ASSIGNMENT:MODULE3

Submitted by

Parvathy V P

HOUSE PRICE PREDICTION DOCUMENTATION

INTRODUCTION

House Price Prediction is a machine learning technique used to estimate the value of a property based on past housing data. It analyses features like location, size, number of rooms, and age of the house to understand how they influence price. By learning patterns from historical data, the model can predict the price of new properties accurately. This helps buyers, sellers, and real-estate companies make better and more informed decisions.

1.Import Required Libraries

```python
import numpy as np
import pandas as pd
```

2.Load the Dataset

```python
house = pd.read_csv("Pune house data.csv")
house.head()
```

	area_type	availability	size	society	total_sqft	bath	balcony	price	site_location
0	Super built-up Area	19-Dec	2 BHK	Coomee	1056	2.0	1.0	39.07	Alandi Road
1	Plot Area	Ready To Move	4 Bedroom	Theanmp	2600	5.0	3.0	120.00	Ambegaon Budruk
2	Built-up Area	Ready To Move	3 BHK	NaN	1440	2.0	3.0	62.00	Anandnagar
3	Super built-up Area	Ready To Move	3 BHK	Soiewre	1521	3.0	1.0	95.00	Aundh
4	Super built-up Area	Ready To Move	2 BHK	NaN	1200	2.0	1.0	51.00	Aundh Road

3.Remove Unnecessary columns

```python
house.drop(columns=["area_type", "availability", "society"], inplace=True)
```

```python
house.head()
```

size	total_sqft	bath	balcony	price	site_location
2 BHK	1056	2.0	1.0	39.07	Alandi Road
4 Bedroom	2600	5.0	3.0	120.00	Ambegaon Budruk
3 BHK	1440	2.0	3.0	62.00	Anandnagar
3 BHK	1521	3.0	1.0	95.00	Aundh
2 BHK	1200	2.0	1.0	51.00	Aundh Road

This removes the columns `area_type`, `availability`, and `society` from the house DataFrame. Using `inplace=True` updates the DataFrame directly without creating a new one. The columns `area_type`, `availability`, and `society` are removed because they usually do not provide useful or reliable information for predicting house prices. These columns may contain too many categories, irrelevant text, or inconsistent values. Removing such columns helps keep the dataset clean and improves the performance of the machine learning model.

4.Create a new “bhk” Column

```
house['bhk'] = house['size'].str.split().str.get(0).astype(int)
```

```
house.head()
```

	size	total_sqft	bath	balcony	price	site_location	bhk
0	2 BHK	1056	2.0	1.0	39.07	Alandi Road	2
1	4 Bedroom	2600	5.0	3.0	120.00	Ambegaon Budruk	4
2	3 BHK	1440	2.0	3.0	62.00	Anandnagar	3
3	3 BHK	1521	3.0	1.0	95.00	Aundh	3
4	2 BHK	1200	2.0	1.0	51.00	Aundh Road	2

This line extracts the numeric count(eg.(2,4)) from the `size` column (e.g., “2 BHK”,”4 BEDROOM”) by splitting the text, taking the first part, converting it to an integer, and storing it in a new column bhk.Then we can drop the size column

5.Remove Unrealistic “bhk” Values

```
house = house[house.bhk<=20]
```

```
house[house.bhk>20]
```

total_sqft	bath	balcony	price	site_location	bhk

Values greater than 20 BHK are unrealistic and are considered outliers. Removing them helps improve the quality of the dataset and model accuracy.

6.Convert “total_sqft” in to Numeric

```
house['total_sqft'].unique()
```

```
array(['1056', '2600', '1440', ..., '1133 - 1384', '774', '4689'],
      dtype=object)
```

```
def convertRange(x):
    temp = x.split("-")
    if len(temp)==2:
        return (float(temp[0]) + float(temp[1])) / 2
    try:
        return float(x)
    except:
        return None
```

```
house['total_sqft'] = house['total_sqft'].apply(convertRange)
```

```
house['total_sqft'].unique()
```

```
array([1056. , 2600. , 1440. , ..., 1258.5,  774. , 4689. ])
```

```
house['total_sqft'].isna().sum()
```

```
np.int64(42)
```

`convertRange()` cleans the `total_sqft` column by converting ranges (like 1133-1384) into their average and converting single numbers to float.If the value cannot be converted, it returns None.So invalid data can be removed.

Code explanation:

temp = x.split("-")

Splits the input string x at the - character.

Example: "1000-1200" → ["1000", "1200"].

If x has no dash: "1056" → ["1056"].

if len(temp) == 2:

return (float(temp[0]) + float(temp[1])) / 2

When the split produces exactly two parts, the function:

Converts both parts to float.

Computes their arithmetic mean (average).

Returns that average as a float.

Example: "1000-1200" → (1000.0 + 1200.0) / 2 → 1100.0.

try:

return float(x)

except:

return None

Attempts to convert the whole string x directly to a float.

Example: "1056" → 1056.0.

If conversion fails (because x contains text, units, or other characters), the except branch returns None.

7.Create “Price_Per_sq_ft” Column

```
house['price_per_sq_ft'] = house['price']*100000 / house['total_sqft']
house['price_per_sq_ft']
```

```
0         3699.810606
1         4615.384615
2         4305.555556
3         6245.890861
4         4250.000000
             ...
12661     6530.612245
12662     6689.834926
12663     5258.545136
12664    10407.336319
12665     3090.909091
Name: price_per_sq_ft, Length: 12666, dtype: float64
```

The `price_per_sq_ft` column is calculated by converting the house price to actual currency units (multiplying by 100,000 if prices are in lakhs) and dividing by the total square footage of the house. This standardizes the price relative to house size, allowing fair comparison across properties of different sizes. It helps identify over- or under-priced houses, analyze price trends across locations or time, and serves as a meaningful feature in predictive models for house prices.

8.Remove Unrealistic "sqft/ bhk"

```
house = house[((house['total_sqft'] / house['bhk']) >= 300)]
house.describe()
```

	level_0	total_sqft	bath	balcony	price	bhk	price_per_sq_ft
count	12013.000000	12013.000000	12013.000000	12013.000000	12013.000000	12013.000000	12013.000000
mean	6355.003996	1542.372588	2.511862	1.587780	105.007894	2.607342	6206.213093
std	3671.108293	1181.080695	1.006206	0.808746	134.204258	0.922976	3985.465773
min	0.000000	300.000000	1.000000	0.000000	9.000000	1.000000	267.829813
25%	3181.000000	1107.000000	2.000000	1.000000	48.450000	2.000000	4199.565960
50%	6362.000000	1285.000000	2.000000	2.000000	68.000000	2.000000	5253.456221
75%	9538.000000	1660.000000	3.000000	2.000000	110.000000	3.000000	6823.529412
max	12707.000000	52272.000000	13.000000	3.000000	2912.000000	13.000000	176470.588235

This line keeps only the houses where the average area per bedroom (`total_sqft ÷` bhk) is at least 300 sqft. Houses with less than 300 sqft per bedroom are likely unrealistic data or unusually small, so they are removed. This step helps ensure the dataset is cleaner and more reliable for analysis or modeling.

9.Outlier Removel Based on location

```
def remove_outlier_sqft(df):
    df_output = pd.DataFrame()
    for key, subdf in df.groupby('site_location'):
        m = np.mean(subdf.price_per_sq_ft)
        st = np.std(subdf.price_per_sq_ft)
        gen_df = subdf[(subdf.price_per_sq_ft > (m-st)) & (subdf.price_per_sq_ft <= (m+st))]
        df_output = pd.concat([df_output, gen_df], ignore_index=True)
    return df_output

house = remove_outlier_sqft(house)
house.describe()
```

	level_0	total_sqft	bath	balcony	price	bhk	price_per_sq_ft
count	10461.000000	10461.000000	10461.000000	10461.000000	10461.000000	10461.000000	10461.000000
mean	6330.446516	1450.477342	2.423095	1.594494	80.710099	2.529968	5341.303057
std	3679.475745	748.579590	0.900466	0.796253	58.718224	0.842005	1554.869125
min	0.000000	300.000000	1.000000	0.000000	10.000000	1.000000	2000.000000
25%	3145.000000	1100.000000	2.000000	1.000000	48.000000	2.000000	4181.184669
50%	6305.000000	1261.000000	2.000000	2.000000	65.000000	2.000000	5084.745763
75%	9531.000000	1600.000000	3.000000	2.000000	93.865000	3.000000	6278.260870
max	12705.000000	30400.000000	13.000000	3.000000	1824.000000	13.000000	17548.524329

This function is designed to remove outliers in house prices per square foot (`price_per_sq_ft`) for each location, ensuring a cleaner dataset for analysis.

Defines a function called `remove_outlier_sqft` that takes a DataFrame `df` as input.

Creates an empty DataFrame `df_output` that will store the filtered data after removing outliers.

Groups the data by `site_location` (each neighborhood or area).

key is the name of the location, and `subdf` is the subset of the DataFrame corresponding to that location.

This ensures that outliers are detected relative to houses in the same location, not the entire dataset.

Calculates the mean (`m`) and standard deviation (`st`) of `price_per_sq_ft` for houses in that location.

Mean gives the average price per square foot, and standard deviation measures how much the prices vary.

Filters the subset `subdf` to keep only houses within one standard deviation of the mean:

Lower limit: `m - st`
Upper limit: `m + st`

Houses priced much lower or much higher than the typical range in that location are considered outliers and are removed.

Adds the filtered subset `gen_df` to `df_output`.

`ignore_index=True` resets the index so the final DataFrame has continuous numbering.

Returns the final DataFrame `df_output` containing all locations with outliers removed.

10.Outlier Removal based on "bhk"

```
def bhk_outlier_remover(df):
    exclude_indices = np.array([])
    for location, location_df in df.groupby('site_location'):
        #print("Location=",location,location_df)
        bhk_stats = {}
        for bhk, bhk_df in location_df.groupby('bhk'):
            bhk_stats[bhk] = {
                'mean':np.mean(bhk_df.price_per_sq_ft),
                'std':np.std(bhk_df.price_per_sq_ft),
                'count':bhk_df.shape[0]
            }
        for bhk, bhk_df in location_df.groupby('bhk'):
            stats = bhk_stats.get(bhk-1)
            #print(stats)
            if stats and stats['count']>5:
                exclude_indices = np.append(exclude_indices, bhk_df[bhk_df.price_per_sq_ft<(stats['mean'])].index.values)
    return df.drop(exclude_indices, axis='index')
```

```
house = bhk_outlier_remover(house)
```

The `bhk_outlier_remover` function removes houses that are potential outliers based on BHK and price per square foot within each location. It ensures that a house with unusually low price per sqft compared to smaller houses (e.g., a 3 BHK cheaper per sqft than typical 2 BHKs in the same location) is excluded. This helps clean the dataset for more reliable analysis or modeling.

Defines the function `bhk_outlier_remover` which takes a DataFrame `df`.

Creates an empty NumPy array `exclude_indices` to store row indices of houses to remove.

Groups the DataFrame by `site_location` (neighborhood/area).

Iterates over each location:

> **`location` = name of the site.**
>
> **`location_df` = subset of houses in that location.**

Initializes an empty dictionary `bhk_stats` to store statistics for each BHK type in the location.

Groups the location data by bhk (number of bedrooms).

For each BHK type:

`'mean'` → average `price_per_sq_ft` for that BHK.

`'std'` → standard deviation of `price_per_sq_ft`.

`'count'` → number of houses for that BHK in the location.

Iterates again over each BHK type in the location.

Retrieves statistics of previous BHK (`bhk-1`) from `bhk_stats`.

Purpose: Compare current BHK with the smaller BHK to detect anomalies.

Checks two conditions:

`stats` exists (there is a smaller BHK to compare).

There are at least 5 houses of the smaller BHK (`stats['count'] > 5`) to have meaningful comparison.

For houses in the current BHK:

If `price_per_sq_ft` is less than the mean of the smaller BHK, consider it an outlier.

Add its index to `exclude_indices`.

11.Save the File

```
house.to_csv('cleaned_data.csv')
```

12.Import Necessary Tools

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score
from sklearn.preprocessing import OneHotEncoder, StandardScaler
```

These imports bring in the necessary tools for building and evaluating machine learning models. train_test_split splits the data, LinearRegression, Lasso, and Ridge are regression algorithms, make_column_transformer and OneHotEncoder handle categorical features, StandardScaler scales numerical features, make_pipeline chains preprocessing and modeling steps, and r2_score evaluates model performance.

13.splitting to train and test sets

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=1)
```

This step splits the dataset into training and testing sets. 80% of the data is used to train the model (x_train, y_train) and 20% is kept aside to evaluate its performance (x_test, y_test). Using random_state=1 ensures that the split is reproducible.

14.One Hot Encoding “Site_location” Cloumn

```
column_trans = make_column_transformer((OneHotEncoder(sparse_output=False), ['site_location']), remainder='passthrough')

print(column_trans)

ColumnTransformer(remainder='passthrough',
                  transformers=[('onehotencoder',
                                 OneHotEncoder(sparse_output=False),
                                 ['site_location'])])
```

Creates a ColumnTransformer, which tells scikit-learn:

Convert the `site_location` column into multiple encoded columns, and leave all other columns unchanged. The sparse_output=False makes the output easier to work with. The remainder='passthrough' keeps all other columns unchanged.

15.Pipeline:Scaling and Linear Regression

```
scaler = StandardScaler()
lr = LinearRegression()


pipe = make_pipeline(column_trans, scaler, lr)
```

This code builds a complete machine-learning workflow by creating a pipeline that automatically transforms the data and then fits a regression model. First, `scaler = StandardScaler()` creates a scaler that normalizes all numerical features so they have mean 0 and standard deviation 1, ensuring that no feature dominates due to larger values and helping Linear Regression perform more efficiently. Next, `lr = LinearRegression()` initializes the Linear Regression model that will learn the relationship between the input features and the target variable (house price). Finally, `pipe = make_pipeline(column_trans, scaler, lr)` combines everything—`column_trans` (which one-hot encodes the categorical column `site_location` and passes all other columns unchanged), the scaler (which standardizes the entire transformed dataset), and the regression model—into a single pipeline. This pipeline ensures that whenever you train or predict, the same preprocessing steps (encoding + scaling) are applied in the correct order automatically, preventing mistakes, improving consistency, and producing a clean, professional machine-learning workflow.

16.Training the Pipeline

```
pipe.fit(x_train, y_train)
```

When we run `pipe.fit(x_train, y_train)`, the pipeline trains itself by applying each step in order. First, the ColumnTransformer learns the categories for one-hot encoding and converts all features into a numeric format. Then the StandardScaler learns the mean and standard deviation of each feature and scales the data. Finally, the LinearRegression model trains on this cleaned, scaled data and learns the coefficients and intercept. All learned settings—encoder categories, scaling values, and model weights—are saved inside the pipeline, ensuring that future predictions use the exact same preprocessing and model logic.

17.Make Predictions and Model Evaluation

```python
y_pred_lr = pipe.predict(x_test)
```

```python
r2_score(y_test, y_pred_lr)
```

```
0.8405184063558485
```

The line `y_pred_lr = pipe.predict(x_test)` uses the trained pipeline to generate predictions for the test dataset, applying the same preprocessing steps—such as encoding and scaling—that were learned during training. After obtaining these predictions, `r2_score(y_test, y_pred_lr)` evaluates the model's accuracy by comparing them with the actual target values. The resulting R^2 score of about 0.84 shows that the model explains roughly 84% of the variation in the test data, indicating strong overall performance.

18.Predicting House Price for new input data

```python
data = [2345, 2, 2, 'Wagholi', 2]
data = pd.Series(data)

type(data)
data = data.values.reshape(1,5)
```

```python
df = pd.DataFrame(data, columns=['total_sqft', 'bath', 'balcony', 'site_location', 'bhk'])
df
```

	total_sqft	bath	balcony	site_location	bhk
0	2345	2	2	Wagholi	2

```python
prediction = pipe.predict(df)
print("Price for your house should be: ", prediction[0]*100000)
```

```
Price for your house should be:  14511558.749751829
```

In this code, a list data containing house features—`total_sqft`, `bath`, `balcony`, `site_location`, and bhk—is first converted into a Pandas Series and then reshaped into a 1×5 array to match the expected input format for the model. This array is used to create a DataFrame `df` with appropriate column names, representing a single house's details. The DataFrame is then passed to a pre-trained pipeline `pipe` which applies all necessary preprocessing and prediction steps. Finally, the predicted price is extracted from the output array, scaled by 100,000, and printed as the estimated house price.

19.Save the Model

```python
import pickle

pickle.dump(pipe, open('LinearModel.pkl', 'wb'))
```

This code saves the trained `pipe` (pipeline or model) to a file named `LinearModel.pkl` using Python's `pickle` module. The `'wb'` mode opens the file in write-binary mode, allowing the model to be stored for later use without retraining.