Lab 6 - CKCS 113 Intro to Machine Learning

Answer the following questions and submit a PDF file on the D2L.

XGBoost Algorithm XGBoost is one of the most popular machine learning algorithm these days. Regardless of the type of prediction task at hand; regression or classification.

In [2]:
```
%autosave 20
```

```
Autosaving every 20 seconds
```

https://xgboost.readthedocs.io/en/latest/ (https://xgboost.readthedocs.io/en/latest/)

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html (https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html)

https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155 (https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155)

https://medium.com/@haydar_ai/learning-data-science-day-9-linear-regression-on-boston-housing-dataset-cd62a80775ef (https://medium.com/@haydar_ai/learning-data-science-day-9-linear-regression-on-boston-housing-dataset-cd62a80775ef)

Import the Boston Housing dataset from scikit-learn and store it in a variable called boston_dataset.

In [3]:
```
from sklearn.datasets import load_boston
#import sklearn.datasets.load_boston()
```

In [4]:
```
boston_dataset = load_boston(return_X_y=False)
```

In [5]:
```
boston_dataset
```

Out[5]:
```
{'data': 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]]),
 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9,
15. ,
        18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
        15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
        13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
        21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
        35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
        19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
        20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
        23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
        33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
        21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
        20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
        23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
        15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
        17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
        25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
        23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
        32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
        34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
        20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
        26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
        31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
        22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
        42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
        36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
        32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
        20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
        20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
        22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
        21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
        19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
        32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
        18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
        16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
        13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
         7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
        12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
        27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
         8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
         9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
        10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
        15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
        19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
        29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
        20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
        23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9]),
 'feature_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'D
IS', 'RAD'
```

Print keys of boston dataset

In [6]:
```
# o/p =dict_keys(['data', 'target', 'feature_names', 'DESCR'])
```

In [7]:
```
boston_dataset.keys()
```

Out[7]:
```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

In [8]:
```
boston_dataset.data
```

Out[8]:
```
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 [9]:
```
boston_dataset.feature_names
```

Out[9]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [10]:
```
len(boston_dataset)
```

Out[10]:
```
5
```

In [11]:
```
print(boston_dataset.target)
```

```
[24.  21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15.  18.9 21.7 20.4
 18.2 19.9 23.1 17.5 20.2 18.2 13.6 19.6 15.2 14.5 15.6 13.9 16.6 14.8
 18.4 21.  12.7 14.5 13.2 13.1 13.5 18.9 20.  21.  24.7 30.8 34.9 26.6
 25.3 24.7 21.2 19.3 20.  16.6 14.4 19.4 19.7 20.5 25.  23.4 18.9 35.4
 24.7 31.6 23.3 19.6 18.7 16.  22.2 25.  33.  23.5 19.4 22.  17.4 20.9
 24.2 21.7 22.8 23.4 24.1 21.4 20.  20.8 21.2 20.3 28.  23.9 24.8 22.9
 23.9 26.6 22.5 22.2 23.6 28.7 22.6 22.  22.9 25.  20.6 28.4 21.4 38.7
 43.8 33.2 27.5 26.5 18.6 19.3 20.1 19.5 19.5 20.4 19.8 19.4 21.7 22.8
 18.8 18.7 18.5 18.3 21.2 19.2 20.4 19.3 22.  20.3 20.5 17.3 18.8 21.4
 15.7 16.2 18.  14.3 19.2 19.6 23.  18.4 15.6 18.1 17.4 17.1 13.3 17.8
 14.  14.4 13.4 15.6 11.8 13.8 15.6 14.6 17.8 15.4 21.5 19.6 15.3 19.4
 17.  15.6 13.1 41.3 24.3 23.3 27.  50.  50.  50.  22.7 25.  50.  23.8
 23.8 22.3 17.4 19.1 23.1 23.6 22.6 29.4 23.2 24.6 29.9 37.2 39.8 36.2
 37.9 32.5 26.4 29.6 50.  32.  29.8 34.9 37.  30.5 36.4 31.1 29.1 50.
 33.3 30.3 34.6 34.9 32.9 24.1 42.3 48.5 50.  22.6 24.4 22.5 24.4 20.
 21.7 19.3 22.4 28.1 23.7 25.  23.3 28.7 21.5 23.  26.7 21.7 27.5 30.1
 44.8 50.  37.6 31.6 46.7 31.5 24.3 31.7 41.7 48.3 29.  24.  25.1 31.5
 23.7 23.3 22.  20.1 22.2 23.7 17.6 18.5 24.3 20.5 24.5 26.2 24.4 24.8
 29.6 42.8 21.9 20.9 44.  50.  36.  30.1 33.8 43.1 48.8 31.  36.5 22.8
 30.7 50.  43.5 20.7 21.1 25.2 24.4 35.2 32.4 32.  33.2 33.1 29.1 35.1
 45.4 35.4 46.  50.  32.2 22.  20.1 23.2 22.3 24.8 28.5 37.3 27.9 23.9
 21.7 28.6 27.1 20.3 22.5 29.  24.8 22.  26.4 33.1 36.1 28.4 33.4 28.2
 22.8 20.3 16.1 22.1 19.4 21.6 23.8 16.2 17.8 19.8 23.1 21.  23.8 23.1
 20.4 18.5 25.  24.6 23.  22.2 19.3 22.6 19.8 17.1 19.4 22.2 20.7 21.1
 19.5 18.5 20.6 19.  18.7 32.7 16.5 23.9 31.2 17.5 17.2 23.1 24.5 26.6
 22.9 24.1 18.6 30.1 18.2 20.6 17.8 21.7 22.7 22.6 25.  19.9 20.8 16.8
 21.9 27.5 21.9 23.1 50.  50.  50.  50.  50.  13.8 13.8 15.  13.9 13.3
 13.1 10.2 10.4 10.9 11.3 12.3  8.8  7.2 10.5  7.4 10.2 11.5 15.1 23.2
  9.7 13.8 12.7 13.1 12.5  8.5  5.   6.3  5.6  7.2 12.1  8.3  8.5  5.
 11.9 27.9 17.2 27.5 15.  17.2 17.9 16.3  7.   7.2  7.5 10.4  8.8  8.4
 16.7 14.2 20.8 13.4 11.7  8.3 10.2 10.9 11.   9.5 14.5 14.1 16.1 14.3
 11.7 13.4  9.6  8.7  8.4 12.8 10.5 17.1 18.4 15.4 10.8 11.8 14.9 12.6
 14.1 13.  13.4 15.2 16.1 17.8 14.9 14.1 12.7 13.5 14.9 20.  16.4 17.7
 19.5 20.2 21.4 19.9 19.  19.1 19.1 20.1 19.9 19.6 23.2 29.8 13.8 13.3
 16.7 12.  14.6 21.4 23.  23.7 25.  21.8 20.6 21.2 19.1 20.6 15.2  7.
  8.1 13.6 20.1 21.8 24.5 23.1 19.7 18.3 21.2 17.5 16.8 22.4 20.6 23.9
 22.  11.9]
```

