PuppalaSucharitha_Milestone4

October 30, 2022

0.0.1 WEEK 9

0.0.2 Assignament 9.2 - Project Milestone 4 - Finalizing Your Results.

0.0.3 Puppala Sucharitha

0.0.4 Date : 10/30/2022

0.1 Project Milestone - 3 : Preliminary Analysis.

0.1.1 1. Will I be able to answer the questions I want to answer with the data I have?

Yes, the questions framed for the project can be answered with the data set I have and remain the same. The dataset is having no null values and duplicate values; and the data remains same as the original dataset which will be enough for the further analysis.

0.1.2 2. What visualizations are especially useful for explaining my data?

For the exploratory data analysis part I have selected Bar plots, Histograms, Boxplot, Pair plot and Correlation coefficient heat map are used which helps in explaining the dataset varibales. Histograms are used for the understanding of the numerical variables of the dataset. Bar plots for the categorical variables and the distribution of the features of the dataset with the target variable. Pair plot and correlation matrix heat map are plotted to understand the correlation between the target variable and the other features of the data set.

0.1.3 3. Do I need to adjust the data and/or driving questions?

Yes, I need to do few changes in the data that helps in arriving at the problem statement, but the questions framed for the project remains the same. The data seems to be good in not having null values or duplicated values, but during the analysis part some of the unwanted variables are to be dropped from the dataset which helps in efficient analysis. In this regard the column "Posted on" can be dropped from the dataset as the dataset contains the data related to the year 2022. The "Floor" variable is to be divided into two columns as Rent Floor and Total Number of Floors for more efficient analysis."Area Locality" column is to be dropped as there are many unique values. The dummy variables are to be created for the categorical variables.

0.1.4 4. Do I need to adjust my model/evaluation choices?

With the Preliminary Analysis of the data, the models/ evaluation choices remain the same. As the problem addressed involves supervised learning with continuous target variable, the models selected remains same and need not be adjusted.

0.1.5 5. Are my original expectations still reasonable?

After the preliminary analysis and with the help of visualizations I can say that the original expectations are still reasonable.

0.2 About the dataset.

The following are the details of the variables in the HOUSE RENT DATASET.

```
* BHK       : Gives the Number of Bedrooms, Hall, Kitchen.(i.e. 2 BHK means 2 Bedrooms, Hall and
* Rent      : Give the amount of Rent of the Houses/Apartments/Flats.
* Size      : Gives the Size of the Houses/Apartments/Flats in Square Feet.
* Floor     : Gives the idea of Houses/Apartments/Flats situated in which Floor and Total Number
* Area Type: Gives the Size of the Houses/Apartments/Flats calculated on either Super Area or C
* Area Locality: Gives Locality of the Houses/Apartments/Flats.
* City: Gives the City where the Houses/Apartments/Flats are Located.
* Furnishing Status: Gives the Furnishing Status of the Houses/Apartments/Flats, either it is I
* Tenant Preferred: Gives the Type of Tenant Preferred i.e. by the Owner or Agent.
* Bathroom: Gives the Number of Bathrooms present in the houses available for rent.
* Point of Contact: Gives Whom should you contact for more information regarding the Houses/Apa
```

0.3 Importing the necessary Libraries

[1]:
```
# Importing the required libraries.
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import scipy
from scipy.stats import skew
from matplotlib import pyplot
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

0.4 Loading the dataset.

[2]:
```
# Loading the dataset.
rentdf = pd.read_csv('House_Rent_Dataset.csv')
```

[3]:
```
# Getting the first five rows of the dataset.
rentdf.head()
```

[3]:

	Posted On	BHK	Rent	Size	Floor	Area Type	Area Locality	City	Furnishing Status	Tenant Preferred	Bathroom	Point of Contact
0	2022-05-18	2	10000	1100	Ground out of 2	Super Area	Bandel	Kolkata	Unfurnished	Bachelors/Family	2	Contact Owner
1	2022-05-13	2	20000	800	1 out of 3	Super Area	Phool Bagan, Kankurgachi	Kolkata	Semi-Furnished	Bachelors/Family	1	Contact Owner
2	2022-05-16	2	17000	1000	1 out of 3	Super Area	Salt Lake City Sector 2	Kolkata	Semi-Furnished	Bachelors/Family	1	Contact Owner
3	2022-07-04	2	10000	800	1 out of 2	Super Area	Dumdum Park	Kolkata	Unfurnished	Bachelors/Family	1	Contact Owner
4	2022-05-09	2	7500	850	1 out of 2	Carpet Area	South Dum Dum	Kolkata	Unfurnished	Bachelors	1	Contact Owner

[4]:
```
# Getting the size of the dataset.
rentdf.size
```

[4]: 56952

[5]:
```
# Let's check the shape of the dataset.
rentdf.shape
```

[5]: (4746, 12)

[6]:
```
# Checking if any missing values present in the dataset.
rentdf.isna().sum()
```

[6]:
```
Posted On            0
BHK                  0
Rent                 0
Size                 0
Floor                0
Area Type            0
```

```
Area Locality        0
City                 0
Furnishing Status    0
Tenant Preferred     0
Bathroom             0
Point of Contact     0
dtype: int64
```

[7]:
```
# Checking if any duplicate values are present in the dataset.
rentdf.duplicated().sum()
```

[7]:
```
0
```

[8]:
```
# Getting the type of each dataset variables.

rentdf.dtypes
```

[8]:
```
Posted On            object
BHK                   int64
Rent                  int64
Size                  int64
Floor                object
Area Type            object
Area Locality        object
City                 object
Furnishing Status    object
Tenant Preferred     object
Bathroom              int64
Point of Contact     object
dtype: object
```

[9]:
```
# Checking for the number of unique values present in the variables of the␣
↪dataset..

rentdf.nunique()
```

[9]:
```
Posted On              81
BHK                     6
Rent                  243
Size                  615
Floor                 480
Area Type               3
Area Locality        2235
City                    6
Furnishing Status       3
Tenant Preferred        3
Bathroom                8
```

```
Point of Contact        3
dtype: int64
```

0.4.1 Observations of the dataset:

- The dataset has 4746 rows and 12 columns.
- We can see that the dataset contains both numerical and categorical data.
- There are no null values in the dataset.
- Extracted the number of unique values of the dataset variables, and can see many unique values in 'Area Locality' column, 'Size', 'Floor' variables.
- The target variable is "Rent" from the dataset.
- During the modelling some of the categorical variables need to be converted to dummy variables.

0.5 Exploratory Data Analysis :

0.5.1 Splitting the necessary columns into two columns

[10]:
```
# Let's divide the 'Floor' column into two columns i.e. 'Renting_Floor' and␣
↪'No_of_Floors'
# Let's split the Floor column using 'lambda'
# First split the 'Renting_Floor'
rentdf["Renting_Floor"] = rentdf["Floor"].apply(lambda x: x.split(" out of␣
↪")[0])

# Now let's split the total no. of Floors as 'No_of_Floors'
rentdf["No_of_Floors"] = rentdf["Floor"].apply(lambda x: x.split(" ")[-1])
```

[11]:
```
# Let's check for the unique values in the 'Renting_Floor' column

rentdf["Renting_Floor"].unique()
```

[11]:
```
array(['Ground', '1', '2', '4', '3', '5', '7', '8', 'Upper Basement',
       '11', 'Lower Basement', '6', '14', '43', '13', '18', '17', '9',
       '19', '60', '34', '12', '26', '25', '53', '16', '10', '39', '32',
       '47', '28', '20', '15', '65', '40', '37', '22', '21', '30', '35',
       '33', '44', '41', '46', '27', '45', '48', '50', '24', '23', '29',
       '49', '36', '76'], dtype=object)
```

0.5.2 Observations:

- From the above output we can see floor numbers between 1 to 76 but few values are Ground, Upper Basement and Lower Basement values that are in string, so lets replace them with numericals,like Ground = 0, Upper Basement = -1 and Lower Basement = -2

[12]:
```
# Replacing the values Ground = 0, Upper Basement = -1 and Lower Basement = -2
```

```
rentdf["Renting_Floor"] = rentdf["Renting_Floor"].replace(["Ground", "Upper␣
↪Basement", "Lower Basement"],
                                                          [0, -1, -2]).
↪astype(int)
# Let's check the unique values in the 'Renting_Floor'
rentdf["Renting_Floor"].unique()
```

[12]:
```
array([ 0,  1,  2,  4,  3,  5,  7,  8, -1, 11, -2,  6, 14, 43, 13, 18, 17,
        9, 19, 60, 34, 12, 26, 25, 53, 16, 10, 39, 32, 47, 28, 20, 15, 65,
       40, 37, 22, 21, 30, 35, 33, 44, 41, 46, 27, 45, 48, 50, 24, 23, 29,
       49, 36, 76])
```

0.5.3 Observations:

- From the above output we can see that the Ground, Upper Basement and Lower Basement are being replaced with the numerical values.

[13]:
```
# Let's check for the unique values in the "No_of_Floors" column created

rentdf["No_of_Floors"].unique()
```

[13]:
```
array(['2', '3', '1', '4', '5', '14', '8', '6', '19', '10', '7', '13',
       '78', '18', '12', '24', '31', '21', '23', '20', '9', '22', '58',
       '16', '66', '48', '40', '44', '42', '41', '60', '32', '30', '29',
       '89', '15', '11', '28', '17', '45', '35', '75', '38', '51', '43',
       '25', '27', '26', '76', '36', '37', '55', '68', '77', '50', '59',
       '62', '39', '52', '54', '33', '46', '85', '71', '81', '34',
       'Ground'], dtype=object)
```

0.5.4 Observations:

- From the above we can see a string value ‘Ground’ , lets replace the value with ‘0’

[14]:
```
# Use the replace() to replace 'Ground' with '0'
rentdf["No_of_Floors"] = rentdf["No_of_Floors"].replace("Ground", 0).astype(int)

# Let's check the unique values in "No_of_Floors"
rentdf["No_of_Floors"].unique()
```

[14]:
```
array([ 2,  3,  1,  4,  5, 14,  8,  6, 19, 10,  7, 13, 78, 18, 12, 24, 31,
       21, 23, 20,  9, 22, 58, 16, 66, 48, 40, 44, 42, 41, 60, 32, 30, 29,
       89, 15, 11, 28, 17, 45, 35, 75, 38, 51, 43, 25, 27, 26, 76, 36, 37,
       55, 68, 77, 50, 59, 62, 39, 52, 54, 33, 46, 85, 71, 81, 34,  0])
```

0.5.5 Observations :

- We can see from the above that the ‘Ground’ is replaced with ‘0’

[15]:
```
# Let's get the shape of the dataset.
rentdf.shape
```

[15]:
```
(4746, 14)
```

0.5.6 Observations:

- All the duplicates are being removed from the dataset.

[16]:
```
# Let's get the statistical summary of the dataset.
# Here we get the summary of numerical varibales only.
rentdf.describe()
```

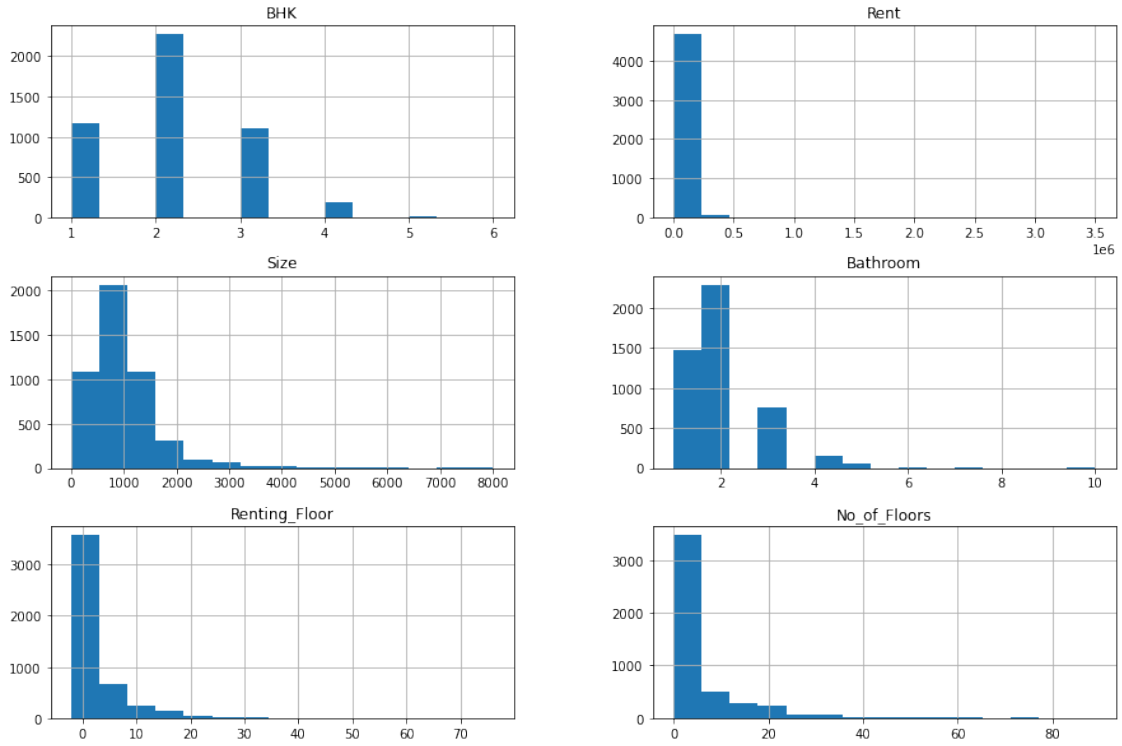
[16]:

	BHK	Rent	Size	Bathroom	Renting_Floor	No_of_Floors
count	4746.000000	4.746000e+03	4746.000000	4746.000000	4746.000000	4746.000000
mean	2.083860	3.499345e+04	967.490729	1.965866	3.436157	6.968605
std	0.832256	7.810641e+04	634.202328	0.884532	5.773950	9.467245
min	1.000000	1.200000e+03	10.000000	1.000000	-2.000000	0.000000
25%	2.000000	1.000000e+04	550.000000	1.000000	1.000000	2.000000
50%	2.000000	1.600000e+04	850.000000	2.000000	2.000000	4.000000
75%	3.000000	3.300000e+04	1200.000000	2.000000	3.000000	6.000000
max	6.000000	3.500000e+06	8000.000000	10.000000	76.000000	89.000000

0.5.7 Understanding the distribution of the numerical variables of the dataset.

[17]:
```
# Plotting the histograms of the numerical variables.
rentdf.hist(bins=15, figsize=(15, 10))
```

[17]:
```
array([[<AxesSubplot:title={'center':'BHK'}>,
        <AxesSubplot:title={'center':'Rent'}>],
       [<AxesSubplot:title={'center':'Size'}>,
        <AxesSubplot:title={'center':'Bathroom'}>],
       [<AxesSubplot:title={'center':'Renting_Floor'}>,
        <AxesSubplot:title={'center':'No_of_Floors'}>]], dtype=object)
```



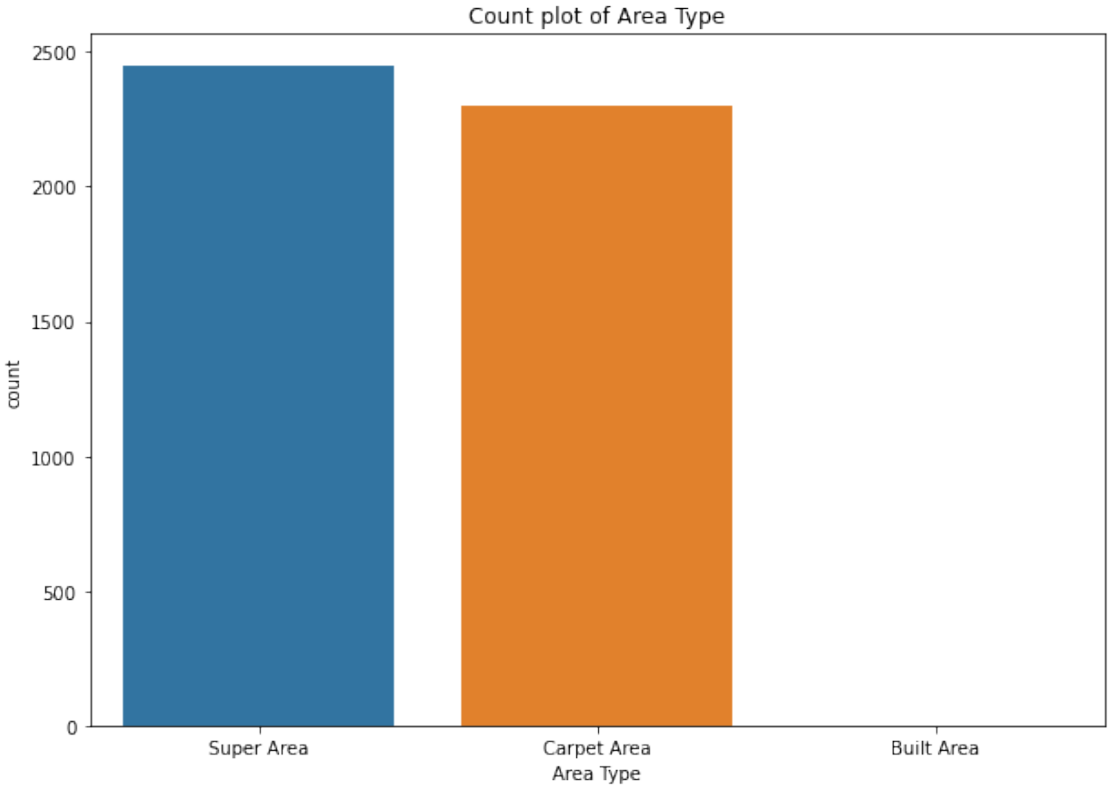
0.5.8 Observations:

- From the distribution of variable 'Rent', we can see that the distribution of rent is skewed towards right.
- From the distribution of variable 'BHK'(Bedroom, Hall and Kitchen), we can see that 2 BHK houses are more in number among other numbers for renting.
- From the distribution of variable 'Size', we can see that 500 to 1100 square feet houses are more for renting.
- From the distribution of variable 'Bathroom', we can see that 2 bathrooms are more in number for renting.
- From the distribution of variable 'Renting Floor', we can see that mostly ground, first,second, third floors are more in number for renting.
- From the distribution of variable 'No_of_Floors', we can see that upto 10 floors are more in number for renting

0.5.9 Let's understand the distribution of the categorical variables of the dataset.

[18]:
```
# Count plot for "Area Type"
# Set the size of the plot
plt.figure(figsize=(10,7))
# Countplot
sns.countplot(x=rentdf["Area Type"])
#Plot title
```

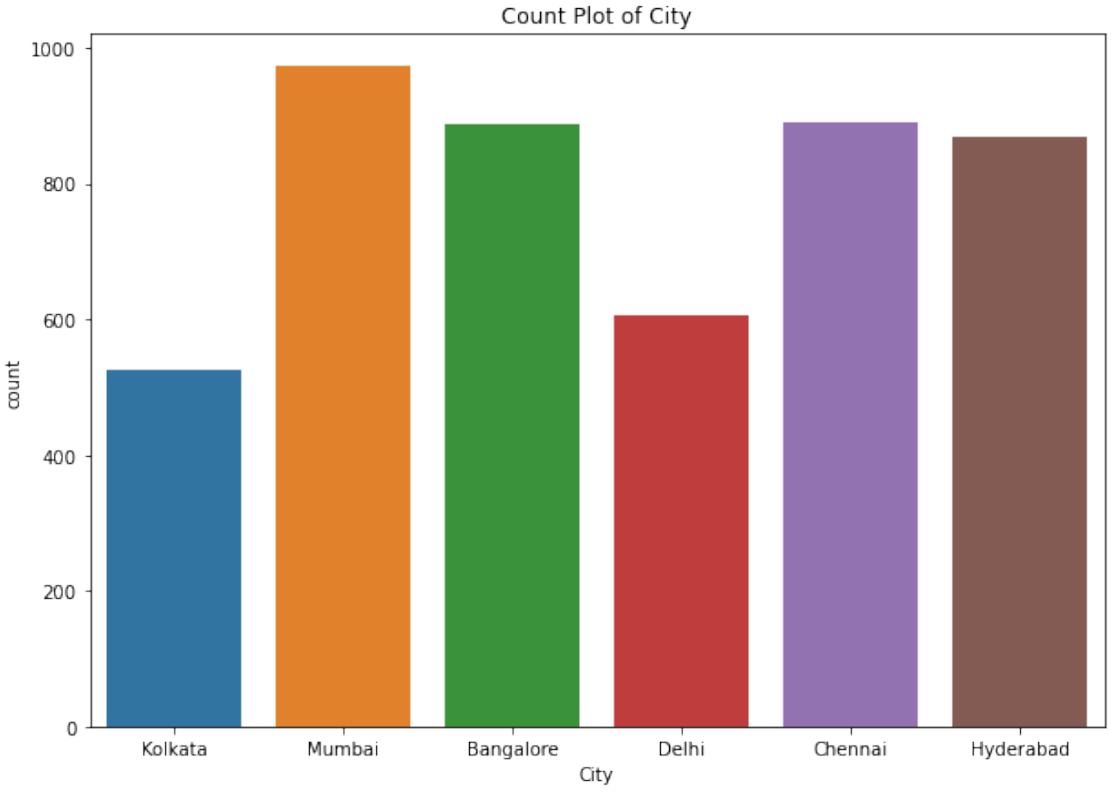
```
plt.title("Count plot of Area Type")
plt.show()
```



Observations:

- From the above count plot we can say that the houses available for rental are having Area Type as 'Super Area' more in number and next comes 'Carpet Area'

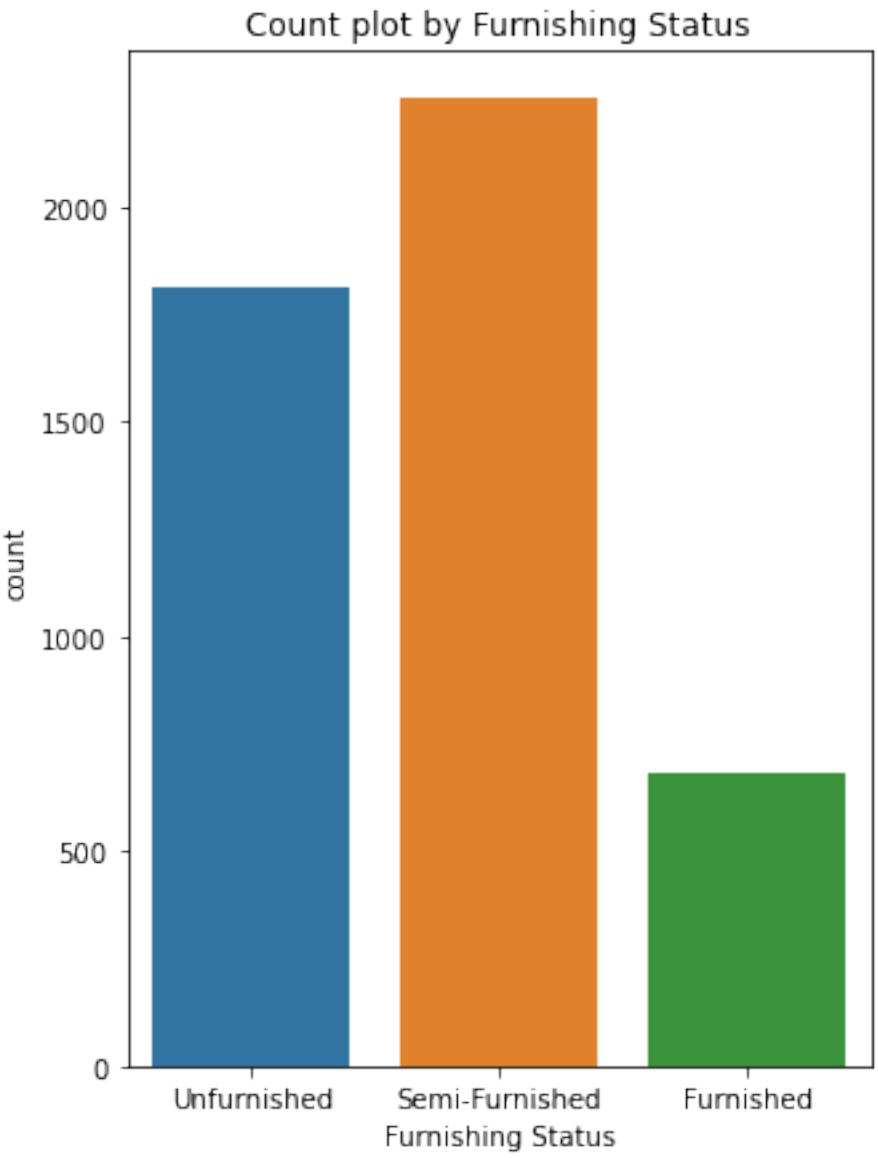
[19]:
```
# Count plot of the variable "City"
# Set the size of the plot.
plt.figure(figsize=(10,7))
# Count plot
sns.countplot(x=rentdf["City"])
# plot title
plt.title("Count Plot of City")
plt.show()
```



Observations:

- From the above count plot it is clear that 'Mumbai' city is having more number of rental houses available for renting.

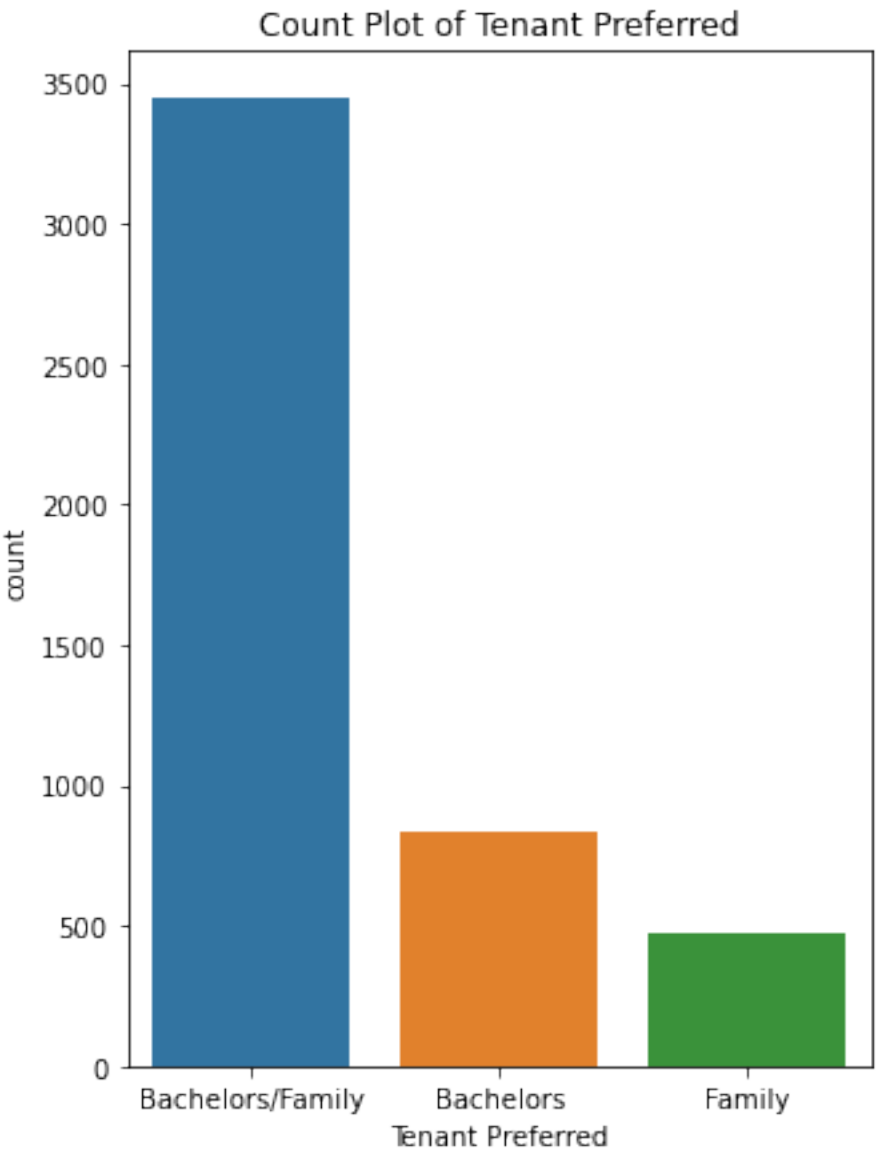
[20]:
```
# Count plot of the variable "Furishing status"
# Set the size of the plot
plt.figure(figsize=(5,7))
# Countplot
sns.countplot(x=rentdf["Furnishing Status"])
# Plot title.
plt.title("Count plot by Furnishing Status")
plt.show()
```



Observations:

- From the above count plot of 'Furnishing Status', we can see that most of the houses are Semi- Furnished that are available for renting.

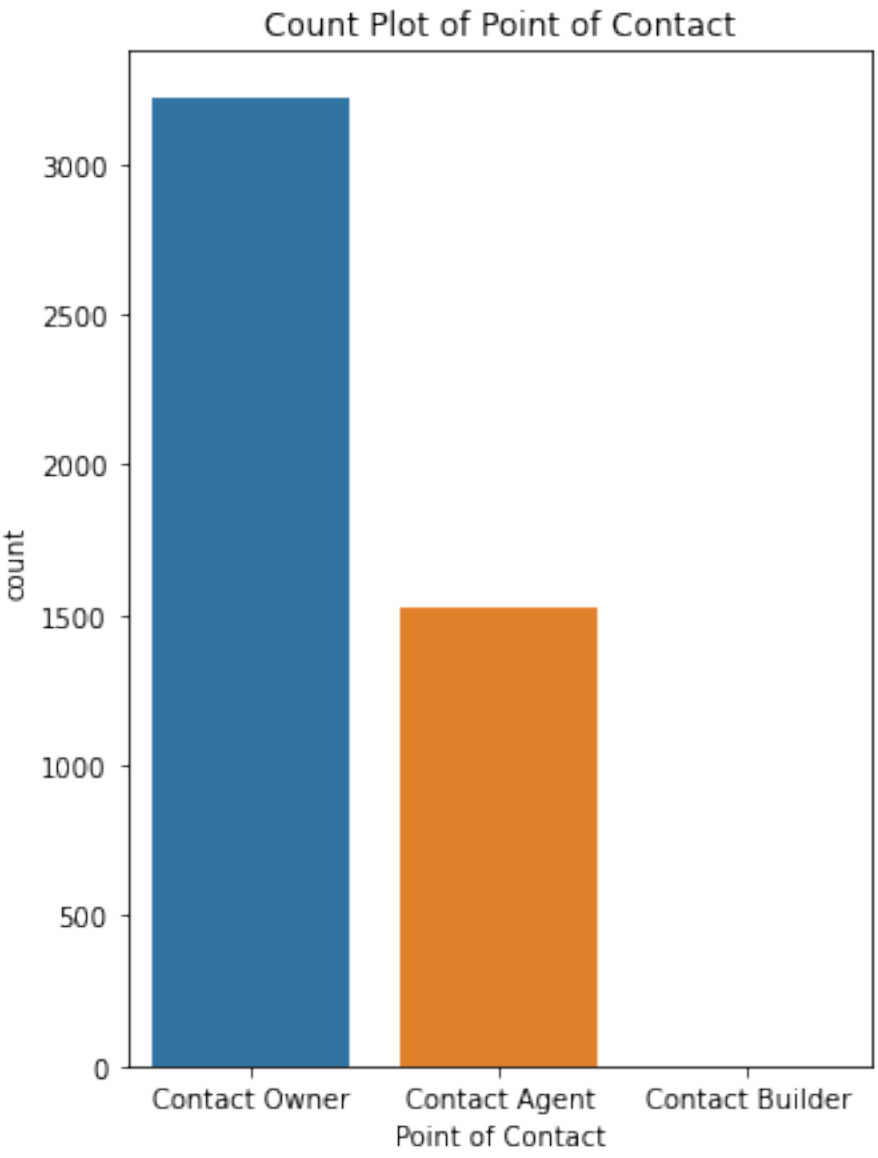
[21]:
```
# Count plot of the variable "Tenant Preferred"
plt.figure(figsize=(5,7))
# Count plot
sns.countplot(x=rentdf["Tenant Preferred"])
# Plot title.
plt.title("Count Plot of Tenant Preferred")
plt.show()
```



Observations:

- From the above plot we can see that Bachelors/ Family tenants are more preferred for renting houses.

[22]:
```
# Count plot of the variable " Point of Contact"
plt.figure(figsize=(5,7))
# Count plot
sns.countplot(x=rentdf["Point of Contact"])
# Plot title
plt.title("Count Plot of Point of Contact")
plt.show()
```



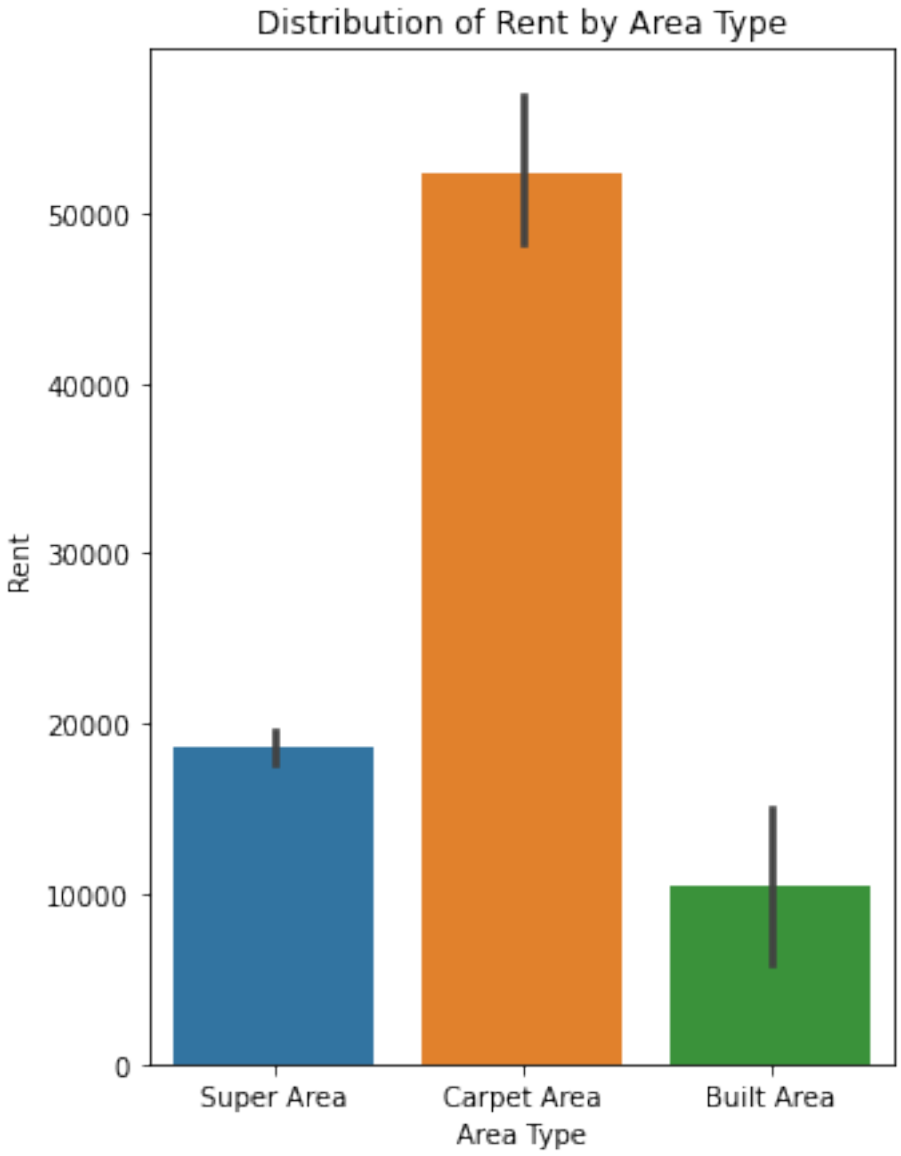
Observations:

- From the plot we can see that point of contact preferred is Contact Owner.

0.5.10 Let's understand the distribution of the categorical features with the target variable(Rent)

[23]:
```
# Plot the distribution of Rent by Area type
plt.figure(figsize=(5,7))
# Bar Plot
sns.barplot(data=rentdf, x="Area Type", y="Rent")
# Title of the plot.
```

```
plt.title("Distribution of Rent by Area Type")
plt.show()
```

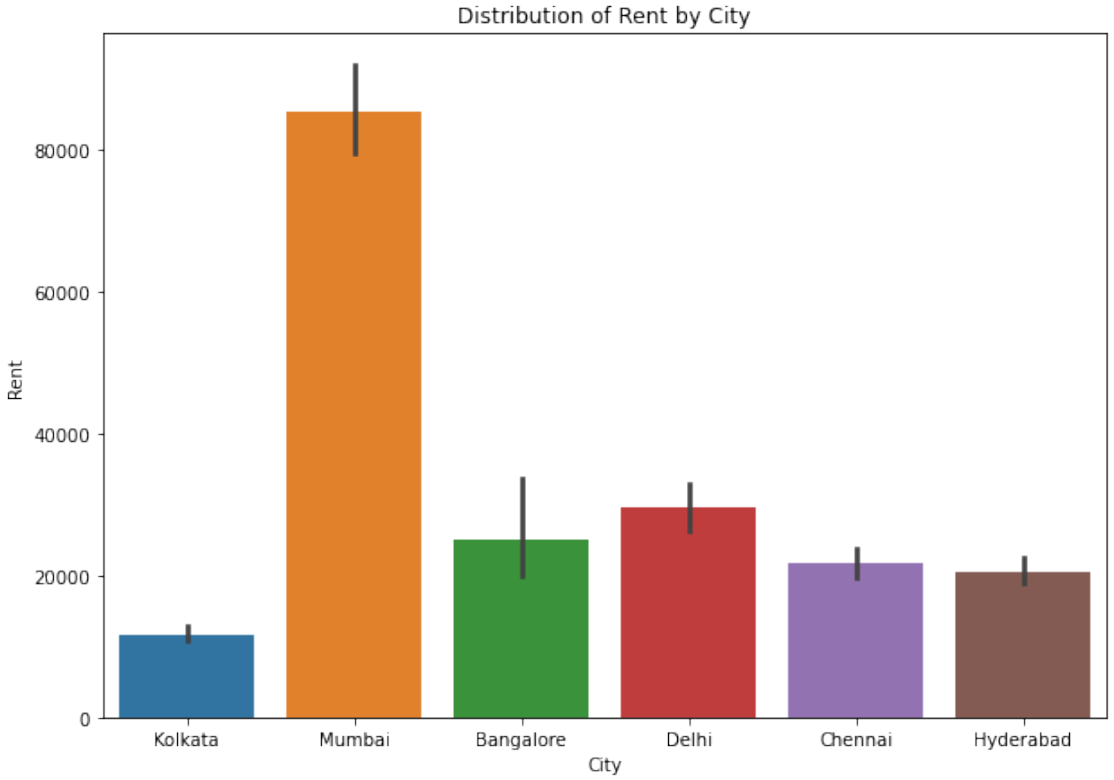


Observations:

- From the above plot we can see that the houses with Carpet Area are having high rents when compared the the Super Area and Built Area.

[24]:
```
# Plot the distribution of Rent by City
plt.figure(figsize=(10,7))
# Bar Plot
```

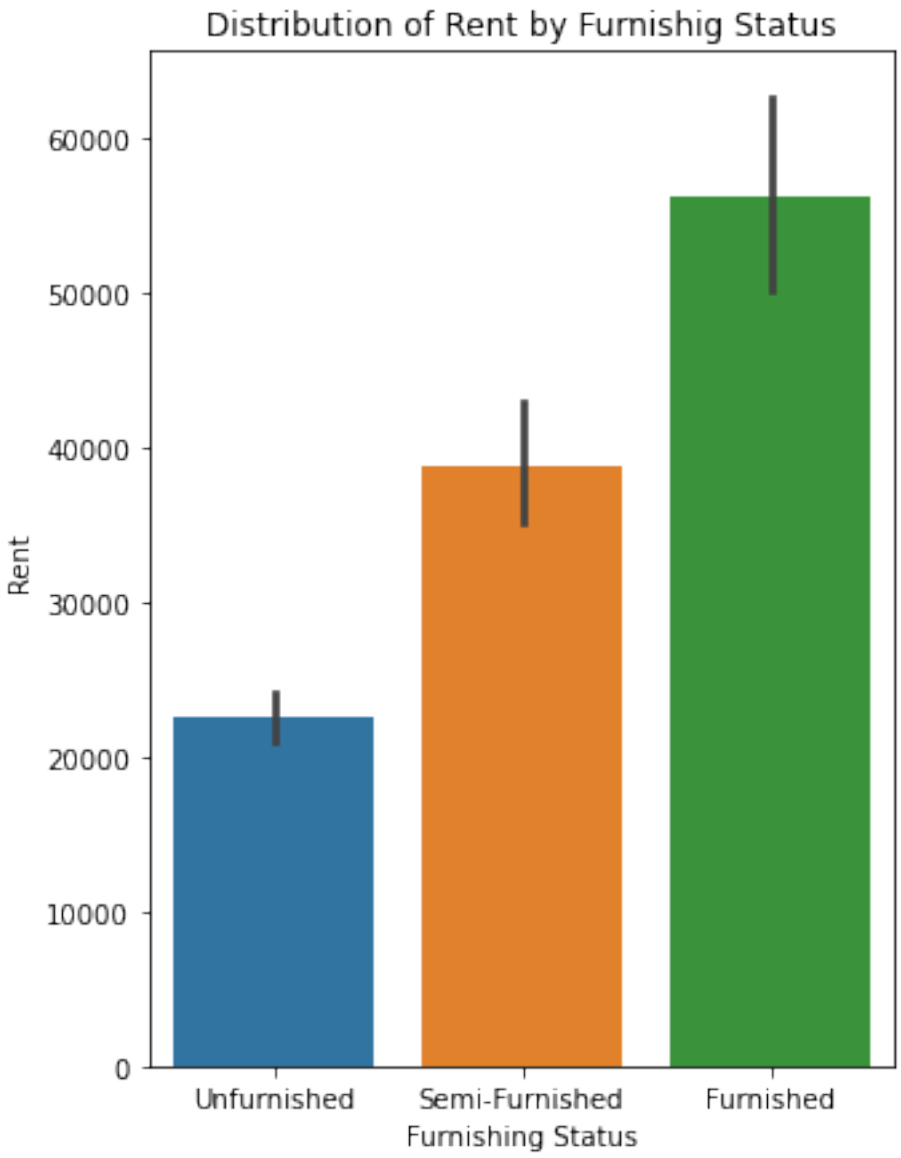
```
sns.barplot(data=rentdf, x="City", y="Rent")
# Title of the Plot
plt.title("Distribution of Rent by City")
plt.show()
```



Observations:

- From the above plot we can say that the Rents are high in MUmbai city when compared to other cities.

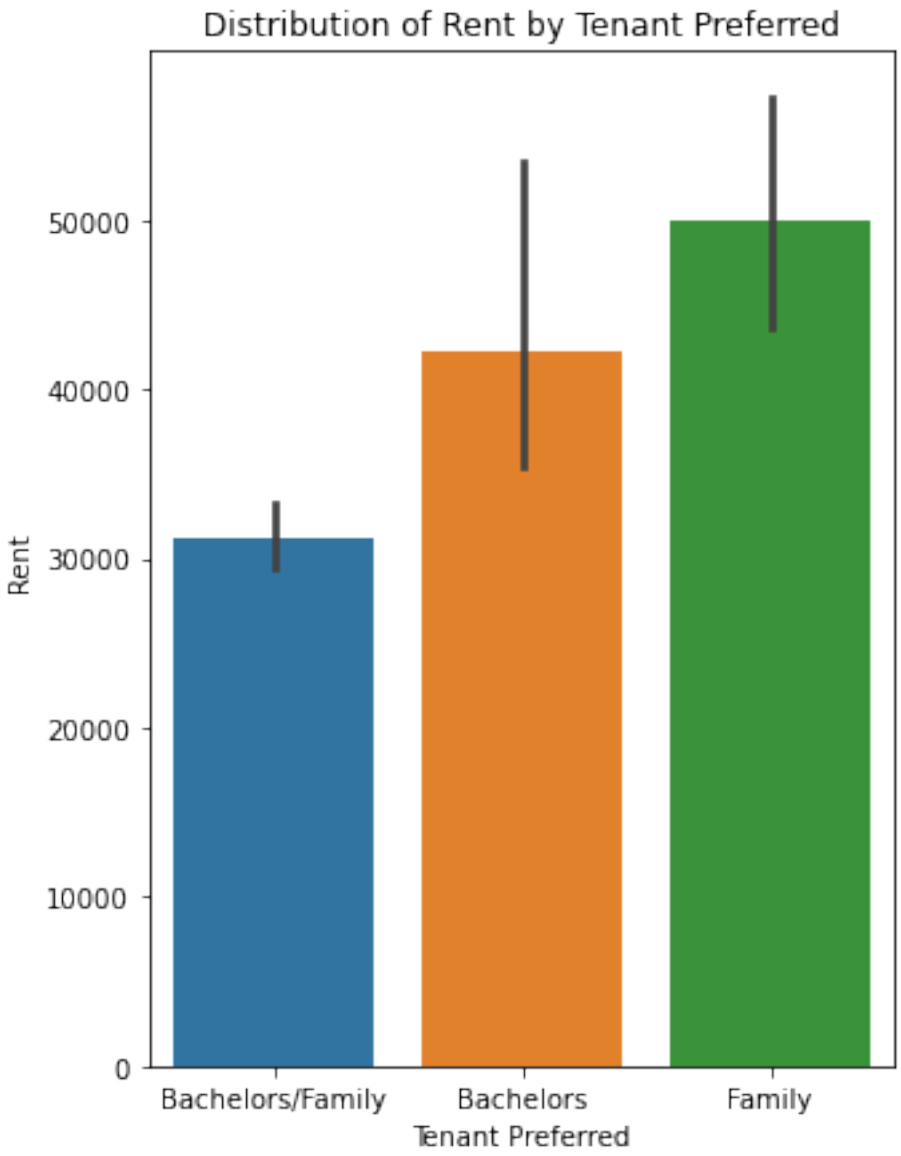
[25]:
```
# Plot the distribution of Rent by Furnishing Status
plt.figure(figsize=(5,7))
# Bar Plot
sns.barplot(data=rentdf, x="Furnishing Status", y="Rent")
# Title of the plot
plt.title("Distribution of Rent by Furnishig Status")
plt.show()
```



Observations:

- From the above we can say that Furnished houses are having more rents, next Semi- Furnished, and Unfurnished houses are having less rents i.e. below Rs.25000.

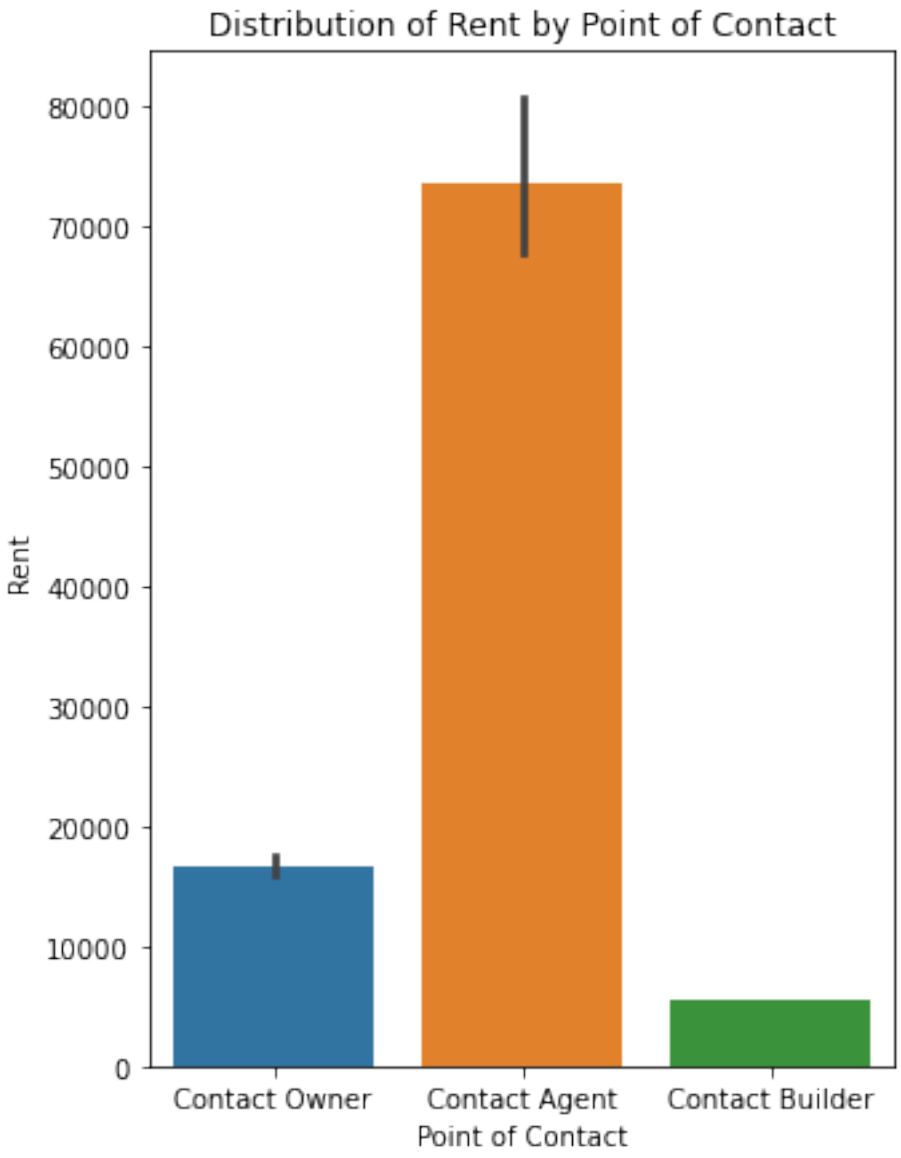
[26]:
```
# Plot the distribution of Rent by Tenant Preferred.
plt.figure(figsize=(5,7))
# Bar Plot
sns.barplot(data=rentdf, x="Tenant Preferred", y="Rent")
# Title pf the Plot
plt.title("Distribution of Rent by Tenant Preferred")
plt.show()
```



Observations:

- From the above plot we can see that rents are more for tenant preferred type Family.

[27]:
```
# Distribution of Rent by Point of Contact
plt.figure(figsize=(5,7))
# Bar Plot
sns.barplot(data=rentdf, x="Point of Contact", y="Rent")
# Title of the plot
plt.title("Distribution of Rent by Point of Contact")
plt.show()
```



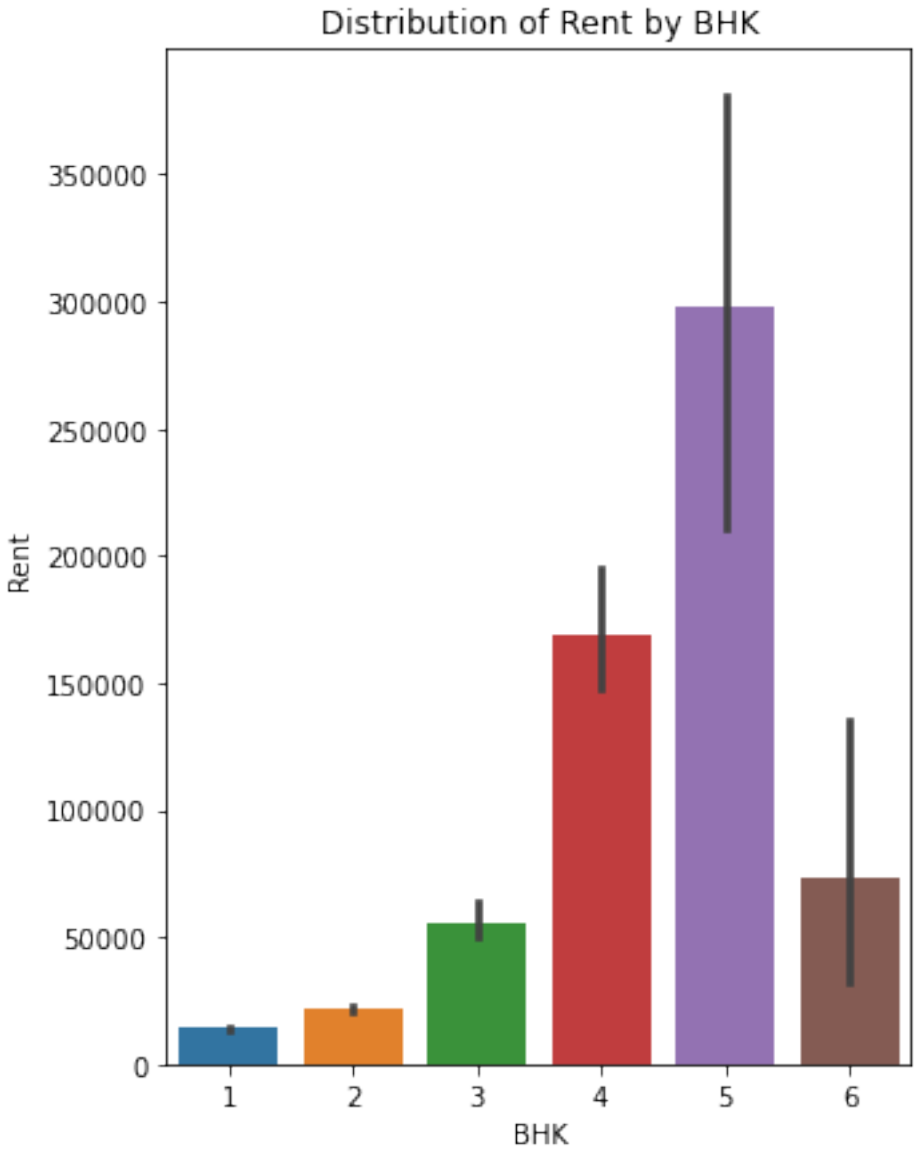
Observations:

- From the above plot we can see that the rents are more if the point of contact is by 'Contact Agent'.

0.5.11 Let's understand the distribution of the numerical variables of the dataset.

[28]:
```
# Distribution of Rent by BHK
plt.figure(figsize=(5,7))
# Bar Plot
sns.barplot(data=rentdf, x="BHK", y="Rent")
# Title of the plot
```

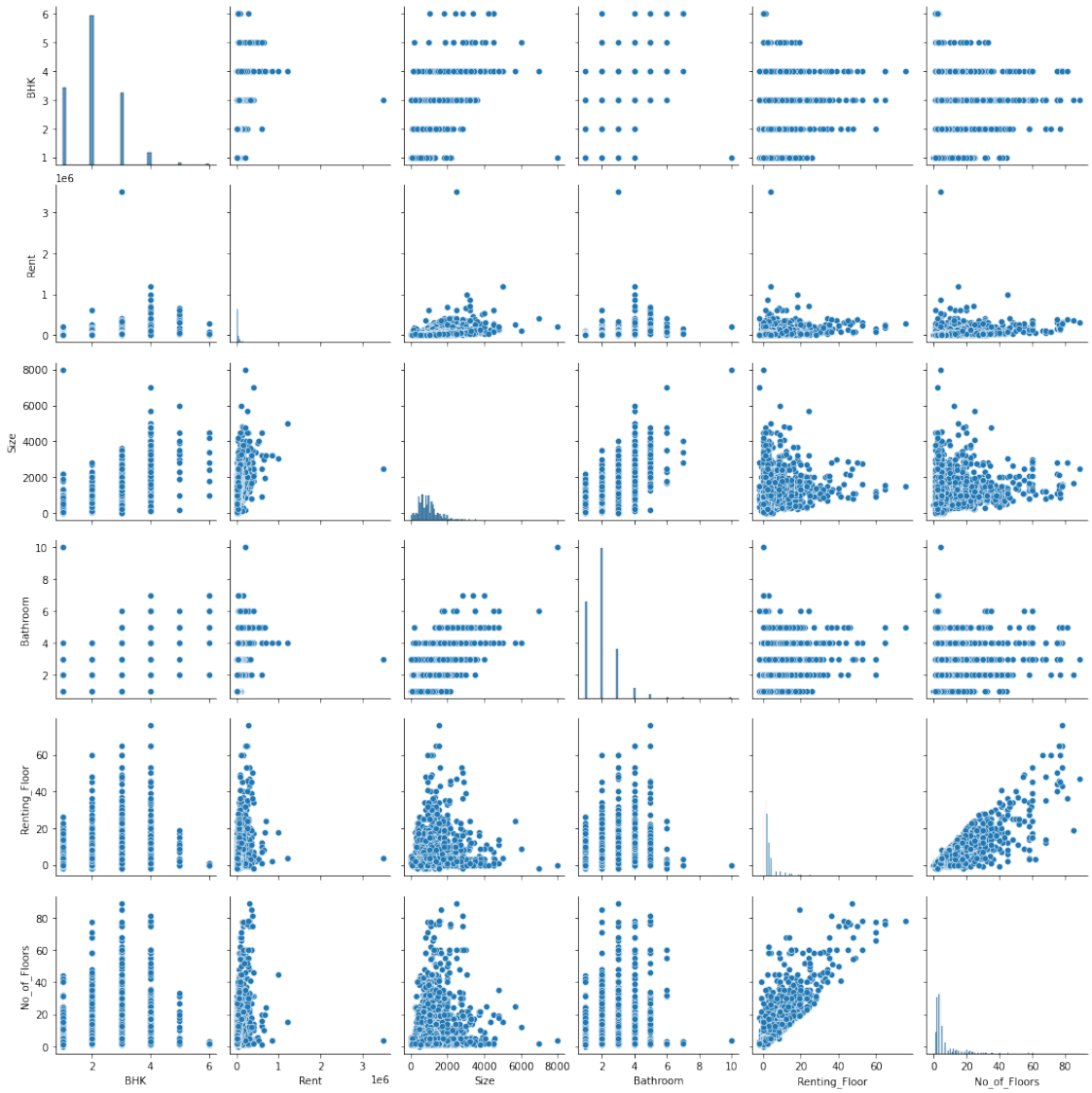
```
plt.title("Distribution of Rent by BHK")
plt.show()
```



0.5.12 Plotting the pair plot and correlation heat map for understanding the relation between all the variables in the data set and the target variable.

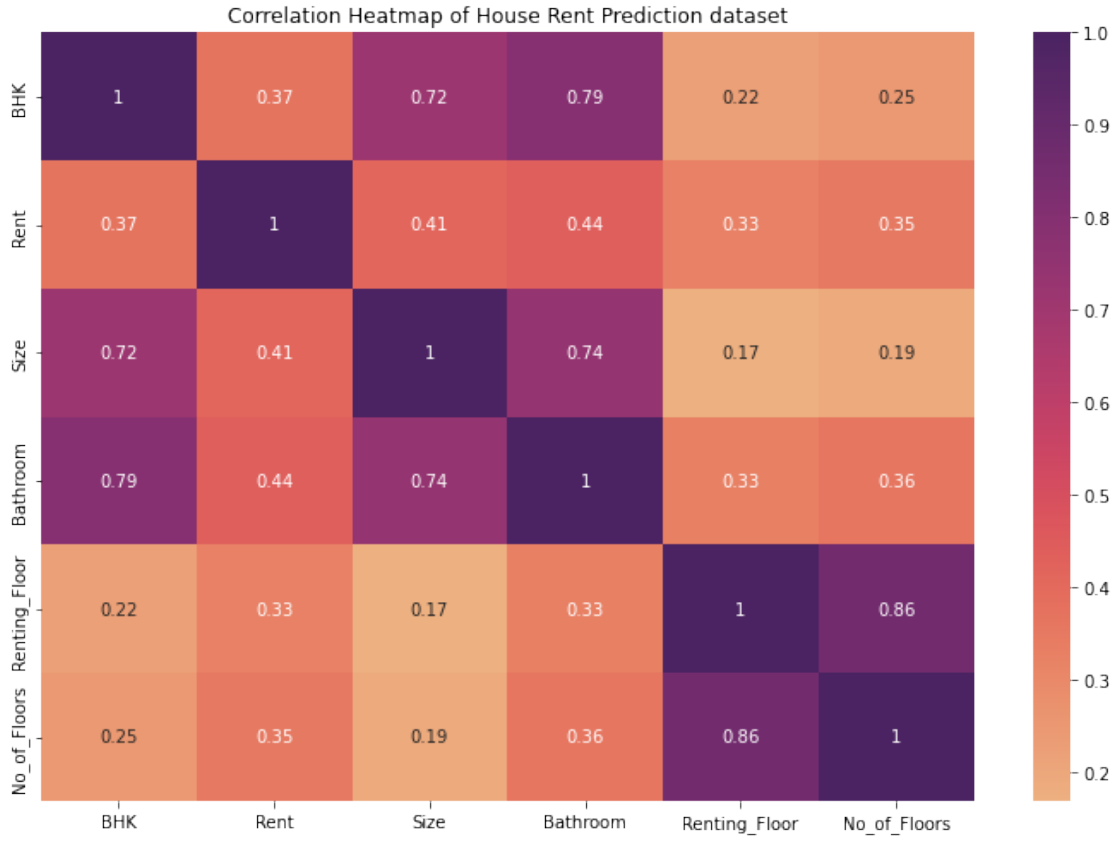
[29]:
```
# Plotting the pairplot which helps in understanding the relation between all␣
 ↪the variables.
sns.pairplot(rentdf)
```

[29]: <seaborn.axisgrid.PairGrid at 0x2b968828a30>



[30]:
```
# Let's plot a correlation heatmap of the heart disease dataset.

# Calculate the correlation coefficient with corr().
corr_number = rentdf.corr()

# Create the heatmap for the correlation coefficients calculated above.
fig, ax = plt.subplots(1, 1, figsize=(10,7), tight_layout = True)
sns.heatmap(corr_number, annot = True, cmap = 'flare')

# Title of the plot
plt.title('Correlation Heatmap of House Rent Prediction dataset')
```

[30]:
```
Text(0.5, 1.0, 'Correlation Heatmap of House Rent Prediction dataset')
```



Observations:

- From the above plot we can say that the target variable "Rent" is having good correlation with the variables BHK,Size, Bathroom, when compared to the Renting_Floor and No_of_Floors.
- We can say that the houses preferred for renting are more depedent on number of Bedrooms, hall and Kitchen, the size of the house, number of bathrooms available.

1 Project Milestone - 4 : Finalizing Your Results

2 1. Explain your process for prepping the data

The process of prepping the data involves the following steps.

1. Removing the unwanted columns from the dataset.

- Some of the columns from the dataset selected are not necessary during the analysis and are dropped from the dataset.
- The columns 'Posted on', 'Floor', 'Area Locality' are dropped from the dataset.
- After dropping the unwanted columns, check for null values and duplicate values is done. If any null values or duplicated values are identified those values are dropped and the shape of

the dataset is identified.

2. Identifying the outliers present in the dataset using boxplots and are to be removed for the further analysis.

- In the analysis the rent column is found to have outliers and are dropped from the dataset.

3. Creating Dummy Variables for the Categorical Features

- The categorical features present in the dataset are to be identified and dummy variables are to be created.
- This creation of dummy variables enables us to use a single regression equation for representing multiple groups.
- Check for null values is done and if any present are dropped from the dataset.
- After that the shape of the dataset is obtained.

4. Split the dataset into train and test datasets using 'Rent' as the target variable.

- Initially the dataset is split into features(X) with the target variable dropped from the dataset and target(y) variable by considering only the 'rent' column from the dataset.
- The data set is to be split into training and test datasets using 'Rent' as the target variable.
- Here the dataset is split into 80% training dataset and 20% test dataset, taking the test size as 0.2 and the random_state as 42.
- X_train,X_test,y_train,y_test are created using train_test_split()
- The train and test data sets shapes are obtained for better understanding the split of the dataset.
- The features train and test datasets are standardized using StandardScaler.
- The X_train dataset is transformed and fit to the standard saler and the the X_test dataset is just transformed but not fit.
- With this the data is ready for model building.

2.0.1 Removing the unwanted columns from the dataset.

- Some of the columns are not necessary during the analysis.We need to drop them from the dataset.
- Column 'Posted on' is removed as the dataset contains the data related to the year 2022.
- The 'Floor' column is split into two columns in the above and we can drop the original column.
- The column 'Area Loacality' can be dropped as it has many unique values.

[31]:
```
# Let's drop the columns'Posted on', 'Floor','Area Locality'
rentdf = rentdf.drop(['Posted On','Floor','Area Locality'], axis = 1)
```

[32]:
```
# Let's get the shape of the dataset after removing the unwnated data columns.
rentdf.shape
```

[32]: (4746, 11)

[33]:
```
# Checking if any missing values present in the dataset.
rentdf.isna().sum()
```

[33]:
```
BHK                  0
Rent                 0
Size                 0
Area Type            0
City                 0
Furnishing Status    0
Tenant Preferred     0
Bathroom             0
Point of Contact     0
Renting_Floor        0
No_of_Floors         0
dtype: int64
```

[34]:
```
# Re check for any duplicated values in the dataset.
rentdf.duplicated().sum()
```

[34]: 40

[35]:
```
# We see that there are 40 duplicated values, lets drop the duplicates from the␣
 ↪dataset.
rentdf = rentdf.drop_duplicates()
```

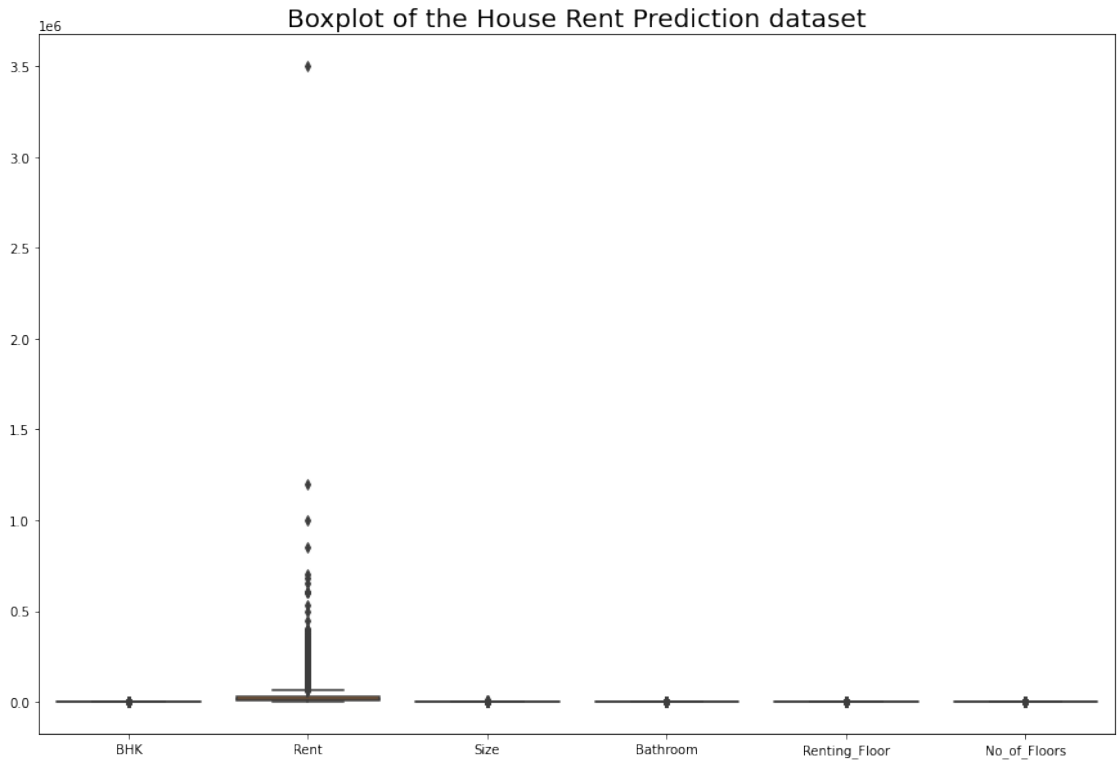
[36]:
```
# Re-check for duplicates
rentdf.duplicated().sum()
```

[36]: 0

2.0.2 Identify the Outliers in the data set using Boxplot

[37]:
```
# let's plot a boxplot for the house rent dataset.
plt.figure(figsize=(15,10))
sns.boxplot(data = rentdf)
# Title of the plot.
plt.title('Boxplot of the House Rent Prediction dataset', fontsize = 20)
```

[37]: Text(0.5, 1.0, 'Boxplot of the House Rent Prediction dataset')

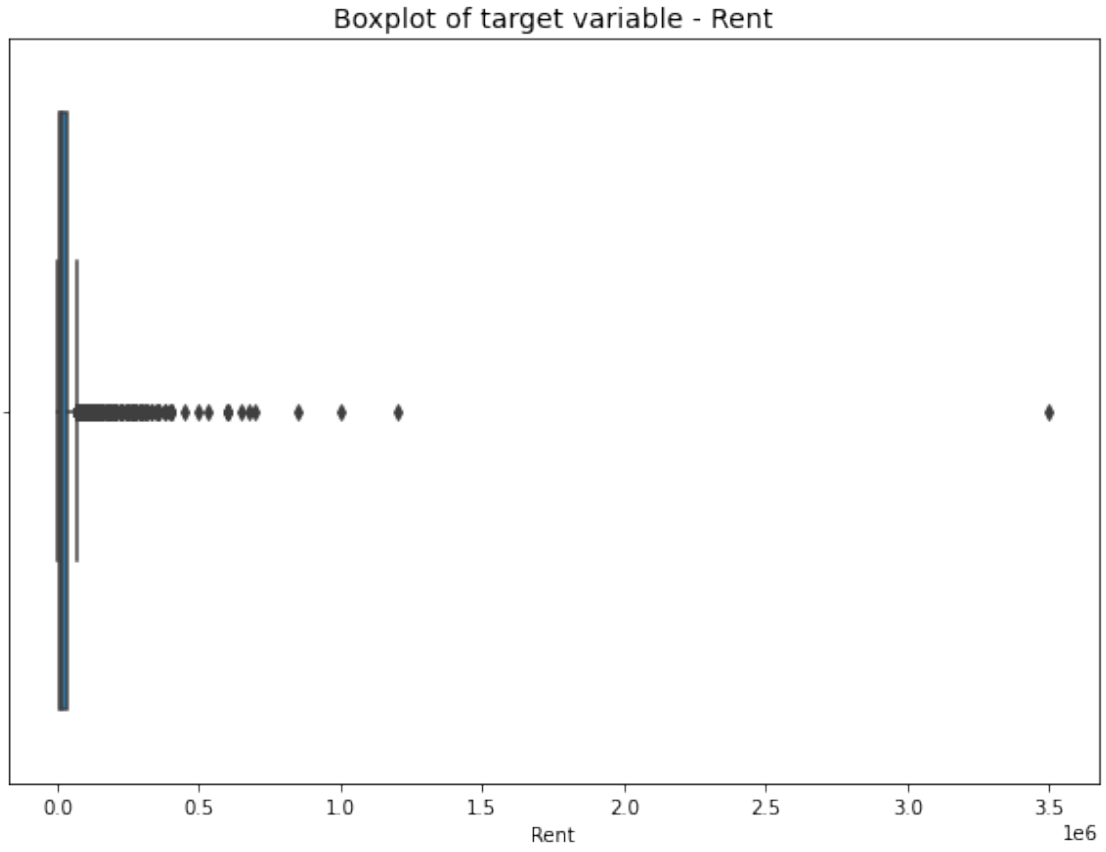


2.0.3 Observations :

- From the above boxplot we can see that there are outliers present in the target variable 'Rent'.

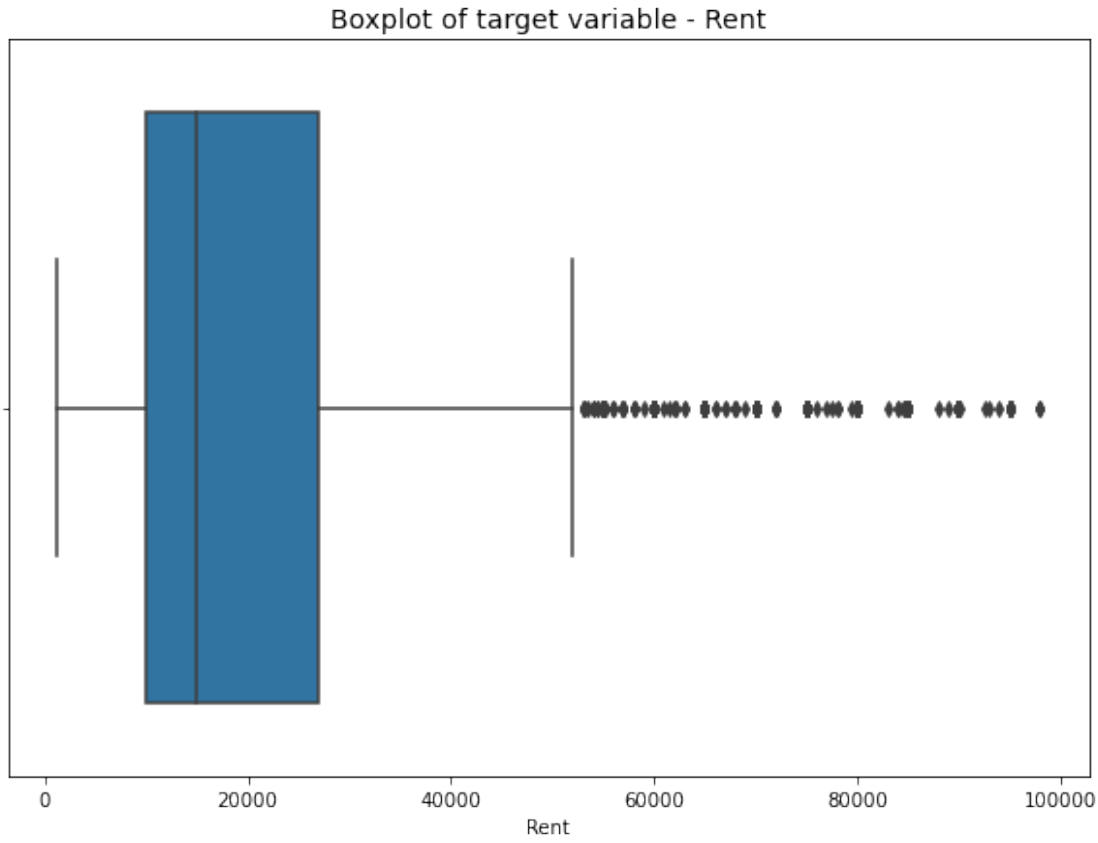
[38]:
```
# let's plot the boxplot for 'Rent' for clear understanding the outliers.
# let's plot a boxplot for the house rent dataset.
plt.figure(figsize=(10,7))
sns.boxplot(x = rentdf['Rent'])
# Title of the plot.
plt.title('Boxplot of target variable - Rent', fontsize = 14)
```

[38]:
```
Text(0.5, 1.0, 'Boxplot of target variable - Rent')
```



[39]:
```
# Let's drop the rent more than 100000, as we have very few houses avialble for␣
 ↪renting at the specified price.
rentdf.drop(rentdf[(rentdf['Rent'] >= 100000)].index, inplace=True)

# Let's plot the boxplot for the 'Rent' variable after dropping the values as␣
 ↪specified above.
plt.figure(figsize=(10,7))
sns.boxplot(x=rentdf["Rent"])
# Title of the plot.
plt.title('Boxplot of target variable - Rent', fontsize = 14)
```

[39]:
```
Text(0.5, 1.0, 'Boxplot of target variable - Rent')
```



[40]:
```
# After the removal of rental values above 100000 the following is the shape of␣
↪the data set.
rentdf.shape
```

[40]:
```
(4391, 11)
```

2.0.4 Observations:

- The size of the dataset before dropping the outliers is (4746, 11), and the size of the dataset after the removal of the outliers in the rent column is (4391,11).

2.0.5 Creating the dummy variables for the categorical features.

[41]:
```
# Let's convert the categorical features into dummy variables.
# let's identify the categorical features of the rent dataset.
categorycol = rentdf.select_dtypes(include = 'object').columns.tolist()
# Print the Categorical Columns identified from the House Rent dataset.
print(categorycol)
```

```
['Area Type', 'City', 'Furnishing Status', 'Tenant Preferred', 'Point of
Contact']
```

[42]:
```
# Let's convert the categrorical features to dummy variables
rentdf = pd.get_dummies(rentdf, columns = categorycol)
```

[43]:
```
# Let's check for any duplicates generated after the dummy variable conversion␣
 ↪of the categorical features.
rentdf.isna().sum().sort_values(ascending = False)
```

[43]:
```
BHK                                  0
Rent                                 0
Point of Contact_Contact Builder     0
Point of Contact_Contact Agent       0
Tenant Preferred_Family              0
Tenant Preferred_Bachelors/Family    0
Tenant Preferred_Bachelors           0
Furnishing Status_Unfurnished        0
Furnishing Status_Semi-Furnished     0
Furnishing Status_Furnished          0
City_Mumbai                          0
City_Kolkata                         0
City_Hyderabad                       0
City_Delhi                           0
City_Chennai                         0
City_Bangalore                       0
Area Type_Super Area                 0
Area Type_Carpet Area                0
Area Type_Built Area                 0
No_of_Floors                         0
Renting_Floor                        0
Bathroom                             0
Size                                 0
Point of Contact_Contact Owner       0
dtype: int64
```

[44]:
```
# Getting the shape of the dataset after creating the dummy variables for the␣
 ↪categorical features.
rentdf.shape
```

[44]:
```
(4391, 24)
```

[45]:
```
# Getting the first five rows of the house rent dataset after the dummy␣
 ↪variable creation for the categorical columns.
rentdf.head(5)
```

[45]:

	BHK	Rent	Size	Bathroom	Renting_Floor	No_of_Floors	\
0	2	10000	1100	2	0	2	
1	2	20000	800	1	1	3	
2	2	17000	1000	1	1	3	

