ass1

April 23, 2024

[71]:
```
#shree swami samarth
```

[1]:
```
import pandas as pd
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[2]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
```

[3]:
```
import warnings
warnings.filterwarnings("ignore")
```

[4]:
```
df = pd.read_csv("boston.csv")
```

[5]:
```
df
```

[5]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	\
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	
..	…	…	…	…	…	…	…	…	…	…	
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	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   5.33  36.2
..       ...     ...    ...   ...
501     21.0  391.99   9.67  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]
```

[6]:
```
df.head()
```

[6]:
```
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

        b  lstat  MEDV
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   5.33  36.2
```

[7]:
```
df.describe()
```

[7]:
```
             crim          zn       indus        chas         nox          rm  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   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    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
```

```
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    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    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

[8]:
```
df.tail()
```

[8]:
```
        crim   zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
501  0.06263  0.0  11.93     0  0.573  6.593  69.1  2.4786    1  273     21.0
502  0.04527  0.0  11.93     0  0.573  6.120  76.7  2.2875    1  273     21.0
503  0.06076  0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273     21.0
504  0.10959  0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273     21.0
505  0.04741  0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273     21.0

          b  lstat  MEDV
501  391.99   9.67  22.4
502  396.90   9.08  20.6
503  396.90   5.64  23.9
504  393.45   6.48  22.0
505  396.90   7.88  11.9
```

[9]:
```
df.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    int64
 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(11), int64(3)
memory usage: 55.5 KB
```

[10]:
```
# Separate the features (X) and target variable (y)
#X = data.drop('target_column', axis=1)
#y = data['target_column']

X = df.drop('MEDV', axis=1)

Y = df['MEDV']
```

[11]:
```
X
```

[11]:
```
        crim    zn  indus  chas    nox     rm   age     dis  rad  tax  \
0    0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296
1    0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222
..       ...   ...    ...   ...    ...    ...   ...     ...  ...  ...
501  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1  273
503  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273
505  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273

     ptratio       b  lstat
0       15.3  396.90   4.98
1       17.8  396.90   9.14
2       17.8  392.83   4.03
3       18.7  394.63   2.94
4       18.7  396.90   5.33
..       ...     ...    ...
501     21.0  391.99   9.67
502     21.0  396.90   9.08
503     21.0  396.90   5.64
504     21.0  393.45   6.48
505     21.0  396.90   7.88

[506 rows x 13 columns]
```

[12]:
```
Y
```

[12]:
```
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       …
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: MEDV, Length: 506, dtype: float64
```

[13]:
```
#This part scales the input features and target variables using the␣
 ↪StandardScaler from scikit-learn. The scaler is fit on the training data,␣
 ↪and then both training and test data are transformed using the same scaler.
scaler = StandardScaler().fit(X)
X = scaler.transform(X)
```

[14]:
```
# Split the data into train and test sets
#  testsize 0.2 Percentage of data to use for testing
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state=42)
```

[15]:
```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

[15]:
```
((404, 13), (404,), (102, 13), (102,))
```

[16]:
```
from keras.layers import Input

model = Sequential()
#model.add for 7 time
model.add(Input(shape = (x_train.shape[1],))) #input layer with input shape
model.add(Dense(12, activation = 'relu'))
model.add(Dense(13, activation = 'relu'))
model.add(Dense(19, activation = 'relu'))
model.add(Dense(19, activation = 'relu'))
model.add(Dense(19, activation = 'relu'))
model.add(Dense(1, activation = 'linear'))
```

[17]:
```
model.summary()
```

```
Model: "sequential"
```

```
Layer (type)                         Output Shape                ␣
↪Param #

dense (Dense)                        (None, 12)                  ␣
↪168

dense_1 (Dense)                      (None, 13)                  ␣
↪169

dense_2 (Dense)                      (None, 19)                  ␣
↪266

dense_3 (Dense)                      (None, 19)                  ␣
↪380

dense_4 (Dense)                      (None, 19)                  ␣
↪380

dense_5 (Dense)                      (None, 1)                   ␣
↪ 20


Total params: 1,383 (5.40 KB)

Trainable params: 1,383 (5.40 KB)

