Import Library

```
[ ] import pandas as pd
```

```
import numpy as np
```

Import CSV as Dataframe

```
[ ] df=pd.read_csv(r"https://github.com/YBI-Foundation/Dataset/raw/main/Boston.csv")
```

Get the first 5 rows of dataframe

```
[ ] df.head()
```

	CRIM	ZN	INDUS	CHAS	NX	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

Get Information of Dataframe

```
[ ] 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   NX       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
```

Get the summary statistics

```
df.describe()
```

	CRIM	ZN	INDUS	CHAS	NX	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

Get Column names

```
[ ] df.columns
```

```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'MEDV'],
      dtype='object')
```

Get Shape of Dataframe

```
df.shape
```

```
(506, 14)
```

Define y(dependant/label/target variable) and X(independant/features/attribute variable)

```
[ ] y=df['MEDV']
```

```
[ ] y.shape
```

```
(506,)
```

```
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: 506, dtype: float64
```

```
[ ] X=[['CRIM','ZN','INDUS','CHAS','NX','RM','AGE','DIS','RAD','TAX','PTRATIO','B','LSTAT']]
```

Or used drop function to define X

```
[ ] X=df.drop('MEDV',axis=1)
```

```
X.shape
```

```
(506, 13)
```

```
[ ] X
```

	CRIM	ZN	INDUS	CHAS	NX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
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
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
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
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
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
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273.0	21.0	396.90	7.88

506 rows × 13 columns

Get X varibles standardized

```
[ ] from sklearn.preprocessing import MinMaxScaler
```

```
[ ] mm=MinMaxScaler()
```

```
[ ] X=mm.fit_transform(X)
```

```
X
```

```
array([[0.00000000e+00, 1.80000000e-01, 6.78152493e-02, ...,
        2.87234043e-01, 1.00000000e+00, 8.96799117e-02],
       [2.35922539e-04, 0.00000000e+00, 2.42302053e-01, ...,
        5.53191489e-01, 1.00000000e+00, 2.04470199e-01],
       [2.35697744e-04, 0.00000000e+00, 2.42302053e-01, ...,
        5.53191489e-01, 9.89737254e-01, 6.34657837e-02],
       ...,
       [6.11892474e-04, 0.00000000e+00, 4.20454545e-01, ...,
        8.93617021e-01, 1.00000000e+00, 1.07891832e-01],
       [1.16072990e-03, 0.00000000e+00, 4.20454545e-01, ...,
        8.93617021e-01, 9.91300620e-01, 1.31070640e-01],
       [4.61841693e-04, 0.00000000e+00, 4.20454545e-01, ...,
        8.93617021e-01, 1.00000000e+00, 1.69701987e-01]])
```

Get Train Test Split

```
[ ] from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=2529)
```

```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

```
((354, 13), (152, 13), (354,), (152,))
```

Get Model Train

```
[ ] from sklearn.linear_model import LinearRegression
```

```
[ ] lr=LinearRegression()
```

```
[ ] lr.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Get Model Prediction

```
y_pred=lr.predict(X_test)
```

```
[ ] y_pred.shape
```

```
(152,)
```

```
[ ] y_pred
```

```
[ ] array([31.71733828, 22.02143302, 21.16613197, 39.77837246, 20.10258512,
       22.86056216, 18.35574643, 14.7902735 , 22.55778646, 21.34594953,
       18.38491085, 27.9664665 , 29.85929012,  6.44680773, 10.68297311,
       26.24809521, 21.89368671, 25.22692365,  3.62385942, 36.21920372,
       24.07812335, 22.94103934, 14.27095261, 20.79013279, 24.22725035,
       16.7379611 , 18.74856986, 20.96709658, 28.513571  , 20.86346628,
        9.23450577, 17.06754852, 22.06953886, 22.23121875, 39.25875323,
       26.16769924, 42.50354003, 19.34517962, 34.51869058, 14.07023676,
       13.81055358, 23.27727535, 11.79100403,  9.01040731, 21.64587594,
       25.55339317, 18.16941728, 16.81991401, 14.66170215, 14.86477172,
       33.78924259, 33.26959074, 15.49208778, 24.08269034, 27.63531226,
       19.58288727, 45.02488529, 20.96959671, 20.07202649, 27.67146866,
       34.59154418, 12.71353064, 23.66247812, 31.65792337, 28.97459925,
       32.45963484, 13.93494747, 35.491924  , 19.35871482, 19.60341885,
        1.43927038, 24.10206738, 33.67200257, 20.62160583, 26.89383792,
       21.28629335, 31.94640391, 29.73908623, 13.93454775, 13.81678383,
       19.75873615, 21.54069878, 20.86933991, 23.62698265, 28.79508068,
       23.64118169,  6.95157816, 22.19831966, -6.82270042, 16.96842453,
       16.76859897, 25.43664303, 14.95151023,  3.71667789, 15.02525824,
       16.90607726, 21.45897878, 31.65915538, 30.72068155, 23.72584448,
       22.18882729, 13.76042247, 18.47384318, 18.1524094 , 36.60119404,
       27.49121167, 11.00093835, 17.26407285, 22.49004463, 16.52993633,
       29.49279312, 22.89418353, 24.67840473, 20.37710587, 19.68603018,
       22.55437435, 27.31673957, 24.86003524, 20.2018396 , 29.14358757,
        7.42840113,  5.85287912, 25.34843348, 38.73123659, 23.94325177,
       25.28198173, 20.11046586, 19.75220882, 25.06978342, 35.15909482,
       27.31951047, 27.2616268 , 31.39965843, 16.55315203, 14.29555368,
       23.76937723,  7.64840244, 23.34914332, 21.36612339, 26.12068678,
```

```
[ ]       25.31847859, 13.1171793 , 17.66685837, 36.19968161, 20.50074493,
          27.94813333, 22.45926502, 18.14585016, 31.24201417, 20.85014715,
          27.35824971, 30.53239318])
```

Get Model Evaluation

```
[ ] from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
```

```
[ ] mean_squared_error(y_test,y_pred)
```

```
20.71801287783855
```

```
mean_absolute_error(y_test,y_pred)
```

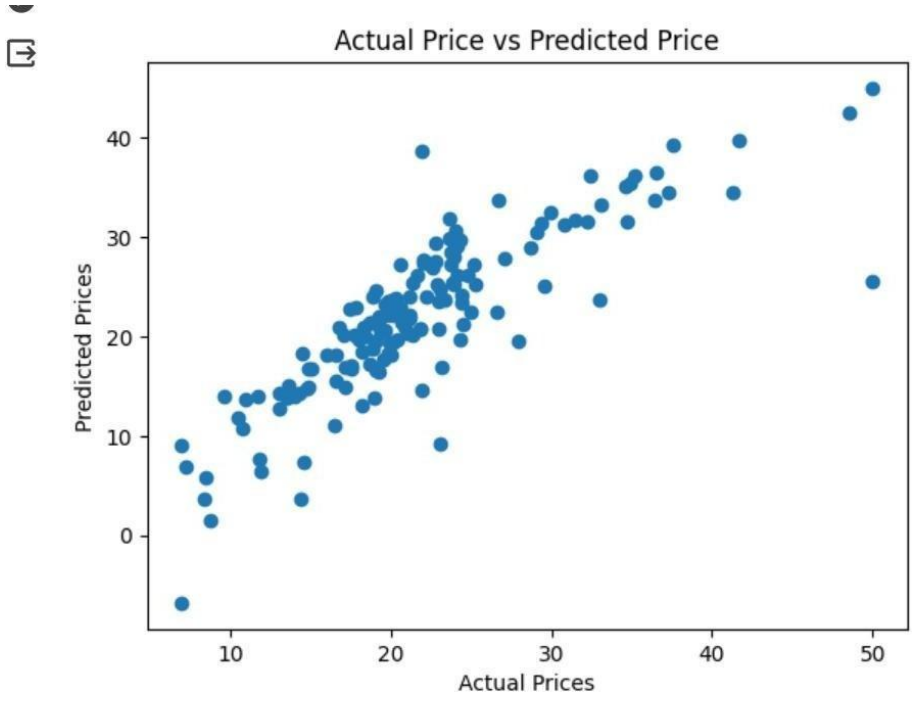
```
3.1550309276024926
```

```
[ ] r2_score(y_test,y_pred)
```

```
0.6551914852365517
```

Get Visualization of Actual vs Predicted Results

```
import matplotlib.pyplot as plt
plt.scatter(y_test,y_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Price vs Predicted Price")
plt.show()
```



Get Future Predictions

Steps to follow : 1.Extract a random row using sample function 2.Separate X and y 3.Standardize X 4.Predict

```
[ ] X_new=df.sample(1)
```

```
[ ] X_new
```

	CRIM	ZN	INDUS	CHAS	NX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
389	8.15174	0.0	18.1	0	0.7	5.39	98.9	1.7281	24	666.0	20.2	396.9	20.85	11.5

```
[ ] X_new.shape
```

```
(1, 14)
```

```
[ ] X_new=X_new.drop('MEDV',axis=1)
```

```
X_new
```

	CRIM	ZN	INDUS	CHAS	NX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
389	8.15174	0.0	18.1	0	0.7	5.39	98.9	1.7281	24	666.0	20.2	396.9	20.85

```
X_new.shape
```

```
(1, 13)
```

```
[ ] X_new=mm.fit_transform(X_new)
```

```
[ ] y_pred_new=lr.predict(X_new)
```

```
[ ] y_pred_new
```

```
array([25.6750862])
```