mszcuxc3a

March 20, 2025

[15]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
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
```

[33]:
```
# Importing DataSet and take a look at Data
data = pd.read_csv('housingdata.csv')
data
```

[33]:

	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				
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				

```
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]
```

[34]:
```
data.isnull().sum()
```

[34]:
```
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
```

[35]:
```
# Handle null values by filling them with the mean of the respective columns
data.fillna(data.mean(), inplace=True)
```

[36]:
```
data.isnull().sum()
```

[36]:
```
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
```

[37]:
```
data.describe()
```

[37]:

	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

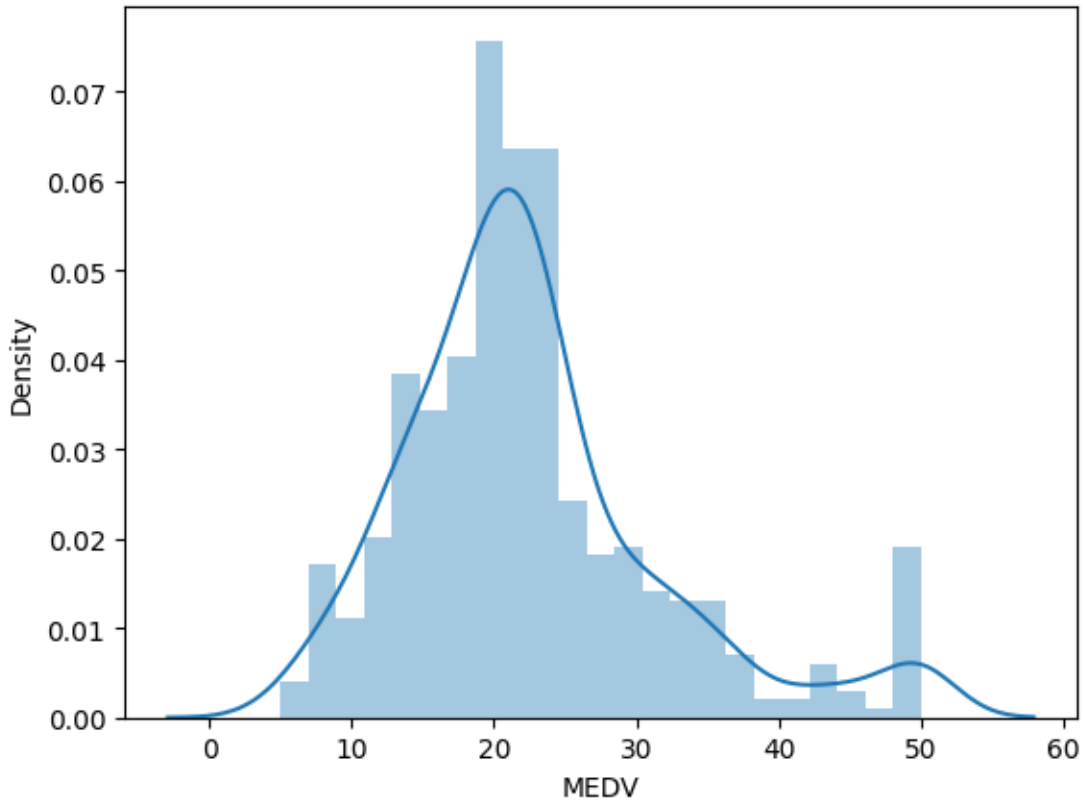
[38]:
```
data.info()
```

