Project_House_Price_Prediction

August 16, 2025

1 Importing the Dependencies

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.datasets
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn import metrics
```

2 Importing the Boston House Price Dataset

[ ]:
```
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)

# Loading the dataset to as Pandas Dataframe
house_price_dataset = pd.DataFrame(
    np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
)

# Add the taget (PRICE) column to the DataFrame
house_price_dataset["PRICE"] = raw_df.values[1::2, 2]

house_price_dataset.columns = [
    "CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM",
    "AGE", "DIS", "RAD", "TAX", "PTRATIO", "B", "LSTAT", "PRICE"
]
```

[ ]:
```
print(house_price_dataset)
```

```
      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  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]
```

[ ]:
```
house_price_dataset.head()
```

[ ]:
```
      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

   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
```

[ ]:
```
# Checking the number of rows and columns in the dataframe
house_price_dataset.shape
```

[ ]:
```
(506, 14)
```

[ ]:
```
# Check for the missing values
house_price_dataset.isnull().sum()
```

[ ]:
```
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
```

[ ]:
```
# Statistical measures of the dataset
house_price_dataset.describe()
```

[ ]:

	CRIM	ZN	INDUS	CHAS	NOX	RM
count	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
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000

	AGE	DIS	RAD	TAX	PTRATIO	B
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032
std	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864
min	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000
25%	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500
50%	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000
75%	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000
max	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000

	LSTAT	PRICE
count	506.000000	506.000000
mean	12.653063	22.532806
std	7.141062	9.197104
min	1.730000	5.000000
25%	6.950000	17.025000
50%	11.360000	21.200000
75%	16.955000	25.000000

```
max      37.970000   50.000000
```

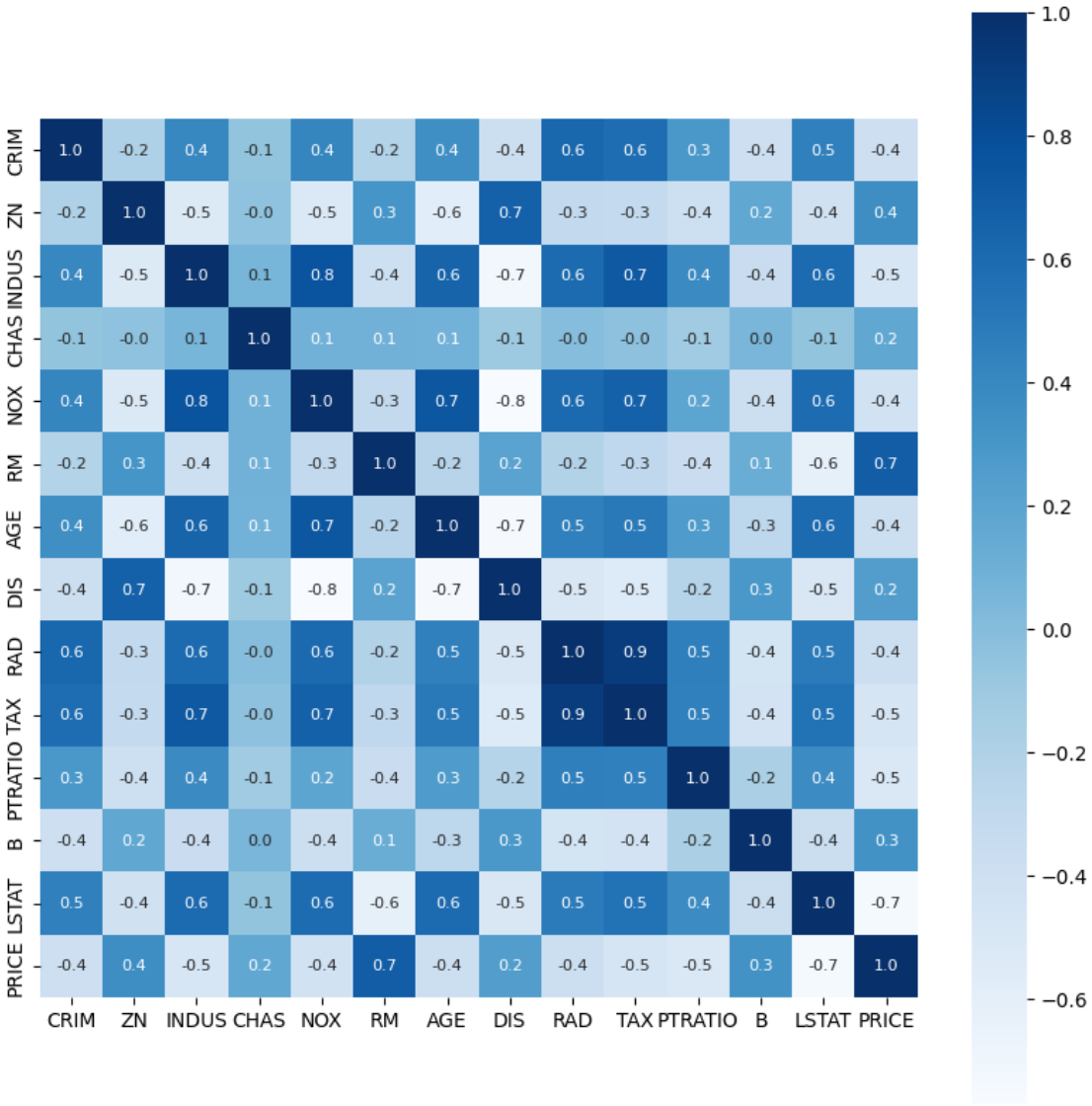
3 Understanding the correlation between various features in the dataset

1. Positive Correlation
2. Negative Correlation

[ ]:
```
correlation = house_price_dataset.corr()
```

[ ]:
```
# Constructing a heatmap to unerstand the correlation
plt.figure(figsize=(10,10))
sns.heatmap(correlation, cbar=True, square=True, fmt='.1f', annot=True,␣
 ↪annot_kws={'size':8}, cmap='Blues')
```

[ ]:
```
<Axes: >
```



4 Splitting the Data and Target

[ ]:
```
X = house_price_dataset.drop(['PRICE'], axis=1)
Y = house_price_dataset['PRICE']
```

[ ]:
```
print(X)
print(Y)
```

```
      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]
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
```

5 Splitting the data into Training data and Test data

[ ]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,␣
↪random_state = 2)
```

[ ]:
```
print(X.shape, X_train.shape, X_test.shape)
```

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

6 Model Training (XGBoost Regressor)

[ ]:
```
# Loading the Model
model = XGBRegressor()
```

[ ]:
```
# Training the Model with X_train
model.fit(X_train, Y_train)
```

[ ]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=None, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=None,
             n_jobs=None, num_parallel_tree=None, …)
```

#Evaluation

7 Prediction on training data

[ ]:
```
# Accuracy for prediction on training data
training_data_prediction = model.predict(X_train)
```

[ ]:
```
print(training_data_prediction)
```

```
[23.112196  20.992601  20.10438   34.67932   13.920501  13.499354
 21.998383  15.206723  10.89543   22.67402   13.795236   5.602332
 29.808502  49.98666   34.89634   20.594336  23.388903  19.2118
 32.69294   19.604128  26.978151   8.405952  46.00062   21.70406
 27.084402  19.372278  19.297894  24.79984   22.608278  31.707775
 18.53683    8.703393  17.40025   23.698814  13.29729   10.504759
 12.693588  24.994888  19.694864  14.911037  24.20254   24.991112
 14.901547  16.987965  15.592753  12.704759  24.505623  15.007718
 49.999355  17.509344  21.18844   31.999287  15.606071  22.902134
 19.309835  18.697083  23.302961  37.19767   30.102247  33.117855
 20.993683  50.00471   13.40048    5.002565  16.50862    8.4016905
 28.651423  19.49218   20.595366  45.404697  39.808857  33.4055
 19.81498   33.406376  25.30206   49.998615  12.544487  17.433802
 18.602612  22.601418  50.004013  23.814182  23.313164  23.097467
 41.71243   16.112017  31.604454  36.09397    7.0009975 20.406271
 19.992195  12.003392  25.027754  49.98552   37.890903  23.091173
 41.289513  17.604618  16.30125   30.05175   22.884857  19.802671
 17.106977  18.903633  18.897047  22.598665  23.170893  33.19197
```

15.00434 11.704804 18.795511 20.817484 17.998543 19.633396
49.998672 17.208574 16.410513 17.506626 14.6008 33.09849
14.504811 43.813366 34.900055 20.388191 14.605566 8.091776
11.777508 11.811628 18.691 6.322443 23.97163 13.073076
19.595 49.99033 22.319597 18.91175 31.203646 20.712711
32.200443 36.188755 14.222898 15.705663 50.000664 20.408077
16.185907 13.410434 50.012474 31.60327 12.288182 19.18906
29.809902 31.49241 22.804003 10.194443 24.09609 23.705154
22.008154 13.790835 28.399841 33.199585 13.102867 19.017357
26.61559 36.963135 30.7939 22.80785 10.206419 22.19713
24.482466 36.19345 23.092129 20.12124 19.498154 10.796299
22.701403 19.49908 20.107922 9.625605 42.797676 48.79655
13.099009 20.29537 24.794712 14.106459 21.698246 22.188694
32.99889 21.09952 24.998121 19.110165 32.401157 13.601795
15.072056 23.06062 27.487326 19.401924 26.481848 27.50343
28.686726 21.19214 18.701029 26.7093 14.01264 21.699009
18.39739 43.11556 29.09378 20.298742 23.711458 18.30434
17.193619 18.321108 24.392206 26.391497 19.10248 13.302614
22.189732 22.199099 8.530714 18.889635 21.800455 19.305798
18.198288 7.4938145 22.400797 20.028303 14.404203 22.500402
28.504164 21.608568 13.798578 20.495127 21.902288 23.100073
50.00128 16.23443 30.298399 49.996014 17.78638 19.060133
10.39715 20.383387 16.496948 17.195917 16.681927 19.509869
30.502445 29.01701 19.558786 23.172018 24.397314 9.528121
23.894762 49.996834 21.196695 22.596247 19.989746 13.393513
19.995872 17.068512 12.718964 23.01111 15.199219 20.609226
26.19055 18.109114 24.098877 14.100204 21.695303 20.096022
25.018776 27.899471 22.918222 18.499252 22.202477 23.99494
14.8048935 19.896328 24.411158 17.790047 24.596226 32.007046
17.778685 23.309103 16.120615 13.003008 10.993355 24.306978
15.597863 35.20248 19.58716 42.29605 8.789314 24.399925
14.109244 15.4010315 17.299047 22.113592 23.106049 44.805172
17.795519 31.499706 22.813938 16.836212 23.911596 12.09551
38.69628 21.387049 16.001123 23.929094 11.897898 24.983562
7.1969633 24.69086 18.187803 22.471941 23.013317 24.295506
17.099222 17.796907 13.503164 27.094381 13.296886 21.90404
19.99361 15.402385 16.588629 22.29326 24.697983 21.428938
22.882269 29.601665 21.881992 19.908726 29.60596 23.408524
13.807421 24.499699 11.901903 7.20547 20.484905 9.706262
48.301437 25.194635 11.691466 17.39672 14.49594 28.584557
19.395731 22.486904 7.0219784 20.60076 22.998001 19.699215
23.700571 25.02278 27.992222 13.39496 14.524017 20.30391
19.304321 24.108646 14.88511 26.387497 33.31608 23.61982
24.60193 18.494753 20.90211 10.411172 23.305649 13.097067
24.699335 22.610847 20.50208 16.82098 10.198874 33.805454
18.60289 50.0009 23.778967 23.91014 21.15922 18.81689
8.491747 21.506403 23.200815 21.043766 16.604784 28.060492
21.197857 28.370916 14.2918625 49.997353 30.989647 24.980095

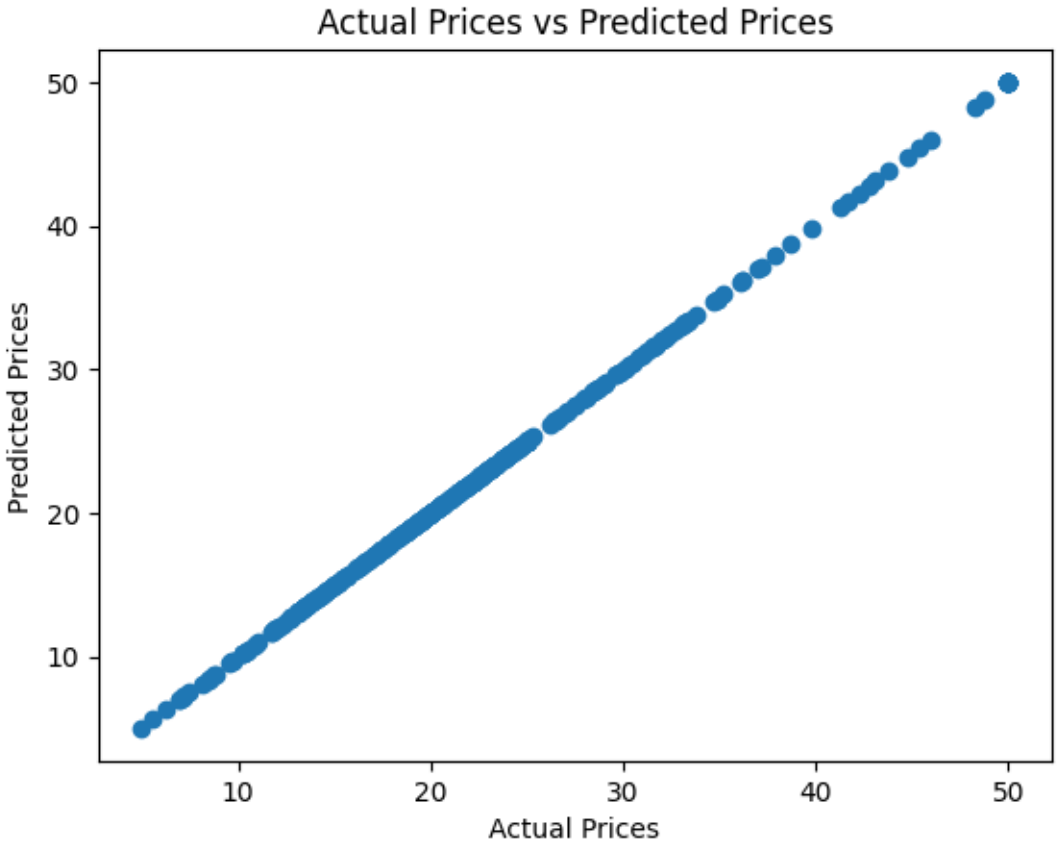
```
21.410505   19.000553   29.00484    15.204052   22.791481   21.791014
19.896528   23.77255   ]
```

[ ]:
```
# R squared error
score_1 = metrics.r2_score(Y_train, training_data_prediction)

# Mean Absolute Error
score_2 = metrics.mean_absolute_error(Y_train, training_data_prediction)

print("R squared error : ", score_1)
print("Mean Absolute Error : ", score_2)
```

```
R squared error :  0.9999980039471451
Mean Absolute Error :  0.0091330346494618
```

8 Visualizing the actual prices and predicted prices

[ ]:
```
plt.scatter(Y_train, training_data_prediction)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Prices vs Predicted Prices")
plt.show()
```



9 Prediction on Test Data

[ ]:
```
# Accuracy for prediction on test data
test_data_prediction = model.predict(X_test)
```

[ ]:
```
# R squared error
score_1 = metrics.r2_score(Y_test, test_data_prediction)

# Mean Absolute Error
score_2 = metrics.mean_absolute_error(Y_test, test_data_prediction)

print("R squared error : ", score_1)
print("Mean Absolute Error : ", score_2)
```

```
R squared error :  0.9051721149855378
Mean Absolute Error :  2.0748727686264927
```