boston

February 2, 2022

https://github.com/selva86/datasets/blob/master/BostonHousing.csv

Raw Data : https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.csv

[ ]:
```
!pip3 install numpy
!pip3 install pandas
!pip3 install matplotlib
!pip3 install seaborn
!pip3 install sklearn
```

```
Requirement already satisfied: numpy in ./assign4_venv/lib/python3.8/site-
packages (1.22.1)
Requirement already satisfied: pandas in ./assign4_venv/lib/python3.8/site-
packages (1.4.0)
Requirement already satisfied: python-dateutil>=2.8.1 in
./assign4_venv/lib/python3.8/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
./assign4_venv/lib/python3.8/site-packages (from pandas) (2021.3)
Requirement already satisfied: numpy>=1.18.5; platform_machine != "aarch64" and
platform_machine != "arm64" and python_version < "3.10" in
./assign4_venv/lib/python3.8/site-packages (from pandas) (1.22.1)
Requirement already satisfied: six>=1.5 in ./assign4_venv/lib/python3.8/site-
packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
Requirement already satisfied: matplotlib in ./assign4_venv/lib/python3.8/site-
packages (3.5.1)
Requirement already satisfied: fonttools>=4.22.0 in
./assign4_venv/lib/python3.8/site-packages (from matplotlib) (4.29.0)
Requirement already satisfied: packaging>=20.0 in
./assign4_venv/lib/python3.8/site-packages (from matplotlib) (21.3)
Requirement already satisfied: cycler>=0.10 in
./assign4_venv/lib/python3.8/site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: python-dateutil>=2.7 in
./assign4_venv/lib/python3.8/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in
./assign4_venv/lib/python3.8/site-packages (from matplotlib) (9.0.0)
Requirement already satisfied: pyparsing>=2.2.1 in
./assign4_venv/lib/python3.8/site-packages (from matplotlib) (3.0.7)
Requirement already satisfied: kiwisolver>=1.0.1 in
./assign4_venv/lib/python3.8/site-packages (from matplotlib) (1.3.2)
```

```
Requirement already satisfied: numpy>=1.17 in ./assign4_venv/lib/python3.8/site-
packages (from matplotlib) (1.22.1)
Requirement already satisfied: six>=1.5 in ./assign4_venv/lib/python3.8/site-
packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Requirement already satisfied: seaborn in ./assign4_venv/lib/python3.8/site-
packages (0.11.2)
Requirement already satisfied: pandas>=0.23 in
./assign4_venv/lib/python3.8/site-packages (from seaborn) (1.4.0)
Requirement already satisfied: matplotlib>=2.2 in
./assign4_venv/lib/python3.8/site-packages (from seaborn) (3.5.1)
Requirement already satisfied: numpy>=1.15 in ./assign4_venv/lib/python3.8/site-
packages (from seaborn) (1.22.1)
Requirement already satisfied: scipy>=1.0 in ./assign4_venv/lib/python3.8/site-
packages (from seaborn) (1.7.3)
Requirement already satisfied: python-dateutil>=2.8.1 in
./assign4_venv/lib/python3.8/site-packages (from pandas>=0.23->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
./assign4_venv/lib/python3.8/site-packages (from pandas>=0.23->seaborn) (2021.3)
Requirement already satisfied: pyparsing>=2.2.1 in
./assign4_venv/lib/python3.8/site-packages (from matplotlib>=2.2->seaborn)
(3.0.7)
Requirement already satisfied: packaging>=20.0 in
./assign4_venv/lib/python3.8/site-packages (from matplotlib>=2.2->seaborn)
(21.3)
Requirement already satisfied: kiwisolver>=1.0.1 in
./assign4_venv/lib/python3.8/site-packages (from matplotlib>=2.2->seaborn)
(1.3.2)
Requirement already satisfied: fonttools>=4.22.0 in
./assign4_venv/lib/python3.8/site-packages (from matplotlib>=2.2->seaborn)
(4.29.0)
Requirement already satisfied: pillow>=6.2.0 in
./assign4_venv/lib/python3.8/site-packages (from matplotlib>=2.2->seaborn)
(9.0.0)
Requirement already satisfied: cycler>=0.10 in
./assign4_venv/lib/python3.8/site-packages (from matplotlib>=2.2->seaborn)
(0.11.0)
Requirement already satisfied: six>=1.5 in ./assign4_venv/lib/python3.8/site-
packages (from python-dateutil>=2.8.1->pandas>=0.23->seaborn) (1.16.0)
Requirement already satisfied: sklearn in ./assign4_venv/lib/python3.8/site-
packages (0.0)
Requirement already satisfied: scikit-learn in
./assign4_venv/lib/python3.8/site-packages (from sklearn) (1.0.2)
Requirement already satisfied: scipy>=1.1.0 in
./assign4_venv/lib/python3.8/site-packages (from scikit-learn->sklearn) (1.7.3)
Requirement already satisfied: numpy>=1.14.6 in
./assign4_venv/lib/python3.8/site-packages (from scikit-learn->sklearn) (1.22.1)
Requirement already satisfied: joblib>=0.11 in
./assign4_venv/lib/python3.8/site-packages (from scikit-learn->sklearn) (1.1.0)
```

```
Requirement already satisfied: threadpoolctl>=2.0.0 in
./assign4_venv/lib/python3.8/site-packages (from scikit-learn->sklearn) (3.0.0)
```

[ ]:
```
import numpy as NP
import pandas as PD
import seaborn as SNS
import matplotlib.pyplot as MPLOT
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
```

[ ]:
```
DF = PD.read_csv("https://raw.githubusercontent.com/selva86/datasets/master/
 ↪BostonHousing.csv")
print("Shape :",DF.shape)
print("Size :",DF.size)
DF.head(10)
```

```
Shape : (506, 14)
Size : 7084
```

[ ]:
```
      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
5  0.02985   0.0   2.18     0  0.458  6.430   58.7  6.0622    3  222     18.7
6  0.08829  12.5   7.87     0  0.524  6.012   66.6  5.5605    5  311     15.2
7  0.14455  12.5   7.87     0  0.524  6.172   96.1  5.9505    5  311     15.2
8  0.21124  12.5   7.87     0  0.524  5.631  100.0  6.0821    5  311     15.2
9  0.17004  12.5   7.87     0  0.524  6.004   85.9  6.5921    5  311     15.2

        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
5  394.12   5.21  28.7
6  395.60  12.43  22.9
7  396.90  19.15  27.1
8  386.63  29.93  16.5
9  386.71  17.10  18.9
```

