The Boston Housing Dataset

The Boston Housing Dataset is a derived from information collected by the U.S. Census Service concerning housing in the area of Boston MA. The following describes the dataset columns:

- 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 dollars
- PTRATIO - pupil-teacher ratio by town
- B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT - percentagelower status of the population
- MEDV - Median value of owner-occupied homes in 1000's $

link to the data set: https://www.kaggle.com/vikrishnan/boston-house-prices (https://www.kaggle.com/vikrishnan/boston-house-prices)

In this notebook we perform Exploratory Data Analysis(EDA) and train an ensemble model(Decision Tree Regressor)

In [36]:

```
#importing data and creating dataframe

import numpy as np
import pandas as pd
import pandas_profiling as pdp
import matplotlib.pyplot as plt
import seaborn as sns
column_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PT
                'B', 'LSTAT', 'MEDV']
data = pd.read_csv('housing.csv',header=None, delimiter=r"\s+", names=column_names)
```

In [2]:

```
data.head(5)
```

Out[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5

In [3]:

```
data.tail(5)
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273.0	21.0	391.99	
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273.0	21.0	396.90	
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273.0	21.0	396.90	
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273.0	21.0	393.45	
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273.0	21.0	396.90	

In [4]:

```
#dimention of dataset
data.shape
```

Out[4]:

```
(506, 14)
```

In [5]:

```
#data type of each feature
data.dtypes
```

Out[5]:

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

In [6]:

```
#summary of all features
print(data.describe())
```

```
              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%      3.677082   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
```

Summary:

- ***ZN*** (proportion of residential land zoned for lots over 25,000 sq.ft.) has value 0 for 25th, 50th percentiles.
- ***CHAS***: Charles River dummy variable (1 if tract bounds river; 0 otherwise) has value 0 for 25th, 50th and 75th percentiles.

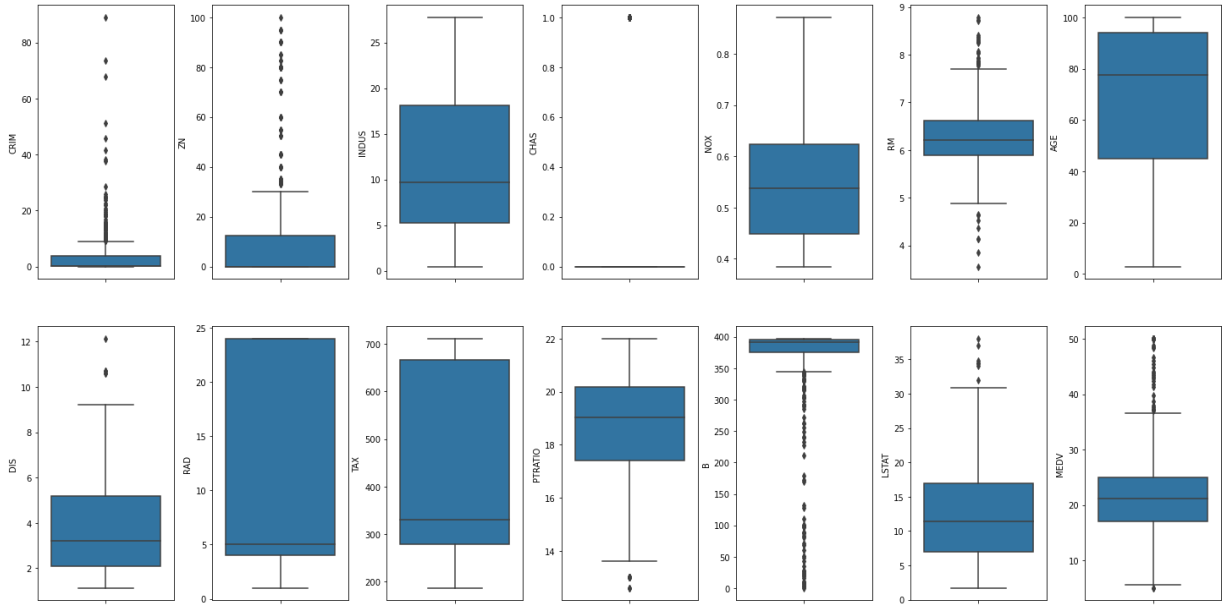
Hence , these 2 variables are conditional and categorical

Finding outliers in data using box-plots:

In [7]:

```
#box-plot of each features:

fig, axs = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
axs = axs.flatten()
for k,v in data.items():
    sns.boxplot(y=k, data=data, ax=axs[index])
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```



Summary:

CRIM, ZN, RM, B features seem to have outliers

In [8]:

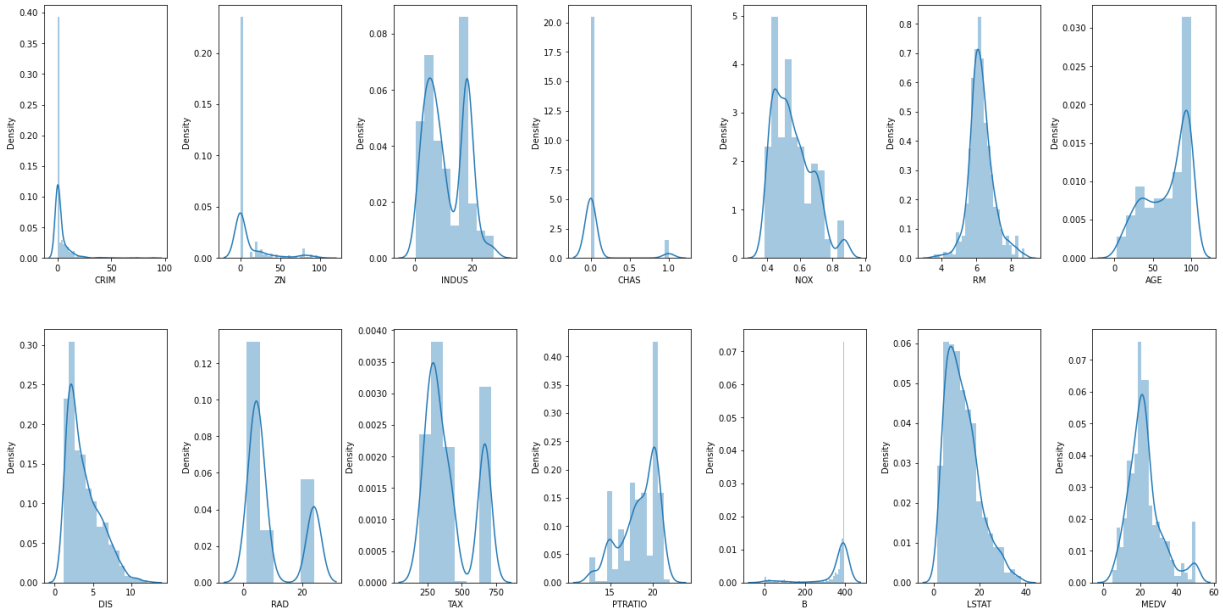
```
#calculating percentage of outliers
for k, v in data.items():
        q1 = v.quantile(0.25)
        q3 = v.quantile(0.75)
        irq = q3 - q1  #inter quantile range
        v_col = v[(v <= q1 - 1.5 * irq) | (v >= q3 + 1.5 * irq)]
        perc = np.shape(v_col)[0] * 100.0 / np.shape(data)[0]
        print("Column %s outliers = %.2f%%" % (k, perc))
```

```
Column CRIM outliers = 13.04%
Column ZN outliers = 13.44%
Column INDUS outliers = 0.00%
Column CHAS outliers = 100.00%
Column NOX outliers = 0.00%
Column RM outliers = 5.93%
Column AGE outliers = 0.00%
Column DIS outliers = 0.99%
Column RAD outliers = 0.00%
Column TAX outliers = 0.00%
Column PTRATIO outliers = 2.96%
Column B outliers = 15.22%
Column LSTAT outliers = 1.38%
Column MEDV outliers = 7.91%
```

Plotting distribution of each feature

In [9]:

```
import warnings
warnings.filterwarnings('ignore')
fig, axs = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
axs = axs.flatten()
for k,v in data.items():
    sns.distplot(v, ax=axs[index])
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```



Summary:

- ***CRIM, ZN, B*** has highly skewed distributions.
- ***MEDV*** looks to have a normal distribution (the predictions) and other colums seem to have norma or bimodel ditribution of data

Plotting corelation heatmap

In [13]:

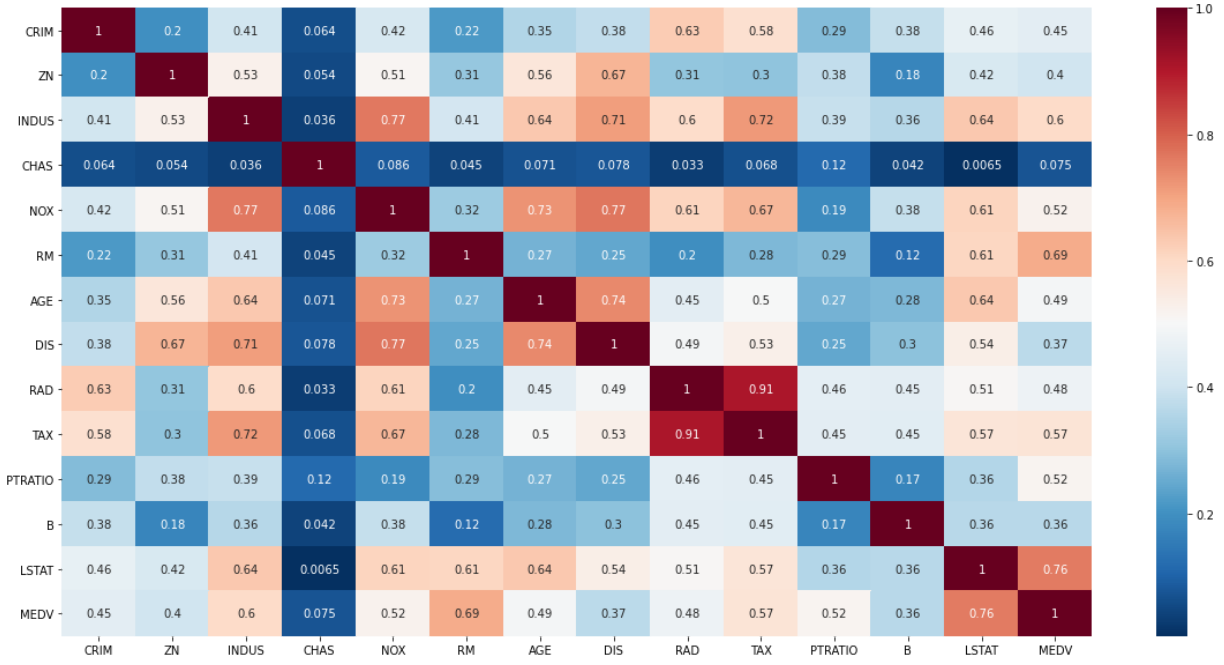
```
corr = data.corr()
data = data[~(data['MEDV'] >= 50.0)]
print(np.shape(data))


plt.figure(figsize=(20, 10))
sns.heatmap(corr.abs(),cmap='RdBu_r',annot=True)
```

```
(490, 14)
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c9c4708148>
```



Summary:

- TAX and RAD are highly correlated features
- columns LSTAT, INDUS, RM, TAX, NOX, PTRAIO has a correlation score above 0.5 with MEDV which is a good indication of using as predictors

Using pandas profiling to create a report

In [12]:

```
report = pdp.ProfileReport(data, title='Pandas Profiling Report')
report
```

Overview

Dataset statistics

Number of variables	15
Number of observations	490
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	57.5 KiB
Average record size in memory	120.3 B

Variable types

Numeric	14
Categorical	1

Warnings

`RAD` is highly correlated with `TAX`	**High correlation**
`TAX` is highly correlated with `RAD`	**High correlation**

Out[12]:

In [35]:

```
#training an ensamble model - decision tree regressor

#importing libraries
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
column_sels = ['LSTAT', 'INDUS', 'NOX', 'PTRATIO', 'RM', 'TAX', 'DIS', 'AGE']
x = data.loc[:,column_sels]
y = data['MEDV']
x = pd.DataFrame(data=min_max_scaler.fit_transform(x), columns=column_sels)
x_scaled = min_max_scaler.fit_transform(x)

y =  np.log1p(y)
for col in x.columns:
    if np.abs(x[col].skew()) > 0.3:
        x[col] = np.log1p(x[col])


kf = KFold(n_splits=10)
scores_map = {}

#fitting Gradient boosted regressor on model:
gbr = GradientBoostingRegressor(alpha=0.9,learning_rate=0.05, max_depth=2, min_samples_leaf

#cross validation to calculate best value of hyperparameters- GridSearch CV:

param_grid={'n_estimators':[100, 200], 'learning_rate': [0.1,0.05,0.02], 'max_depth':[2, 4,
grid_sv = GridSearchCV(gbr, cv=kf, param_grid=param_grid, scoring='neg_mean_squared_error')
grid_sv.fit(x_scaled, y)
#print("Best classifier :", grid_sv.best_estimator_)
scores = cross_val_score(gbr, x_scaled, y, cv=kf, scoring='neg_mean_squared_error')
scores_map['GradientBoostingRegressor'] = scores
print("MSE: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()))
```

```
MSE: -0.03 (+/- 0.02)
```

GradientBoostingRegressor shows good performance with -12.39 (+/- 5.86).