Non-trainable params: 0 (0.00 B)
```

[18]:
```
#The model is compiled with the Adam optimizer, mean squared error (MSE) as the␣
 ↪loss function, and mean absolute error (MAE) as an additional evaluation␣
 ↪metric

model.compile(loss='mean_squared_error', optimizer='adam' )
```

[19]:
```
#The model is trained on the scaled training data (x_train and y_train) for 20␣
 ↪epochs with a batch size of 32. The scaled test data (x_test and y_test) is␣
 ↪used for validation during training. The training history, including loss␣
 ↪and metrics values for each epoch, is stored in the history variable.
# Fitting the data to the model

history = model.fit(x_train, y_train, epochs=100, batch_size=16, verbose = 0)
```

[20]:
```
mse = model.evaluate(x_test, y_test, verbose = 0)
mse
```

[20]: 12.522753715515137

[21]:
```
#After training, make prediction on test data
y_pred = model.predict(x_test)
```

```
4/4             0s 16ms/step
```

[22]:
```
#calculate RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error(RMSE):" , rmse)
```

```
Root Mean Squared Error(RMSE): 3.538750386302523
```

[23]:
```
y_pred = model.predict(x_test)
y_pred
```

```
4/4             0s 5ms/step
```

[23]:
```
array([[27.454987],
       [32.89278 ],
       [17.452944],
       [25.303612],
       [15.917062],
       [19.693556],
       [17.11813 ],
       [14.602602],
       [25.2784  ],
       [19.036325],
       [21.321346],
       [17.513882],
       [ 6.464231],
       [17.906958],
       [16.52573 ],
       [22.423206],
       [20.137053],
       [10.732348],
       [47.894962],
       [14.764127],
       [24.974274],
       [27.381502],
       [14.572911],
       [23.990072],
       [18.044992],
       [19.46825 ],
       [20.81469 ],
       [13.034436],
       [19.293957],
       [17.511202],
```

```
[24.493526],
[23.551088],
[19.056452],
[24.560196],
[15.956036],
[15.600607],
[33.454098],
[20.494028],
[18.553719],
[25.743597],
[14.882683],
[31.436174],
[51.556454],
[18.139557],
[25.907238],
[18.20188 ],
[15.068213],
[26.377502],
[19.662882],
[25.35477 ],
[18.736301],
[35.250175],
[16.665594],
[24.05209 ],
[40.416695],
[22.207521],
[17.158577],
[33.774757],
[23.876213],
[17.610085],
[24.261   ],
[33.345276],
[31.48246 ],
[19.758612],
[23.175898],
[17.36896 ],
[16.825018],
[24.317253],
[29.672707],
[12.728228],
[21.778105],
[28.38147 ],
[ 8.041182],
[21.330404],
[20.856318],
[ 6.80382 ],
[19.844055],
```

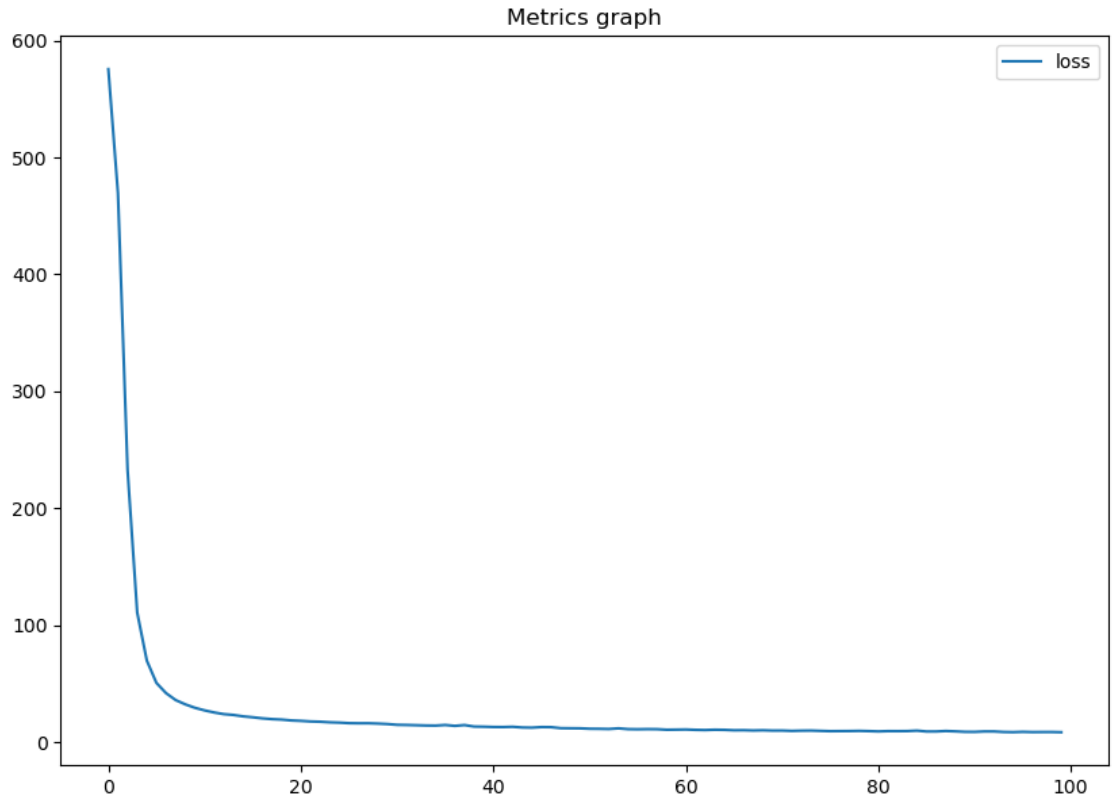
```
       [48.756413],
       [11.667135],
       [12.927367],
       [18.647955],
       [13.615104],
       [19.72286 ],
       [12.643882],
       [19.615639],
       [26.34903 ],
       [15.533831],
       [24.353762],
       [25.85339 ],
       [18.436605],
       [24.297964],
       [ 6.671335],
       [19.005749],
       [19.093147],
       [34.084385],
       [20.42649 ],
       [28.166824],
       [ 6.129143],
       [12.479241],
       [13.914244],
       [22.049635],
       [21.373142]], dtype=float32)
```

[24]:
```
y_test_np = np.array(y_test)
y_pred_np = np.array(y_pred)

