PROJECT : Predicting House Prices Using Machine Learning

Submitted By : Kishore kumar

Mail ID : kishore8778939148@gmail.com

Phase-04 : Development Part-02

ABOUT THIS PHASE :

In this phase we need to do performing different activities like feature engineering, model training, evaluation etc as per the instructions in the project

Step 1:

Splitting data and target

In this step we need to split the data into two parts namely DATA and TARGET . in this step we declare the variable X for data and variable Y for target

Step 2:

Splitting the data into training and testing data

In this step I split my data into two component they are training data and testing data by using **train _test_split** command

Step 3:

Model Training

In this step I train my data by using **XGBoost regressor** algorithm

Step 4:

Fixing the train and test data to the model (XGBoost Regressor)

In this step I fit my train and test data to the model by using **model.fit** command

Step 5:

Prediction on train and test data

In this step to predict the train and test data by using **model.predict** command. And also find r square error and mean absolute error for train and test data

Step 6:

Visualizing the actual price and predicted price

In this step to generate prediction graph to to evaluate my project the gaph is created by using the module matplotlib.pyplot

Import 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
```

Impoeting the california house prise dataset

```
 from sklearn.datasets import fetch_california_housing
house_price_dataset = 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_na
```

```
# loading the dataset to the Pands DataFrame
house_price_dataframe = pd.DataFrame(house_price_dataset.data, columns = house_price_dataset.feature_names)
```

```
# print first 5 rows of our DataFrame
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

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

```
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

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

```
(20640, 9)
```

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

```
MedInc        0
HouseAge      0
AveRooms      0
AveBedrms     0
Population    0
AveOccup      0
Latitude      0
Longitude     0
price         0
dtype: int64
```

```
# statical measure of the dataset
house_price_dataframe.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOcc
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.0000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	3.0706
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.3860
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.6923
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.4297
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.8181
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.2822
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.3333

underatanding various feature in the dataset

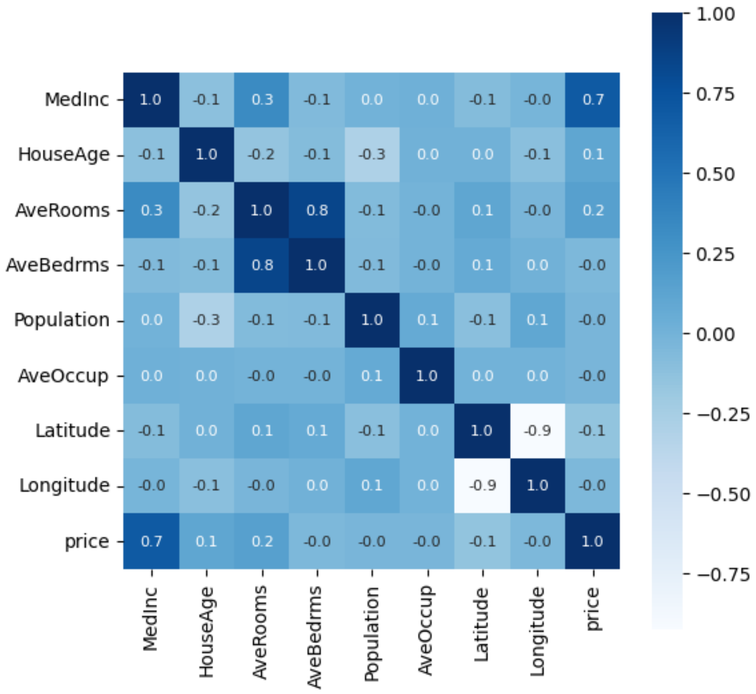
1.positive correlation 2.negative correlation

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

constructing the heatmap

```
# constructing the heatmap to understand the correlation
plt.figure(figsize=(6,6))
sns.heatmap(correlation, cbar=True, square=True, fmt='.1f', annot=True, annot_kws={'size':8}, cmap='Blues')
```

```
<Axes: >
```



splitting data and target

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

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

```
       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
...       ...       ...       ...        ...         ...       ...       ...        ...
20635  1.5603      25.0  5.045455   1.133333       845.0  2.560606     39.48    -121.09
20636  2.5568      18.0  6.114035   1.315789       356.0  3.122807     39.49    -121.21
20637  1.7000      17.0  5.205543   1.120092      1007.0  2.325635     39.43    -121.22
20638  1.8672      18.0  5.329513   1.171920       741.0  2.123209     39.43    -121.32
20639  2.3886      16.0  5.254717   1.162264      1387.0  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

```
# 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, 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

predection on training data

```
# accuracy for prediction on training data
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
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 :  0.943650140819218
mean absolute error :  0.1933648700612105
```

visualizing the actual price and predicted price

```
plt.scatter(Y_train, training_data_prediction)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Prices vs Predicted Prices")
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 squared error : ", score_1)
print("mean absolute error : ", score_2)
```

```
R squared error :  0.8338000331788725
mean absolute error :  0.3108631800268186
```