[ ]:
```
DF.dtypes
```

[ ]:
```
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
```

[ ]:
```
DF.isna().sum()
```

[ ]:
```
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
```

[ ]:
```
DF.isnull().sum()
```

[ ]:
```
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
```

---

1 Corelation Matrix

[ ]:
```
DF.corr()
```

[ ]:
```
             crim        zn     indus      chas       nox        rm       age  \
crim     1.000000 -0.200469  0.406583 -0.055892  0.420972 -0.219247  0.352734
zn      -0.200469  1.000000 -0.533828 -0.042697 -0.516604  0.311991 -0.569537
indus    0.406583 -0.533828  1.000000  0.062938  0.763651 -0.391676  0.644779
chas    -0.055892 -0.042697  0.062938  1.000000  0.091203  0.091251  0.086518
nox      0.420972 -0.516604  0.763651  0.091203  1.000000 -0.302188  0.731470
rm      -0.219247  0.311991 -0.391676  0.091251 -0.302188  1.000000 -0.240265
age      0.352734 -0.569537  0.644779  0.086518  0.731470 -0.240265  1.000000
dis     -0.379670  0.664408 -0.708027 -0.099176 -0.769230  0.205246 -0.747881
rad      0.625505 -0.311948  0.595129 -0.007368  0.611441 -0.209847  0.456022
tax      0.582764 -0.314563  0.720760 -0.035587  0.668023 -0.292048  0.506456
ptratio  0.289946 -0.391679  0.383248 -0.121515  0.188933 -0.355501  0.261515
b       -0.385064  0.175520 -0.356977  0.048788 -0.380051  0.128069 -0.273534
lstat    0.455621 -0.412995  0.603800 -0.053929  0.590879 -0.613808  0.602339
medv    -0.388305  0.360445 -0.483725  0.175260 -0.427321  0.695360 -0.376955

              dis       rad       tax   ptratio         b     lstat      medv
crim    -0.379670  0.625505  0.582764  0.289946 -0.385064  0.455621 -0.388305
zn       0.664408 -0.311948 -0.314563 -0.391679  0.175520 -0.412995  0.360445
indus   -0.708027  0.595129  0.720760  0.383248 -0.356977  0.603800 -0.483725
chas    -0.099176 -0.007368 -0.035587 -0.121515  0.048788 -0.053929  0.175260
nox     -0.769230  0.611441  0.668023  0.188933 -0.380051  0.590879 -0.427321
rm       0.205246 -0.209847 -0.292048 -0.355501  0.128069 -0.613808  0.695360
age     -0.747881  0.456022  0.506456  0.261515 -0.273534  0.602339 -0.376955
dis      1.000000 -0.494588 -0.534432 -0.232471  0.291512 -0.496996  0.249929
rad     -0.494588  1.000000  0.910228  0.464741 -0.444413  0.488676 -0.381626
tax     -0.534432  0.910228  1.000000  0.460853 -0.441808  0.543993 -0.468536
ptratio -0.232471  0.464741  0.460853  1.000000 -0.177383  0.374044 -0.507787
b        0.291512 -0.444413 -0.441808 -0.177383  1.000000 -0.366087  0.333461
lstat   -0.496996  0.488676  0.543993  0.374044 -0.366087  1.000000 -0.737663
medv     0.249929 -0.381626 -0.468536 -0.507787  0.333461 -0.737663  1.000000
```

[ ]:
```
fig, ax = MPLOT.subplots(figsize=(20,15))
SNS.heatmap(DF.corr(), annot = True, ax=ax)
```