```
3    2  10000   800         1              1             2
4    2   7500   850         1              1             2

   Area Type_Built Area  Area Type_Carpet Area  Area Type_Super Area  \
0                     0                      0                     1
1                     0                      0                     1
2                     0                      0                     1
3                     0                      0                     1
4                     0                      1                     0

   City_Bangalore  …  City_Mumbai  Furnishing Status_Furnished  \
0               0  …            0                            0
1               0  …            0                            0
2               0  …            0                            0
3               0  …            0                            0
4               0  …            0                            0

   Furnishing Status_Semi-Furnished  Furnishing Status_Unfurnished  \
0                                 0                              1
1                                 1                              0
2                                 1                              0
3                                 0                              1
4                                 0                              1

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

   Tenant Preferred_Family  Point of Contact_Contact Agent  \
0                        0                               0
1                        0                               0
2                        0                               0
3                        0                               0
4                        0                               0

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

[5 rows x 24 columns]
```

2.0.6 Split the dataset into train and test datasets using 'Rent' as the target variable.

[46]:
```
# Let's first define the features and target veariable as below
# X for features.
X = rentdf.drop(['Rent'], axis = 1)
# y for the target variable.
y = rentdf[['Rent']]
```

[47]:
```
# Now let's split the dataset into 80% training and 20 % test datasets␣
 ↪respectively.
# The test size is 0.2
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 42)
```

[48]:
```
# Let's get the shape of the original dataset and the training and test␣
 ↪datasets shape.

print("Original dataset shape : {} rows and {} columns.".format(rentdf.
 ↪shape[0],rentdf.shape[1]))
print("X_train shape :{} rows and {} columns.".format(X_train.shape[0],X_train.
 ↪shape[1]))
print("y_train shape : {} rows and {} columns.".format(y_train.shape[0],y_train.
 ↪shape[1]))
print("X_test shape : {} rows and {} columns.".format(X_test.shape[0],X_test.
 ↪shape[1]))
print("y_test shape : {} rows and {} columns.".format(y_test.shape[0],y_test.
 ↪shape[1]))
```

```
Original dataset shape : 4391 rows and 24 columns.
X_train shape :3512 rows and 23 columns.
y_train shape : 3512 rows and 1 columns.
X_test shape : 879 rows and 23 columns.
y_test shape : 879 rows and 1 columns.
```

[49]:
```
# Let's standardize the features train and test datasets.
# Create a StandardScaler
sdscaler = StandardScaler()
# Fit and transform the X_train dataset.
X_train_standardized = sdscaler.fit_transform(X_train)
# Transform but not fit X_test,
X_test_standardized = sdscaler.transform(X_test)
```

2.0.7 Model building and evaluation

2.1 2. Build and evaluate at least one model

- Here for the selected dataset I have used Linear Regression model, Decision Tree Model, Random Forest Model and LASSO Regression model.

- All the four models are created and fit with the X_train and y_train datasets respectively, predictions are created for the test dataset and the predictions are created for the train data for model validation.
- For the evaluation of the models selected I have used the R Squared and RMSE (Root Mean Squared Error). All the selected metrics are calculated each mmodel respectively and compared for selecting the best fit model for the dataset.

2.1.1 1. LiInear Regression

[50]:
```
# Let's create a Linear Regression Model.
lrreg = LinearRegression()

# Fit the Linear Regression model to training datasets.
lrreg.fit(X_train, y_train)

# Creating the predictions on the test data for model validation.
lrreg_pred = lrreg.predict(X_test)

# Creating the predictions on the trianed data for model validation.
lrreg_pred_train = lrreg.predict(X_train)
```

[51]:
```
# Calculating the R Squared value and the Root Mean Squared Error(RMSE) for the␣
 ↪Linear Regression Model.
# Calculating R Squared Value on Test dataset
rsquared_lr = r2_score(y_test, lrreg_pred)

# Calculating R Squared value on trained dataset
rsquared_lrtrain = r2_score(y_train, lrreg_pred_train)

# Calculating the Mean Squared Error on Test dataset.
mse_lr = mean_squared_error(y_test, lrreg_pred)

# Calculating the RMSE(Root Mean Squared Error)
rmselr = np.sqrt(mse_lr)

# Print the evaluation metrics R Squared and RMSE values for the Linear␣
 ↪Regression Model.
print("R Squared value of the Linear Regression Model on Test Dataset :",␣
 ↪round(rsquared_lr,4))
print("R Squared value of the Linear Regression Model on Train Dataset :",␣
 ↪round(rsquared_lrtrain,4))
print("MSE value of the Linear Regression Model  :", round(mse_lr,4))
print("RMSE value of the Linear Regression Model  :", round(rmselr,4))
```

```
R Squared value of the Linear Regression Model on Test Dataset : 0.7128
R Squared value of the Linear Regression Model on Train Dataset : 0.7001
MSE value of the Linear Regression Model  : 117012863.6065
```

```
RMSE value of the Linear Regression Model   : 10817.2484
```

[52]:
```
# Intialise importance for the Linear Regression

lrimportance = lrreg.coef_
# Print the coefficients
lrimportance
```

[52]:
```
array([[ 4.03583217e+03,  8.95490219e+00,  2.96139634e+03,
         2.09234155e+02,  2.59874412e+02, -4.29440748e+03,
         2.55742497e+03,  1.73698250e+03, -2.62850000e+03,
        -3.40222677e+03,  1.01843376e+03, -5.76408429e+03,
        -5.85571774e+03,  1.66320950e+04,  4.04837385e+03,
        -1.54255077e+03, -2.50582308e+03,  1.32713826e+03,
         6.73429426e+02, -2.00056769e+03,  4.26429178e+03,
         7.45271259e+02, -5.00956304e+03]])
```

2.1.2 2. Decision Tree Model

[53]:
```
# Let's create Decision Tree Model.
dectrereg = DecisionTreeRegressor()

# Fit the Random Forest model to training datasets.
dectrereg.fit(X_train, y_train)

# Creating the predictions on the test data for model validation.
dectrereg_pred = dectrereg.predict(X_test)

# Creating the predictions on the trianed data for model validation.
dectrereg_pred_train = dectrereg.predict(X_train)
```

[54]:
```
# Calculating the R Squared value and the Root Mean Squared Error(RMSE) for the␣
 ↪Decision Tree Model.
# Calculating R Squared Value on Test dataset
rsquared_dectrereg = r2_score(y_test, dectrereg_pred)

# Calculating R Squared value on trained dataset
rsquared_dectrereg_train = r2_score(y_train, dectrereg_pred_train)

# Calculating the Mean Squared Error on Test dataset.
mse_dectrereg = mean_squared_error(y_test, dectrereg_pred)

# Calculating the RMSE(Root Mean Squared Error)
rmsedectrereg = np.sqrt(mse_dectrereg)

# Print the evaluation metrics R Squared nd RMSE values for the Decision Tree␣
 ↪Model.
```

```
print("R Squared value of the Decision Tree Model on Test Data :",␣
 ↪round(rsquared_dectrereg,4))
print("R Squared value of the Decision Tree Model on Train Data is:",␣
 ↪round(rsquared_dectrereg_train,4))
print("MSE value of the Decision Tree Model is:", round(mse_dectrereg,4))
print("RMSE value of the Decision Tree Model is:", round(rmsedectrereg,4))
```

```
R Squared value of the Decision Tree Model on Test Data : 0.6263
R Squared value of the Decision Tree Model on Train Data is: 0.9979
MSE value of the Decision Tree Model is: 152220654.7231
RMSE value of the Decision Tree Model is: 12337.7735
```

2.1.3 3. Random Forest Model

[55]:
```
# Let's create Random Forest Model.
rfreg = RandomForestRegressor()

# Fit the Random Forest model to training datasets.
rfreg.fit(X_train, y_train)

# Creating the predictions on the test data for model validation.
rfreg_pred = rfreg.predict(X_test)

# Creating the predictions on the trianed data for model validation.
rfreg_pred_train = rfreg.predict(X_train)
```

[56]:
```
# Calculating the R Squared value and the Root Mean Squared Error(RMSE) for the␣
 ↪Random Forest Model.
# Calculating R Squared Value on Test dataset
rsquared_rfreg = r2_score(y_test, rfreg_pred)

# Calculating R Squared value on trained dataset
rsquared_rfregtrain = r2_score(y_train, rfreg_pred_train)

# Calculating the Mean Squared Error on Test dataset.
mse_rfreg = mean_squared_error(y_test, rfreg_pred)

# Calculating the RMSE(Root Mean Squared Error)
rmserfreg = np.sqrt(mse_rfreg)

# Print the evaluation metrics R Squared nd RMSE values for the Random Forest␣
 ↪Model.
print("R Squared value of the Random Forest Model on Test Data :",␣
 ↪round(rsquared_rfreg,4))
print("R Squared value of the Random Forest Model on Train Data :",␣
 ↪round(rsquared_rfregtrain,4))
print("MSE value of the Random Forest Model :", round(mse_rfreg,4))
```

```
print("RMSE value of the Random Forest Model :", round(rmserfreg,4))
```

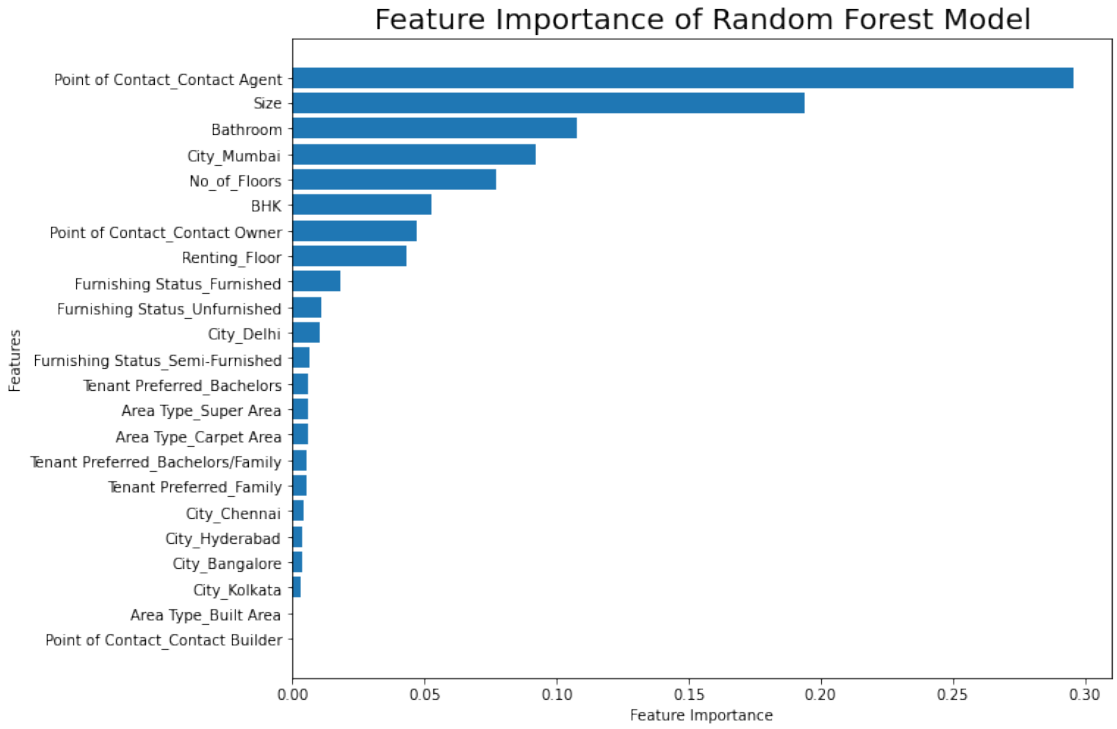
```
R Squared value of the Random Forest Model on Test Data : 0.7666
R Squared value of the Random Forest Model on Train Data : 0.9644
MSE value of the Random Forest Model : 95095368.8336
RMSE value of the Random Forest Model : 9751.6854
```

[57]:
```
# The importance of a feature is basically: how much this feature is used in␣
 ↪each tree of the forest.
importfeatures = rfreg.feature_importances_
importfeatures
```

[57]:
```
array([5.29147099e-02, 1.94014987e-01, 1.07813007e-01, 4.34400674e-02,
       7.71179387e-02, 8.34305056e-06, 5.93598064e-03, 5.93750443e-03,
       3.60311388e-03, 4.48478817e-03, 1.06156337e-02, 3.93115358e-03,
       2.89236848e-03, 9.18584887e-02, 1.84009277e-02, 6.61516694e-03,
       1.07212820e-02, 5.95966868e-03, 5.63538471e-03, 5.45354755e-03,
       2.95470566e-01, 2.46577555e-07, 4.71751248e-02])
```

[58]:
```
# initialize the idicies
indices = np.argsort(importfeatures)
# Defining the size of the plot
fig, ax = plt.subplots(figsize = (10,8))
# Plotting the graph
ax.barh(range(len(importfeatures)), importfeatures[indices])
# Y-ticks
ax.set_yticks(range(len(importfeatures)))
_ = ax.set_yticklabels(np.array(X_train.columns)[indices])
# X-label
plt.xlabel("Feature Importance", fontsize = 10)
# Y - label
plt.ylabel("Features", fontsize = 10)
# Title of the plot
plt.title("Feature Importance of Random Forest Model", fontsize = 20)
```

[58]:
```
Text(0.5, 1.0, 'Feature Importance of Random Forest Model')
```



2.1.4 Observations :

- From the above plot we can see that the Point of contact by Agent, Size, Bathroom , Mumbai city are the important features for this model.

2.1.5 4. LASSO Regression Model

[59]:
```
# Let's create LASSO Regression Forest Model.
lassoreg = Lasso()

# Fit the LASSO Regression model to training datasets.
lassoreg.fit(X_train, y_train)

# Creating the predictions on the test data for model validation.
lassoreg_pred = lassoreg.predict(X_test)

# Creating the predictions on the trianed data for model validation.
lassoreg_pred_train = lassoreg.predict(X_train)
```

[60]:
```
# Calculating the R Squared value and the Root Mean Squared Error(RMSE) for the␣
 ↪LASSO Regression Model.
# Calculating R Squared Value on Test dataset.
rsquared_lassoreg = r2_score(y_test, lassoreg_pred)
```

```
# Calculating R Squared value on trained dataset
rsquared_lassoregtrain = r2_score(y_train, lassoreg_pred_train)

# Calculating the Mean Squared Error on Test dataset.
mse_lassoreg = mean_squared_error(y_test, lassoreg_pred)

# Calculating the RMSE(Root Mean Squared Error)
rmselassoreg = np.sqrt(mse_lassoreg)

# Print the evaluation metrics R Squared nd RMSE values for the LASSO␣
 ↪Regression Model.
print("R Squared value of the Lasso Regression Model on Test Data :",␣
 ↪round(rsquared_lassoreg,4))
print("R Squared value of the Lasso Regression Model on Train Data :",␣
 ↪round(rsquared_lassoregtrain,4))
print("MSE value of the Lasso Regression Model :", round(mse_lassoreg,4))
print("RMSE value of the Lasso Regression Model :", round(rmselassoreg,4))
```

```
R Squared value of the Lasso Regression Model on Test Data : 0.7129
R Squared value of the Lasso Regression Model on Train Data : 0.7001
MSE value of the Lasso Regression Model : 116954040.7728
RMSE value of the Lasso Regression Model : 10814.5292
```

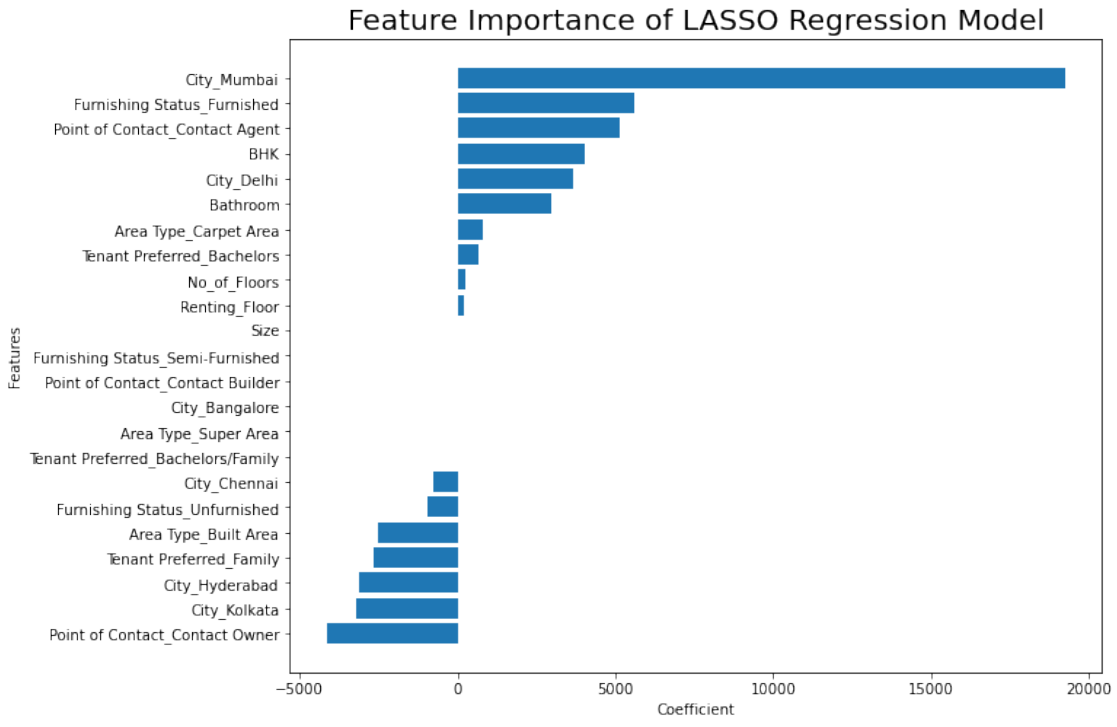
[61]:
```
# Lasso feature importance is by the coefficient values

lassoimportance = lassoreg.coef_

for i,v in enumerate(lassoimportance):
    print('Feature: %0d, Score: %.5f'% (i,v))
```

```
Feature: 0, Score: 4032.04275
Feature: 1, Score: 8.95485
Feature: 2, Score: 2960.18718
Feature: 3, Score: 209.36581
Feature: 4, Score: 260.20942
Feature: 5, Score: -2503.25724
Feature: 6, Score: 818.75871
Feature: 7, Score: -0.00000
Feature: 8, Score: 0.00000
Feature: 9, Score: -768.40839
Feature: 10, Score: 3646.54136
Feature: 11, Score: -3121.62110
Feature: 12, Score: -3211.52940
Feature: 13, Score: 19253.44847
Feature: 14, Score: 5575.87871
Feature: 15, Score: 0.00000
Feature: 16, Score: -961.00297
Feature: 17, Score: 648.15456
```

```
Feature: 18, Score: -0.00000
Feature: 19, Score: -2661.55955
Feature: 20, Score: 5127.21442
Feature: 21, Score: 0.00000
Feature: 22, Score: -4151.60615
```

[62]:
```
# initialize the idicies
indices = np.argsort(lassoimportance)
# Defining the size of the plot
fig, ax = plt.subplots(figsize = (10,8))
# Plotting the graph
ax.barh(range(len(lassoimportance)), lassoimportance[indices])
# Y-ticks
ax.set_yticks(range(len(lassoimportance)))
_ = ax.set_yticklabels(np.array(X_train.columns)[indices])
# X-label
plt.xlabel("Coefficient", fontsize = 10)
# Y - label
plt.ylabel("Features", fontsize = 10)
# Title of the plot
plt.title("Feature Importance of LASSO Regression Model", fontsize = 20)
```

[62]:
```
Text(0.5, 1.0, 'Feature Importance of LASSO Regression Model')
```



2.1.6 Observations :

- From the above plot we can see that Mumbai city , Furnishing status, Point of contact by agent,BHK are the important features for this model.

2.2 3. Interpret your results

- The Random Forest Model performed good with low RMSE(Root Mean Squared Error) value as 9770.76, R Squared value is good on the train dataset i.e. 0.9639, but the R Squared value on the test dataset is dropped to 0.76565.This indicates that the model is not generalising properly and indicates overfitting of the train dataset.
- The Decision tree model can be neglected as the RMSE and the R Squared values are more and the R squared values for the test and train data are having huge difference, which indicates the model is being overfit with the train dataset.
- When observed the R Squared values and the RMSE values of the train dataset and the test dataset of Linear Regression and LASSO Regression models,the values are almost the same for both the models which are very close.We can say that both the models performed well on the given dataset.

All the above results are for the four models build and evaluated are summarized as below for better understanding.

2.2.1 Summary of the results of the four models.

[63]:
```
# Let's form arrays for the calculated R squared value and the RMSE(Root Mean␣
 ↪Squared Error) for the above three models.
linear_reg = {'Model':'Linear Regression',
               'R Squared(test)':rsquared_lr,
              'R Squared(train)':rsquared_lrtrain,
               'RMSE':rmselr,}

decisiontree_reg = {'Model':'Decision Tree ',
               'R Squared(test)':rsquared_dectrereg,
              'R Squared(train)':rsquared_dectrereg_train,
               'RMSE':rmsedectrereg,}

randomforest_reg = {'Model':'Random Forest',
                'R Squared(test)':rsquared_rfreg,
                'R Squared(train)':rsquared_rfregtrain,
               'RMSE':rmserfreg,}

lasso_reg = {'Model':'LASSO Regression',
               'R Squared(test)':rsquared_lassoreg,
             'R Squared(train)':rsquared_lassoregtrain,
               'RMSE':rmselassoreg,}
```

[64]:
```
# Let's group the results of the three models using the pd.series()
models_evalmetrics = pd.DataFrame({'Linear Regression Model':pd.
↪Series(linear_reg),
                        'Decision Tree Model': pd.Series(decisiontree_reg),
                       'Random Forest Model':pd.Series(randomforest_reg),
                       'Lasso Regression Model':pd.Series(lasso_reg),
                      })

models_evalmetrics
```

[64]:

	Linear Regression Model	Decision Tree Model	\
Model	Linear Regression	Decision Tree	
R Squared(test)	0.712764	0.626338	
R Squared(train)	0.700134	0.997924	
RMSE	10817.24843	12337.773491	

	Random Forest Model	Lasso Regression Model
Model	Random Forest	LASSO Regression
R Squared(test)	0.766566	0.712908
R Squared(train)	0.964429	0.700121
RMSE	9751.685436	10814.529152

2.2.2 Observations:

- From the above result the Random Forest Model performed good with low RMSE(Root Mean Squared Error) value as 9770.76, R Squared value is good on the train dataset i.e. 0.9639, but the R Squared value on the test dataset is dropped to 0.76565.This indicates that the model is not generalising properly and indicates overfitting of the train dataset.
- From the above result we see that Decision tree model can be neglected as the RMSE and the R Squared values are more and the R squared values for the test and train data are having huge difference, which indicates the model is being overfit with the train dataset.
- When observed the R Squared values and the RMSE values of the train dataset and the test dataset of Linear Regression and LASSO Regression models,the values are almost the same for both the models which are very close.We can say that both the models performed well on the given dataset.

2.2.3 Summary :

- As per the exploratory data analysis Mumbai city in India is having more number houses for rent(as per the dataset).
- The rentals in Mumbai city are more when compared to other cities in India.
- The houses available rent as per the dataset are mainly of 2 BHK and size of 500 to 1100 square feet.
- The machine learning models that best fits for the House Rent dataset is LASSO regression model having the features as Mumbai city , Furnishing status, Point of contact by agent, BHK.

2.3 4. Begin to formulate a conclusion/recommendations

2.4 Recommendations :

- In future the project recommendations are, the project can be imporved by applying the hyper parameter tuning on the models selected to reduce the problem of over fitting and variance errors in the model and improve the performance of the models.
- Cross-Validation technique can be used to assess how well a model performs.The simplest example of cross-validation is when you split your data into three groups: training data, validation data, and testing data, where you see the training data to build the model, the validation data to tune the hyper parameters, and the testing data to evaluate your final model.

2.5 Conclusion :

- There are several factors that play an important role in the prediction of the house rents in any place. With the above analysis the LASSO regression model best fits for the analysis of the House Rent dataset selected. As per the above analysis of the the House Rent Dataset of India, we can say that in India the houses rents are mostly dependent upon the cities where the houses are situated,the number of Bedrooms, Hall and Kitchen available for houses, Bathroom available and Size of the house.This analysis will be helpful for the rental listing sites for contatcing the owners of the rental properties available and updating the sites for easy access for the persons searching for houses for rent there by increasing the business.

2.5.1 References:

Google, inria.github.io, scikitlearn, feature selection, https://inria.github.io/scikit-learn-mooc/python_scripts/dev_features_importance.html

Google, towardsdatascience , feature section in machine learning using lasso regression, https://towardsdatascience.com/feature-selection-in-machine-learning-using-lasso-regression-7809c7c2771a

Google, Machine Learning Mastery, How to calculate Feature Importance with Python,https://machinelearningmastery.com/calculate-feature-importance-with-python/

Google, towardsdatascience, hyperparameter tuning in linear regression, https://medium.com/analytics-vidhya/hyperparameter-tuning-in-linear-regression-e0e0f1f968a1

[ ]: