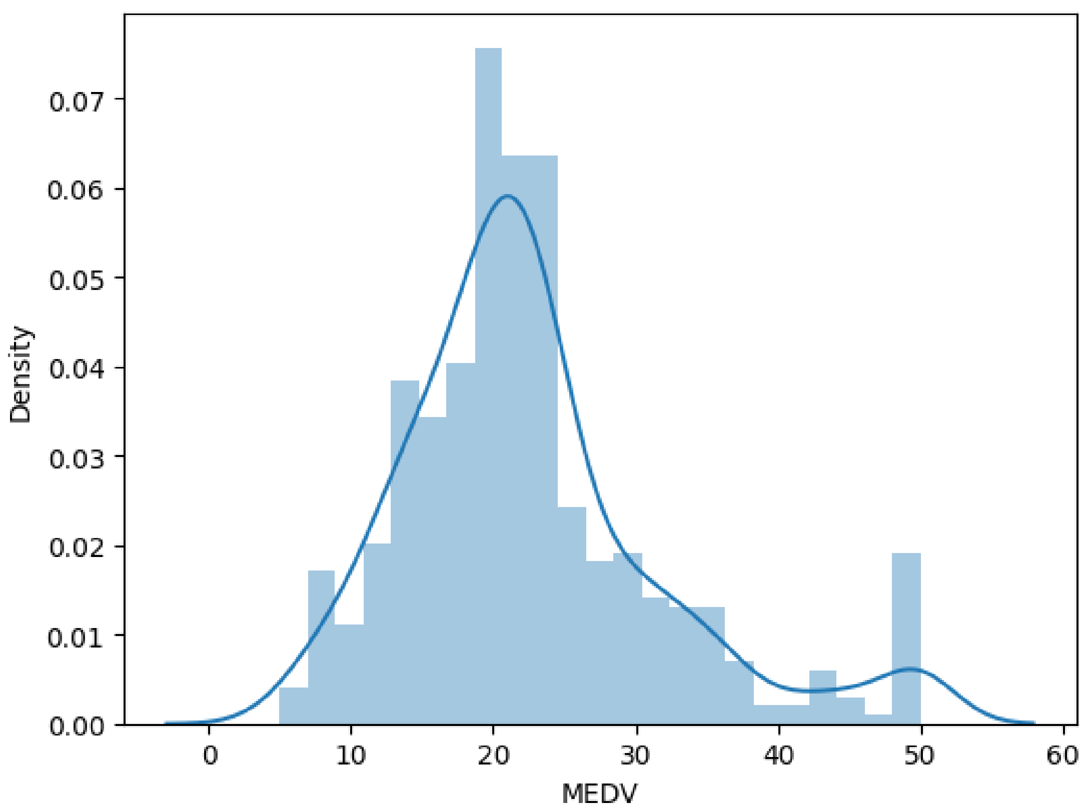
[13]:
```
data.info()
```

```
<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    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
```

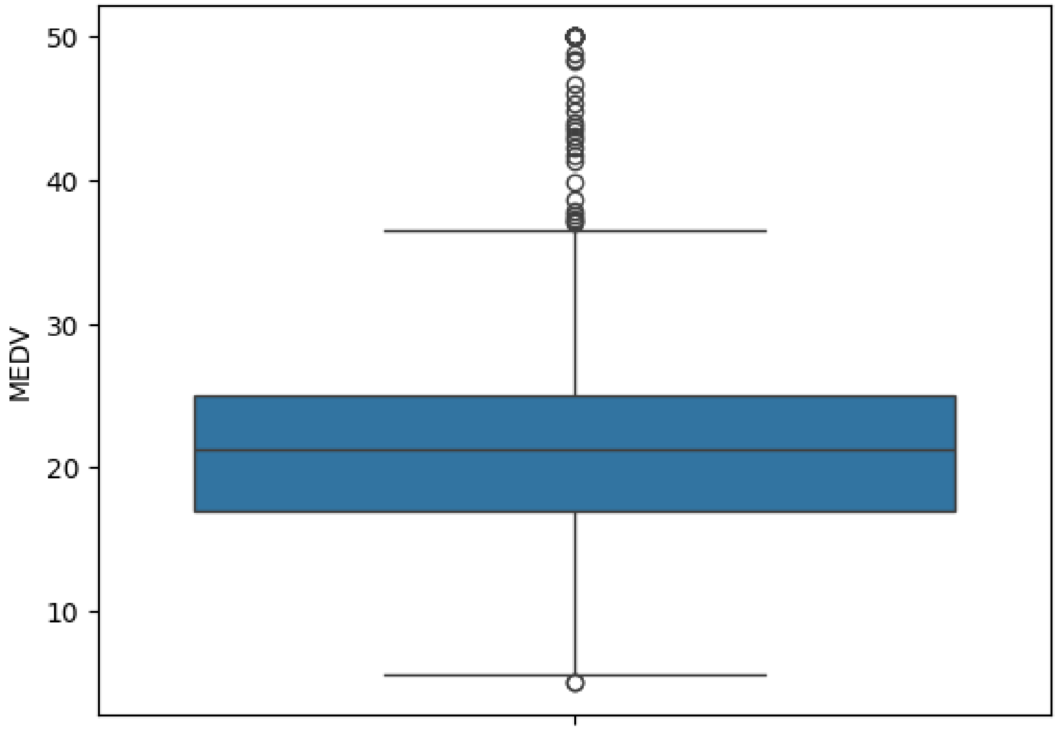
[14]:
```python
import seaborn as sns
sns.distplot(data.MEDV)
```

[14]: `<Axes: xlabel='MEDV', ylabel='Density'>`



[15]:
```python
sns.boxplot(data.MEDV)
```

[15]: <Axes: ylabel='MEDV'>



[16]:
```
correlation = data.corr()
correlation.loc['MEDV']
```

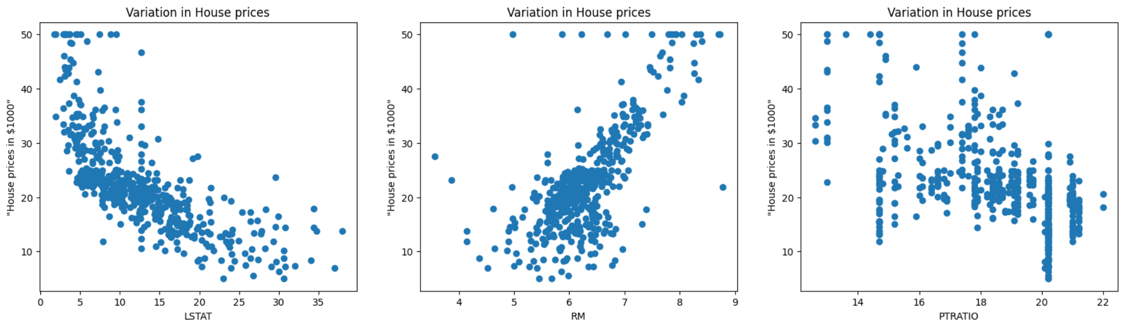
[16]:
```
CRIM      -0.379695
ZN         0.365943
INDUS     -0.478657
CHAS       0.179882
NOX       -0.427321
RM         0.695360
AGE       -0.380223
DIS        0.249929
RAD       -0.381626
TAX       -0.468536
PTRATIO   -0.507787
B          0.333461
LSTAT     -0.721975
MEDV       1.000000
Name: MEDV, dtype: float64
```

[17]:
```
# plotting the heatmap
import matplotlib.pyplot as plt
```

```python
fig,axes = plt.subplots(figsize=(15,12))
sns.heatmap(correlation,square = True,annot = True)
```

[17]: <Axes: >

[18]:
```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"')
```



[19]:
```
# Splitting the dependent feature and independent feature
#X = data[['LSTAT','RM','PTRATIO']]
X = data.iloc[:,:-1]
y= data.MEDV
```

[20]:
```
import numpy as np
from sklearn.model_selection import train_test_split
```

[21]:
```
# 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)
```

[22]:
```
# Now you can proceed with the code you provided
# Importing necessary libraries
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
```

[23]:
```
# Scaling the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[24]:
```
mean = X_train.mean(axis=0)
std = X_train.std(axis=0)
X_train = (X_train - mean) / std
X_test = (X_test - mean) / std
```

[25]:
```
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)
# Model Evaluation
```

[25]: LinearRegression()

[26]:
```
#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)
```

5.0017668901941725

[27]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2)
```

0.6588520195508143

[28]:
```
import keras
from keras.layers import Dense
from keras.models import Sequential
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

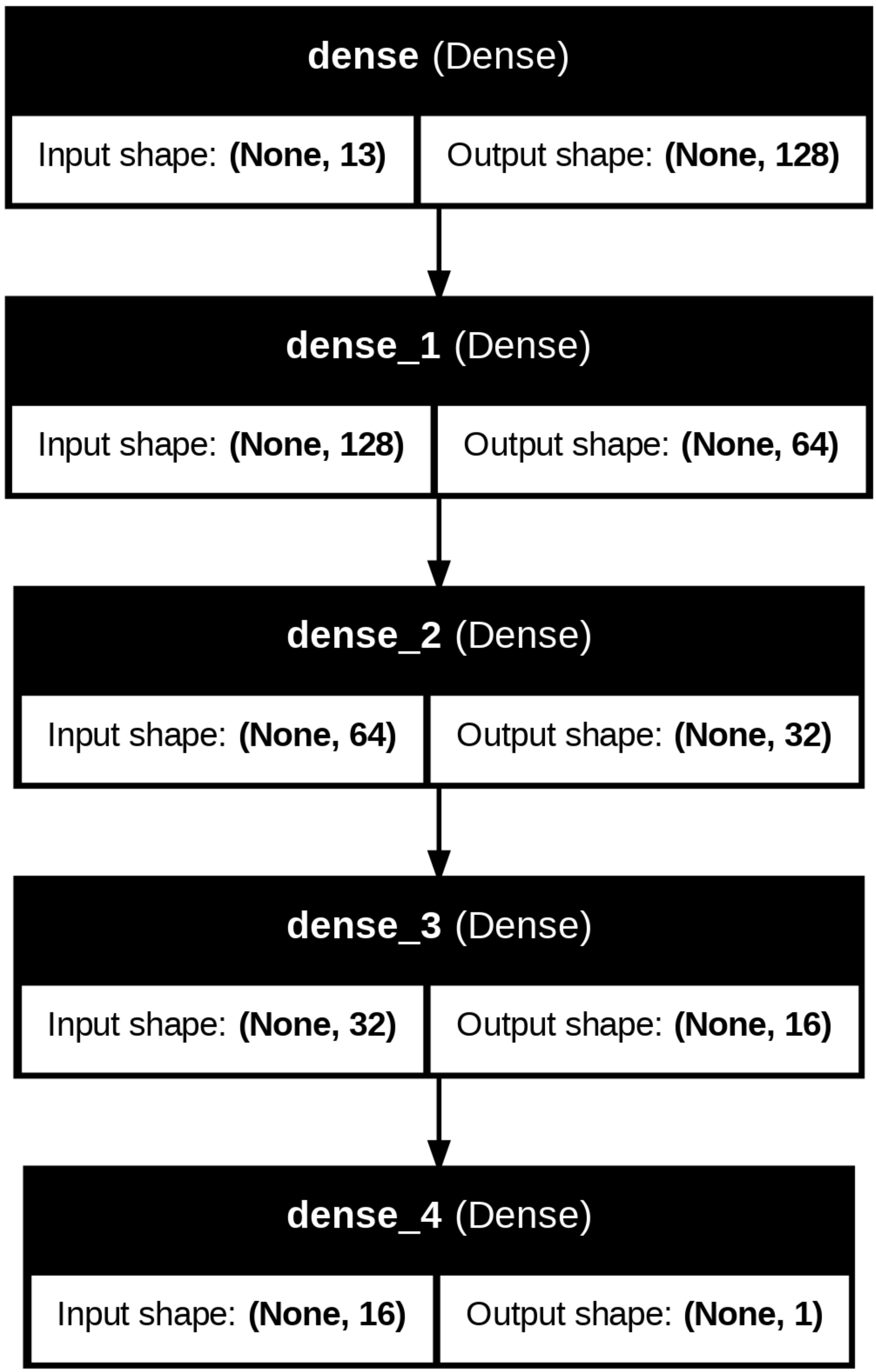
[29]:
```
# Assuming X_train and X_test are defined and initialized previously
# Assuming y_train is also defined and initialized

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

[30]:
```
# 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))
```

[31]:
```
# Compiling the model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
```

[32]:
```
# Visualizing the model architecture
keras.utils.plot_model(model, to_file='model.png', show_shapes=True,␣
 ↪show_layer_names=True)
```

[32]:


dense (Dense)
Input shape: (None, 13)
Output shape: (None, 128)
dense_1 (Dense)
Input shape: (None, 128)
Output shape: (None, 64)
dense_2 (Dense)
Input shape: (None, 64)
Output shape: (None, 32)
dense_3 (Dense)
Input shape: (None, 32)
Output shape: (None, 16)
dense_4 (Dense)
Input shape: (None, 16)
Output shape: (None, 1)

[33]:
```
# 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)
```

```
Epoch 1/100
12/12           4s 58ms/step -
loss: 593.3771 - mae: 22.5601 - val_loss: 503.5705 - val_mae: 20.9755
Epoch 2/100
12/12           1s 18ms/step -
loss: 547.7867 - mae: 21.6828 - val_loss: 436.8997 - val_mae: 19.4067
Epoch 3/100
12/12           0s 18ms/step -
loss: 456.3904 - mae: 19.3748 - val_loss: 289.0431 - val_mae: 15.3523
Epoch 4/100
12/12           0s 12ms/step -
loss: 268.2373 - mae: 14.3357 - val_loss: 89.9304 - val_mae: 7.2675
Epoch 5/100
12/12           0s 8ms/step - loss:
82.8179 - mae: 7.1041 - val_loss: 68.6730 - val_mae: 5.5175
Epoch 6/100
12/12           0s 8ms/step - loss:
58.6889 - mae: 5.9850 - val_loss: 65.0551 - val_mae: 4.9164
Epoch 7/100
12/12           0s 8ms/step - loss:
41.1953 - mae: 4.8138 - val_loss: 52.2092 - val_mae: 4.3137
Epoch 8/100
12/12           0s 8ms/step - loss:
23.9512 - mae: 3.7674 - val_loss: 54.9431 - val_mae: 4.4574
Epoch 9/100
12/12           0s 9ms/step - loss:
20.2279 - mae: 3.2662 - val_loss: 54.0492 - val_mae: 4.3647
Epoch 10/100
12/12           0s 8ms/step - loss:
21.5862 - mae: 3.2780 - val_loss: 53.0482 - val_mae: 4.3750
Epoch 11/100
12/12           0s 8ms/step - loss:
17.8879 - mae: 3.0311 - val_loss: 54.9262 - val_mae: 4.4538
Epoch 12/100
12/12           0s 10ms/step -
loss: 18.3915 - mae: 3.0601 - val_loss: 49.5495 - val_mae: 4.2035
Epoch 13/100
12/12           0s 8ms/step - loss:
14.6358 - mae: 2.7446 - val_loss: 47.2963 - val_mae: 4.1303
Epoch 14/100
12/12           0s 8ms/step - loss:
12.3518 - mae: 2.5399 - val_loss: 52.2705 - val_mae: 4.3731
```

```
Epoch 15/100
12/12            0s 8ms/step - loss:
11.6730 - mae: 2.6078 - val_loss: 45.9116 - val_mae: 4.0463
Epoch 16/100
12/12            0s 8ms/step - loss:
13.0567 - mae: 2.6684 - val_loss: 44.0094 - val_mae: 4.0050
Epoch 17/100
12/12            0s 9ms/step - loss:
11.2538 - mae: 2.4903 - val_loss: 48.5124 - val_mae: 4.1990
Epoch 18/100
12/12            0s 8ms/step - loss:
12.4796 - mae: 2.6017 - val_loss: 43.8827 - val_mae: 3.9347
Epoch 19/100
12/12            0s 10ms/step -
loss: 10.3371 - mae: 2.3831 - val_loss: 42.4275 - val_mae: 3.8965
Epoch 20/100
12/12            0s 8ms/step - loss:
10.1163 - mae: 2.3862 - val_loss: 43.9888 - val_mae: 4.0034
Epoch 21/100
12/12            0s 8ms/step - loss:
12.1080 - mae: 2.6097 - val_loss: 42.1764 - val_mae: 3.8751
Epoch 22/100
12/12            0s 8ms/step - loss:
13.0279 - mae: 2.6222 - val_loss: 42.3794 - val_mae: 3.8820
Epoch 23/100
12/12            0s 8ms/step - loss:
12.4705 - mae: 2.6305 - val_loss: 42.1903 - val_mae: 3.8872
Epoch 24/100
12/12            0s 8ms/step - loss:
10.1416 - mae: 2.3600 - val_loss: 39.4052 - val_mae: 3.7511
Epoch 25/100
12/12            0s 9ms/step - loss:
12.3685 - mae: 2.5854 - val_loss: 40.8676 - val_mae: 3.8082
Epoch 26/100
12/12            0s 10ms/step -
loss: 10.6349 - mae: 2.4289 - val_loss: 39.5760 - val_mae: 3.7362
Epoch 27/100
12/12            0s 12ms/step -
loss: 11.5190 - mae: 2.3533 - val_loss: 40.3245 - val_mae: 3.7892
Epoch 28/100
12/12            0s 8ms/step - loss:
11.7611 - mae: 2.4875 - val_loss: 37.7588 - val_mae: 3.6927
Epoch 29/100
12/12            0s 8ms/step - loss:
9.6872 - mae: 2.3049 - val_loss: 37.7210 - val_mae: 3.6995
Epoch 30/100
12/12            0s 8ms/step - loss:
9.4729 - mae: 2.2603 - val_loss: 37.6014 - val_mae: 3.6595
```

```
Epoch 31/100
12/12            0s 8ms/step - loss:
9.8275 - mae: 2.3420 - val_loss: 37.2779 - val_mae: 3.6696
Epoch 32/100
12/12            0s 9ms/step - loss:
9.3786 - mae: 2.2445 - val_loss: 36.1026 - val_mae: 3.6263
Epoch 33/100
12/12            0s 8ms/step - loss:
8.9594 - mae: 2.3041 - val_loss: 37.2190 - val_mae: 3.6683
Epoch 34/100
12/12            0s 10ms/step -
loss: 9.9074 - mae: 2.2940 - val_loss: 36.5846 - val_mae: 3.6581
Epoch 35/100
12/12            0s 8ms/step - loss:
8.6307 - mae: 2.1964 - val_loss: 36.3040 - val_mae: 3.6130
Epoch 36/100
12/12            0s 9ms/step - loss:
8.9524 - mae: 2.2336 - val_loss: 36.4816 - val_mae: 3.6294
Epoch 37/100
12/12            0s 8ms/step - loss:
9.9382 - mae: 2.2425 - val_loss: 34.3630 - val_mae: 3.5571
Epoch 38/100
12/12            0s 8ms/step - loss:
9.2554 - mae: 2.3015 - val_loss: 34.5850 - val_mae: 3.5578
Epoch 39/100
12/12            0s 7ms/step - loss:
9.4607 - mae: 2.2191 - val_loss: 35.1494 - val_mae: 3.6242
Epoch 40/100
12/12            0s 9ms/step - loss:
8.5204 - mae: 2.1706 - val_loss: 33.6694 - val_mae: 3.4636
Epoch 41/100
12/12            0s 13ms/step -
loss: 8.2859 - mae: 2.1481 - val_loss: 33.1372 - val_mae: 3.5285
Epoch 42/100
12/12            0s 8ms/step - loss:
9.1196 - mae: 2.2534 - val_loss: 31.3729 - val_mae: 3.4401
Epoch 43/100
12/12            0s 8ms/step - loss:
8.0888 - mae: 2.1628 - val_loss: 33.4671 - val_mae: 3.5304
Epoch 44/100
12/12            0s 8ms/step - loss:
8.6502 - mae: 2.1664 - val_loss: 32.2912 - val_mae: 3.4499
Epoch 45/100
12/12            0s 8ms/step - loss:
8.2090 - mae: 2.1190 - val_loss: 32.1869 - val_mae: 3.4769
Epoch 46/100
12/12            0s 8ms/step - loss:
7.6148 - mae: 2.0659 - val_loss: 30.1620 - val_mae: 3.3653
```

```
Epoch 47/100
12/12            0s 9ms/step - loss:
7.2166 - mae: 2.0003 - val_loss: 33.5671 - val_mae: 3.5768
Epoch 48/100
12/12            0s 10ms/step -
loss: 6.2708 - mae: 1.8750 - val_loss: 30.2040 - val_mae: 3.3250
Epoch 49/100
12/12            0s 8ms/step - loss:
8.0726 - mae: 2.0969 - val_loss: 28.5830 - val_mae: 3.3349
Epoch 50/100
12/12            0s 8ms/step - loss:
7.0572 - mae: 2.0626 - val_loss: 28.7979 - val_mae: 3.3236
Epoch 51/100
12/12            0s 8ms/step - loss:
6.8565 - mae: 1.9374 - val_loss: 30.9917 - val_mae: 3.4276
Epoch 52/100
12/12            0s 8ms/step - loss:
6.5159 - mae: 1.9832 - val_loss: 29.8466 - val_mae: 3.4046
Epoch 53/100
12/12            0s 8ms/step - loss:
7.7821 - mae: 2.0660 - val_loss: 29.2727 - val_mae: 3.3463
Epoch 54/100
12/12            0s 13ms/step -
loss: 6.4354 - mae: 1.9539 - val_loss: 26.8660 - val_mae: 3.1803
Epoch 55/100
12/12            0s 8ms/step - loss:
7.1614 - mae: 1.9898 - val_loss: 28.5595 - val_mae: 3.5531
Epoch 56/100
12/12            0s 9ms/step - loss:
7.0890 - mae: 2.0280 - val_loss: 28.9531 - val_mae: 3.3067
Epoch 57/100
12/12            0s 8ms/step - loss:
6.2158 - mae: 1.8775 - val_loss: 27.1317 - val_mae: 3.2421
Epoch 58/100
12/12            0s 8ms/step - loss:
7.2125 - mae: 2.0564 - val_loss: 26.1707 - val_mae: 3.2010
Epoch 59/100
12/12            0s 8ms/step - loss:
6.5971 - mae: 1.9505 - val_loss: 25.8838 - val_mae: 3.1695
Epoch 60/100
12/12            0s 8ms/step - loss:
5.7254 - mae: 1.8261 - val_loss: 27.2371 - val_mae: 3.3160
Epoch 61/100
12/12            0s 8ms/step - loss:
6.1464 - mae: 1.8838 - val_loss: 25.3189 - val_mae: 3.1653
Epoch 62/100
12/12            0s 14ms/step -
loss: 6.4360 - mae: 1.9151 - val_loss: 28.8867 - val_mae: 3.3132
```

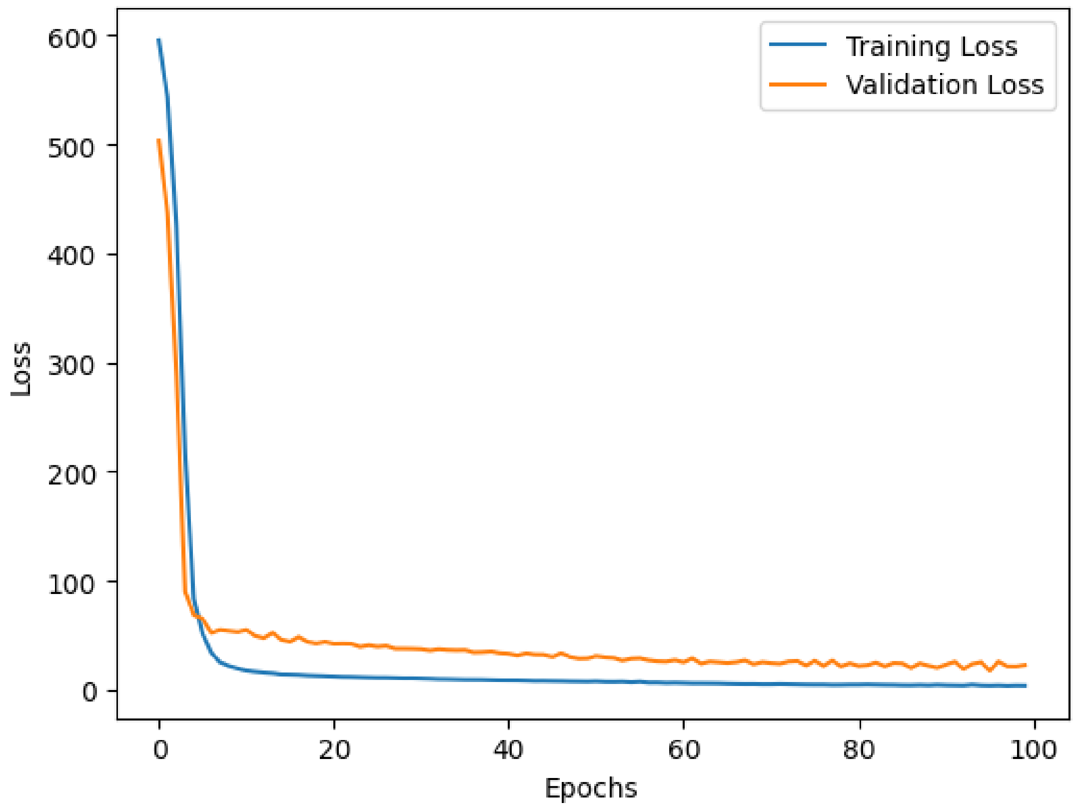
```
Epoch 63/100
12/12            0s 8ms/step - loss:
6.4742 - mae: 1.9304 - val_loss: 23.9193 - val_mae: 3.1014
Epoch 64/100
12/12            0s 15ms/step -
loss: 5.8761 - mae: 1.8188 - val_loss: 25.8858 - val_mae: 3.2442
Epoch 65/100
12/12            0s 16ms/step -
loss: 5.4624 - mae: 1.8257 - val_loss: 25.2661 - val_mae: 3.2248
Epoch 66/100
12/12            0s 19ms/step -
loss: 5.3750 - mae: 1.8012 - val_loss: 24.5179 - val_mae: 3.0781
Epoch 67/100
12/12            0s 16ms/step -
loss: 5.0549 - mae: 1.7077 - val_loss: 25.3454 - val_mae: 3.1402
Epoch 68/100
12/12            0s 14ms/step -
loss: 5.3533 - mae: 1.7359 - val_loss: 27.0084 - val_mae: 3.1959
Epoch 69/100
12/12            1s 32ms/step -
loss: 4.9062 - mae: 1.7063 - val_loss: 23.4451 - val_mae: 3.0110
Epoch 70/100
12/12            1s 24ms/step -
loss: 5.1193 - mae: 1.7162 - val_loss: 25.3332 - val_mae: 3.1843
Epoch 71/100
12/12            1s 22ms/step -
loss: 4.9982 - mae: 1.7085 - val_loss: 24.4066 - val_mae: 3.0924
Epoch 72/100
12/12            1s 31ms/step -
loss: 4.9755 - mae: 1.6785 - val_loss: 23.8578 - val_mae: 3.0615
Epoch 73/100
12/12            0s 15ms/step -
loss: 5.2635 - mae: 1.7548 - val_loss: 25.9125 - val_mae: 3.1621
Epoch 74/100
12/12            0s 15ms/step -
loss: 5.0754 - mae: 1.7407 - val_loss: 26.4211 - val_mae: 3.2626
Epoch 75/100
12/12            0s 14ms/step -
loss: 4.6822 - mae: 1.6245 - val_loss: 22.2759 - val_mae: 2.9427
Epoch 76/100
12/12            0s 19ms/step -
loss: 4.5572 - mae: 1.6580 - val_loss: 26.6447 - val_mae: 3.2049
Epoch 77/100
12/12            0s 12ms/step -
loss: 4.9407 - mae: 1.6649 - val_loss: 21.7052 - val_mae: 2.9334
Epoch 78/100
12/12            0s 12ms/step -
loss: 4.5315 - mae: 1.5941 - val_loss: 26.8572 - val_mae: 3.2439
```

```
Epoch 79/100
12/12            0s 8ms/step - loss:
4.7339 - mae: 1.6763 - val_loss: 21.3963 - val_mae: 3.0011
Epoch 80/100
12/12            0s 12ms/step -
loss: 4.5042 - mae: 1.6061 - val_loss: 24.2767 - val_mae: 3.1398
Epoch 81/100
12/12            0s 8ms/step - loss:
4.0584 - mae: 1.5254 - val_loss: 21.7839 - val_mae: 2.8775
Epoch 82/100
12/12            0s 8ms/step - loss:
4.6836 - mae: 1.6063 - val_loss: 22.4774 - val_mae: 3.0364
Epoch 83/100
12/12            0s 8ms/step - loss:
4.3797 - mae: 1.6137 - val_loss: 24.9015 - val_mae: 3.2107
Epoch 84/100
12/12            0s 8ms/step - loss:
4.1062 - mae: 1.5591 - val_loss: 21.3421 - val_mae: 2.8286
Epoch 85/100
12/12            0s 13ms/step -
loss: 4.2152 - mae: 1.5523 - val_loss: 24.5129 - val_mae: 3.1006
Epoch 86/100
12/12            0s 12ms/step -
loss: 4.3249 - mae: 1.5818 - val_loss: 24.1449 - val_mae: 3.0349
Epoch 87/100
12/12            0s 8ms/step - loss:
3.8499 - mae: 1.5082 - val_loss: 19.8628 - val_mae: 2.8582
Epoch 88/100
12/12            0s 8ms/step - loss:
4.2583 - mae: 1.5388 - val_loss: 24.2172 - val_mae: 3.1398
Epoch 89/100
12/12            0s 8ms/step - loss:
4.1735 - mae: 1.4921 - val_loss: 21.9524 - val_mae: 2.9326
Epoch 90/100
12/12            0s 8ms/step - loss:
3.9708 - mae: 1.5376 - val_loss: 20.1504 - val_mae: 2.9104
Epoch 91/100
12/12            0s 10ms/step -
loss: 4.3735 - mae: 1.6228 - val_loss: 23.0084 - val_mae: 3.0017
Epoch 92/100
12/12            0s 8ms/step - loss:
4.0967 - mae: 1.5490 - val_loss: 25.6773 - val_mae: 3.1165
Epoch 93/100
12/12            0s 9ms/step - loss:
3.8550 - mae: 1.4513 - val_loss: 19.0482 - val_mae: 2.7907
Epoch 94/100
12/12            0s 8ms/step - loss:
4.8954 - mae: 1.6367 - val_loss: 23.5930 - val_mae: 3.0091
```

```
Epoch 95/100
12/12            0s 8ms/step - loss:
3.9491 - mae: 1.5074 - val_loss: 25.3295 - val_mae: 3.1891
Epoch 96/100
12/12            0s 8ms/step - loss:
3.9333 - mae: 1.4771 - val_loss: 17.5333 - val_mae: 2.7032
Epoch 97/100
12/12            0s 8ms/step - loss:
3.3824 - mae: 1.4005 - val_loss: 25.8522 - val_mae: 3.1260
Epoch 98/100
12/12            0s 13ms/step -
loss: 3.4989 - mae: 1.3967 - val_loss: 21.4005 - val_mae: 2.9185
Epoch 99/100
12/12            0s 9ms/step - loss:
3.5655 - mae: 1.4107 - val_loss: 21.2673 - val_mae: 2.9697
Epoch 100/100
12/12            0s 12ms/step -
loss: 4.0185 - mae: 1.5149 - val_loss: 22.5418 - val_mae: 2.9693
```

[34]:

```python
# 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()
```



[35]:
```
# Plotting the Mean Absolute Error using Plotly
import plotly.graph_objects as go
fig = go.Figure()
fig.add_trace(go.Scattergl(y=history.history['mae'], name='Train'))
fig.add_trace(go.Scattergl(y=history.history['val_mae'], name='Valid'))
fig.update_layout(height=500, width=700, xaxis_title='Epoch', yaxis_title='Mean␣
 ↪Absolute Error')
fig.show()
```

[36]:
```
#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             1s 115ms/step
4/4             0s 20ms/step - loss:
9.4300 - mae: 2.1022
Mean squared error on test data:  12.967096328735352
Mean absolute error on test data:  2.2716712951660156
```

[37]:
```
#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:  25.017672023842852
Mean absolute error on test data:  3.1499233573458025
0.8231770462356038
```

[38]:
```
# 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)
```

```
3.600985441393236
```

[39]:
```
# Make predictions on new data
import sklearn
new_data = sklearn.preprocessing.StandardScaler().fit_transform(([[0.1, 10.0,
5.0, 0, 0.4, 6.0, 50, 6.0, 1, 400, 20, 300, 10]]))
prediction = model.predict(new_data)
print("Predicted house price:", prediction)
```

```
1/1              0s 38ms/step
Predicted house price: [[13.182609]]
```

[ ]: