Mehul Oswal

Class: TE-4 (M-4)

Roll No: 31444

DSBDAL Assignment - 4

Data Analytics I

Importing Python Modules

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

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

#%matplotlib inline:- to enable the inline plotting, where the plots/graphs
# r2_score :- (coefficient of determination) regression score function.
#mean_squared_error :- Mean squared error regression loss.
```

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

Loading Dataset

Dataset: https://github.com/selva86/datasets/blob/master/BostonHousing.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 oxide concentration (parts per 10 million)
- RM: Average number of rooms per house
- 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 dollars
- PTRATIO: Pupil-teacher ratio by town
- B: $1000(Bk — 0.63)^2$, where Bk is the proportion of [people of African American descent] by town
- LSTAT: Percentage of lower status of the population
- MEDV: Median value of owner-occupied homes in 1000s dollars

In [ ]:
```
# loading the dataset
```

```
df = pd.read_csv("https://raw.githubusercontent.com/selva86/datasets/master
df
```

Out[ ]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lst
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.9
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.1
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.0
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.9
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.3
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.6
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.0
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.6
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.4
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.8

506 rows × 14 columns

Data Preprocessing

In [ ]:

```
print("There are " + str(df.shape[0]) + " records with " + str(df.shape[1])
```

```
There are 506 records with 14 features each.
```

In [ ]:

```
# displaying technical information of dataset

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
```

In [ ]:

```
# displaying statistical information of numerical fields in dataset
```

```
df.describe()
```

Out[ ]:

	crim	zn	indus	chas	nox	rm	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000

In [ ]:

```
# number of null fields

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

Out[ ]:

```
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
```

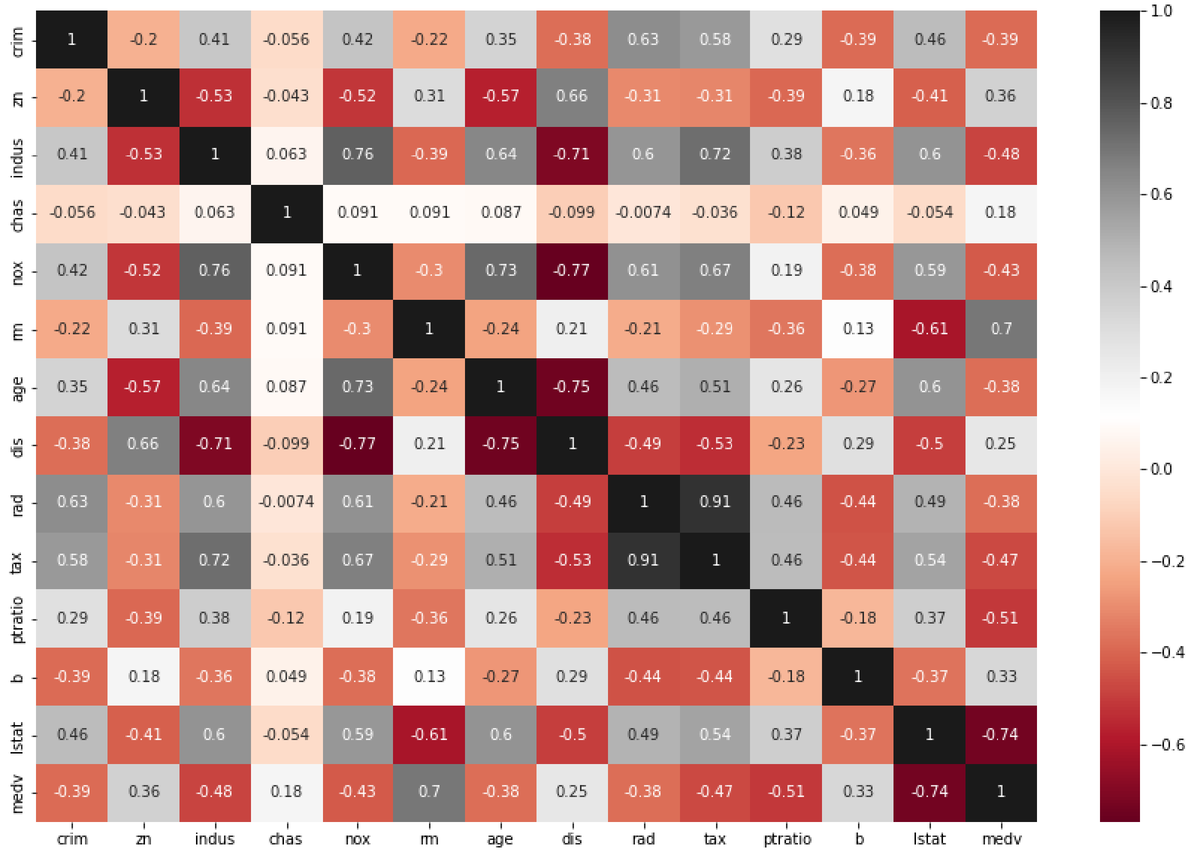
Graphical Representation

In [ ]:

```
# Heatmap to find CORRELATION
#0 -> no correlation
#+ve -> directly proportional
#-ve -> inversely proportional
plt.figure(figsize = (15, 10))
sns.heatmap(data = df.corr(), annot = True, cmap = "RdGy")
```

Out[ ]:

```
<AxesSubplot:>
```

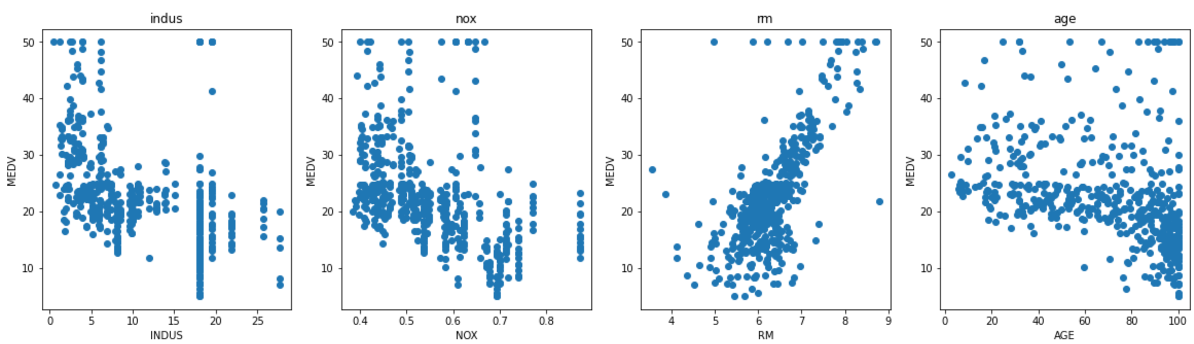


Observations:

indus, **nox**, **rm**, **age**, **tax**, **ptratio**, **lstat** show better correlation with **medv**

In [ ]:

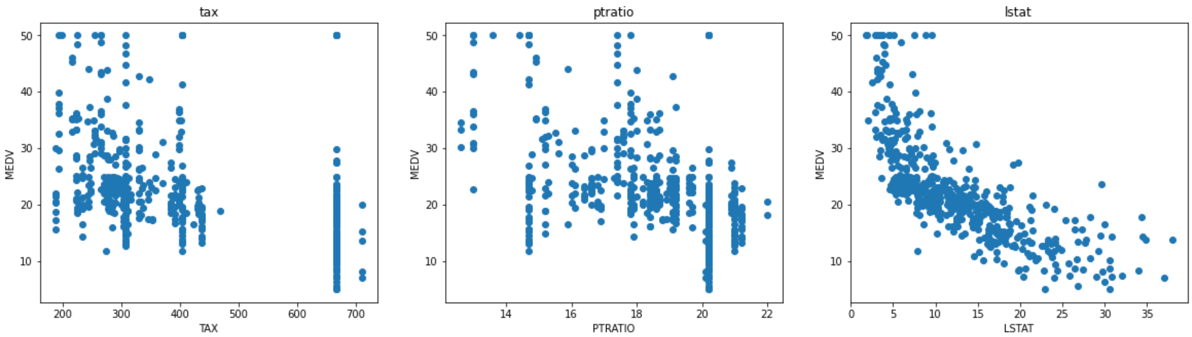
```
# scatter plot

plt.figure(figsize = (20, 5))

options = ["indus", "nox", "rm", "age"]
target = df["medv"]

for i, col in enumerate(options):
    plt.subplot(1, len(options) , i+1)
    x = df[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title(col)
    plt.xlabel(col.upper())
    plt.ylabel("MEDV")
```



In [ ]:

```
# scatter plot: To check CORRELATION as well
```

```python
plt.figure(figsize = (20, 5))

options = ["tax", "ptratio", "lstat"]
target = df["medv"]

for i, col in enumerate(options):
    plt.subplot(1, len(options) , i+1)
    x = df[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title(col)
    plt.xlabel(col.upper())
    plt.ylabel("MEDV")
```



In [ ]:

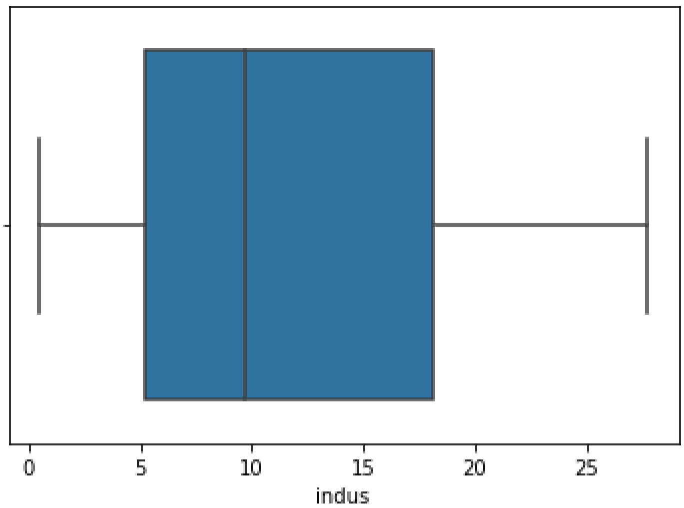
```python
dfc = df.copy()
```

Boxplot to Detect Outliers

In [ ]:

```python
sns.boxplot(x = dfc['indus'])
```

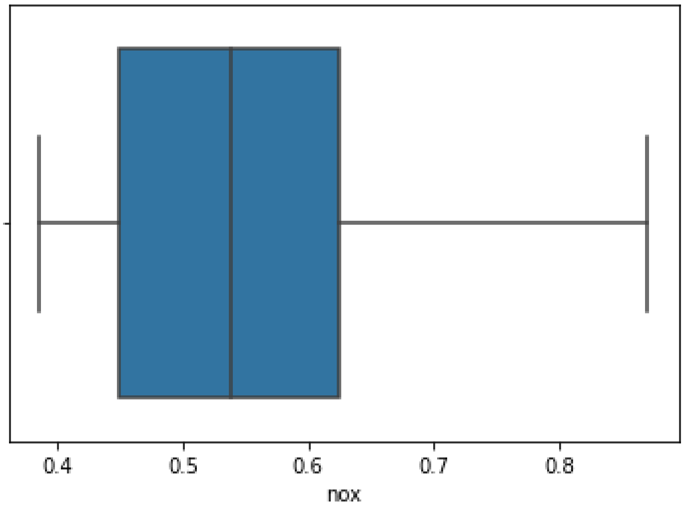
Out[ ]: <AxesSubplot:xlabel='indus'>



In [ ]:

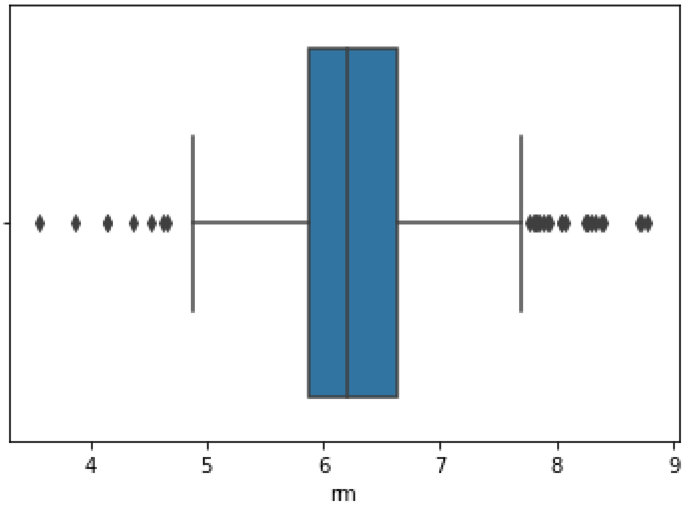
```python
sns.boxplot(x = dfc['nox'])
```

Out[ ]: <AxesSubplot:xlabel='nox'>



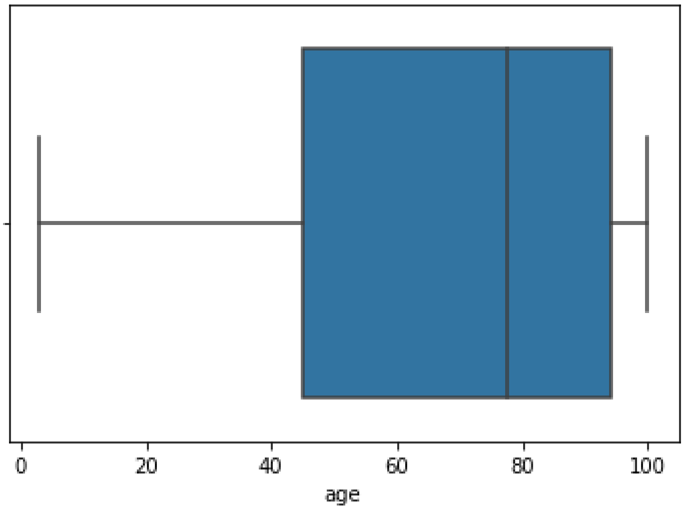
In [ ]:
```
sns.boxplot(x = dfc['rm'])
```

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



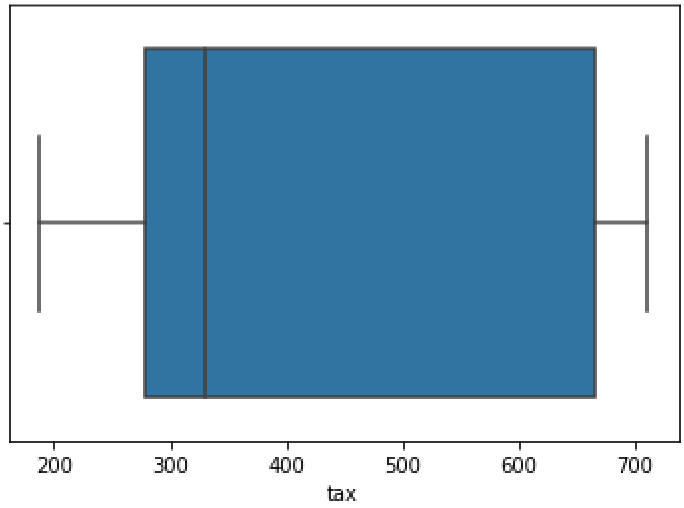
In [ ]:
```
sns.boxplot(x = dfc['age'])
```

Out[ ]: <AxesSubplot:xlabel='age'>



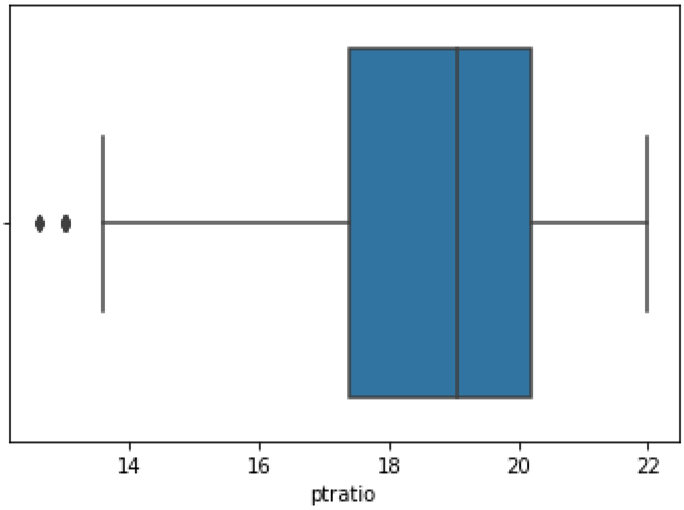
In [ ]:
```
sns.boxplot(x = dfc['tax'])
```

Out[ ]: <AxesSubplot:xlabel='tax'>



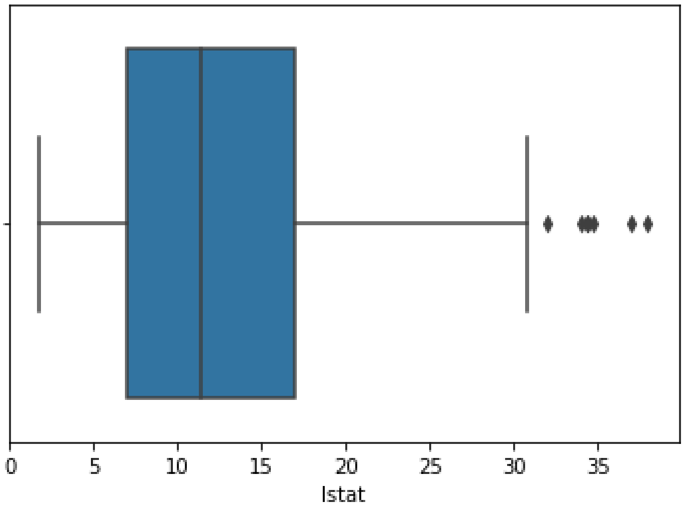
In [ ]:
```
sns.boxplot(x = dfc['ptratio'])
```

Out[ ]: <AxesSubplot:xlabel='ptratio'>



In [ ]:
```
sns.boxplot(x = dfc['lstat'])
```

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



Handling Outliers using Inter Quartile Range

In [ ]:

```
# to calculate inter quartile range

from scipy import stats
_cols = ['lstat']
result = stats.iqr(dfc[_cols], axis = 0)
result
```

Out[ ]:

```
array([10.005])
```

In [ ]:

```
# before removing outliers
dfc.shape
```

Out[ ]:

```
(506, 14)
```

In [ ]:

```
# dropping records that contain outliers

dfc.drop(dfc[dfc['lstat'] > (dfc['lstat'].quantile(0.75) + 1.5 * result[0])
```

In [ ]:

```
# after removing outliers
dfc.shape
```

Out[ ]:

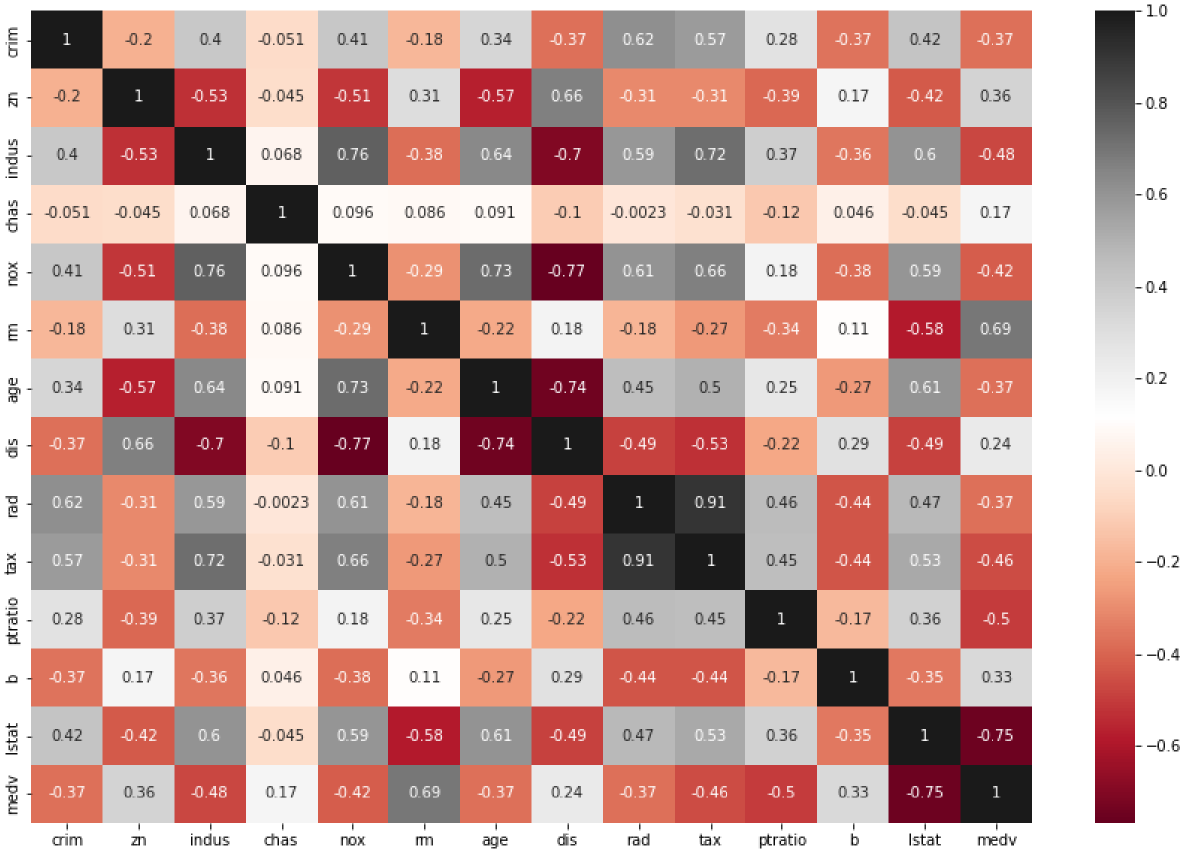
```
(499, 14)
```

In [ ]:

```
plt.figure(figsize = (15, 10))
sns.heatmap(data = dfc.corr(), annot = True, cmap = "RdGy")
# annot -> to print values inside the sqaure
```

Out[ ]:

```
<AxesSubplot:>
```



Splitting the Dataset for Training & Testing

After removal of **outliers**, the correlation matrix displays better results.

In [ ]:

```
# feature matrix
X = pd.DataFrame(np.c_[dfc["indus"], dfc["nox"], dfc["rm"], dfc["age"], dfc

# target variable
Y = dfc["medv"]

print("Feature Matrix X: \n", X)
print("\nTarget Variable Y:\n", Y)
```

```
Feature Matrix X:
      indus    nox     rm   age    tax  ptratio  lstat
0     2.31  0.538  6.575  65.2  296.0     15.3   4.98
1     7.07  0.469  6.421  78.9  242.0     17.8   9.14
2     7.07  0.469  7.185  61.1  242.0     17.8   4.03
3     2.18  0.458  6.998  45.8  222.0     18.7   2.94
4     2.18  0.458  7.147  54.2  222.0     18.7   5.33
..     ...    ...    ...   ...    ...      ...    ...
494  11.93  0.573  6.593  69.1  273.0     21.0   9.67
495  11.93  0.573  6.120  76.7  273.0     21.0   9.08
496  11.93  0.573  6.976  91.0  273.0     21.0   5.64
497  11.93  0.573  6.794  89.3  273.0     21.0   6.48
498  11.93  0.573  6.030  80.8  273.0     21.0   7.88

[499 rows x 7 columns]

Target Variable Y:
 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: 499, dtype: float64
```

In [ ]:

```
# Splitting the dataset into training and testing sets (80% training, 20% t

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, 
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(399, 7)
(100, 7)
(399,)
(100,)
```

Training the Linear Regression Model

In [ ]:

```
lin_reg_model = LinearRegression() #Using the ordinary Least Squares
lin_reg_model.fit(X_train, Y_train)
```

Out[ ]: LinearRegression()

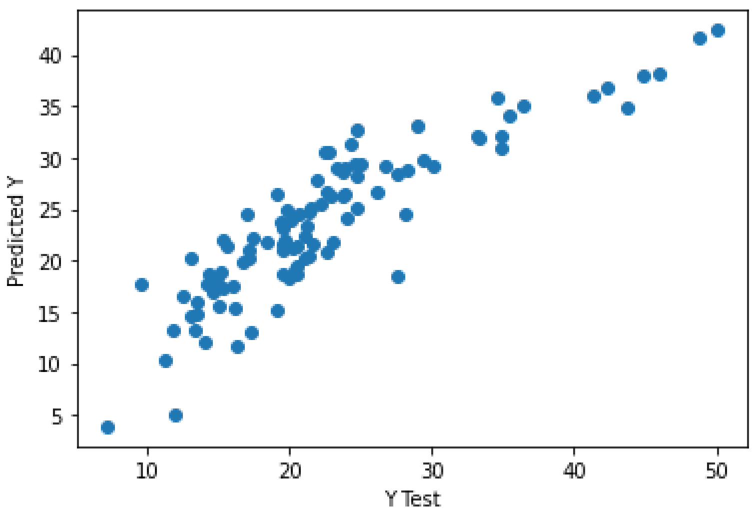
Testing the Model for Error & Accuracy

In [ ]:
```
# model performance for training set
train_prediction = lin_reg_model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, train_prediction)))
residual_error = (np.sqrt(mean_squared_error(Y_train,train_prediction))/np.
r2 = r2_score(Y_train, train_prediction) #(coefficient of determination) re

print("For training dataset:")
print("Root Mean Square Error is {}".format(rmse))
print("Residual Error is {}".format(residual_error))
print("Accuracy is {}%".format(r2 * 100))

# model performance for testing set
test_prediction = lin_reg_model.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, test_prediction)))
residual_error = (np.sqrt(mean_squared_error(Y_test,test_prediction))/np.sq
r2 = r2_score(Y_test, test_prediction)

print("\nFor testing dataset:")
print("Root Mean Square Error is {}".format(rmse))
print("Residual Error is {}".format(residual_error))
print("Accuracy is {}%".format(r2 * 100))
```

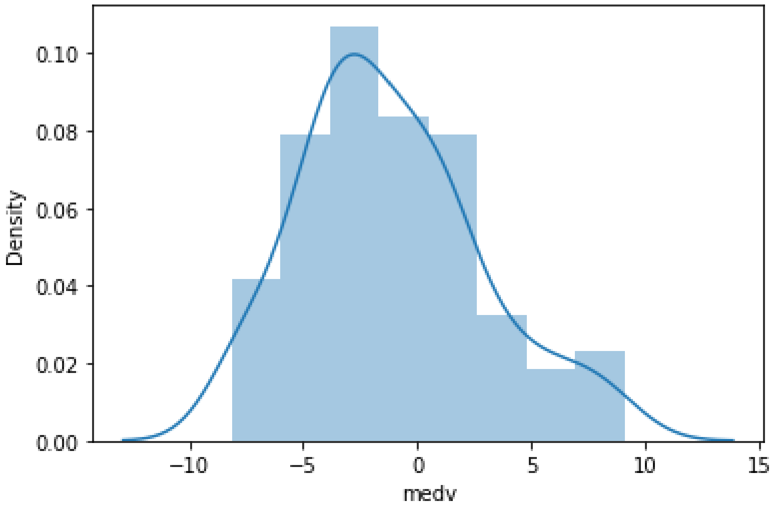
```
For training dataset:
Root Mean Square Error is 5.191650058154763
Residual Error is 0.2599075907031356
Accuracy is 68.64431543515799%

For testing dataset:
Root Mean Square Error is 4.149221462371583
Residual Error is 0.4149221462371583
Accuracy is 77.02415952526145%
```

In [ ]:
```
# Actual value VS Predicted value

plt.scatter(Y_test, test_prediction)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

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



In [ ]:
```
# Distribution plot for House Price

sns.distplot((Y_test - test_prediction));
```



In [ ]:
```
# Parameters / Coefficients for Linear Regression Model

coefficients = pd.DataFrame(lin_reg_model.coef_, X.columns)
coefficients.columns = ['coefficients']
coefficients
```

Out[ ]:

	coefficients
indus	0.050012
nox	-2.337444
rm	3.893782
age	0.031107
tax	-0.002464
ptratio	-0.993278
lstat	-0.733613

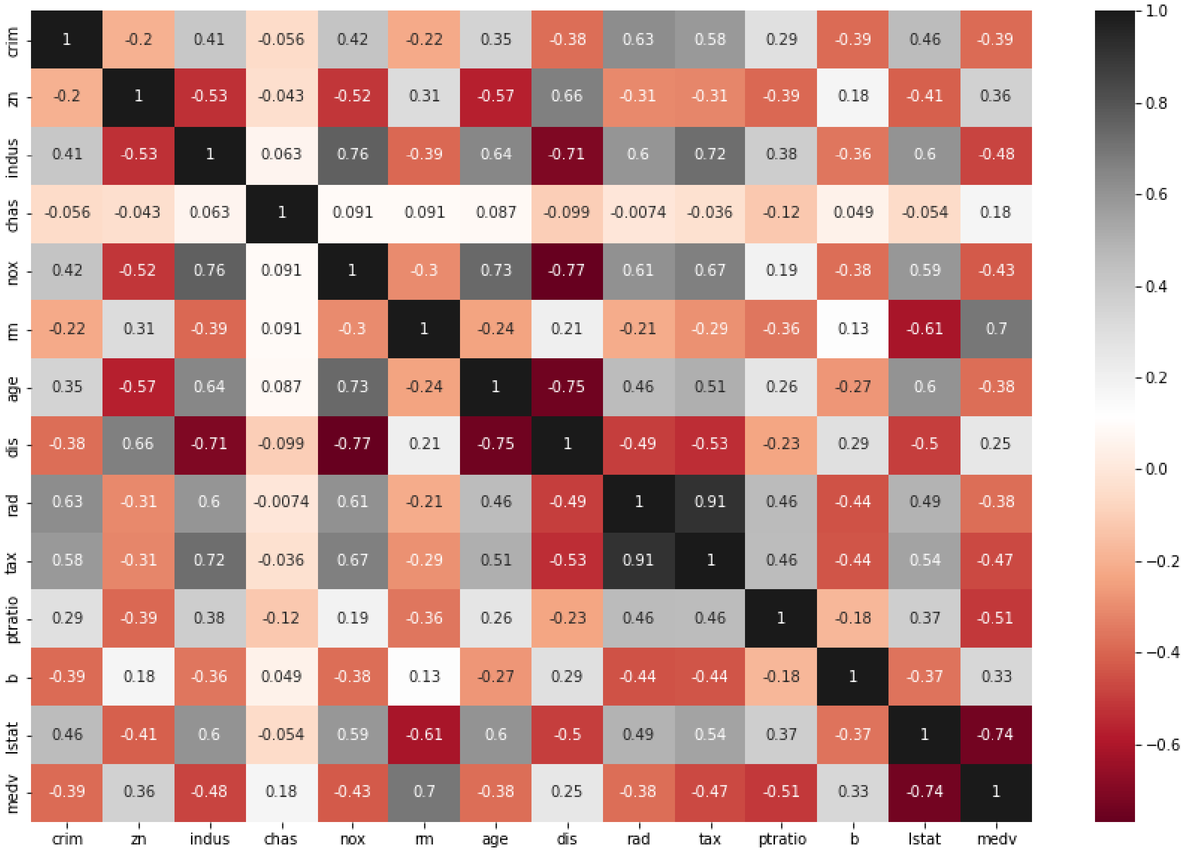
Trying to Improve Accuracy

Remove all the outliers in the dataset and check for accuracy

In [ ]:
```
dft = df.copy()
```

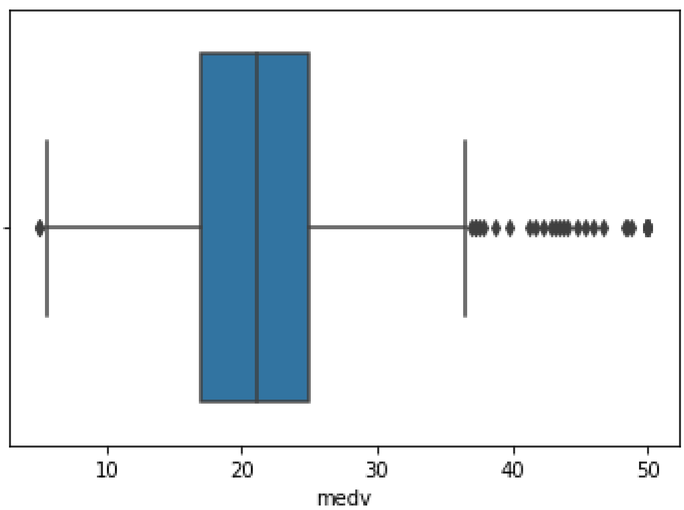
In [ ]:
```
plt.figure(figsize = (15, 10))
sns.heatmap(data = dft.corr(), annot = True, cmap = "RdGy")
```

Out[ ]: <AxesSubplot:>



In [ ]:
```
sns.boxplot(x = dft['medv'])
```

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



In [ ]:
```
from scipy import stats
_cols = ['lstat', 'medv', 'ptratio', 'rm']
result = stats.iqr(dft[_cols], axis = 0)
result
```

Out[ ]: `array([10.005,  7.975,  2.8  ,  0.738])`

In [ ]:
```
dft.shape
```

Out[ ]: `(506, 14)`

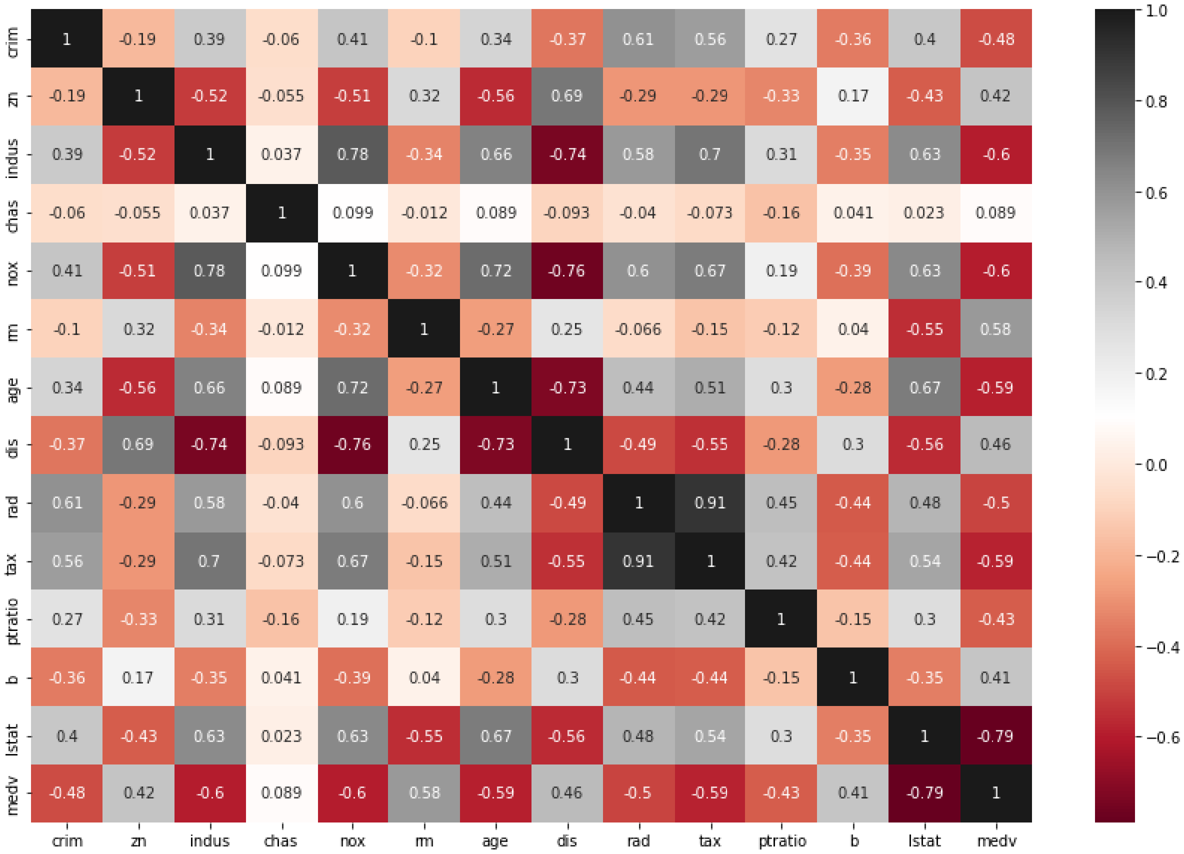
In [ ]:
```
dft.drop(dft[dft['lstat'] > (dft['lstat'].quantile(0.75) + 1.5 * result[0])
dft.drop(dft[dft['medv'] > (dft['medv'].quantile(0.75) + 1.5 * result[1])].
dft.drop(dft[dft['ptratio'] < (dft['ptratio'].quantile(0.25) - 1.5 * result
dft.drop(dft[dft['rm'] > (dft['rm'].quantile(0.75) + 1.5 * result[3])].inde
dft.drop(dft[dft['rm'] < (dft['rm'].quantile(0.25) - 1.5 * result[3])].inde
```

In [ ]:
```
dft.shape
```

Out[ ]:
```
(444, 14)
```

In [ ]:
```
plt.figure(figsize = (15, 10))
sns.heatmap(data = dft.corr(), annot = True, cmap = "RdGy")
```

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



In [ ]:
```
X = dft[['crim', 'zn', 'indus', 'nox', 'rm', 'age', 'dis', 'rad', 'tax', 'p
Y = dft["medv"]

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, 

print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)

print()
lin_reg_model = LinearRegression()
lin_reg_model.fit(X_train, Y_train)


# model performance for training set
train_prediction = lin_reg_model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, train_prediction)))
```

```
residual_error = (np.sqrt(mean_squared_error(Y_train,train_prediction))/np.
r2 = r2_score(Y_train, train_prediction)

print("For training dataset:")
print("Root Mean Square Error is {}".format(rmse))
print("Residual Error is {}".format(residual_error))
print("Accuracy is {}%".format(r2 * 100))

# model performance for testing set
test_prediction = lin_reg_model.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, test_prediction)))
residual_error = (np.sqrt(mean_squared_error(Y_test,test_prediction))/np.sq
r2 = r2_score(Y_test, test_prediction)

print("\nFor testing dataset:")
print("Root Mean Square Error is {}".format(rmse))
print("Residual Error is {}".format(residual_error))
print("Accuracy is {}%".format(r2 * 100))
```

```
(355, 12)
(89, 12)
(355,)
(89,)

For training dataset:
Root Mean Square Error is 2.8674641395521103
Residual Error is 0.15218919463057262
Accuracy is 79.07913582538792%

For testing dataset:
Root Mean Square Error is 3.0133028220379647
Residual Error is 0.3194094603177425
Accuracy is 74.82068387610576%
```

In [ ]: