SUPERIOR UNIVERSITY

Submitted By:

Name: Ali Raza

Roll number Su92-bsdsm-f23-018

Section: 4A

Task: 02

Subject: Programming For AI (lab)

Submitted to: Sir Rasikh Ali

https://www.kaggle.com/code/alijatt1/house-price-prediction?scriptVersionId=224648643

Introduction

The **House Price Prediction** dataset is used for **regression tasks**, where the goal is to predict the price of houses based on various features. This report explains the structure of the Jupyter Notebook, key terms, and its applications.

Overview of the Notebook

The notebook consists of the following steps:

- **Data loading and preprocessing**
- **Exploratory Data Analysis (EDA)**
- **Feature selection and engineering**
- **Model training using machine learning algorithms**
- **Predictions and evaluation**

Key Terms and Their Definitions

1. **NumPy (numpy)** – A library for numerical computations, useful for handling arrays and mathematical operations.
2. **Pandas (pandas)** – A data manipulation library for reading, modifying, and analyzing tabular data.
3. **Matplotlib (matplotlib.pyplot) & Seaborn (seaborn)** – Libraries used for data visualization.
4. **Scikit-Learn (sklearn)** – A machine learning library that provides tools for data preprocessing, model training, and evaluation.
5. **XGBoost (XGBRegressor)** – A powerful machine learning algorithm used for regression tasks.
6. **Regression Task** – A type of machine learning problem where the goal is to predict continuous numerical values (e.g., house prices).
7. **Model Training** – The process of teaching a machine learning model to recognize patterns in data.
8. **Evaluation Metrics** – Measures such as RMSE (Root Mean Squared Error) and R^2 Score used to assess model performance.

Code Explanation

1. Importing Libraries

The notebook begins by importing essential Python libraries:

Importing Pandas and NumPy for data handling

import **pandas** as **pd**

import **numpy** as **np**

import **matplotlib.pyplot** as **plt**

import **seaborn** as **sns**

These libraries help with **data manipulation and visualization**.

2. Importing Machine Learning Libraries

import **sklearn.datasets**

from **sklearn.model_selection** import **train_test_split**

from **sklearn.base** import **BaseEstimator, RegressorMixin**

from **xgboost** import **XGBRegressor**

from **sklearn** import **metrics**

- **sklearn.datasets** – Loads datasets.
- **train_test_split** – Splits the data into training and testing sets.
- **XGBRegressor** – The machine learning model used for price prediction.
- **metrics** – Evaluates model performance.

3. Loading the Dataset

house_price_dataset = sklearn.datasets.fetch_california_housing()

This loads the **California Housing Dataset**, a well-known dataset for house price prediction.

4. Exploring the Dataset

print(house_price_dataset)

This prints the dataset's **features and target values** (house prices).

house_price_dataframe = pd.DataFrame(house_price_dataset.data, columns=house_price_dataset.feature_names)

The dataset is converted into a **Pandas DataFrame** for easier manipulation.

---

5. Feature Engineering & Preprocessing

- **Adding the Target Column (House Prices)**
- house_price_dataframe['Price'] = house_price_dataset.target

This adds the **house price** column to the dataset.

- **Checking for Missing Values**

- house_price_dataframe.isnull().sum()

Identifies missing values in the dataset.

- **Visualizing Feature Distributions**
- sns.pairplot(house_price_dataframe)
- plt.show()

This creates pair plots to visualize relationships between features and house prices.

---

6. Splitting the Data

Splitting dataset into training and testing sets

X = house_price_dataframe.drop(columns='Price', axis=1)

y = house_price_dataframe['Price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

- **X**: Input features (all columns except "Price").
- **y**: Target variable (house prices).
- **train_test_split**: Splits the data into **80% training** and **20% testing** for model evaluation.

---

7. Training the Machine Learning Model

Training an XGBoost Regression model

model = XGBRegressor()

model.fit(X_train, y_train)

- XGBRegressor() – Initializes the XGBoost model.
- fit(X_train, y_train) – Trains the model using training data.

---

8. Making Predictions

y_pred = model.predict(X_test)

- predict(X_test) – Predicts house prices for the test dataset.

---

9. Evaluating Model Performance

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

```
mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)

r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error: {mae}")

print(f"Mean Squared Error: {mse}")

print(f"Root Mean Squared Error: {rmse}")

print(f"R² Score: {r2}")
```

- **Mean Absolute Error (MAE):** Measures the average absolute difference between actual and predicted prices.
- **Mean Squared Error (MSE):** Measures the average squared difference.
- **Root Mean Squared Error (RMSE):** A lower value indicates better model performance.
- **R^2 Score:** Measures how well the model explains the variance in house prices.

Applications of the Notebook

This notebook is useful for **house price prediction** and other real estate applications:

- **Real Estate Pricing Models:** Predicting house prices based on location, size, and other features.
- **Property Valuation:** Estimating property value based on historical data.
- **Market Analysis:** Understanding trends in the housing market.
- **Investment Decision Making:** Helping investors make informed decisions.

Conclusion

The **House Price Prediction** notebook demonstrates how to solve a **regression problem** using **XGBoost**. The key steps include:

- **Loading and preprocessing data**
- **Exploratory Data Analysis (EDA)**
- **Feature engineering**
- **Training an XGBoost regression model**
- **Evaluating model performance using regression metrics**

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
import sklearn.datasets
from sklearn.model_selection import train_test_split
from sklearn.base import BaseEstimator, RegressorMixin
from xgboost import XGBRegressor
from sklearn import metrics
```

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

```python
print(house_price_dataset)
```

Python

```
{'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', '
```

```python
house_price_dataframe = pd.DataFrame(house_price_dataset.data, columns = house_price_dataset.feature_names)
```

Python

```python
house_price_dataframe.head()
```

Python

	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

```python
house_price_dataframe['price'] = house_price_dataset.target
```

Python

```python
house_price_dataframe.head()
```

[8] Python

	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

```python
house_price_dataframe.to_csv('House_price_prediction_Boston.csv', index=False)
```

[9] Python

```python
print("Number of rows are: ",house_price_dataframe.shape[0])
print("Number of columns are: ",house_price_dataframe.shape[1])
```

[10] Python

```
Number of rows are:  20640
Number of columns are:  9
```

```
house_price_dataframe.info()
```
[11]

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   MedInc      20640 non-null  float64
 1   HouseAge    20640 non-null  float64
 2   AveRooms    20640 non-null  float64
 3   AveBedrms   20640 non-null  float64
 4   Population  20640 non-null  float64
 5   AveOccup    20640 non-null  float64
 6   Latitude    20640 non-null  float64
 7   Longitude   20640 non-null  float64
 8   price       20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

```
house_price_dataframe.isnull().sum
```
[12]

```
<bound method DataFrame.sum of        MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0       False     False     False     False       False     False     False
1       False     False     False     False       False     False     False
2       False     False     False     False       False     False     False
3       False     False     False     False       False     False     False
4       False     False     False     False       False     False     False
...       ...       ...       ...       ...         ...       ...       ...
20635   False     False     False     False       False     False     False
20636   False     False     False     False       False     False     False
20637   False     False     False     False       False     False     False
20638   False     False     False     False       False     False     False
20639   False     False     False     False       False     False     False

       Longitude  price
0          False  False
1          False  False
2          False  False
3          False  False
4          False  False
...          ...    ...
20635      False  False
20636      False  False
20637      False  False
20638      False  False
20639      False  False

[20640 rows x 9 columns]>
```

```
[13] house_price_dataframe.columns
```

```
Index(['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup',
       'Latitude', 'Longitude', 'price'],
      dtype='object')
```

```
[14] house_price_dataframe.describe(include='all').round(2)
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	price
count	20640.00	20640.00	20640.00	20640.00	20640.00	20640.00	20640.00	20640.00	20640.00
mean	3.87	28.64	5.43	1.10	1425.48	3.07	35.63	-119.57	2.07
std	1.90	12.59	2.47	0.47	1132.46	10.39	2.14	2.00	1.15
min	0.50	1.00	0.85	0.33	3.00	0.69	32.54	-124.35	0.15
25%	2.56	18.00	4.44	1.01	787.00	2.43	33.93	-121.80	1.20
50%	3.53	29.00	5.23	1.05	1166.00	2.82	34.26	-118.49	1.80
75%	4.74	37.00	6.05	1.10	1725.00	3.28	37.71	-118.01	2.65
max	15.00	52.00	141.91	34.07	35682.00	1243.33	41.95	-114.31	5.00

```
[15] for i in house_price_dataframe.columns.tolist():
         print("No of unique Values in", i ,"is", house_price_dataframe[i].nunique())
```

```
No of unique Values in MedInc is 12928
No of unique Values in HouseAge is 52
No of unique Values in AveRooms is 19392
No of unique Values in AveBedrms is 14233
No of unique Values in Population is 3888
No of unique Values in AveOccup is 18841
No of unique Values in Latitude is 862
No of unique Values in Longitude is 844
No of unique Values in price is 3842
```

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

```python
print(X,Y)
```

[17]

```
       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
...
20637    0.923
20638    0.847
20639    0.894
Name: price, Length: 20640, dtype: float64
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.25, random_state=2)
```

[18]

```python
print(X.shape, X_train.shape, X_test.shape)
```

[19]

```
(20640, 8) (15480, 8) (5160, 8)
```

```python
model = XGBRegressor()
```

[20]

```python
model.fit(X_train, Y_train)
```

[21]

```
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, ...)
```

```
training_data_prediction = model.predict(X_train)
```
[22]

```
print(training_data_prediction)
```
[23]

```
[2.527071  0.6174013 1.1158092 ... 1.8347801 1.7563723 0.7643776]
```

```
score_1 = metrics.r2_score(Y_train, training_data_prediction)
score_2 = metrics.mean_absolute_error(Y_train, training_data_prediction)
print("R Squared Error: ", score_1)
print("Mean Absolute Error: ", score_2)
```
[24]

```
R Squared Error:  0.9492539821379308
Mean Absolute Error:  0.18398371071249137
```

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