```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#import warnings
#warnings.filterwarnings('ignore')

df = pd.read_csv('housing.csv')

# crim     Per capita crime rate by town.
# zn    Proportion of residential land zoned for lots over 25,000 sq.
ft.
# indus    Proportion of non-retail business acres per town.
# chas     Charles River dummy variable (1 if tract bounds river; 0
otherwise).
# nox      Nitric oxides concentration (parts per 10 million).
# rm  Average number of rooms per dwelling.
# age      Proportion of owner-occupied units built prior to 1940.
# dis      Weighted distances to five Boston employment centers.
# rad      Index of accessibility to radial highways.
# tax      Full-value property tax rate per $10,000.
# ptratio  Pupil-teacher ratio by town.
# b        1000(Bk - 0.63)^2, where Bk is the proportion of Black
residents by town.
# lstat    % lower status of the population.
# medv     Median value of owner-occupied homes in $1000s (target
variable).

df.head(3)

      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

        b  lstat  medv
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
```

**MEDV is the Median value of owner-occupied homes in $1000s (target variable). This is our target variable**

```python
df.shape

(506, 14)
```

```
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       501 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
```

```
df.isnull().sum()
```

```
crim       0
zn         0
indus      0
chas       0
nox        0
rm         5
age        0
dis        0
rad        0
tax        0
ptratio    0
b          0
lstat      0
medv       0
dtype: int64
```

```
df.describe()
```

```
             crim          zn       indus        chas         nox
rm  \
count  506.000000  506.000000  506.000000  506.000000  506.000000
501.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695
6.284341
```

```
std       8.601545   23.322453    6.860353    0.253994    0.115878
0.705587
min       0.006320    0.000000    0.460000    0.000000    0.385000
3.561000
25%       0.082045    0.000000    5.190000    0.000000    0.449000
5.884000
50%       0.256510    0.000000    9.690000    0.000000    0.538000
6.208000
75%       3.677083   12.500000   18.100000    0.000000    0.624000
6.625000
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
```

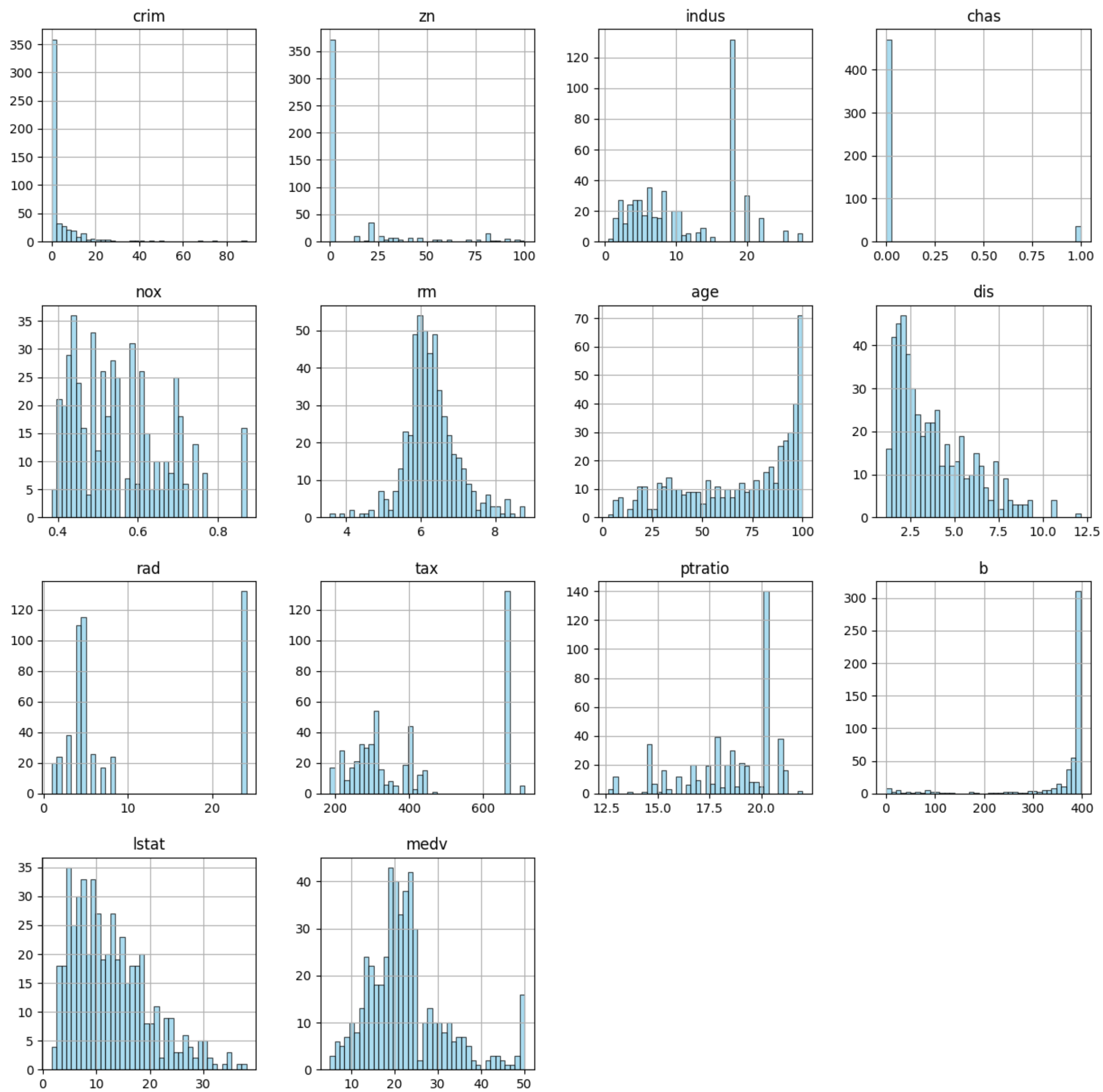
**There are 5 null values in the rm column**

**We can remove the rows with null values of rm.**

```
df['rm'].fillna(df['rm'].median(), inplace=True) #The null values has been filled with the mdeian instead
```

```
C:\Users\gaikw\AppData\Local\Temp\ipykernel_12776\2158360379.py:1:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df['rm'].fillna(df['rm'].median(), inplace=True) #The null values
has been filled with the mdeian instead
```

```
df.hist(bins=40, color="skyblue", edgecolor="black", alpha=0.7,
linewidth=0.8, figsize = (15,15))
plt.show()
```




**The extremities in the histogram plots can be used to visualize the distribution of attributes and their count**

```
df['chas'].value_counts()

chas
0    471
1     35
Name: count, dtype: int64
```

**We can see that the CHAS values has only 2 values that is 1 and 0. Also the number of 0 is very high comparatively which can lead to incorrect predictions later.We have to handle those values too**

**Every feature has different scale and some algorithms are sensitive to the scale of features, and their performance or convergence can be significantly affected**

**This might lead to optimizarion failure**

# StandardScaling

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
features = df.drop(columns=['medv'])
sc_features = sc.fit_transform(features)

sc_df = pd.DataFrame(sc_features, columns=features.columns)
sc_df['medv'] = df.medv
sc_df.head(3)
```

```
       crim        zn     indus      chas       nox        rm       
age  \
0 -0.419782  0.284830 -1.287909 -0.272599 -0.144217  0.415455 -
0.120013   
1 -0.417339 -0.487722 -0.593381 -0.272599 -0.740262  0.195904  
0.367166   
2 -0.417342 -0.487722 -0.593381 -0.272599 -0.740262  1.285105 -
0.265812   

        dis       rad       tax   ptratio         b     lstat  medv  
0  0.140214 -0.982843 -0.666608 -1.459000  0.441052 -1.075562  24.0  
1  0.557160 -0.867883 -0.987329 -0.303094  0.441052 -0.492439  21.6  
2  0.557160 -0.867883 -0.987329 -0.303094  0.396427 -1.208727  34.7  
```

**The data has been successfully standardized**

# Train_Test_Splitting

**We will use STRATIFIED SAMPLING**

```
x = sc_df.iloc[:,:-1]
y = df.medv

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 
0.2,stratify=x['chas'] ,random_state = 42)

x_train['chas'].value_counts()
```

```
chas
-0.272599    367
 3.668398     27
Name: count, dtype: int64
```

```python
x_test['chas'].value_counts()
```

```
chas
-0.272599    92
 3.668398     7
Name: count, dtype: int64
```

```python
95/7 , 376/28
```

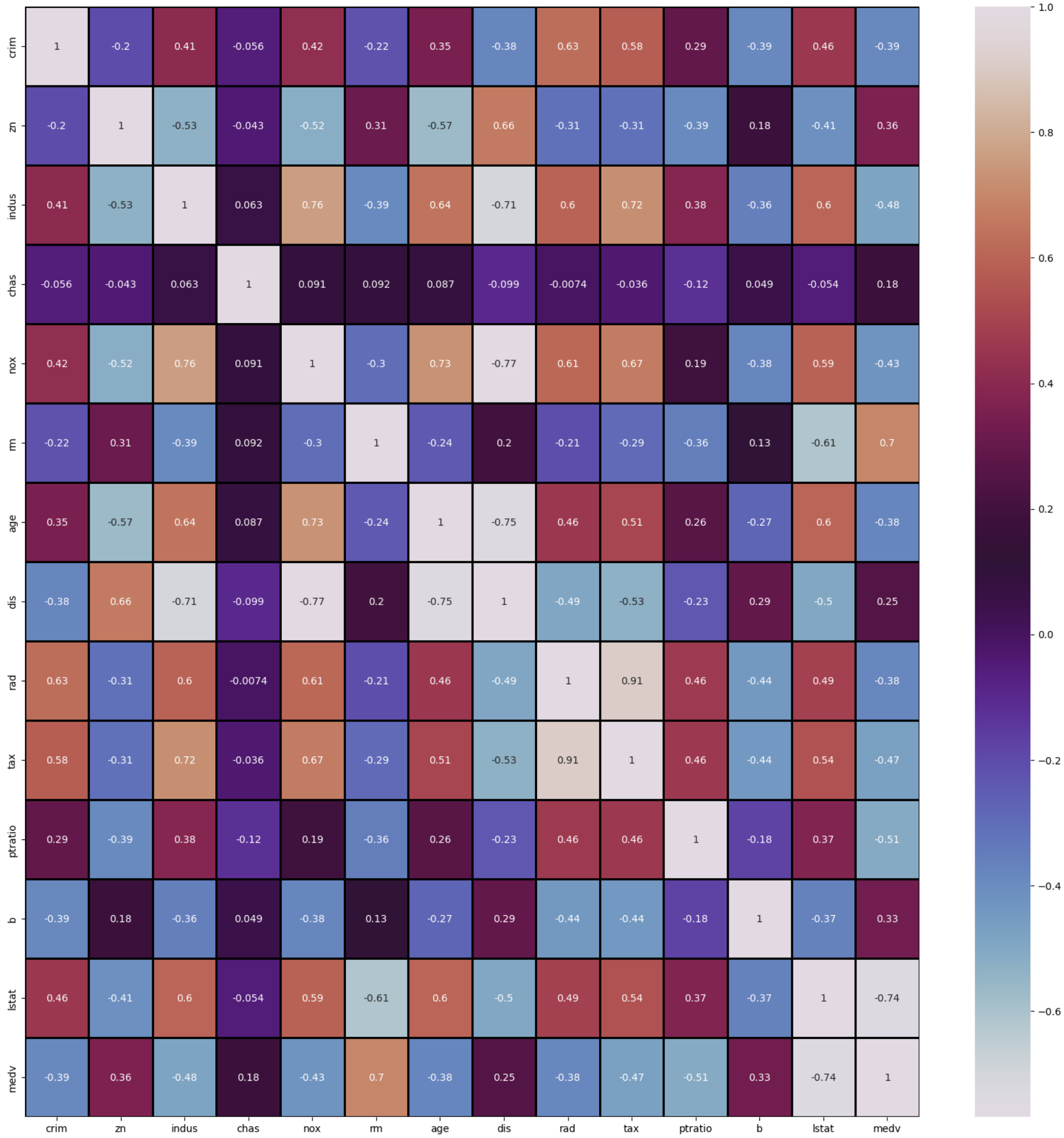
```
(13.571428571428571, 13.428571428571429)
```

**In both, the training set and the testing dataset, the proportion of the two values of CHAS feature is exactly the same. Therefore the factor of biasness is eradicated**

**Now we have to choose the model for prediction for which we need the correlations. The scatter plot can be used to do so**

```python
# import seaborn as sns
# sns.pairplot(sc_df, hue = 'medv', kind='scatter',diag_kind='hist',plot_kws={'alpha': 0.6, 's': 80},diag_kws={'bins': 10, 'color': 'blue'},height=3,corner=True)

plt.figure(figsize = (20,20))
sns.heatmap(sc_df.corr(),annot = True,linewidth= 1, linecolor= 'black', cmap = 'twilight')
```

```
<Axes: >
```




**One can see that features like rm and lstat highly affect the medv label**

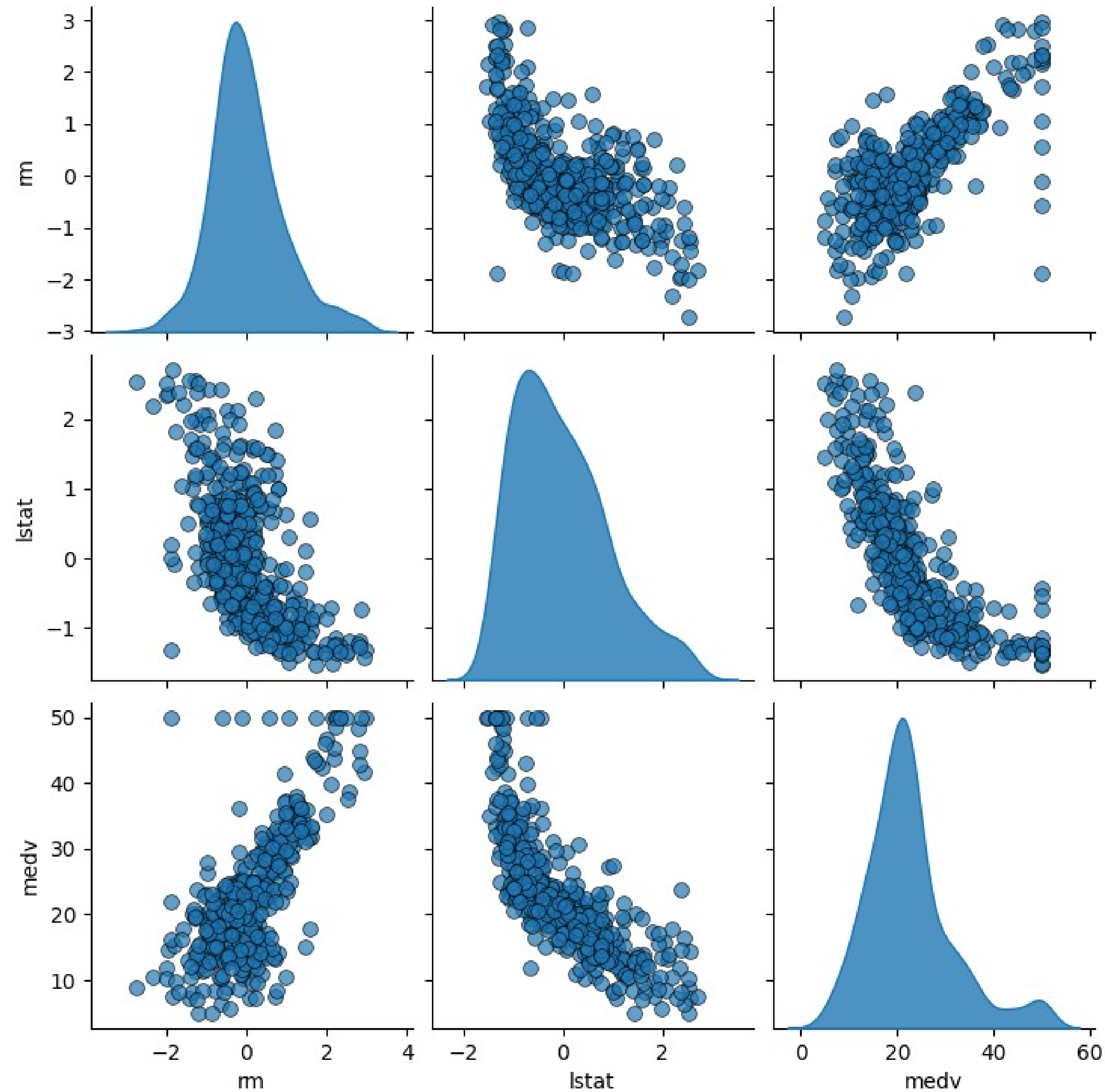
```
corr_matrix =  sc_df.corr()
corr_matrix['medv'].sort_values(ascending = False)
#This is used to directly get the correlation coefficient between the features and label

medv       1.000000
rm         0.695668
zn         0.360445
```

```
b          0.333461
dis        0.249929
chas       0.175260
age       -0.376955
rad       -0.381626
crim      -0.388305
nox       -0.427321
tax       -0.468536
indus     -0.483725
ptratio   -0.507787
lstat     -0.737663
Name: medv, dtype: float64
```

```
#Lets see the scatter plot for the two highly correlated features
against the label

pairplot = sns.pairplot(
    sc_df[['rm', 'lstat', 'medv']],
    diag_kind="kde",        # Use kernel density estimation for the
diagonal
    markers="o",            # Customize marker style
    palette="viridis",      # Use a color palette
    plot_kws={"alpha": 0.7, "s": 50, "edgecolor": "k"},  # Marker
properties
    diag_kws={"shade": True, "alpha": 0.8}  # Diagonal KDE properties
)
plt.show()
#I have used ChatGpt to make the grapph fancier
```

```
C:\Users\gaikw\AppData\Local\Programs\Python\Python313\Lib\site-
packages\seaborn\axisgrid.py:1513: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(x=vector, **plot_kwargs)
C:\Users\gaikw\AppData\Local\Programs\Python\Python313\Lib\site-
packages\seaborn\axisgrid.py:1513: UserWarning: Ignoring `palette`
because no `hue` variable has been assigned.
  func(x=vector, **plot_kwargs)
C:\Users\gaikw\AppData\Local\Programs\Python\Python313\Lib\site-
packages\seaborn\axisgrid.py:1513: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(x=vector, **plot_kwargs)
C:\Users\gaikw\AppData\Local\Programs\Python\Python313\Lib\site-
packages\seaborn\axisgrid.py:1513: UserWarning: Ignoring `palette`
because no `hue` variable has been assigned.
```

```
  func(x=vector, **plot_kwargs)
C:\Users\gaikw\AppData\Local\Programs\Python\Python313\Lib\site-
packages\seaborn\axisgrid.py:1513: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(x=vector, **plot_kwargs)
C:\Users\gaikw\AppData\Local\Programs\Python\Python313\Lib\site-
packages\seaborn\axisgrid.py:1513: UserWarning: Ignoring `palette`
because no `hue` variable has been assigned.
  func(x=vector, **plot_kwargs)
C:\Users\gaikw\AppData\Local\Programs\Python\Python313\Lib\site-
packages\seaborn\axisgrid.py:1615: UserWarning: Ignoring `palette`
because no `hue` variable has been assigned.
  func(x=x, y=y, **kwargs)
C:\Users\gaikw\AppData\Local\Programs\Python\Python313\Lib\site-
packages\seaborn\axisgrid.py:1615: UserWarning: Ignoring `palette`
because no `hue` variable has been assigned.
  func(x=x, y=y, **kwargs)
C:\Users\gaikw\AppData\Local\Programs\Python\Python313\Lib\site-
packages\seaborn\axisgrid.py:1615: UserWarning: Ignoring `palette`
because no `hue` variable has been assigned.
  func(x=x, y=y, **kwargs)
C:\Users\gaikw\AppData\Local\Programs\Python\Python313\Lib\site-
packages\seaborn\axisgrid.py:1615: UserWarning: Ignoring `palette`
because no `hue` variable has been assigned.
  func(x=x, y=y, **kwargs)
C:\Users\gaikw\AppData\Local\Programs\Python\Python313\Lib\site-
packages\seaborn\axisgrid.py:1615: UserWarning: Ignoring `palette`
because no `hue` variable has been assigned.
  func(x=x, y=y, **kwargs)
C:\Users\gaikw\AppData\Local\Programs\Python\Python313\Lib\site-
packages\seaborn\axisgrid.py:1615: UserWarning: Ignoring `palette`
because no `hue` variable has been assigned.
  func(x=x, y=y, **kwargs)
```

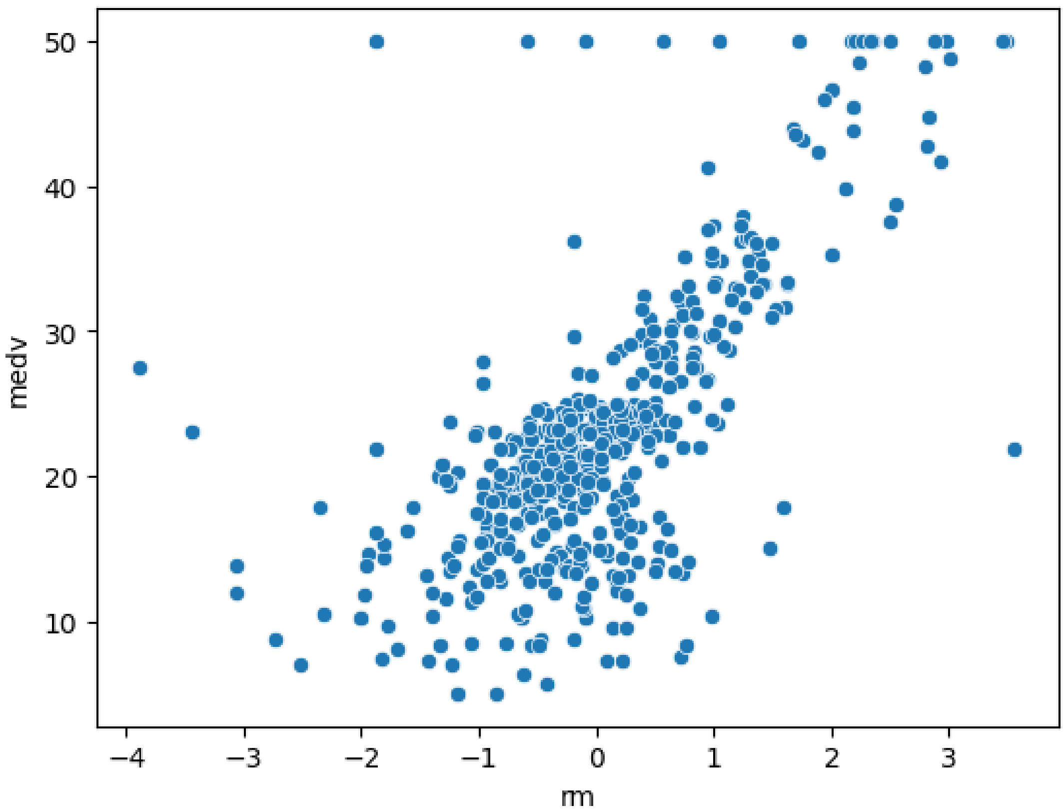



**medv and lstat has a very strong negative correlation. Also they are relation is linear ****

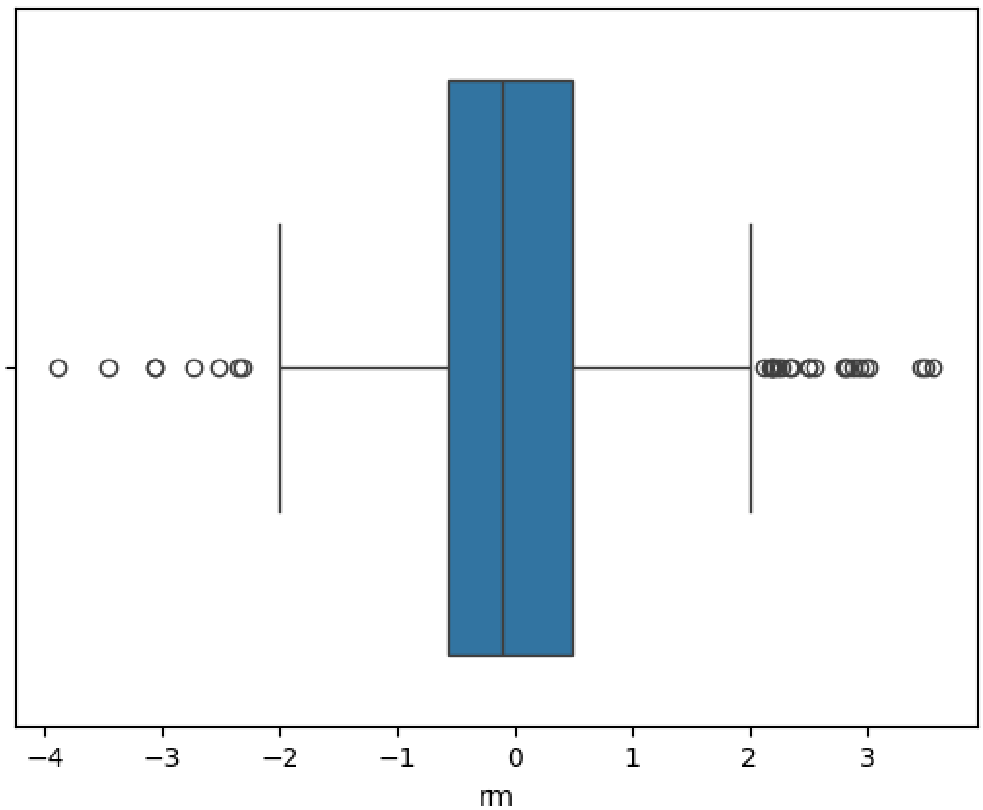
**There are few mistakes in the data which are apparent from the scatter plot. In the medv vs rm plot one can see capping around 50 in y direction**

```
sns.scatterplot(x = 'rm', y = 'medv', data = sc_df)
```

```
<Axes: xlabel='rm', ylabel='medv'>
```




```
sns.boxplot(x = 'rm', data = sc_df)
plt.show()
```

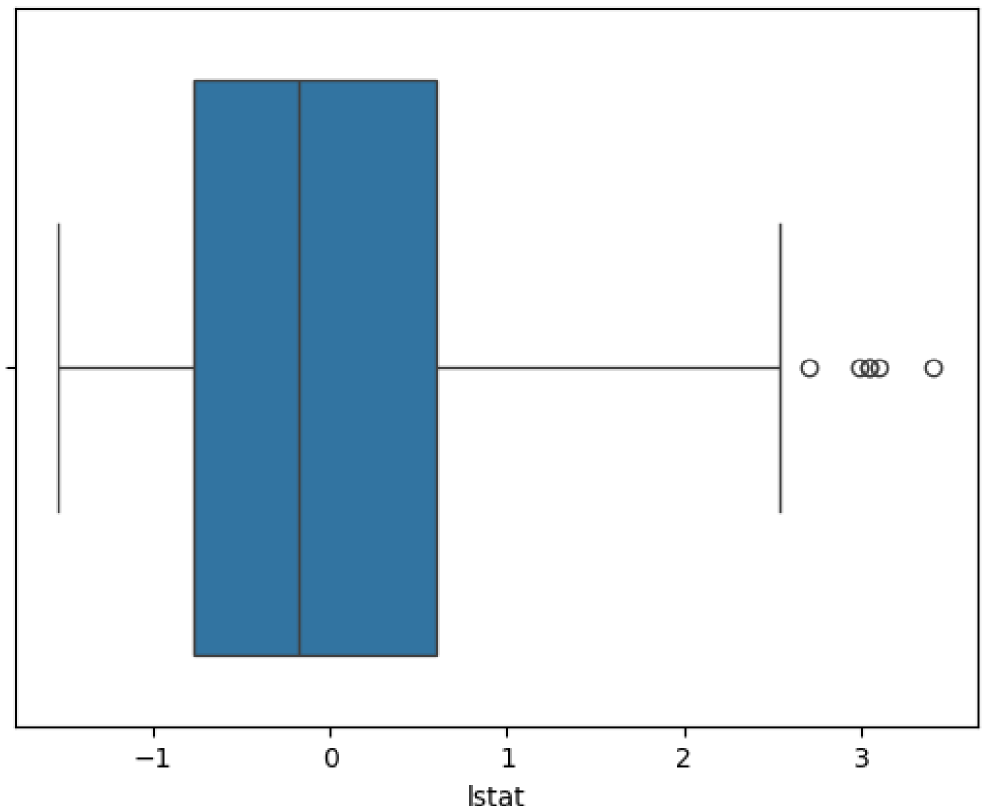



```
#We are now going to remove outliers using the Z_score Method

sc_df = sc_df[((sc_df['rm'] - sc_df['rm'].mean()) /
sc_df['rm'].std()).abs() <=3]

sc_df.shape
```

```
(498, 14)
```

```
sns.boxplot(x = 'lstat', data = sc_df)
#The above statment has been compiled again just to check whether thhe
outliers has been removed successfully or not
```

```
<Axes: xlabel='lstat'>
```




```
sc_df = sc_df[((sc_df['lstat'] - sc_df['lstat'].mean()) /
sc_df['lstat'].std()).abs() <=3]

sc_df.describe()
```

```
                crim          zn       indus        chas         nox
rm  \
count  493.000000  493.000000  493.000000  493.000000  493.000000
493.000000
mean    -0.025121    0.009379   -0.016027   -0.000806   -0.022830
0.018428
std      0.974941    1.011183    0.999406    0.999645    1.000666
0.896147
min     -0.419782   -0.487722   -1.557842   -0.272599   -1.465882    -
2.730967
25%     -0.411233   -0.487722   -0.876445   -0.272599   -0.921667    -
0.559694
50%     -0.392690   -0.487722   -0.375976   -0.272599   -0.196047    -
0.107761
75%     -0.052359    0.048772    1.015999   -0.272599    0.598679
0.476758
max      9.933931    3.804234    2.422565    3.668398    2.732346
2.981635

                 age         dis         rad         tax     ptratio
```

```
b  \
count  493.000000  493.000000  493.000000  493.000000  493.000000
493.000000
mean    -0.023704    0.026383   -0.026550   -0.023600   -0.006075
0.019807
std      1.002461    1.000002    0.987554    0.990579    0.993427
0.974518
min     -2.335437   -1.267069   -0.982843   -1.313990   -2.707379   -
3.907193
25%     -0.895234   -0.795218   -0.637962   -0.767576   -0.534275
0.213103
50%      0.281821   -0.227009   -0.523001   -0.464673    0.251741
0.384147
75%      0.897019    0.682657   -0.178120    1.530926    0.806576
0.433706
max      1.117494    3.960518    1.661245    1.798194    1.638828
0.441052

            lstat        medv
count  493.000000  493.000000
mean    -0.032064   22.501217
std      0.942373    8.982521
min     -1.531127    5.000000
25%     -0.791010   17.100000
50%     -0.186861   21.200000
75%      0.560266   25.000000
max      2.710532   50.000000
```

```
sc_df.shape
```

```
(493, 14)
```

## Selecting a model

```
x = sc_df.iloc[:,:-1]
y = sc_df.medv
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size =
0.2,stratify=x['chas'] ,random_state = 42)


from sklearn.linear_model import LinearRegression, Lasso, Ridge
lr = LinearRegression()
lr.fit(x_train,y_train)
```

```
LinearRegression()
```

```
y_pred = lr.predict(x_test)
```

```
#Our model has predicted the above values
#Now let us check the accuracy of the model

lr.score(x_test,y_test)

0.6699809447404514

lr.score(x_train,y_train)

0.7849229826177202

from sklearn.metrics import mean_squared_error
lin_mse = mean_squared_error(y_test , y_pred)

lin_mse

27.94672542499108

#We can use some other regression model to get a better accuracy

lr1 = Lasso()
lr1.fit(x_train,y_train)
lr1.score(x_test,y_test) , lr1.score(x_train,y_train)

(0.6606225212953002, 0.7079411756985744)

rr1 = Ridge()
rr1.fit(x_train,y_train)
rr1.score(x_test,y_test) , rr1.score(x_train,y_train)

(0.6711332985708123, 0.7849024732394349)

#Now we can try the DecisionTreeRgressor

from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor()
dtr.fit(x_train,y_train)
dtr.score(x_test,y_test) , dtr.score(x_train,y_train)

(0.4486608802233316, 1.0)
```

# Overfitting has occured

```
##Our model has learned the noise and not the train
```

# Using cross-validation

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(dtr, x_train, y_train, scoring
='neg_mean_squared_error')
```

```
mse_scores = np.sqrt(-scores)
mse_scores
```

```
array([3.72771013, 4.18458594, 4.41686927, 7.34993326, 4.27426707])
```

```
#The error is very less as compared to the previos error
```

## Using The Random Forest Algorithm

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor()
rfr.fit(x_train,y_train)
rfr.score(x_test,y_test) , rfr.score(x_train,y_train)
```

```
(0.8789832986625947, 0.9828439832376973)
```

```
scores = cross_val_score(rfr, x_train, y_train, scoring
='neg_mean_squared_error')
mse_scores = np.sqrt(-scores)
mse_scores
```

```
array([3.19473328, 3.5503786 , 3.67503746, 2.76150563, 3.24468773])
```

**Random Forest Algorithm has the best score and we chose this model for our House Price Prediction**