```
<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
```

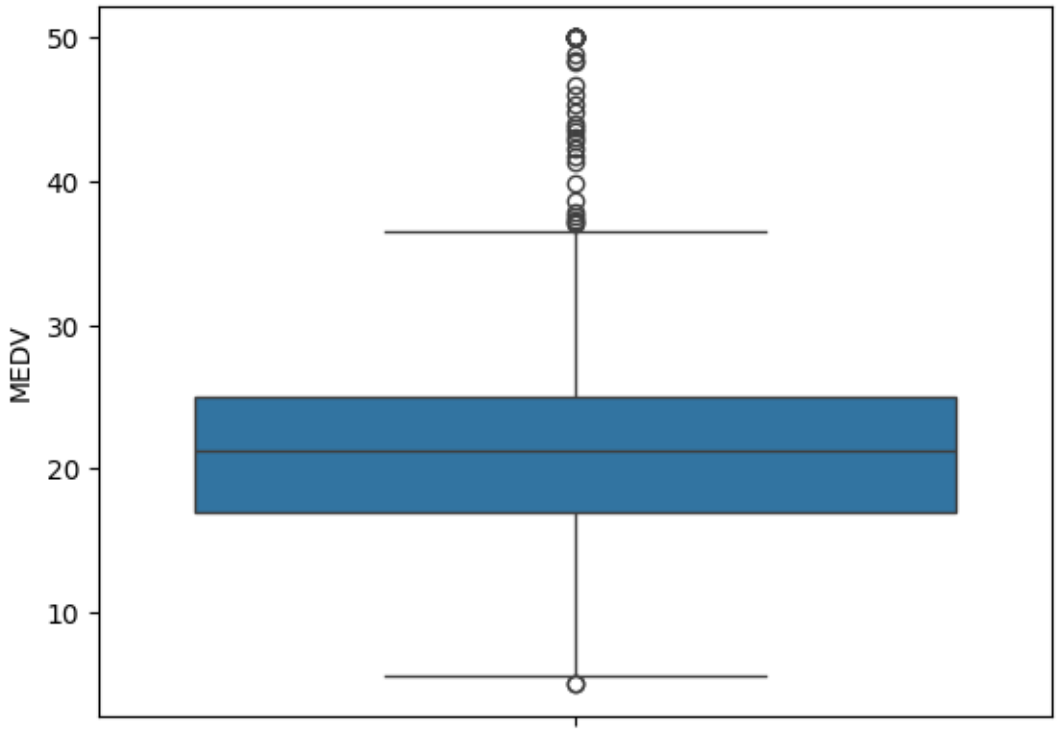
[39]:
```
import seaborn as sns
sns.distplot(data.MEDV)
```

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



[40]:
```
sns.boxplot(data.MEDV)
```

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



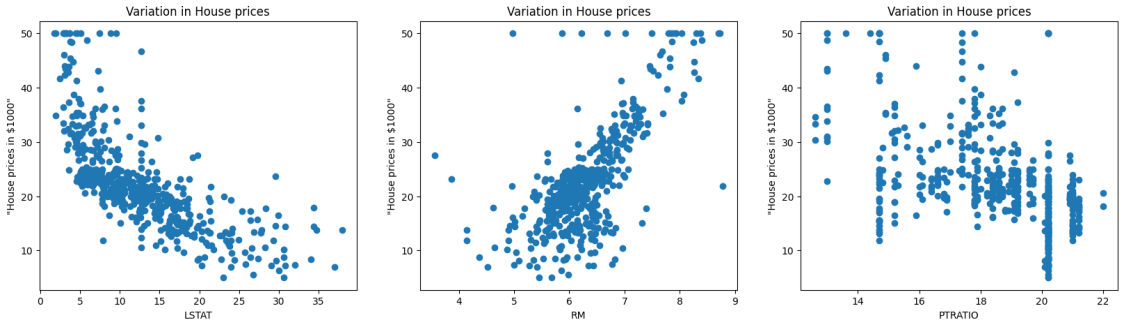
[41]:
```
correlation = data.corr()
correlation.loc['MEDV']
```

[41]:
```
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
```

[42]:
```
# plotting the heatmap
import matplotlib.pyplot as plt
fig,axes = plt.subplots(figsize=(15,12))
sns.heatmap(correlation,square = True,annot = True)
```

[42]: <Axes: >

[43]:
```
# 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"')
```



[44]:
```
# Splitting the dependent feature and independent feature
#X = data[['LSTAT','RM','PTRATIO']]
X = data.iloc[:,:-1]
y= data.MEDV
```

[45]:
```
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)
```

```
# Model Evaluation
```

[45]:
```
LinearRegression()
```

[46]:
```
#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)
```

```
5.001766890194174
```

[47]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2)
```

```
0.658852019550814
```

[48]:
```
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 graphviz (see instructions at
https://graphviz.gitlab.io/download/) for `plot_model` to work.
Epoch 1/100
12/12             6s 63ms/step -
loss: 637.5823 - mae: 23.3392 - val_loss: 501.9428 - val_mae: 20.9218
Epoch 2/100
12/12             0s 16ms/step -
loss: 576.0250 - mae: 21.9062 - val_loss: 426.7097 - val_mae: 19.0900
Epoch 3/100
12/12             0s 16ms/step -
loss: 472.9474 - mae: 19.5632 - val_loss: 260.3259 - val_mae: 14.2853
Epoch 4/100
12/12             0s 22ms/step -
loss: 226.6125 - mae: 12.8536 - val_loss: 70.4344 - val_mae: 5.7795
Epoch 5/100
12/12             1s 18ms/step -
loss: 101.2044 - mae: 7.9105 - val_loss: 63.4414 - val_mae: 5.2930
Epoch 6/100
12/12             0s 18ms/step -
loss: 66.6758 - mae: 6.2065 - val_loss: 67.2448 - val_mae: 5.1342
Epoch 7/100
12/12             0s 19ms/step -
loss: 39.7218 - mae: 4.6565 - val_loss: 56.6962 - val_mae: 4.7042
Epoch 8/100
12/12             0s 16ms/step -
loss: 22.6439 - mae: 3.5725 - val_loss: 53.8994 - val_mae: 4.7284
Epoch 9/100
12/12             0s 17ms/step -
loss: 22.5735 - mae: 3.4815 - val_loss: 55.6253 - val_mae: 4.6438
Epoch 10/100
12/12             0s 19ms/step -
loss: 20.3210 - mae: 3.1774 - val_loss: 55.0472 - val_mae: 4.5780
Epoch 11/100
12/12             0s 22ms/step -
loss: 19.6458 - mae: 3.1178 - val_loss: 53.4736 - val_mae: 4.5105
```

```
Epoch 12/100
12/12            0s 16ms/step -
loss: 15.4462 - mae: 2.8748 - val_loss: 50.7672 - val_mae: 4.3458
Epoch 13/100
12/12            0s 15ms/step -
loss: 19.8370 - mae: 3.2098 - val_loss: 49.7969 - val_mae: 4.3428
Epoch 14/100
12/12            0s 18ms/step -
loss: 14.6462 - mae: 2.7288 - val_loss: 49.2393 - val_mae: 4.3534
Epoch 15/100
12/12            0s 13ms/step -
loss: 15.0232 - mae: 2.9019 - val_loss: 45.9021 - val_mae: 4.2246
Epoch 16/100
12/12            0s 11ms/step -
loss: 15.3578 - mae: 2.8083 - val_loss: 46.2940 - val_mae: 4.2638
Epoch 17/100
12/12            0s 18ms/step -
loss: 13.5767 - mae: 2.6155 - val_loss: 44.7240 - val_mae: 4.1594
Epoch 18/100
12/12            0s 17ms/step -
loss: 12.9807 - mae: 2.6075 - val_loss: 44.4587 - val_mae: 4.1081
Epoch 19/100
12/12            0s 19ms/step -
loss: 12.7454 - mae: 2.6092 - val_loss: 44.1038 - val_mae: 4.1315
Epoch 20/100
12/12            0s 18ms/step -
loss: 10.9545 - mae: 2.4212 - val_loss: 42.1538 - val_mae: 4.0154
Epoch 21/100
12/12            0s 18ms/step -
loss: 13.6765 - mae: 2.6469 - val_loss: 42.3075 - val_mae: 4.0334
Epoch 22/100
12/12            0s 14ms/step -
loss: 11.0072 - mae: 2.4324 - val_loss: 42.4709 - val_mae: 4.0275
Epoch 23/100
12/12            0s 15ms/step -
loss: 11.9222 - mae: 2.4238 - val_loss: 40.3017 - val_mae: 3.9351
Epoch 24/100
12/12            0s 13ms/step -
loss: 11.6086 - mae: 2.5241 - val_loss: 41.2279 - val_mae: 4.0053
Epoch 25/100
12/12            0s 28ms/step -
loss: 10.6481 - mae: 2.3478 - val_loss: 40.2435 - val_mae: 3.9149
Epoch 26/100
12/12            0s 18ms/step -
loss: 13.4676 - mae: 2.6480 - val_loss: 39.4666 - val_mae: 3.9127
Epoch 27/100
12/12            0s 15ms/step -
loss: 11.2125 - mae: 2.4112 - val_loss: 39.4673 - val_mae: 3.8951
```

