Boston House Price Data Set,

- Predict the house Price

In [1]:

```
# Sklearn import boston dataset
from sklearn.datasets import load_boston

boston = load_boston()

type(boston)
```

```
/usr/local/lib/python3.7/site-packages/sklearn/utils/deprecation.py:87: F
utureWarning: Function load_boston is deprecated; `load_boston` is deprec
ated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can ref
er to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of
this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=Non
e)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :
2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames hous
ing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

Out[1]:

```
sklearn.utils.Bunch
```

In [2]:

```
boston.keys()
```

Out[2]:

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename', 'data_
```

```
module'])
```

In [3]:

```
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Valu
e (attribute 14) is usually the target.

    :Attribute Information (in order):
        - 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 rive
r; 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 black
people by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Ca
rnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnos
tics
...', Wiley, 1980.   N.B. Various transformations are used in the table o
n
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers
that address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influent
ial Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learnin
```

```
g. In Proceedings on the Tenth International Conference of Machine Learni
ng, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
```

In [4]:

```python
boston.data
```

Out[4]:

```
array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
         4.9800e+00],
        [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
         9.1400e+00],
        [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
         4.0300e+00],
        ...,
        [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         5.6400e+00],
        [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
         6.4800e+00],
        [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         7.8800e+00]])
```

In [5]:

```python
# Create data frame with all features
import pandas as pd
boston_df = pd.DataFrame(boston.data , columns= boston.feature_names)
boston_df
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.9
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.8
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.6
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.9
...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.9
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.9
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.4
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.9

506 rows × 13 columns

In [6]:

```python
boston_df['MEDV'] = boston.target

boston_df.head()
```

Out[6]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90

In [7]:

```
boston_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    float64
 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    float64
 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(14)
memory usage: 55.5 KB
```

In [8]:

```
boston_df.describe().T
```

Out[8]:

	count	mean	std	min	25%	50%	75%	
CRIM	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.677083	88
ZN	506.0	11.363636	23.322453	0.00000	0.000000	0.00000	12.500000	100
INDUS	506.0	11.136779	6.860353	0.46000	5.190000	9.69000	18.100000	27
CHAS	506.0	0.069170	0.253994	0.00000	0.000000	0.00000	0.000000	1
NOX	506.0	0.554695	0.115878	0.38500	0.449000	0.53800	0.624000	0
RM	506.0	6.284634	0.702617	3.56100	5.885500	6.20850	6.623500	8
AGE	506.0	68.574901	28.148861	2.90000	45.025000	77.50000	94.075000	100
DIS	506.0	3.795043	2.105710	1.12960	2.100175	3.20745	5.188425	12
RAD	506.0	9.549407	8.707259	1.00000	4.000000	5.00000	24.000000	24
TAX	506.0	408.237154	168.537116	187.00000	279.000000	330.00000	666.000000	711
PTRATIO	506.0	18.455534	2.164946	12.60000	17.400000	19.05000	20.200000	22
B	506.0	356.674032	91.294864	0.32000	375.377500	391.44000	396.225000	396
LSTAT	506.0	12.653063	7.141062	1.73000	6.950000	11.36000	16.955000	37
MEDV	506.0	22.532806	9.197104	5.00000	17.025000	21.20000	25.000000	50

In [9]:

```
#Checking for null values
boston_df.isnull().sum()
```

Out[9]:

```
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
```

Splitting X & y

X - variable which holds all the features

y - variable which holds the target value

In [10]:

```
# splitting X & y from the bunch object
X = boston.data
y = boston.target
print(X.shape)
```

```
(506, 13)
```

In [11]:

```
# splitting X & y from the dataframe
X = boston_df.drop('MEDV' , axis=1)
y = boston_df['MEDV']
```

Split the dataset into train set and test set

In [14]:

```
from sklearn.model_selection import train_test_split
```

In [15]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y , test_size = 0

# test size = (0,1)
# 0 - denotes that the entire data has been considered for training
#1 - denotes that the entire data has been considered for testing
# 0.2 -- 20% of data is taken for testing
# 0.3 -- 30% of data is taken for testing
```

In [16]:

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(404, 13)
(102, 13)
(404,)
(102,)
```

Build regression model

In [17]:
```
from sklearn.linear_model import LinearRegression #Import the necessary 

model_lr = LinearRegression() # instantiate the estimator object
#estimator object is model_lr
```

In [18]:
```
#training the model with the data of this problem statement
model_lr.fit(X_train,y_train)
```

Out[18]:
```
LinearRegression()
```

In [19]:
```
print("The intercept for the LR model is : " ,model_lr.intercept_)
```

```
The intercept for the LR model is :  40.653176529790805
```

In [20]:
```
pd.DataFrame({'Feature name': boston.feature_names , 'Coefficient' : mod
```

Out[20]:

	Feature name	Coefficient
0	CRIM	-0.087742
1	ZN	0.048777
2	INDUS	0.019475
3	CHAS	3.063144
4	NOX	-18.482116
5	RM	3.347042
6	AGE	0.003220
7	DIS	-1.425695
8	RAD	0.325184
9	TAX	-0.012026
10	PTRATIO	-1.055828
11	B	0.010768
12	LSTAT	-0.538357

In [21]:
```
print("The coefficient for the LR model is : " ,model_lr.coef_)
```

```
The coefficient for the LR model is :  [-8.77422649e-02  4.87770336e-02
1.94746142e-02  3.06314365e+00
 -1.84821160e+01  3.34704170e+00  3.22024333e-03 -1.42569490e+00
  3.25184188e-01 -1.20259158e-02 -1.05582832e+00  1.07682087e-02
 -5.38356500e-01]
```

Evaluate the regression model

In [22]:
```
y_pred_test = model_lr.predict(X_test)
```

In [23]:
```
pd.DataFrame({'Actual y_test': y_test , 'Predicted y_test' : y_pred_test
```

Actual y_test	Predicted y_test

Out[23]:

455	14.1	15.311568
142	13.4	15.324187
311	22.1	26.890855
232	41.7	37.384876
290	28.5	33.375220
...	...	...
486	19.1	20.027912
468	19.1	17.513802
302	26.4	29.172278
244	17.6	16.904621
321	23.1	24.767296

102 rows × 2 columns

In [24]:

```
from sklearn.metrics import mean_squared_error

import numpy as np

print ('RMSE value of testing dataset')

print(np.sqrt(mean_squared_error(y_test,y_pred_test)))
```

```
RMSE value of testing dataset
5.179324335658013
```

In [25]:

```
## figuring out R2 score

print("R2 score is : {}".format(model_lr.score(X_test,y_test)))
```

```
R2 score is : 0.714936416139222
```

In [ ]:

```

```