A4_31378

February 5, 2022

Name: Sourav Kotkar

Roll No: 31378

0.1 Assignment-4 : Data Analytics II

Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset (https://www.kaggle.com/c/boston-housing). The Boston Housing dataset contains information about various houses in Boston through different parameters. There are 506 samples and 14 feature variables in this dataset.

The objective is to predict the value of prices of the house using the given features.

[1]:
```
# 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 centres
# 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 blacks by town
# LSTAT - % lower status of the population
# MEDV - Median value of owner-occupied homes in $1000's
```

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

[3]:
```
df = pd.read_csv('HousingData.csv')
```

[4]:
```
df
```

[4]:
```
        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
```

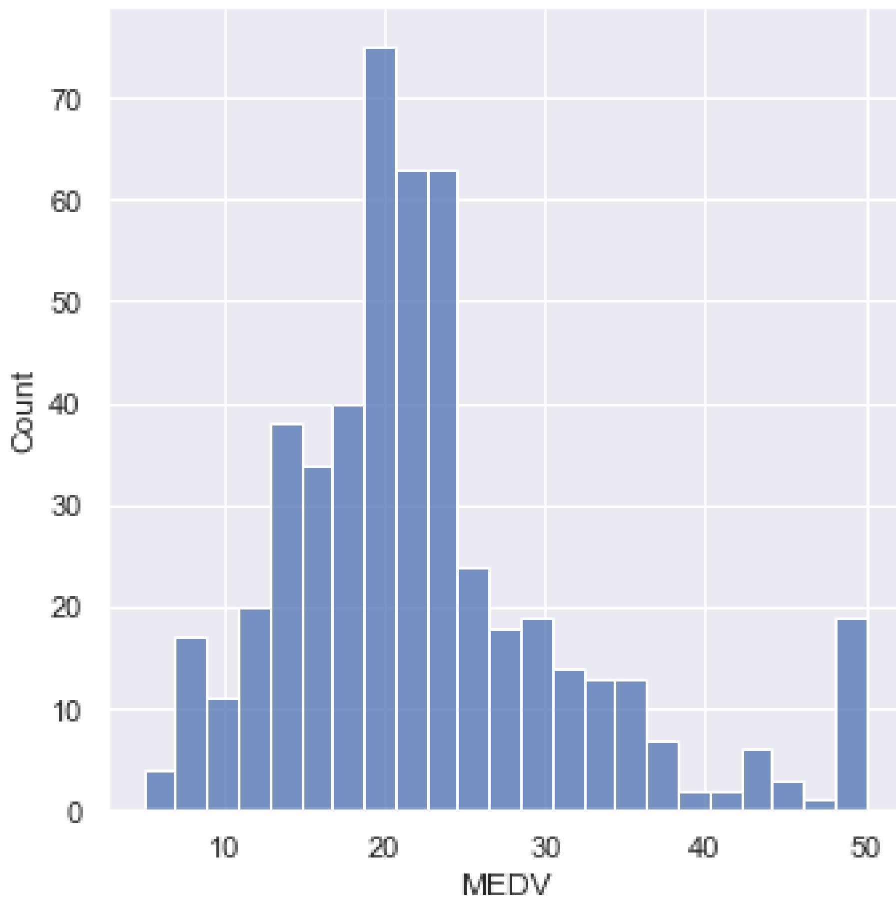
```
1    0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622    3  222
..       …   …     …    …     …     …    …       …   …   …
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]
```

0.1.1 Visual Analysis

[5]:
```python
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.displot(df['MEDV'])
plt.show()
```

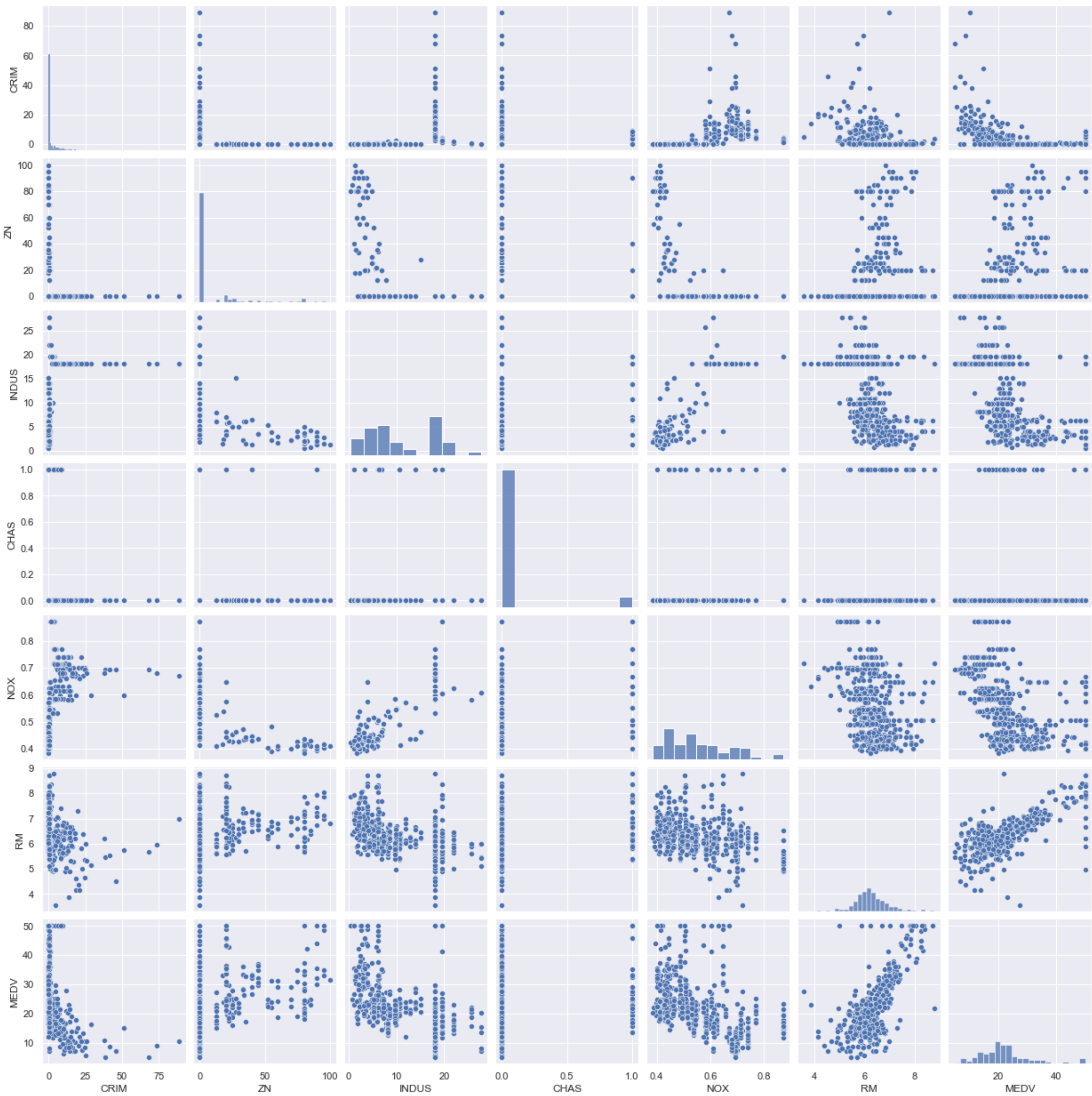


[6]:
```
df['MEDV'].skew()
```

[6]: 1.1080984082549072

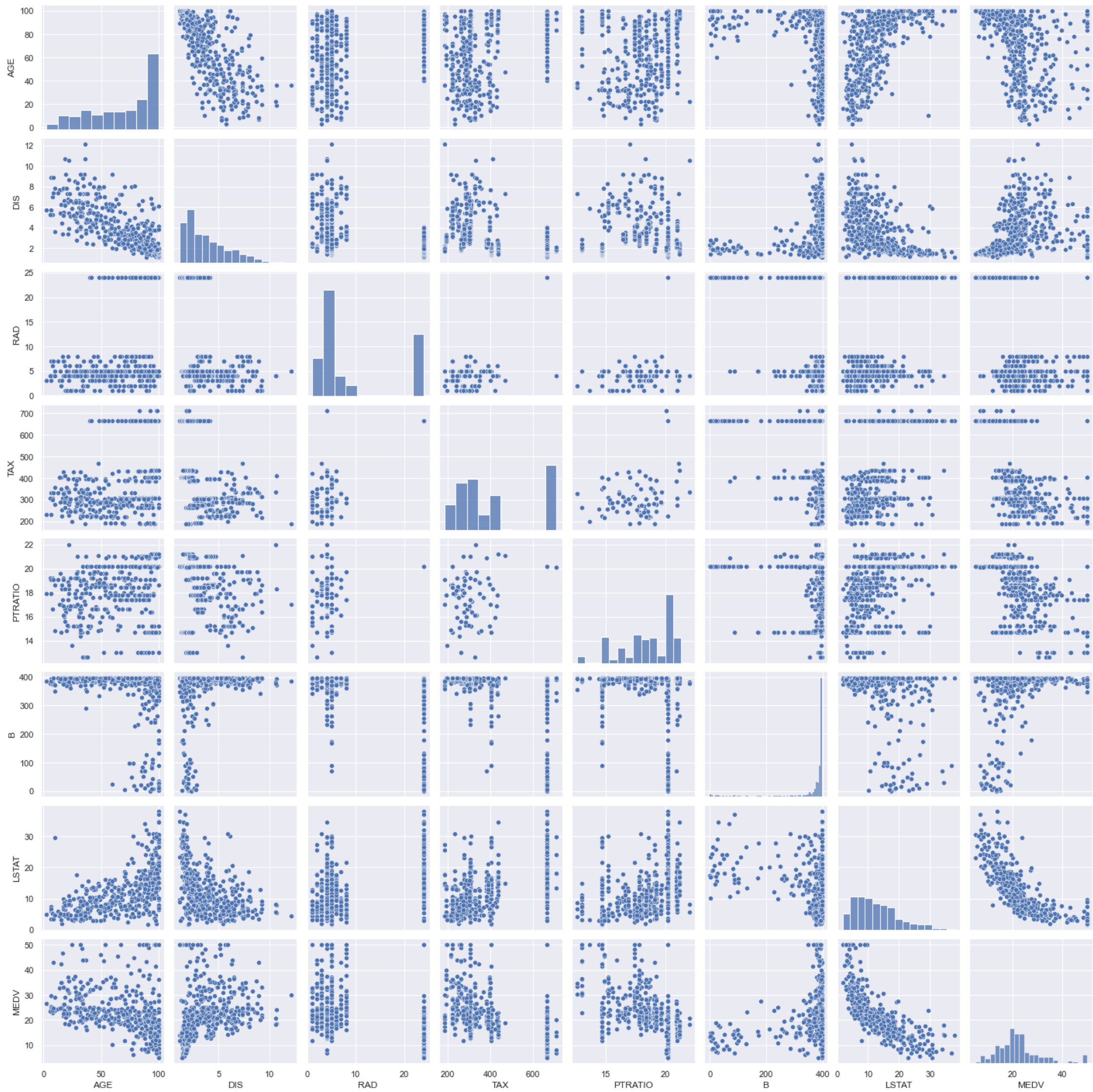
[7]:
```
#Pair Plot
sns.pairplot(df, vars = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'MEDV'])
```

[7]: <seaborn.axisgrid.PairGrid at 0x260bf355730>




```
[8]: sns.pairplot(df, vars = ['AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT',␣
     ↪'MEDV'])
```

```
[8]: <seaborn.axisgrid.PairGrid at 0x260c09a0880>
```



[9]:
```
#Heat Map
correlation_matrix = df.corr().round(2)
sns.heatmap(data=correlation_matrix, annot=True)
```

[9]: <AxesSubplot:>

0.1.2 Removing Outliers

[10]:
```
MEDV_Q1 = df['MEDV'].quantile(0.25) #First quartile
MEDV_Q3 = df['MEDV'].quantile(0.75) #Third quartile
MEDV_IQR = MEDV_Q3 - MEDV_Q1 #Inter quartile range
MEDV_lower_limit = MEDV_Q1 - 1.5 * MEDV_IQR
MEDV_upper_limit = MEDV_Q3 + 1.5 * MEDV_IQR
MEDV_lower_limit, MEDV_upper_limit
```

[10]: (5.0624999999999964, 36.962500000000006)

[11]:
```
df[(df['MEDV']<MEDV_lower_limit) | (df['MEDV']>MEDV_upper_limit)]
```

[11]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	\
97	0.12083	0.0	2.89	0.0	0.4450	8.069	76.0	3.4952	2	276	
98	0.08187	0.0	2.89	0.0	0.4450	7.820	36.9	3.4952	2	276	
157	1.22358	NaN	19.58	0.0	0.6050	6.943	97.4	1.8773	5	403	
161	1.46336	0.0	19.58	0.0	0.6050	7.489	90.8	1.9709	5	403	
162	1.83377	0.0	19.58	1.0	0.6050	7.802	98.2	2.0407	5	403	
163	1.51902	0.0	19.58	1.0	0.6050	8.375	NaN	2.1620	5	403	
166	2.01019	0.0	19.58	0.0	0.6050	7.929	96.2	2.0459	5	403	
179	0.05780	0.0	2.46	0.0	0.4880	6.980	58.4	2.8290	3	193	

180	0.06588	0.0	2.46	0.0	0.4880	7.765	83.3	2.7410	3	193
182	0.09103	0.0	2.46	0.0	0.4880	7.155	92.2	2.7006	3	193
186	0.05602	NaN	2.46	0.0	0.4880	7.831	53.6	3.1992	3	193
190	0.09068	45.0	3.44	0.0	0.4370	6.951	21.5	6.4798	5	398
195	0.01381	80.0	0.46	0.0	0.4220	7.875	32.0	5.6484	4	255
202	0.02177	82.5	2.03	0.0	0.4150	7.610	15.7	6.2700	2	348
203	0.03510	95.0	2.68	0.0	0.4161	7.853	33.2	5.1180	4	224
204	0.02009	95.0	2.68	0.0	0.4161	8.034	31.9	5.1180	4	224
224	0.31533	0.0	6.20	0.0	0.5040	8.266	78.3	2.8944	8	307
225	0.52693	0.0	6.20	0.0	0.5040	8.725	83.0	2.8944	8	307
226	0.38214	0.0	6.20	0.0	0.5040	8.040	86.5	3.2157	8	307
228	0.29819	0.0	6.20	0.0	0.5040	7.686	17.0	3.3751	8	307
232	0.57529	0.0	6.20	0.0	0.5070	8.337	73.3	3.8384	8	307
233	0.33147	0.0	6.20	0.0	0.5070	8.247	NaN	3.6519	8	307
253	0.36894	22.0	5.86	0.0	0.4310	8.259	8.4	8.9067	7	330
256	0.01538	90.0	3.75	0.0	0.3940	7.454	34.2	6.3361	3	244
257	0.61154	20.0	3.97	0.0	0.6470	8.704	86.9	1.8010	5	264
261	0.53412	20.0	3.97	0.0	0.6470	7.520	89.4	2.1398	5	264
262	NaN	20.0	3.97	0.0	0.6470	8.398	91.5	2.2885	5	264
267	0.57834	20.0	3.97	0.0	0.5750	8.297	67.0	2.4216	5	264
268	0.54050	20.0	3.97	0.0	0.5750	7.470	52.6	2.8720	5	264
280	0.03578	20.0	3.33	0.0	0.4429	7.820	64.5	4.6947	5	216
282	0.06129	20.0	3.33	1.0	0.4429	7.645	49.7	5.2119	5	216
283	0.01501	90.0	1.21	1.0	0.4010	7.923	24.8	5.8850	1	198
291	0.07886	80.0	4.95	0.0	0.4110	7.148	27.7	5.1167	4	245
368	4.89822	0.0	18.10	0.0	0.6310	4.970	NaN	1.3325	24	666
369	NaN	0.0	18.10	1.0	0.6310	6.683	96.8	1.3567	24	666
370	6.53876	0.0	18.10	1.0	0.6310	7.016	97.5	1.2024	24	666
371	9.23230	0.0	18.10	0.0	0.6310	6.216	100.0	1.1691	24	666
372	8.26725	0.0	18.10	1.0	0.6680	5.875	89.6	1.1296	24	666
398	38.35180	0.0	18.10	0.0	0.6930	5.453	100.0	1.4896	24	666
405	67.92080	0.0	18.10	0.0	0.6930	5.683	100.0	1.4254	24	666

	PTRATIO	B	LSTAT	MEDV
97	18.0	396.90	4.21	38.7
98	18.0	393.53	3.57	43.8
157	14.7	363.43	4.59	41.3
161	14.7	374.43	1.73	50.0
162	14.7	389.61	1.92	50.0
163	14.7	388.45	3.32	50.0
166	14.7	369.30	3.70	50.0
179	17.8	396.90	5.04	37.2
180	17.8	395.56	7.56	39.8
182	17.8	394.12	4.82	37.9
186	17.8	392.63	4.45	50.0
190	15.2	377.68	5.10	37.0
195	14.4	394.23	2.97	50.0

```
202      14.7  395.38   3.11  42.3
203      14.7  392.78   3.81  48.5
204      14.7  390.55   2.88  50.0
224      17.4  385.05   4.14  44.8
225      17.4  382.00   4.63  50.0
226      17.4  387.38    NaN  37.6
228      17.4  377.51    NaN  46.7
232      17.4  385.91   2.47  41.7
233      17.4  378.95   3.95  48.3
253      19.1  396.90   3.54  42.8
256      15.9  386.34   3.11  44.0
257      13.0  389.70   5.12  50.0
261      13.0  388.37   7.26  43.1
262      13.0  386.86   5.91  48.8
267      13.0  384.54   7.44  50.0
268      13.0  390.30   3.16  43.5
280      14.9  387.31   3.76  45.4
282      14.9  377.07   3.01  46.0
283      13.6  395.52   3.16  50.0
291      19.2  396.90   3.56  37.3
368      20.2  375.52   3.26  50.0
369      20.2  375.33   3.73  50.0
370      20.2  392.05   2.96  50.0
371      20.2  366.15   9.53  50.0
372      20.2  347.88   8.88  50.0
398      20.2  396.90  30.59   5.0
405      20.2  384.97  22.98   5.0
```

[12]:
```
df_without_outliers = df[(df['MEDV']>MEDV_lower_limit) &
 ↪(df['MEDV']<MEDV_upper_limit)]
df_without_outliers
```

[12]:
```
        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

[466 rows x 14 columns]
```

0.1.3 Removing Null Values

[13]:
```
df_without_outliers.isnull().sum()
```

[13]:
```
CRIM       18
ZN         18
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        17
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      18
MEDV        0
dtype: int64
```

[14]:
```
mean_value=df_without_outliers['CRIM'].mean()
df_without_outliers['CRIM'].fillna(value=mean_value, inplace=True)

median_value=df_without_outliers['ZN'].median()
df_without_outliers['ZN'].fillna(value=median_value, inplace=True)

mean_value=df_without_outliers['INDUS'].mean()
df_without_outliers['INDUS'].fillna(value=mean_value, inplace=True)

median_value=df_without_outliers['CHAS'].median()
df_without_outliers['CHAS'].fillna(value=median_value, inplace=True)

mean_value=df_without_outliers['AGE'].mean()
df_without_outliers['AGE'].fillna(value=mean_value, inplace=True)
```

```
mean_value=df_without_outliers['LSTAT'].mean()
df_without_outliers['LSTAT'].fillna(value=mean_value, inplace=True)
```

```
c:\users\kotka\appdata\local\programs\python\python39\lib\site-
packages\pandas\core\generic.py:6383: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return self._update_inplace(result)
c:\users\kotka\appdata\local\programs\python\python39\lib\site-
packages\pandas\core\generic.py:6383: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return self._update_inplace(result)
c:\users\kotka\appdata\local\programs\python\python39\lib\site-
packages\pandas\core\generic.py:6383: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return self._update_inplace(result)
```

[15]:
```
df_without_outliers.isnull().sum()
```

[15]:
```
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
```

[16]:
```
df.dtypes #Print datatypes of variables
```

[16]:
```
CRIM       float64
ZN         float64
```

```
INDUS       float64
CHAS        float64
NOX         float64
RM          float64
AGE         float64
DIS         float64
RAD           int64
TAX           int64
PTRATIO     float64
B           float64
LSTAT       float64
MEDV        float64
dtype: object
```

[17]:
```python
df_without_outliers.describe() #Print statistical information
```

[17]:
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  466.000000  466.000000  466.000000  466.000000  466.000000  466.000000
mean     3.585240    9.934549   11.335942    0.057940    0.556826    6.179633
std      8.183585   21.813025    6.618821    0.233881    0.117400    0.576325
min      0.006320    0.000000    0.740000    0.000000    0.385000    3.561000
25%      0.085120    0.000000    5.860000    0.000000    0.453000    5.876250
50%      0.274475    0.000000    9.955000    0.000000    0.538000    6.163500
75%      3.585240    0.000000   18.100000    0.000000    0.624000    6.506250
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  466.000000  466.000000  466.000000  466.000000  466.000000  466.000000
mean    68.739421    3.833586    9.669528  413.105150   18.617382  354.197790
std     27.336371    2.124901    8.792361  168.544572    2.063273   94.679607
min      2.900000    1.137000    1.000000  187.000000   12.600000    0.320000
25%     46.400000    2.104425    4.000000  284.000000   17.400000  374.590000
50%     74.650000    3.272100    5.000000  335.000000   19.100000  391.955000
75%     93.750000    5.241300   24.000000  666.000000   20.200000  396.397500
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT        MEDV
count  466.000000  466.000000
mean    13.327612   20.719099
std      6.813401    6.451416
min      1.980000    5.600000
25%      7.927500   16.500000
50%     12.670000   20.600000
75%     17.107500   24.075000
max     37.970000   36.500000
```

[18]:
```python
df_without_outliers.corr() #Compute pairwise correlation of columns
```

[18]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
CRIM	1.000000	-0.182756	0.393912	-0.060629	0.410918	-0.196496	0.347436
ZN	-0.182756	1.000000	-0.505774	-0.045624	-0.493196	0.317041	-0.520164
INDUS	0.393912	-0.505774	1.000000	0.026347	0.742247	-0.354947	0.616798
CHAS	-0.060629	-0.045624	0.026347	1.000000	0.068979	0.079851	0.068677
NOX	0.410918	-0.493196	0.742247	0.068979	1.000000	-0.308291	0.707426
RM	-0.196496	0.317041	-0.354947	0.079851	-0.308291	1.000000	-0.269437
AGE	0.347436	-0.520164	0.616798	0.068677	0.707426	-0.269437	1.000000
DIS	-0.375943	0.651853	-0.721482	-0.078174	-0.771634	0.270629	-0.718536
RAD	0.620041	-0.292820	0.586829	-0.029363	0.607829	-0.161677	0.448049
TAX	0.575985	-0.289986	0.702630	-0.068513	0.665039	-0.230787	0.502746
PTRATIO	0.268471	-0.377863	0.368180	-0.105483	0.181092	-0.223777	0.271747
B	-0.405284	0.170617	-0.357874	0.046847	-0.383398	0.098115	-0.272792
LSTAT	0.429066	-0.405783	0.582806	0.004969	0.587031	-0.569962	0.613818
MEDV	-0.468821	0.430525	-0.604744	0.106445	-0.569014	0.568940	-0.541252

	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
CRIM	-0.375943	0.620041	0.575985	0.268471	-0.405284	0.429066	-0.468821
ZN	0.651853	-0.292820	-0.289986	-0.377863	0.170617	-0.405783	0.430525
INDUS	-0.721482	0.586829	0.702630	0.368180	-0.357874	0.582806	-0.604744
CHAS	-0.078174	-0.029363	-0.068513	-0.105483	0.046847	0.004969	0.106445
NOX	-0.771634	0.607829	0.665039	0.181092	-0.383398	0.587031	-0.569014
RM	0.270629	-0.161677	-0.230787	-0.223777	0.098115	-0.569962	0.568940
AGE	-0.718536	0.448049	0.502746	0.271747	-0.272792	0.613818	-0.541252
DIS	1.000000	-0.496763	-0.545198	-0.260617	0.305245	-0.536224	0.439844
RAD	-0.496763	1.000000	0.910061	0.450837	-0.452738	0.473695	-0.504338
TAX	-0.545198	0.910061	1.000000	0.439957	-0.448525	0.526652	-0.590236
PTRATIO	-0.260617	0.450837	0.439957	1.000000	-0.167452	0.322453	-0.488734
B	0.305245	-0.452738	-0.448525	-0.167452	1.000000	-0.367690	0.408316
LSTAT	-0.536224	0.473695	0.526652	0.322453	-0.367690	1.000000	-0.754751
MEDV	0.439844	-0.504338	-0.590236	-0.488734	0.408316	-0.754751	1.000000

0.1.4 Preparing Data for Training the Model

[19]:

```
#X = df_without_outliers[['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',␣
↪'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']]
X = df_without_outliers[['CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE', 'DIS',␣
↪'RAD', 'TAX', 'PTRATIO', 'LSTAT']]
y = df_without_outliers['MEDV']
```

[20]:

```
X
```

[20]:

	CRIM	ZN	INDUS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO
0	0.00632	18.0	2.31	0.538	6.575	65.200000	4.0900	1	296	15.3
1	0.02731	0.0	7.07	0.469	6.421	78.900000	4.9671	2	242	17.8
2	0.02729	0.0	7.07	0.469	7.185	61.100000	4.9671	2	242	17.8
3	0.03237	0.0	2.18	0.458	6.998	45.800000	6.0622	3	222	18.7

```
4    0.06905   0.0   2.18  0.458  7.147  54.200000  6.0622    3  222     18.7
..        …    …     …     …     …        …      …   …  …        …
501  0.06263   0.0  11.93  0.573  6.593  69.100000  2.4786    1  273     21.0
502  0.04527   0.0  11.93  0.573  6.120  76.700000  2.2875    1  273     21.0
503  0.06076   0.0  11.93  0.573  6.976  91.000000  2.1675    1  273     21.0
504  0.10959   0.0  11.93  0.573  6.794  89.300000  2.3889    1  273     21.0
505  0.04741   0.0  11.93  0.573  6.030  68.739421  2.5050    1  273     21.0

         LSTAT
0     4.980000
1     9.140000
2     4.030000
3     2.940000
4    13.327612
..         …
501  13.327612
502   9.080000
503   5.640000
504   6.480000
505   7.880000

[466 rows x 11 columns]
```

[21]:
```python
y
```

[21]:
```
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: 466, dtype: float64
```

0.1.5 Training the Regression Model

[22]:
```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

[23]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=17)
```

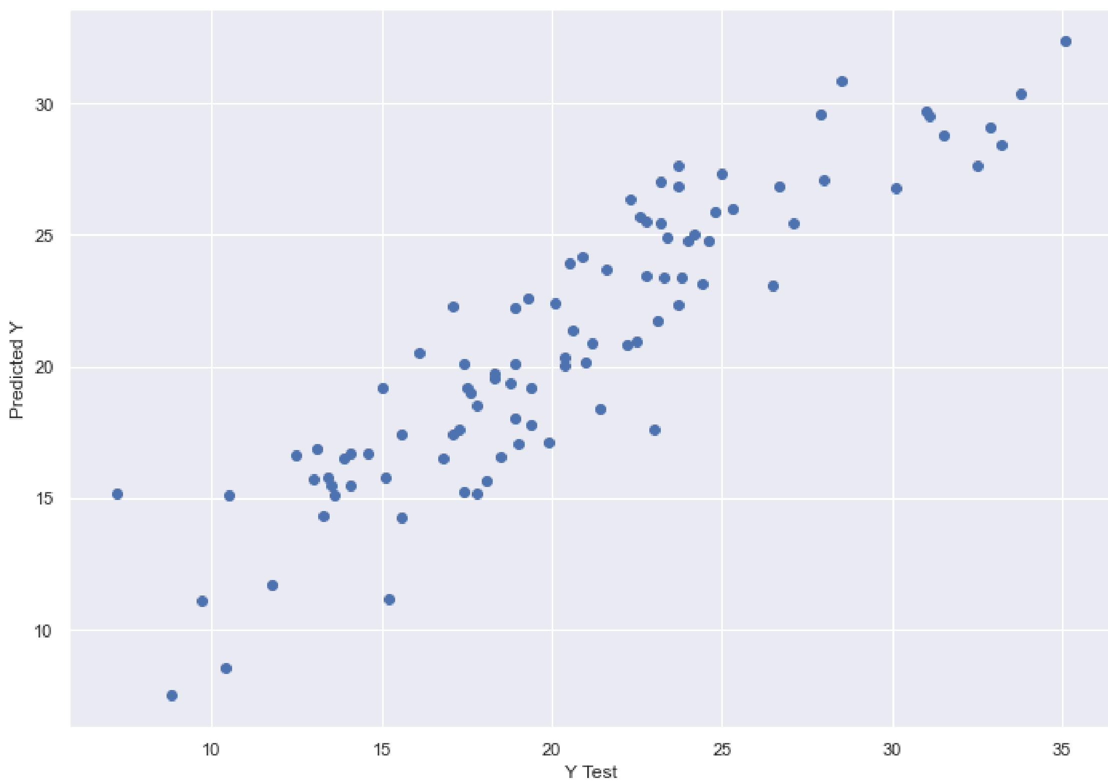
[24]:
```
lm = LinearRegression()
lm.fit(X_train, y_train)
```

[24]:
```
LinearRegression()
```

[25]:
```
predictions = lm.predict(X_test)
```

[26]:
```
plt.scatter(y_test,predictions)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

[26]:
```
Text(0, 0.5, 'Predicted Y')
```



[27]:
```
from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, predictions))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, predictions))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,␣
↪predictions)))
```

```
Mean Absolute Error: 2.118706319062847
Mean Squared Error: 6.631813466489961
Root Mean Squared Error: 2.5752307598523982
```

[28]:

```
lm.score(X_test, y_test)
```

[28]: 0.8135328747861935

[ ]:

```

```