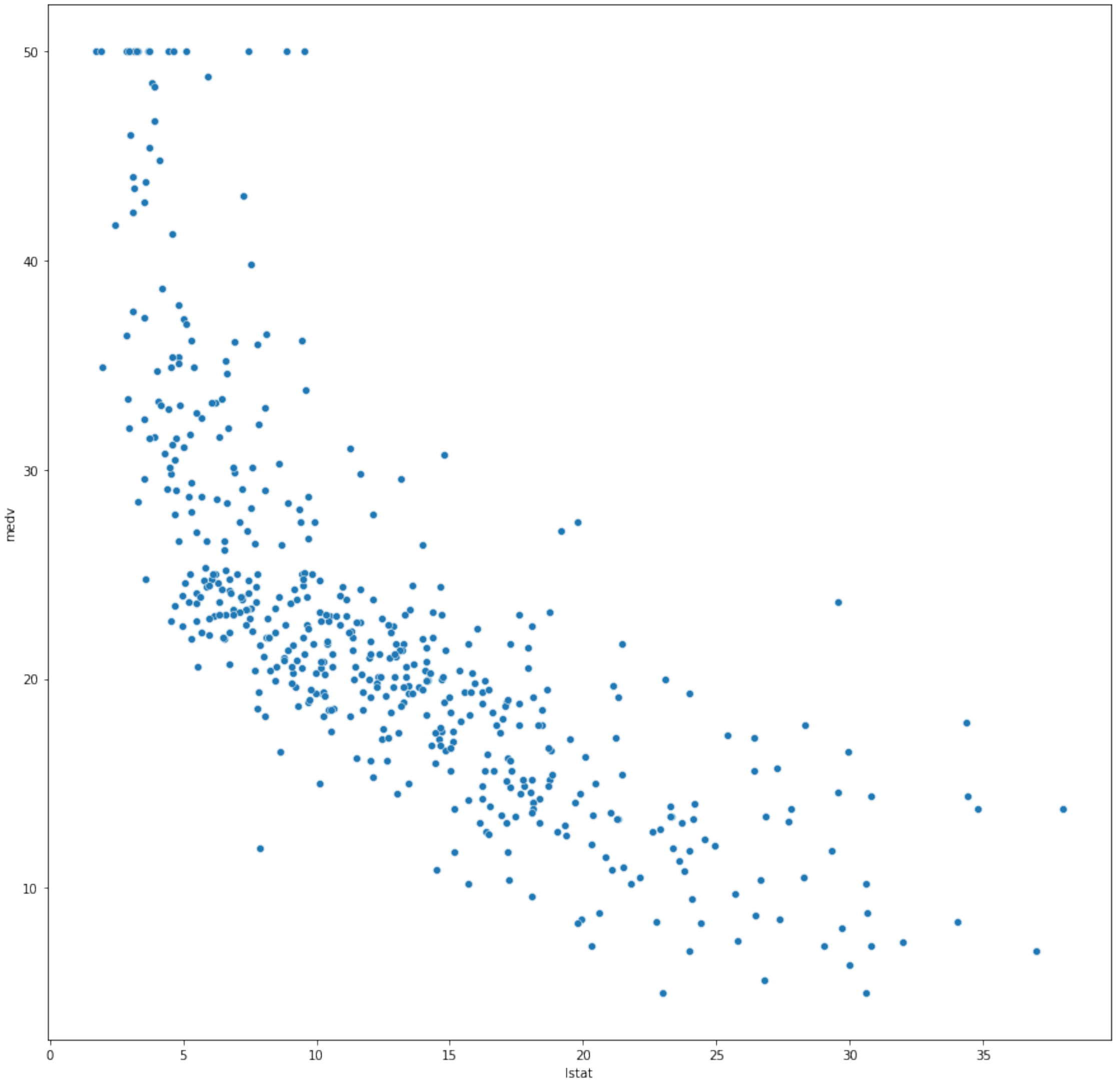
[ ]:
```
<AxesSubplot:>
```

---

2 Plotting Scatterplot map

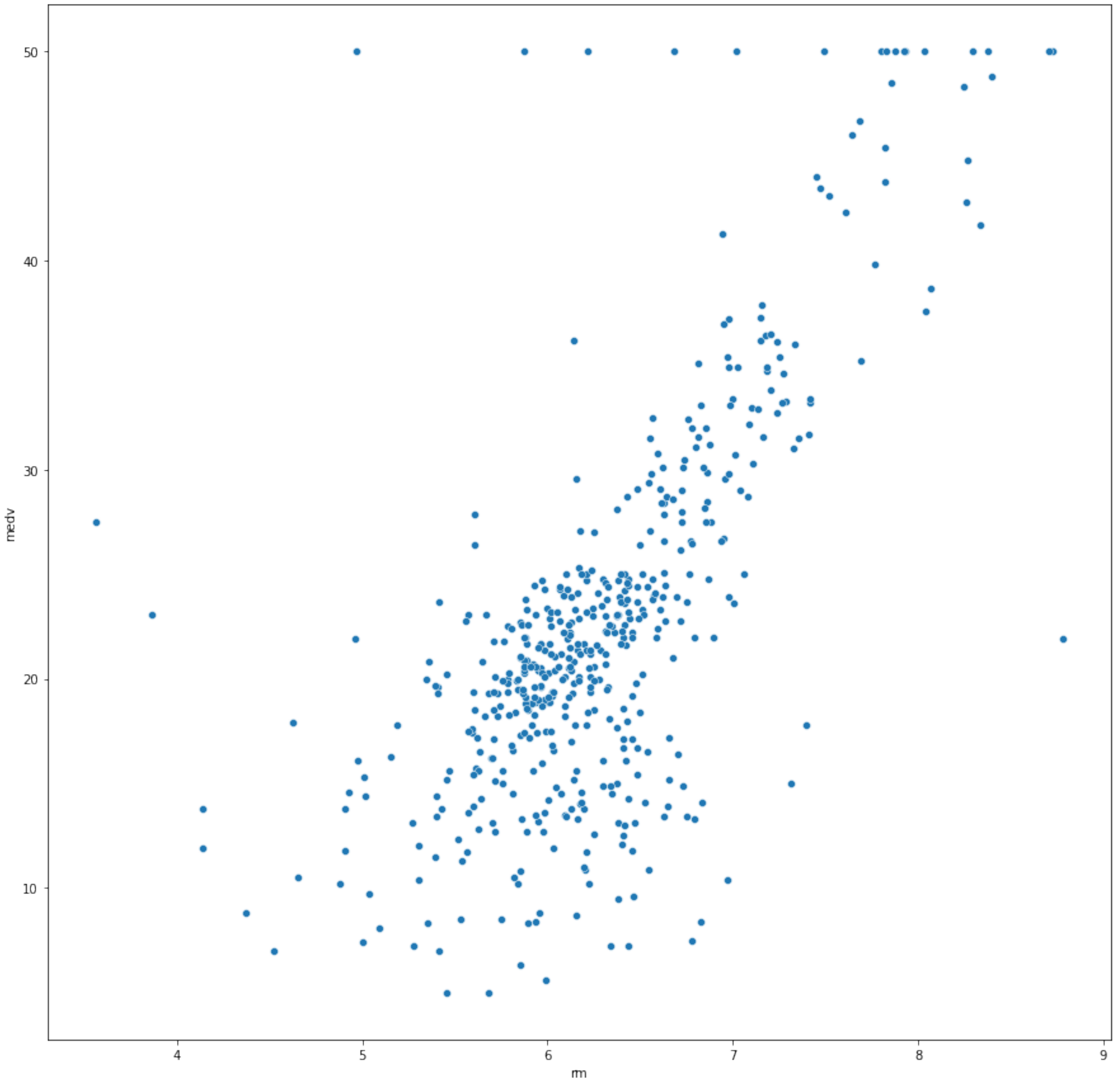
[ ]:
```
fig, ax = MPLOT.subplots(figsize=(15,15))
SNS.scatterplot(
    x=DF['lstat'],
    y=DF['medv'],
    ax=ax)
```

[ ]: <AxesSubplot:xlabel='lstat', ylabel='medv'>



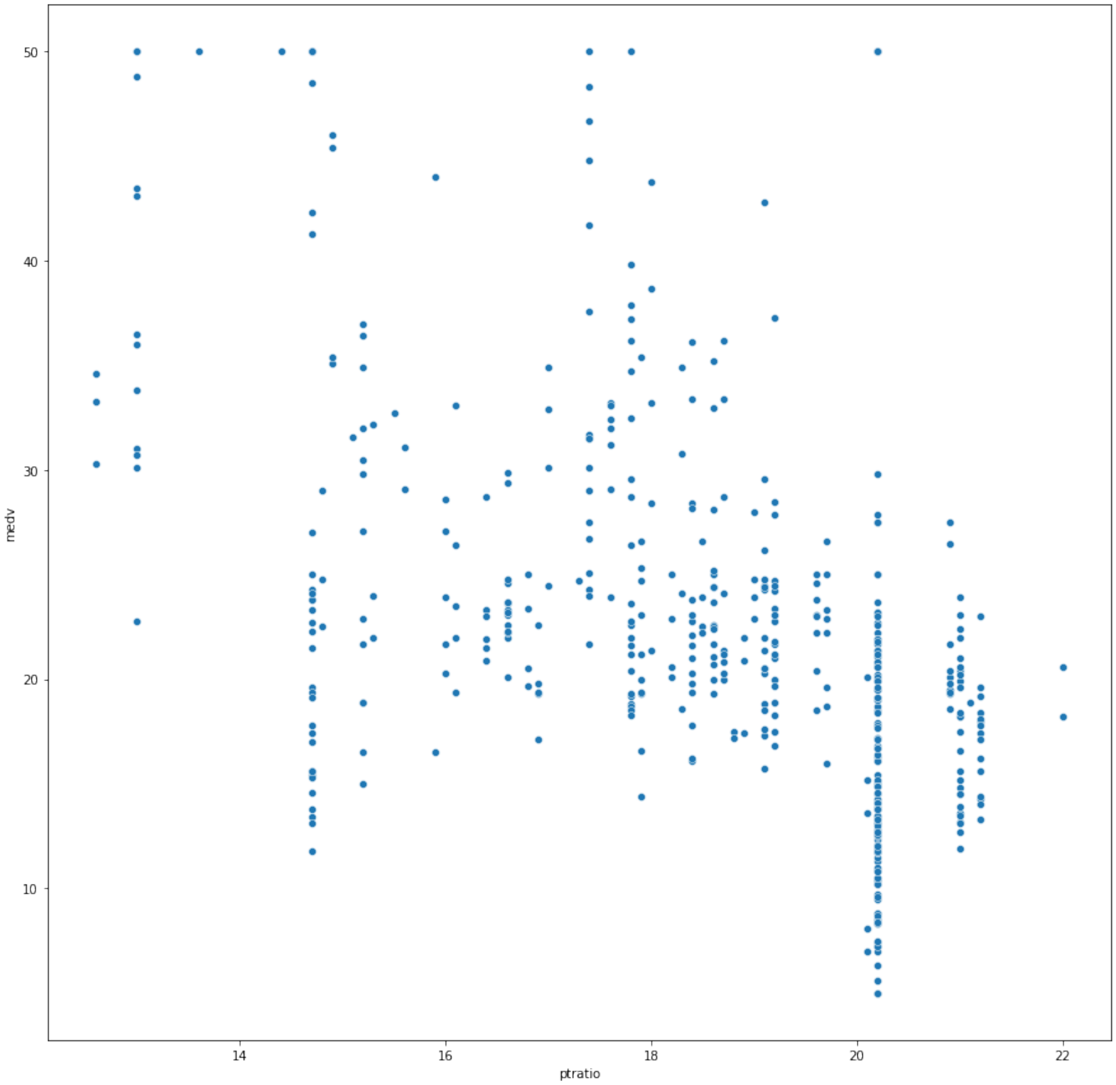
[ ]:
```
fig, ax = MPLOT.subplots(figsize=(15,15))
SNS.scatterplot(
    x=DF['rm'],
    y=DF['medv'],
    ax=ax)
```

