House Price Prediction

Boston House Prices

https://www.kaggle.com/datasets/vikrishnan/boston-house-prices

Each record in the database describes a Boston suburb or town. The data was drawn from the Boston Standard Metropolitan Statistical Area (SMSA) in 1970. The attributes are defined as follows (taken from the UCI Machine Learning Repository1): CRIM: per capita crime rate by town

- 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 USD
- 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

Data Preprocessing

In [133…

```
# Importing the important libraries for data preprocessing and data visualization
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import missingno as msno
import sklearn.datasets
import warnings
warnings.filterwarnings("ignore")
```

In [134…

```
# Importing the Boston house price dataset
df = pd.read_csv("boston.csv")
df.head()
```

Out[134]:

	CRIM	ZN	INDUS	CHAS	NOX	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

In [135…

```
# Rename the target column
df.rename(columns={'MEDV':'Price'},inplace=True)
df.head()
```

Out[135]:

	CRIM	ZN	INDUS	CHAS	NOX	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

In [136…

```
# shape of df
df.shape
```

Out[136]:

```
(506, 14)
```

In [137…

```
# Size of df
df.size
```

Out[137]:

```
7084
```

In [138…

```
# Checking Null value
df.isnull().sum()
```

Out[138]:
```
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
Price      0
dtype: int64
```

In [139…
```
# Checking no of duplicate in each column
df[df.duplicated()].sum()
```

Out[139]:
```
CRIM       0.0
ZN         0.0
INDUS      0.0
CHAS       0.0
NOX        0.0
RM         0.0
AGE        0.0
DIS        0.0
RAD        0.0
TAX        0.0
PTRATIO    0.0
B          0.0
LSTAT      0.0
Price      0.0
dtype: float64
```

In [140…
```
# Info about dataset
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   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  Price    506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

In [141…
```
# Summary of dataset
df.describe().T
```

Out[141]:

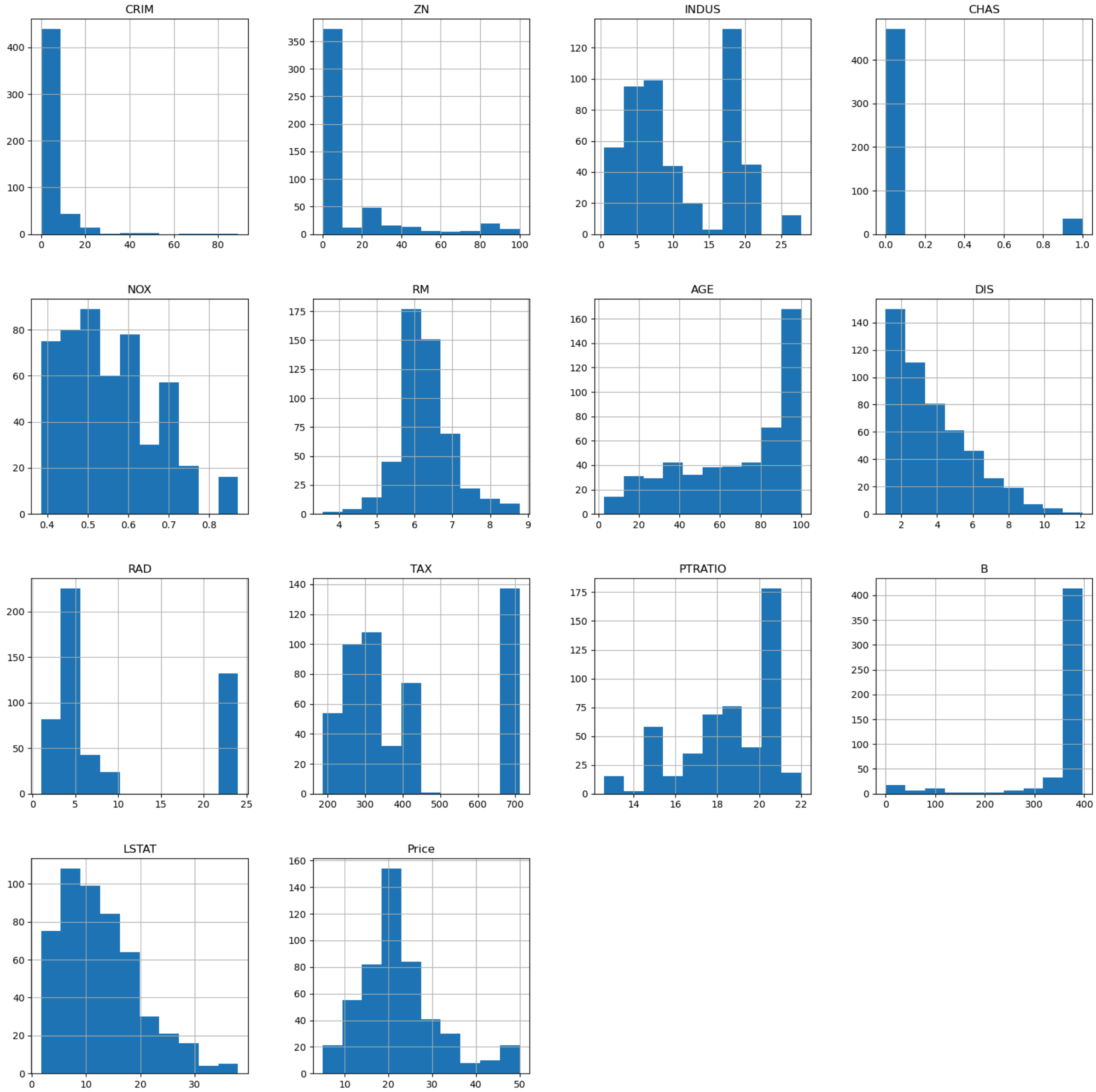
	count	mean	std	min	25%	50%	75%	max
CRIM	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.677083	88.9762
ZN	506.0	11.363636	23.322453	0.00000	0.000000	0.00000	12.500000	100.0000
INDUS	506.0	11.136779	6.860353	0.46000	5.190000	9.69000	18.100000	27.7400
CHAS	506.0	0.069170	0.253994	0.00000	0.000000	0.00000	0.000000	1.0000
NOX	506.0	0.554695	0.115878	0.38500	0.449000	0.53800	0.624000	0.8710
RM	506.0	6.284634	0.702617	3.56100	5.885500	6.20850	6.623500	8.7800
AGE	506.0	68.574901	28.148861	2.90000	45.025000	77.50000	94.075000	100.0000
DIS	506.0	3.795043	2.105710	1.12960	2.100175	3.20745	5.188425	12.1265
RAD	506.0	9.549407	8.707259	1.00000	4.000000	5.00000	24.000000	24.0000
TAX	506.0	408.237154	168.537116	187.00000	279.000000	330.00000	666.000000	711.0000
PTRATIO	506.0	18.455534	2.164946	12.60000	17.400000	19.05000	20.200000	22.0000
B	506.0	356.674032	91.294864	0.32000	375.377500	391.44000	396.225000	396.9000
LSTAT	506.0	12.653063	7.141062	1.73000	6.950000	11.36000	16.955000	37.9700
Price	506.0	22.532806	9.197104	5.00000	17.025000	21.20000	25.000000	50.0000

In [142…

```
correlation = df.corr()
```

In [143…

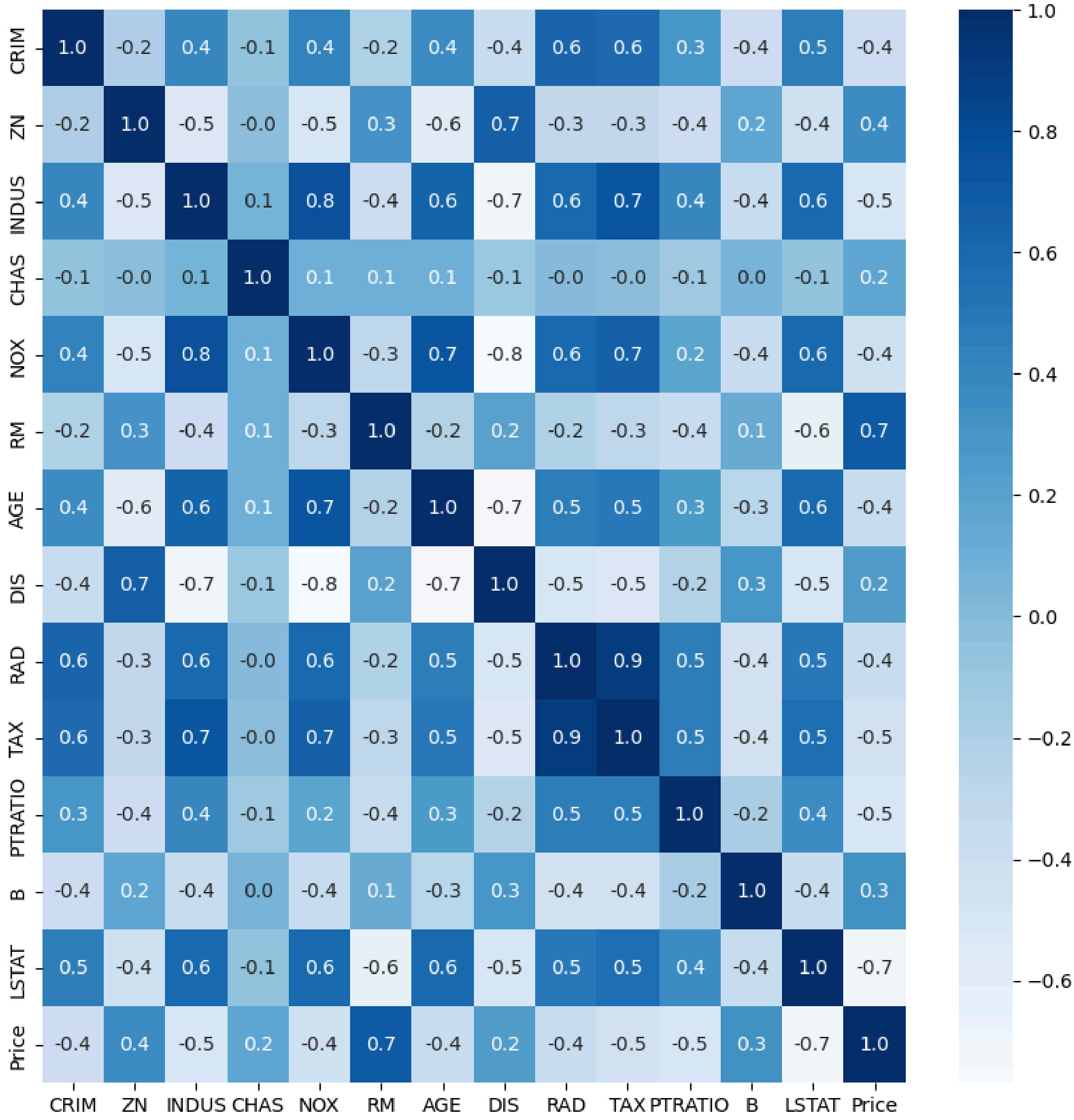
```
# Understanding ecah numerical column through histogram
p=df.hist(figsize = (20,20))
```



In [144…

```python
# Heatmap
plt.figure(figsize=(10,10))
sns.heatmap(correlation,annot=True,cmap='Blues',fmt='.1f')
```

Out[144]: <Axes: >



In [145...

```
# Creating the dependent and independent variables from dataset df
X = df.drop(columns='Price',axis=1) # independent variable column
Y = df['Price'] # dependent variable column
```

In [146...

```
print(X)
```

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

In [147…

```python
print(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: Price, Length: 506, dtype: float64
```

Train Test Split

In [148…

```python
# Import Train Test Split
from sklearn.model_selection import train_test_split
```

In [149…

```python
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2,random_state=
```

In [150…

```python
print(X.shape, X_train.shape, X_test.shape)
```

```
(506, 13) (404, 13) (102, 13)
```

Model Training

Linear Regression

In [151…
```
# Import library for Linear Regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

In [152…
```
# Initialization of Linear Regression model
lr_model = LinearRegression()
lr_model.fit(X_train, Y_train)
```

Out[152]:
```
▾ LinearRegression
LinearRegression()
```

Predictions on train set

In [153…
```
# Making predictions on the train set
Y_train_pred_lr = lr_model.predict(X_train)
```

In [154…
```
print(Y_train_pred_lr)
```

```
[10.96952405 19.41196567 23.06419602 12.1470648  18.3738116  25.24677946
 20.77024774 23.90932632  7.81713319 19.60988098 21.8202963  27.59615864
 32.67986504 15.12308446 35.3964561  12.99688651 20.728181   28.30223542
 15.61724836 24.45143096  4.61794591 23.76681932 25.56178249 22.98928526
 24.5213025  34.06407919 19.71166707 39.11233072 14.62515846 24.81139885
 18.02332883 20.85836445  9.57577261 20.87246835 22.28583096 31.79327155
 31.04748307 15.70611763 17.01382935 28.23332703 24.27661276 16.88670215
  6.90720745 26.75808901 22.586493   17.53664716 13.77197016 41.04840929
 16.44690754 18.23531669 25.37038646 23.64581399 22.05322581 20.83620499
 16.93508273 22.797579   29.13333934  7.69310515 24.60571452 17.2358028
 21.10846551 25.15150324 27.33394823 21.30494963 41.5811902  19.19666651
 15.37955448 19.33545877 17.04687638 22.96801532 23.11094953 33.6977586
 22.77436405 20.28968381 25.35517813 31.02479125 33.05103792 28.44712333
  8.50926331  5.61220643 12.81228164 19.81854491 34.8603548  33.47481463
 15.81288676  4.16863764 32.81131556 21.22307142 18.97752706 26.36174269
 18.38053781 17.80316891 11.8730344  31.84801205 24.45344478 20.0222241
 19.5225374  11.8723419  28.90289906 19.7133604  32.47093634 33.20696505
 24.79405395 21.25197228 25.03045081 43.36995367 29.54151469 33.75302939
 26.27516427 27.04791799 15.1908027  31.34177077 20.85218327 31.05070715
 28.74449991 21.16535503 23.06717742 12.48881717 36.48751917 37.24291141
 33.23617345  5.30863493 20.82773333 22.16769067 35.62579793 17.10890633
 22.76667    19.50567599 26.36980524 16.03780391 20.63186041 27.04508116
 31.39596353 31.12597743 22.78355908 39.11521781 28.28378993 30.53090392
 28.89518723 21.08389783 29.04801464 16.151087   22.08243372 24.61505524
 18.95878457  2.06366066 20.51120467 26.85927261 23.0775764  18.40141847
 22.70378324 15.95162657 31.54559763 27.82356386 34.19210038 20.70876151
 15.1538496  19.55740929  8.31853813 13.62525632 26.48611752 16.52769982
  4.13593772 24.73356662 12.21856959 28.24704463 33.60549853 36.84177072
 24.28136146 20.7199677  30.79885576 36.93823489 19.92152434 19.59433404
 28.76197718 13.28347615 13.41116334 10.89910148 19.07573086 22.65911351
 30.27080271 29.77482371 17.89252221 29.83838757 14.41987694 13.24056207
 33.87859379 19.6406762  14.54072369 23.66498192 22.41851151 19.63248447
 24.660086   35.00833944  4.12171868 20.79593953 33.240402   18.04778742
 22.6208596  22.39592759 21.4012853  11.74782838 38.18786922 25.76751814
 24.80115706 16.36524419 31.98045427 36.35420843 39.11016375 20.33864814
 22.16464728 16.34276966 39.35137104  6.74954317 21.35789894 15.53370378
 26.91527204  8.80382559 20.93301971 -0.20588155 17.21501492 16.17150935
  8.48374754 23.08202496 17.23085262 29.42673011 44.36436789 28.09470335
 23.75911874 36.75959172  8.71594791 25.90885934 20.83832124 23.68119542
 19.85753143 22.2992237  14.72942336 18.4266911  22.45346464 23.75648384
 28.82492146 23.58310886 27.30142666 18.06532614 13.16799771 31.66795775
 28.77291955 12.60179253 17.38668341 24.05174693 40.7163898  23.0214736
 12.83596226 28.55024128 36.850343   23.26391794 25.14113573 38.36624745
 18.21318365 25.69614391 15.22833068 24.14345412 36.27095489 31.03140871
 24.82017075 17.7834844  17.99307546  8.61827851 41.51965821 19.63259696
 30.16241039 19.69581658 14.59963591 20.29467675 24.22053288 34.79282666
 26.51552807 41.665796   12.32829593 14.32092274 23.69090327 18.01762114
 19.72399411 29.13009315 11.10216617 24.11213143 18.54668994 23.69765843
 30.11853879 19.34756033 12.52355093 33.74737806 16.90295464 17.83165159
 19.34064029 26.74379491 35.24575625 12.92253178 26.69073573 19.19640769
 30.29549933 17.83878909 22.92058129 29.13254708 20.02093559 25.18893157
 20.95650827 20.57624549 32.93168351 20.43565472 25.41798459 28.14952635
 37.59066882 25.0548289  28.61534646 17.97695227 30.78085325 23.57491321
 34.56676284 18.52592551 23.86972656 13.82862001 25.18152388 17.67219765
 12.63704156 17.03927502 14.2510159  28.56862619 22.99037971 13.42262376
 17.40965124 34.44268371 12.93938984 14.62427657 27.50209814 21.28772373
 21.8475453  27.75030943 16.84106111 35.70395065 23.19717187 19.70894368
 20.39856551 31.03155183  5.16165827 36.26386827 38.27409562 21.44507004
 21.53203698 13.25546637 35.43733953 19.75468373 21.59325014 27.28654912
 14.70336355 20.10948908 20.98738625 20.42268561 26.20840113 11.28815662
 34.57059129 22.65270668 22.68814063 33.20155069 26.77878535 21.55230365
  8.80963918 28.40878163 25.10012639 25.47646583 17.70215249 25.63601841
 18.61140436 32.77937269 35.77461311 18.3180684  30.14080347  7.72488159
 26.25987699 10.52826879 27.30604251 44.10078731 28.92351314 14.7836951
```

```
 20.79445301 17.96782515 19.333174   33.02714571 25.71055958 25.89232968
 17.07165041 21.95432205 11.3511532  13.27742402 22.66485295 22.52252947
 12.30424735 32.08396429 22.11175771 17.24071878 22.00480027 26.7237425
 12.97674212 19.14279551]
```

In [155…

```
# Evaluating the model
mse = mean_squared_error(Y_train, Y_train_pred_lr)
rmse = mean_squared_error(Y_train, Y_train_pred_lr, squared=False)  # taking square
r2 = r2_score(Y_train, Y_train_pred_lr)

print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared (R2):", r2)
```

```
Mean Squared Error (MSE): 21.641412753226312
Root Mean Squared Error (RMSE): 4.6520331848801675
R-squared (R2): 0.7508856358979673
```

Visualizing the actual Prices and predicted prices (Train Set)

In [156…

```
plt.scatter(Y_train, Y_train_pred_lr)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Price vs Preicted Price (Train Set)")
plt.show()
```

Predictions on test set

In [157…

```
# Making predictions on the test set
Y_test_pred_lr = lr_model.predict(X_test)
```

In [158…

```
print(Y_test_pred_lr)
```

```
[28.99672362 36.02556534 14.81694405 25.03197915 18.76987992 23.25442929
 17.66253818 14.34119    23.01320703 20.63245597 24.90850512 18.63883645
 -6.08842184 21.75834668 19.23922576 26.19319733 20.64773313  5.79472718
 40.50033966 17.61289074 27.24909479 30.06625441 11.34179277 24.16077616
 17.86058499 15.83609765 22.78148106 14.57704449 22.43626052 19.19631835
 22.43383455 25.21979081 25.93909562 17.70162434 16.76911711 16.95125411
 31.23340153 20.13246729 23.76579011 24.6322925  13.94204955 32.25576301
 42.67251161 17.32745046 27.27618614 16.99310991 14.07009109 25.90341861
 20.29485982 29.95339638 21.28860173 34.34451856 16.04739105 26.22562412
 39.53939798 22.57950697 18.84531367 32.72531661 25.0673037  12.88628956
 22.68221908 30.48287757 31.52626806 15.90148607 20.22094826 16.71089812
 20.52384893 25.96356264 30.61607978 11.59783023 20.51232627 27.48111878
 11.01962332 15.68096344 23.79316251  6.19929359 21.6039073  41.41377225
 18.76548695  8.87931901 20.83076916 13.25620627 20.73963699  9.36482222
 23.22444271 31.9155003  19.10228271 25.51579303 29.04256769 20.14358566
 25.5859787   5.70159447 20.09474756 14.95069156 12.50395648 20.72635294
 24.73957161 -0.164237   13.68486682 16.18359697 22.27621999 24.47902364]
```

In [159…

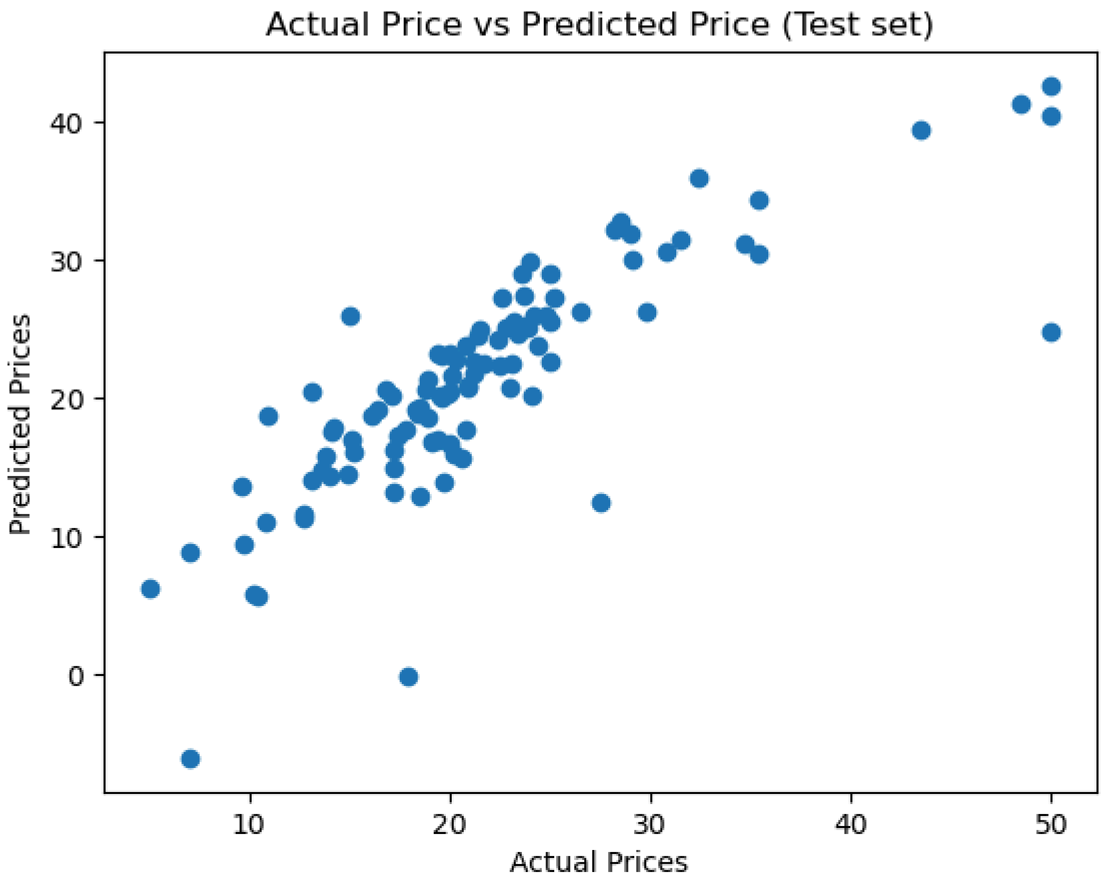
```
# Evaluating the model
mse = mean_squared_error(Y_test, Y_test_pred_lr)
rmse = mean_squared_error(Y_test, Y_test_pred_lr, squared=False)  # taking square r
r2 = r2_score(Y_test, Y_test_pred_lr)

print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared (R2):", r2)
```

```
Mean Squared Error (MSE): 24.29111947497323
Root Mean Squared Error (RMSE): 4.9286021826653075
R-squared (R2): 0.668759493535636
```

Visualizing the actual Prices and predicted prices (Test Set)

In [160…

```
plt.scatter(Y_test, Y_test_pred_lr)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Price vs Predicted Price (Test set)")
plt.show()
```



XGBoost Regressor

In [161…

```
# Import library for XGBoost Regressor
from xgboost import XGBRegressor
```

In [162…

```
# Initialization of XGBoost Regressor
xgb_model = XGBRegressor()
xgb_model.fit(X_train,Y_train)
```

Out[162]:

```
XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=
1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwi
se',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=
4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_wei
ght=1,
```

Predictions on train set

In [163…

```
# Making predictions on the train set
Y_train_pred_xgb = xgb_model.predict(X_train)
```

In [164…

```
print(Y_train_pred_xgb)
```

```
[12.006792  19.914322  19.41386   13.41272   18.233686  24.599525
 21.084385  24.691845   8.696693  27.479736  20.694065  36.1663
 31.603483  11.695366  39.791046  13.88976   21.810648  23.713993
 17.597576  24.410297   8.793519  19.173897  25.288088  20.433308
 23.098907  37.891342  15.598552  45.398563  15.698146  22.600924
 14.546442  18.711456  17.798622  16.106794  20.609913  31.608261
 29.09079   15.603799  17.517525  22.501944  19.392653  19.290897
  8.49561   20.603264  16.995317  17.08292   14.509991  49.995197
 14.281856  12.59869   28.704367  21.207237  19.308727  23.09419
 19.099642  25.001644  33.402706   5.00865   29.600845  18.669462
 21.661974  23.051264  22.805435  20.95566   48.781216  14.632303
 16.598646  27.074171  20.081707  19.794664  21.010654  41.29935
 23.153635  20.358414  18.55529   29.412376  36.40918   24.369547
 11.815963  13.803553  12.26066   17.789642  33.0957    26.747456
 13.385397  14.390727  50.0101    21.988642  19.892096  23.77793
 17.521416  12.686036   5.604692  31.102188  26.214396  19.394894
 16.700687  13.799928  22.910137  15.304866  27.509277  36.0918
 22.874144  24.472837  24.960222  49.997704  34.87582   31.73247
 24.098587  22.109516  14.123575  42.796196  19.325815  32.187332
 26.372147  21.805946  21.689602   8.296155  46.700947  43.148296
 31.480734  10.482078  16.70251   19.989946  33.329723  17.792757
 49.994827  20.512388  23.181997  13.081055  19.617819  22.799692
 28.703516  30.71666   22.890793  21.898142  23.892689  32.70598
 24.303967  21.490267  24.583395   8.526644  26.394464  23.027435
 15.012845   8.805406  19.373169  23.928858  24.659529  19.79578
 23.780231  13.290449  29.002087  27.0892    34.586098  13.292319
 15.608231  12.528827  14.5950165 10.973822  24.783852  17.302105
  8.100943  21.405794  15.597286  23.329689  32.013943  38.696953
 30.09569   20.507383  32.489983  42.29373   24.289165  20.607313
 22.060888  18.201572  15.001471   6.3028407 20.103695  21.386175
 28.40405   30.01001   20.80584   23.01445   14.367735  11.696215
 37.304226  17.098387  10.399942  22.971632  22.716787  20.313557
 21.676828  49.99972    8.39927   18.822573  37.21775   16.096394
 16.50256   22.224373  20.60043   13.515775  48.29156   23.810686
 22.696497  17.40862   30.293821  35.981915  41.707233  18.307549
 21.997921  18.603247  44.799805  11.915975  18.711937  16.186481
 22.017584   7.2080135 20.400402  13.782337  13.003643  18.360273
 23.101606  21.200968  23.092686  23.505844  50.00068   26.570215
 22.177225  50.0089     8.298076  23.304117  21.712616  18.972599
 18.40083   17.432724  13.408636  12.061736  26.588373  21.687403
 28.407064  20.50816   22.009855  13.898941  11.331526  29.88582
 26.62105   10.494745  23.161224  24.385551  45.999058  21.906164
  7.507369  36.170033  43.988125  17.784395  27.487415  37.59683
 14.112935  28.094582  10.222398  19.136097  43.805214  27.895914
 25.035696  15.999449  16.605703  13.216297  50.012806  22.201168
 32.906933  15.213805  14.808653  13.833087  24.293734  33.79947
 22.303656  49.992897   9.505929  13.308888  22.207737  18.094526
 18.004805  25.022642  16.50053   23.016598  20.088505  32.997852
 24.81184   18.25233   13.110781  34.90601   10.20654   19.900312
 27.893597  23.304026  35.09893   12.779468  22.008572  18.491777
 25.139647  22.485453  22.392208  28.596422  19.53014   24.801777
 24.448523  21.41991   33.10218   22.887705  20.701757  24.094767
 50.00511   24.703756  28.674664   7.221856  36.9461    20.306702
 30.109777  19.493416  23.362553  11.491878  21.593145  14.906346
 15.193777  19.403543   8.401711  27.974258  22.61201   13.498264
 14.487641  30.976194  10.908737  21.885714  22.020124  18.999578
 21.385395  25.019173  17.502197  36.495224  20.102486  20.348663
 16.19895   23.60941    7.422648  35.210255  50.010445  19.294695
 21.224382  15.598267  33.419327  19.119143  21.029436  23.70131
 18.899162  16.807821  19.708471  17.73315   22.58946   11.790132
 34.959263  20.551994  20.190685  31.952879  22.32405   23.304356
 14.3989525 31.187862  23.981054  29.603672  19.550035  21.602537
 19.9373    26.990494  33.17728   15.41417   30.476479   7.2053533
 23.898226  16.296663  23.910059  49.994213  22.825403  15.397418
```

```
 19.206684  19.593248  22.593292  33.17042   49.991238  22.252522
 14.896758  19.808842  23.698444  18.973715  20.320982  11.928814
 13.597039  29.822527  21.718994  19.479317  21.09221   24.528873
 13.398071  18.600246 ]
```

In [165…

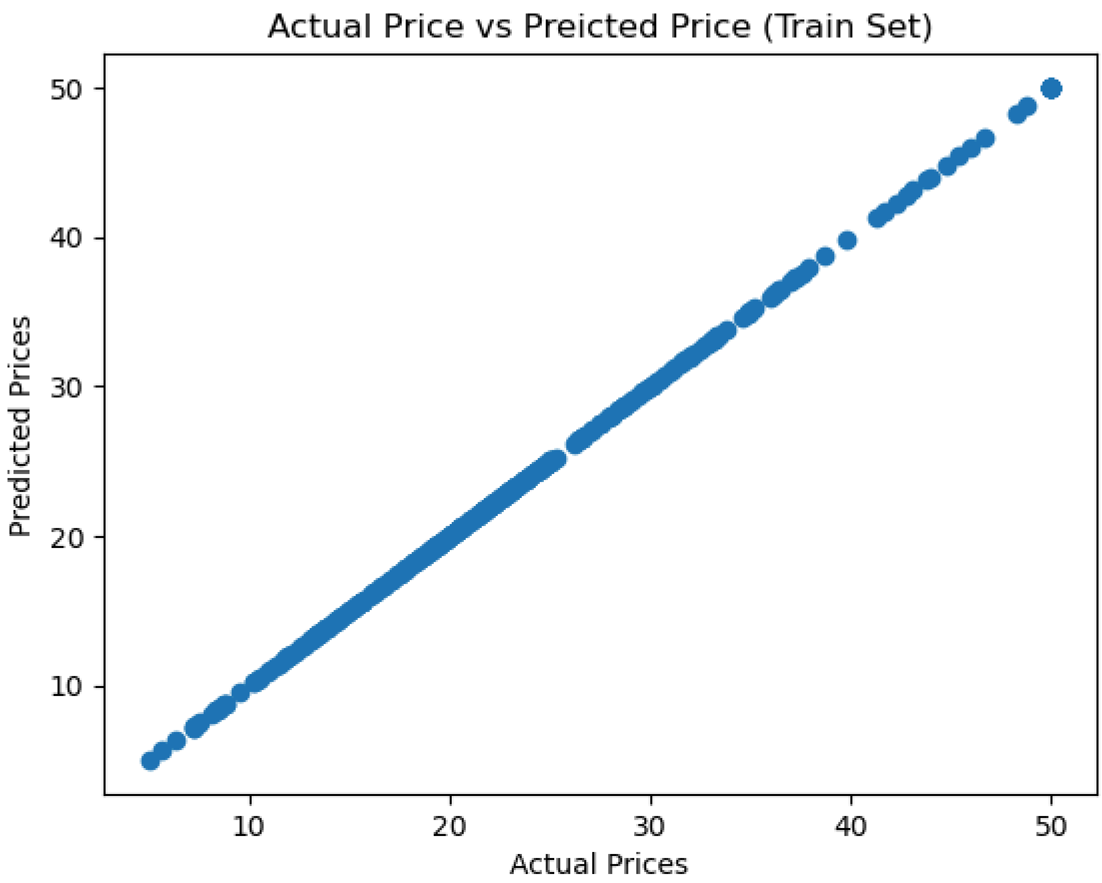
```
# Evaluating the model
mse = mean_squared_error(Y_train, Y_train_pred_xgb)
rmse = mean_squared_error(Y_train, Y_train_pred_xgb, squared=False)  # taking squar
r2 = r2_score(Y_train, Y_train_pred_xgb)

print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared (R2):", r2)
```

```
Mean Squared Error (MSE): 0.00040290003939232\51
Root Mean Squared Error (RMSE): 0.02007237004920757
R-squared (R2): 0.9999953622164942
```

Visualizing the actual Prices and predicted prices (Train Set)

In [166…

```
plt.scatter(Y_train, Y_train_pred_xgb)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Price vs Preicted Price (Train Set)")
plt.show()
```



Predictions on test set

In [167…

```
# Making predictions on the test set
Y_test_pred_xgb = xgb_model.predict(X_test)
```

In [168…

```
print(Y_test_pred_xgb)
```

```
[23.25328   30.024755  15.632249  23.313478  17.775118  21.142563
 20.19583   15.010124  21.23614   22.242369  20.457346  19.209145
  8.551788  21.210636  20.696491  26.74365   18.824339  10.525872
 45.68885   14.116162  26.618996  24.94542   13.3510275 20.87231
 15.400073  15.636547  22.324673  12.777009  20.726126  22.56401
 20.346395  22.303246  18.523277  21.764612  15.568828  15.683646
 33.073547  19.115112  21.955132  22.399914  18.998787  31.328337
 43.464993  18.20766   22.09233   14.353467  14.607512  22.716745
 19.700527  27.072327  22.579268  35.133675  16.241447  25.214682
 46.013332  21.89786   15.043295  32.93268   20.53731   16.568089
 24.07178   34.34796   28.542194  16.977676  25.867334  15.649837
 13.039615  23.00082   27.26897   15.414835  21.546648  31.72919
 10.665012  20.770847  21.848396   6.475782  20.939093  46.59454
 12.456056   8.739085  22.215406  13.390212  20.454681  10.45914
 19.722834  27.327946  16.254663  23.860172  25.414312  17.06042
 22.9362     8.106883  19.001764  18.869307  24.129864  19.66075
 40.517284  13.981451  11.416717  15.428753  19.41982   24.281776 ]
```

In [169…

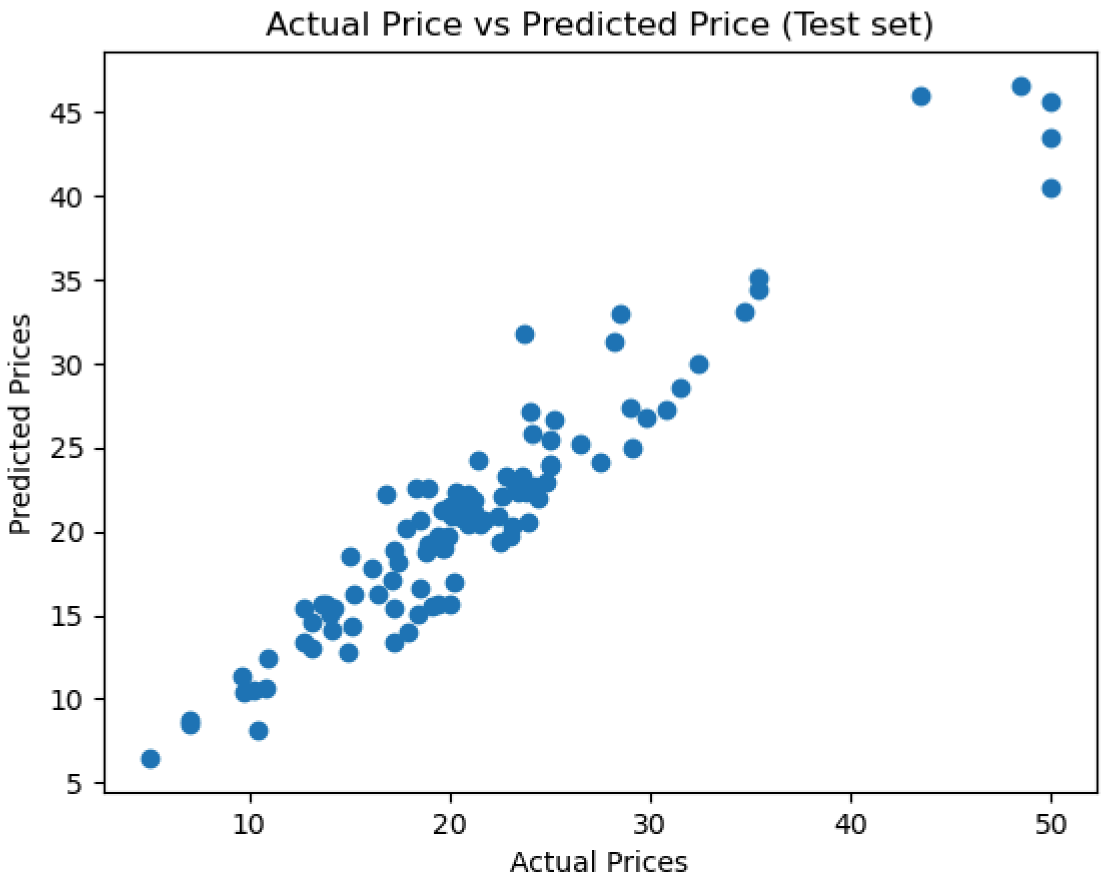
```
# Evaluating the model
mse = mean_squared_error(Y_test, Y_test_pred_xgb)
rmse = mean_squared_error(Y_test, Y_test_pred_xgb, squared=False)  # taking square
r2 = r2_score(Y_test, Y_test_pred_xgb)

print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared (R2):", r2)
```

```
Mean Squared Error (MSE): 6.560527271813469
Root Mean Squared Error (RMSE): 2.561352625433185
R-squared (R2): 0.9105388132305845
```

Visualizing the actual Prices and predicted prices (Test Set)

In [170…

```
plt.scatter(Y_test, Y_test_pred_xgb)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Price vs Predicted Price (Test set)")
plt.show()
```


Actual Price vs Predicted Price (Test set)
Predicted Prices
Actual Prices
45
40
35
30
25
20
15
10
5
10
20
30
40
50

```
In [ ]:
```