
# PREDICTIVE ANALYTICS PROJECT

TOPIC:

Paris Housing Price Prediction

SUBMITTED BY:

3147 ROHAN UTTAM KHAPANE

Mahatma Education Society's
Pillai College of Arts, Commerce & Science
(Autonomous)
Affiliated to University of Mumbai
NAAC Accredited 'A' grade (3 cycles)
Best College Award by University of Mumbai
ISO 9001:2015 Certified

MAHATMA EDUCATION SOCIETY'S

PILLAI COLLEGE OF ARTS, COMMERCE & SCIENCE

(Autonomous)

NEW PANVEL

PROJECT REPORT ON

# "Paris Housing Price Prediction"

IN PARTIAL FULFILLMENT OF

MASTERS OF DATA ANALYTICS

SEMESTER 1– 2023-24

PROJECT GUIDE

Prof. Sanjana Bhangale

SUBMITTED BY: Rohan Uttam Khapane

ROLL NO: 3147

## Evaluation sheet for continuous assessment with rubrics

## Class: Data Analytics

## Subject: Predictive Analytics

Details about the continuous Assessment 2/Project work

Name of the Student : ROHAN UTTAM KHAPANE

Roll Number : 3147

Class / Division : Msc DA Part 1

Name of Evaluator:

Please circle appropriate score

| Grading Criteria | Fair | Good | Excellent | Total |
|---|---|---|---|---|
| Introduction/ Description of the Case<br>Mahatma Education Society's<br>Pillai College of Arts, Commerce & Science<br>(Autonomous)<br>Affiliated to University of Mumbai<br>NAAC Accredited 'A' grade (3 cycles)<br>Best College Award by University of Mumbai<br>ISO 9001:2015 Certified | 1 | 2 | 3 | /3 |
| Predictive Analysis of the Spotify company used for case analysis pertaining to the case (**strength** of CA2 topic: e.g main important feature of CA1, **Weakness:** limitations of the project,<br>**Opportunities:** in carrier in the future,<br>**Threat:** obstacles that can cause failure to project CA2 | 3 | 4 | 5 | /5 |
| Learnings from the case | 2 | 3 | 4 | /4 |
| Delivery/presentation skills | 1 | 2 | 3 | /3 |
| Total | | | | /15 |

# CA-II Project

## Paris Housing Price Prediction

The dataset tells us about the price of houses in Paris and various attribute affecting the prices of the house.

The dataset contains 17 columns and 10000 row entries in each column . As it is pre-processed data ,hence it has no null values.

**The dataset is taken from Kaggle.com hence it may contain falsy values and the prediction may vary as compared to real life situations in ML prediction.**

Here , the **target variable** can be considered as **Price and Category.**

**Dataset link:**
https://www.kaggle.com/datasets/mssmartypants/paris-housing-price-prediction

Using the above dataset will be performing all the Life Cycle phases of Data Science , i.e , Data Understanding, Data Preparation ,Data Visualization(EDA), Data Modelling, Model Evaluation.

Business Understanding and Model Deployment are also the phases , but as this in not an industry oriented project ,hence it is not included above.

The Below table specifies the name of the columns, their data types , the feature is categorical or numerical and their description

| Column Name | Data Types | Categorical / Numerical Values | Description |
|---|---|---|---|
| squareMeters | Integer | Numerical | Area of the house in square meters |
| numberOfRooms | Integer | Numerical | Number of rooms |
| hasYard | Category | Categorical | Whether the house has yard or not |
| hasPool | Category | Categorical | Whether the house has Pool or not |
| floors | Integer | Numerical | Number of floors |
| cityCode | Integer | Numerical | Zip code of city |
| cityPartRange | Category | Categorical | Type of locality nearby |
| numPrevOwners | Integer | Numerical | Number of previous owners |
| made | Integer | Numerical | Year the house was built |
| isNewBuilt | Category | Categorical | Whether the house is Newly Built or not |
| hasStormProtector | Category | Categorical | Whether the house has Strom Protector or not |
| basement | Integer | Numerical | Basement of house in square meters |
| attic | Integer | Numerical | Rooftop of house in square meters |
| garage | Integer | Numerical | Size of garage in  the house |
| hasStorageRoom | Category | Categorical | Whether the house has storage room or not |
| hasGuestRoom | Category | Categorical | Number of guest rooms |
| price | Float | Numerical | The price of the house |
| category | Category | Categorical | Whether the house is Luxury or Basic |

# Code and Output:

## Importing required in-built libraries and packages

```
[2] import seaborn as sns
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LinearRegression
    from sklearn.metrics import mean_squared_error, r2_score,mean_absolute_error
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

## Uploading file using google.colab package using pandas library to read the .csv file

```
[3] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

[4] path="/content/drive/My Drive/Colab Notebooks/ParisHousingClass.csv"

    df=pd.read_csv(path)
    df
```

## Data Analysis:

```
df.head()
```

| | squareMeters | numberOfRooms | hasYard | hasPool | floors | cityCode | cityPartRange | numPrevOwners | made | isNewBuilt | hasStormProtector | basement | attic |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 75523 | 3 | 0 | 1 | 63 | 9373 | 3 | 8 | 2005 | 0 | 1 | 4313 | 9005 |
| 1 | 80771 | 39 | 1 | 1 | 98 | 39381 | 8 | 6 | 2015 | 1 | 0 | 3653 | 2436 |
| 2 | 55712 | 58 | 0 | 1 | 19 | 34457 | 6 | 8 | 2021 | 0 | 0 | 2937 | 8852 |
| 3 | 32316 | 47 | 0 | 0 | 6 | 27939 | 10 | 4 | 2012 | 0 | 1 | 659 | 7141 |
| 4 | 70429 | 19 | 1 | 1 | 90 | 38045 | 3 | 7 | 1990 | 1 | 0 | 8435 | 2429 |

```
[7] df.tail()
```

| | squareMeters | numberOfRooms | hasYard | hasPool | floors | cityCode | cityPartRange | numPrevOwners | made | isNewBuilt | hasStormProtector | basement | atti |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 9995 | 1726 | 89 | 0 | 1 | 5 | 73133 | 7 | 6 | 2009 | 0 | 1 | 9311 | 169 |
| 9996 | 44403 | 29 | 1 | 1 | 12 | 34606 | 9 | 4 | 1990 | 0 | 1 | 9061 | 174 |
| 9997 | 83841 | 3 | 0 | 0 | 69 | 80933 | 10 | 10 | 2005 | 1 | 1 | 8304 | 773 |
| 9998 | 59036 | 70 | 0 | 0 | 96 | 55856 | 1 | 3 | 2010 | 0 | 1 | 2590 | 617 |
| 9999 | 1440 | 84 | 0 | 0 | 49 | 18412 | 6 | 10 | 1994 | 1 | 0 | 8485 | 202 |

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   squareMeters       10000 non-null  int64
 1   numberOfRooms      10000 non-null  int64
 2   hasYard            10000 non-null  int64
 3   hasPool            10000 non-null  int64
 4   floors             10000 non-null  int64
 5   cityCode           10000 non-null  int64
 6   cityPartRange      10000 non-null  int64
 7   numPrevOwners      10000 non-null  int64
 8   made               10000 non-null  int64
 9   isNewBuilt         10000 non-null  int64
 10  hasStormProtector  10000 non-null  int64
 11  basement           10000 non-null  int64
 12  attic              10000 non-null  int64
 13  garage             10000 non-null  int64
 14  hasStorageRoom     10000 non-null  int64
 15  hasGuestRoom       10000 non-null  int64
 16  price              10000 non-null  float64
 17  category           10000 non-null  object
dtypes: float64(1), int64(16), object(1)
memory usage: 1.4+ MB
```

## Shape of dataset :

```
df.shape
```

```
(10000, 18)
```

## Convert few column to categorical as they are wrongly present

Loading the dataset in df1 as a dataframe

```
[12] df1=pd.DataFrame(df)
     df1
```

| | squareMeters | numberOfRooms | hasYard | hasPool | floors | cityCode | cityPartRange | numPrevOwners | made | isNewBuilt | hasStormProtector | basement |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 75523 | 3 | 0 | 1 | 63 | 9373 | 3 | 8 | 2005 | 0 | 1 | 4313 |
| **1** | 80771 | 39 | 1 | 1 | 98 | 39381 | 8 | 6 | 2015 | 1 | 0 | 3653 |
| **2** | 55712 | 58 | 0 | 1 | 19 | 34457 | 6 | 8 | 2021 | 0 | 0 | 2937 |
| **3** | 32316 | 47 | 0 | 0 | 6 | 27939 | 10 | 4 | 2012 | 0 | 1 | 659 |
| **4** | 70429 | 19 | 1 | 1 | 90 | 38045 | 3 | 7 | 1990 | 1 | 0 | 8435 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **9995** | 1726 | 89 | 0 | 1 | 5 | 73133 | 7 | 6 | 2009 | 0 | 1 | 9311 |
| **9996** | 44403 | 29 | 1 | 1 | 12 | 34606 | 9 | 4 | 1990 | 0 | 1 | 9061 |
| **9997** | 83841 | 3 | 0 | 0 | 69 | 80933 | 10 | 10 | 2005 | 1 | 1 | 8304 |
| **9998** | 59036 | 70 | 0 | 0 | 96 | 55856 | 1 | 3 | 2010 | 0 | 1 | 2590 |
| **9999** | 1440 | 84 | 0 | 0 | 49 | 18412 | 6 | 10 | 1994 | 1 | 0 | 8485 |

10000 rows × 18 columns

Analyse df1 by head(),tail() and shape() function

```
[13] df1.head()
```

| | squareMeters | numberOfRooms | hasYard | hasPool | floors | cityCode | cityPartRange | numPrevOwners | made | isNewBuilt | hasStormProtector | basement |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 75523 | 3 | 0 | 1 | 63 | 9373 | 3 | 8 | 2005 | 0 | 1 | 4313 |
| **1** | 80771 | 39 | 1 | 1 | 98 | 39381 | 8 | 6 | 2015 | 1 | 0 | 3653 |
| **2** | 55712 | 58 | 0 | 1 | 19 | 34457 | 6 | 8 | 2021 | 0 | 0 | 2937 |
| **3** | 32316 | 47 | 0 | 0 | 6 | 27939 | 10 | 4 | 2012 | 0 | 1 | 659 |
| **4** | 70429 | 19 | 1 | 1 | 90 | 38045 | 3 | 7 | 1990 | 1 | 0 | 8435 |

```
df1.tail()
```

| | squareMeters | numberOfRooms | hasYard | hasPool | floors | cityCode | cityPartRange | numPrevOwners | made | isNewBuilt | hasStormProtector | basement |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **9995** | 1726 | 89 | 0 | 1 | 5 | 73133 | 7 | 6 | 2009 | 0 | 1 | 9311 |
| **9996** | 44403 | 29 | 1 | 1 | 12 | 34606 | 9 | 4 | 1990 | 0 | 1 | 9061 |
| **9997** | 83841 | 3 | 0 | 0 | 69 | 80933 | 10 | 10 | 2005 | 1 | 1 | 8304 |
| **9998** | 59036 | 70 | 0 | 0 | 96 | 55856 | 1 | 3 | 2010 | 0 | 1 | 2590 |
| **9999** | 1440 | 84 | 0 | 0 | 49 | 18412 | 6 | 10 | 1994 | 1 | 0 | 8485 |

```
df1.shape
(10000, 18)

[16] df1.size
180000
```

Check for Null values in dataset

```
[83] df1.isnull().sum()

squareMeters         0
numberOfRooms        0
hasYard              0
hasPool              0
floors               0
cityCode             0
cityPartRange        0
numPrevOwners        0
made                 0
isNewBuilt           0
hasStormProtector    0
basement             0
attic                0
garage               0
hasStorageRoom       0
hasGuestRoom         0
price                0
category             0
dtype: int64
```

Converting the datatype of few variables into categories

```
[84] df1['hasYard'].value_counts()
     df1['hasYard']=df1['hasYard'].astype('category')
     d=df1['hasYard'].dtype
     d

     CategoricalDtype(categories=[0, 1], ordered=False)

     df1['hasPool'].value_counts()
     df1['hasPool']=df1['hasPool'].astype('category')
     d1=df1['hasPool'].dtype
     d1

     CategoricalDtype(categories=[0, 1], ordered=False)

[86] df1['hasStorageRoom'].value_counts()
     df1['hasStorageRoom']=df1['hasStorageRoom'].astype('category')
     d2=df1['hasStorageRoom'].dtype
     d2

     CategoricalDtype(categories=[0, 1], ordered=False)

[87] df1['hasGuestRoom'].value_counts()
     df1['hasGuestRoom']=df1['hasGuestRoom'].astype('category')
     d3=df1['hasGuestRoom'].dtype
     d3

     CategoricalDtype(categories=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], ordered=False)
```

```
     df1['isNewBuilt'].value_counts()
     df1['isNewBuilt']=df1['isNewBuilt'].astype('category')
     d4=df1['isNewBuilt'].dtype
     d4

     CategoricalDtype(categories=[0, 1], ordered=False)

[89] df1['hasStormProtector']
     df1['hasStormProtector']=df1['hasStormProtector'].astype('category')
     d5=df1['hasStormProtector'].dtype
     d5

     CategoricalDtype(categories=[0, 1], ordered=False)

[90] df1['cityPartRange'].value_counts()
     df1['cityPartRange']=df1['cityPartRange'].replace({1: 1, 2: 2,3:3,4:4,5:5,6:6,7:7,8:8,9:9,10:10})
     df1['cityPartRange']=df1['cityPartRange'].astype('category')
     df1['cityPartRange'].dtype

     CategoricalDtype(categories=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10], ordered=False)

[91] df1['category'].value_counts()
     df1['category']=df1['category'].replace({'Luxury': 0, 'Basic': 1})
     df1['category']=df1['category'].astype('category')
     df1['category'].dtype

     CategoricalDtype(categories=[0, 1], ordered=False)
```

# Analysing the data type of variable after conversion

```
[92] df1.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   squareMeters       10000 non-null  int64
 1   numberOfRooms      10000 non-null  int64
 2   hasYard            10000 non-null  category
 3   hasPool            10000 non-null  category
 4   floors             10000 non-null  int64
 5   cityCode           10000 non-null  int64
 6   cityPartRange      10000 non-null  category
 7   numPrevOwners      10000 non-null  int64
 8   made               10000 non-null  int64
 9   isNewBuilt         10000 non-null  category
 10  hasStormProtector  10000 non-null  category
 11  basement           10000 non-null  int64
 12  attic              10000 non-null  int64
 13  garage             10000 non-null  int64
 14  hasStorageRoom     10000 non-null  category
 15  hasGuestRoom       10000 non-null  category
 16  price              10000 non-null  float64
 17  category           10000 non-null  category
dtypes: category(8), float64(1), int64(9)
memory usage: 861.0 KB
```

# Statistical summary of numeric variables

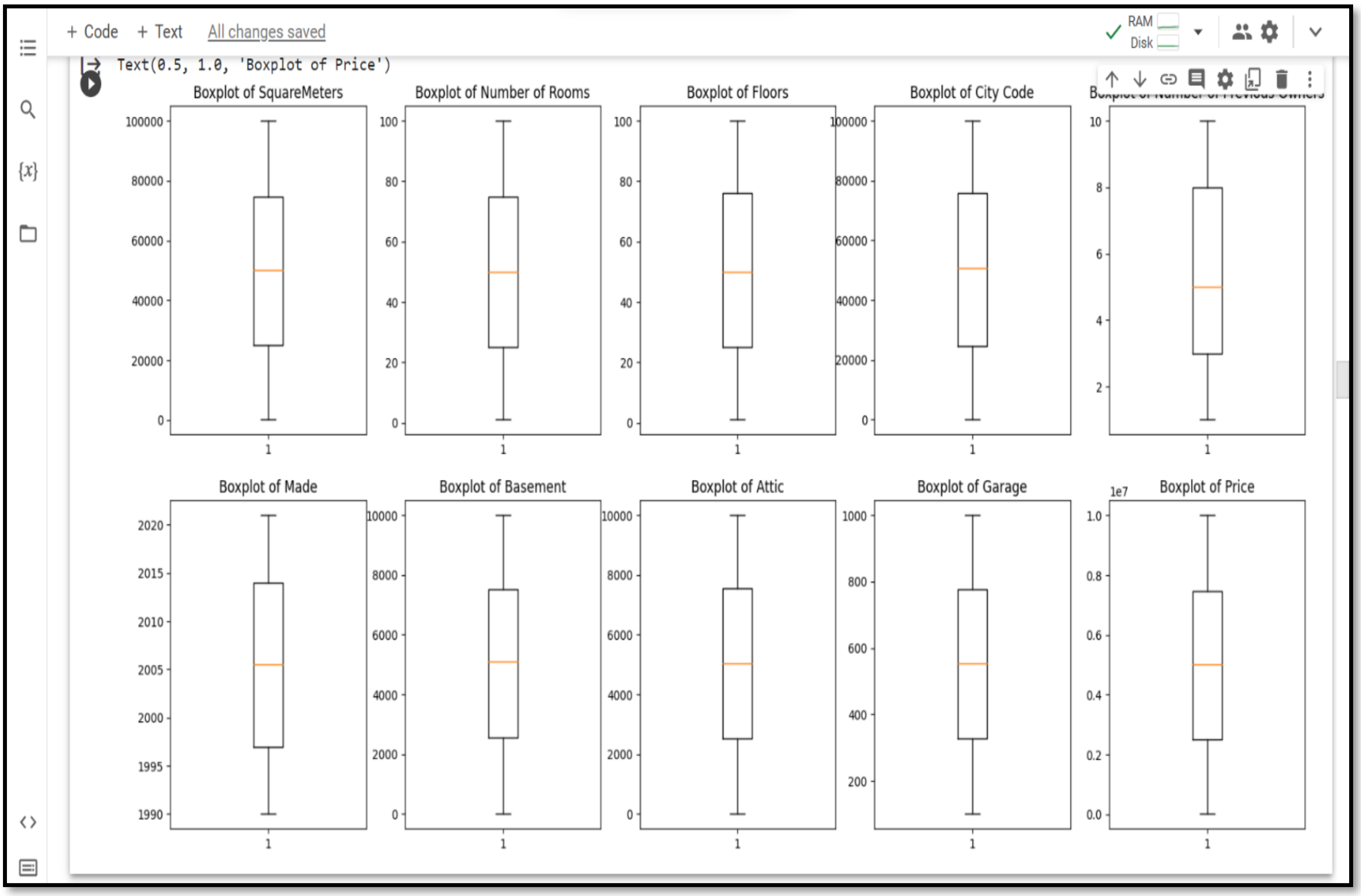
```
[93] df1.describe()
```

| | squareMeters | numberOfRooms | floors | cityCode | numPrevOwners | made | basement | attic | garage | price |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 10000.00000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.00000 | 10000.000000 | 10000.00000 | 10000.00000 | 1.000000e+04 |
| **mean** | 49870.13120 | 50.358400 | 50.276300 | 50225.486100 | 5.521700 | 2005.48850 | 5033.103900 | 5028.01060 | 553.12120 | 4.993448e+06 |
| **std** | 28774.37535 | 28.816696 | 28.889171 | 29006.675799 | 2.856667 | 9.30809 | 2876.729545 | 2894.33221 | 262.05017 | 2.877424e+06 |
| **min** | 89.00000 | 1.000000 | 1.000000 | 3.000000 | 1.000000 | 1990.00000 | 0.000000 | 1.00000 | 100.00000 | 1.031350e+04 |
| **25%** | 25098.50000 | 25.000000 | 25.000000 | 24693.750000 | 3.000000 | 1997.00000 | 2559.750000 | 2512.00000 | 327.75000 | 2.516402e+06 |
| **50%** | 50105.50000 | 50.000000 | 50.000000 | 50693.000000 | 5.000000 | 2005.50000 | 5092.500000 | 5045.00000 | 554.00000 | 5.016180e+06 |
| **75%** | 74609.75000 | 75.000000 | 76.000000 | 75683.250000 | 8.000000 | 2014.00000 | 7511.250000 | 7540.50000 | 777.25000 | 7.469092e+06 |
| **max** | 99999.00000 | 100.000000 | 100.000000 | 99953.000000 | 10.000000 | 2021.00000 | 10000.000000 | 10000.00000 | 1000.00000 | 1.000677e+07 |

## Outlier Detection:

```
plt.figure(figsize=(20,10))
plt.subplot(2,5,1)
plt.boxplot(df1['squareMeters'])
plt.title('Boxplot of SquareMeters')

plt.subplot(2,5,2)
plt.boxplot(df1['numberOfRooms'])
plt.title('Boxplot of Number of Rooms')

plt.subplot(2,5,3)
plt.boxplot(df1['floors'])
plt.title('Boxplot of Floors')

plt.subplot(2,5,4)
plt.boxplot(df1['cityCode'])
plt.title('Boxplot of City Code')

plt.subplot(2,5,5)
plt.boxplot(df1['numPrevOwners'])
plt.title('Boxplot of Number of Previous Owners')

plt.subplot(2,5,6)
plt.boxplot(df1['made'])
plt.title('Boxplot of Made')

plt.subplot(2,5,7)
plt.boxplot(df1['basement'])
plt.title('Boxplot of Basement')
```

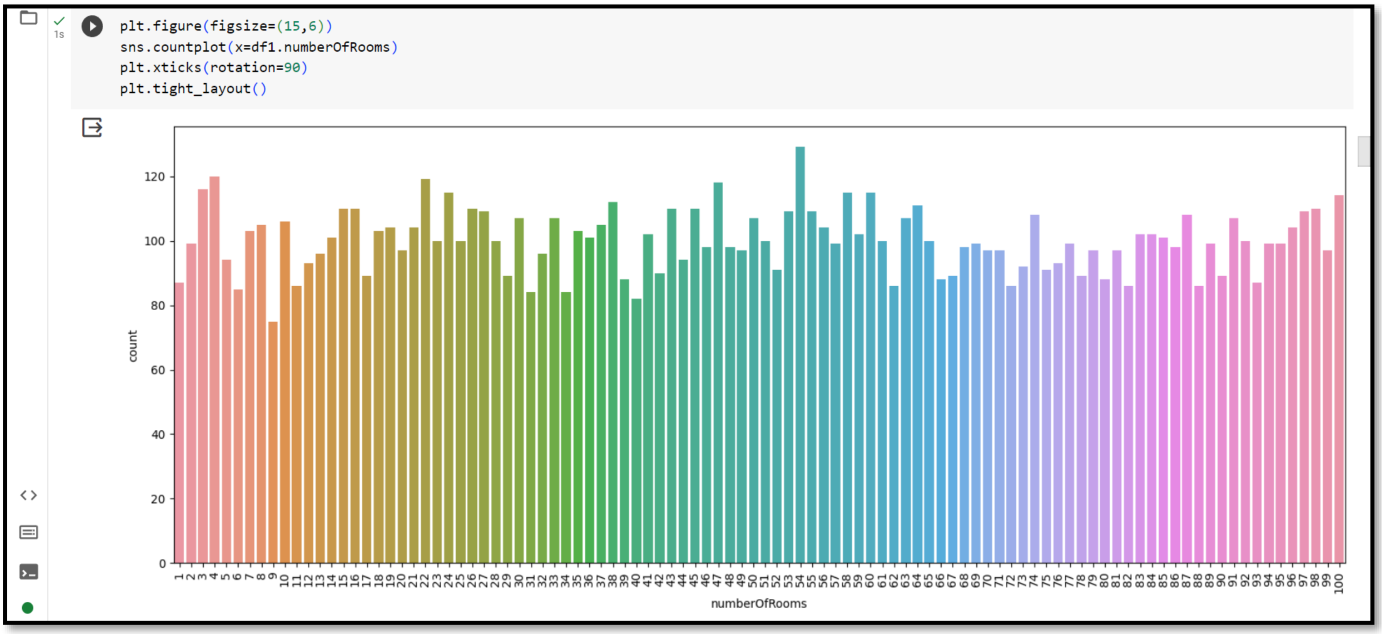
```
plt.subplot(2,5,8)
plt.boxplot(df1['attic'])
plt.title('Boxplot of Attic')

plt.subplot(2,5,9)
plt.boxplot(df1['garage'])
plt.title('Boxplot of Garage')

plt.subplot(2,5,10)
plt.boxplot(df1['price'])
plt.title('Boxplot of Price')
```




No outlier are present in any of the column

**EDA for categorical varaibles:**

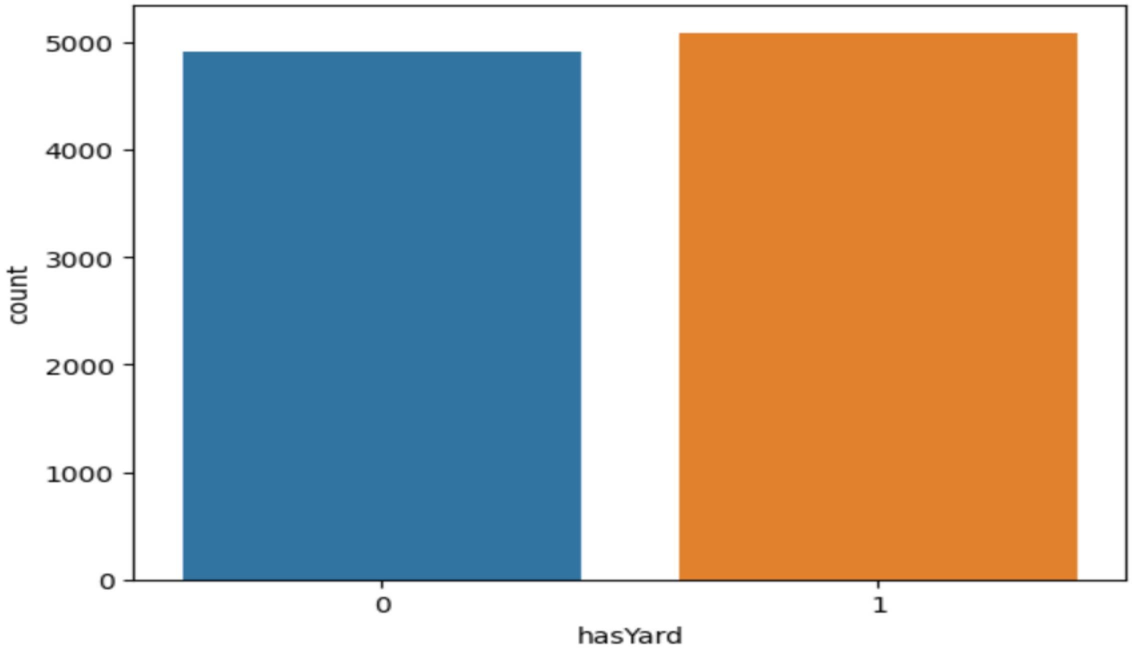
Countplot of number of rooms



Countplot of has yard:

```
sns.countplot(x=df1.hasYard)
```

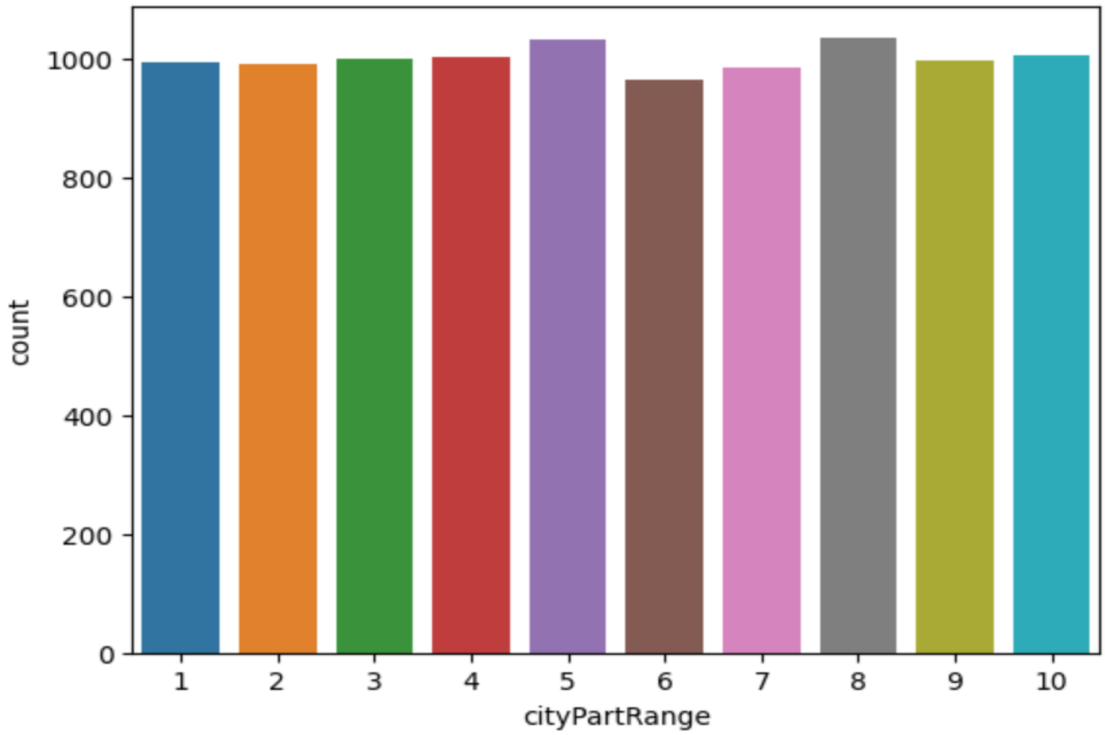
<Axes: xlabel='hasYard', ylabel='count'>




Countplot of cityPartRange :

```
sns.countplot(x=df1.cityPartRange)
```

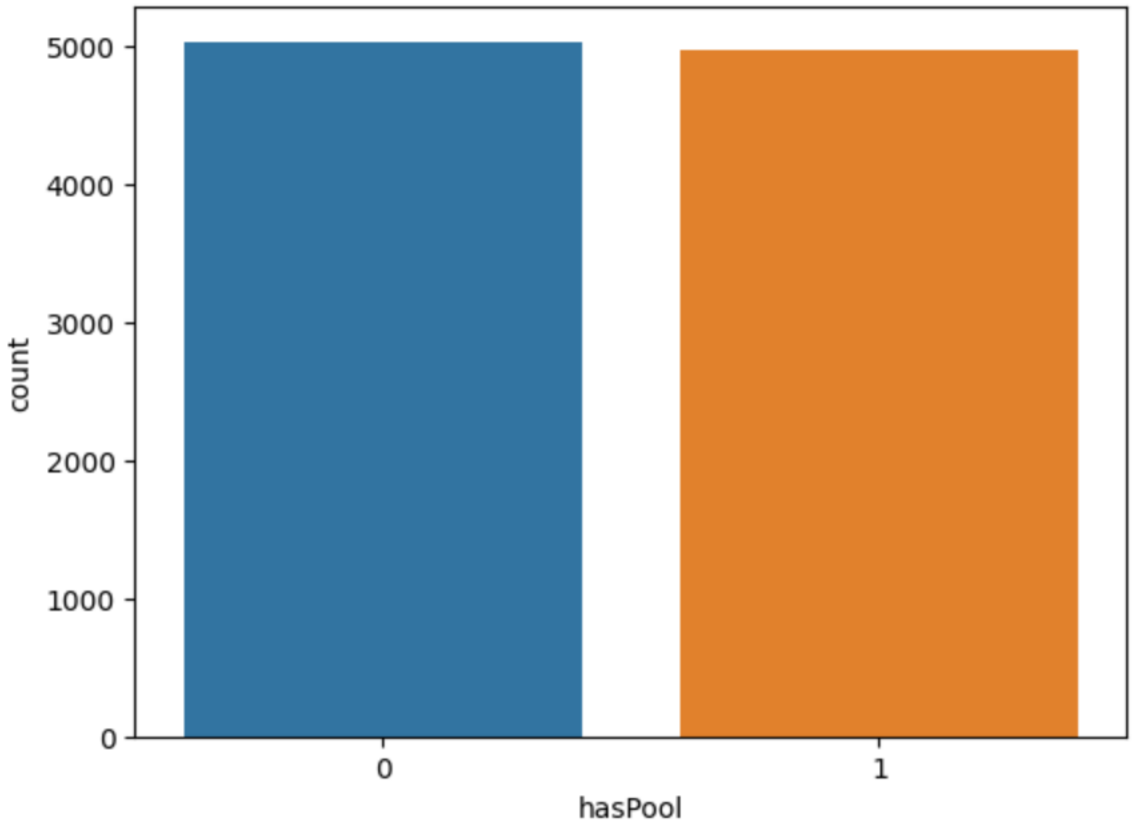
<Axes: xlabel='cityPartRange', ylabel='count'>



Countplot of hasPool:

```
sns.countplot(x=df1.hasPool)
```

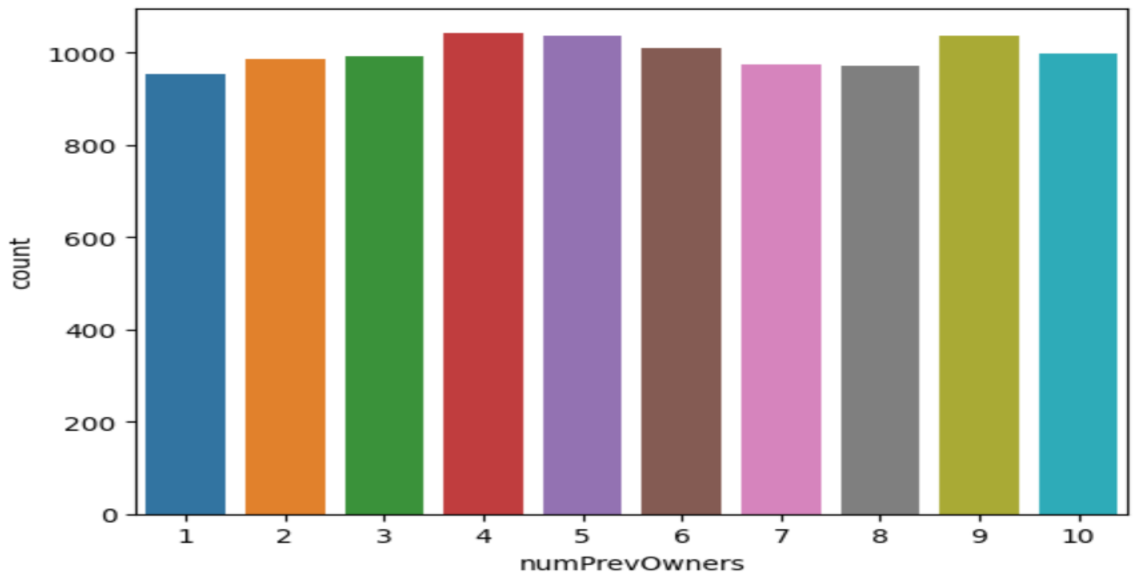
```
<Axes: xlabel='hasPool', ylabel='count'>
```




Countplot of numPrevOwners:

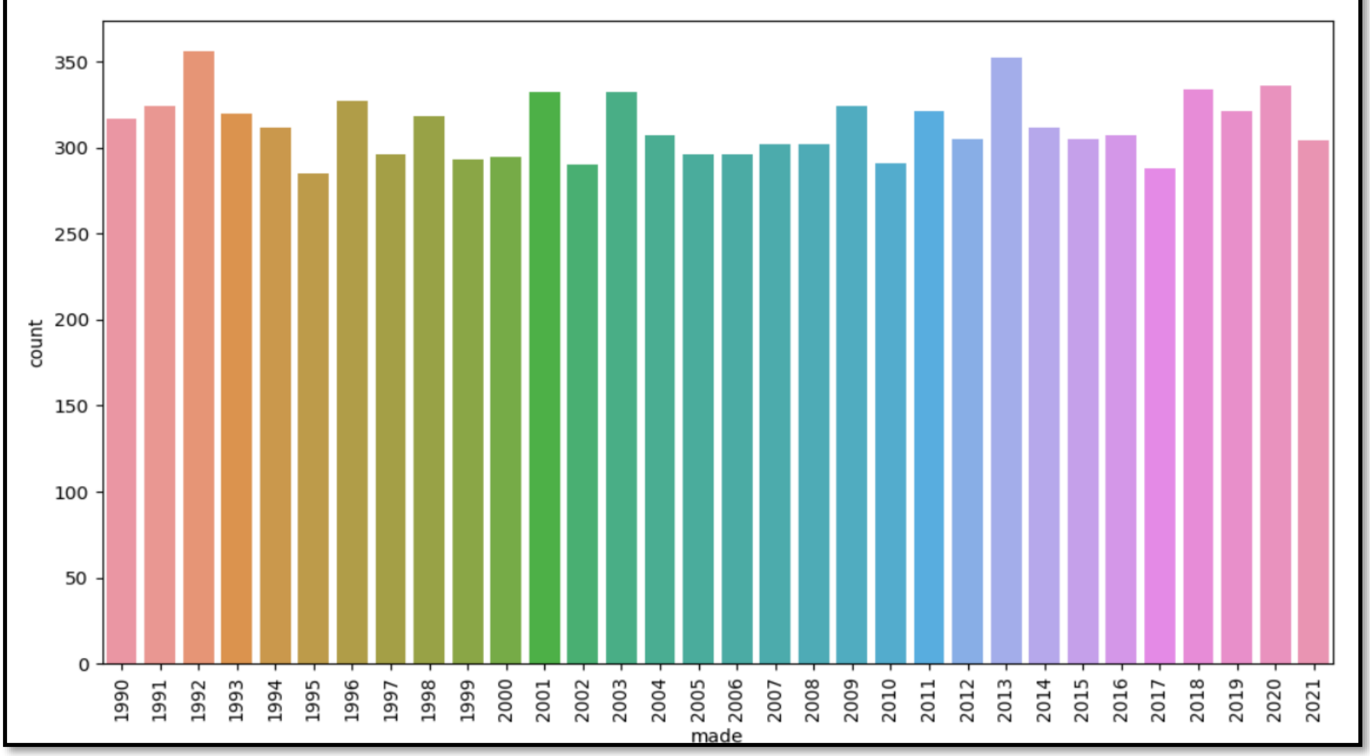
```
df1['numPrevOwners'].value_counts()
sns.countplot(x=df1.numPrevOwners)
```

```
<Axes: xlabel='numPrevOwners', ylabel='count'>
```



## Countplot of made:

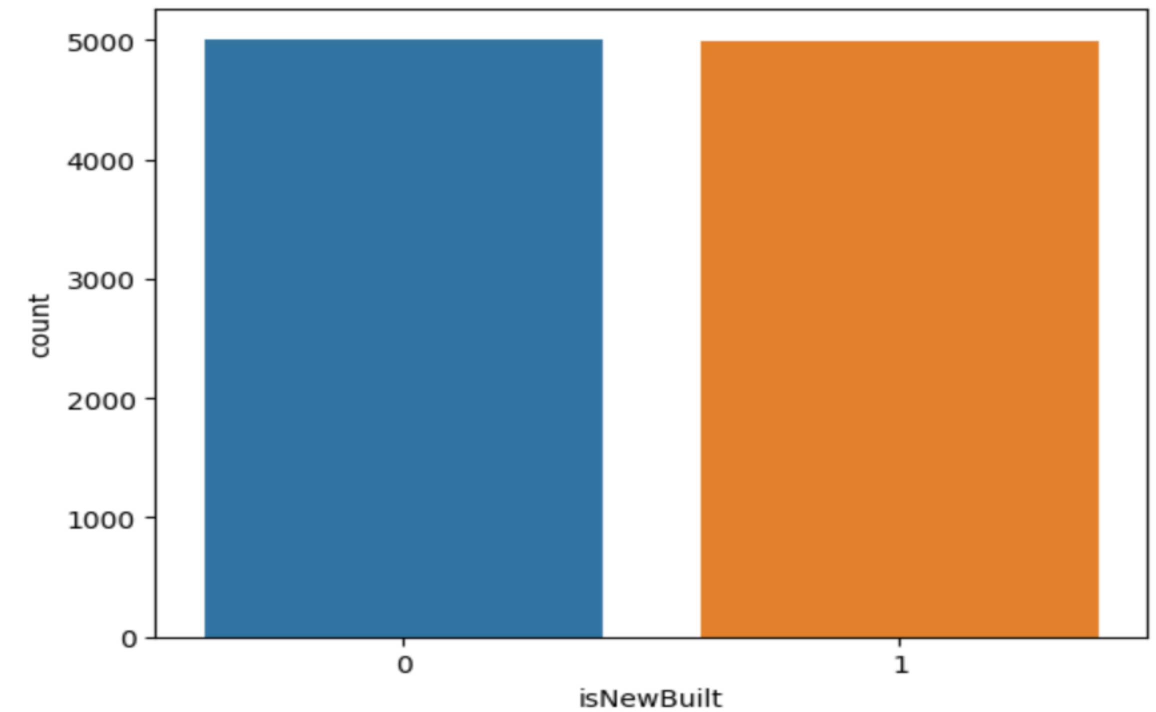
```
df1['made'].value_counts()
plt.figure(figsize=(10,6))
sns.countplot(x=df1.made)
plt.xticks(rotation=90)
plt.tight_layout()
```




## Countplot of isNewBuilt:

```
sns.countplot(x=df1.isNewBuilt)
```

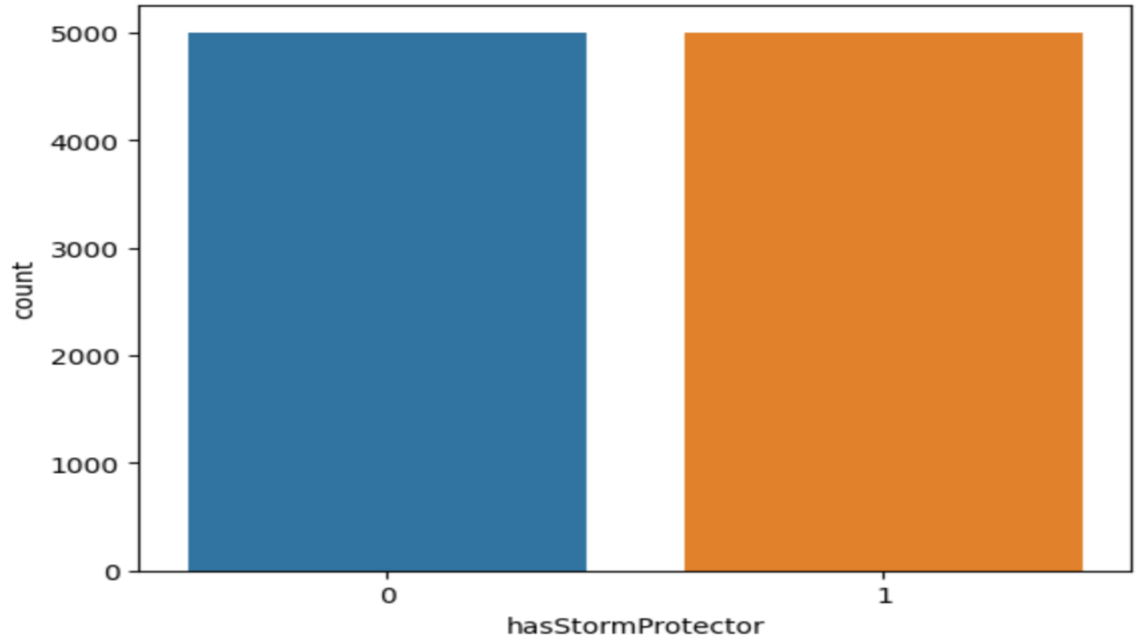
```
<Axes: xlabel='isNewBuilt', ylabel='count'>
```



Countplot of hasStromProtector:

```
sns.countplot(x=df1.hasStormProtector)
```

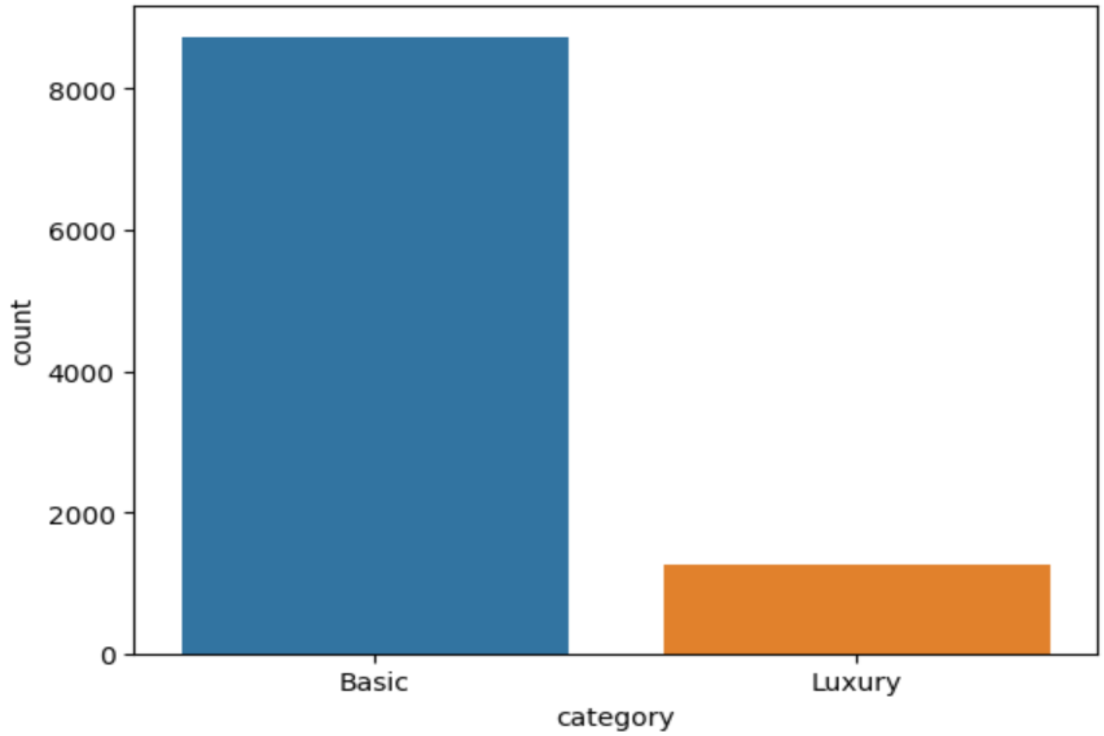
```
<Axes: xlabel='hasStormProtector', ylabel='count'>
```




Countplot of category

```
sns.countplot(x=df.category)
```

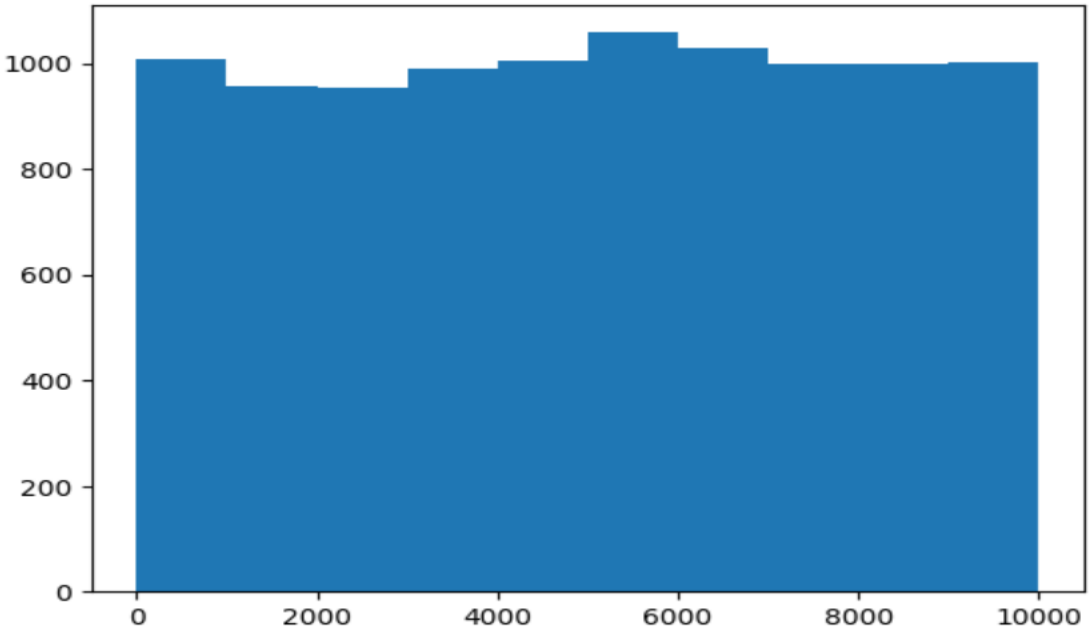
```
<Axes: xlabel='category', ylabel='count'>
```



Histogram of basement:

```
#sns.countplot(x=df1.basement)
plt.hist(df1['basement'],histtype='bar')
```

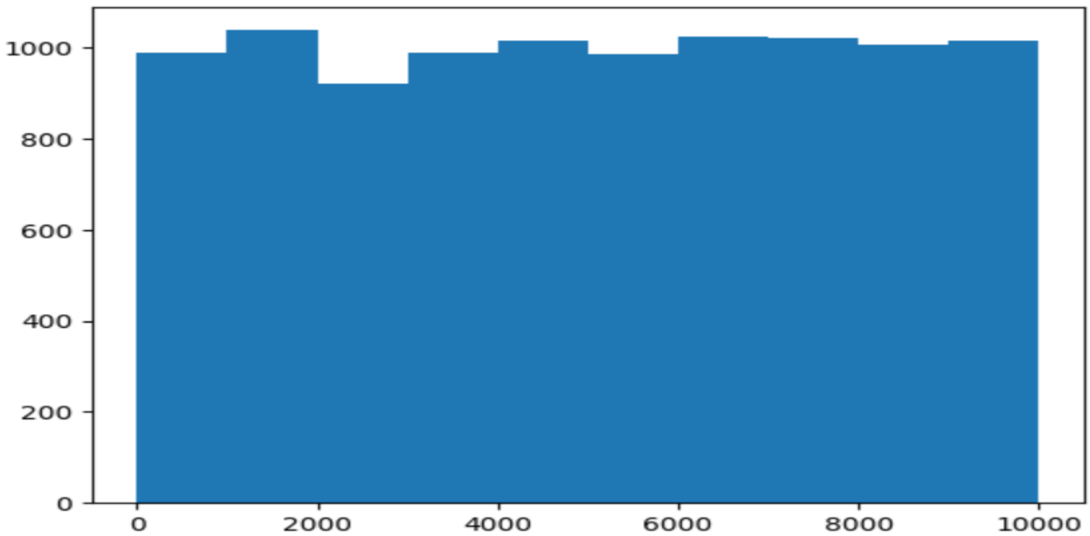
```
(array([1007.,  956.,  954.,  991., 1004., 1058., 1029.,  999.,  999.,
        1003.]),
 array([    0.,  1000.,  2000.,  3000.,  4000.,  5000.,  6000.,  7000.,
         8000.,  9000., 10000.]),
 <BarContainer object of 10 artists>)
```




Histogram of attic

```
plt.hist(df1['attic'],histtype='bar')
```

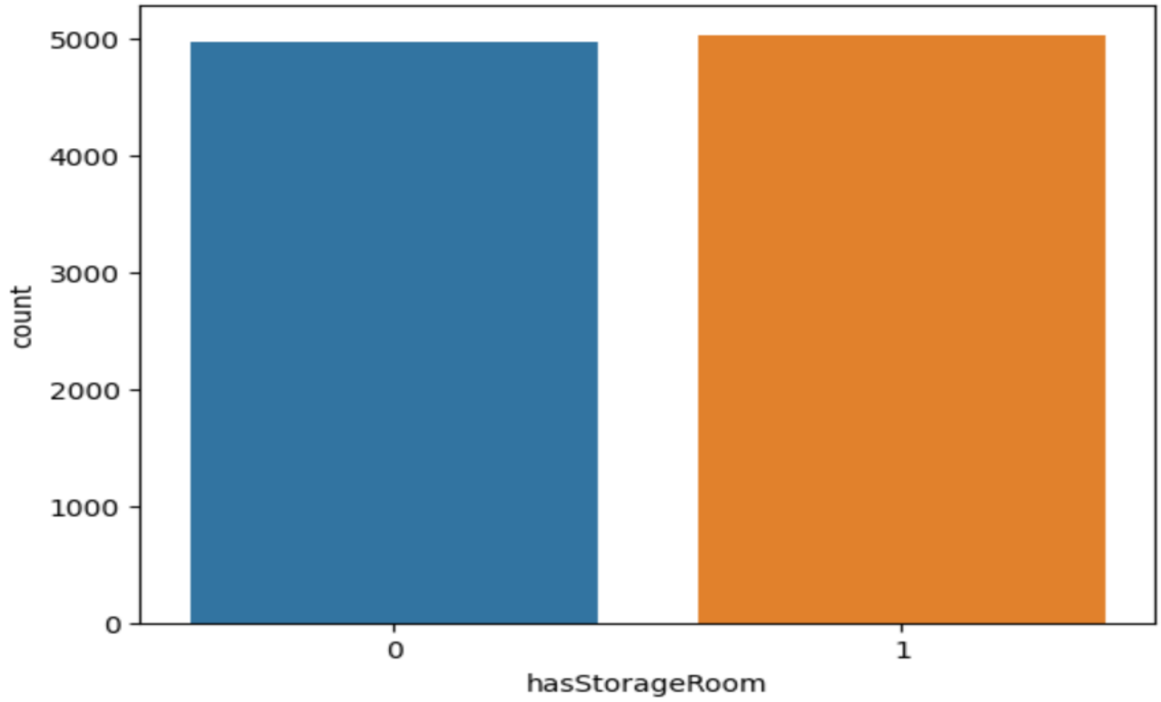
```
(array([ 989., 1038.,  921.,  989., 1014.,  985., 1024., 1021., 1005.,
        1014.]),
 array([1.0000e+00, 1.0009e+03, 2.0008e+03, 3.0007e+03, 4.0006e+03,
        5.0005e+03, 6.0004e+03, 7.0003e+03, 8.0002e+03, 9.0001e+03,
        1.0000e+04]),
 <BarContainer object of 10 artists>)
```



Countplot of hasStrogeRoom:

```
sns.countplot(x=df1.hasStorageRoom)
```

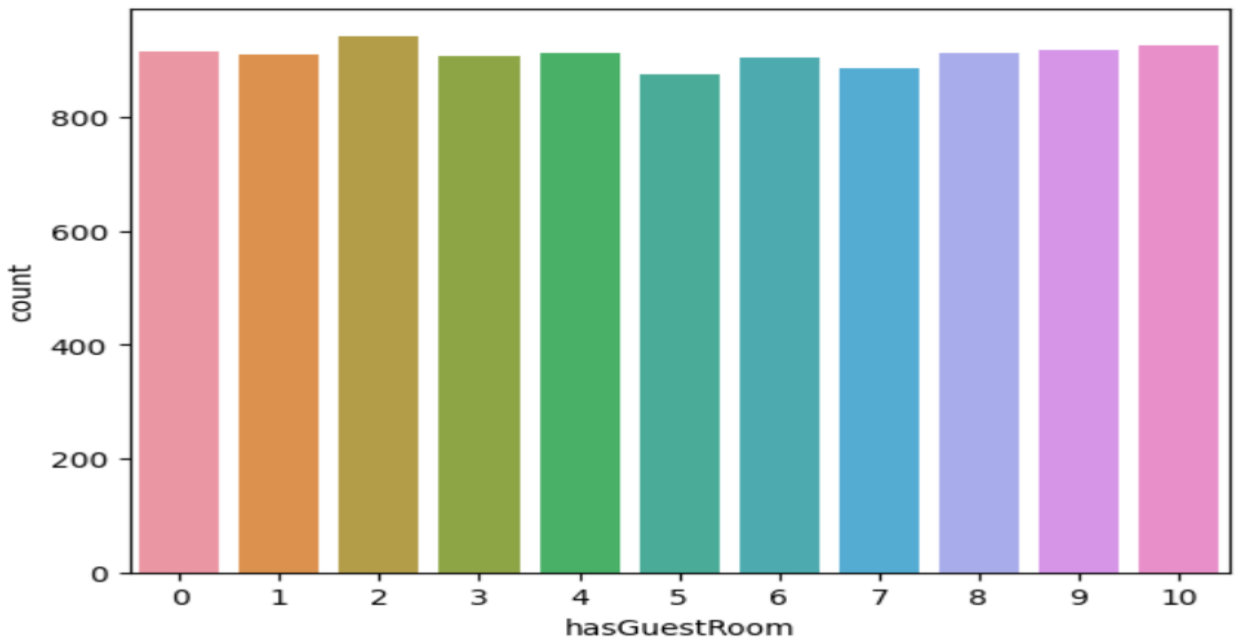
```
<Axes: xlabel='hasStorageRoom', ylabel='count'>
```




Countplot of hasGuestRoom

```
sns.countplot(x=df1.hasGuestRoom)
```

```
<Axes: xlabel='hasGuestRoom', ylabel='count'>
```



# Correlation between the variables:

```
d6=df1.corr()
d6
```

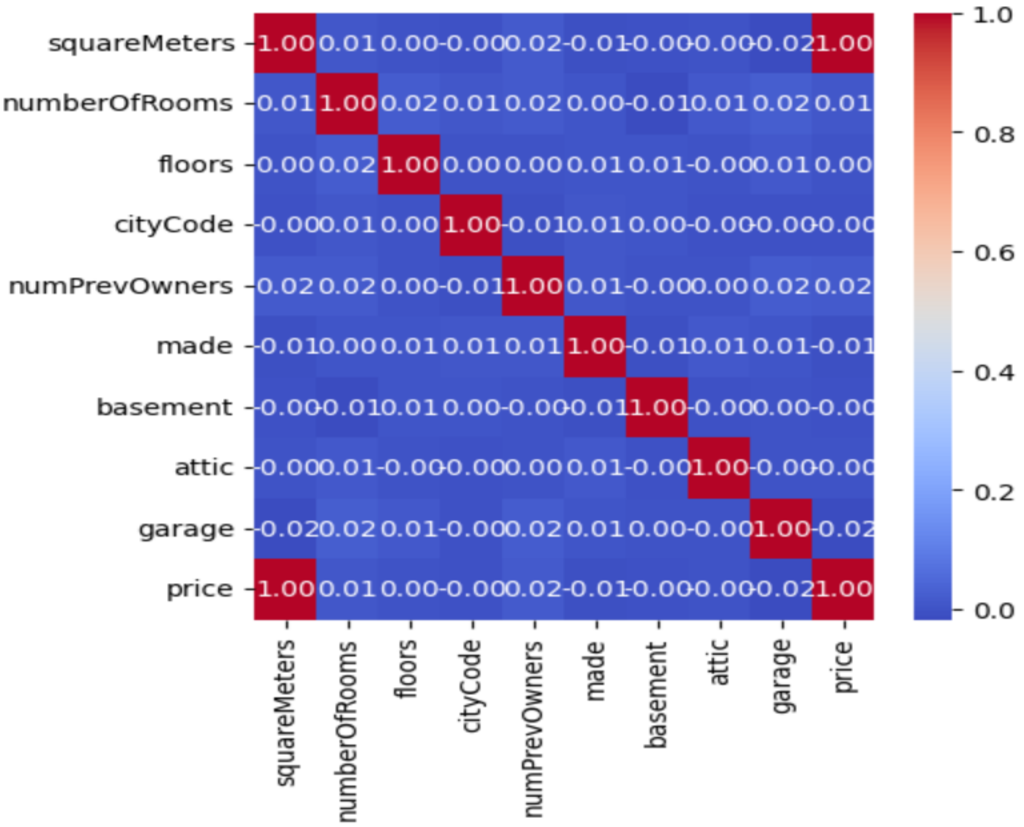
```
<ipython-input-114-548420e46a4a>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ve
  d6=df1.corr()
```

| | squareMeters | numberOfRooms | floors | cityCode | numPrevOwners | made | basement | attic | garage | price |
|---|---|---|---|---|---|---|---|---|---|---|
| **squareMeters** | 1.000000 | 0.009573 | 0.001109 | -0.001541 | 0.016619 | -0.007207 | -0.003960 | -0.000588 | -0.017246 | 0.999999 |
| **numberOfRooms** | 0.009573 | 1.000000 | 0.022244 | 0.009040 | 0.016766 | 0.003978 | -0.013990 | 0.012061 | 0.023188 | 0.009591 |
| **floors** | 0.001109 | 0.022244 | 1.000000 | 0.002207 | 0.002463 | 0.005022 | 0.006228 | -0.000270 | 0.011303 | 0.001654 |
| **cityCode** | -0.001541 | 0.009040 | 0.002207 | 1.000000 | -0.007549 | 0.009266 | 0.002652 | -0.002019 | -0.002208 | -0.001539 |
| **numPrevOwners** | 0.016619 | 0.016766 | 0.002463 | -0.007549 | 1.000000 | 0.006858 | -0.000862 | 0.000719 | 0.020268 | 0.016619 |
| **made** | -0.007207 | 0.003978 | 0.005022 | 0.009266 | 0.006858 | 1.000000 | -0.005506 | 0.013773 | 0.005687 | -0.007210 |
| **basement** | -0.003960 | -0.013990 | 0.006228 | 0.002652 | -0.000862 | -0.005506 | 1.000000 | -0.003180 | 0.000117 | -0.003967 |
| **attic** | -0.000588 | 0.012061 | -0.000270 | -0.002019 | 0.000719 | 0.013773 | -0.003180 | 1.000000 | -0.000611 | -0.000600 |
| **garage** | -0.017246 | 0.023188 | 0.011303 | -0.002208 | 0.020268 | 0.005687 | 0.000117 | -0.000611 | 1.000000 | -0.017229 |
| **price** | 0.999999 | 0.009591 | 0.001654 | -0.001539 | 0.016619 | -0.007210 | -0.003967 | -0.000600 | -0.017229 | 1.000000 |

# Heatmap of corelation :

```
plt.figure(figsize=(5,5))
sns.heatmap(df1.corr(),annot=True,fmt=".2f",cmap='coolwarm',)
```

```
<ipython-input-135-181667995521>:2: FutureWarning: The default value of
  sns.heatmap(df1.corr(),annot=True,fmt=".2f",cmap='coolwarm',)
<Axes: >
```




**'Price'and 'SquareMeters' have a perfect correlation with value 1**

## Model Building:

Model building using 'Price' features

```
#Linear Regression
y = df1['price']
y
# Define the features (all other columns except 'price')
X = df1.drop(columns=['price'])
X
# Split the data into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

(8000, 17)
(2000, 17)
(8000,)
(2000,)
```

Here,all variables except 'Price' are stored in the x variable and the target variable is passed as 'Price'and using train_test_split() the data has been split into training data and testing data

```
model = LinearRegression()
model.fit(X_train, y_train)

▾ LinearRegression
LinearRegression()
```

LinearRegression is used for model building

```
[119] y_pred = model.predict(X_test)
```

The values predicted by the model on test data are stored in y_pred

Model Evaluation for linear regression

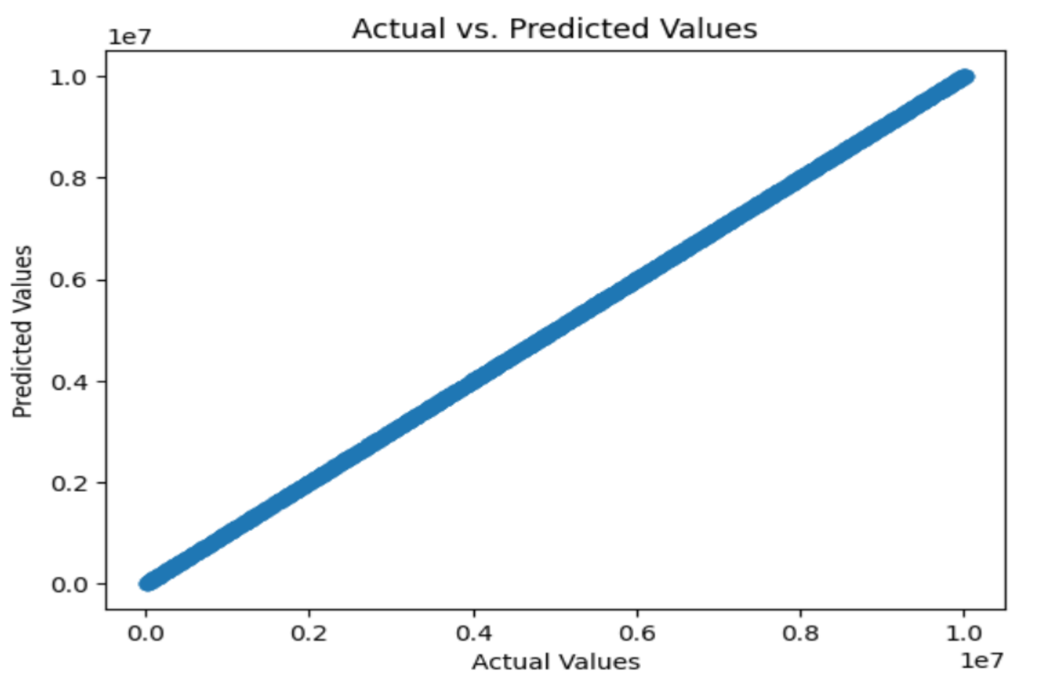
## Model Evaluation

```
mae=mean_absolute_error(y_test,y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse=np.sqrt(mse)
rmsle=np.log(rmse)
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"Root Mean Squared Log Error:{rmsle}")
print(f"R-squared (R2) Score: {r2}")
```

```
Mean Absolute Error:1510.000626697313
Mean Squared Error: 3695983.458085205
Root Mean Squared Error:1922.494072314712
Root Mean Squared Log Error:7.561378618053699
R-squared (R2) Score: 0.9999995780241576
```

## Plot of Actual values vs predicted values

```
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs. Predicted Values")
plt.show()
```




Evaluation of training dataset of linear regression model

```
] y_pred_train=model.predict(X_train)
```

```
mae=mean_absolute_error(y_train,y_pred_train)
mse = mean_squared_error(y_train, y_pred_train)
rmse=np.sqrt(mse)
rmsle=np.log(rmse)
r2 = r2_score(y_train, y_pred_train)
print(f"Mean Absolute Error:{mae}")
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error:{rmse}")
print(f"Root Mean Squared Log Error:{rmsle}")
print(f"R-squared (R2) Score: {r2}")
```

```
Mean Absolute Error:1470.7676641911207
Mean Squared Error: 3575290.0994709968
Root Mean Squared Error:1890.8437533204578
Root Mean Squared Log Error:7.544778438761601
R-squared (R2) Score: 0.9999995615523728
```

## Model building using ‘category’ as a target variable

```
y1 = df1['category']
y1
X1 = df1.drop(columns=['category'])
X1
# Split the data into train and test sets (80% train, 20% test)
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size=0.2, random_state=42)
print(X_train1.shape)
print(X_test1.shape)
print(y_train1.shape)
print(y_test1.shape)
```

```
(8000, 17)
(2000, 17)
(8000,)
(2000,)
```

Here,all variables except ‘Category’ are stored in the x variable and the target variable is passed as ‘Category’and using train_test_split() the data has been split into training data and testing data

LinearRegression is used for model building

```
model1= LogisticRegression()

# Fit the model to the training data
model1.fit(X_train1, y_train1)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
[129] y_pred2=model1.predict(X_test1)
```

Model evaluation for logistic regression model on test data

Model Evaluation for Logistic Regression

```
[130] accuracy = accuracy_score(y_test1, y_pred2)
      print(f'Accuracy: {accuracy:.2f}')

      # Confusion matrix
      cm = confusion_matrix(y_test1, y_pred2)
      print('Confusion Matrix:\n', cm)

      # Classification report
      report = classification_report(y_test1, y_pred2)
      print('Classification Report:\n', report)
```

```
Accuracy: 0.87
Confusion Matrix:
 [[   0  256]
 [   0 1744]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       256
           1       0.87      1.00      0.93      1744

    accuracy                           0.87      2000
   macro avg       0.44      0.50      0.47      2000
weighted avg       0.76      0.87      0.81      2000
```

**Accuracy of Model is 0.87**

**Validation of Model:**

For above dataset , I've used Linear Regression algorithm to build a model

I have selected '**Price**' as target variable . After performing all the steps of the Algorithm ,the **RMSE for the Training dataset as well as Testing dataset was observed to be low and the R2 score was high**. The values are given below in a tabular form:

| | **MAE** | **MSE** | **MSLE** | **RMSE** | **R2 Score** |
|---|---|---|---|---|---|
| **Training Dataset** | **1510.00** | **3695983.4580** | **7.5613** | **1922.4940** | **0.9999** |
| **Testing Dataset** | **1470.76** | **3575390** | **7.54475** | **1890.8437** | **0.9999** |

Taken Target variable as '**Category'** ,and using Logistic regression algorithm to build a model.

**The accuracy of logistic model is 0.87**

**Confusion Matrix:**

| | Luxury(0) | Basic(1) |
|---|---|---|
| Luxury(0) | TP | FN |
| | 0 | 256 |
| Basic(1) | TN | TN |
| | 0 | 1744 |