[ ]: `<AxesSubplot:xlabel='rm', ylabel='medv'>`



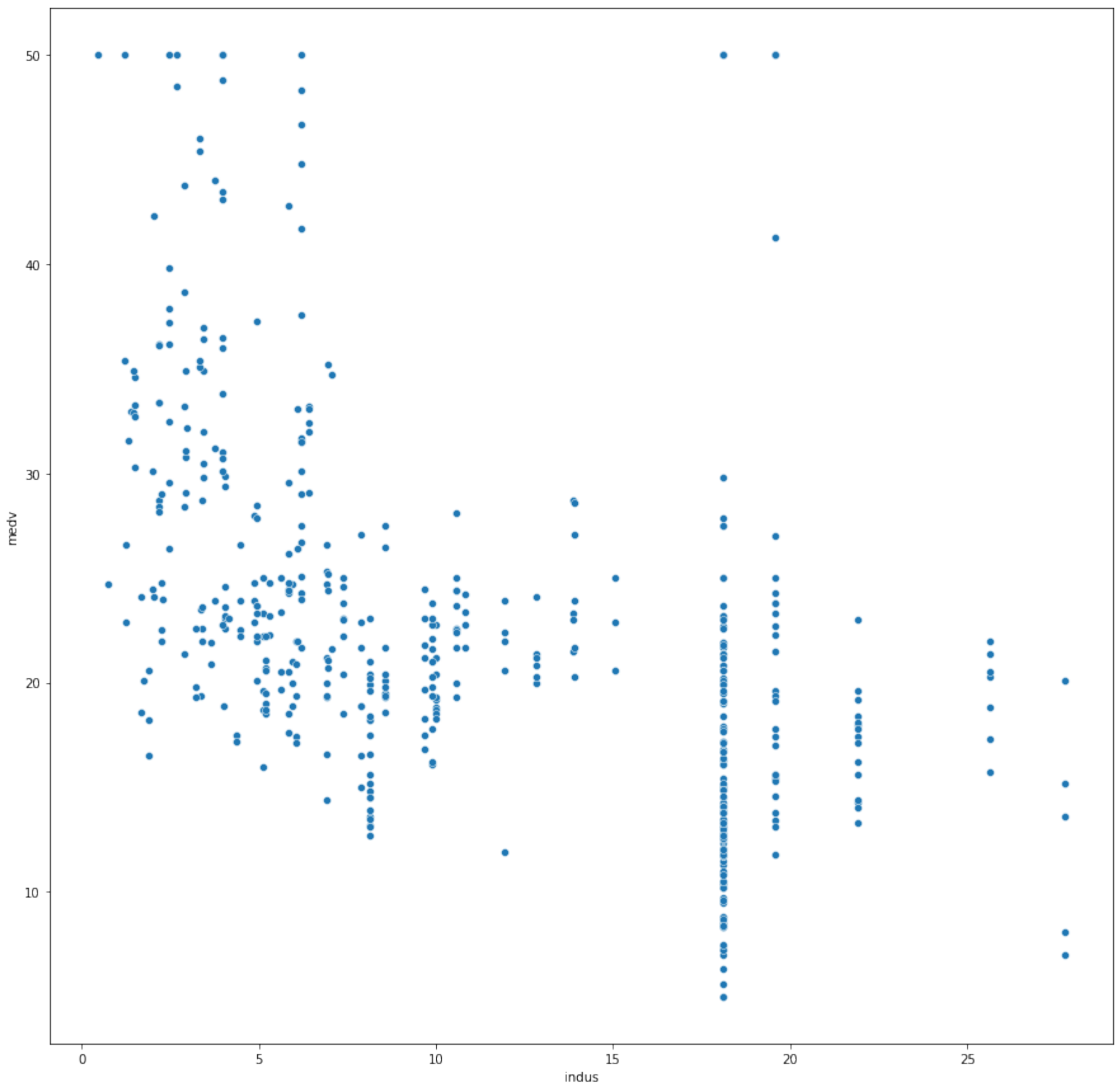
[ ]:
```
fig, ax = MPLOT.subplots(figsize=(15,15))
SNS.scatterplot(
    x=DF['ptratio'],
    y=DF['medv'],
    ax=ax)
```

[ ]: <AxesSubplot:xlabel='ptratio', ylabel='medv'>




```
[ ]: fig, ax = MPLOT.subplots(figsize=(15,15))
     SNS.scatterplot(
         x=DF['indus'],
         y=DF['medv'],
         ax=ax)
```

```
[ ]: <AxesSubplot:xlabel='indus', ylabel='medv'>
```



2.0.1 Training Data Models

[ ]:
```
X = DF[['rm','lstat']]
Y = DF['medv']

X_TRAIN, X_TEST, Y_TRAIN, Y_TEST = train_test_split(
    X,
    Y,
    test_size=0.2,
    random_state=5)
```

[ ]:
```
print('X : ',X.shape)
print('Y : ',Y.shape)
print('X_TRAIN : ',X_TRAIN.shape)
print('X_TEST : ',X_TEST.shape)
print('Y_TRAIN : ',Y_TRAIN.shape)
print('Y_TEST : ',Y_TEST.shape)
```

```
X :  (506, 2)
Y :  (506,)
X_TRAIN :  (404, 2)
X_TEST :  (102, 2)
Y_TRAIN :  (404,)
Y_TEST :  (102,)
```

[ ]:
```
model = LinearRegression().fit(X_TRAIN,Y_TRAIN)
output = model.predict(X_TEST)
output
```

[ ]:
```
array([37.38999403, 29.79290611, 25.86755297,  0.31370828, 33.31385559,
        7.97136102, 30.7066387 , 27.83076842, 26.26693081, 21.97871678,
       32.33149332, 23.21791374, 22.9932889 , 30.97465356, 27.19747687,
       20.7171544 , -0.67524986, 18.01248654, 12.3108109 , 21.90615827,
        4.82262227, 24.00423026, 37.70279396, 24.59521859, 29.6355729 ,
       12.5396288 , 27.07081337, 22.44485896, 27.64895322, 28.99223597,
       11.38689571, 10.39119661, 18.02726641, 24.65916571, 26.3259374 ,
       22.97547671, 26.32610451, 12.30204538, 37.03909693, 33.55198947,
       20.2779501 ,  1.0905118 , 27.65786778, 16.52789139, 27.49181818,
       29.91634422, -3.04746229, 17.23365847, 20.71953914, 13.74285813,
       20.74965837, 21.48012369, 25.17310326, 16.12470269, 17.61200383,
       27.89189158, 36.0647476 , 19.67862758, 28.88714637, 20.4560256 ,
       20.11858445, 23.1131674 , 16.53445226, 31.30827991, 22.62162748,
       13.10525045, 23.36377939, 25.90474345, 23.00735629, 21.62430016,
       19.09598641, 26.54344302, 16.82687517, 19.99592881, 19.77353574,
       30.38611207, 19.34927447, 13.03847313, 28.29385627, 19.03960282,
       22.09279603, 38.79116551, 15.63348401, 20.34679092, 21.95060008,
       18.33664278, 17.46437785,  7.42994531, 17.94788281, 24.15298313,
       35.48406888, 20.19265715, 21.92066094, 19.37880824, 26.53637178,
       28.83077576, 16.74210556, 30.35752395, 17.8896288 , 15.35541079,
       23.680157  , 25.43175021])
```

[ ]:
```
Y_TEST.head(10)
```

[ ]:
```
226    37.6
292    27.9
90     22.6
373    13.8
273    35.2
```

```
417     10.4
503     23.9
234     29.0
111     22.8
472     23.2
Name: medv, dtype: float64
```

[ ]:
```
(output-Y_TEST).head(10)
```

[ ]:
```
226     -0.210006
292      1.892906
90       3.267553
373    -13.486292
273     -1.886144
417     -2.428639
503      6.806639
234     -1.169232
111      3.466931
472     -1.221283
Name: medv, dtype: float64
```

[ ]:
```
print("Mean Absolute Error :", mean_absolute_error(Y_TEST, output))
print("Model Score (Accuracy) :", model.score(X_TEST, Y_TEST))
```

```
Mean Absolute Error : 3.791310213343104
Model Score (Accuracy) : 0.6628996975186952
```

Create Custom Linear Regression Function

[ ]:
```
def LinearRegressionModel(X_Train, X_Test, Y_Train, Y_Test):
    model = LinearRegression()
    model.fit(X_Train, Y_Train)
    output = model.predict(X_Test)
    print("Mean Absolute Error :", mean_absolute_error(Y_Test, output))
    print("Model Accuracy :", model.score(X_Test, Y_Test))
```

[ ]:
```
def Quick_Model_Traier(Input_Data, Output_Data):
    print("-------------------")
    print("Input : ", Input_Data.columns.values)
    X_TRAIN, X_TEST, Y_TRAIN, Y_TEST = train_test_split( Input_Data,␣
 ↪Output_Data, test_size=0.2, random_state=42)
    LinearRegressionModel(X_TRAIN, X_TEST, Y_TRAIN, Y_TEST)
```

[ ]:
```
print("Without Outliers Removed and Data Normalized\n")

Quick_Model_Traier(DF[['rm','lstat']], DF['medv'])
Quick_Model_Traier(DF[['rm','indus']], DF['medv'])
Quick_Model_Traier(DF[['rm','ptratio']], DF['medv'])
```

```
Quick_Model_Traier(DF[['lstat','indus']], DF['medv'])
Quick_Model_Traier(DF[['lstat','ptratio']], DF['medv'])
Quick_Model_Traier(DF[['indus','ptratio']], DF['medv'])
Quick_Model_Traier(DF[['lstat', 'ptratio', 'indus']], DF['medv'])
Quick_Model_Traier(DF[['rm', 'ptratio', 'indus']], DF['medv'])
Quick_Model_Traier(DF[['rm','lstat', 'indus']], DF['medv'])
Quick_Model_Traier(DF[['rm','lstat', 'ptratio']], DF['medv'])
Quick_Model_Traier(DF[['rm','lstat', 'ptratio', 'indus']], DF['medv'])
```

```
Without Outliers Removed and Data Normalized

------------------
Input :  ['rm' 'lstat']
Mean Absolute Error : 3.8987597213823584
Model Accuracy : 0.5739577415025858
------------------
Input :  ['rm' 'indus']
Mean Absolute Error : 3.916202739510398
Model Accuracy : 0.4738047952188048
------------------
Input :  ['rm' 'ptratio']
Mean Absolute Error : 3.992441091741771
Model Accuracy : 0.48426471238253344
------------------
Input :  ['lstat' 'indus']
Mean Absolute Error : 4.119913776242081
Model Accuracy : 0.549867883942468
------------------
Input :  ['lstat' 'ptratio']
Mean Absolute Error : 3.552771713198138
Model Accuracy : 0.6237952757915204
------------------
Input :  ['indus' 'ptratio']
Mean Absolute Error : 4.789636456576802
Model Accuracy : 0.4018068071390196
------------------
Input :  ['lstat' 'ptratio' 'indus']
Mean Absolute Error : 3.569780530297716
Model Accuracy : 0.6232538193298415
------------------
Input :  ['rm' 'ptratio' 'indus']
Mean Absolute Error : 3.658719255783872
Model Accuracy : 0.5289984789622016
------------------
Input :  ['rm' 'lstat' 'indus']
Mean Absolute Error : 3.863755932060461
Model Accuracy : 0.5784397814546065
```

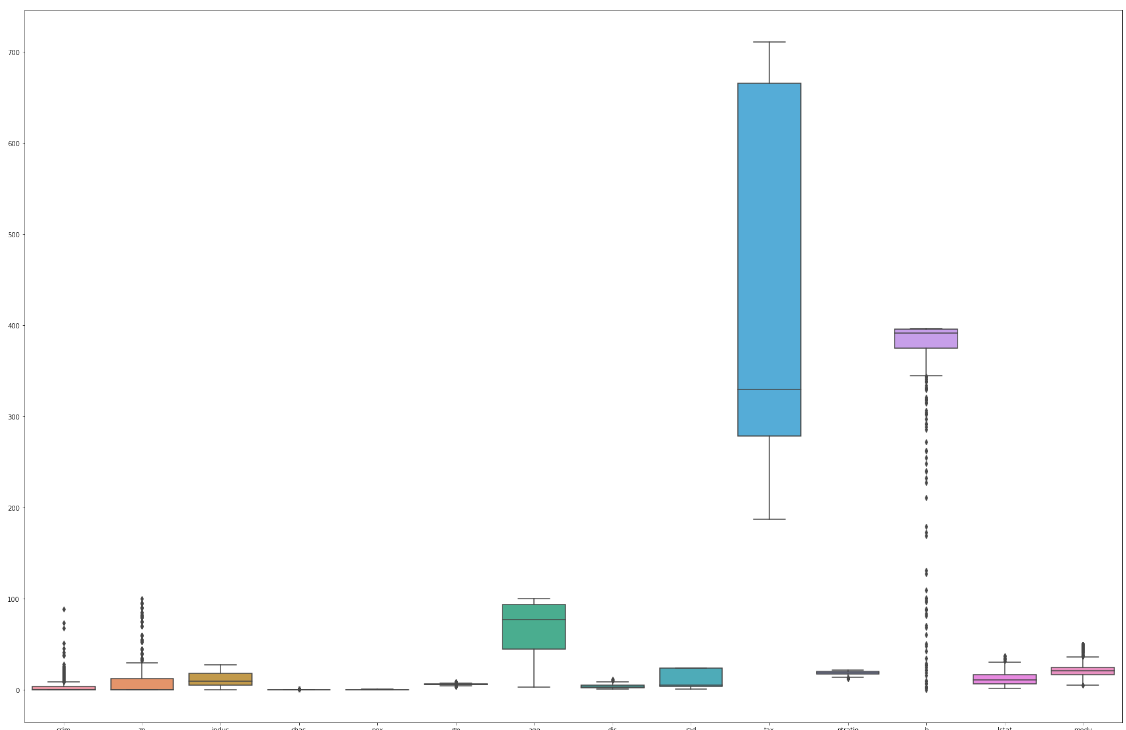
```
------------------
Input :  ['rm' 'lstat' 'ptratio']
Mean Absolute Error : 3.3325380783240957
Model Accuracy : 0.6302528487272827
------------------
Input :  ['rm' 'lstat' 'ptratio' 'indus']
Mean Absolute Error : 3.352928707925681
Model Accuracy : 0.628420675407839
```

---

3 Removing Outliers

[ ]:
```
fig, ax = MPLOT.subplots(figsize=(30,20))
SNS.boxplot(data=DF, ax=ax)
```

[ ]:
```
<AxesSubplot:>
```

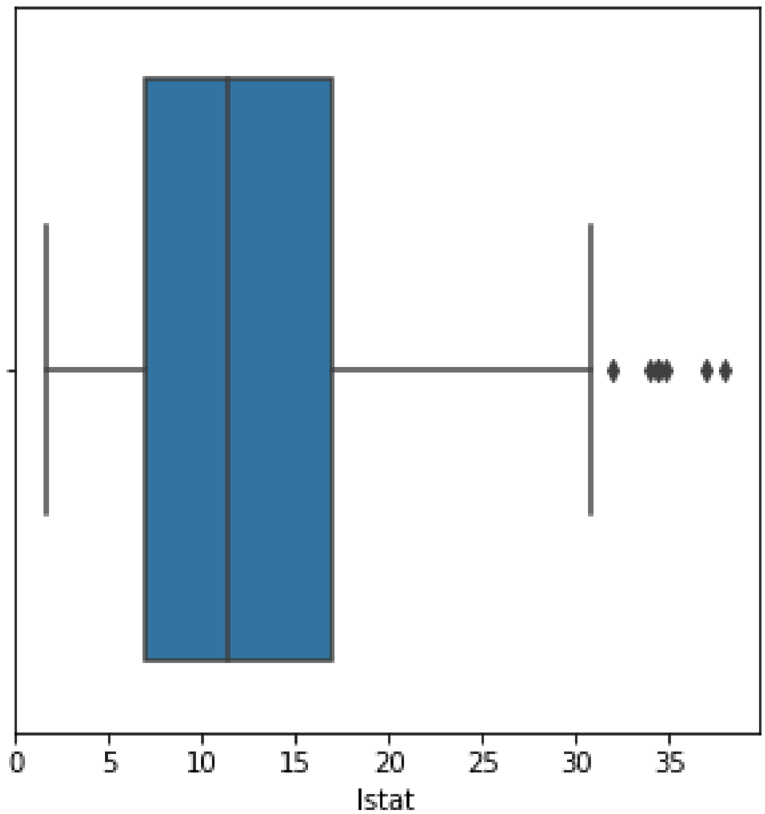


[ ]:
```
fig, ax = MPLOT.subplots(figsize=(5,5))
SNS.boxplot(DF['lstat'],ax=ax)
```

```
/home/yash-d3/Codium/6_Sem_Practicals/DSBDA/Assign_4/assign4_venv/lib/python3.8/
site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following
variable as a keyword arg: x. From version 0.12, the only valid positional
```

```
argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
  warnings.warn(
```

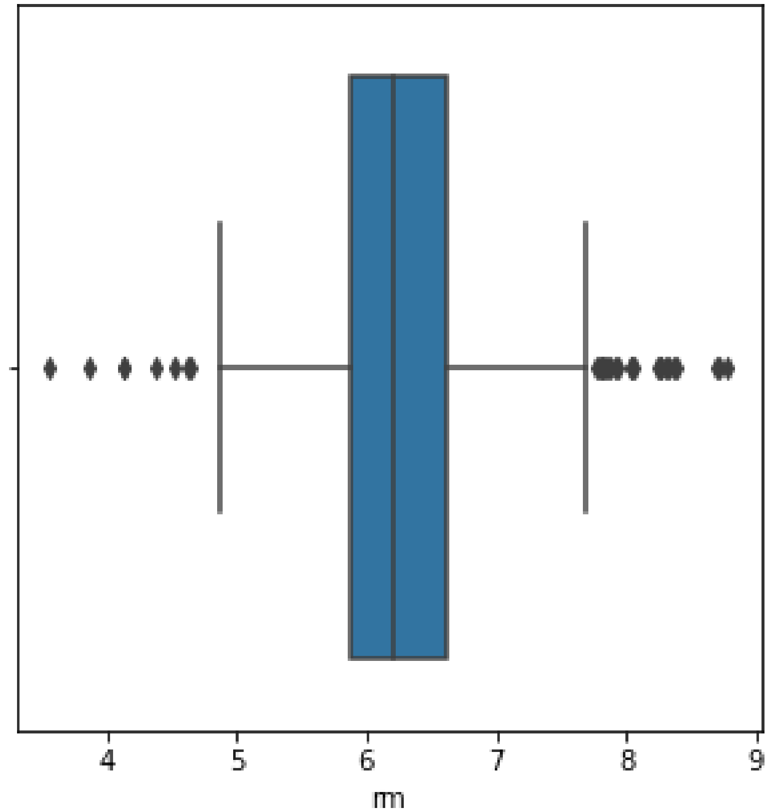
[ ]: <AxesSubplot:xlabel='lstat'>




```
[ ]: fig, ax = MPLOT.subplots(figsize=(5,5))
SNS.boxplot(DF['rm'],ax=ax)
```

```
/home/yash-d3/Codium/6_Sem_Practicals/DSBDA/Assign_4/assign4_venv/lib/python3.8/
site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following
variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
  warnings.warn(
```

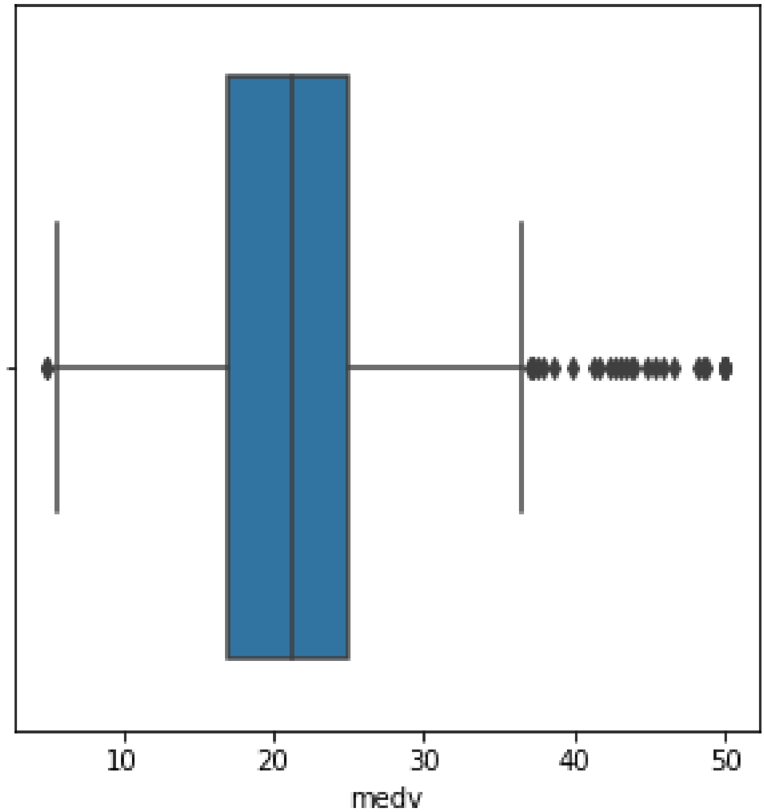
[ ]: <AxesSubplot:xlabel='rm'>



[ ]:
```
fig, ax = MPLOT.subplots(figsize=(5,5))
SNS.boxplot(DF['medv'],ax=ax)
```

```
/home/yash-d3/Codium/6_Sem_Practicals/DSBDA/Assign_4/assign4_venv/lib/python3.8/
site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following
variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
  warnings.warn(
```

[ ]: <AxesSubplot:xlabel='medv'>



[ ]:
```
# lstat_Q1 = NP.percentile(DF['lstat'], 25, interpolation = 'midpoint')
# lstat_Q3 = NP.percentile(DF['lstat'], 75, interpolation = 'midpoint')
# lstat_IQR = lstat_Q3 - lstat_Q1
# lstat_Upper_Threshold = lstat_Q3 + 1.5 * lstat_IQR
# lstat_Lower_Threshold = lstat_Q1 - 1.5 * lstat_IQR

# rm_Q1 = NP.percentile(DF['rm'], 25, interpolation = 'midpoint')
# rm_Q3 = NP.percentile(DF['rm'], 75, interpolation = 'midpoint')
# rm_IQR = rm_Q3 - rm_Q1
# rm_Upper_Threshold = rm_Q3 + 1.5 * rm_IQR
# rm_Lower_Threshold = rm_Q1 - 1.5 * rm_IQR

# medv_Q1 = NP.percentile(DF['medv'], 25, interpolation = 'midpoint')
# medv_Q3 = NP.percentile(DF['medv'], 75, interpolation = 'midpoint')
# medv_IQR = medv_Q3 - medv_Q1
# medv_Upper_Threshold = medv_Q3 + 1.5 * medv_IQR
# medv_Lower_Threshold = medv_Q1 - 1.5 * medv_IQR

# lstat_Upper_Threshold = DF['lstat'].mean() + 1.5*DF['lstat'].std()
# lstat_Lower_Threshold = DF['lstat'].mean() - 1.5*DF['lstat'].std()
```

```
# rm_Upper_Threshold    = DF['rm'].mean()    + 1.5*DF['rm'].std()
# rm_Lower_Threshold    = DF['rm'].mean()    - 1.5*DF['rm'].std()
# medv_Upper_Threshold  = DF['medv'].mean()  + 1.5*DF['medv'].std()
# medv_Lower_Threshold  = DF['medv'].mean()  - 1.5*DF['medv'].std()

# print("\nFor lstat:")
# print("Highest allowed",lstat_Upper_Threshold)
# print("Lowest allowed",lstat_Lower_Threshold)
# print("\nFor rm:")
# print("Highest allowed",rm_Upper_Threshold)
# print("Lowest allowed",rm_Lower_Threshold)
# print("\nFor medv:")
# print("Highest allowed",medv_Upper_Threshold)
# print("Lowest allowed",medv_Lower_Threshold)
```

3.0.1 Create an Outlier Function

[ ]:
```
def Remove_Outlier(DataFrame, Col_Name) :
    Upper_Threshold = DataFrame[Col_Name].mean() + 3*DataFrame[Col_Name].std()
    Lower_Threshold = DataFrame[Col_Name].mean() - 3*DataFrame[Col_Name].std()
    Count = DataFrame[(DataFrame[Col_Name] >= Upper_Threshold ) |␣
 ↪(DataFrame[Col_Name] <= Lower_Threshold)][Col_Name].count()
    print("[ * ] In", str(Col_Name), "->", "High :" ,Upper_Threshold, "| Low :
 ↪", Lower_Threshold, "| -> Outliers Detected :", Count)
    return DataFrame[((DataFrame[Col_Name] >= Lower_Threshold ) &␣
 ↪(DataFrame[Col_Name] <= Upper_Threshold))]
```

[ ]:
```
DF = Remove_Outlier(DF, 'crim')
DF = Remove_Outlier(DF, 'zn')
DF = Remove_Outlier(DF, 'indus')
DF = Remove_Outlier(DF, 'chas')
DF = Remove_Outlier(DF, 'nox')
DF = Remove_Outlier(DF, 'rm')
DF = Remove_Outlier(DF, 'age')
DF = Remove_Outlier(DF, 'dis')
DF = Remove_Outlier(DF, 'rad')
DF = Remove_Outlier(DF, 'tax')
DF = Remove_Outlier(DF, 'ptratio')
DF = Remove_Outlier(DF, 'b')
DF = Remove_Outlier(DF, 'lstat')
```

```
[ * ] In crim -> High : 29.41815887330972 | Low : -22.19111175868521 | ->
Outliers Detected : 8
[ * ] In zn -> High : 81.93953114918878 | Low : -58.84716167127714 | -> Outliers
Detected : 14
[ * ] In indus -> High : 31.62985151287539 | Low : -9.065099446759694 | ->
Outliers Detected : 0
```

```
[ * ] In chas -> High : 0.8377343966042569 | Low : -0.6972385288356618 | ->
Outliers Detected : 34
[ * ] In nox -> High : 0.8893287775824079 | Low : 0.21814944463981423 | ->
Outliers Detected : 0
[ * ] In rm -> High : 8.271119246518502 | Low : 4.234507420148164 | -> Outliers
Detected : 9
[ * ] In age -> High : 152.58441245278814 | Low : -16.18758705596275 | ->
Outliers Detected : 0
[ * ] In dis -> High : 9.992190634713909 | Low : -2.353658208410053 | ->
Outliers Detected : 4
[ * ] In rad -> High : 35.444454454977105 | Low : -16.552005942391297 | ->
Outliers Detected : 0
[ * ] In tax -> High : 911.5055976408532 | Low : -95.25388139371364 | ->
Outliers Detected : 0
[ * ] In ptratio -> High : 24.900941955181565 | Low : 12.262902438411112 | ->
Outliers Detected : 0
[ * ] In b -> High : 629.3535304300758 | Low : 84.5422132770181 | -> Outliers
Detected : 23
[ * ] In lstat -> High : 32.31886840804462 | Low : -7.506501258286169 | ->
Outliers Detected : 2
```

[ ]:
```
print("With Outliers Removed\n")

Quick_Model_Traier(DF[['rm','lstat']], DF['medv'])
Quick_Model_Traier(DF[['rm','indus']], DF['medv'])
Quick_Model_Traier(DF[['rm','ptratio']], DF['medv'])
Quick_Model_Traier(DF[['lstat','indus']], DF['medv'])
Quick_Model_Traier(DF[['lstat','ptratio']], DF['medv'])
Quick_Model_Traier(DF[['indus','ptratio']], DF['medv'])
Quick_Model_Traier(DF[['lstat', 'ptratio', 'indus']], DF['medv'])
Quick_Model_Traier(DF[['rm', 'ptratio', 'indus']], DF['medv'])
Quick_Model_Traier(DF[['rm','lstat', 'indus']], DF['medv'])
Quick_Model_Traier(DF[['rm','lstat', 'ptratio']], DF['medv'])
Quick_Model_Traier(DF[['rm','lstat', 'ptratio', 'indus']], DF['medv'])
```

```
With Outliers Removed

------------------
Input :  ['rm' 'lstat']
Mean Absolute Error : 3.5901658479914498
Model Accuracy : 0.5707756956537006
------------------
Input :  ['rm' 'indus']
Mean Absolute Error : 4.214145632694808
Model Accuracy : 0.3785351362452488
------------------
Input :  ['rm' 'ptratio']
Mean Absolute Error : 3.9565352986203743
```

```
Model Accuracy : 0.42135202821536055
-------------------
Input :  ['lstat' 'indus']
Mean Absolute Error : 4.08950469160613
Model Accuracy : 0.5151382042361962
-------------------
Input :  ['lstat' 'ptratio']
Mean Absolute Error : 3.6532858035580666
Model Accuracy : 0.5483688926590018
-------------------
Input :  ['indus' 'ptratio']
Mean Absolute Error : 5.59353842300006
Model Accuracy : 0.1482510677156117
-------------------
Input :  ['lstat' 'ptratio' 'indus']
Mean Absolute Error : 3.7055264826165977
Model Accuracy : 0.5359354082378105
-------------------
Input :  ['rm' 'ptratio' 'indus']
Mean Absolute Error : 3.878942886819066
Model Accuracy : 0.42063874169720794
-------------------
Input :  ['rm' 'lstat' 'indus']
Mean Absolute Error : 3.661655172701733
Model Accuracy : 0.5446632170641079
-------------------
Input :  ['rm' 'lstat' 'ptratio']
Mean Absolute Error : 3.123622617328399
Model Accuracy : 0.5836276654856962
-------------------
Input :  ['rm' 'lstat' 'ptratio' 'indus']
Mean Absolute Error : 3.161863119766873
Model Accuracy : 0.5707497199016172
```

---

4 Normalizing Data and Training Model

[ ]:
```
from sklearn.preprocessing import MinMaxScaler

def Quick_Model_Traier_with_Norm(Input_Data, Output_Data):
    print("-------------------")
    print("Input : ", Input_Data.columns.values)
    X_TRAIN, X_TEST, Y_TRAIN, Y_TEST = train_test_split( Input_Data,␣
 ↪Output_Data, test_size=0.2, random_state=42)

    norm = MinMaxScaler().fit(X_TRAIN) # fit scaler on training data
```

```python
    X_TRAIN = norm.transform(X_TRAIN) # transform training data
    norm = MinMaxScaler().fit(X_TEST) # fit scaler on training data
    X_TEST = norm.transform(X_TEST) # transform training data

    LinearRegressionModel(X_TRAIN, X_TEST, Y_TRAIN, Y_TEST)
```

[ ]:
```python
print("With Outliers Removed and Data Normalized\n")

Quick_Model_Traier_with_Norm(DF[['rm','lstat']], DF['medv'])
Quick_Model_Traier_with_Norm(DF[['rm','indus']], DF['medv'])
Quick_Model_Traier_with_Norm(DF[['rm','ptratio']], DF['medv'])
Quick_Model_Traier_with_Norm(DF[['lstat','indus']], DF['medv'])
Quick_Model_Traier_with_Norm(DF[['lstat','ptratio']], DF['medv'])
Quick_Model_Traier_with_Norm(DF[['indus','ptratio']], DF['medv'])
Quick_Model_Traier_with_Norm(DF[['lstat', 'ptratio', 'indus']], DF['medv'])
Quick_Model_Traier_with_Norm(DF[['rm', 'ptratio', 'indus']], DF['medv'])
Quick_Model_Traier_with_Norm(DF[['rm','lstat', 'indus']], DF['medv'])
Quick_Model_Traier_with_Norm(DF[['rm','lstat', 'ptratio']], DF['medv'])
Quick_Model_Traier_with_Norm(DF[['rm','lstat', 'ptratio', 'indus']], DF['medv'])
```

```
With Outliers Removed and Data Normalized

------------------
Input :  ['rm' 'lstat']
Mean Absolute Error : 3.7576925987727905
Model Accuracy : 0.49485441393070373
------------------
Input :  ['rm' 'indus']
Mean Absolute Error : 5.032263280759843
Model Accuracy : 0.13971703014567904
------------------
Input :  ['rm' 'ptratio']
Mean Absolute Error : 5.002156795258233
Model Accuracy : 0.2182171714061114
------------------
Input :  ['lstat' 'indus']
Mean Absolute Error : 4.09309262599304
Model Accuracy : 0.523720523365256
------------------
Input :  ['lstat' 'ptratio']
Mean Absolute Error : 3.7054580906051995
Model Accuracy : 0.5642802129007987
------------------
Input :  ['indus' 'ptratio']
Mean Absolute Error : 5.714244824431733
Model Accuracy : 0.09382667556021662
------------------
```

```
Input :  ['lstat' 'ptratio' 'indus']
Mean Absolute Error : 3.7482415598981778
Model Accuracy : 0.5481783399665785
------------------
Input :  ['rm' 'ptratio' 'indus']
Mean Absolute Error : 4.73762838297241
Model Accuracy : 0.21558389290626723
------------------
Input :  ['rm' 'lstat' 'indus']
Mean Absolute Error : 3.7751619350638
Model Accuracy : 0.44883096982695536
------------------
Input :  ['rm' 'lstat' 'ptratio']
Mean Absolute Error : 3.523686516494807
Model Accuracy : 0.5154702317232409
------------------
Input :  ['rm' 'lstat' 'ptratio' 'indus']
Mean Absolute Error : 3.5605200420659657
Model Accuracy : 0.4927639000853453
```

4.0.1 Summary

- Without Outliers Removed and Data Normalized

[0.5739577415025858, 0.4738047952188048, 0.48426471238253344, 0.549867883942468, 0.6237952757

- With Outliers Removed

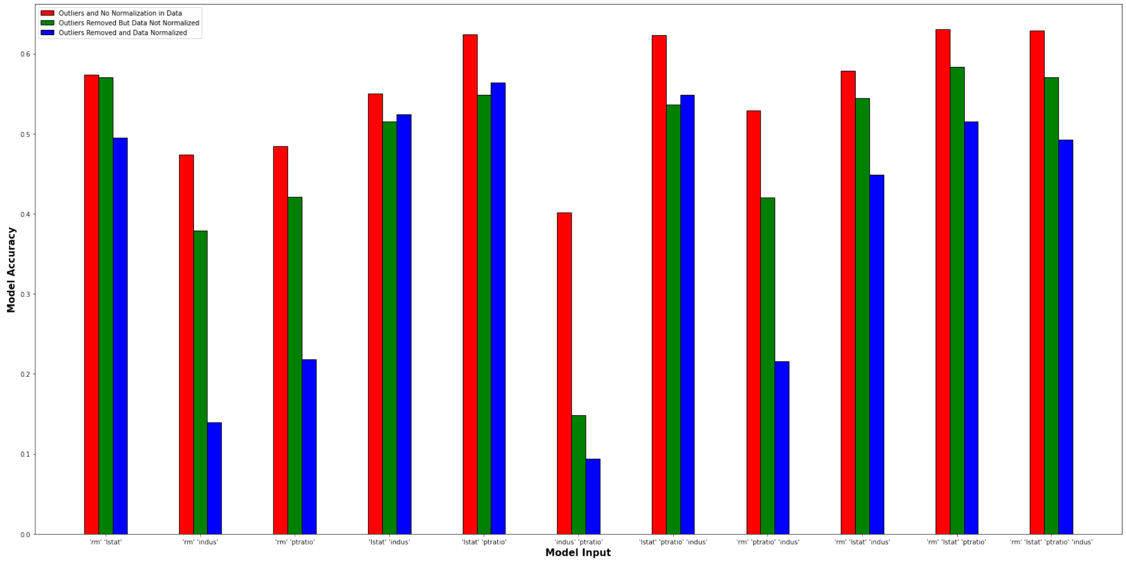
[0.5707756956537006, 0.3785351362452488, 0.42135202821536055, 0.5151382042361962, 0.548368892

- With Outliers Removed and Data Normalized

[0.49485441393070373, 0.13971703014567904, 0.2182171714061114, 0.523720523365256, 0.564280212

[ ]:
```
barWidth = 0.15
fig = MPLOT.subplots(figsize =(30, 15))

# set height of bar
outliers_and_no_norm = [ 0.5739577415025858, 0.4738047952188048, 0.
 ↪48426471238253344, 0.549867883942468, 0.6237952757915204, 0.
 ↪4018068071390196, 0.6232538193298415, 0.5289984789622016, 0.
 ↪5784397814546065, 0.6302528487272827, 0.628420675407839 ]
no_outliers_and_no_norm = [ 0.5707756956537006, 0.3785351362452488, 0.
 ↪42135202821536055, 0.5151382042361962, 0.5483688926590018, 0.
 ↪1482510677156117, 0.5359354082378105, 0.42063874169720794, 0.
 ↪5446632170641079, 0.5836276654856962, 0.5707497199016172 ]
```

```
no_outliers_and_norm = [ 0.49485441393070373, 0.13971703014567904, 0.
↪2182171714061114, 0.523720523365256, 0.5642802129007987, 0.
↪09382667556021662, 0.5481783399665785, 0.21558389290626723, 0.
↪44883096982695536, 0.5154702317232409, 0.4927639000853453 ]



# Set position of bar on X axis
br1 = NP.arange(len(outliers_and_no_norm))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]

# Make the plot
MPLOT.bar(br1, outliers_and_no_norm, color ='r', width = barWidth,
        edgecolor ='black', label ='Outliers and No Normalization in Data')
MPLOT.bar(br2, no_outliers_and_no_norm, color ='g', width = barWidth,
        edgecolor ='black', label ='Outliers Removed But Data Not Normalized')
MPLOT.bar(br3, no_outliers_and_norm, color ='b', width = barWidth,
        edgecolor ='black', label ='Outliers Removed and Data Normalized')



# Adding Xticks
MPLOT.xlabel('Model Input', fontweight ='bold', fontsize = 15)
MPLOT.ylabel('Model Accuracy', fontweight ='bold', fontsize = 15)
MPLOT.xticks([r + barWidth for r in range(len(outliers_and_no_norm))],
        [
                "'rm' 'lstat'",
                "'rm' 'indus'",
                "'rm' 'ptratio'",
                "'lstat' 'indus'",
                "'lstat' 'ptratio'",
                "'indus' 'ptratio'",
                "'lstat' 'ptratio' 'indus'",
                "'rm' 'ptratio' 'indus'",
                "'rm' 'lstat' 'indus'",
                "'rm' 'lstat' 'ptratio'",
                "'rm' 'lstat' 'ptratio' 'indus'"
        ])



MPLOT.legend()
MPLOT.show()
```



Outliers and No Normalization in Data
Outliers Removed But Data Not Normalized
Outliers Removed and Data Normalized
Model Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
'rm' 'lstat'
'rm' 'indus'
'rm' 'ptratio'
'lstat' 'indus'
'lstat' 'ptratio'
'indus' 'ptratio'
'lstat' 'ptratio' 'indus'
'rm' 'ptratio' 'indus'
'rm' 'lstat' 'indus'
'rm' 'lstat' 'ptratio'
'rm' 'lstat' 'ptratio' 'indus'
Model Input