```python
# This Python 3 environment comes with many helpful analytics
libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session
```

```
/kaggle/input/house-rent-prediction-dataset/Dataset Glossary.txt
/kaggle/input/house-rent-prediction-dataset/House_Rent_Dataset.csv
```

TEAM: DA EXPLORERS **TEAM-MATES:** ARJUN VINAY AVADHANI PES2UG20CS901 DISHA SINGH D PES2UG20CS906 VAISHNAVI R BHAT PES2UG20CS922

Importing the Required Directories

```python
#Importing the Required Directories

#Data Analysis Libraries
import pandas as pd
import numpy as np
#Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns

from scipy.stats import probplot, boxcox
from scipy.special import inv_boxcox
#Data Preprocessing
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.model_selection import
train_test_split,cross_val_score,KFold
#Importing Models
```

```
from sklearn.linear_model import
LinearRegression,Lasso,Ridge,BayesianRidge
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.svm import SVR
from catboost import CatBoostRegressor
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor

from sklearn.metrics import r2_score,mean_squared_error
```

```
<IPython.core.display.HTML object>
```

```
import warnings
warnings.filterwarnings('ignore')
```

About Dataset

Context Housing in India varies from palaces of erstwhile maharajas to modern apartment buildings in big cities to tiny huts in far-flung villages. There has been tremendous growth in India's housing sector as incomes have risen. The Human Rights Measurement Initiative finds that India is doing 60.9% of what should be possible at its level of income for the right to housing.

Renting, also known as hiring or letting, is an agreement where a payment is made for the temporary use of a good, service, or property owned by another. A gross lease is when the tenant pays a flat rental amount and the landlord pays for all property charges regularly incurred by the ownership. Renting can be an example of the sharing economy.

Content In this Dataset, we have information on almost 4700+ Houses/Apartments/Flats Available for Rent with different parameters like BHK, Rent, Size, No. of Floors, Area Type, Area Locality, City, Furnishing Status, Type of Tenant Preferred, No. of Bathrooms, Point of Contact.

```
#Accessing the Dataset
final_df=pd.read_csv('../input/house-rent-prediction-dataset/House_Ren
t_Dataset.csv')
```

```
#Looking at the Dataset
final_df.head()
```

```
    Posted On  BHK   Rent  Size            Floor    Area Type  \
0  2022-05-18    2  10000  1100  Ground out of 2   Super Area   
1  2022-05-13    2  20000   800       1 out of 3   Super Area   
2  2022-05-16    2  17000  1000       1 out of 3   Super Area   
3  2022-07-04    2  10000   800       1 out of 2   Super Area   
4  2022-05-09    2   7500   850       1 out of 2  Carpet Area   
```

```
              Area Locality     City Furnishing Status  Tenant 
Preferred  \
0                    Bandel  Kolkata       Unfurnished  
Bachelors/Family   
1  Phool Bagan, Kankurgachi  Kolkata    Semi-Furnished  
Bachelors/Family   
2   Salt Lake City Sector 2  Kolkata    Semi-Furnished  
Bachelors/Family   
3               Dumdum Park  Kolkata       Unfurnished  
Bachelors/Family   
4             South Dum Dum  Kolkata       Unfurnished         
Bachelors   

   Bathroom Point of Contact  
0         2    Contact Owner  
1         1    Contact Owner  
2         1    Contact Owner  
3         1    Contact Owner  
4         1    Contact Owner  
```

```
final_df.shape
```

```
(4746, 12)
```

Observation:

- The dataset has 4746 rows and 12 columns

```
#Accessing the columns of the Dataset
final_df.columns
```

```
Index(['Posted On', 'BHK', 'Rent', 'Size', 'Floor', 'Area Type',
       'Area Locality', 'City', 'Furnishing Status', 'Tenant 
Preferred',
       'Bathroom', 'Point of Contact'],
      dtype='object')
```

The dataset has 12 columns. These are the features or attributes depending on which the house rent is predicted.

```
final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4746 entries, 0 to 4745
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Posted On          4746 non-null   object
 1   BHK                4746 non-null   int64 
 2   Rent               4746 non-null   int64 
 3   Size               4746 non-null   int64 
 4   Floor              4746 non-null   object
```

```
 5   Area Type          4746 non-null   object
 6   Area Locality      4746 non-null   object
 7   City               4746 non-null   object
 8   Furnishing Status  4746 non-null   object
 9   Tenant Preferred   4746 non-null   object
 10  Bathroom           4746 non-null   int64
 11  Point of Contact   4746 non-null   object
dtypes: int64(4), object(8)
memory usage: 445.1+ KB
```

Observation:

- This dataset doesn't contain any Null values.

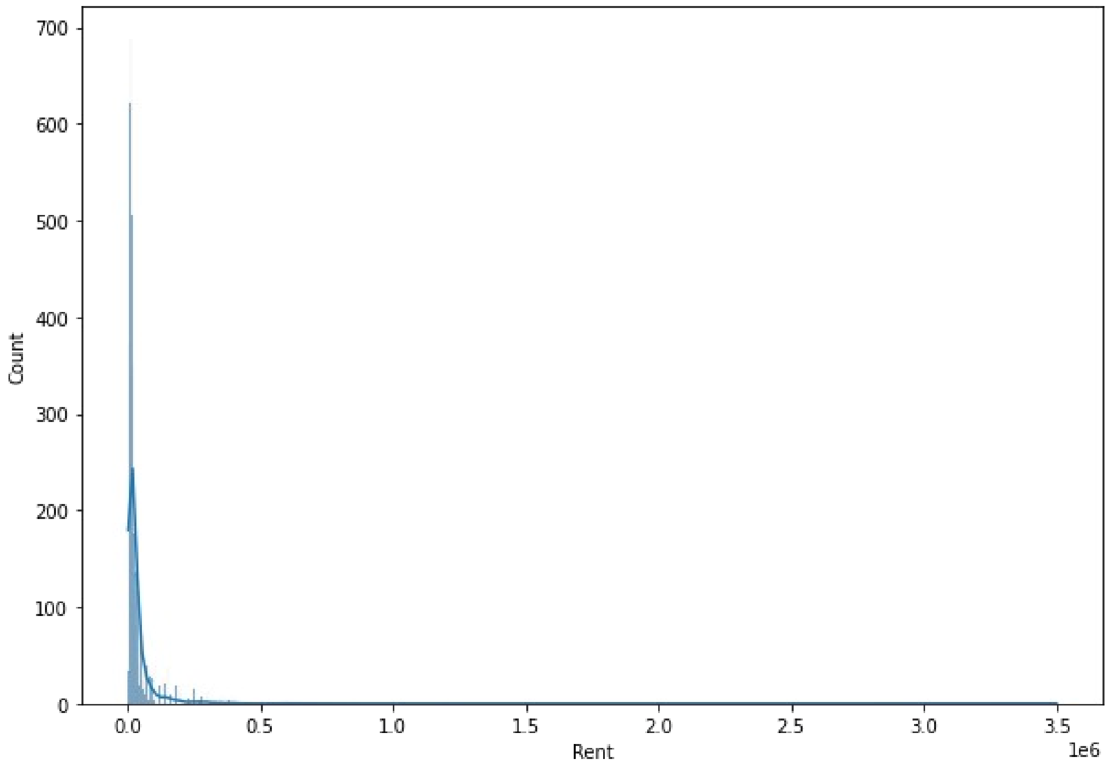
```
#Checking the Mean of the Rent
print('The Mean of the Rent is {}'.format(final_df['Rent'].mean()))
print('The Standard Deviation of Rent is
{}'.format(final_df['Rent'].std()))
```

```
The Mean of the Rent is 34993.45132743363
The Standard Deviation of Rent is 78106.4129373483
```

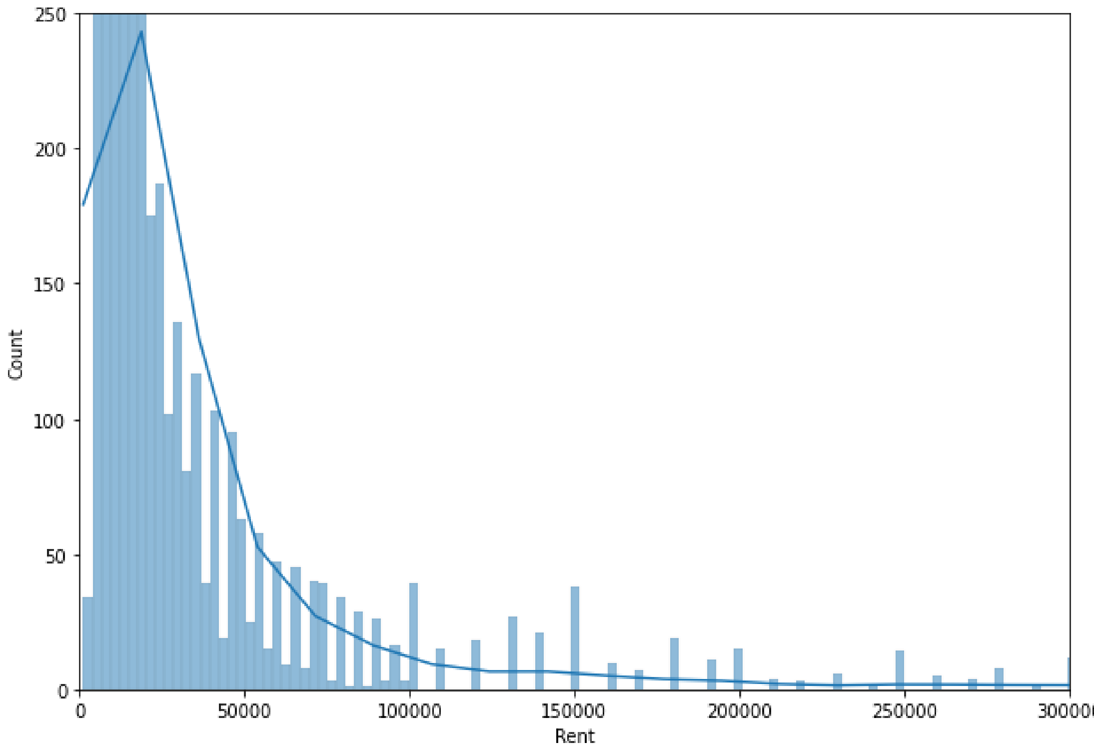
Standard deviation returns a calculated value that describes whether the data points are in close proximity or whether they are spread out for which the mean is required.

Exploratory Data Analysis

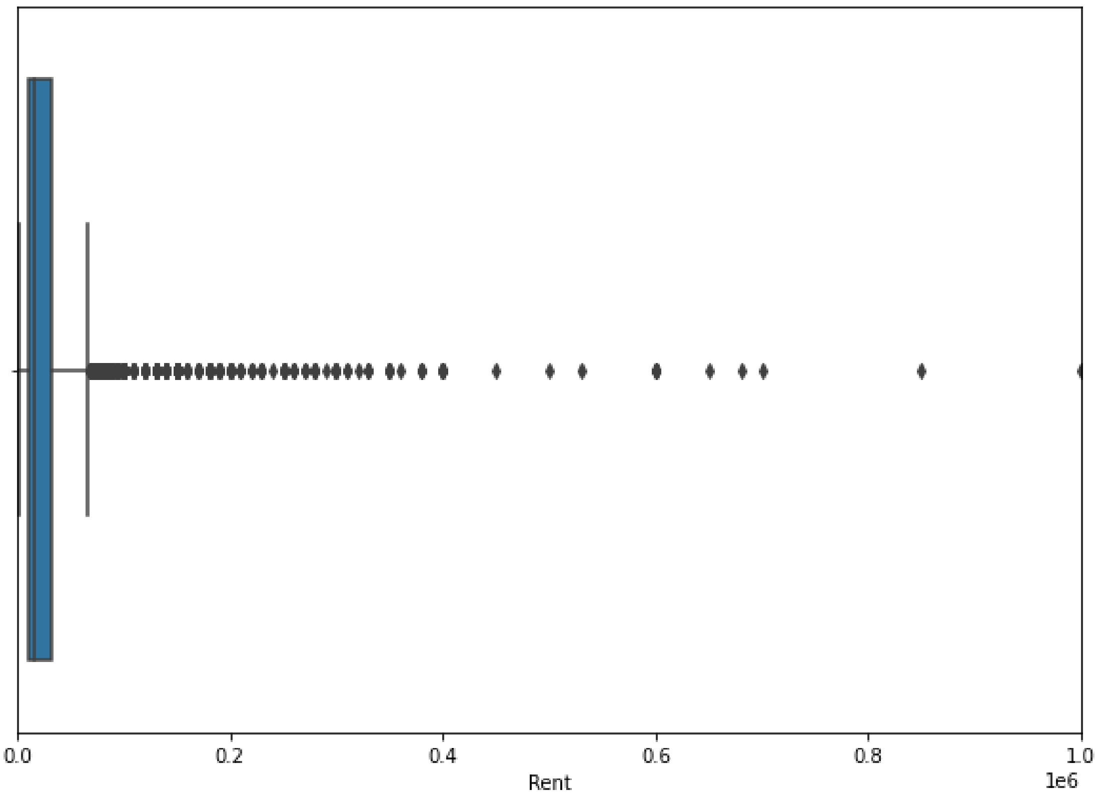
```
#Analyzing the Rent
plt.figure(figsize=(10,7))
sns.histplot(final_df.Rent,kde=True)
plt.show()
```




```
plt.figure(figsize=(10,7))
sns.histplot(final_df.Rent,kde=True)
plt.xlim(0,300000)
plt.ylim(0,250)
plt.show()
```




```
plt.figure(figsize=(10,7))
sns.boxplot(final_df.Rent)
plt.xlim(0,1000000)
plt.show()
```

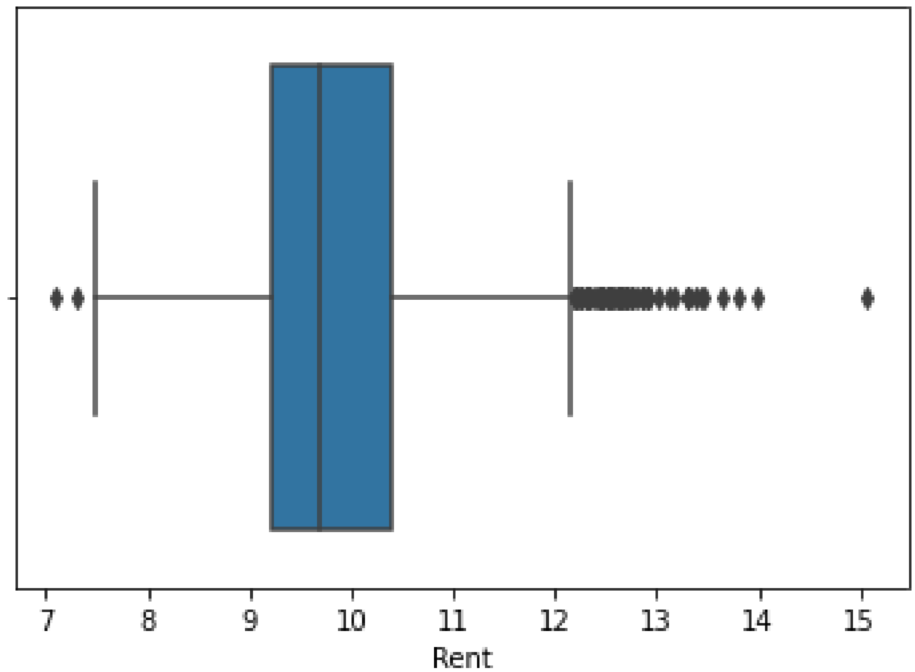



```
#We will apply Log Transformation in order to convert Rent into Normal Distribution
final_df['Rent']=np.log1p(final_df['Rent'])
```

When our original continuous data do not follow the bell curve, we can log transform this data to make it as “normal” as possible so that the statistical analysis results from this data become more valid. In other words, the log transformation reduces or removes the skewness of our original data. It improves linearity between our dependent and independent variables. It boosts validity of our statistical analyses.

```
sns.boxplot('Rent',data=final_df)
```

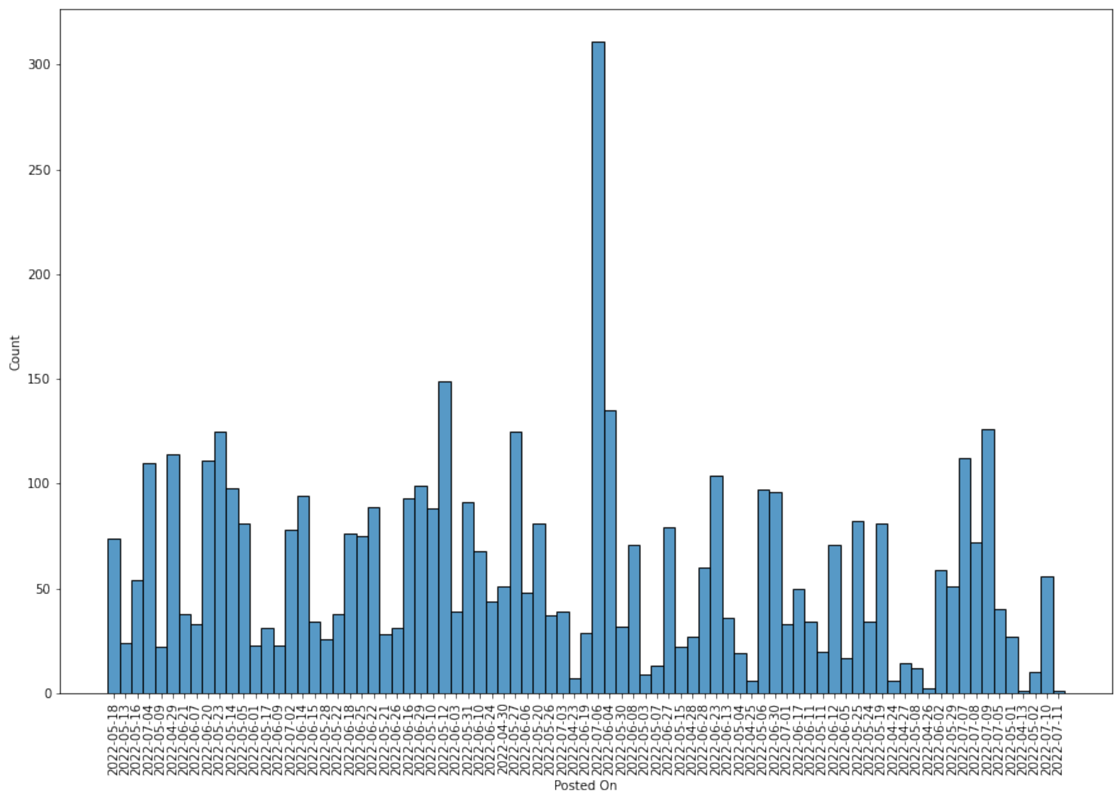
```
<AxesSubplot:xlabel='Rent'>
```




```
#Analyzing the Posted On column
final_df['Posted On'].nunique()
```

```
81
```

```
plt.figure(figsize=(15,10))
sns.histplot(final_df['Posted On'])
plt.xticks(rotation='vertical')
plt.show()
```

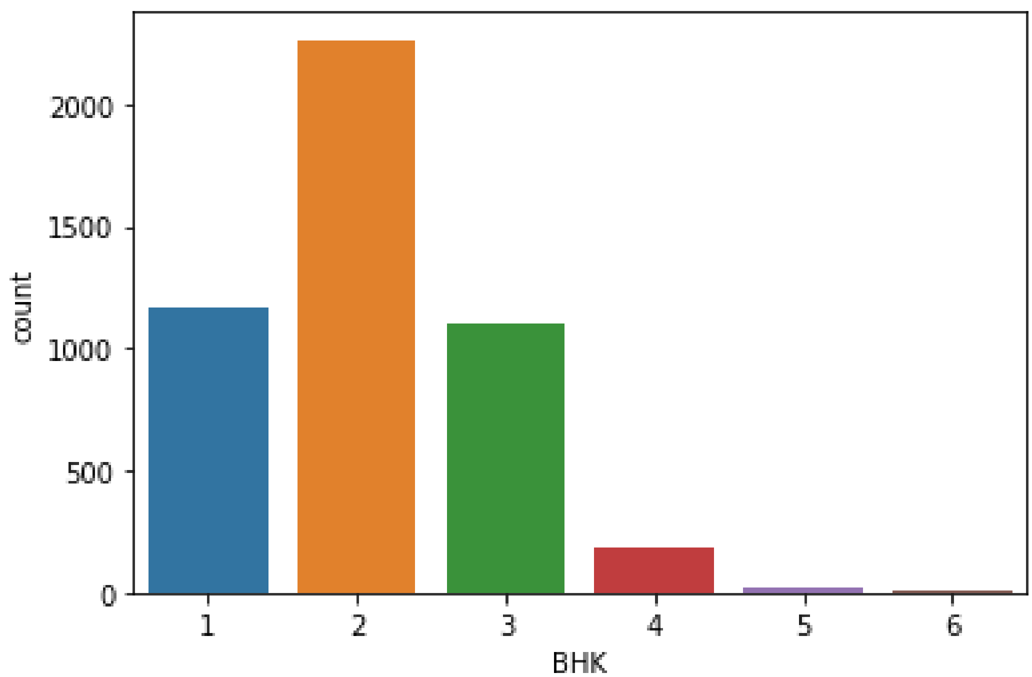



```python
#Analyzing the BHK column
final_df.BHK.unique()
```

```
array([2, 1, 3, 6, 4, 5])
```

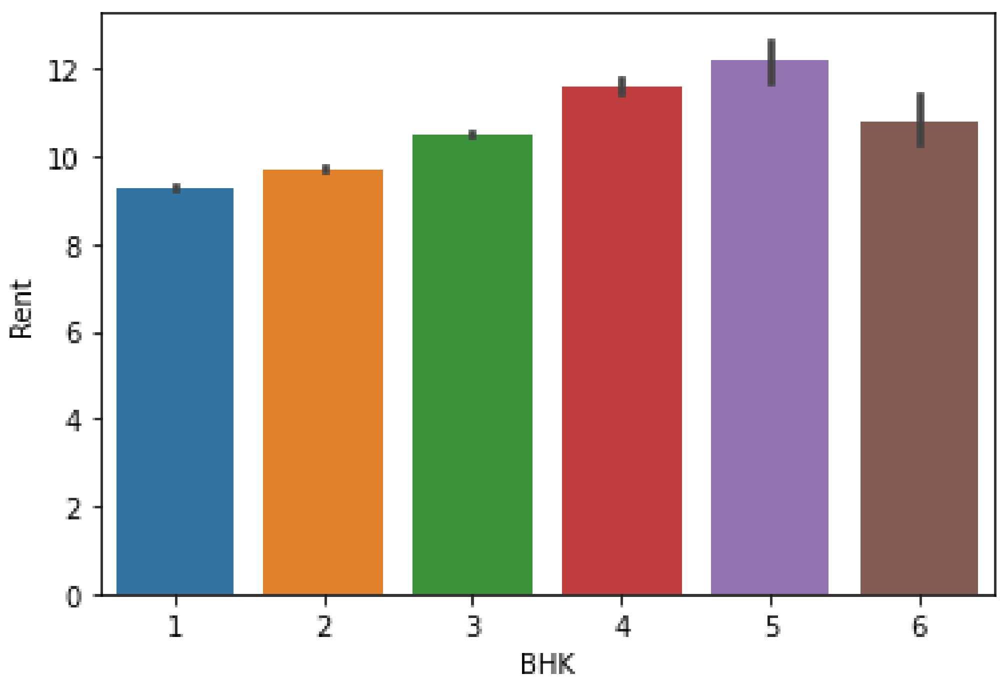
```python
sns.countplot('BHK',data=final_df)
```

```
<AxesSubplot:xlabel='BHK', ylabel='count'>
```




```
sns.barplot(x='BHK',y='Rent',data=final_df)
```

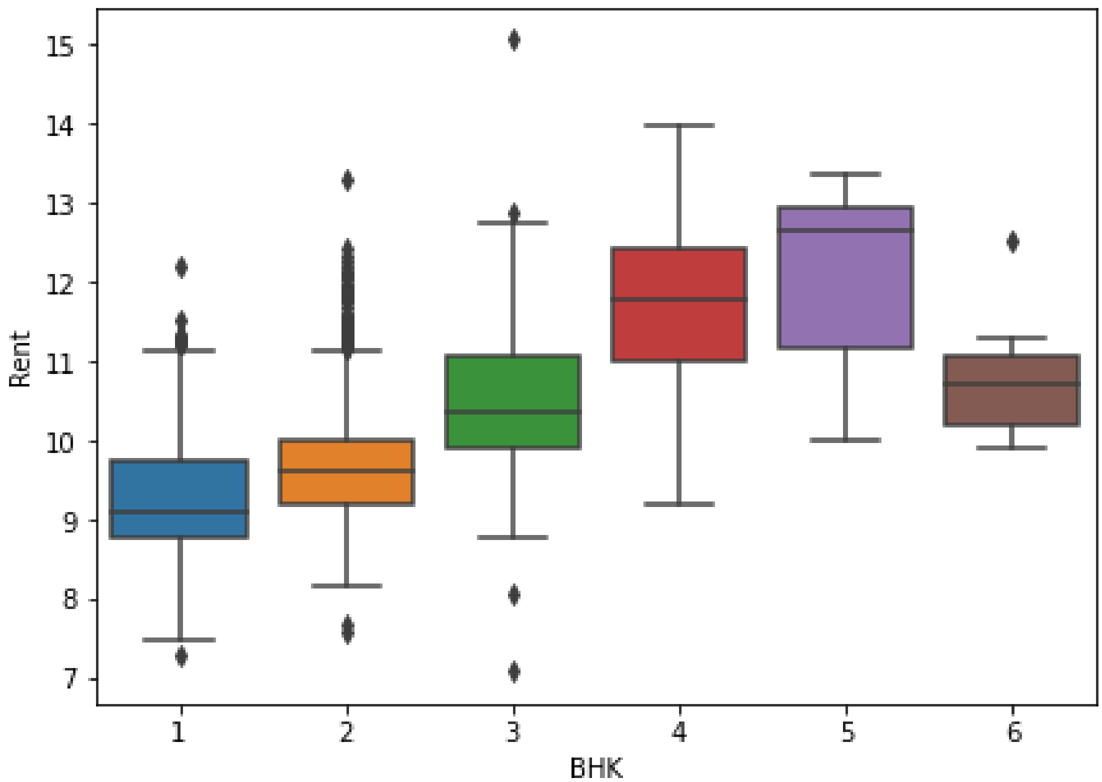
```
<AxesSubplot:xlabel='BHK', ylabel='Rent'>
```



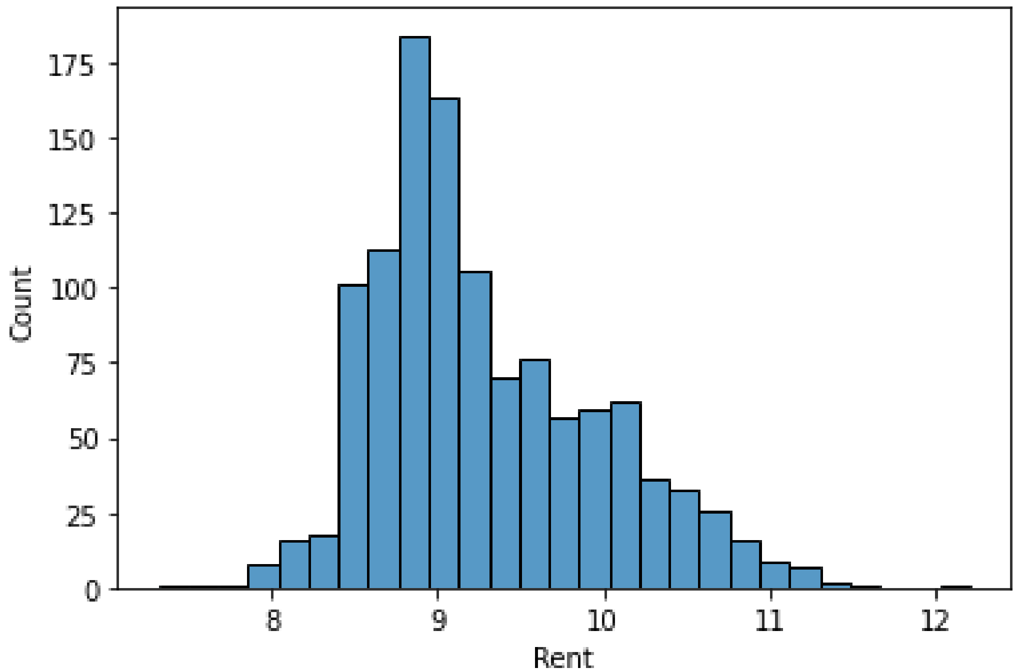
Observation:

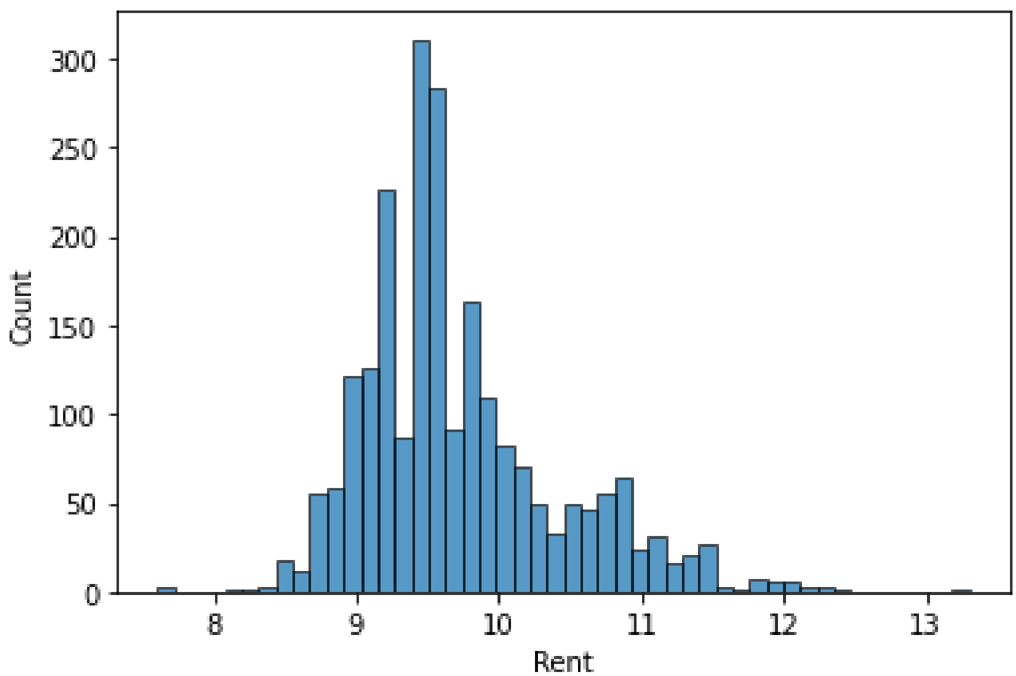
- The Average Rent of the 5BHK is the highest compared to all.

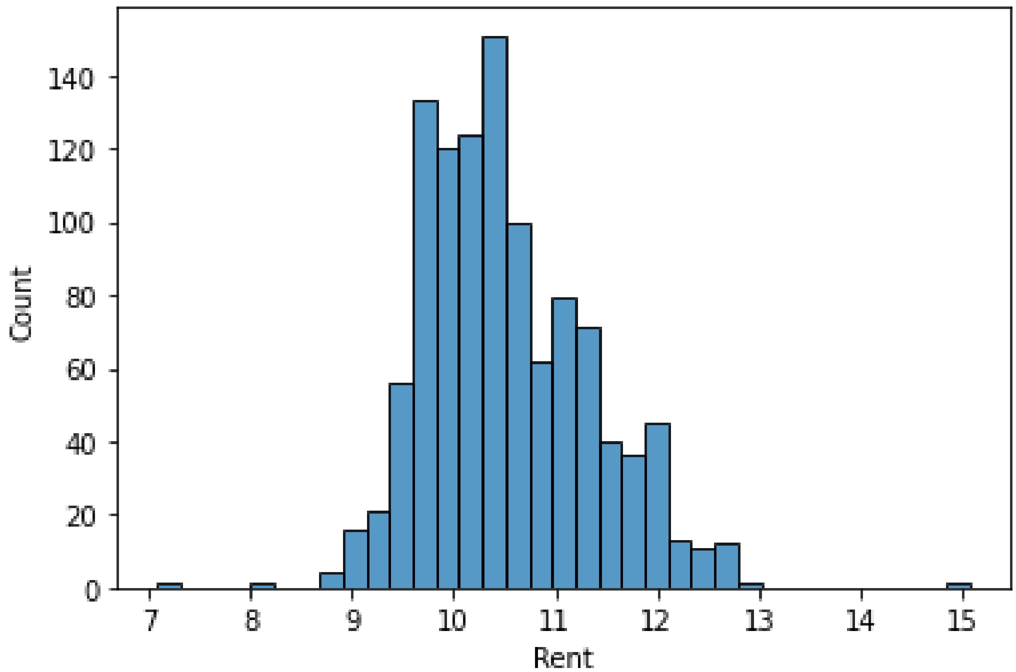
```
plt.figure(figsize=(7,5))
sns.boxplot(x='BHK',y='Rent',data=final_df)
plt.show()
```

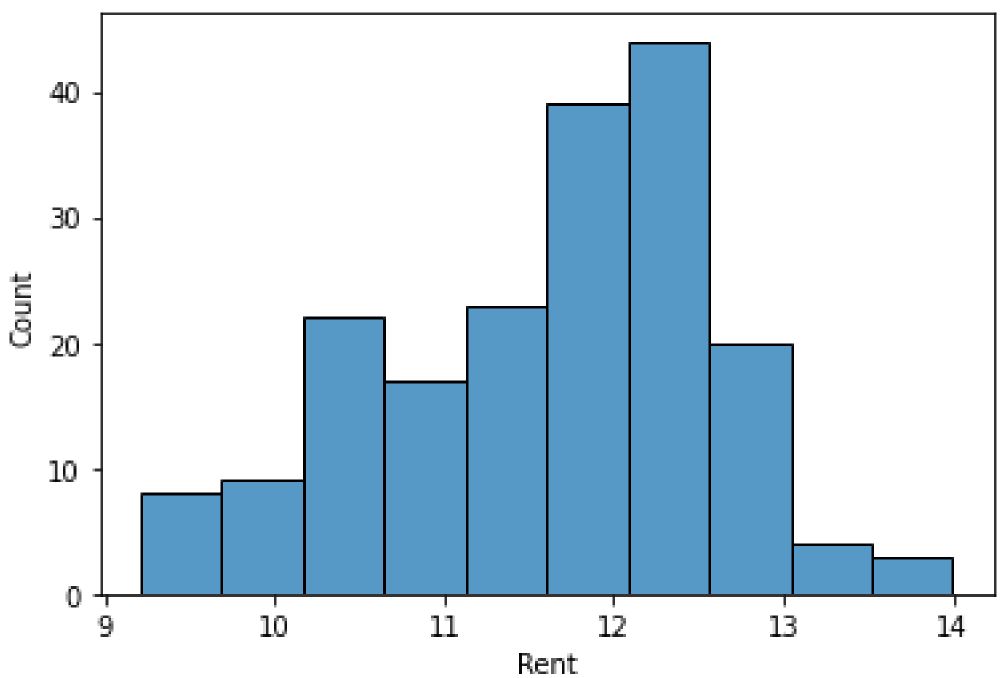


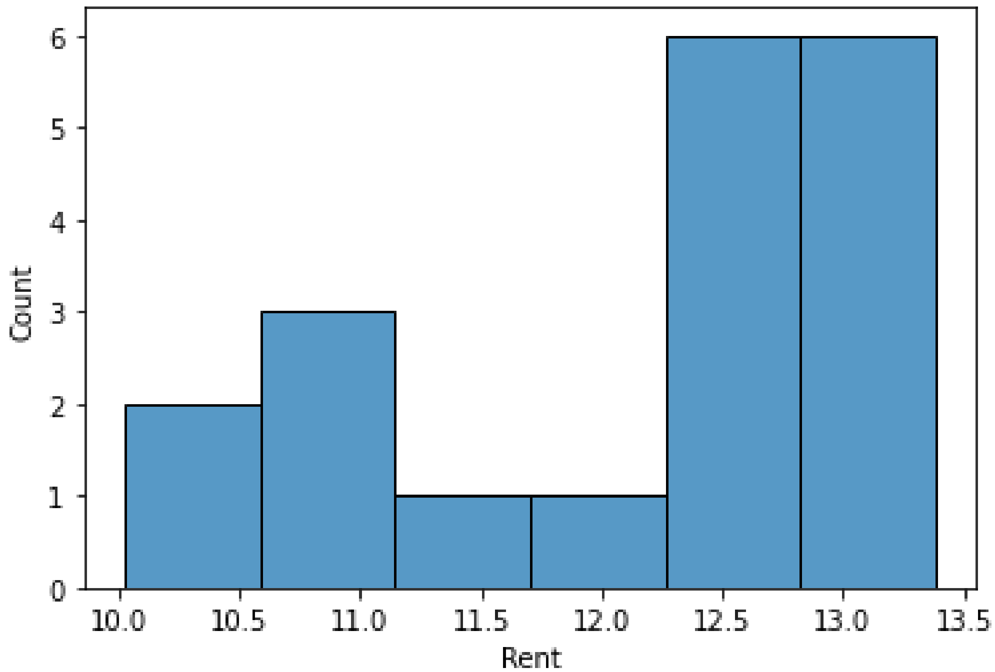

```
for i in range(6):
    sns.histplot(final_df[final_df['BHK']==i+1].Rent)
    plt.show()
```

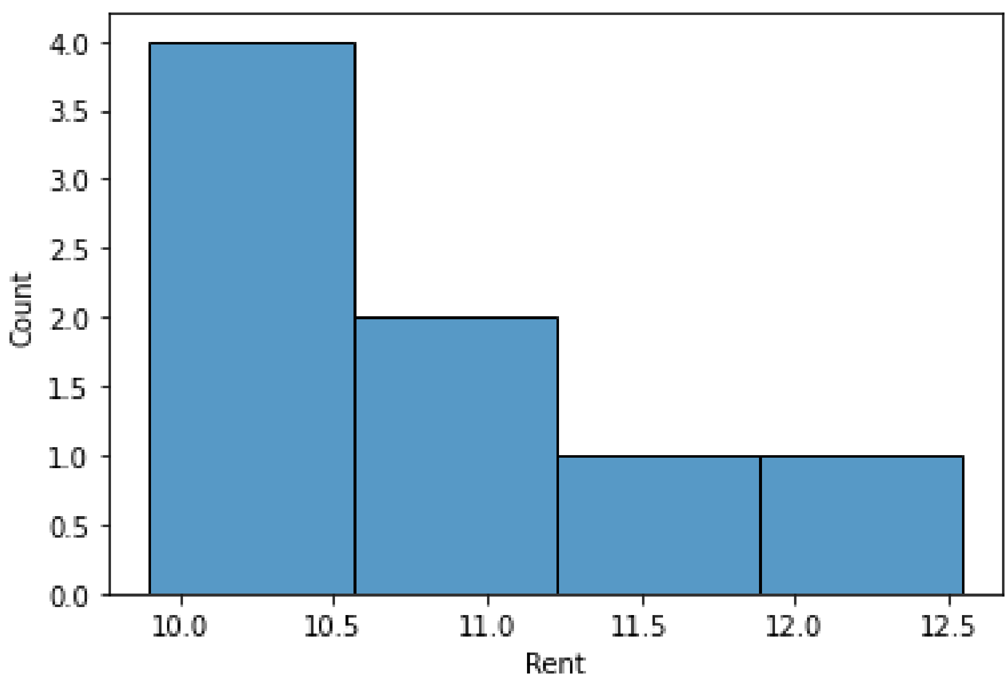

Count
Rent


Count
Rent


Count
Rent
0
20
40
60
80
100
120
140
7
8
9
10
11
12
13
14
15


Count
Rent
0
10
20
30
40
9
10
11
12
13
14





Observation:

- Each BHK contains a lot of Outliers.

Insights

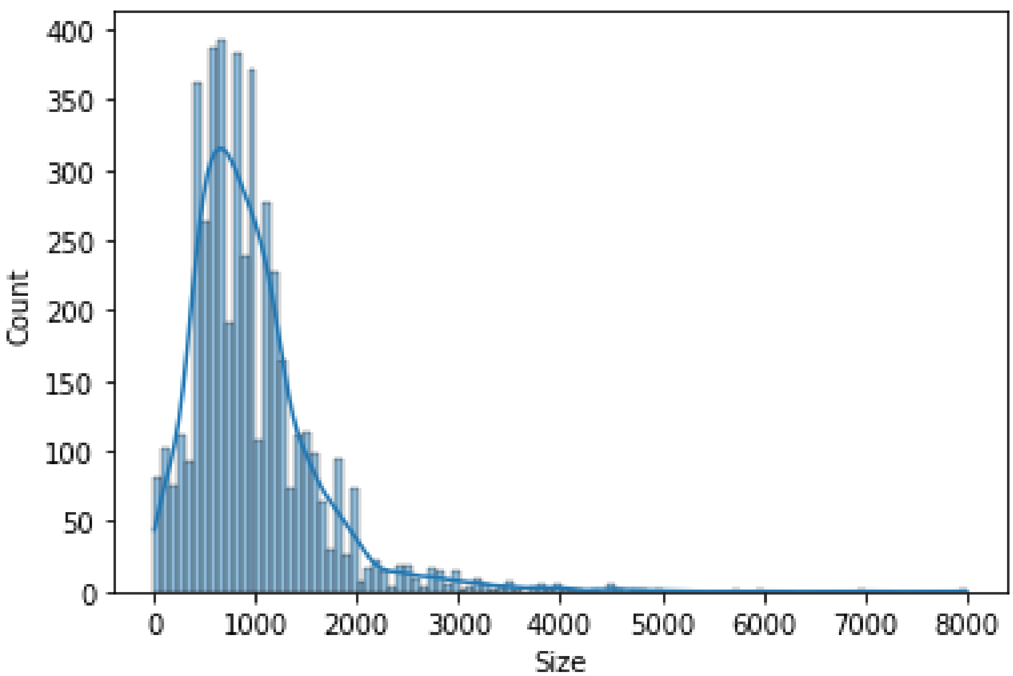
- Will Remove few of them without making the Dataset Thin

```
#Analyzing the Size column

#Unique values of Size
final_df['Size'].nunique()
```

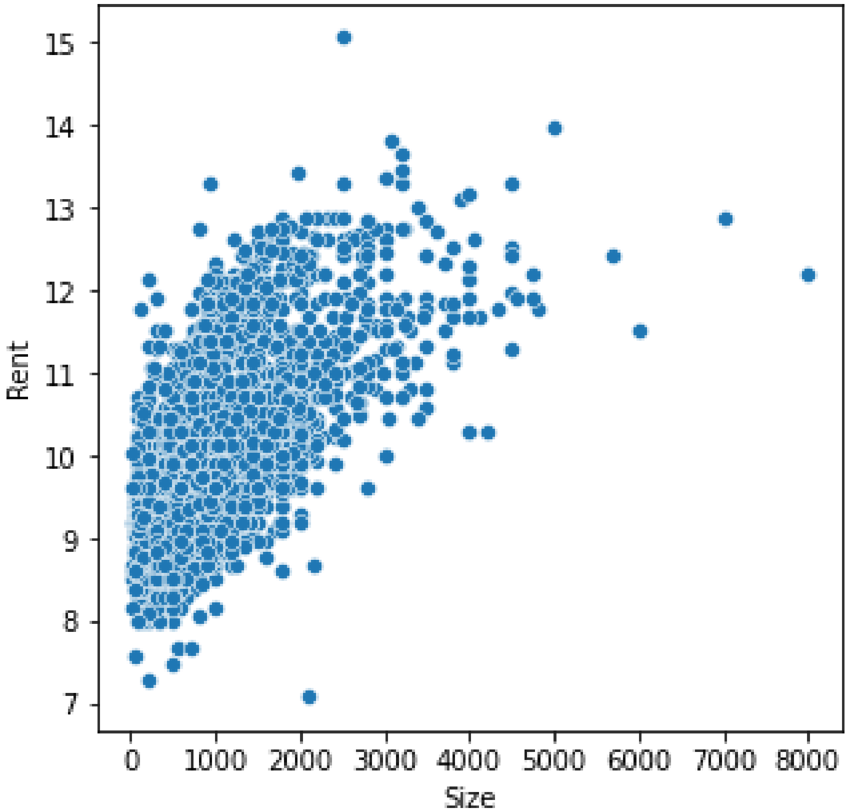
```
615
```

```
#Distribution of Size column
sns.histplot(final_df.Size,kde=True)
```

```
<AxesSubplot:xlabel='Size', ylabel='Count'>
```

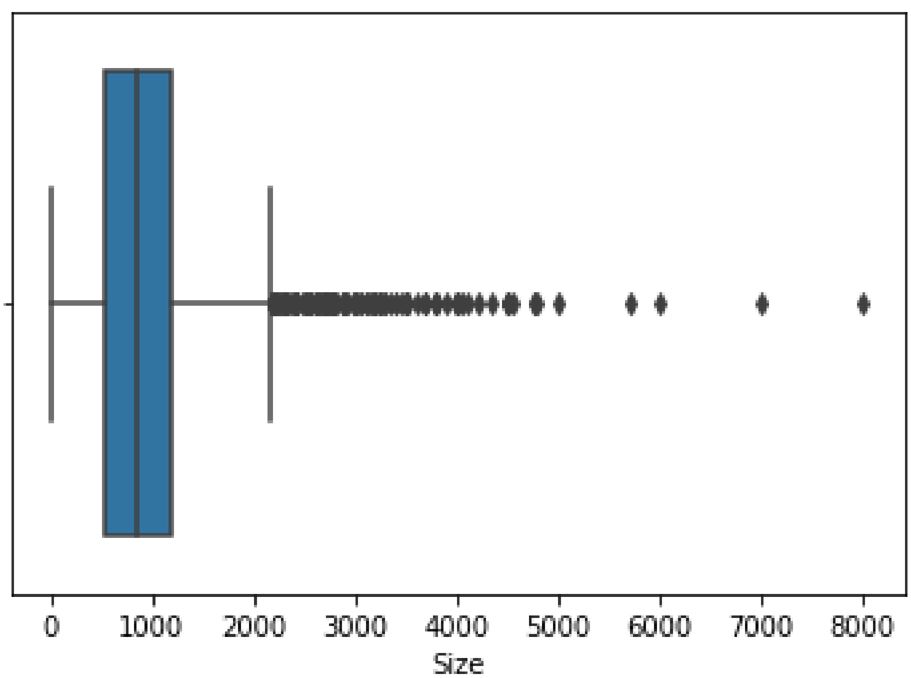



```
plt.figure(figsize=(5,5))
sns.scatterplot(x=final_df.Size,y=final_df.Rent)
plt.show()
```




```
sns.boxplot('Size',data=final_df)
```

```
<AxesSubplot:xlabel='Size'>
```



Observation:

- More the Size of House,More the Rent is
- Contains a lot of Outliers.

```
#Analyzing the Floor Column
final_df.Floor.value_counts()
```

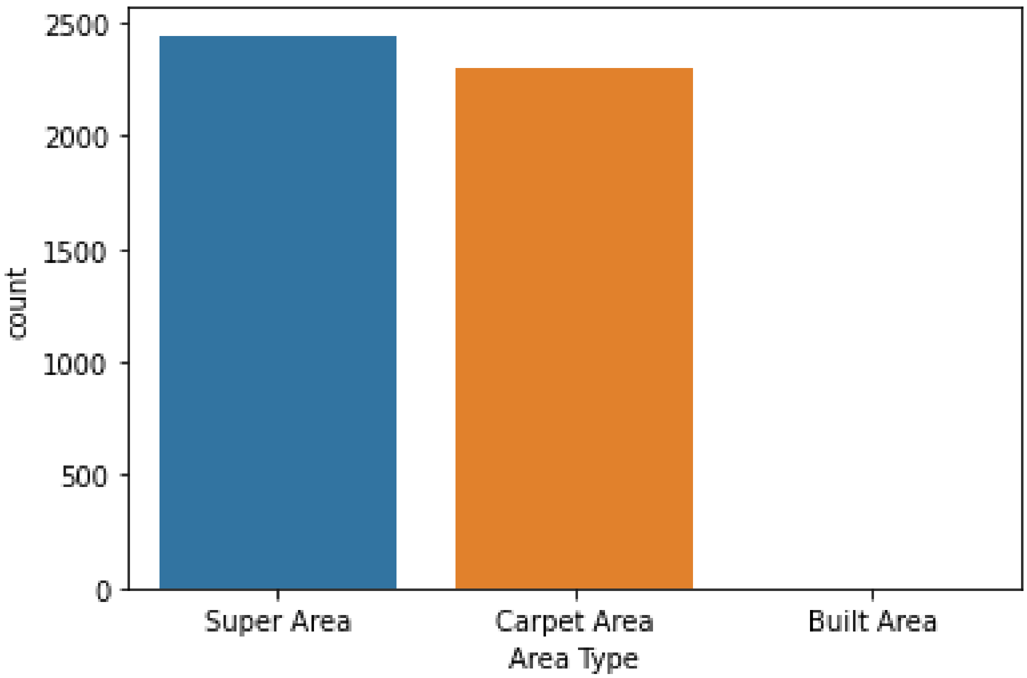
```
1 out of 2         379
Ground out of 2    350
2 out of 3         312
2 out of 4         308
1 out of 3         293
                  ...
11 out of 31         1
50 out of 75         1
18 out of 26         1
12 out of 27         1
23 out of 34         1
Name: Floor, Length: 480, dtype: int64
```

```
#Analyzing the Area Type
final_df['Area Type'].value_counts()
```

```
Super Area     2446
Carpet Area    2298
Built Area        2
Name: Area Type, dtype: int64
```

```
sns.countplot('Area Type',data=final_df)
```

```
<AxesSubplot:xlabel='Area Type', ylabel='count'>
```



Observation:

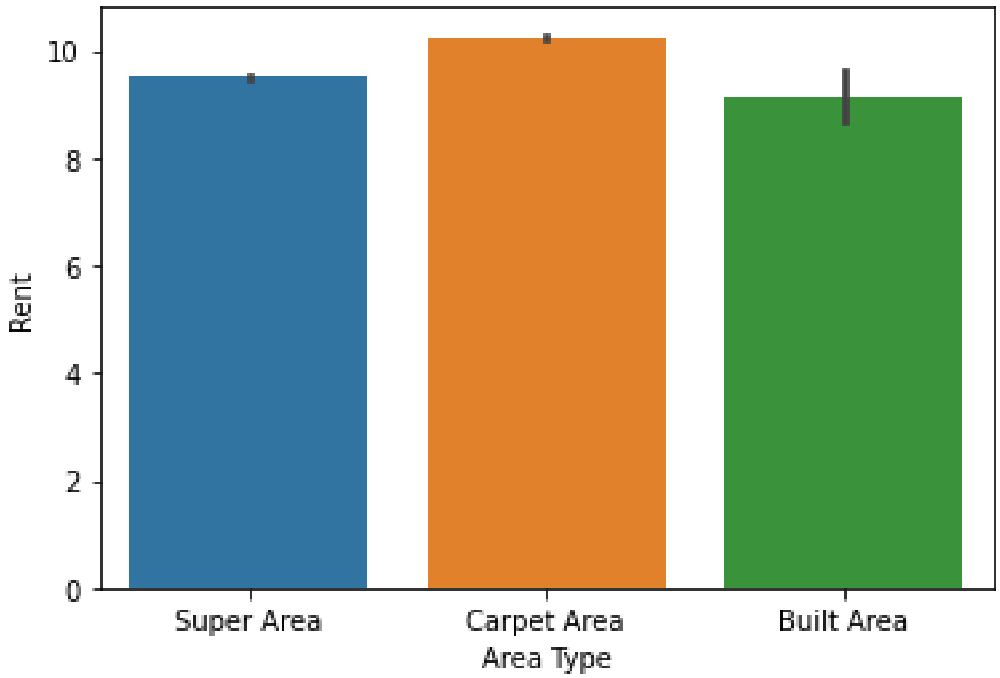
- The no. of Super Area is higher than all of the areas.
- Built Area contains only two houses for Rent in the Dataset.

```
final_df.groupby('Area Type')['Rent'].mean()
```

```
Area Type
Built Area       9.157777
Carpet Area     10.248720
Super Area       9.530714
Name: Rent, dtype: float64
```

```
sns.barplot(final_df['Area Type'],y=final_df['Rent'])
```

```
<AxesSubplot:xlabel='Area Type', ylabel='Rent'>
```



Observation:

- Built Area Houses are the Cheapest.
- Super Area Houses are cheaper as compared to Carpet Area.

```
#Analyzing the Area Locality
final_df['Area Locality'].value_counts()
```

```
Bandra West                                 37
Gachibowli                                  29
Electronic City                             24
Velachery                                   22
Miyapur, NH 9                               22
                                            ..
Kengeri Upanagara                            1
Ittamadu, Banashankari, Outer Ring Road      1
Rmv Extension, Armane Nagar                  1
snv la                                       1
Manikonda, Hyderabad                         1
Name: Area Locality, Length: 2235, dtype: int64
```

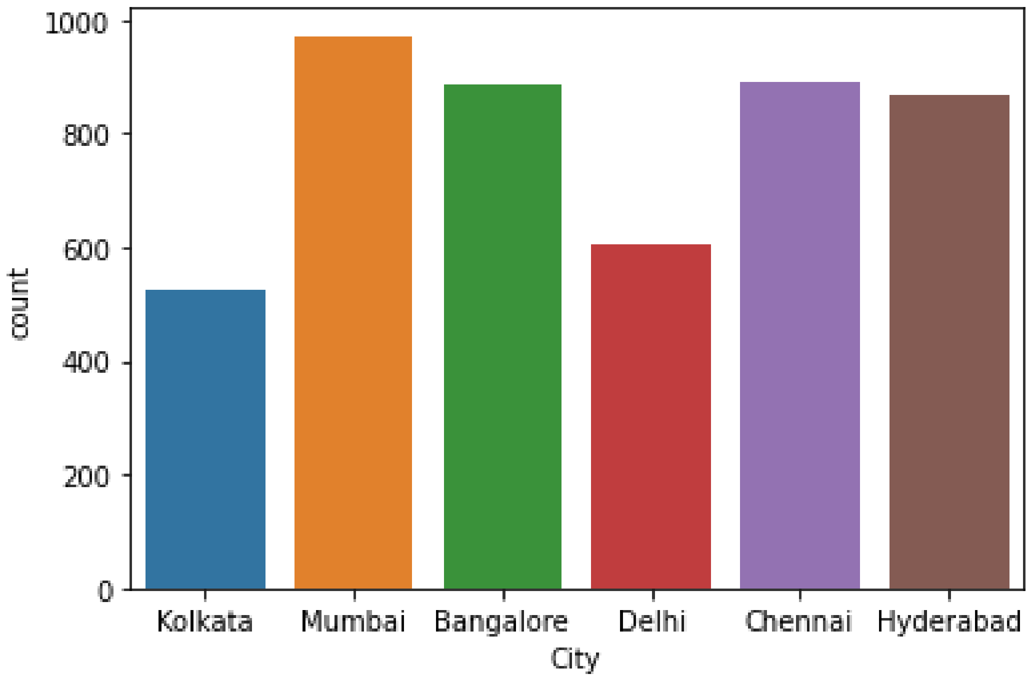
```
#Analyzing the City Column
final_df['City'].value_counts()
```

```
Mumbai       972
Chennai      891
Bangalore    886
Hyderabad    868
Delhi        605
```

```
Kolkata      524
Name: City, dtype: int64
```

```
sns.countplot(final_df.City)
```

```
&lt;AxesSubplot:xlabel='City', ylabel='count'&gt;
```

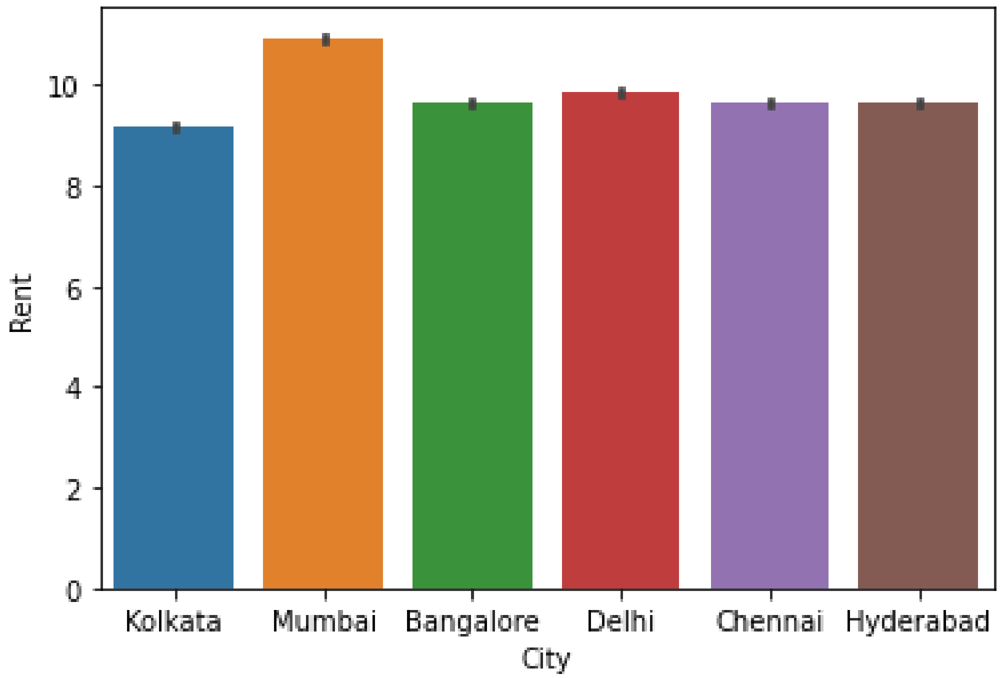



```
final_df.groupby('City')['Rent'].mean()
```

```
City
Bangalore     9.634777
Chennai       9.645544
Delhi         9.855434
Hyderabad     9.634704
Kolkata       9.150935
Mumbai       10.937095
Name: Rent, dtype: float64
```

```
sns.barplot('City','Rent',data=final_df)
```

```
&lt;AxesSubplot:xlabel='City', ylabel='Rent'&gt;
```

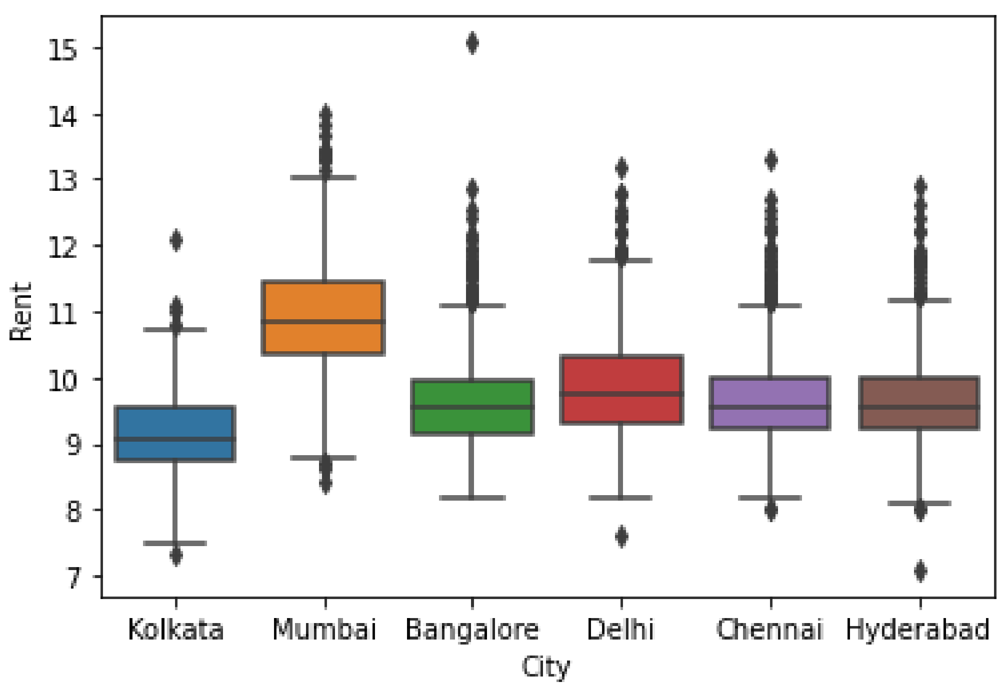


Observation

- The Average Rent of Mumbai House is the Highest than all of the Cities.

```
sns.boxplot('City','Rent',data=final_df)
```

```
<AxesSubplot:xlabel='City', ylabel='Rent'>
```



```
#Analyzing the Furnishing Status
final_df['Furnishing Status'].value_counts()
```

```
Semi-Furnished    2251
Unfurnished       1815
Furnished          680
Name: Furnishing Status, dtype: int64
```

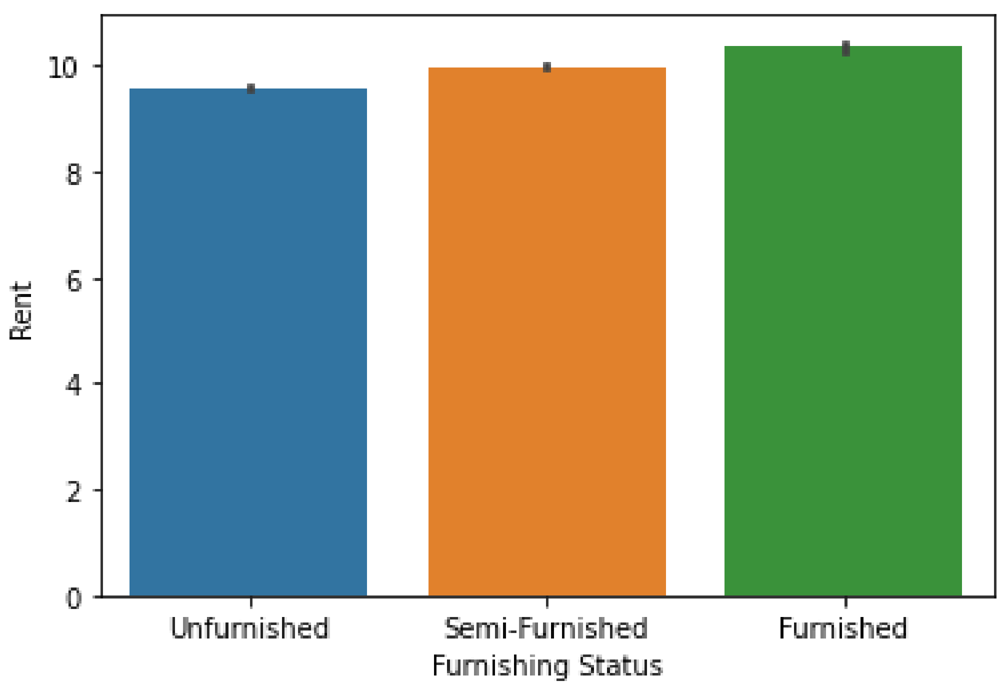
- 50% of the Houses are Semi Furnished.
- 40% of the Houses are Unfurnished.
- Rest of the Houses are Furnished.

```
final_df.groupby('Furnishing Status')['Rent'].mean()
```

```
Furnishing Status
Furnished         10.352446
Semi-Furnished     9.979319
Unfurnished        9.575146
Name: Rent, dtype: float64
```

```
sns.barplot('Furnishing Status','Rent',data=final_df)
```

```
<AxesSubplot:xlabel='Furnishing Status', ylabel='Rent'>
```

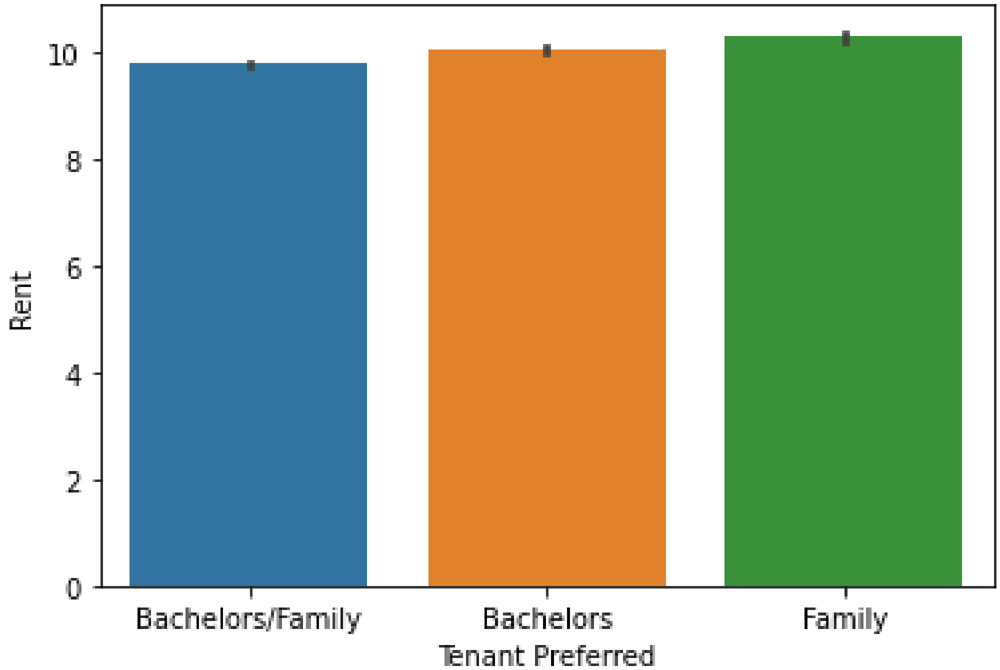



```
#Analyzing the Tenant Preferred
final_df['Tenant Preferred'].value_counts()
```

```
Bachelors/Family    3444
Bachelors            830
Family               472
Name: Tenant Preferred, dtype: int64
```

```
sns.barplot('Tenant Preferred','Rent',data=final_df)
```

<AxesSubplot:xlabel='Tenant Preferred', ylabel='Rent'>

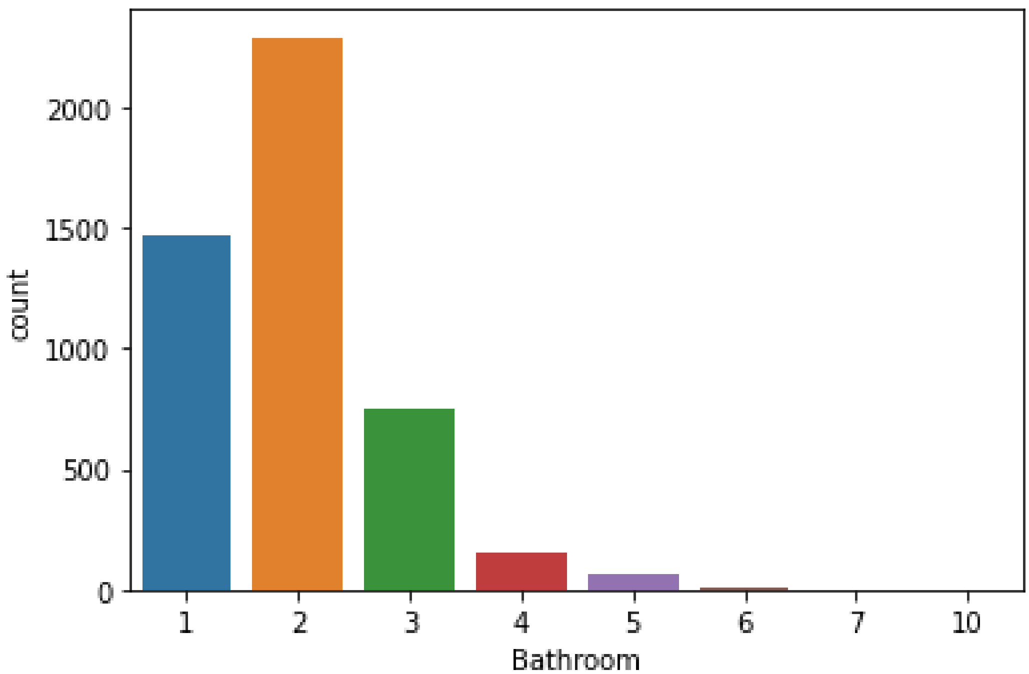



```
#Analyzing the Bathroom Column
final_df['Bathroom'].value_counts()
```

```
2     2291
1     1474
3      749
4      156
5       60
6       12
7        3
10       1
Name: Bathroom, dtype: int64
```

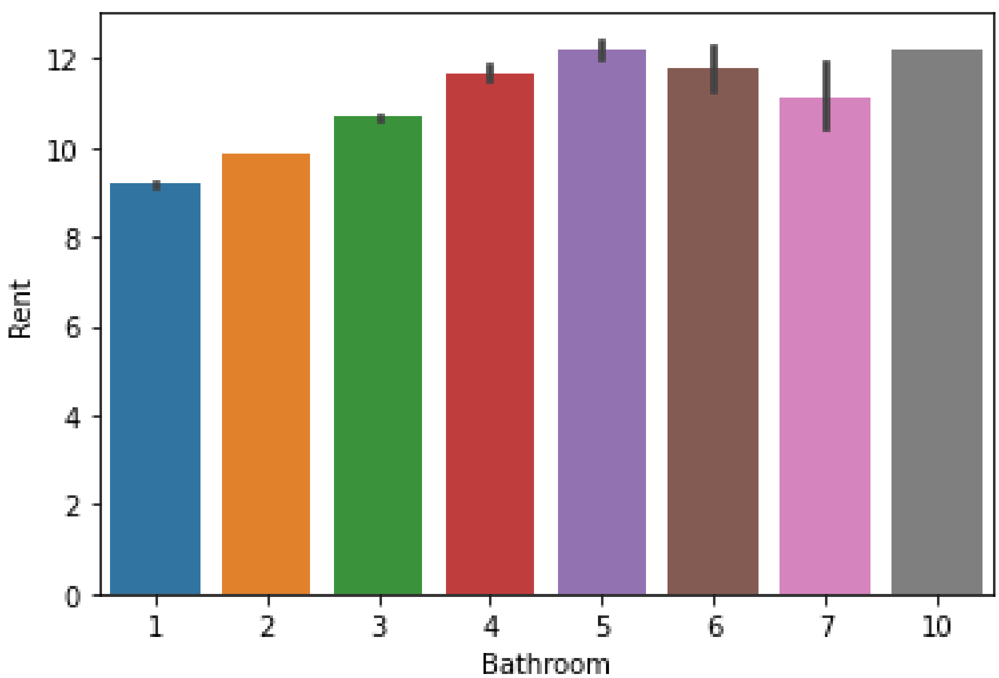
```
sns.countplot('Bathroom',data=final_df)
```

<AxesSubplot:xlabel='Bathroom', ylabel='count'>




```
sns.barplot('Bathroom','Rent',data=final_df)
```

```
<AxesSubplot:xlabel='Bathroom', ylabel='Rent'>
```




```
#Analyzing the Point of Contact
final_df['Point of Contact'].value_counts()
```

```
Contact Owner      3216
Contact Agent      1529
Contact Builder       1
Name: Point of Contact, dtype: int64
```

```
sns.barplot('Point of Contact','Rent',data=final_df)
```

```
<AxesSubplot:xlabel='Point of Contact', ylabel='Rent'>
```

Data Cleaning

```
#Supressing the Outliers
final_df=final_df[(final_df['Rent']<14) & (final_df['Size']<5000)]
```

```
final_df.shape
```

```
(4740, 12)
```

Feature Engineering

```
#Splitting the Posted on Date into Day,Month and Year
final_df['Day']=final_df['Posted On'].str.split('-',expand=True)
[2].astype(int)
final_df['Month']=final_df['Posted On'].str.split('-',expand=True)
[1].astype(int)
final_df['Year']=final_df['Posted On'].str.split('-',expand=True)
[0].astype(int)
```

```
#Removing the Posted On column
final_df.drop('Posted On',axis=1,inplace=True)

#Converting the Categorical Variables into Numeric Form
final_df=pd.get_dummies(data=final_df,columns=['Area
Type','City','Furnishing Status','Bathroom','Tenant Preferred','Point
of Contact'])

final_df.head()
```

```
   BHK      Rent  Size            Floor            Area Locality  Day
Month  \
0    2  9.210440  1100  Ground out of 2                   Bandel   18
5
1    2  9.903538   800       1 out of 3  Phool Bagan, Kankurgachi   13
5
2    2  9.741027  1000       1 out of 3   Salt Lake City Sector 2   16
5
3    2  9.210440   800       1 out of 2              Dumdum Park    4
7
4    2  8.922792   850       1 out of 2            South Dum Dum    9
5

   Year  Area Type_Built Area  Area Type_Carpet Area  ...  Bathroom_4
\
0  2022                     0                      0  ...           0

1  2022                     0                      0  ...           0

2  2022                     0                      0  ...           0

3  2022                     0                      0  ...           0

4  2022                     0                      1  ...           0


   Bathroom_5  Bathroom_6  Bathroom_7  Tenant Preferred_Bachelors  \
0           0           0           0                           0
1           0           0           0                           0
2           0           0           0                           0
3           0           0           0                           0
4           0           0           0                           1

   Tenant Preferred_Bachelors/Family  Tenant Preferred_Family  \
0                                  1                        0
1                                  1                        0
2                                  1                        0
3                                  1                        0
4                                  0                        0
```

```
   Point of Contact_Contact Agent  Point of Contact_Contact Builder  \
0                               0                                  0
1                               0                                  0
2                               0                                  0
3                               0                                  0
4                               0                                  0

   Point of Contact_Contact Owner
0                               1
1                               1
2                               1
3                               1
4                               1

[5 rows x 33 columns]
```

```python
#Splitting the Floor into valuable Information
final_df['Living Floor']=final_df['Floor'].str.split('out of',expand=True)[0]
final_df['Total Floors']=final_df['Floor'].str.split('out of',expand=True)[1]

final_df['Living Floor'].value_counts()
```

```
1                  1159
2                   945
Ground              925
3                   511
4                   270
5                   164
6                    93
7                    74
10                   67
8                    66
9                    64
12                   47
11                   43
15                   41
14                   34
18                   26
Upper Basement       23
17                   22
16                   21
19                   16
13                   15
20                   12
25                   12
Lower Basement       10
23                    9
21                    6
24                    5
```

```
30                      5
34                      4
28                      4
27                      3
35                      3
22                      3
32                      3
26                      3
65                      3
60                      3
40                      2
36                      2
48                      2
45                      2
53                      2
1                       2
44                      1
41                      1
46                      1
33                      1
37                      1
50                      1
39                      1
43                      1
29                      1
49                      1
47                      1
76                      1
3                       1
Ground                  1
Name: Living Floor, dtype: int64
```

```
final_df[final_df['Living Floor']=='Ground']=0
final_df[final_df['Living Floor']=='Ground ']=0
final_df[final_df['Living Floor']=='Upper Basement ']=1
final_df[final_df['Living Floor']=='Lower Basement ']=0

final_df[final_df['Living Floor']=='Upper Basement']=1
final_df[final_df['Living Floor']=='Lower Basement']=0

final_df['Living Floor']=final_df['Living Floor'].astype(int)

final_df['Total Floors'].value_counts()
```

```
0      936
 4     819
 3     701
 2     511
 5     392
      ...
 66      1
```

```
 85      1
 71      1
 81      1
 39      1
Name: Total Floors, Length: 68, dtype: int64
```

```python
for i in final_df[final_df['Total Floors'].isna()]['Living Floor'].index:
    final_df.loc[i,'Total Floors']=final_df.loc[i,'Living Floor']

final_df['Total Floors']=final_df['Total Floors'].astype(int)

#Remove the Floor Feature
final_df.drop('Floor',axis=1,inplace=True)
```

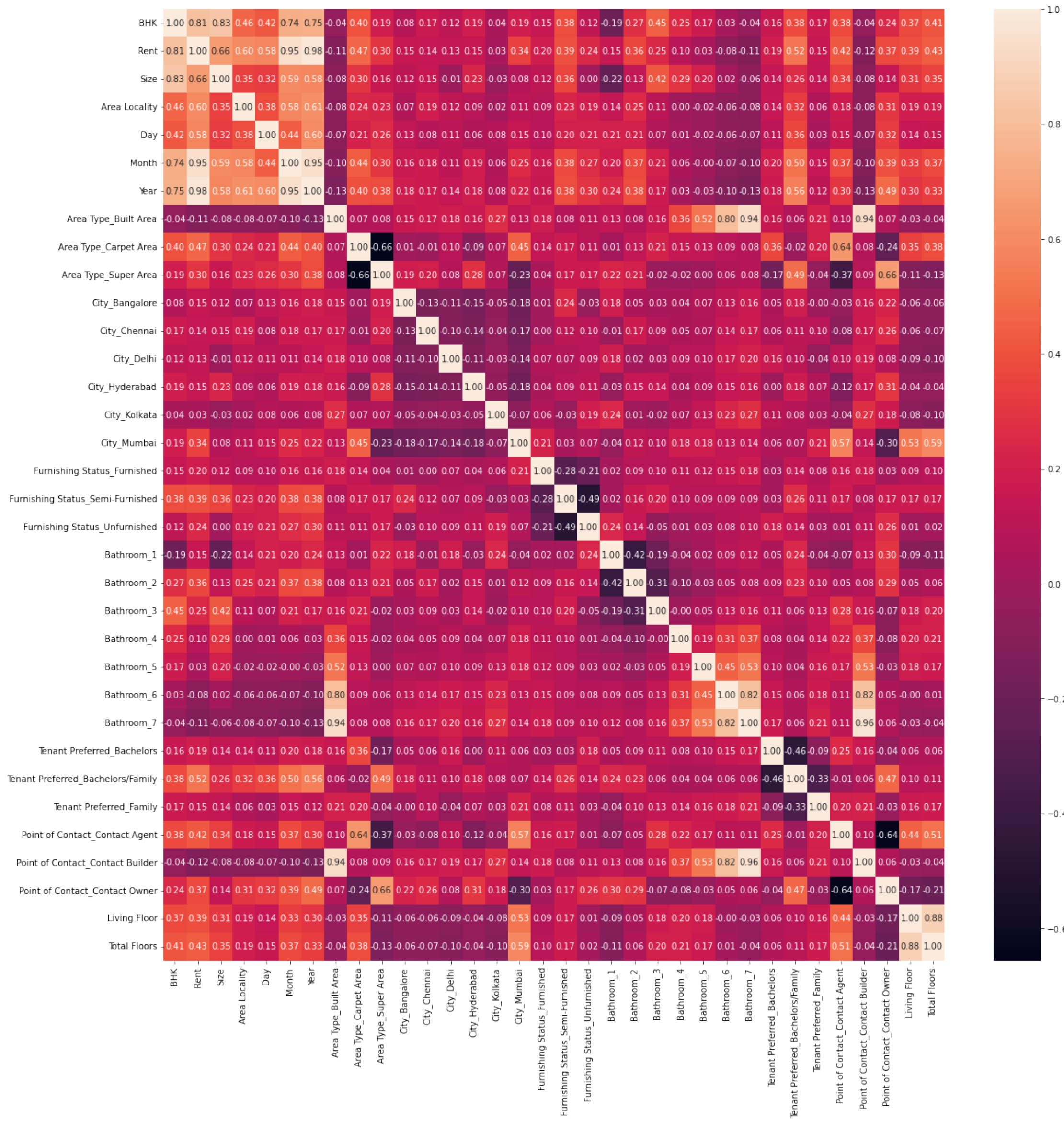
Encoding

```python
#Applying HotLabelEncoder on Area Locality
le=LabelEncoder()
final_df['Area Locality']=le.fit_transform(final_df['Area Locality'].astype(str))

final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4740 entries, 0 to 4745
Data columns (total 34 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   BHK                                4740 non-null   int64  
 1   Rent                               4740 non-null   float64
 2   Size                               4740 non-null   int64  
 3   Area Locality                      4740 non-null   int64  
 4   Day                                4740 non-null   int64  
 5   Month                              4740 non-null   int64  
 6   Year                               4740 non-null   int64  
 7   Area Type_Built Area               4740 non-null   uint8  
 8   Area Type_Carpet Area              4740 non-null   uint8  
 9   Area Type_Super Area               4740 non-null   uint8  
 10  City_Bangalore                     4740 non-null   uint8  
 11  City_Chennai                       4740 non-null   uint8  
 12  City_Delhi                         4740 non-null   uint8  
 13  City_Hyderabad                     4740 non-null   uint8  
 14  City_Kolkata                       4740 non-null   uint8  
 15  City_Mumbai                        4740 non-null   uint8  
 16  Furnishing Status_Furnished        4740 non-null   uint8  
 17  Furnishing Status_Semi-Furnished   4740 non-null   uint8  
 18  Furnishing Status_Unfurnished      4740 non-null   uint8  
 19  Bathroom_1                         4740 non-null   uint8  
 20  Bathroom_2                         4740 non-null   uint8  
 21  Bathroom_3                         4740 non-null   uint8  
 22  Bathroom_4                         4740 non-null   uint8  
```

```
 23  Bathroom_5                            4740 non-null   uint8
 24  Bathroom_6                            4740 non-null   uint8
 25  Bathroom_7                            4740 non-null   uint8
 26  Tenant Preferred_Bachelors            4740 non-null   uint8
 27  Tenant Preferred_Bachelors/Family     4740 non-null   uint8
 28  Tenant Preferred_Family               4740 non-null   uint8
 29  Point of Contact_Contact Agent        4740 non-null   uint8
 30  Point of Contact_Contact Builder      4740 non-null   uint8
 31  Point of Contact_Contact Owner        4740 non-null   uint8
 32  Living Floor                          4740 non-null   int64
 33  Total Floors                          4740 non-null   int64
dtypes: float64(1), int64(8), uint8(25)
memory usage: 615.1 KB
```

```
#Correlation between all Features
plt.figure(figsize=(20,20))
sns.heatmap(final_df.corr(),annot=True,fmt='.2f')
plt.show()
```

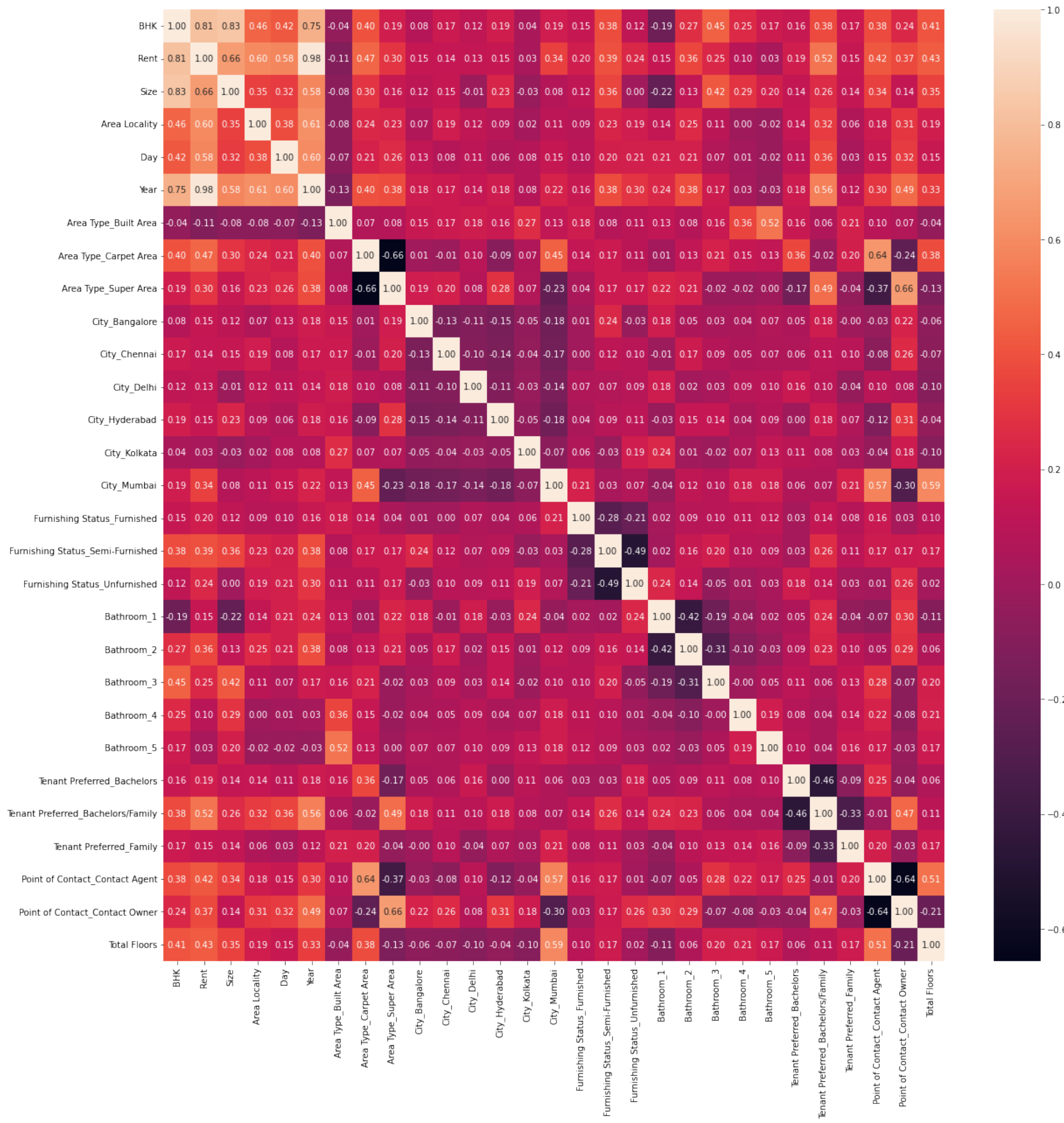


```python
#Dropping these Features in order to Reduce the Problem of Multi-Collinearity
final_df.drop('Living Floor',axis=1,inplace=True)
final_df.drop('Point of Contact_Contact Builder',axis=1,inplace=True)
final_df.drop('Bathroom_6',axis=1,inplace=True)
final_df.drop('Bathroom_7',axis=1,inplace=True)
final_df.drop('Month',axis=1,inplace=True)
```

A heat map is a data visualization technique that shows magnitude of a phenomenon as color in two dimensions. The variation in color may be by hue or intensity, giving obvious visual cues to the reader about how the phenomenon is clustered or varies over space. It is an excellent way to show and analyse correlations among various variables.

> If the value is 1, then there is a positive correlation. If the value is 0, then there is no correlation. If the value is -1, there is negetive correlation.

```
#Correlation between all Features
plt.figure(figsize=(20,20))
sns.heatmap(final_df.corr(),annot=True,fmt='.2f')
plt.show()
```



We can see that there is a high correlation in the diagonal elements of the matrix coz they represent the same attributes. There is high positive correlation of 0.98 between Year and Rent. There is high negetive correlation of -0.66 between the Area types Super Area and Carpet Area. There is negetive correlation of -0.64 between the point of contacts OWner and Agent.

Splitting of Train and Test Data

```
X=final_df.drop('Rent',axis=1)
y=final_df['Rent']
# Train-Test Split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,rando
m_state=42)
ss=StandardScaler()
X_train=ss.fit_transform(X_train)
X_test=ss.transform(X_test)
```

Modelling

A regression is a statistical technique that relates a dependent variable to one or more independent (explanatory) variables. A regression model is able to show whether changes observed in the dependent variable are associated with changes in one or more of the explanatory variables. There are many regression models and we have used a few common ones that may be apt to our dataset.

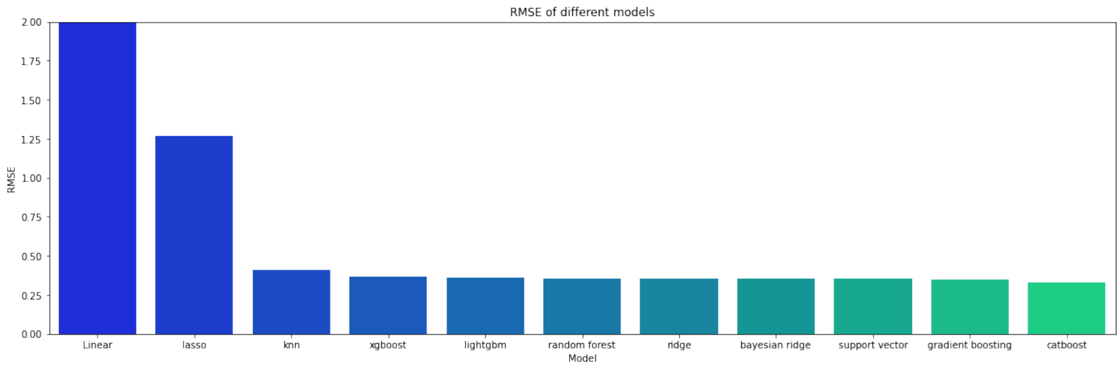
```
models = {
    'Linear' : LinearRegression(),
    'ridge' : Ridge(),
    'xgboost' : XGBRegressor(),
    'catboost' : CatBoostRegressor(verbose=0),
    'lightgbm' : LGBMRegressor(),
    'gradient boosting' : GradientBoostingRegressor(),
    'lasso' : Lasso(),
    'random forest' : RandomForestRegressor(),
    'bayesian ridge' : BayesianRidge(),
    'support vector': SVR(),
    'knn' : KNeighborsRegressor(n_neighbors = 4)
}
```

```
#Training Different Models
for name, model in models.items():
    model.fit(X_train, y_train)
    print(f'{name} trained')
```

```
Linear trained
ridge trained
xgboost trained
catboost trained
lightgbm trained
gradient boosting trained
lasso trained
random forest trained
bayesian ridge trained
support vector trained
knn trained
```

Evaluating the Models

```
results = {}
kf = KFold(n_splits= 10)

for name, model in models.items():
    result = np.mean(np.sqrt(-cross_val_score(model, X_train,
y_train,scoring='neg_mean_squared_error', cv= kf)))
    results[name] = result

for name, result in results.items():
    print(f"{name} : {round(result, 3)}")

Linear : 14085033618.791
ridge : 0.354
xgboost : 0.37
catboost : 0.331
lightgbm : 0.358
gradient boosting : 0.349
lasso : 1.268
random forest : 0.355
bayesian ridge : 0.354
support vector : 0.352
knn : 0.409

results_df = pd.DataFrame(results,
index=range(0,1)).T.rename(columns={0: 'RMSE'}).sort_values('RMSE',
ascending=False)
results_df.T

            Linear     lasso       knn   xgboost  lightgbm  random
forest  \
RMSE  1.408503e+10  1.268299  0.408553  0.369527  0.357817
0.355119

         ridge  bayesian ridge  support vector  gradient boosting
catboost
RMSE  0.354354        0.354275        0.351989           0.349207
0.331262

plt.figure(figsize = (20, 6))
sns.barplot(x= results_df.index, y = results_df['RMSE'], palette =
'winter')
plt.ylim(0,2)
plt.xlabel('Model')
plt.ylabel('RMSE')
plt.title('RMSE of different models');
```



We have used RMSE because Root Mean Square Error (RMSE) is a standard way to measure the error of a model in predicting quantitative data.This tells us heuristically that RMSE can be thought of as some kind of (normalized) distance between the vector of predicted values and the vector of observed values which clearly gives us the error difference required.

Here we find that, Linear and lasso regression models have very high rmse values compared to the other models. Light gbm has comparitively lesser error and greater accuracy and hence, it is better to use that model amongst all.

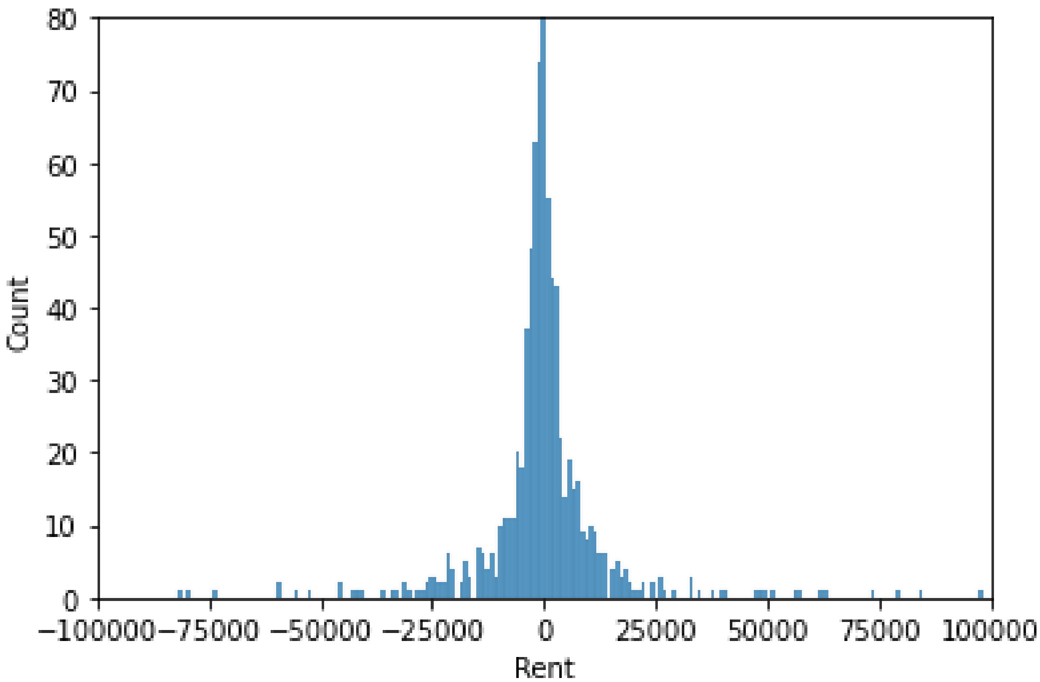
```
#Defining the Method in order to get the r2_score of each Model on Test Data and Based
# on that,We will select our Model

def prediction(model,X_train,y_train,X_test,y_test):
    model.fit(X_train,y_train)
    pred_data=np.exp(model.predict(X_test))
    return r2_score(np.exp(y_test),pred_data)

for name,model in models.items():
    score=prediction(model,X_train,y_train,X_test,y_test)
    print(f'{name} r2_score is {score}')
```

```
Linear r2_score is 0.667356035807981
ridge r2_score is 0.6687001695717816
xgboost r2_score is 0.6782093439768724
catboost r2_score is 0.6699485536735215
lightgbm r2_score is 0.7130113622736785
gradient boosting r2_score is 0.6799177407919296
lasso r2_score is -0.03677438959812562
random forest r2_score is 0.6830999664385836
bayesian ridge r2_score is 0.6689207974172564
support vector r2_score is 0.6502212669411573
knn r2_score is 0.5753463137414174
```

Light Gradient Boosting

*Lightgbm is giving the r2_score of more than **0.71** Hence,We Select the **LightGBM** .* Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks

```
#Error of the Predicted Y with the True Y
model=LGBMRegressor()
model.fit(X_train,y_train)
pred_y=np.exp(model.predict(X_test))
sns.histplot(np.exp(y_test)-pred_y)
plt.xlim(-100000,100000)
plt.ylim(0,80)
plt.show()
```



Applying Voting Ensemble on the 3 Models

```
from sklearn.ensemble import VotingRegressor
estimators=[('rf',RandomForestRegressor()),
('CB',CatBoostRegressor(verbose=0)),('lgbm',LGBMRegressor())]
vc=VotingRegressor(estimators=estimators)
x=cross_val_score(vc,X_train,y_train,cv=10,scoring='r2')
print(np.round(np.mean(x),2))
```

```
0.99
```

```
#Fitting the Model
vc.fit(X_train,y_train)
```

```
VotingRegressor(estimators=[('rf', RandomForestRegressor()),
                            ('CB',
                             <catboost.core.CatBoostRegressor object
at 0x7fe0d4716150>),
                            ('lgbm', LGBMRegressor())])
```

```
pred_y=vc.predict(X_test)
print(r2_score(np.exp(y_test),np.exp(pred_y)))
```

```
0.7038617738995336
```

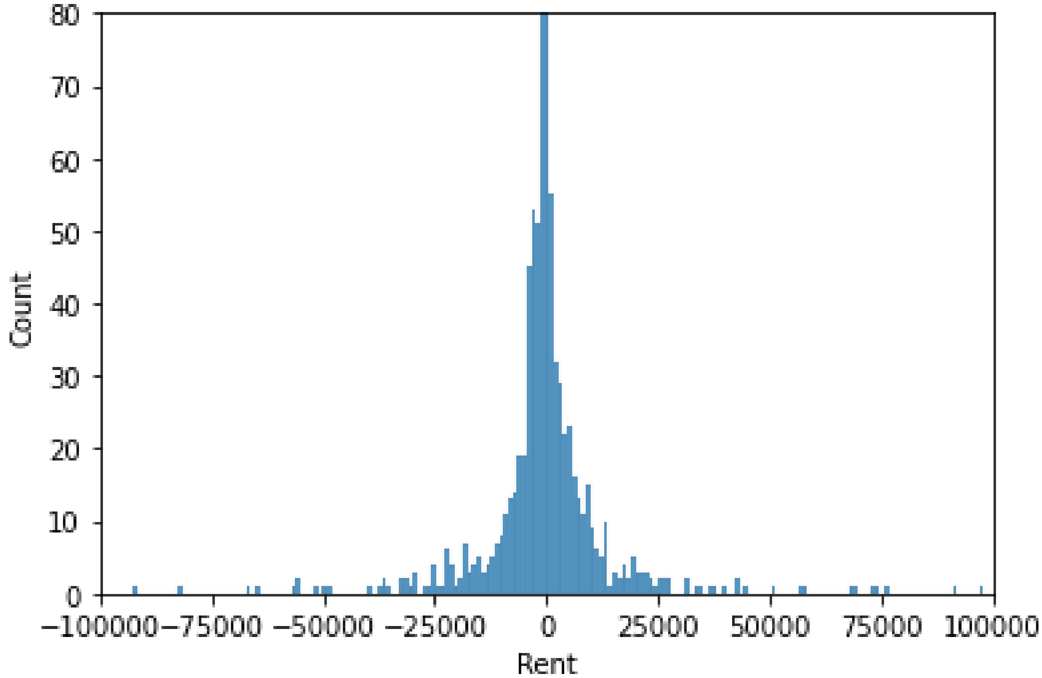
Working on Light Gradient Boosting

A gradient boosting model has variety of parameters which can be tuned to get better accuracy and results. LightGBM uses the leaf-wise tree growth algorithm, while many other popular tools use depth-wise tree growth. Compared with depth-wise growth, the leaf-wise algorithm can converge much faster. However, the leaf-wise growth may be over-fitting if not used with the appropriate parameters.

Parameters

> num_leaves: This is the main parameter to control the complexity of the tree model. min_data_in_leaf: This is a very important parameter to prevent over-fitting in a leaf-wise tree. max_depth: To limit the tree depth explicitly.

```
model=LGBMRegressor(boosting_type='gbdt', num_leaves=10, max_depth=-1,
learning_rate=0.1, n_estimators=100)
model.fit(X_train,y_train)
pred_y=np.exp(model.predict(X_test))
sns.histplot(np.exp(y_test)-pred_y)
plt.xlim(-100000,100000)
plt.ylim(0,80)
plt.show()
```



Even after tuning the parameters, there is no much differnence in the accuracy and the voting model. Therefore, this is the best model we can get.

Thank you