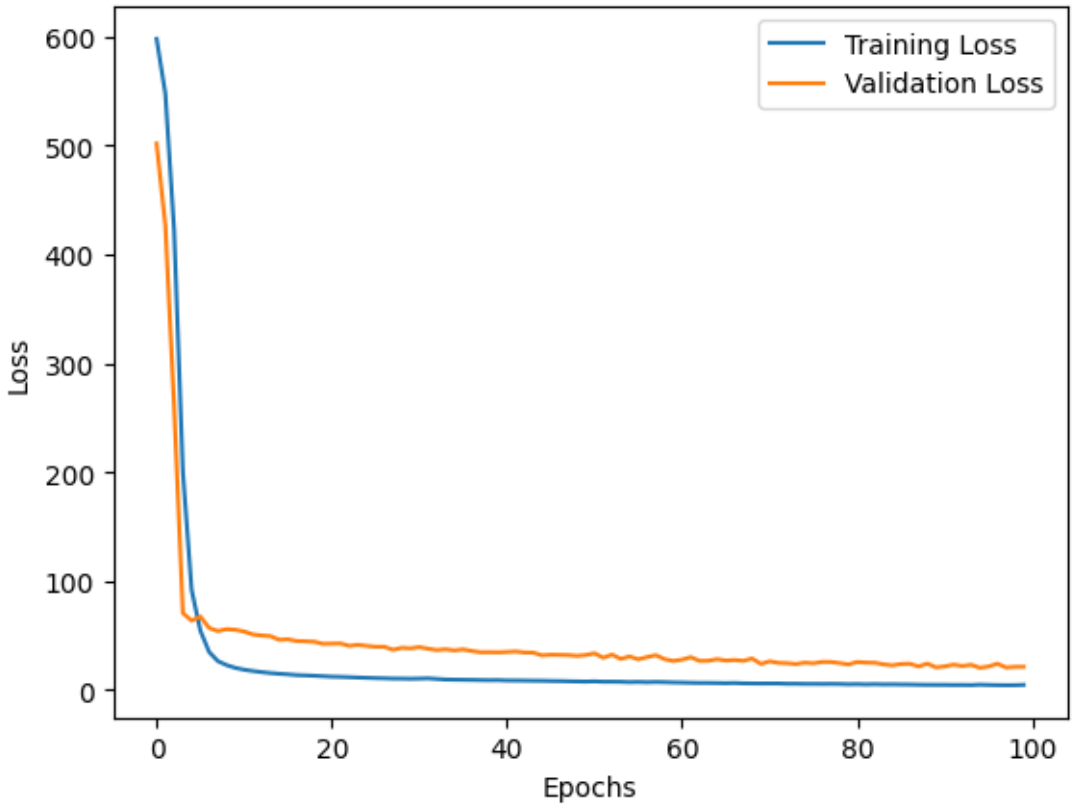
```
Epoch 28/100
12/12            0s 16ms/step -
loss: 11.7875 - mae: 2.5431 - val_loss: 36.6572 - val_mae: 3.7522
Epoch 29/100
12/12            0s 16ms/step -
loss: 11.2983 - mae: 2.4388 - val_loss: 38.5011 - val_mae: 3.9132
Epoch 30/100
12/12            0s 17ms/step -
loss: 11.5294 - mae: 2.3625 - val_loss: 38.0044 - val_mae: 3.8109
Epoch 31/100
12/12            0s 19ms/step -
loss: 10.8664 - mae: 2.4427 - val_loss: 39.1625 - val_mae: 3.9447
Epoch 32/100
12/12            0s 11ms/step -
loss: 11.4726 - mae: 2.4421 - val_loss: 37.6246 - val_mae: 3.8349
Epoch 33/100
12/12            0s 15ms/step -
loss: 10.3745 - mae: 2.3295 - val_loss: 36.4306 - val_mae: 3.7920
Epoch 34/100
12/12            0s 15ms/step -
loss: 8.2271 - mae: 2.0752 - val_loss: 37.1040 - val_mae: 3.8626
Epoch 35/100
12/12            0s 16ms/step -
loss: 7.5315 - mae: 1.9786 - val_loss: 36.1065 - val_mae: 3.7770
Epoch 36/100
12/12            0s 16ms/step -
loss: 9.0459 - mae: 2.1542 - val_loss: 37.0387 - val_mae: 3.8579
Epoch 37/100
12/12            0s 15ms/step -
loss: 8.7370 - mae: 2.1251 - val_loss: 35.5475 - val_mae: 3.7793
Epoch 38/100
12/12            0s 18ms/step -
loss: 9.1603 - mae: 2.2012 - val_loss: 34.4235 - val_mae: 3.7528
Epoch 39/100
12/12            0s 15ms/step -
loss: 9.5289 - mae: 2.2026 - val_loss: 34.4647 - val_mae: 3.7543
Epoch 40/100
12/12            0s 14ms/step -
loss: 8.6058 - mae: 2.1994 - val_loss: 34.3443 - val_mae: 3.7876
Epoch 41/100
12/12            0s 16ms/step -
loss: 8.6005 - mae: 2.1878 - val_loss: 34.6477 - val_mae: 3.7994
Epoch 42/100
12/12            0s 15ms/step -
loss: 8.1823 - mae: 2.1236 - val_loss: 35.1218 - val_mae: 3.7617
Epoch 43/100
12/12            0s 17ms/step -
loss: 8.4245 - mae: 2.1372 - val_loss: 34.2113 - val_mae: 3.7304
```

```
Epoch 44/100
12/12            0s 16ms/step -
loss: 8.1443 - mae: 2.1329 - val_loss: 34.0813 - val_mae: 3.6969
Epoch 45/100
12/12            0s 21ms/step -
loss: 7.4777 - mae: 2.0959 - val_loss: 31.5419 - val_mae: 3.6159
Epoch 46/100
12/12            0s 27ms/step -
loss: 8.7502 - mae: 2.1694 - val_loss: 32.1069 - val_mae: 3.6619
Epoch 47/100
12/12            0s 16ms/step -
loss: 7.2737 - mae: 2.0741 - val_loss: 31.9654 - val_mae: 3.6828
Epoch 48/100
12/12            0s 34ms/step -
loss: 7.6746 - mae: 2.1023 - val_loss: 31.7952 - val_mae: 3.6014
Epoch 49/100
12/12            1s 22ms/step -
loss: 7.0337 - mae: 1.9327 - val_loss: 31.1338 - val_mae: 3.5977
Epoch 50/100
12/12            0s 24ms/step -
loss: 7.3431 - mae: 2.0354 - val_loss: 31.8056 - val_mae: 3.6743
Epoch 51/100
12/12            1s 76ms/step -
loss: 7.8528 - mae: 2.0837 - val_loss: 33.4314 - val_mae: 3.7830
Epoch 52/100
12/12            1s 27ms/step -
loss: 7.5038 - mae: 1.9968 - val_loss: 29.3334 - val_mae: 3.4450
Epoch 53/100
12/12            1s 48ms/step -
loss: 8.9781 - mae: 2.2202 - val_loss: 32.2382 - val_mae: 3.6970
Epoch 54/100
12/12            0s 21ms/step -
loss: 7.4570 - mae: 2.0201 - val_loss: 28.3175 - val_mae: 3.4512
Epoch 55/100
12/12            0s 20ms/step -
loss: 7.5158 - mae: 2.0010 - val_loss: 30.5191 - val_mae: 3.6869
Epoch 56/100
12/12            0s 20ms/step -
loss: 7.1159 - mae: 2.0444 - val_loss: 27.9140 - val_mae: 3.4476
Epoch 57/100
12/12            0s 21ms/step -
loss: 6.5534 - mae: 1.9562 - val_loss: 29.9596 - val_mae: 3.5577
Epoch 58/100
12/12            0s 20ms/step -
loss: 7.2865 - mae: 2.0860 - val_loss: 31.7106 - val_mae: 3.7737
Epoch 59/100
12/12            0s 23ms/step -
loss: 6.8538 - mae: 1.9976 - val_loss: 28.0757 - val_mae: 3.4892
```

```
Epoch 60/100
12/12            0s 20ms/step -
loss: 7.3235 - mae: 1.9828 - val_loss: 26.4554 - val_mae: 3.4103
Epoch 61/100
12/12            0s 19ms/step -
loss: 5.7066 - mae: 1.8667 - val_loss: 27.6188 - val_mae: 3.4808
Epoch 62/100
12/12            0s 18ms/step -
loss: 6.8279 - mae: 1.9656 - val_loss: 29.8978 - val_mae: 3.6490
Epoch 63/100
12/12            0s 19ms/step -
loss: 6.9390 - mae: 1.9441 - val_loss: 26.4475 - val_mae: 3.3885
Epoch 64/100
12/12            0s 23ms/step -
loss: 6.1152 - mae: 1.9217 - val_loss: 26.6189 - val_mae: 3.4163
Epoch 65/100
12/12            0s 21ms/step -
loss: 5.8128 - mae: 1.8767 - val_loss: 28.0359 - val_mae: 3.5491
Epoch 66/100
12/12            0s 23ms/step -
loss: 6.2168 - mae: 1.9205 - val_loss: 26.7844 - val_mae: 3.4631
Epoch 67/100
12/12            0s 16ms/step -
loss: 6.2298 - mae: 1.8120 - val_loss: 27.2993 - val_mae: 3.4859
Epoch 68/100
12/12            1s 43ms/step -
loss: 5.8782 - mae: 1.8068 - val_loss: 26.5276 - val_mae: 3.4292
Epoch 69/100
12/12            0s 20ms/step -
loss: 6.1733 - mae: 1.8634 - val_loss: 28.6765 - val_mae: 3.6696
Epoch 70/100
12/12            0s 18ms/step -
loss: 4.8117 - mae: 1.6656 - val_loss: 23.6609 - val_mae: 3.2500
Epoch 71/100
12/12            0s 17ms/step -
loss: 5.3697 - mae: 1.7578 - val_loss: 26.0816 - val_mae: 3.4494
Epoch 72/100
12/12            0s 16ms/step -
loss: 4.8997 - mae: 1.6800 - val_loss: 24.6617 - val_mae: 3.3255
Epoch 73/100
12/12            0s 16ms/step -
loss: 5.0019 - mae: 1.7287 - val_loss: 24.3470 - val_mae: 3.3590
Epoch 74/100
12/12            0s 22ms/step -
loss: 4.9347 - mae: 1.6787 - val_loss: 23.5476 - val_mae: 3.3513
Epoch 75/100
12/12            0s 19ms/step -
loss: 5.2418 - mae: 1.7595 - val_loss: 24.8325 - val_mae: 3.3322
```

```
Epoch 76/100
12/12            0s 25ms/step -
loss: 5.4352 - mae: 1.7829 - val_loss: 24.2416 - val_mae: 3.3624
Epoch 77/100
12/12            0s 21ms/step -
loss: 5.2187 - mae: 1.7652 - val_loss: 25.5319 - val_mae: 3.4924
Epoch 78/100
12/12            0s 21ms/step -
loss: 5.3340 - mae: 1.7537 - val_loss: 25.4600 - val_mae: 3.4550
Epoch 79/100
12/12            0s 19ms/step -
loss: 5.2333 - mae: 1.7186 - val_loss: 24.4441 - val_mae: 3.3485
Epoch 80/100
12/12            0s 23ms/step -
loss: 4.5731 - mae: 1.5979 - val_loss: 23.2122 - val_mae: 3.3367
Epoch 81/100
12/12            1s 48ms/step -
loss: 5.5938 - mae: 1.8376 - val_loss: 25.4527 - val_mae: 3.4370
Epoch 82/100
12/12            1s 22ms/step -
loss: 4.3674 - mae: 1.6225 - val_loss: 24.8968 - val_mae: 3.4075
Epoch 83/100
12/12            0s 15ms/step -
loss: 5.1668 - mae: 1.7162 - val_loss: 24.7691 - val_mae: 3.3807
Epoch 84/100
12/12            0s 14ms/step -
loss: 4.8396 - mae: 1.6666 - val_loss: 23.3055 - val_mae: 3.3047
Epoch 85/100
12/12            0s 15ms/step -
loss: 4.9508 - mae: 1.7420 - val_loss: 22.3698 - val_mae: 3.1932
Epoch 86/100
12/12            0s 21ms/step -
loss: 5.2578 - mae: 1.7667 - val_loss: 23.5968 - val_mae: 3.3787
Epoch 87/100
12/12            0s 16ms/step -
loss: 5.3156 - mae: 1.6898 - val_loss: 23.8333 - val_mae: 3.3489
Epoch 88/100
12/12            0s 23ms/step -
loss: 4.4919 - mae: 1.6335 - val_loss: 21.3748 - val_mae: 3.1345
Epoch 89/100
12/12            0s 21ms/step -
loss: 5.1205 - mae: 1.7094 - val_loss: 23.9714 - val_mae: 3.3762
Epoch 90/100
12/12            0s 21ms/step -
loss: 4.6511 - mae: 1.6098 - val_loss: 20.4697 - val_mae: 3.1307
Epoch 91/100
12/12            1s 40ms/step -
loss: 3.7245 - mae: 1.4520 - val_loss: 21.4718 - val_mae: 3.1953
```

```
Epoch 92/100
12/12            0s 21ms/step -
loss: 4.3476 - mae: 1.5853 - val_loss: 22.9994 - val_mae: 3.2758
Epoch 93/100
12/12            0s 23ms/step -
loss: 4.6215 - mae: 1.5950 - val_loss: 21.7747 - val_mae: 3.1819
Epoch 94/100
12/12            0s 25ms/step -
loss: 4.1278 - mae: 1.5520 - val_loss: 22.8728 - val_mae: 3.3002
Epoch 95/100
12/12            1s 37ms/step -
loss: 4.2500 - mae: 1.5774 - val_loss: 20.0942 - val_mae: 3.0648
Epoch 96/100
12/12            0s 22ms/step -
loss: 4.2548 - mae: 1.5868 - val_loss: 21.3404 - val_mae: 3.1491
Epoch 97/100
12/12            0s 27ms/step -
loss: 4.4553 - mae: 1.6012 - val_loss: 23.8823 - val_mae: 3.3865
Epoch 98/100
12/12            0s 26ms/step -
loss: 4.1180 - mae: 1.4955 - val_loss: 20.3904 - val_mae: 3.0532
Epoch 99/100
12/12            1s 16ms/step -
loss: 4.2431 - mae: 1.5470 - val_loss: 20.9968 - val_mae: 3.1629
Epoch 100/100
12/12            0s 27ms/step -
loss: 4.5520 - mae: 1.6360 - val_loss: 21.0372 - val_mae: 3.1553
```



[49]:
```
# 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()
```

[50]:
```
#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             2s 241ms/step
4/4             0s 25ms/step - loss:
9.5879 - mae: 2.1784
Mean squared error on test data:  12.936566352844238
Mean absolute error on test data:  2.3468189239501953
```

[51]:
```
#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:  25.017672023842856
Mean absolute error on test data:  3.1499233573458025
0.8235933681883683
```

[52]:
```
# 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)
```

```
3.5967437614204782
```

[53]:
```
# 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 484ms/step
Predicted house price: [[11.963507]]
```