# Flatten or ravel the arrays
y_test_flat = y_test_np.flatten()
y_pred_flat = y_pred_np.flatten()

# Create the DataFrame
results_df = pd.DataFrame({'Actual Values': y_test_flat, 'Predicted Values':␣
 ↪y_pred_flat})

# Print the DataFrame
print(results_df)
```

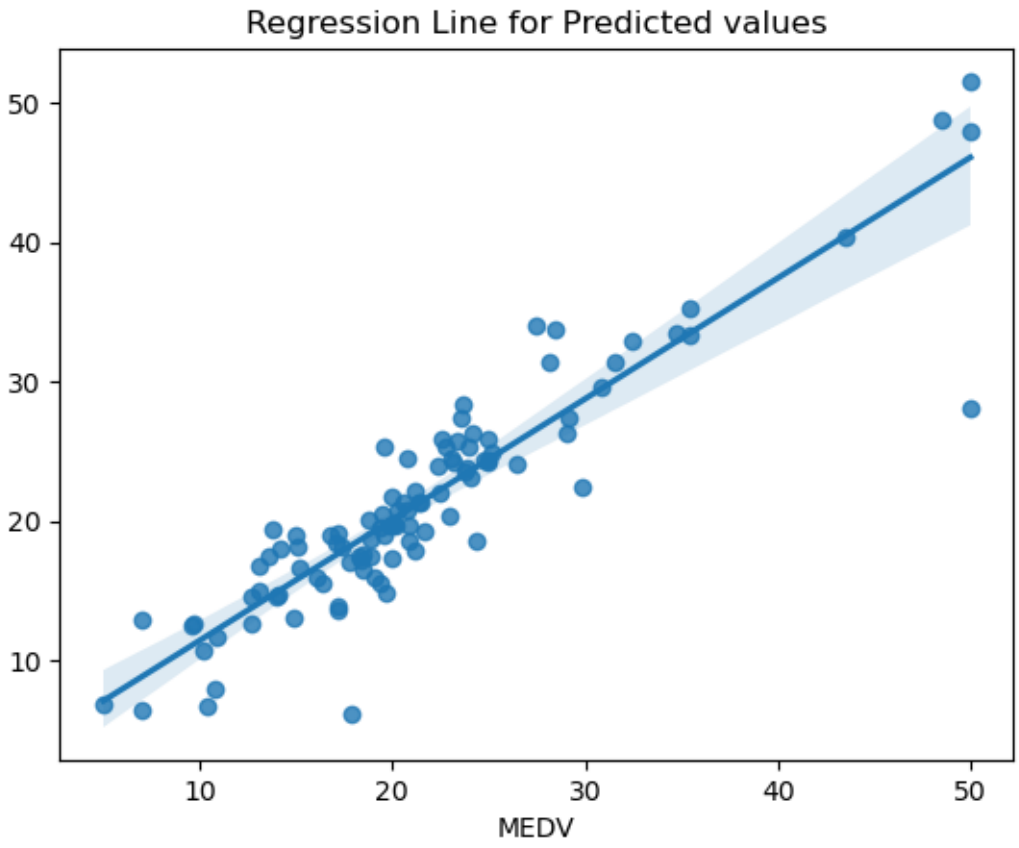
```
     Actual Values  Predicted Values
0             23.6         27.454987
1             32.4         32.892780
2             13.6         17.452944
3             22.8         25.303612
4             16.1         15.917062
..             …               …
```

```
97            17.9          6.129143
98             9.6         12.479241
99            17.2         13.914244
100           22.5         22.049635
101           21.4         21.373142

[102 rows x 2 columns]
```

[25]:
```
# This part creates a line plot using the training history stored in history.
 ↪history. The plot displays the training and validation loss and MAE values␣
 ↪over the epochs.
pd.DataFrame(history.history).plot(figsize=(10,7))
plt.title("Metrics graph")
plt.show()
```



[26]:
```
# lotting the true test values (y_test) against the predicted values (y_pred).␣
 ↪A regression line is also plotted to visualize the relationship between the␣
 ↪true and predicted values.
sns.regplot(x=y_test, y=y_pred)
plt.title("Regression Line for Predicted values")
plt.show()
```



[27]:
```
# Evaluating <keras.src.callbacks.History at 0x7c5098f9db40> the model
def regression_metrics_display(y_test, y_pred):
  print(f"MAE is {metrics.mean_absolute_error(y_test, y_pred)}")
  print(f"MSE is {metrics.mean_squared_error(y_test,y_pred)}")
  print(f"R2 score is {metrics.r2_score(y_test, y_pred)}")
```

[28]:
```
#fun call
regression_metrics_display(y_test, y_pred)
```

```
MAE is 2.2527441146327
MSE is 12.522754296556258
R2 score is 0.8292362161491259
```

[29]:
```
#basic
df.shape
```

[29]: (506, 14)

[30]:
```
df.dtypes
```

[30]:
```
crim       float64
zn         float64
indus      float64
chas         int64
nox        float64
rm         float64
age        float64
dis        float64
rad          int64
tax          int64
ptratio    float64
b          float64
lstat      float64
MEDV       float64
dtype: object
```

[31]:
```
df.isna().sum()
```

[31]:
```
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
```

[32]:
```
df.describe()
```

[32]:

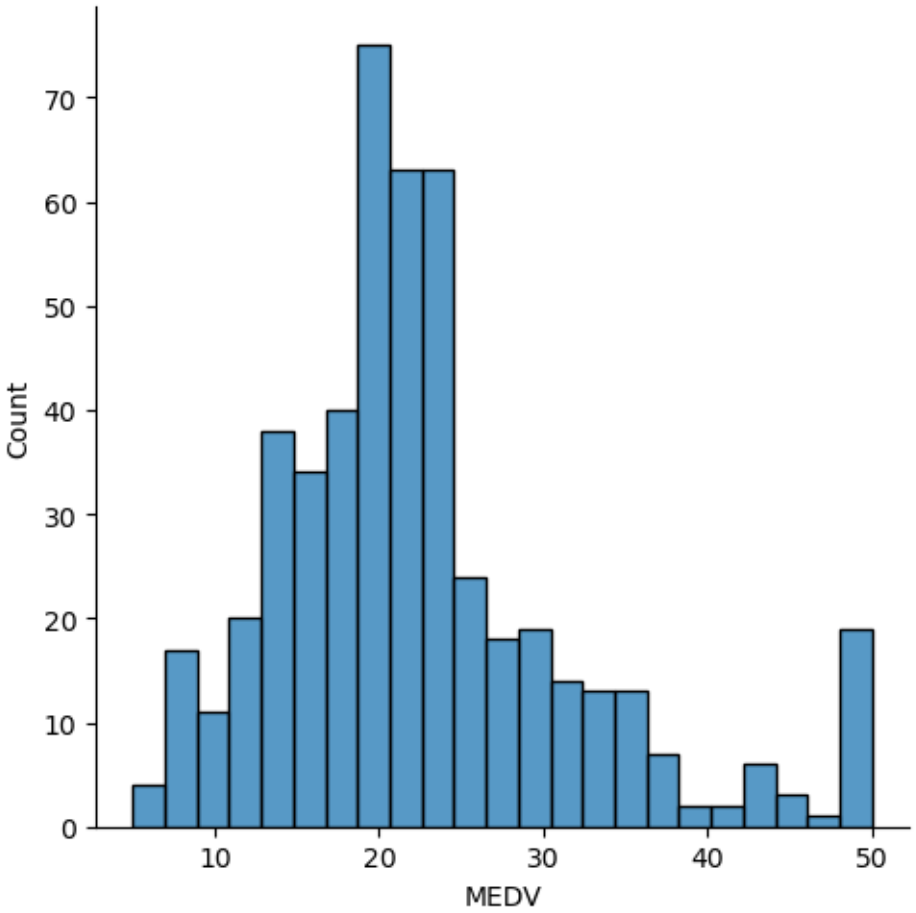
	crim	zn	indus	chas	nox	rm	\
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	
75%	3.677083	12.500000	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    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    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    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

[33]:
```
#data visualization
import seaborn as sns

sns.displot(df.MEDV)
```

[33]:
```
<seaborn.axisgrid.FacetGrid at 0x29cda3005d0>
```



[34]:
```
import seaborn as sns
import matplotlib.pyplot as plt
# Assuming 'df' is your DataFrame
correlation = df.corr()

# Creating a heatmap with correlation values annotated
fig, axes = plt.subplots(figsize=(15, 12))
sns.heatmap(correlation, square=True, annot=True)

# Adding title
plt.title('Correlation Matrix Heatmap')

# Showing the plot
plt.show()
```


Correlation Matrix Heatmap
crim 1 -0.2 0.41 -0.056 0.42 -0.22 0.35 -0.38 0.63 0.58 0.29 -0.39 0.46 -0.39
zn -0.2 1 -0.53 -0.043 -0.52 0.31 -0.57 0.66 -0.31 -0.31 -0.39 0.18 -0.41 0.36
indus 0.41 -0.53 1 0.063 0.76 -0.39 0.64 -0.71 0.6 0.72 0.38 -0.36 0.6 -0.48
chas -0.056 -0.043 0.063 1 0.091 0.091 0.087 -0.099 -0.0074 -0.036 -0.12 0.049 -0.054 0.18
nox 0.42 -0.52 0.76 0.091 1 -0.3 0.73 -0.77 0.61 0.67 0.19 -0.38 0.59 -0.43
rm -0.22 0.31 -0.39 0.091 -0.3 1 -0.24 0.21 -0.21 -0.29 -0.36 0.13 -0.61 0.7
age 0.35 -0.57 0.64 0.087 0.73 -0.24 1 -0.75 0.46 0.51 0.26 -0.27 0.6 -0.38
dis -0.38 0.66 -0.71 -0.099 -0.77 0.21 -0.75 1 -0.49 -0.53 -0.23 0.29 -0.5 0.25
rad 0.63 -0.31 0.6 -0.0074 0.61 -0.21 0.46 -0.49 1 0.91 0.46 -0.44 0.49 -0.38
tax 0.58 -0.31 0.72 -0.036 0.67 -0.29 0.51 -0.53 0.91 1 0.46 -0.44 0.54 -0.47
ptratio 0.29 -0.39 0.38 -0.12 0.19 -0.36 0.26 -0.23 0.46 0.46 1 -0.18 0.37 -0.51
b -0.39 0.18 -0.36 0.049 -0.38 0.13 -0.27 0.29 -0.44 -0.44 -0.18 1 -0.37 0.33
lstat 0.46 -0.41 0.6 -0.054 0.59 -0.61 0.6 -0.5 0.49 0.54 0.37 -0.37 1 -0.74
MEDV -0.39 0.36 -0.48 0.18 -0.43 0.7 -0.38 0.25 -0.38 -0.47 -0.51 0.33 -0.74 1
crim zn indus chas nox rm age dis rad tax ptratio b lstat MEDV
1.0 0.8 0.6 0.4 0.2 0.0 −0.2 −0.4 −0.6

[ ]:

[ ]:

[ ]: