### 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 sklearn import metrics
from xgboost import XGBRegressor
```

### Importing the Datasets

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

```
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.html\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.. rubric:: References\n\n- Pace, R. Kelley and Ronald Barry, Sparse Spatial
Autoregressions,\n  Statistics and Probability Letters, 33 (1997) 291-297\n'}
```

```
1 # Transforming Dataset into DataFrame using Pandas
2 house_price_dataframe = pd.DataFrame(house_price_dataset.data)
```

```
1 house_price_dataframe.head()
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **0** | 8.3252 | 41.0 | 6.984127 | 1.023810 | 322.0 | 2.555556 | 37.88 | -122.23 |
| **1** | 8.3014 | 21.0 | 6.238137 | 0.971880 | 2401.0 | 2.109842 | 37.86 | -122.22 |
| **2** | 7.2574 | 52.0 | 8.288136 | 1.073446 | 496.0 | 2.802260 | 37.85 | -122.24 |
| **3** | 5.6431 | 52.0 | 5.817352 | 1.073059 | 558.0 | 2.547945 | 37.85 | -122.25 |
| **4** | 3.8462 | 52.0 | 6.281853 | 1.081081 | 565.0 | 2.181467 | 37.85 | -122.25 |

Next steps: Generate code with `house_price_dataframe` | View recommended plots | New interactive sheet

```
1 # Adding columns name to the DataFrame
2 house_price_dataframe = pd.DataFrame(house_price_dataset.data, columns=house_price_dataset.feature_names)
```

```
1 house_price_dataframe.head()
```

| | MedInc | HouseAge | AveRooms | AveBedrms | Population | AveOccup | Latitude | Longitude |
|---|---|---|---|---|---|---|---|---|
| **0** | 8.3252 | 41.0 | 6.984127 | 1.023810 | 322.0 | 2.555556 | 37.88 | -122.23 |
| **1** | 8.3014 | 21.0 | 6.238137 | 0.971880 | 2401.0 | 2.109842 | 37.86 | -122.22 |
| **2** | 7.2574 | 52.0 | 8.288136 | 1.073446 | 496.0 | 2.802260 | 37.85 | -122.24 |
| **3** | 5.6431 | 52.0 | 5.817352 | 1.073059 | 558.0 | 2.547945 | 37.85 | -122.25 |
| **4** | 3.8462 | 52.0 | 6.281853 | 1.081081 | 565.0 | 2.181467 | 37.85 | -122.25 |

Next steps: Generate code with `house_price_dataframe` | View recommended plots | New interactive sheet

```
1 # Adding price column to the Dataset
2 house_price_dataframe['price'] = house_price_dataset.target
```

```
1 house_price_dataframe.head()
```

| | MedInc | HouseAge | AveRooms | AveBedrms | Population | AveOccup | Latitude | Longitude | price |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 8.3252 | 41.0 | 6.984127 | 1.023810 | 322.0 | 2.555556 | 37.88 | -122.23 | 4.526 |
| 1 | 8.3014 | 21.0 | 6.238137 | 0.971880 | 2401.0 | 2.109842 | 37.86 | -122.22 | 3.585 |
| 2 | 7.2574 | 52.0 | 8.288136 | 1.073446 | 496.0 | 2.802260 | 37.85 | -122.24 | 3.521 |
| 3 | 5.6431 | 52.0 | 5.817352 | 1.073059 | 558.0 | 2.547945 | 37.85 | -122.25 | 3.413 |
| 4 | 3.8462 | 52.0 | 6.281853 | 1.081081 | 565.0 | 2.181467 | 37.85 | -122.25 | 3.422 |

Next steps: Generate code with `house_price_dataframe` View recommended plots New interactive sheet

```
# Finding the Shape of the DataFrame
house_price_dataframe.shape
```

```
(20640, 9)
```

```
# Finding any Null value present int the DataFrame
house_price_dataframe.isnull().sum()
```

| | 0 |
|---|---|
| MedInc | 0 |
| HouseAge | 0 |
| AveRooms | 0 |
| AveBedrms | 0 |
| Population | 0 |
| AveOccup | 0 |
| Latitude | 0 |
| Longitude | 0 |
| price | 0 |

**dtype:** int64

```
house_price_dataframe.dtypes
```

| | 0 |
|---|---|
| **MedInc** | float64 |
| **HouseAge** | float64 |
| **AveRooms** | float64 |
| **AveBedrms** | float64 |
| **Population** | float64 |
| **AveOccup** | float64 |
| **Latitude** | float64 |
| **Longitude** | float64 |
| **price** | float64 |

**dtype:** object

```
# Summary Statistics
house_price_dataframe.describe()
```

| | MedInc | HouseAge | AveRooms | AveBedrms | Population | AveOccup | Latitude | Longitude | price |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 |
| **mean** | 3.870671 | 28.639486 | 5.429000 | 1.096675 | 1425.476744 | 3.070655 | 35.631861 | -119.569704 | 2.068558 |
| **std** | 1.899822 | 12.585558 | 2.474173 | 0.473911 | 1132.462122 | 10.386050 | 2.135952 | 2.003532 | 1.153956 |
| **min** | 0.499900 | 1.000000 | 0.846154 | 0.333333 | 3.000000 | 0.692308 | 32.540000 | -124.350000 | 0.149990 |
| **25%** | 2.563400 | 18.000000 | 4.440716 | 1.006079 | 787.000000 | 2.429741 | 33.930000 | -121.800000 | 1.196000 |
| **50%** | 3.534800 | 29.000000 | 5.229129 | 1.048780 | 1166.000000 | 2.818116 | 34.260000 | -118.490000 | 1.797000 |
| **75%** | 4.743250 | 37.000000 | 6.052381 | 1.099526 | 1725.000000 | 3.282261 | 37.710000 | -118.010000 | 2.647250 |
| **max** | 15.000100 | 52.000000 | 141.909091 | 34.066667 | 35682.000000 | 1243.333333 | 41.950000 | -114.310000 | 5.000010 |

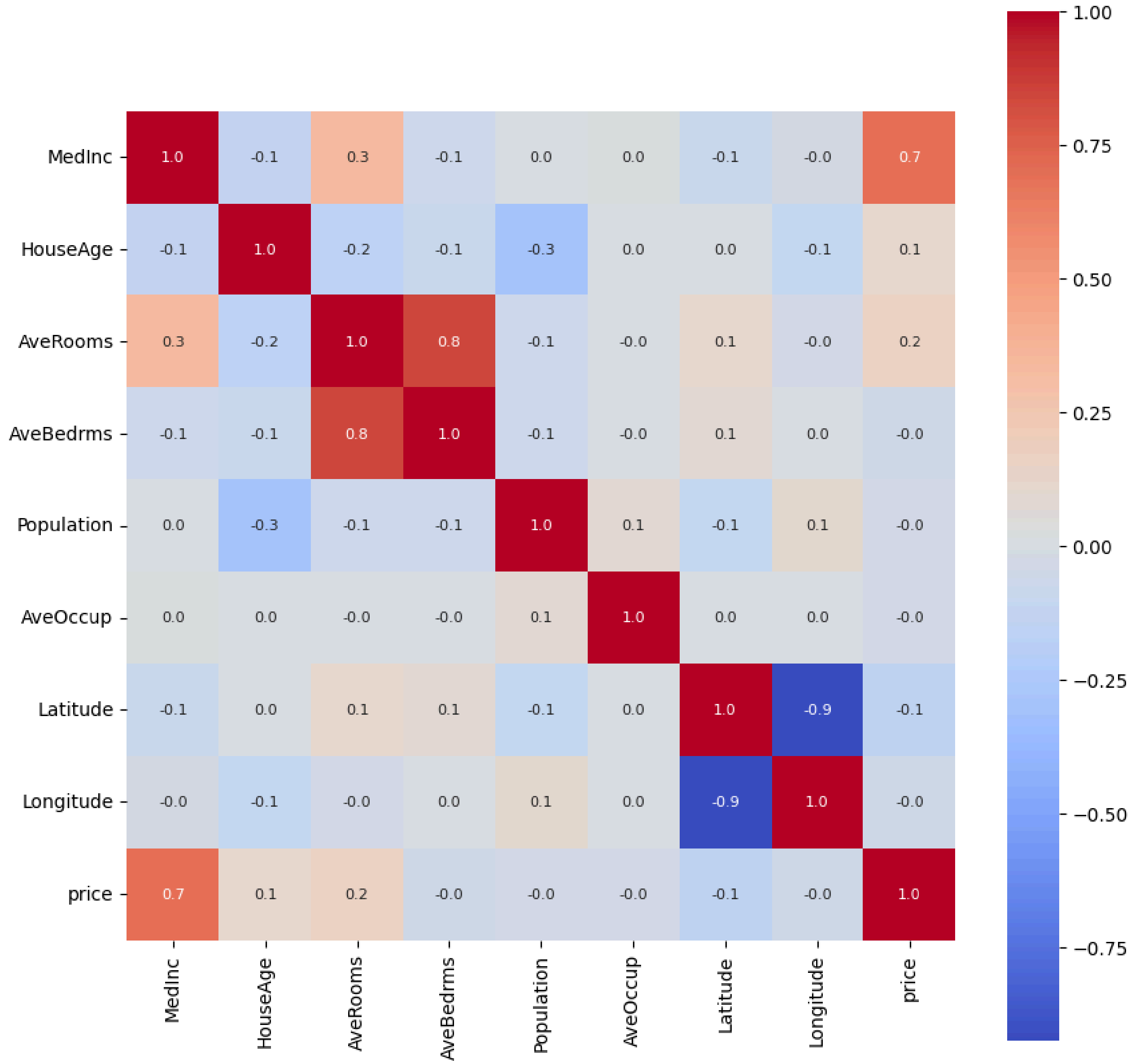
**Understanding the Correlation between variables present in the Dataset**

1. Positive Correlation
2. Negative Correlation

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

```
# Constructing a heatmap to understand the correlation between the variables
plt.figure(figsize=(10, 10))
sns.heatmap(correlation, cbar=True, square=True, fmt='0.1f', annot=True, annot_kws={'size': 8}, cmap='coolwarm')
```

```
<Axes: >
```




**Splitting the Data & Target**

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

```
print(X)
print(Y)
```

```
       MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0      8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1      8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2      7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3      5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4      3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85
...       ...       ...       ...        ...         ...       ...       ...
20635  1.5603      25.0  5.045455   1.133333       845.0  2.560606     39.48
20636  2.5568      18.0  6.114035   1.315789       356.0  3.122807     39.49
20637  1.7000      17.0  5.205543   1.120092      1007.0  2.325635     39.43
20638  1.8672      18.0  5.329513   1.171920       741.0  2.123209     39.43
20639  2.3886      16.0  5.254717   1.162264      1387.0  2.616981     39.37

       Longitude
0        -122.23
1        -122.22
2        -122.24
3        -122.25
4        -122.25
...          ...
20635    -121.09
20636    -121.21
20637    -121.22
20638    -121.32
20639    -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 & Testing 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

```
# Loading the Model
model = XGBRegressor()
```

```
# Training the Model with X_train
# model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test)
model.fit(X_train, Y_train)
```

```
XGBRegressor

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

#### Evaluation

#### Prediction on Training Data

```
# Accuracy for Prediction on Training Data
#training_data_prediction = classifier.predict(X_train)
#training_data_accuracy = accuracy_score(training_data_prediction, Y_train)
#print('Accuracy score of the training data : ', training_data_accuracy)
training_data_prediction = model.predict(X_train)
```

```
print(training_data_prediction)
```

```
[0.5523039 3.0850039 0.5835302 ... 1.9204227 1.952873  0.6768683]
```

```
# R Squared Error
error_score_1 = metrics.r2_score(Y_train, training_data_prediction)
print('R squared error : ', error_score_1)
# Mean Absolute Error
error_score_2 = metrics.mean_absolute_error(Y_train, training_data_prediction)
print('Mean Absolute Error : ', error_score_2)
```

```
R squared error :  0.943650140819218
Mean Absolute Error :  0.1933648700612105
```

If error score is 0 that is perfect accuracy. if 5-10 we have to adjust the model

### Visualizing the Actual Price & Predicted Price

```
plt.scatter(Y_train, training_data_prediction)
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Actual Prices VS Predicted Prices')
plt.show()
```


### Prediction on Test Data

```
# Accuracy for Prediction on Training Data
test_data_prediction = model.predict(X_test)
```

```
print(test_data_prediction)
```

```
[2.8649795  1.790346   0.92074925 ... 1.5385513  0.92647874 2.043316  ]
```

```
# R Squared Error
error_score_1 = metrics.r2_score(Y_test, test_data_prediction)
print('R squared error : ', error_score_1)
# Mean Absolute Error
error_score_2 = metrics.mean_absolute_error(Y_test, test_data_prediction)
print('Mean Absolute Error : ', error_score_2)
```

```
R squared error :  0.8338000331788725
Mean Absolute Error :  0.3108631800268186
```

```
# Thats It
```

```
Start coding or generate with AI.
```