```
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_dataset = sklearn.datasets.fetch_california_housing()

print(house_price_dataset)
```

```
{'data': array([[   8.3252    ,   41.        ,    6.98412698, ...,
2.55555556,
          37.88      , -122.23      ],
       [   8.3014    ,   21.        ,    6.23813708, ...,
2.10984183,
          37.86      , -122.22      ],
       [   7.2574    ,   52.        ,    8.28813559, ...,
2.80225989,
          37.85      , -122.24      ],
       ...,
       [   1.7       ,   17.        ,    5.20554273, ...,    2.3256351
,
          39.43      , -121.22      ],
       [   1.8672    ,   18.        ,    5.32951289, ...,
2.12320917,
          39.43      , -121.32      ],
       [   2.3886    ,   16.        ,    5.25471698, ...,
2.61698113,
          39.37      , -121.24      ]]), 'target': array([4.526,
3.585, 3.521, ..., 0.923, 0.847, 0.894]), 'frame': None,
'target_names': ['MedHouseVal'], 'feature_names': ['MedInc',
'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup',
'Latitude', 'Longitude'], 'DESCR': '.. _california_housing_dataset:\n\
nCalifornia Housing dataset\n--------------------------\n\n**Data Set
Characteristics:**\n\n    :Number of Instances: 20640\n\n    :Number
of Attributes: 8 numeric, predictive attributes and the target\n\
n    :Attribute Information:\n        - MedInc        median income in
block group\n        - HouseAge      median house age in block group\n
- AveRooms      average number of rooms per household\n        -
AveBedrms     average number of bedrooms per household\n        -
Population    block group population\n        - AveOccup      average
number of household members\n        - Latitude      block group
```

```
latitude\n        - Longitude     block group longitude\n\
n    :Missing Attribute Values: None\n\nThis dataset was obtained from
the StatLib
repository.\nhttps://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.h
tml\n\nThe target variable is the median house value for California
districts,\nexpressed in hundreds of thousands of dollars ($100,000).\
n\nThis dataset was derived from the 1990 U.S. census, using one row
per census\nblock group. A block group is the smallest geographical
unit for which the U.S.\nCensus Bureau publishes sample data (a block
group typically has a population\nof 600 to 3,000 people).\n\nA
household is a group of people residing within a home. Since the
average\nnumber of rooms and bedrooms in this dataset are provided per
household, these\ncolumns may take surprisingly large values for block
groups with few households\nand many empty houses, such as vacation
resorts.\n\nIt can be downloaded/loaded using the\
n:func:`sklearn.datasets.fetch_california_housing` function.\n\n..
topic:: References\n\n    - Pace, R. Kelley and Ronald Barry, Sparse
Spatial Autoregressions,\n      Statistics and Probability Letters, 33
(1997) 291-297\n'}
```

```
# Loading the dataset to a pandas dataframe
house_price_dataframe = pd.DataFrame(house_price_dataset.data, columns
= house_price_dataset.feature_names)
```

```
house_price_dataframe.head()
```

```
   MedInc  HouseAge  AveRooms  ...  AveOccup  Latitude  Longitude
0  8.3252      41.0  6.984127  ...  2.555556     37.88    -122.23
1  8.3014      21.0  6.238137  ...  2.109842     37.86    -122.22
2  7.2574      52.0  8.288136  ...  2.802260     37.85    -122.24
3  5.6431      52.0  5.817352  ...  2.547945     37.85    -122.25
4  3.8462      52.0  6.281853  ...  2.181467     37.85    -122.25

[5 rows x 8 columns]
```

```
# add the target column to the dataframe
house_price_dataframe['price'] = house_price_dataset.target
```

```
house_price_dataframe.head()
```

```
   MedInc  HouseAge  AveRooms  AveBedrms  ...  AveOccup  Latitude
Longitude  price
0  8.3252      41.0  6.984127   1.023810  ...  2.555556     37.88    -
122.23  4.526
1  8.3014      21.0  6.238137   0.971880  ...  2.109842     37.86    -
122.22  3.585
2  7.2574      52.0  8.288136   1.073446  ...  2.802260     37.85    -
122.24  3.521
3  5.6431      52.0  5.817352   1.073059  ...  2.547945     37.85    -
122.25  3.413
4  3.8462      52.0  6.281853   1.081081  ...  2.181467     37.85    -
```

```
122.25  3.422

[5 rows x 9 columns]
```

```
# checking the number of rows and columns in the dataframe
house_price_dataframe.shape
```

```
(20640, 9)
```

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

```
<bound method NDFrame._add_numeric_operations.<locals>.sum of
MedInc  HouseAge  AveRooms  ...  Latitude  Longitude  price
0       False     False     False  ...     False      False  False
1       False     False     False  ...     False      False  False
2       False     False     False  ...     False      False  False
3       False     False     False  ...     False      False  False
4       False     False     False  ...     False      False  False
...       ...       ...       ...  ...       ...        ...    ...
20635   False     False     False  ...     False      False  False
20636   False     False     False  ...     False      False  False
20637   False     False     False  ...     False      False  False
20638   False     False     False  ...     False      False  False
20639   False     False     False  ...     False      False  False

[20640 rows x 9 columns]>
```

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

```
             MedInc      HouseAge  ...     Longitude         price
count  20640.000000  20640.000000  ...  20640.000000  20640.000000
mean       3.870671     28.639486  ...   -119.569704      2.068558
std        1.899822     12.585558  ...      2.003532      1.153956
min        0.499900      1.000000  ...   -124.350000      0.149990
25%        2.563400     18.000000  ...   -121.800000      1.196000
50%        3.534800     29.000000  ...   -118.490000      1.797000
75%        4.743250     37.000000  ...   -118.010000      2.647250
max       15.000100     52.000000  ...   -114.310000      5.000010

[8 rows x 9 columns]
```

Understanding the **correlation** between various features in the dataset

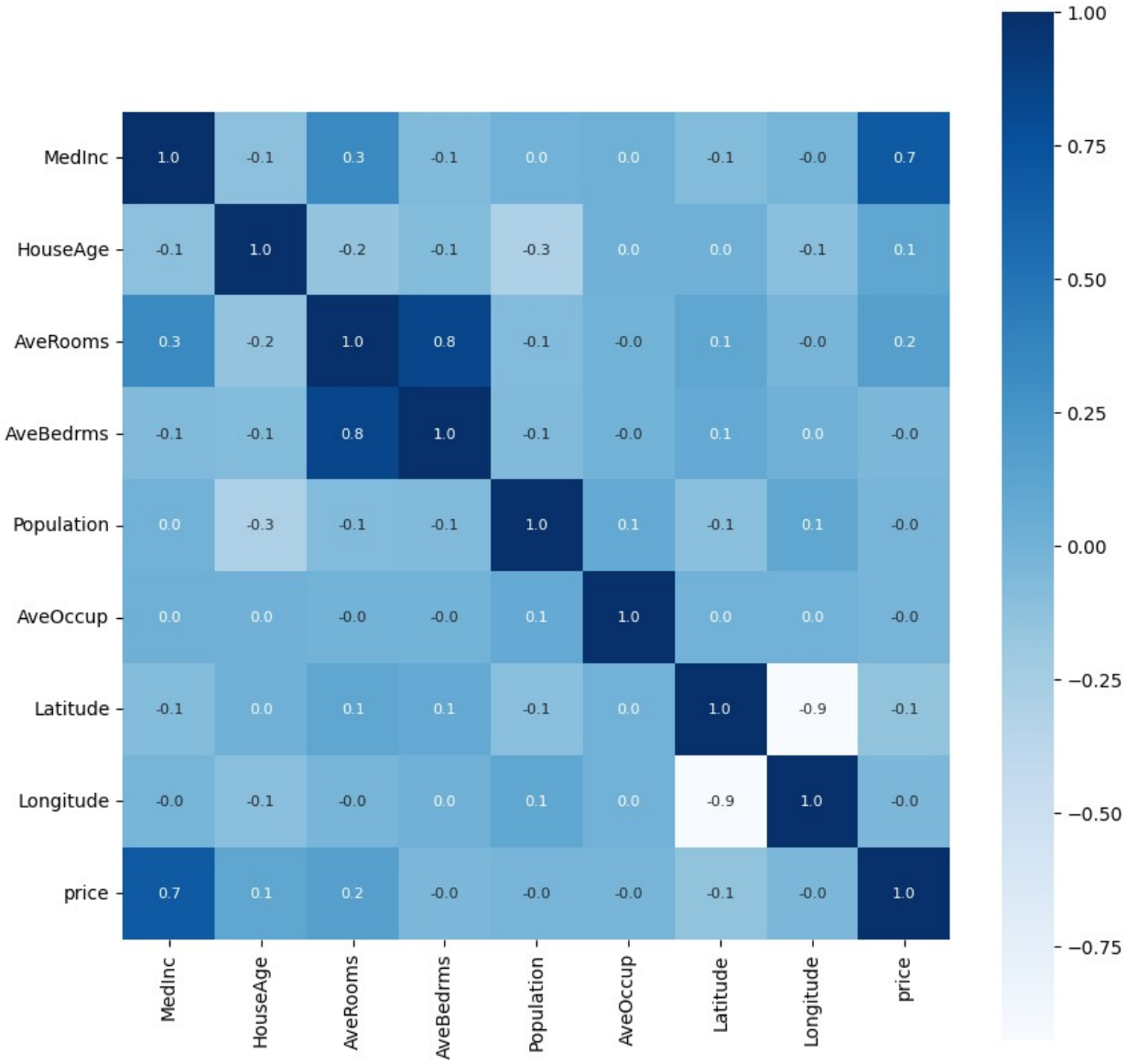
1. Positive Correlation
2. Negative Correlation

```
correlation = house_price_dataframe.corr()

# constructing a heatmap to understand 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 and target

```
X = house_price_dataframe.drop(['price'], axis=1)
Y = house_price_dataframe['price']

print(X,Y)
```

```
       MedInc  HouseAge  AveRooms  ...  AveOccup  Latitude  Longitude
0      8.3252      41.0  6.984127  ...  2.555556     37.88    -122.23
1      8.3014      21.0  6.238137  ...  2.109842     37.86    -122.22
2      7.2574      52.0  8.288136  ...  2.802260     37.85    -122.24
3      5.6431      52.0  5.817352  ...  2.547945     37.85    -122.25
4      3.8462      52.0  6.281853  ...  2.181467     37.85    -122.25
...       ...       ...       ...  ...       ...       ...         ...
20635  1.5603      25.0  5.045455  ...  2.560606     39.48    -121.09
20636  2.5568      18.0  6.114035  ...  3.122807     39.49    -121.21
20637  1.7000      17.0  5.205543  ...  2.325635     39.43    -121.22
20638  1.8672      18.0  5.329513  ...  2.123209     39.43    -121.32
20639  2.3886      16.0  5.254717  ...  2.616981     39.37    -121.24

[20640 rows x 8 columns] 0        4.526
1        3.585
2        3.521
3        3.413
4        3.422
         ...
20635    0.781
20636    0.771
20637    0.923
20638    0.847
20639    0.894
Name: price, Length: 20640, dtype: float64
```

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)

(20640, 8) (16512, 8) (4128, 8)
```

Model Training

XGBoost Regressor

```
# load 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, 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,
             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,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
```

Evaluation

Prediction on training data

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

print(training_data_prediction)
```

```
[0.6893792  2.986824   0.48874274 ... 1.8632544  1.7800125
0.7565893 ]
```

```
# 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 Sqaured Error:', score_1)
print('Mean Absolute Error:', score_2)
```

```
R Sqaured Error: 0.9451221492760822
Mean Absolute Error: 0.1919170860794262
```

Visualize the actuale prices and predicted prices

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

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 Sqaured Error:', score_1)
print('Mean Absolute Error:', score_2)
```

```
R Sqaured Error: 0.8412904408180302
Mean Absolute Error: 0.30753655785801337
```