In [12]:
```
print(boston_dataset.DESCR)
```

```
.. _boston_dataset:

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

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (at
tribute 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 s
q.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 t
own
        - 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 Carnegi
e 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 diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
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 Influential D
ata and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In
Proceedings on the Tenth International Conference of Machine Learning, 236-24
3, University of Massachusetts, Amherst. Morgan Kaufmann.
```

This is not a data frame!

so we ned to create a df DataFrame NOW from DataSet.

In [13]:
```
import pandas
```

In [14]:
```
# this is wrong, it will have no data as .data is missing
pandas.DataFrame(data=boston_dataset, columns=boston_dataset.feature_names)
# this is wrong, it will have no data as .data is missing
```

Out[14]:

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT

In [15]:
```
df_boston = pandas.DataFrame(data=boston_dataset.data, columns=boston_dataset.fe
ature_names)
df_boston
```

Out[15]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
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	4.98
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	9.14
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	4.03
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	2.94
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	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

506 rows × 13 columns

Print dimensions of this dataset

In [16]:
```
# o/p =(506, 13)
```

In [17]:
```
df_boston.shape
```

Out[17]:
```
(506, 13)
```

In [18]:
```
df_boston.describe
```

Out[18]:
```
<bound method NDFrame.describe of         CRIM    ZN  INDUS  CHAS    NOX     R
M   AGE     DIS  RAD    TAX  \
0    0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1    0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2    0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3    0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4    0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0
..       ...   ...    ...   ...    ...    ...   ...     ...  ...    ...
501  0.06263   0.0  11.93   0.0  0.573  6.593  69.1  2.4786  1.0  273.0
502  0.04527   0.0  11.93   0.0  0.573  6.120  76.7  2.2875  1.0  273.0
503  0.06076   0.0  11.93   0.0  0.573  6.976  91.0  2.1675  1.0  273.0
504  0.10959   0.0  11.93   0.0  0.573  6.794  89.3  2.3889  1.0  273.0
505  0.04741   0.0  11.93   0.0  0.573  6.030  80.8  2.5050  1.0  273.0

     PTRATIO       B  LSTAT
0       15.3  396.90   4.98
1       17.8  396.90   9.14
2       17.8  392.83   4.03
3       18.7  394.63   2.94
4       18.7  396.90   5.33
..       ...     ...    ...
501     21.0  391.99   9.67
502     21.0  396.90   9.08
503     21.0  396.90   5.64
504     21.0  393.45   6.48
505     21.0  396.90   7.88

[506 rows x 13 columns]>
```

Print all features

In [19]:
```
print(boston_dataset.feature_names)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

In [20]:
```
df_boston.head(0)
```

Out[20]:

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT

In [21]:
```
print(df_boston.head(0))
```

```
Empty DataFrame
Columns: [CRIM, ZN, INDUS, CHAS, NOX, RM, AGE, DIS, RAD, TAX, PTRATIO, B, LSTA
T]
Index: []
```

Print description of the dataset

o/p =

"Boston House Prices dataset\n===========================\n\nNotes\n------\nData Set Characteristics: \n\n :Number of Instances: 506 \n\n :Number of Attributes: 13 numeric/categorical predictive\n \n :Median Value (attribute 14) is usually the target\n\n :Attribute Information (in order):\n - CRIM per capita crime rate by town\n - ZN proportion of residential land zoned for lots over 25,000 sq.ft.\n - INDUS proportion of non-retail business acres per town\n - CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)\n - NOX nitric oxides concentration (parts per 10 million)\n - RM average number of rooms per dwelling\n - AGE proportion of owner-occupied units built prior to 1940\n - DIS weighted distances to five Boston employment centres\n - RAD index of accessibility to radial highways\n - TAX full-value property-tax rate per

$10,000\backslash n - PTRATIO pupil - teacher ratio by town \backslash n$ 1000's\n\n :Missing Attribute

$- B 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town \backslash n - LSTAT$

Values: None\n\n :Creator: Harrison, D. and Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing dataset.\nhttp://archive.ics.uci.edu/ml/datasets/Housing\n\n\nThis (http://archive.ics.uci.edu/ml/datasets/Housing\n\n\nThis) dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.\n\nThe Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic\nprices and the demand for clean air', J. Environ. Economics & Management,\nvol.5, 81-102, 1978. Used in Belsley, Kuh & Welsch, 'Regression diagnostics\n...', Wiley, 1980. N.B. Various transformations are used in the table on\npages 244-261 of the latter.\n\nThe Boston house-price data has been used in many machine learning papers that address regression\nproblems. \n \n**References**\n\n - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.\n - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.\n - many more! (see http://archive.ics.uci.edu/ml/datasets/Housing)\n (http://archive.ics.uci.edu/ml/datasets/Housing)\n)"

In [22]:
```
boston_dataset.DESCR
```

Out[22]:
```
".. _boston_dataset:\n\nBoston house prices dataset\n
---------------------------\n\n**Data Set Characteristics:**  \n\n    :Number
of Instances: 506 \n\n    :Number of Attributes: 13 numeric/categorical predic
tive. Median Value (attribute 14) is usually the target.\n\n    :Attribute Inf
ormation (in order):\n        - CRIM     per capita crime rate by town\n
- ZN       proportion of residential land zoned for lots over 25,000 sq.ft.\n
- INDUS    proportion of non-retail business acres per town\n        - CHAS
Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)\n
- NOX      nitric oxides concentration (parts per 10 million)\n        - RM
average number of rooms per dwelling\n        - AGE      proportion of owner-o
ccupied units built prior to 1940\n        - DIS      weighted distances to fi
ve Boston employment centres\n        - RAD      index of accessibility to rad
ial highways\n        - TAX      full-value property-tax rate per $10,000\n
- PTRATIO  pupil-teacher ratio by town\n        - B        1000(Bk - 0.63)^2 w
here Bk is the proportion of blacks by town\n        - LSTAT    % lower status
of the population\n        - MEDV     Median value of owner-occupied homes in
$1000's\n\n    :Missing Attribute Values: None\n\n    :Creator: Harrison, D. a
nd Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing dataset.\nhttps://archi
ve.ics.uci.edu/ml/machine-learning-databases/housing/\n\n\nThis dataset was ta
ken from the StatLib library which is maintained at Carnegie Mellon Universit
y.\n\nThe Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedoni
c\nprices and the demand for clean air', J. Environ. Economics & Management,\n
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics\
n...', Wiley, 1980.   N.B. Various transformations are used in the table on\np
ages 244-261 of the latter.\n\nThe Boston house-price data has been used in ma
ny machine learning papers that address regression\nproblems.   \n     \n.. to
pic:: References\n\n   - Belsley, Kuh & Welsch, 'Regression diagnostics: Ident
ifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.\n
- Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Pro
ceedings on the Tenth International Conference of Machine Learning, 236-243, U
niversity of Massachusetts, Amherst. Morgan Kaufmann.\n"
```

Create a DataFrame from boston.data

In [23]:
```
df_boston = pandas.DataFrame(data=boston_dataset.data, columns=boston_dataset.fe
ature_names)
```

In [24]:
```
df_boston.head(3)
```

Out[24]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
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	4.98
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	9.14
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	4.03

Print 5 records from the dataset

In [25]:
```
df_boston.sample(5)
```

Out[25]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
430	8.49213	0.0	18.10	0.0	0.584	6.348	86.1	2.0527	24.0	666.0	20.2	83.45	17.64
53	0.04981	21.0	5.64	0.0	0.439	5.998	21.4	6.8147	4.0	243.0	16.8	396.90	8.43
469	13.07510	0.0	18.10	0.0	0.580	5.713	56.7	2.8237	24.0	666.0	20.2	396.90	14.76
224	0.31533	0.0	6.20	0.0	0.504	8.266	78.3	2.8944	8.0	307.0	17.4	385.05	4.14
66	0.04379	80.0	3.37	0.0	0.398	5.787	31.1	6.6115	4.0	337.0	16.1	396.90	10.24

Add boston.target to your pandas DataFrame with the name PRICE

In [26]:
```
df_boston["CRIM"].sample(5)
```

Out[26]:
```
489    0.18337
54     0.01360
471    4.03841
55     0.01311
93     0.02875
Name: CRIM, dtype: float64
```

In [27]:
```
#Here we created a new column into our dataset named : "boston_dataset"
df_boston["PRICE"] = boston_dataset.target
# price is column name for dF dataFrame
# target is value getting from dataset going into this newly created column("pri
ce")
```

In [28]:
```
#check its created or not
df_boston.sample(4)
```

Out[28]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
396	5.87205	0.0	18.1	0.0	0.693	6.405	96.0	1.6768	24.0	666.0	20.2	396.9	19.37	12.5
234	0.44791	0.0	6.2	1.0	0.507	6.726	66.5	3.6519	8.0	307.0	17.4	360.2	8.05	29.0
54	0.01360	75.0	4.0	0.0	0.410	5.888	47.6	7.3197	3.0	469.0	21.1	396.9	14.80	18.9
311	0.79041	0.0	9.9	0.0	0.544	6.122	52.8	2.6403	4.0	304.0	18.4	396.9	5.98	22.1

Call the info and describe methods for the dataframe data

o/p is =

<class 'pandas.core.frame.DataFrame'> RangeIndex: 506 entries, 0 to 505 Data columns (total 14 columns): CRIM 506 non-null float64 ZN 506 non-null float64 INDUS 506 non-null float64 CHAS 506 non-null float64 NOX 506 non-null float64 RM 506 non-null float64 AGE 506 non-null float64 DIS 506 non-null float64 RAD 506 non-null float64 TAX 506 non-null float64 PTRATIO 506 non-null float64 B 506 non-null float64 LSTAT 506 non-null float64 PRICE 506 non-null float64 dtypes: float64(14) memory usage: 55.4 KB

In [29]:
```
df_boston.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
CRIM       506 non-null float64
ZN         506 non-null float64
INDUS      506 non-null float64
CHAS       506 non-null float64
NOX        506 non-null float64
RM         506 non-null float64
AGE        506 non-null float64
DIS        506 non-null float64
RAD        506 non-null float64
TAX        506 non-null float64
PTRATIO    506 non-null float64
B          506 non-null float64
LSTAT      506 non-null float64
PRICE      506 non-null float64
dtypes: float64(14)
memory usage: 55.5 KB
```

In [30]:
```
df_boston.describe
```

Out[30]:
```
<bound method NDFrame.describe of         CRIM    ZN  INDUS  CHAS    NOX     R
M   AGE     DIS  RAD    TAX  \
0    0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1    0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2    0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3    0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4    0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0
..       ...   ...    ...   ...    ...    ...   ...     ...  ...    ...
501  0.06263   0.0  11.93   0.0  0.573  6.593  69.1  2.4786  1.0  273.0
502  0.04527   0.0  11.93   0.0  0.573  6.120  76.7  2.2875  1.0  273.0
503  0.06076   0.0  11.93   0.0  0.573  6.976  91.0  2.1675  1.0  273.0
504  0.10959   0.0  11.93   0.0  0.573  6.794  89.3  2.3889  1.0  273.0
505  0.04741   0.0  11.93   0.0  0.573  6.030  80.8  2.5050  1.0  273.0

     PTRATIO       B  LSTAT  PRICE
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   5.33   36.2
..       ...     ...    ...    ...
501     21.0  391.99   9.67   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]>
```

Import the XGB Algorithm, mean_squared_error and numpy

bash conda install py-xgboost

In [31]:
```
import xgboost
from sklearn.metrics import mean_squared_error
import numpy
```

In [32]:
```
df_boston.head(3)
```

Out[32]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
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	4.98	24.0
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	9.14	21.6
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	4.03	34.7

Select your X and y from the data

In [33]:
```
x = df_boston.iloc[:,:-1]
#y = df_boston.iloc[0:len(df_boston),-1:len(df_boston)]# this will give me the l
ast only column
y = df_boston.iloc[:,-1]
```

In [34]:
```
len(df_boston) #this is for rows
```

Out[34]: 506

In [35]:
```
len(df_boston.columns)
```

Out[35]: 14

In [36]:
```
z = df_boston.iloc[0:len(df_boston),0:-1] #[R:C,R:C] #iloc= integer based locati
on
#y = df_boston.iloc[0:len(df_boston),-1:len(df_boston)]# this will give me the l
ast only column

z.head(3)
```

Out[36]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
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	4.98
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	9.14
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	4.03

In [37]:
```
y.head(3)
```

Out[37]:
```
0    24.0
1    21.6
2    34.7
Name: PRICE, dtype: float64
```

Now you will convert the dataset into an optimized data structure called Dmatrix that XGBoost supports and gives it acclaimed performance and efficiency gains. You will use this later in the tutorial.

In [38]:
```
# New cmd
data_dmatrix = xgboost.DMatrix(data=x,label=y)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\xgboost\core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
C:\ProgramData\Anaconda3\lib\site-packages\xgboost\core.py:588: FutureWarning:
Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

Split the data in 30% testing and 70% training

common formula since week-3, week-4.pdf

In [39]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.30, random_
state=1029)
```

Instantiate an XGBoost regressor object by calling the XGBRegressor() class from the XGBoost library with the hyper-parameters passed as arguments. For classification problems, you would have used the XGBClassifier() class.

In [41]:
```
myXGBoost = xgboost.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.
3, learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 10)
```

Fit the regressor to the training set and make predictions on the test set using the familiar .fit() and .predict() methods.

o/p = [17:52:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

In [49]:
```
myXGBoost.fit(x_train, y_train)
```

```
[21:39:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.

C:\ProgramData\Anaconda3\lib\site-packages\xgboost\core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

Out[49]:
```
XGBRegressor(alpha=10, base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.3, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=5, min_child_weight=1, missing=None, n_estimators=10,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

Print the actual prices from test data

In [52]:
```
myXGBoost.missing
```
Out[52]:
```
nan
```
In [54]:
```
myXGBoost.learning_rate
```
Out[54]:
```
0.1
```
In [56]:
```
myXGBoost.base_score
```
Out[56]:
```
0.5
```
In [64]:
```
myXGBoost.kwargs
```
Out[64]:
```
{'alpha': 10}
```

Print the actual prices from test data

In [66]:
```
# Print the actual prices from test data
# x_test data was not used to train the model so this data is pure/untouched dat
a
myXGBoost.predict(x_test)  # so x_test can give us a real time prediction.
```
Out[66]:
```
array([ 9.613199 , 10.932054 , 12.863521 , 14.545456 , 10.435616 ,
        9.868552 , 10.991049 ,  8.537203 , 10.1254425, 11.781072 ,
       22.830309 , 18.831198 , 13.259726 , 15.512948 , 18.93575  ,
       13.779034 , 16.963024 , 22.611881 , 13.0925255, 16.653694 ,
       15.592541 ,  8.304296 , 11.052871 , 14.394213 , 16.638666 ,
       12.728169 , 19.53332  , 14.984259 , 19.996819 , 17.17081  ,
        9.062304 , 14.937686 , 15.187487 , 18.831198 , 19.154423 ,
       12.384975 , 20.658506 ,  9.787031 ,  7.3919277, 15.046192 ,
       15.583125 , 18.435575 , 18.667007 , 16.061813 , 22.656502 ,
       15.475901 , 13.376813 , 10.999661 , 13.416661 , 14.954578 ,
       10.77917  , 10.749382 , 12.106483 , 11.696469 , 18.952229 ,
       11.25369  , 18.57501  , 16.054462 , 16.173088 ,  7.250405 ,
       12.489716 , 16.35532  ,  6.763396 , 17.010836 , 17.575794 ,
        8.790408 , 22.523949 , 15.4468155, 12.060374 , 23.778217 ,
       22.129562 , 18.18319  , 14.627773 , 13.259726 , 15.330975 ,
       13.330317 , 15.801362 , 11.415566 , 14.403312 , 15.025169 ,
       11.979152 , 20.327179 , 13.330317 ,  8.800793 , 23.106798 ,
       19.55888  , 13.156414 , 10.262989 , 21.725063 , 14.770722 ,
       13.801719 , 11.181586 , 19.747766 , 14.004316 , 15.821512 ,
       15.116333 , 13.53992  , 21.220598 , 16.272764 , 16.529219 ,
       16.280914 , 15.985626 , 14.857656 , 11.410328 , 11.9857645,
       13.639041 , 16.140192 , 16.208584 ,  8.552616 , 22.479782 ,
       13.53992  ,  9.438037 ,  7.4606333, 14.188744 , 13.376813 ,
        7.9746485, 10.220309 , 17.753311 , 11.028822 ,  7.6453543,
       11.420262 , 14.095941 , 12.75036  , 16.248932 , 14.59546  ,
       15.56374  , 23.16584  , 14.770722 , 10.563107 , 12.486195 ,
       11.515943 , 15.475901 , 19.567482 ,  9.638401 , 15.883735 ,
        8.889849 , 23.106798 , 18.611135 , 13.009741 , 16.146973 ,
       12.857576 , 15.349565 , 10.678354 , 16.132462 , 19.567755 ,
       14.364777 , 19.118233 , 15.3762245, 23.172472 , 23.16584  ,
       13.363259 , 12.486348 ], dtype=float32)
```

Print the predicted prices

o/p=

array([15. , 26.6, 45.4, 20.8, 34.9, 21.9, 28.7, 7.2, 20. , 32.2, 24.1, 18.5, 13.5, 27. , 23.1, 18.9, 24.5, 43.1, 19.8, 13.8, 15.6, 50. , 37.2, 46. , 50. , 21.2, 14.9, 19.6, 19.4, 18.6, 26.5, 32. , 10.9,

```
20. , 21.4, 31. , 25. , 15.4, 13.1, 37.6, 37. , 18.9, 27.9, 50. ,
14.4, 22. , 19.9, 21.6, 15.6, 15. , 32.4, 29.6, 20.4, 12.3, 19.1,
14.9, 17.8,  8.8, 35.4, 11.5, 19.6, 20.6, 15.6, 19.9, 23.3, 22.3,
24.8, 16.1, 22.8, 30.5, 20.4, 24.4, 16.6, 26.2, 16.4, 20.1, 13.9,
19.4, 22.8, 13.8, 31.6, 10.5, 23.8, 22.4, 19.3, 22.2, 12.6, 19.4,
22.2, 29.8,  9.6, 34.9, 21.4, 25.3, 32.9, 26.6, 14.6, 31.5, 23.3,
33.3, 17.5, 19.1, 48.5, 17.1, 23.1, 28.4, 18.9, 13. , 17.2, 24.1,
18.5, 21.8, 13.3, 23. , 14.1, 23.9, 24. , 17.2, 21.5, 19.1, 20.8,
36. , 20.1,  8.7, 13.6, 22. , 22.2, 21.1, 13.4, 17.4, 20.1, 10.2,
23.1, 10.2, 13.1, 14.3, 14.5,  7.2, 19.6, 20.6, 22.7, 26.4,  7.5,
20.3, 50. ,  8.5, 20.3, 16.1, 22. , 19.6, 10.2, 23.2])
```

In [76]:

```
#print(y_test.values.tolist())
y_test.values
```

Out[76]:

```
array([12.5, 14. , 18.8, 23.7, 10.8,  8.4, 15.6,  9.7, 13.5, 13.1, 50. ,
       23.6, 19. , 22.6, 24.8, 14.4, 22. , 38.7, 14.5, 19.4, 23.3, 16.3,
       14.1, 20.3, 21.9, 19.1, 37. , 29.8, 29. , 24.8, 11.8, 28.7, 19.8,
       33.2, 35.4, 14.8, 33.8, 10.2,  5.6, 21.6, 20.6, 31.5, 31.1, 23.3,
       50. , 25. , 24.5, 17. , 15. , 22.2, 15.4, 19.4, 13.2, 16.8, 32.4,
       15.6, 35.1, 27. , 23.3,  5. , 16.6, 25.2,  8.5, 24.8, 32.5,  8.4,
       50. , 22.9, 15.3, 43.5, 30.1, 28.4, 16.6, 18.5, 23.8, 19.3, 25.3,
       19.1, 27.1, 23.9, 27.5, 32. , 20.4, 13.1, 48.3, 31.5, 16.5, 14.9,
       33.3, 21.7, 21. , 18.4, 33.1, 19. , 23.6, 23.2, 23.1, 34.9, 28.6,
       27.9, 25. , 24.3, 23.4, 14.6, 22.7, 19.8, 36.2, 13.8,  8.7, 50. ,
       11.9, 13.4, 10.2, 23.1, 22. ,  8.8, 15.4, 27.5, 14.1,  7.2, 16.1,
       23. , 19.6, 20.9, 22.1, 24.6, 46. , 20.8, 20. , 25. , 19.5, 28.1,
       33.1,  6.3, 19.4,  7.5, 46.7, 24.1, 21.2, 24.6, 21.7, 18.2, 11.8,
       22. , 34.7, 19.7, 50. , 24.5, 36. , 45.4, 22.7, 20.2])
```

Compare y's

In [79]:

```
y.head(3)
```

Out[79]:

```
0    24.0
1    21.6
2    34.7
Name: PRICE, dtype: float64
```

In [86]:
```
y = df_boston.iloc[0:len(df_boston),-1:len(df_boston)]
y.head(3)
```

Out[86]:

	PRICE
0	24.0
1	21.6
2	34.7

In [88]:
```
y_test.head(3)
```

Out[88]:
```
396    12.5
140    14.0
124    18.8
Name: PRICE, dtype: float64
```

In [103]:
```
x,y, print("Hi")
```

```
Hi
```

Out[103]:
```
(          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  1.0  296.0
 1     0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
 2     0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
 3     0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
 4     0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0
 ..        ...   ...    ...   ...    ...    ...   ...     ...  ...    ...
 501   0.06263   0.0  11.93   0.0  0.573  6.593  69.1  2.4786  1.0  273.0
 502   0.04527   0.0  11.93   0.0  0.573  6.120  76.7  2.2875  1.0  273.0
 503   0.06076   0.0  11.93   0.0  0.573  6.976  91.0  2.1675  1.0  273.0
 504   0.10959   0.0  11.93   0.0  0.573  6.794  89.3  2.3889  1.0  273.0
 505   0.04741   0.0  11.93   0.0  0.573  6.030  80.8  2.5050  1.0  273.0

      PTRATIO       B  LSTAT
 0       15.3  396.90   4.98
 1       17.8  396.90   9.14
 2       17.8  392.83   4.03
 3       18.7  394.63   2.94
 4       18.7  396.90   5.33
 ..       ...     ...    ...
 501     21.0  391.99   9.67
 502     21.0  396.90   9.08
 503     21.0  396.90   5.64
 504     21.0  393.45   6.48
 505     21.0  396.90   7.88

 [506 rows x 13 columns],      PRICE
 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

 [506 rows x 1 columns], None)
```

In [111]:
```
x_train, x_test, y_train, y_test
```

Out[111]:

```
(        CRIM    ZN  INDUS  CHAS    NOX     RM    AGE     DIS   RAD    TAX  \
 470  4.34879   0.0  18.10   0.0  0.580  6.167   84.0  3.0334  24.0  666.0
 115  0.17134   0.0  10.01   0.0  0.547  5.928   88.2  2.4631   6.0  432.0
 111  0.10084   0.0  10.01   0.0  0.547  6.715   81.6  2.6775   6.0  432.0
 114  0.14231   0.0  10.01   0.0  0.547  6.254   84.2  2.2565   6.0  432.0
 471  4.03841   0.0  18.10   0.0  0.532  6.229   90.7  3.0993  24.0  666.0
 ..       ...   ...    ...   ...    ...    ...    ...     ...   ...    ...
 142  3.32105   0.0  19.58   1.0  0.871  5.403  100.0  1.3216   5.0  403.0
 371  9.23230   0.0  18.10   0.0  0.631  6.216  100.0  1.1691  24.0  666.0
 192  0.08664  45.0   3.44   0.0  0.437  7.178   26.3  6.4798   5.0  398.0
 253  0.36894  22.0   5.86   0.0  0.431  8.259    8.4  8.9067   7.0  330.0
 138  0.24980   0.0  21.89   0.0  0.624  5.857   98.2  1.6686   4.0  437.0

      PTRATIO       B  LSTAT
 470     20.2  396.90  16.29
 115     17.8  344.91  15.76
 111     17.8  395.59  10.16
 114     17.8  388.74  10.45
 471     20.2  395.33  12.87
 ..       ...     ...    ...
 142     14.7  396.90  26.82
 371     20.2  366.15   9.53
 192     15.2  390.49   2.87
 253     19.1  396.90   3.54
 138     21.2  392.04  21.32

 [354 rows x 13 columns],
          CRIM    ZN  INDUS  CHAS     NOX     RM    AGE     DIS   RAD    TAX
\
 396   5.87205   0.0  18.10   0.0  0.6930  6.405   96.0  1.6768  24.0  666.0
 140   0.29090   0.0  21.89   0.0  0.6240  6.174   93.6  1.6119   4.0  437.0
 124   0.09849   0.0  25.65   0.0  0.5810  5.879   95.8  2.0063   2.0  188.0
 481   5.70818   0.0  18.10   0.0  0.5320  6.750   74.9  3.3317  24.0  666.0
 444  12.80230   0.0  18.10   0.0  0.7400  5.854   96.6  1.8956  24.0  666.0
 ..        ...   ...    ...   ...     ...    ...    ...     ...   ...    ...
 248   0.16439  22.0   5.86   0.0  0.4310  6.433   49.1  7.8265   7.0  330.0
 258   0.66351  20.0   3.97   0.0  0.6470  7.333  100.0  1.8946   5.0  264.0
 280   0.03578  20.0   3.33   0.0  0.4429  7.820   64.5  4.6947   5.0  216.0
 164   2.24236   0.0  19.58   0.0  0.6050  5.854   91.8  2.4220   5.0  403.0
 463   5.82115   0.0  18.10   0.0  0.7130  6.513   89.9  2.8016  24.0  666.0

      PTRATIO       B  LSTAT
 396     20.2  396.90  19.37
 140     21.2  388.08  24.16
 124     19.1  379.38  17.58
 481     20.2  393.07   7.74
 444     20.2  240.52  23.79
 ..       ...     ...    ...
 248     19.1  374.71   9.52
 258     13.0  383.29   7.79
 280     14.9  387.31   3.76
 164     14.7  395.11  11.64
 463     20.2  393.82  10.29

 [152 rows x 13 columns],
 470    19.9
 115    18.3
 111    22.8
 114    18.5
 471    19.6
        ...
 142    13.4
```

In [112]:
```
myXGBoost.predict(x_test)
```

Out[112]:
```
array([ 9.613199 , 10.932054 , 12.863521 , 14.545456 , 10.435616 ,
        9.868552 , 10.991049 ,  8.537203 , 10.1254425, 11.781072 ,
       22.830309 , 18.831198 , 13.259726 , 15.512948 , 18.93575  ,
       13.779034 , 16.963024 , 22.611881 , 13.0925255, 16.653694 ,
       15.592541 ,  8.304296 , 11.052871 , 14.394213 , 16.638666 ,
       12.728169 , 19.53332  , 14.984259 , 19.996819 , 17.17081  ,
        9.062304 , 14.937686 , 15.187487 , 18.831198 , 19.154423 ,
       12.384975 , 20.658506 ,  9.787031 ,  7.3919277, 15.046192 ,
       15.583125 , 18.435575 , 18.667007 , 16.061813 , 22.656502 ,
       15.475901 , 13.376813 , 10.999661 , 13.416661 , 14.954578 ,
       10.77917  , 10.749382 , 12.106483 , 11.696469 , 18.952229 ,
       11.25369  , 18.57501  , 16.054462 , 16.173088 ,  7.250405 ,
       12.489716 , 16.35532  ,  6.763396 , 17.010836 , 17.575794 ,
        8.790408 , 22.523949 , 15.4468155, 12.060374 , 23.778217 ,
       22.129562 , 18.18319  , 14.627773 , 13.259726 , 15.330975 ,
       13.330317 , 15.801362 , 11.415566 , 14.403312 , 15.025169 ,
       11.979152 , 20.327179 , 13.330317 ,  8.800793 , 23.106798 ,
       19.55888  , 13.156414 , 10.262989 , 21.725063 , 14.770722 ,
       13.801719 , 11.181586 , 19.747766 , 14.004316 , 15.821512 ,
       15.116333 , 13.53992  , 21.220598 , 16.272764 , 16.529219 ,
       16.280914 , 15.985626 , 14.857656 , 11.410328 , 11.9857645,
       13.639041 , 16.140192 , 16.208584 ,  8.552616 , 22.479782 ,
       13.53992  ,  9.438037 ,  7.4606333, 14.188744 , 13.376813 ,
        7.9746485, 10.220309 , 17.753311 , 11.028822 ,  7.6453543,
       11.420262 , 14.095941 , 12.75036  , 16.248932 , 14.59546  ,
       15.56374  , 23.16584  , 14.770722 , 10.563107 , 12.486195 ,
       11.515943 , 15.475901 , 19.567482 ,  9.638401 , 15.883735 ,
        8.889849 , 23.106798 , 18.611135 , 13.009741 , 16.146973 ,
       12.857576 , 15.349565 , 10.678354 , 16.132462 , 19.567755 ,
       14.364777 , 19.118233 , 15.3762245, 23.172472 , 23.16584  ,
       13.363259 , 12.486348 ], dtype=float32)
```

In [152]:
```
print(x.shape)
print(y.shape)
print(x_test.shape)
print(x_train.shape)
print(y_test.shape)
print(y_train.shape)
print(myXGBoost.predict(x_test).shape)
```

```
(506, 13)
(506, 1)
(152, 13)
(354, 13)
(152,)
(354,)
(152,)
```

Compute the RMSE by invoking the mean_sqared_error function from sklearn's metrics module.

o/p = RMSE: 9.715257

In [127]:
```
mse=numpy.sqrt(mean_squared_error(y_test, myXGBoost.predict(x_test)))
mse
```

Out[127]:
```
10.40452018557639
```

In [128]:
```
import math
```

In [165]:
```
rmse= "%f" % mse
rmse
```

Out[165]:
```
'10.404520'
```

To be continue...

https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155 (https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155)