House Price Prediction

> About This Project

In this house price prediction project, we employed the XGBoost regression model to predict house prices based on various features from the House Price Dataset. Here's a summary of the steps we followed and the key insights from the analysis:

1. **Data Preprocessing:**
    - Loaded the House Price Dataset and checked its dimensions.
    - Explored the dataset for missing values and statistical measures.
    - Visualized the correlations between features using a heatmap to understand feature relationships.
2. **Data Splitting:**
    - Split the dataset into training and testing sets using the `train_test_split` function.
    - Verified the shape of the training and testing sets.
3. **Model Training and Evaluation:**
    - Loaded the XGBoost regressor model.
    - Trained the model using the training data.
    - Evaluated the model's performance on the training set:
        - Calculated the R-squared error, which indicated that the model explained the variance in the training data very well.
        - Calculated the Mean Absolute Error, which quantified the average difference between actual and predicted prices.
4. **Visualization:**
    - Plotted a scatter plot of actual prices vs. predicted prices for both the training and testing sets. The visualizations showed how well the model's predictions aligned with the actual prices.
5. **Test Prediction and Evaluation:**
    - Made predictions on the test dataset using the trained model.
    - Evaluated the model's performance on the test set:
        - Calculated the R-squared error, which indicated the model's predictive power on unseen data.
        - Calculated the Mean Absolute Error, providing insight into the average prediction error.
6. **Conclusion:**
    - The XGBoost regression model demonstrated remarkable predictive capabilities for the Boston House Price Dataset.
    - The high R-squared value on both the training and test sets indicated that the model generalized well to unseen data.

- The low Mean Absolute Error further confirmed the model's effectiveness in predicting house prices.

Overall, the results of the house price prediction project using XGBoost regression were promising. The model's strong performance on both training and testing datasets suggests that it can provide accurate predictions for house prices based on the given features. The scatter plots visualized the alignment between predicted and actual prices, highlighting the model's accuracy. However, it's important to keep in mind that further optimization, hyperparameter tuning, and potential feature engineering could be explored to enhance the model's performance even more.

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
```

Importing the Boston House Price Dataset

```
house_price_dataframe = pd.read_csv('/content/HousingData.csv')
```

```
print(house_price_dataframe)
```

```
        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  296
1    0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622    3  222
..       ...   ...    ...   ...    ...    ...   ...     ...  ...  ...
501  0.06263   0.0  11.93   0.0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93   0.0  0.573  6.120  76.7  2.2875    1  273
503  0.06076   0.0  11.93   0.0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93   0.0  0.573  6.794  89.3  2.3889    1  273
505  0.04741   0.0  11.93   0.0  0.573  6.030   NaN  2.5050    1  273

     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    NaN   36.2
```

```
..     ...    ...    ...    ...
501    21.0   391.99   NaN   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]
```

```
# Print First 5 rows of our DataFrame
house_price_dataframe.head()
```

	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	296	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	242	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	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	NaN	36.2

```
# checking the number of rows and Columns in the data frame
house_price_dataframe.shape
```

```
(506, 14)
```

```
# check for missing values
house_price_dataframe.isnull().sum()
```

```
CRIM       20
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
PRICE       0
dtype: int64
```

```
# statistical measures of the dataset
house_price_dataframe.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	
count	486.000000	486.000000	486.000000	486.000000	506.000000	506.000000	486.000000	506.000000	506.
mean	3.611874	11.211934	11.083992	0.069959	0.554695	6.284634	68.518519	3.795043	9.
std	8.720192	23.388876	6.835896	0.255340	0.115878	0.702617	27.999513	2.105710	8.
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.
25%	0.081900	0.000000	5.190000	0.000000	0.449000	5.885500	45.175000	2.100175	4.
50%	0.253715	0.000000	9.690000	0.000000	0.538000	6.208500	76.800000	3.207450	5.
75%	3.560263	12.500000	18.100000	0.000000	0.624000	6.623500	93.975000	5.188425	24.
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.

Understanding the correlation between various features in the dataset

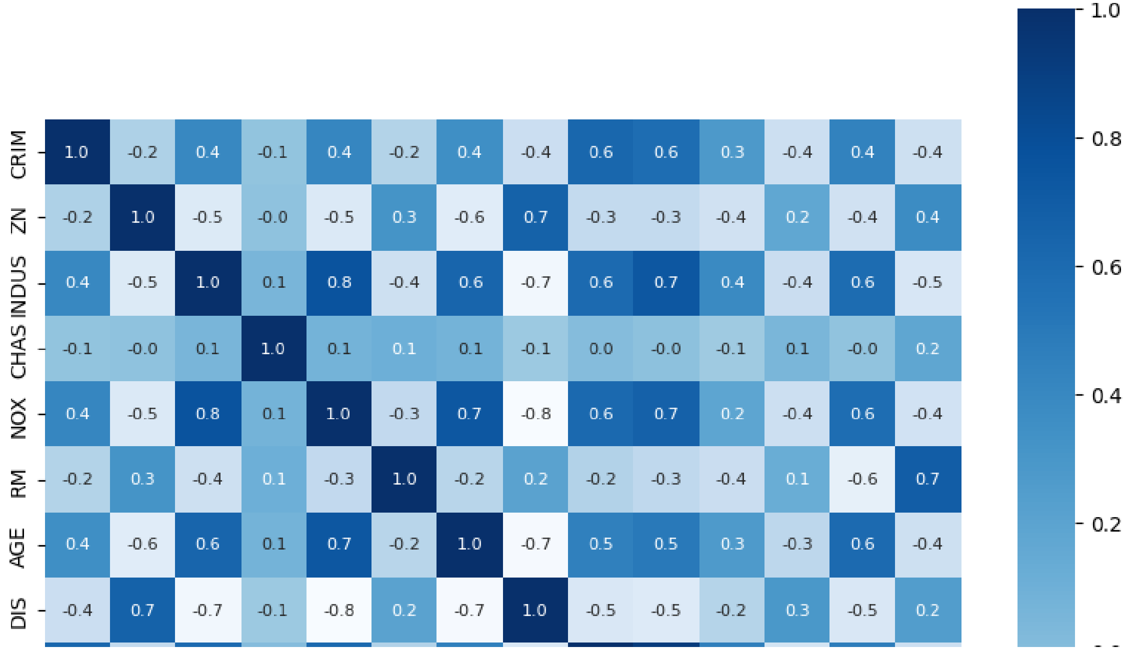
Positive Correlation

Negative Correlation

```
correlation = house_price_dataframe.corr()
```

```
# constructing a heatmap to nderstand 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: >
```



Splitting the data & Target

```
X = house_price_dataframe.drop(['PRICE'], axis=1)
Y = house_price_dataframe['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  296
1    0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622    3  222
..       ...   ...    ...   ...    ...    ...   ...     ...  ...  ...
501  0.06263   0.0  11.93   0.0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93   0.0  0.573  6.120  76.7  2.2875    1  273
503  0.06076   0.0  11.93   0.0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93   0.0  0.573  6.794  89.3  2.3889    1  273
505  0.04741   0.0  11.93   0.0  0.573  6.030   NaN  2.5050    1  273

     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    NaN
..       ...     ...    ...
501     21.0  391.99    NaN
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
```

Training Data & test

```
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)
```

Model Training

```
# loading the model
model = XGBRegressor()
```

```
# training the model with X_train
model.fit(X_train, Y_train)
```

```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
```

Evaluation

```
             predictor=None, random_state=None, ...)
```

```
# accuracy for prediction on training data
training_data_prediction = model.predict(X_train)
```

```
print(training_data_prediction)
```

```
 29.804506  49.987877  34.888184  20.615887  23.385252  19.219105
 32.693237  19.607283  26.987926   8.400735  46.006306  21.70009
 27.08245   19.38461   19.299566  24.807394  22.600876  31.715944
 18.551641   8.707558  17.408579  23.699173  13.299077  10.500352
 12.715582  25.006472  19.694242  14.893264  24.218592  24.984715
 14.913808  17.005949  15.597038  12.702356  24.503147  15.003218
 49.999443  17.526314  21.187368  32.001503  15.596226  22.899336
 19.307703  18.713356  23.291584  37.196495  30.100304  33.104607
 20.99445   49.98314   13.39885    4.9928923 16.48852    8.403917
 28.697334  19.492647  20.58597   45.39965   39.802162  33.39493
 19.806774  33.40387   25.294323  49.999493  12.515287  17.44132
 18.611597  22.601511  50.002136  23.804588  23.334707  23.10605
 41.709064  16.125729  31.617798  36.096508   6.997423  20.380814
 19.996199  12.029594  24.961348  49.988277  37.897827  23.098347
 41.294544  17.60291   16.302446  30.032356  22.907495  19.810986
 17.09506   18.901705  18.948738  22.578093  23.171654  33.21098
 15.005704  11.70465   18.800985  20.792173  17.985819  19.648312
 50.004223  17.206322  16.416636  17.507555  14.60386   33.103535
 14.498311  43.815643  34.948956  20.40538   14.605213   8.092749
 11.785861  11.831506  18.693441   6.3084145 23.953945  13.0773945
 19.594473  49.998096  22.31341   18.905788  31.195295  20.697218
 32.203373  36.18067   14.214334  15.695019  49.998615  20.405207
 16.200977  13.408519  49.99794   31.600187  12.290271  19.216589
 29.799414  31.501423  22.813274  10.191812  24.096865  23.714815
 22.007042  13.803389  28.411673  33.18088   13.10673   18.995567
 26.598248  36.971924  30.79777   22.779976  10.211277  22.200876
 24.467127  36.19966   23.094261  20.11149   19.48739   10.796084
 22.66935   19.488937  20.105448   9.614282  42.789986  48.796795
 13.078567  20.304855  24.783684  14.0974865 21.697916  22.20561
 32.999634  21.11631   25.009466  19.109894  32.405125  13.601782
 15.092909  23.06247   27.497938  19.375496  26.495235  27.498268
 28.697725  21.232346  18.695868  26.731031  14.007644  21.689535
 18.388357  43.089417  29.0748    20.285393  23.710909  18.284605
 17.209354  18.319569  24.40191   26.391329  19.077065  13.293503
```

```
50.006916  16.22731   30.301104  50.017963  17.784174  19.056034
10.387393  20.391016  16.50506   17.192429  16.702799  19.511196
30.51736   28.99166   19.55188   23.183167  24.382183   9.504991
23.899569  49.989056  21.17416   22.604053  19.994152  13.396168
19.984293  17.110525  12.7490635 22.997908  15.223642  20.594662
26.237635  18.111963  24.099932  14.086146  21.697147  20.083914
25.014418  27.89823   22.931677  18.497055  22.178623  24.003244
14.795677  19.887085  24.404215  17.7806    24.589611  31.975996
17.80095   23.331669  16.110304  13.005892  10.997909  24.29056
15.575491  35.209496  19.619333  42.29822    8.792996  24.402912
14.126401  15.379655  17.305126  22.120369  23.094246  44.790134
17.80082   31.505554  22.814024  16.8487    23.912342  12.096439
38.687733  21.384914  16.006336  23.926025  11.9002285 24.975077
 7.1953726 24.699255  18.193438  22.484354  23.043955  24.287437
17.10062   17.798908  13.511288  27.066021  13.304795  21.904535
20.021526  15.383979  16.599194  22.294048  24.703049  21.40998
22.914837  29.597427  21.887817  19.887808  29.605515  23.405313
13.791948  24.459793  11.904582   7.2066965 20.496056   9.691774
48.30093   25.18632   11.695794  17.403475  14.494502  28.586044
19.38797   22.472443   7.019776  20.598795  22.977278  19.691525
23.683409  25.019066  27.948833  13.3966675 14.509144  20.314003
19.30595   24.096642  14.891865  26.387436  33.29164   23.610031
```

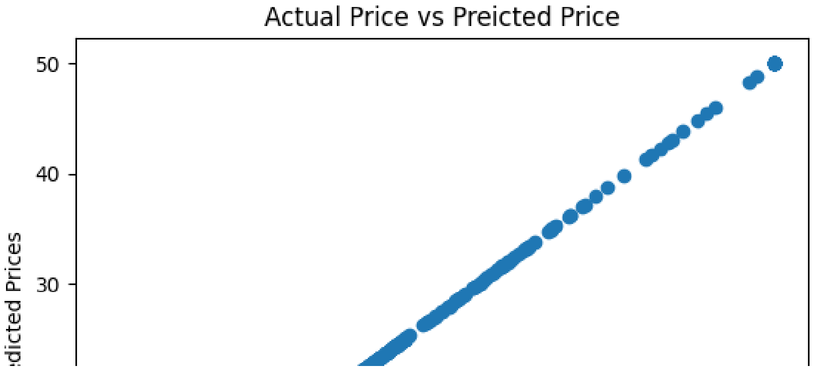
```
# R squared error
score_1 = metrics.r2_score(Y_train, training_data_prediction)

# 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 :  0.9999970506674762
Mean Absolute Error :  0.011185854968458139
```

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



Test Predection

```
# accuracy for prediction on test data
test_data_prediction = model.predict(X_test)
```

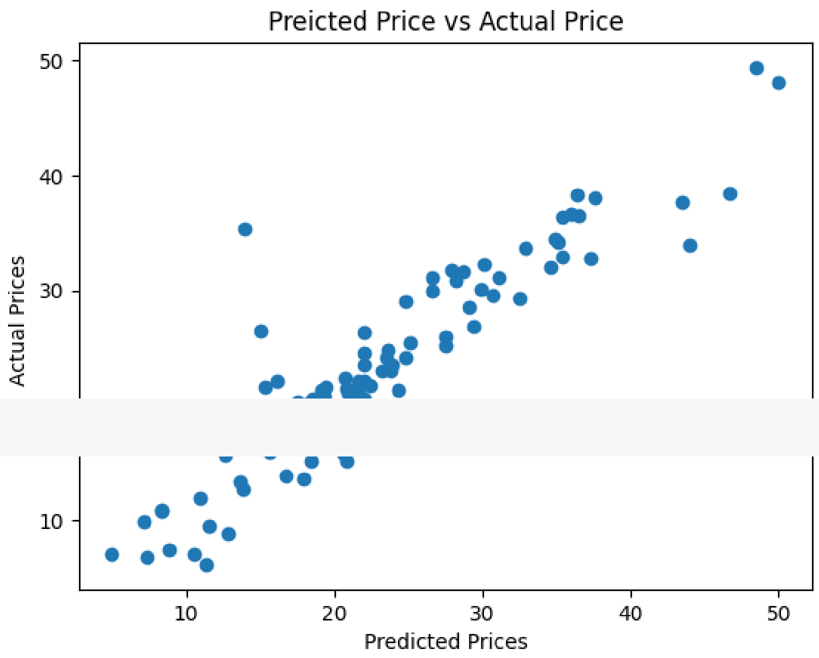
```
# R squared error
score_1 = metrics.r2_score(Y_test, test_data_prediction)

# 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 :  0.8361097215940602
Mean Absolute Error :  2.466887313244389
```

```
plt.scatter(Y_test, test_data_prediction)
plt.xlabel("Predicted Prices")
plt.ylabel("Actual Prices ")
plt.title("Preicted Price vs Actual Price")
plt.show()
```


Preicted Price vs Actual Price
Actual Prices
Predicted Prices
50
40
30
10
10
20
30
40
50