Assignment 4 - Data Analytics I

Atharva Taras (TE A - 73)

Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset ([https://www.kaggle.com/c/boston-housing (https://www.kaggle.com/c/boston-housing)](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.

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

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

In [2]:

```
data = pd.read_csv('housing.csv', delimiter=r"\s+", header=None, names=['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RA
data.head()
```

Out[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

In [3]:

```
data.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    float64
 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(12), int64(2)
memory usage: 55.5 KB
```

In [4]:

```
data.describe()
```

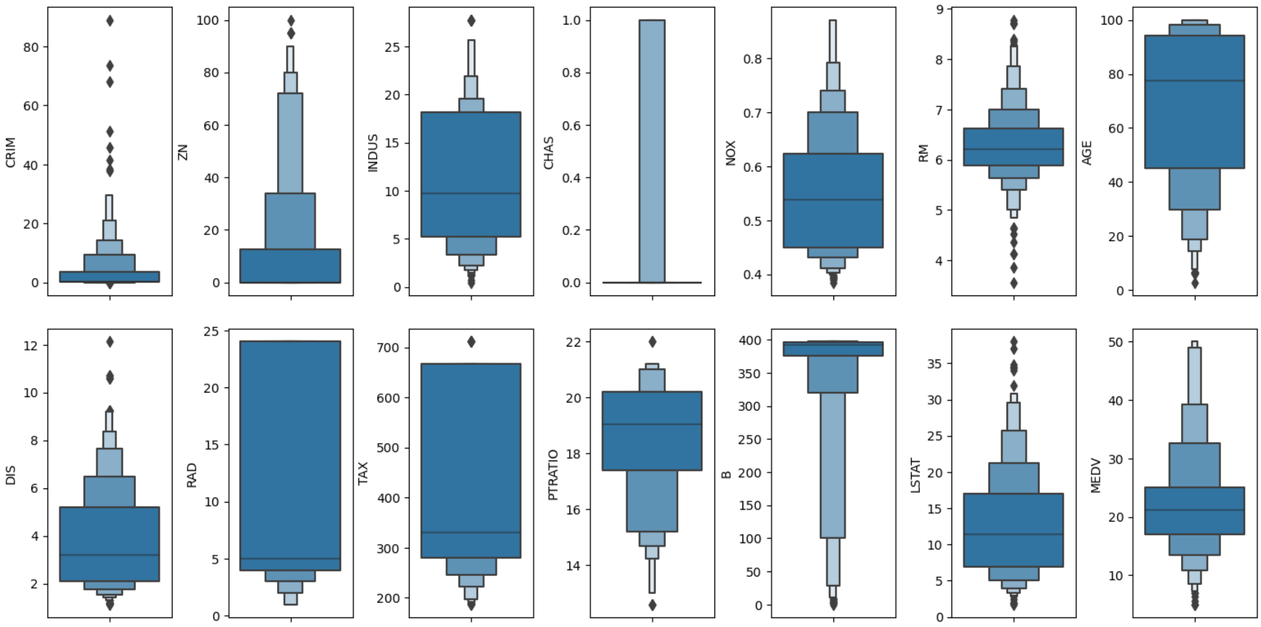
Out[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	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	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000

Checking outliers

In [5]:

```
fig, axs = plt.subplots(ncols=7, nrows=2, figsize=(14, 7))
index = 0
axs = axs.flatten()
for k,v in data.items():
    sns.boxenplot(y=k, data=data, ax=axs[index])
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=2.0)
```



In [6]:

```
print('Outliers per column \n')

for k, v in data.items():
        q1 = v.quantile(0.25)
        q3 = v.quantile(0.75)
        irq = q3 - q1

        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(f'{k} = {round(perc, 2)}%')
```

```
Outliers per column

CRIM = 13.04%
ZN = 13.44%
INDUS = 0.0%
CHAS = 100.0%
NOX = 0.0%
RM = 5.93%
AGE = 0.0%
DIS = 0.99%
RAD = 0.0%
TAX = 0.0%
PTRATIO = 2.96%
B = 15.22%
LSTAT = 1.38%
MEDV = 7.91%
```

Dropping 'CHAS' since it has too many outliers

In [7]:

```
data.drop('CHAS', axis=1, inplace=True)
```

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 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 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 1000sDolaars

In [8]:

```
boston = data.copy()
X = pd.DataFrame(np.c_[boston['LSTAT'], boston['RM'], boston['AGE'], boston['TAX'], boston['ZN']], columns = ['LSTAT','RM', 'AGE', 'T
Y = boston['MEDV']
```

In [9]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)
print(f' X-Train {X_train.shape} \n X-Test {X_test.shape} \n Y-Train {Y_train.shape} \n Y-Test {Y_test.shape}')
lin_model = LinearRegression()
lin_model.fit(X_train, Y_train)
```

```
 X-Train (404, 5)
 X-Test (102, 5)
 Y-Train (404,)
 Y-Test (102,)
```

Out[9]:

```
▾ LinearRegression
LinearRegression()
```

In [10]:

```
# Model evaluation for training set
y_train_predict = lin_model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2 = r2_score(Y_train, y_train_predict)
print("TRAINING SET \n-------------")
print('Root Mean Square Error (RMSE) is {}'.format(rmse))
print('Accuracy is {} % \n'.format(r2*100))

# Model evaluation for testing set
y_test_predict = lin_model.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
r2 = r2_score(Y_test, y_test_predict)
print("TESTING SET \n-------------")
print('Root Mean Square Error (RMSE) is {}'.format(rmse))
print('Accuracy is {} %'.format(r2*100))
```

```
TRAINING SET
-------------
Root Mean Square Error (RMSE) is 5.524657077725284
Accuracy is 64.4688793223649 %

TESTING SET
-------------
Root Mean Square Error (RMSE) is 5.0287069065809265
Accuracy is 67.70131100841681 %
```