MAKAAN PROJECT

Project report on

Property Price Prediction

Submitted by

SRUTI DUTTA

Mentor

Mr.Bose

Top Mentor

Makaan project Report

Makaan project Report

1. Overview:

People and real estate agencies buy or sell properties, people buy properties either to live in or as an investment and the agencies buy to run a business. There are multiple factors on which price of a property depends which includes city, location, size and sometimes the name of the builder can also be a deciding factor. Taking those factors in account and studying the given in detail we can train and deploy ML model to predict the price of the property. Predicting the prices will help the customer as well as company to select regions depending upon their budget. Also, using EDA we can classify city wise prices, availability and find other insights as well. In this project we are working on the dataset of the company Makaan.com for Price prediction.

2. about Dataset:

This dataset was scraped from one of the housing website called as makaan.com. **Makaan.com** has quickly emerged as the preferred partner for consumers looking to rent, buy or sell a home. Makaan.com offers its online consumers maximum property options and has become one of the largest advertising platforms in online real estate in India.

3. Problem Statement:

The company wants to predict prices of various properties that will be listed in their site using Machine Learning Models. Based on the past data given to us, we need to predict the price.

4. Data Description:

Dataset -1 details (Details about the properties/different features)

1. Property Name: Name of the Property
2. Property_id: ID number
3. Property_type : Type of property (Apartment, Residential Plot ,Independent Floor, Independent House, Villa)
4. Property_status: Status of property (Ready to move/Under construction)
5. Price_per_unit_area: Price per sq. feet area
6. Posted_On: Time since posted
7. builder_id: ID number
8. Builder name: Builder's name
9. Property_building_status: property build or not (active/inactive/unverified)
10. No_of_BHK: Number of bedrooms
11. **Price: Price of the Property (Target Variable)**
12. Size: Total size of property in sq feet
13. Description: Description given by the people who posted
14. is_furnished: Is (furnished,semi-furnished,unfurished)
15. listing_domain_score: domain score
16. is_plot : Whether a plot or not
17. is_RERA_registered: if registered under real estate authority
18. is_Apartment: Whether apartment or not
19. is_ready_to_move: Whether ready to move or not
20. is_commercial_Listing: Whether a commercial or not
21. is_PentaHouse: Whether penthouse or not
22. is_studio: Whether studio or not
23. Listing_Category: For selling or rent

Dataset -2 Makaan_property_location_details

1. Property_id: Unique Property ID
2. City_id: Unique ID of city
3. City_name: Unique city name
4. Locality_ID: Unique Locality ID
5. Locality_Name: Unique locality name
6. Longitude: Longitudinal Co-ordinates
7. Latitude: Latitudinal Co-ordinates
8. Sub_urban_ID: Unique sub urban id
9. Sub_urban_name: Unique sub urban name

5. Loading dataset:

PREDICTION OF THE HOUSE PRICES---

Importing of Packages--

In [1]:
```
import pandas as pd
import numpy as np
import math as m
import seaborn as sns
sns.set_style("whitegrid")
import matplotlib.pyplot as plt
import plotly.express as px
```

Reading the first dataset:

In [2]:
```
def read_df1():
    df1=pd.read_csv("G:/Makaan_Properties_Details.csv",encoding='latin1')
    return df1
print("Calling the read_data function--")
df1=read_df1()
print(df1.head(2))
df1.columns
```

```
Calling the read_data function--
               Property_Name  Property_id Property_type      Property_status  \
0               Arkiton Luxe     15446514     Apartment   Under Construction
1  Keshav Akshar Ocean Pearl     15367414     Apartment   Under Construction

  Price_per_unit_area   Posted_On  \
0               4,285   1 day ago
1               7,000  2 days ago

                                         Project_URL   builder_id  \
0  https://www.makaan.com/ahmedabad/arkiton-life-...  100563465.0
1  https://www.makaan.com/ahmedabad/keshav-naraya...  100009433.0

          Builder_name Property_building_status  ... is_furnished  \
0    Arkiton life Space                  ACTIVE  ...  Unfurnished
1  Keshav Narayan Group                  ACTIVE  ...  Unfurnished

  listing_domain_score is_plot is_RERA_registered is_Apartment  \
0                  4.0   False               True         True
1                  4.0   False               True         True

  is_ready_to_move  is_commercial_Listing  is_PentaHouse  is_studio  \
0            False                  False          False      False
1            False                  False          False      False

  Listing_Category
0             sell
1             sell

[2 rows x 24 columns]
```

Out[2]:
```
Index(['Property_Name', 'Property_id', 'Property_type', 'Property_status',
       'Price_per_unit_area', 'Posted_On', 'Project_URL', 'builder_id',
       'Builder_name', 'Property_building_status', 'No_of_BHK', 'Price',
       'Size', 'description', 'is_furnished', 'listing_domain_score',
       'is_plot', 'is_RERA_registered', 'is_Apartment', 'is_ready_to_move',
       'is_commercial_Listing', 'is_PentaHouse', 'is_studio',
       'Listing_Category'],
      dtype='object')
```

Reading the second dataset:

```
In [3]: def read_df2():
            df2=pd.read_csv("C:/top mentor data sci assignmets/18 jun/Capstone_project/Makaan_property_location_details.csv")
            return df2
        print("Calling the read_data function--")
        df2=read_df2()
        print(df2.head(2))
        df2.columns

        Calling the read_data function--
           Property_id  City_id  City_name  Locality_ID Locality_Name  Longitude  \
        0     15579866        1  Ahmedabad        51749      Bodakdev  72.520195
        1     15579809        1  Ahmedabad        51749      Bodakdev  72.502571

            Latitude  Sub_urban_ID Sub_urban_name
        0  23.040195         10003     SG Highway
        1  23.032154         10003     SG Highway
Out[3]: Index(['Property_id', 'City_id', 'City_name', 'Locality_ID', 'Locality_Name',
               'Longitude', 'Latitude', 'Sub_urban_ID', 'Sub_urban_name'],
              dtype='object')
```

Performing inner join to merge two data files:

```
In [4]: data=df1.merge(df2,left_on='Property_id', right_on='Property_id',how = 'inner')
        pd.set_option("display.max.columns",None)
        data.head(2)
```

Out[4]:

	Property_Name	Property_id	Property_type	Property_status	Price_per_unit_area	Posted_On	Project_URL	builder_id	Builder_name	Property_building_status	No_of_BHK	Price	Size	descripti
0	Arkiton Luxe	15446514	Apartment	Under Construction	4,285	1 day ago	https://www.makaan.com/ahmedabad/arkiton-life-...	100563465.0	Arkiton life Space	ACTIVE	3 BHK	75,00,000	1,750 sq ft	The hou unfurnish It has parking
1	Arkiton Luxe	15446514	Apartment	Under Construction	4,285	1 day ago	https://www.makaan.com/ahmedabad/arkiton-life-...	100563465.0	Arkiton life Space	ACTIVE	3 BHK	75,00,000	1,750 sq ft	The hou unfurnish It has parking

Print basic info about data:

Print basic info about data-

```
In [5]: print(data.columns)
        print("---------------------------------------------")
        print("Rows,Columns--",data.shape)
        print("---------------------------------------------")
        print(data.info())

        Index(['Property_Name', 'Property_id', 'Property_type', 'Property_status',
               'Price_per_unit_area', 'Posted_On', 'Project_URL', 'builder_id',
               'Builder_name', 'Property_building_status', 'No_of_BHK', 'Price',
               'Size', 'description', 'is_furnished', 'listing_domain_score',
               'is_plot', 'is_RERA_registered', 'is_Apartment', 'is_ready_to_move',
               'is_commercial_Listing', 'is_PentaHouse', 'is_studio',
               'Listing_Category', 'City_id', 'City_name', 'Locality_ID',
               'Locality_Name', 'Longitude', 'Latitude', 'Sub_urban_ID',
               'Sub_urban_name'],
              dtype='object')
        ---------------------------------------------
        Rows,Columns-- (4942704, 32)
        ---------------------------------------------
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4942704 entries, 0 to 4942703
Data columns (total 32 columns):
 #   Column                    Dtype  
---  ------                    -----  
 0   Property_Name             object 
 1   Property_id               int64  
 2   Property_type             object 
 3   Property_status           object 
 4   Price_per_unit_area       object 
 5   Posted_On                 object 
 6   Project_URL               object 
 7   builder_id                float64
 8   Builder_name              object 
 9   Property_building_status  object 
 10  No_of_BHK                 object 
 11  Price                     object 
 12  Size                      object 
 13  description               object 
 14  is_furnished              object 
 15  listing_domain_score      float64
 16  is_plot                   bool   
 17  is_RERA_registered        bool   
 18  is_Apartment              bool   
 19  is_ready_to_move          bool   
 20  is_commercial_Listing     bool   
 21  is_PentaHouse             bool   
 22  is_studio                 bool   
 23  Listing_Category          object 
 24  City_id                   int64  
 25  City_name                 object 
 26  Locality_ID               int64  
 27  Locality_Name             object 
 28  Longitude                 float64
 29  Latitude                  float64
 30  Sub_urban_ID              int64  
 31  Sub_urban_name            object 
dtypes: bool(7), float64(4), int64(4), object(17)
memory usage: 1013.5+ MB
None
```

5. Data Pre-processing:

1. Data Cleaning:

We observe some of the variables have incorrect datatype so we rectify those variables with the correct datatypes.

Data Cleaning--

```
In [7]: data['Price_per_unit_area'].unique()

Out[7]: array(['4,285', '3,600', '7,000', ..., '11,763', '39,464', '38,910'],
              dtype=object)

In [8]: data['Price'].unique()

Out[8]: array(['75,00,000', '63,00,000', '2,36,88,000', ..., '26,49,999',
               '26,98,434', '35,29,577'], dtype=object)

In [9]: data['Size'].unique()

Out[9]: array(['1,750 sq ft', '3,384 sq ft', '2,295 sq ft', ..., '4,556 sq ft',
               '18,837 sq ft', '226 sq ft'], dtype=object)

In [10]: data['Price_per_unit_area'] = data['Price_per_unit_area'].replace(',', '',regex=True)
         data['Price_per_unit_area']=data['Price_per_unit_area'].astype(int)

In [11]: data['Price'] = data['Price'].replace(',', '',regex=True)
         data['Price']=data['Price'].astype(int)

In [12]: data['Size']=data['Size'].replace("sq ft","",regex=True)
         data['Size']=data['Size'].replace(",","",regex=True)
         data['Size']=data['Size'].astype(int)
```

> 1.Columns 'Price_per_unit_area','Price' have object datatype we are changing it to int type and also removing the comma.
> 2. From column size we are removing "sq ft" and "," plus changing its datatype from object to int.

1. Columns 'Price_per_unit_area','Price' have object datatype we are changing it to int type and also removing the comma.
2. From column 'Size' we are removing "sq ft" and "," plus changing its datatype from object to int.

Dropping few rows with RK's:

As we can see we have few RKs in BHK column.If we consider our data they are few thousands in number.so lets drop this RKs.

```
In [13]: data['No_of_BHK'].unique()

Out[13]: array(['3 BHK', '4 BHK', '2 BHK', '5 BHK', '1 BHK', '1 RK', '0 BHK',
                '12 BHK', '7 BHK', '6 BHK', '8 BHK', '10 BHK', '11 BHK', '9 BHK',
                '15 BHK', '3 RK', '14 BHK', '2 RK'], dtype=object)

In [14]: print(len(data[data['No_of_BHK']=='1 RK']))
         print(len(data[data['No_of_BHK']=='2 RK']))
         print(len(data[data['No_of_BHK']=='3 RK']))
         print(data.shape)

7271
2
4
(4942704, 32)
```

> As we can see we have few RKs in BHK column.If we consider our data they are few thousands in number.so lets drop this RKs.

```
In [15]: data.drop(data[(data['No_of_BHK']=='1 RK') |(data['No_of_BHK']=='2 RK') | (data['No_of_BHK']=='3 RK')].index,inplace=True)
```

“0 BHKs” are Residential Plots:

```
In [16]: data['No_of_BHK'].unique()

Out[16]: array(['3 BHK', '4 BHK', '2 BHK', '5 BHK', '1 BHK', '0 BHK', '12 BHK',
               '7 BHK', '6 BHK', '8 BHK', '10 BHK', '11 BHK', '9 BHK', '15 BHK',
               '14 BHK'], dtype=object)
```

We have "0 BHK" lets analyse them-

```
In [17]: data[data['No_of_BHK']=='0 BHK'].head(2)
```

Out[17]:

	Property_Name	Property_id	Property_type	Property_status	Price_per_unit_area	Posted_On	Project_URL	builder_id	Builder_name	Property_building_status	No_of_BHK	Price	Size	descript
1191	NaN	15528030	Residential Plot	Ready to move	13650	9 days ago	https://www.makaan.com/ahmedabad/builder-proje...	NaN	NaN	UNVERIFIED	0 BHK	43000000	3150	A plo availabl a pr locatio J
1192	NaN	15528240	Residential Plot	Ready to move	518	9 days ago	https://www.makaan.com/ahmedabad/builder-proje...	NaN	NaN	UNVERIFIED	0 BHK	700000	1350	A plo availabl a pr locatio G

As we can see this '0 BHKs' are Residential Plots.

Cleaning BHK column:

Cleaning BHK column-

```
In [18]: data['No_of_BHK']=data['No_of_BHK'].replace('BHK', '',regex=True)
         data['No_of_BHK']=data['No_of_BHK'].astype(int)

In [19]: data.columns
         data.head(2)
```

Out[19]:

	Property_Name	Property_id	Property_type	Property_status	Price_per_unit_area	Posted_On	Project_URL	builder_id	Builder_name	Property_building_status	No_of_BHK	Price	Size	descriptio
0	Arkiton Luxe	15446514	Apartment	Under Construction	4285	1 day ago	https://www.makaan.com/ahmedabad/arkiton-life-...	100563465.0	Arkiton life Space	ACTIVE	3	7500000	1750	The hous i unfurnishec It has ca parking...
1	Arkiton Luxe	15446514	Apartment	Under Construction	4285	1 day ago	https://www.makaan.com/ahmedabad/arkiton-life-...	100563465.0	Arkiton life Space	ACTIVE	3	7500000	1750	The hous i unfurnishec It has ca parking...

Dropping irrelevant columns:

```
In [20]: data["Listing_Category"].unique()

Out[20]: array(['sell'], dtype=object)
```

Dropping irrelevant columns--

```
In [21]: data.drop(columns=['Property_id','Posted_On', 'Project_URL','builder_id','Builder_name','description',
                  'listing_domain_score','Listing_Category', 'City_id','Locality_ID','Longitude', 'Latitude','Sub_urban_ID'],inplace=True)
         data.head(2)
```

Out[21]:

	Property_Name	Property_type	Property_status	Price_per_unit_area	Property_building_status	No_of_BHK	Price	Size	is_furnished	is_plot	is_RERA_registered	is_Apartment	is_ready_to_move	is_commercial_Listing	is_PentaHo
0	Arkiton Luxe	Apartment	Under Construction	4285	ACTIVE	3	7500000	1750	Unfurnished	False	True	True	False	False	F
1	Arkiton Luxe	Apartment	Under Construction	4285	ACTIVE	3	7500000	1750	Unfurnished	False	True	True	False	False	F

> 1.'Property_id','Posted_On', 'Project_URL','builder_id','Builder_name','description','listing_domain_score' this parameters will have no effect on price of house. hence,we are dropping this columns.
> 2."Listing_Category" column infers that all the houses are on sell.thus,this column isn't representing any distinct attribute.hence,we are dropping this column.
> 3.We are keeping City_name,Locality_Name,Sub_urban_name columns.'City_id','Locality_ID','Sub_urban_ID' display the same thing.hence,we are dropping this columns as well.
> 4.We alredy have a city_name hence,keeping 'Longitude' and 'Latitude' column will be of no use.henceforth,we are dropping this columns.

```
In [22]: data.columns

Out[22]: Index(['Property_Name', 'Property_type', 'Property_status',
                'Price_per_unit_area', 'Property_building_status', 'No_of_BHK', 'Price',
                'Size', 'is_furnished', 'is_plot', 'is_RERA_registered', 'is_Apartment',
                'is_ready_to_move', 'is_commercial_Listing', 'is_PentaHouse',
                'is_studio', 'City_name', 'Locality_Name', 'Sub_urban_name'],
               dtype='object')
```

1. 'Property_id','Posted_On','Project_URL','builder_id','Builder_name','description','listing_domain_score' this parameters will have no effect on price of house. Hence, we are dropping this columns.
2. "Listing_Category" column infers that all the houses are on sell. Thus, this column isn't representing any distinct attribute. hence, we are dropping this column.
3. We are keeping City_name, Locality_Name, Sub_urban_name columns.'City_id','Locality_ID','Sub_urban_ID' display the same thing. Hence, we are dropping this columns as well.
4. We already have a city_name hence, keeping 'Longitude' and 'Latitude' column will be of no use. Henceforth, we are dropping this columns.

Dropping some more columns:

Dropping some more columns--

In [23]:
```
data[data['is_plot']==True].head(2)
```

Out[23]:

	Property_Name	Property_type	Property_status	Price_per_unit_area	Property_building_status	No_of_BHK	Price	Size	is_furnished	is_plot	is_RERA_registered	is_Apartment	is_ready_to_move	is_commercial_Listing	is_Pen
1191	NaN	Residential Plot	Ready to move	13650	UNVERIFIED	0	43000000	3150	Unfurnished	True	False	False	True	False	
1192	NaN	Residential Plot	Ready to move	518	UNVERIFIED	0	700000	1350	Unfurnished	True	False	False	True	False	

In [24]:
```
data[data['is_plot']==False].head(2)
```

Out[24]:

	Property_Name	Property_type	Property_status	Price_per_unit_area	Property_building_status	No_of_BHK	Price	Size	is_furnished	is_plot	is_RERA_registered	is_Apartment	is_ready_to_move	is_commercial_Listing	is_PentaHo
0	Arkiton Luxe	Apartment	Under Construction	4285	ACTIVE	3	7500000	1750	Unfurnished	False	True	True	False	False	F
1	Arkiton Luxe	Apartment	Under Construction	4285	ACTIVE	3	7500000	1750	Unfurnished	False	True	True	False	False	F

In [25]:
```
data.drop(['is_plot'],axis=1,inplace=True)
```

> We can observe that if property is Residential Plot then 'is_plot' column display True as outcome.which simply means 'is_plot' is revealing one of Property_type only .hence,we are dropping this column.

We can observe that if property is Residential Plot then 'is_plot' column display True as outcome. which simply means 'is_plot' is revealing one of Property_type only .hence, we are dropping this column.

In [26]:
```
data.is_Apartment.unique()
```

Out[26]:
```
array([ True, False])
```

In [27]:
```
data[data['is_Apartment']==True].head(2)
```

Out[27]:

	Property_Name	Property_type	Property_status	Price_per_unit_area	Property_building_status	No_of_BHK	Price	Size	is_furnished	is_RERA_registered	is_Apartment	is_ready_to_move	is_commercial_Listing	is_PentaHouse	is_:
0	Arkiton Luxe	Apartment	Under Construction	4285	ACTIVE	3	7500000	1750	Unfurnished	True	True	False	False	False	
1	Arkiton Luxe	Apartment	Under Construction	4285	ACTIVE	3	7500000	1750	Unfurnished	True	True	False	False	False	

In [28]:
```
data[data['is_Apartment']==False].head(2)
```

Out[28]:

	Property_Name	Property_type	Property_status	Price_per_unit_area	Property_building_status	No_of_BHK	Price	Size	is_furnished	is_RERA_registered	is_Apartment	is_ready_to_move	is_commercial_Listing	is_PentaHouse
205	Avirat Silver Luxuria	Independent House	Under Construction	4300	ACTIVE	4	17845000	4150	Unfurnished	True	False	False	False	False
206	Avirat Silver Luxuria	Independent House	Under Construction	4300	ACTIVE	4	17845000	4150	Unfurnished	True	False	False	False	False

> We can observe that if property is Apartment then 'is_Apartment' column display True as outcome.which simply means 'is_Apartment' is revealing one of Property_type only. hence,we are dropping this column.

In [29]:
```
data.drop(['is_Apartment'],axis=1,inplace=True)
```

We can observe that if property is Apartment then 'is_Apartment' column display True as outcome.which simply means 'is_Apartment' is revealing one of Property_type only. hence,we are dropping this column.

```
In [30]: data['is_ready_to_move'].unique()

Out[30]: array([False,  True])

In [31]: data[data['is_ready_to_move']==False].head(2)
```

Out[31]:

	Property_Name	Property_type	Property_status	Price_per_unit_area	Property_building_status	No_of_BHK	Price	Size	is_furnished	is_RERA_registered	is_ready_to_move	is_commercial_Listing	is_PentaHouse	is_studio	City_n
0	Arkiton Luxe	Apartment	Under Construction	4285	ACTIVE	3	7500000	1750	Unfurnished	True	False	False	False	False	Ahmeda
1	Arkiton Luxe	Apartment	Under Construction	4285	ACTIVE	3	7500000	1750	Unfurnished	True	False	False	False	False	Ahmeda

> We can see that columns 'is_ready_to_move' and 'Property_status' depicts the same thing. hence, we are dropping 'is_ready_to_move' column.

```
In [32]: data.drop('is_ready_to_move',axis=1,inplace=True)
```

We can see that columns 'is_ready_to_move' and 'Property_status' depicts the same thing. hence, we are dropping 'is_ready_to_move' column.

```
In [33]: data['is_commercial_Listing'].unique()

Out[33]: array([False])

In [34]: data.drop('is_commercial_Listing',axis=1,inplace=True)
```

> We can see that column 'is_commercial_Listing' has only one outcome False which simplifies that no house is commercially listed. this column isn't relaying any valuable information.hence,we are dropping this column.

```
In [35]: data['is_studio'].unique()

Out[35]: array([False])

In [36]: data.drop('is_studio',axis=1,inplace=True)
```

> We can see that column 'is_studio' has only one outcome False which depicts that no house is studio. this column isn't relaying any valuable information.hence,we are dropping this column.

```
In [37]: data.columns,data.shape

Out[37]: (Index(['Property_Name', 'Property_type', 'Property_status',
                'Price_per_unit_area', 'Property_building_status', 'No_of_BHK', 'Price',
                'Size', 'is_furnished', 'is_RERA_registered', 'is_PentaHouse',
                'City_name', 'Locality_Name', 'Sub_urban_name'],
               dtype='object'),
          (4935427, 14))
```

1. We can see that column 'is_commercial_Listing' has only one outcome False which simplifies that no house is commercially listed. this column isn't relaying any valuable information.hence,we are dropping this column.
2. We can see that column 'is_studio' has only one outcome False which depicts that no house is studio. this column isn't relaying any valuable information.hence,we are dropping this column.

2. Null Value Treatment:

```
In [38]: data.isnull().sum()

Out[38]: Property_Name              1711591
         Property_type                    0
         Property_status            2895441
         Price_per_unit_area              0
         Property_building_status         0
         No_of_BHK                        0
         Price                            0
         Size                             0
         is_furnished                     0
         is_RERA_registered               0
         is_PentaHouse                    0
         City_name                        0
         Locality_Name                    2
         Sub_urban_name                   0
         dtype: int64
```

> We are going to use Property_Name column to split our data into train and test. So for now lets work on filling Property_status null values.

We are going to use Property_Name column to split our data into train and test. So for now let’s work on filling Property_status null values.

```
In [39]: data['Property_status'].unique()

Out[39]: array(['Under Construction', 'Ready to move', nan], dtype=object)

In [40]: X=data[data["Property_status"].isnull()]
```

> We create a 'X' variable to store data where property status is null. lets examine property type of this null data.

```
In [41]: X["Property_type"].value_counts()

Out[41]: Residential Plot    2895420
         Apartment                18
         Villa                     3
         Name: Property_type, dtype: int64

In [42]: data[data["Property_type"]=='Residential Plot'].head(2)
```

Out[42]:

	Property_Name	Property_type	Property_status	Price_per_unit_area	Property_building_status	No_of_BHK	Price	Size	is_furnished	is_RERA_registered	is_PentaHouse	City_name	Locality_Name	Sub_urban_name
1191	NaN	Residential Plot	Ready to move	13650	UNVERIFIED	0	43000000	3150	Unfurnished	False	False	Ahmedabad	Juhapura	Ahmedabad West
1192	NaN	Residential Plot	Ready to move	518	UNVERIFIED	0	700000	1350	Unfurnished	False	False	Ahmedabad	Goraj	Other

> We can note that Property_status of Residential Plot is 'Ready to move'.accordingly lets fill the Property_status as 'Ready to move' for nan values.but prior to filling we are dropping data where 'Property_status' is null and 'Property_type' is either Apartment or Villa.(Note that they are very few in numbers.)

```
In [43]: i=data[(data['Property_status'].isnull()) & ((data['Property_type']=='Apartment') | (data['Property_type']=='Villa'))].index

In [44]: data.drop(i,inplace=True)

In [45]: data['Property_status'].fillna('Ready to move',inplace=True)
```

We can note that Property_status of Residential Plot is 'Ready to move'.accordingly lets fill the Property_status as 'Ready to move' for nan values.but prior to filling we are dropping data where 'Property_status' is null and 'Property_type' is either Apartment or Villa.(Note that they are very few in numbers.)

Dropping null values from Locality_Name:

Dropping null values from Locality_Name--

```
In [46]: y=data[data['Locality_Name'].isnull()].index

In [47]: data.drop(y,inplace=True)

In [48]: data.isnull().sum()

Out[48]: Property_Name               1711591
         Property_type                     0
         Property_status                   0
         Price_per_unit_area               0
         Property_building_status          0
         No_of_BHK                         0
         Price                             0
         Size                              0
         is_furnished                      0
         is_RERA_registered                0
         is_PentaHouse                     0
         City_name                         0
         Locality_Name                     0
         Sub_urban_name                    0
         dtype: int64
```

Changing few more datatypes:

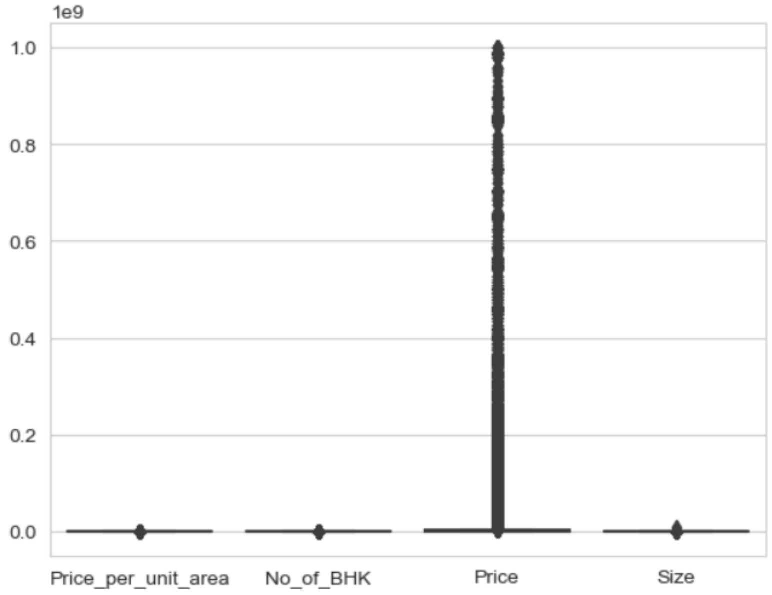
Changing datatype--

```
In [50]: data['is_RERA_registered'].unique(),data['is_PentaHouse'].unique()

Out[50]: (array([ True, False]), array([False,  True]))

In [51]: data['is_RERA_registered'].dtype,data['is_PentaHouse'].dtype

Out[51]: (dtype('bool'), dtype('bool'))

In [52]: data['is_RERA_registered']=data['is_RERA_registered'].astype('object')
         data['is_PentaHouse']=data['is_PentaHouse'].astype('object')

In [53]: data['is_RERA_registered'].unique(),data['is_PentaHouse'].unique()

Out[53]: (array([True, False], dtype=object), array([False, True], dtype=object))

In [54]: data.dtypes

Out[54]: Property_Name               object
         Property_type               object
         Property_status             object
         Price_per_unit_area          int32
         Property_building_status    object
         No_of_BHK                    int32
         Price                        int32
         Size                         int32
         is_furnished                object
         is_RERA_registered          object
         is_PentaHouse               object
         City_name                   object
         Locality_Name               object
         Sub_urban_name              object
         dtype: object
```

```
In [55]: data.columns

Out[55]: Index(['Property_Name', 'Property_type', 'Property_status',
                'Price_per_unit_area', 'Property_building_status', 'No_of_BHK', 'Price',
                'Size', 'is_furnished', 'is_RERA_registered', 'is_PentaHouse',
                'City_name', 'Locality_Name', 'Sub_urban_name'],
               dtype='object')

In [56]: data.head(2),data.shape

Out[56]: (  Property_Name Property_type     Property_status  Price_per_unit_area  \
          0  Arkiton Luxe     Apartment  Under Construction                 4285
          1  Arkiton Luxe     Apartment  Under Construction                 4285

            Property_building_status  No_of_BHK    Price  Size is_furnished  \
          0                   ACTIVE          3  7500000  1750  Unfurnished
          1                   ACTIVE          3  7500000  1750  Unfurnished

            is_RERA_registered is_PentaHouse  City_name Locality_Name Sub_urban_name
          0               True         False  Ahmedabad          Bopal  Ahmedabad West
          1               True         False  Ahmedabad          Bopal  Ahmedabad West  ,
          (4935404, 14))
```

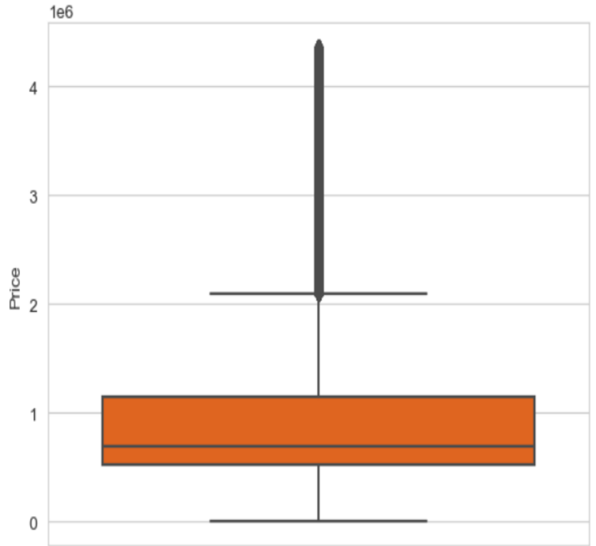
3. Outlier Treatment:

```
In [57]: sns.boxplot(data=data.loc[:,[ 'Price_per_unit_area','No_of_BHK', 'Price','Size']])
Out[57]: <AxesSubplot:>
```



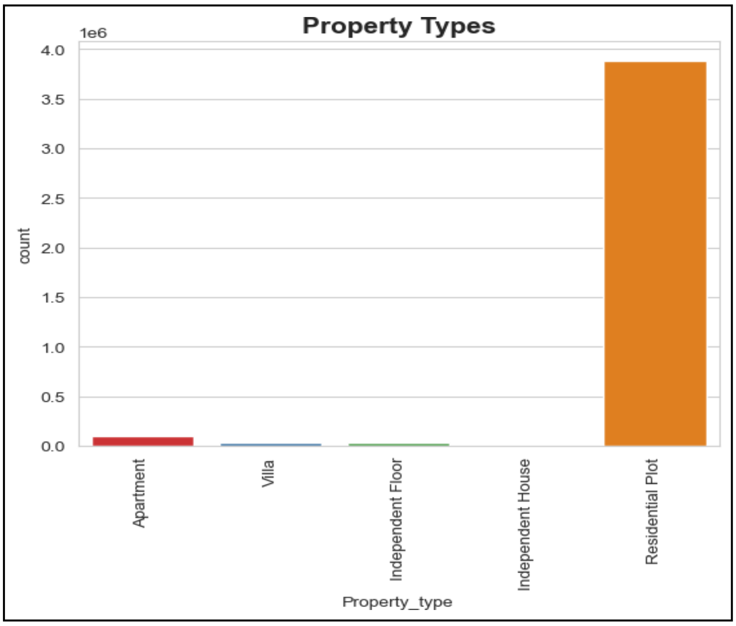
Box plot shows the distribution of the data points by dividing them into different quartiles. Box plot marks lower quartile, median and upper quartile, Any data points which lie outside of the box are treated as outliers.

```
In [58]: Q1 = data.Price.quantile(0.25)
         Q3 = data.Price.quantile(0.75)
         IQR = Q3 - Q1

         lower = Q1 - 1.5*IQR
         print(lower)
         upper = Q3 + 1.5*IQR
         print(upper)
         data = data[(data.Price > (Q1 - 1.5*IQR)) & (data.Price < (Q3 + 1.5*IQR))]
         sns.boxplot(y='Price',data=data,palette='hot')

         -1661500.0
         4366500.0
Out[58]: <AxesSubplot:ylabel='Price'>
```



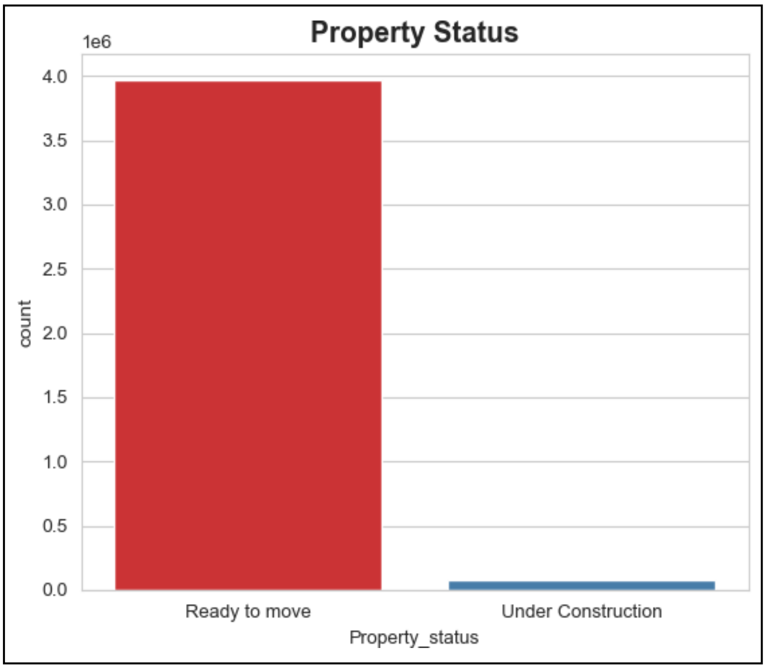
4. Exploratory Data Analysis:

```
In [59]: plt.xticks(rotation=90,fontsize="medium")
         print(data['Property_type'].value_counts())
         sns.countplot(data=data,x=data['Property_type'],palette="Set1")
         plt.title("Property Types",fontweight="bold",fontsize=15)

         Residential Plot     3880736
         Apartment             102400
         Villa                  30613
         Independent Floor      28903
         Independent House       6180
         Name: Property_type, dtype: int64
Out[59]: Text(0.5, 1.0, 'Property Types')
```



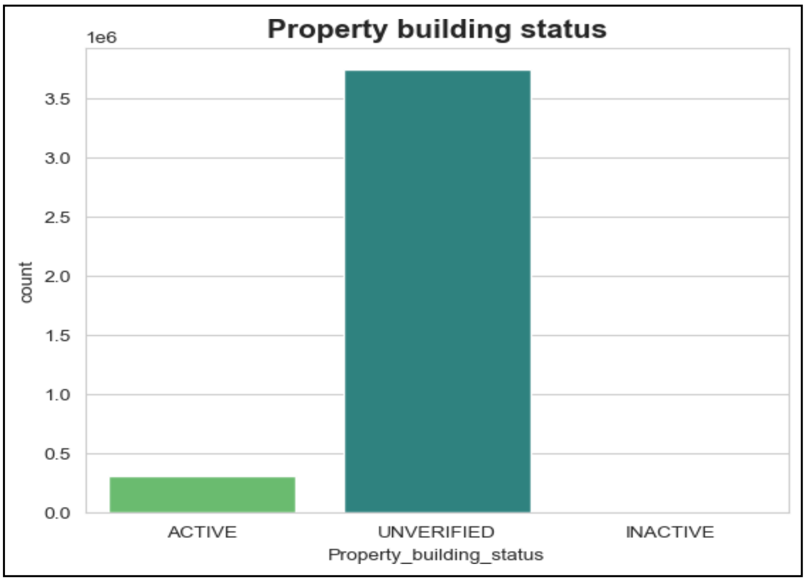
Mostly property is Residential followed by Apartments.

```
In [60]: print(data['Property_status'].value_counts())
         sns.countplot(data=data,x=data['Property_status'],palette="Set1")
         plt.title("Property Status",fontweight="bold",fontsize=15)

         Ready to move         3970666
         Under Construction      78166
         Name: Property_status, dtype: int64
Out[60]: Text(0.5, 1.0, 'Property Status')
```



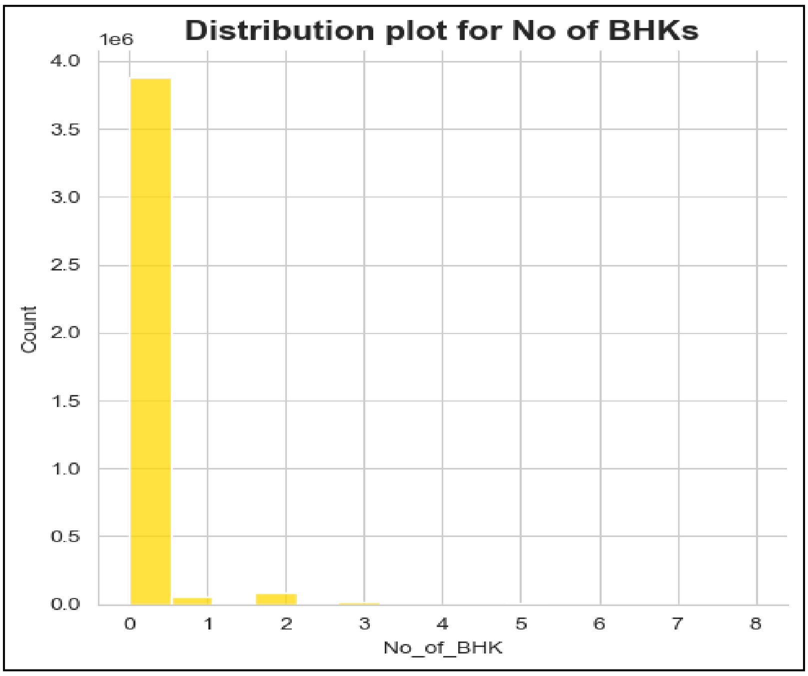
Most plots are ready to move only Few are under construction.

```
In [61]: print(data['Property_building_status'].value_counts())
         sns.countplot(data=data,x=data['Property_building_status'],palette="viridis_r")
         plt.title("Property building status",fontweight="bold",fontsize=15)

         UNVERIFIED    3737414
         ACTIVE         311351
         INACTIVE           67
         Name: Property_building_status, dtype: int64
Out[61]: Text(0.5, 1.0, 'Property building status')
```



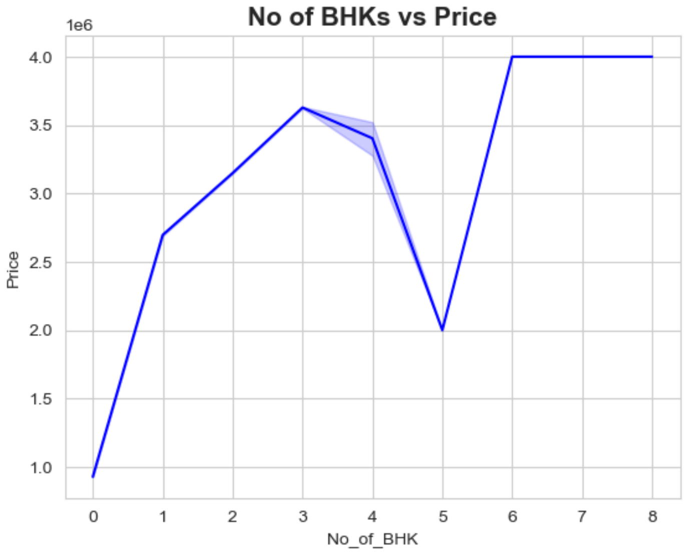
For most properties building status is unverified.

```
In [62]: sns.displot(x='No_of_BHK',data=data,color="gold",bins=15)
         print(data['No_of_BHK'].value_counts())
         plt.title("Distribution plot for No of BHKs",fontweight="bold",fontsize=15)

         0    3880736
         2      86667
         1      53609
         3      20162
         5       7546
         4        110
         8          1
         6          1
         Name: No_of_BHK, dtype: int64
Out[62]: Text(0.5, 1.0, 'Distribution plot for No of BHKs')
```



0 BHKs represents Residential plots thus, from above distribution we can conclude that Residential plots are highly available.

In [63]:
```
sns.lineplot(data=data,x='No_of_BHK',y='Price',color="blue")
plt.title("No of BHKs vs Price",fontweight="bold",fontsize=15)
```
Out[63]: Text(0.5, 1.0, 'No of BHKs vs Price')

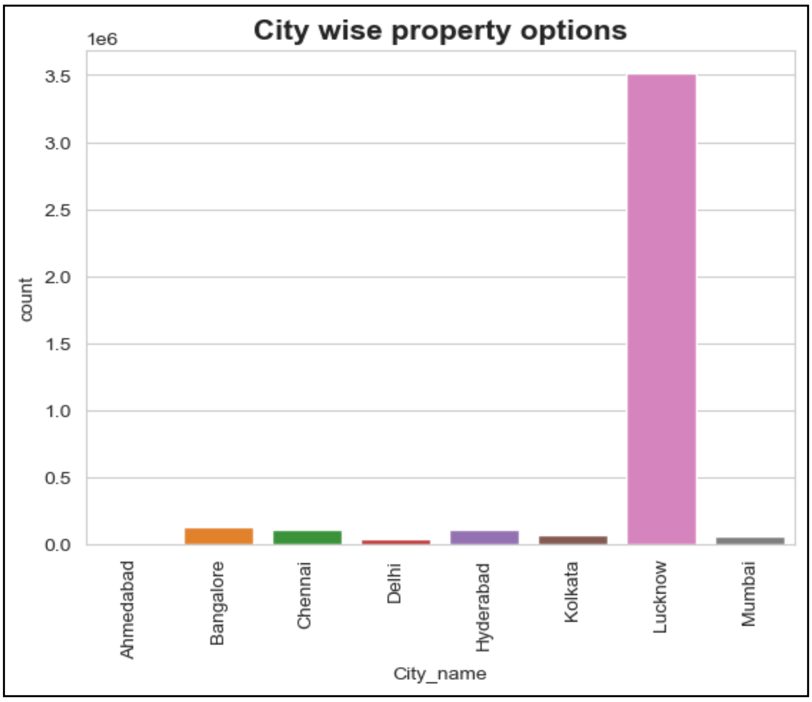


As No of BHK increase from 0 to 3 overall price is also rising however, there is fluctuation in price thereupon.

In [64]:
```
sns.scatterplot(data=data,x='Size',y='Price',color="green")
plt.title("Size vs Price",fontweight="bold",fontsize=15)
plt.xlabel("Size in sq ft")
```
Out[64]: Text(0.5, 0, 'Size in sq ft')

Above scatterplot depicts that Size of property have impact on price.

```
In [65]: plt.xticks(rotation=90,fontsize="medium")
         sns.countplot(x='City_name',data=data)
         plt.title("City wise property options",fontweight="bold",fontsize=15)
         data.City_name.value_counts()

Out[65]: Lucknow      3510702
         Bangalore     129470
         Hyderabad     114286
         Chennai       111825
         Kolkata        70076
         Mumbai         64794
         Delhi          43370
         Ahmedabad       4309
         Name: City_name, dtype: int64
```

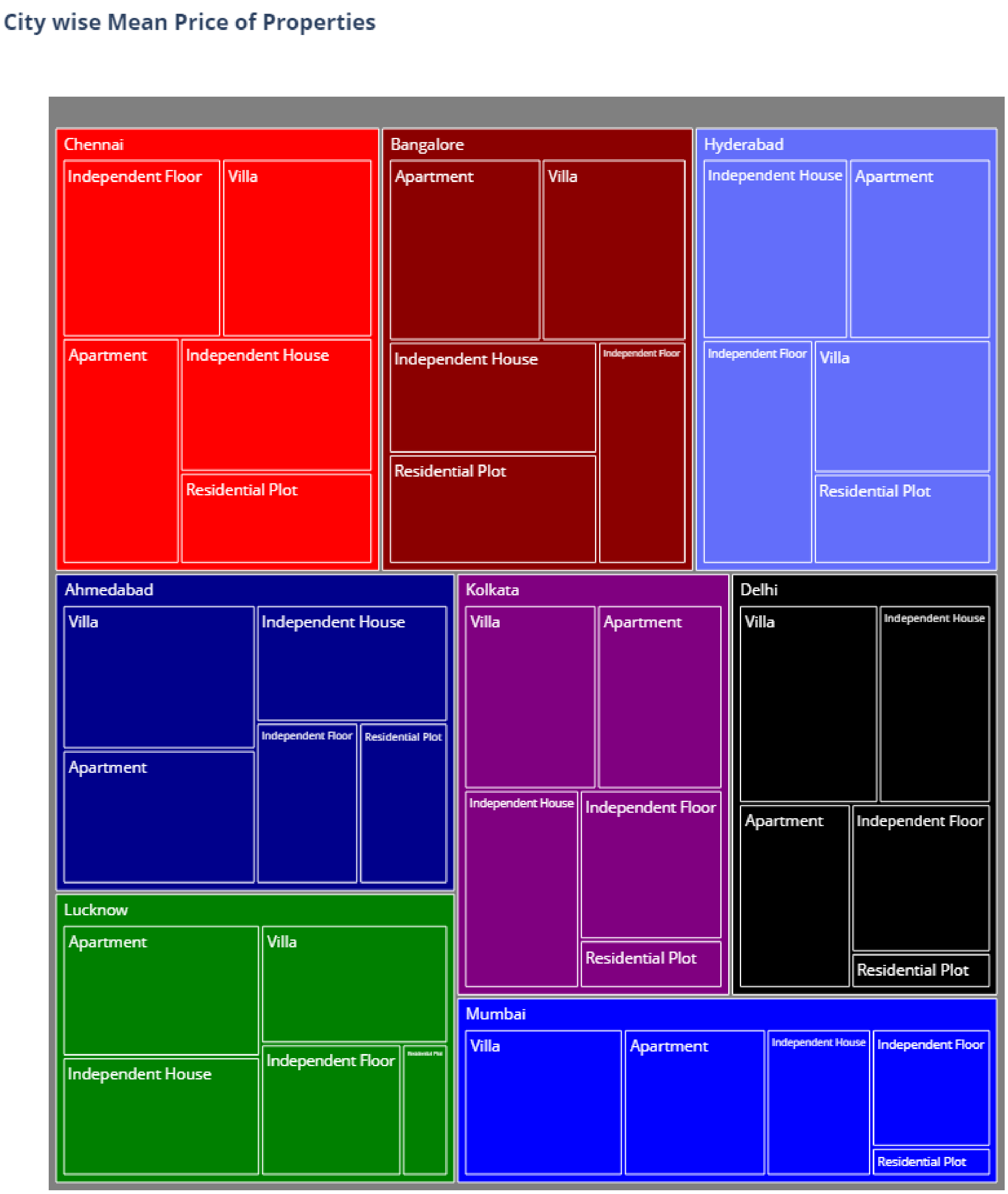


Lucknow has highest number of property options.

```
In [66]: plt.xticks(rotation=90,fontsize="medium")
         sns.barplot(data=data,x='City_name',y="Price",palette="Set2")
         plt.title("City wise Price",fontweight="bold",fontsize=15)

Out[66]: Text(0.5, 1.0, 'City wise Price')
```

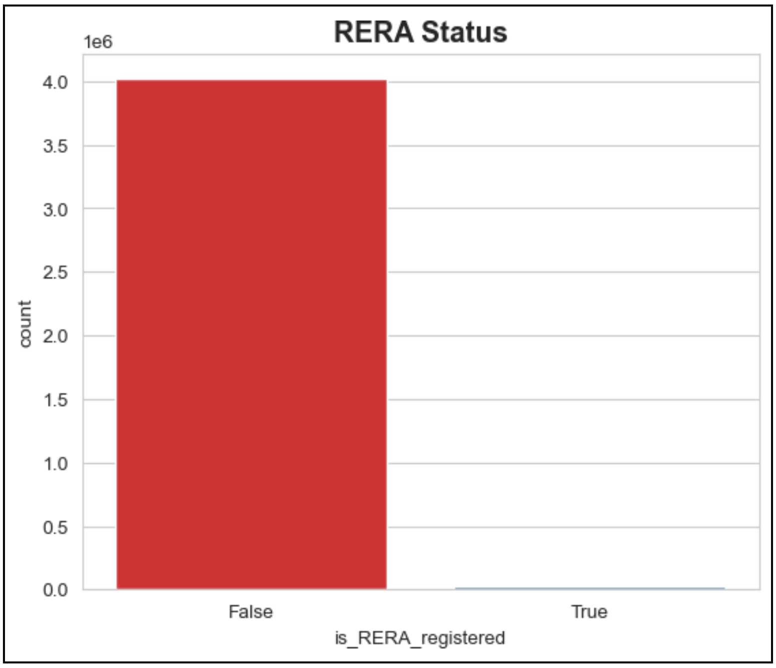
Ahmedabad has highest property prices while Lucknow offers cheaper properties.

```
In [67]: meanprice = data.groupby(["City_name",'Property_type'])['Price'].mean().reset_index()
fig = px.treemap(meanprice, path=["City_name",'Property_type'], values='Price', color='City_name',width=900, height=1000,
             title="Property wise Mean Price in Cities",
        color_discrete_map={'Chennai':'red', 'Ahmedabad':'darkblue',"Delhi":"black","Bangalore":"darkred","Kolkata":"purple",
                     "Mumbai":"blue",'Hyderabad':'darkcoral',"Lucknow":"green"})
fig.update_layout(title='<b>' "City wise Mean Price of Properties" '<b>')
fig.update_traces(root_color="grey")
fig.show(renderer="notebook")
```



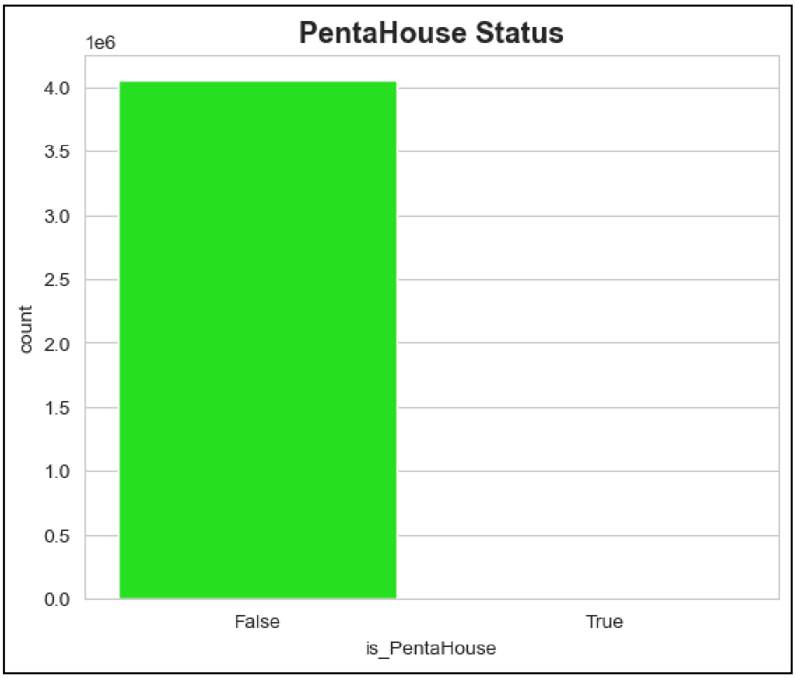
Above treemap illustrates that Independent Floor in Chennai are most expensive whilst Residential plots in Mumbai cheaper among our properties.

```
In [68]: print(data['is_RERA_registered'].value_counts())
         sns.countplot(x=data['is_RERA_registered'],palette="Set1")
         plt.title("RERA Status",fontweight="bold",fontsize=15)

         False    4017967
         True       30865
         Name: is_RERA_registered, dtype: int64
Out[68]: Text(0.5, 1.0, 'RERA Status')
```



Very few properties are registered under RERA.

```
In [69]: print(data['is_PentaHouse'].value_counts())
         sns.countplot(x=data['is_PentaHouse'],palette='hsv')
         plt.title("PentaHouse Status",fontweight="bold",fontsize=15)

         False    4048829
         True           3
         Name: is_PentaHouse, dtype: int64
Out[69]: Text(0.5, 1.0, 'PentaHouse Status')
```

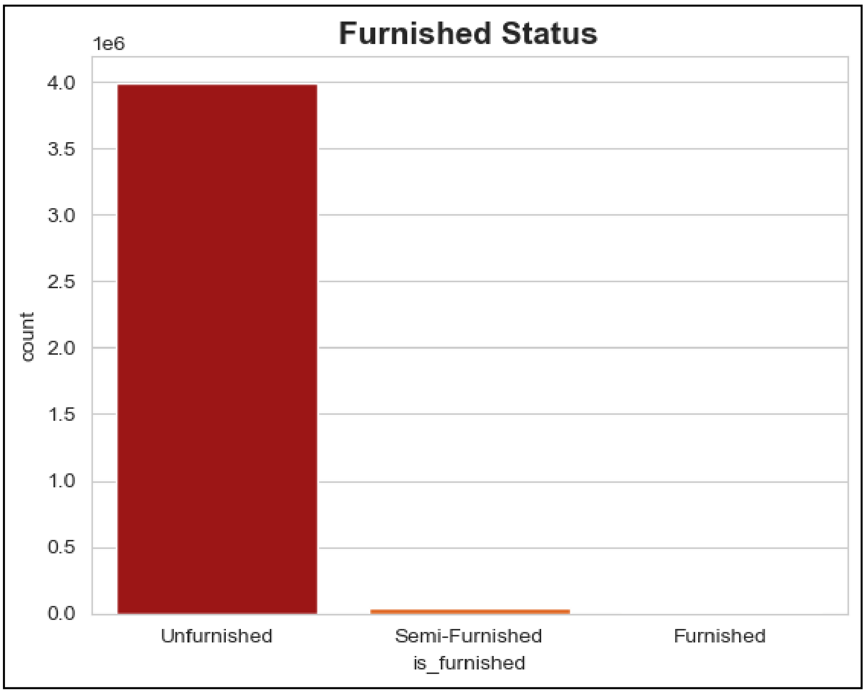


Only 3 Pentahouses are available.

```
In [70]: print(data['is_furnished'].value_counts())
         sns.countplot(data=data,x=data['is_furnished'],palette="hot")
         plt.title("Furnished Status",fontweight="bold",fontsize=15)
```

```
Unfurnished       3991021
Semi-Furnished      43998
Furnished           13813
Name: is_furnished, dtype: int64
```

```
Out[70]: Text(0.5, 1.0, 'Furnished Status')
```



Most of the properties are Unfurnished.

5. Correlation Matrix Heatmap:

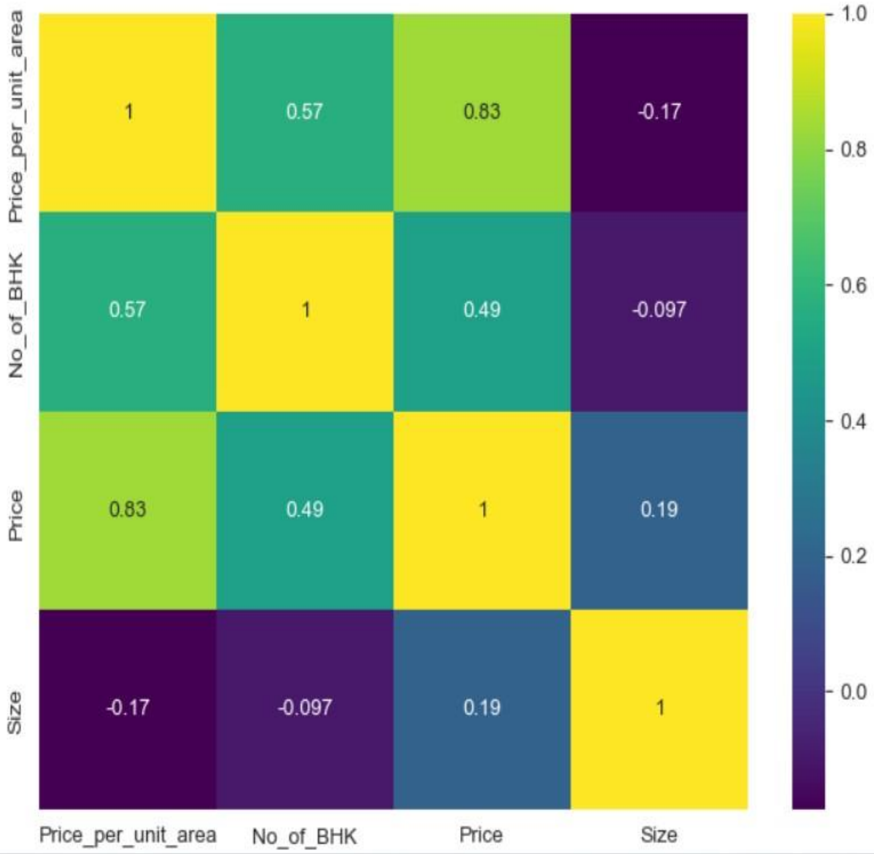
In [71]:
```
data[['Property_Name', 'Property_type', 'Property_status',
      'Price_per_unit_area', 'Property_building_status', 'No_of_BHK', 'Price',
      'Size', 'is_furnished', 'is_RERA_registered', 'is_PentaHouse',
      'City_name', 'Locality_Name', 'Sub_urban_name']].corr()
```

Out[71]:

	Price_per_unit_area	No_of_BHK	Price	Size
Price_per_unit_area	1.000000	0.567791	0.830722	-0.174036
No_of_BHK	0.567791	1.000000	0.489565	-0.096577
Price	0.830722	0.489565	1.000000	0.191478
Size	-0.174036	-0.096577	0.191478	1.000000

In [72]:
```
plt.figure(figsize=(8,6))
sns.heatmap(data[['Property_Name', 'Property_type', 'Property_status',
      'Price_per_unit_area', 'Property_building_status', 'No_of_BHK', 'Price',
      'Size', 'is_furnished', 'is_RERA_registered', 'is_PentaHouse',
      'City_name', 'Locality_Name', 'Sub_urban_name']].corr(),annot=True,cmap="viridis")
```

Out[72]: <AxesSubplot:>



From above Heatmap we can interpret that:

1. Price and Price_per_unit_area are strongly positively correlated.(r=0.83)
2. Price and No_of_BHK are moderately positively correlated. (r=0.49)
3. Price and Size are very weakly positively correlated. (r=0.19)
4. Size and Price_per_unit_area are very weakly negatively correlated. (r= - 0.17)
5. Size and No_of_BHK have no association.(r= - 0.097)

6. Feature Engineering:

1. Encoding Labels:

Encode labels--

```
In [73]: from sklearn.preprocessing import LabelEncoder
         lb= LabelEncoder()
         data['is_RERA_registered']=lb.fit_transform(data['is_RERA_registered'])
         data['is_PentaHouse']=lb.fit_transform(data['is_PentaHouse'])
```

```
In [74]: data['Sub_urban_name'].nunique(),data["Locality_Name"].nunique()

Out[74]: (94, 3396)
```

Dropping Sub_urban_name and Locality_Name--

```
In [75]: data.drop("Sub_urban_name",axis=1,inplace=True)
         data.drop("Locality_Name",axis=1,inplace=True)
         data.head(2)
         data_=data.copy()
```

2. Computing Indicator/ Dummy variables:

Computing indicator / dummy variables--

```
In [76]: data=pd.get_dummies(data,columns=['Property_type','Property_status','Property_building_status','is_furnished','City_name'])
         data.head(2)
```

Out[76]:

	Property_Name	Price_per_unit_area	No_of_BHK	Price	Size	is_RERA_registered	is_PentaHouse	Property_type_Apartment	Property_type_Independent Floor	Property_type_Independent House	Property_type_Residential Plot	Property_type_Villa
22	Satyam Sarjan	2486	2	2283000	918	0	0	1	0	0	0	0
27	Kailash The Willows	2593	2	3385000	1305	1	0	1	0	0	0	0

3. Scaling of data:

Feature Scaling is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing to handle highly varying magnitudes or values or units. Here we apply Standard Scaler because it works better on normally distributed data. Standard Scaler is the type of scaling where the mean is 0 and the variance is 1.

```
In [77]: from sklearn.preprocessing import StandardScaler
         sc= StandardScaler()
         sc1= StandardScaler()
         #Standard scale No_of_BHK, Price_per_unit_area, Size
         data['No_of_BHK']=sc.fit_transform(data[['No_of_BHK']])
         data['Price_per_unit_area']=sc.fit_transform(data[['Price_per_unit_area']])
         data['Size']=sc.fit_transform(data[['Size']])
         data['Price']=sc1.fit_transform(data[['Price']])
         data.head(2)
```

Out[77]:

	Property_Name	Price_per_unit_area	No_of_BHK	Price	Size	is_RERA_registered	is_PentaHouse	Property_type_Apartment	Property_type_Independent Floor	Property_type_Independent House	Property_type_Residential Plot	Property_type
22	Satyam Sarjan	1.578174	4.471819	1.643631	-0.456507	0	0	1	0	0	0	
27	Kailash The Willows	1.691799	4.471819	3.071929	0.711509	1	0	1	0	0	0	

7. Building a model:

1. Splitting of data:

Property name column have Nan values and this is our test data. We are filling this Nan values with ‘T’ prior to defining it as test data.

```
In [78]: print("Splitting data into train and test--")
         data["Property_Name"].fillna('T',inplace=True)
         train=data[data["Property_Name"]!='T']
         test=data[data["Property_Name"]=='T']

         from sklearn.linear_model import LinearRegression
         from sklearn.metrics import r2_score,mean_squared_error

         X_train=pd.concat([train.iloc[:,1:3],train.iloc[:,4:29]],axis=1)
         y_train=train.iloc[:,3]

         X_test=pd.concat([test.iloc[:,1:3],test.iloc[:,4:29]],axis=1)
         y_test=test.iloc[:,3]

         print(X_train.shape),
         print(y_train.shape),
         print(X_test.shape),
         print(y_test.shape)

         Splitting data into train and test--
         (2613771, 26)
         (2613771,)
         (1435061, 26)
         (1435061,)
```

2. Building of a Model:

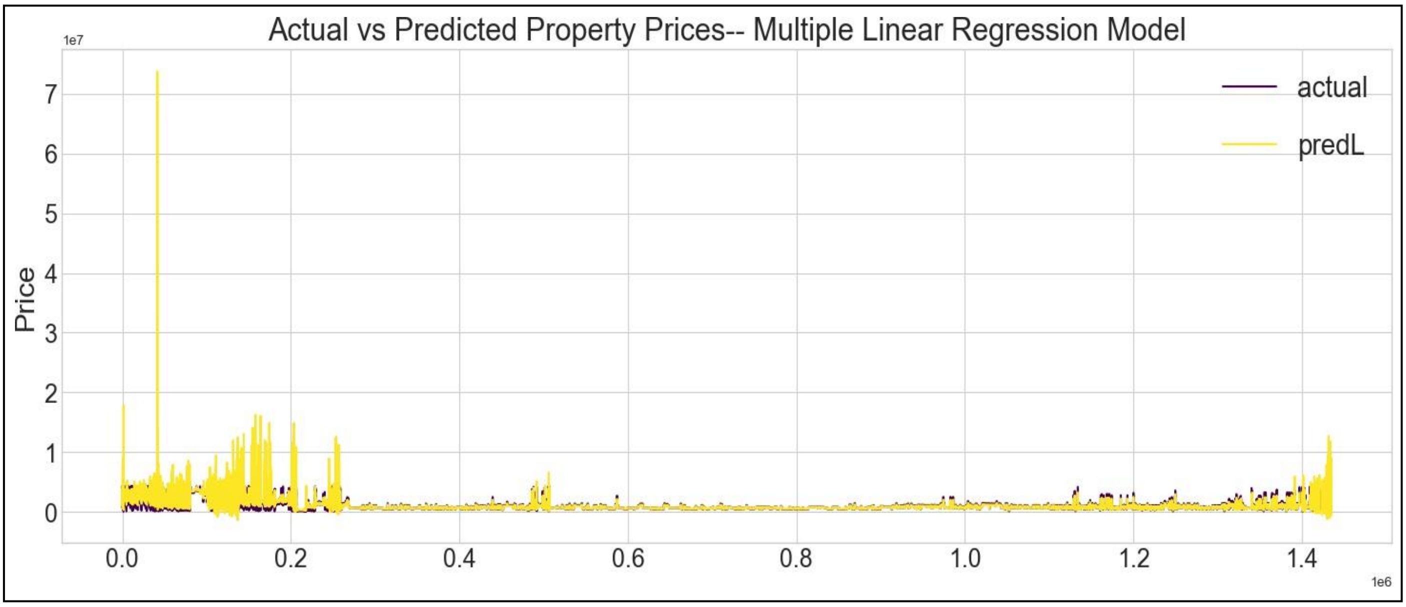
1. Multiple Linear Regression:

```
In [97]: print("Lets build the Multiple Linear regression model")
         def modelling(X_train,y_train,X_test):
             modelL=LinearRegression()
             modelL_train=modelL.fit(X_train,y_train)
             print("ModelL training is completed--")
             return modelL_train
         print("Calling the modelling function--")
         modelL_train=modelling(X_train,y_train,X_test)

         def prediction():
             predL=modelL_train.predict(X_test)
             return predL
         print("Calling prediction function--")
         predL=prediction()
         print(predL)

         r2score_MLR=(round(r2_score(y_test,predL)*100,2))
         rmse = m.sqrt(mean_squared_error(y_test,predL))

         print("Multiple Linear Regression--")
         print('r2score:',r2score_MLR)
         print('RMSE:',rmse)
         print('************************************************')

         Lets build the Multiple Linear regression model
         Calling the modelling function--
         ModelL training is completed--
         Calling prediction function--
         [8.57850647 1.14803314 1.98999023 ... 1.5760498  1.5760498  1.5760498 ]
         Multiple Linear Regression--
         r2score: 86.77
         RMSE: 0.38374152682971274
         ************************************************
```

Inverse transforming Scaled Values:

```
In [80]: actual_scaled= pd.Series(data=y_test, index=test.index)
         pred_scaled=pd.Series(data=predL, index=test.index)
         scaled=pd.concat([actual_scaled,pred_scaled],axis=1)
         scaled.columns = ["actual_scaled", "pred_scaled"]

         print("Inverse transform scaled values--")
         combined=sc1.inverse_transform(scaled)
         df = pd.DataFrame(combined, columns =['actual', 'predL'])
         print(df)
         plt.style.use('seaborn-whitegrid')
         df.plot(figsize= (18,6),colormap="viridis")
         plt.legend(loc='best',bbox_to_anchor=(1,1),labelspacing=1,fontsize=20)
         plt.title("Actual vs Predicted House Prices", fontsize= 22)
         plt.ylabel("Price",fontsize = 20)
         plt.xticks(fontsize = 18)
         plt.yticks(fontsize = 18)
         plt.show()
```

```
Inverse transform scaled values--
            actual         predL
0        4300000.0  7.633588e+06
1        2700000.0  1.900622e+06
2        2500000.0  2.550233e+06
3        2200000.0  2.008509e+06
4        4200000.0  5.346016e+06
...            ...           ...
1435056   783650.0  7.252410e+05
1435057  1700000.0  2.230858e+06
1435058  1700000.0  2.230858e+06
1435059  1700000.0  2.230858e+06
1435060  1700000.0  2.230858e+06

[1435061 rows x 2 columns]
```



2. Ridge Regression:

```
In [98]: print("Lets build the Ridge regression model")
         from sklearn.linear_model import Ridge
         def modelling1(X_train,y_train,X_test):
             model1=Ridge()
             model1_train=model1.fit(X_train,y_train)
             print("Model1 training is completed--")
             return model1_train
         print("Calling the modelling1 function--")
         model1_train=modelling1(X_train,y_train,X_test)

         def prediction():
             pred1=model1_train.predict(X_test)
             return pred1
         print("Calling prediction function--")
         pred1=prediction()
         print(pred1)

         r2score_Ridge=(round(r2_score(y_test,pred1)*100,2))
         rmse = m.sqrt(mean_squared_error(y_test,pred1))

         print("Ridge Regression--")
         print('r2score:',r2score_Ridge)
         print('RMSE:',rmse)
         print('**************************************************')
```

```
Lets build the Ridge regression model
Calling the modelling1 function--
Model1 training is completed--
Calling prediction function--
[8.57746535 1.15553819 1.99790626 ... 1.57330409 1.57330409 1.57330409]
Ridge Regression--
r2score: 86.76
RMSE: 0.38388433184990983
**************************************************
```

3. Decision Tree Regression:

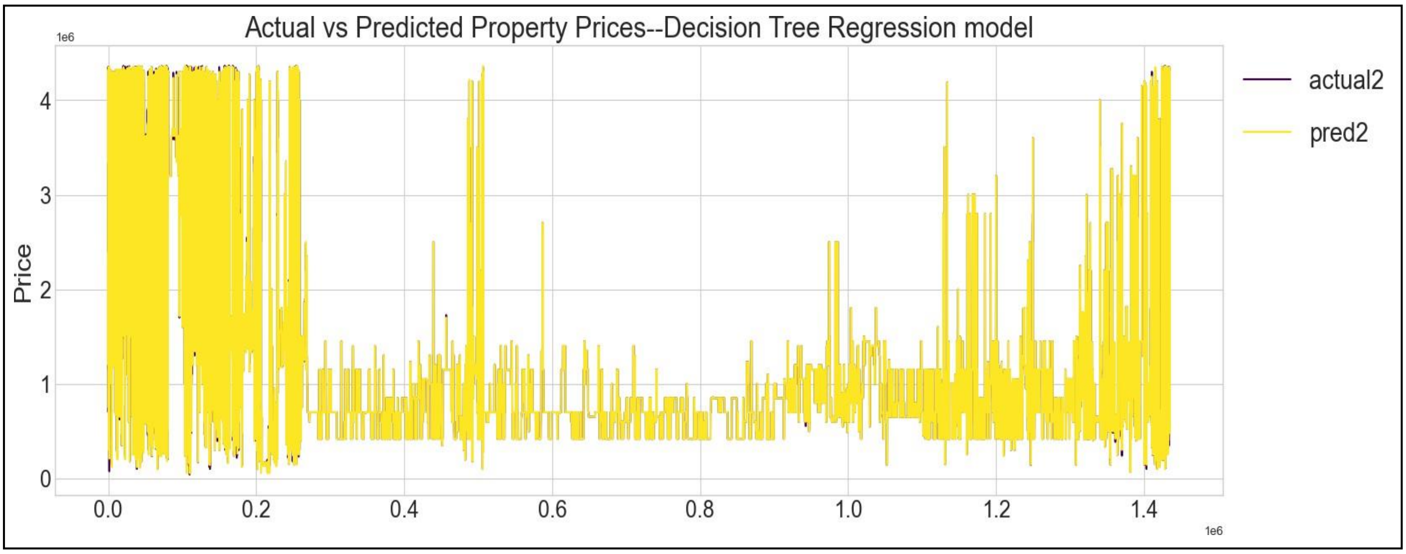
```
In [99]: print("Lets build the Decision Tree Regression model")
         from sklearn.tree import DecisionTreeRegressor
         def modelling2():
             model2=DecisionTreeRegressor(criterion='squared_error')
             model2_train=model2.fit(X_train,y_train)
             print("Model training is completed.")
             return model2_train
         print("Calling modelling2 function--")
         model2_train=modelling2()

         def prediction():
             pred2=model2_train.predict(X_test)
             return pred2
         print("Calling prediction function--")
         pred2=prediction()
         print(pred2)

         r2score_DT=(round(r2_score(y_test,pred2)*100,2))
         rmse = m.sqrt(mean_squared_error(y_test,pred2))
         print("Decision Tree Regression--")
         print('r2score:',r2score_DT)
         print('RMSE:',rmse)
         print('**************************************************')
```

```
Lets build the Decision Tree Regression model
Calling modelling2 function--
Model training is completed.
Calling prediction function--
[4.25785659 2.31371278 1.92067819 ... 0.91392924 0.91392924 0.91392924]
Decision Tree Regression--
r2score: 99.96
RMSE: 0.02153664861365922
**************************************************
```

Inverse transforming Scaled Values:

```
In [102…  actual_scaled2= pd.Series(data=y_test, index=test.index)
          pred_scaled2=pd.Series(data=pred2, index=test.index)
          scaled2=pd.concat([actual_scaled2,pred_scaled2],axis=1)
          scaled2.columns = ["actual_scaled2", "pred_scaled2"]

          print("Inverse transform scaled values--")
          combined2=sc1.inverse_transform(scaled2)
          df2 = pd.DataFrame(combined2, columns =['actual2', 'pred2'])
          print(df2)
          plt.style.use('seaborn-whitegrid')
          df2.plot(figsize= (18,6),colormap="viridis")
          plt.legend(loc='best',bbox_to_anchor=(1,1),labelspacing=1,fontsize=20)
          plt.title("Actual vs Predicted House Prices--Decision Tree Regression model", fontsize= 22)
          plt.ylabel("Price",fontsize = 20)
          plt.xticks(fontsize = 18)
          plt.yticks(fontsize = 18)
          plt.show()
```

```
Inverse transform scaled values--
           actual2      pred2
0        4300000.0  4300000.0
1        2700000.0  2800000.0
2        2500000.0  2496755.0
3        2200000.0  2200000.0
4        4200000.0  4200000.0
...            ...        ...
1435056   783650.0   774000.0
1435057  1700000.0  1720000.0
1435058  1700000.0  1720000.0
1435059  1700000.0  1720000.0
1435060  1700000.0  1720000.0

[1435061 rows x 2 columns]
```



4. Random Forest Regression:

```
In [103…  print("Lets build the Random Forest Regression model")
          from sklearn.ensemble import RandomForestRegressor
          def modelling3():
              model3=RandomForestRegressor(criterion="squared_error")
              model3_train=model3.fit(X_train,y_train)
              print("Model training is completed.")
              return model3_train
          print("Calling modelling3 function--")
          model3_train=modelling3()

          def prediction():
              pred3=model3_train.predict(X_test)
              return pred3
          print("Calling prediction function--")
          pred3=prediction()
          print(pred3)

          r2score_RF=(round(r2_score(y_test,pred3)*100,2))
          rmse = m.sqrt(mean_squared_error(y_test,pred3))
          print("Random Forest Regression--")
          print('r2score:',r2score_RF)
          print('RMSE:',rmse)
          print('************************************************')
```

```
Lets build the Random Forest Regression model
Calling modelling3 function--
Model training is completed.
Calling prediction function--
[4.23469776 2.22689789 1.92391668 ... 0.85411621 0.85411621 0.85411621]
Random Forest Regression--
r2score: 99.99
RMSE: 0.012081843006319277
***********************************************
```

```
details = {
    'Model' : ['Multiple Linear Regression', 'Ridge Regression', 'Decision Tree Regression', 'Random Forest'],
    'Accuracy %' : [r2score_MLR, r2score_Ridge, r2score_DT,r2score_RF]}
df = pd.DataFrame(details)
df
```

	Model	Accuracy %
0	Multiple Linear Regression	86.76
1	Ridge Regression	86.76
2	Decision Tree Regression	99.96
3	Random Forest	99.99

1. Random Forest models are giving highest accuracy.
2. Although here we are choosing model with optimum accuracy. We will consider Property prices predicted by Multiple Linear Regression model for our further analysis.

8. Saving the model using joblib:

In [110…

```
import joblib
```

Saving the Model Using Joblib--

In [111…

```
joblib.dump(modelL_train,"Makaan_Linear_Model.pkl")
joblib.dump(modelL_train,"Makaan_Linear_Model.joblib")
```

Out[111]:

```
['Makaan_Linear_Model.joblib']
```

Loading the Saved Model Using Joblib--

In [112…

```
reg= joblib.load("Makaan_Linear_Model.joblib")
```

In [113…

```
predictions = reg.predict(X_test)
predictions
```

Out[113]:

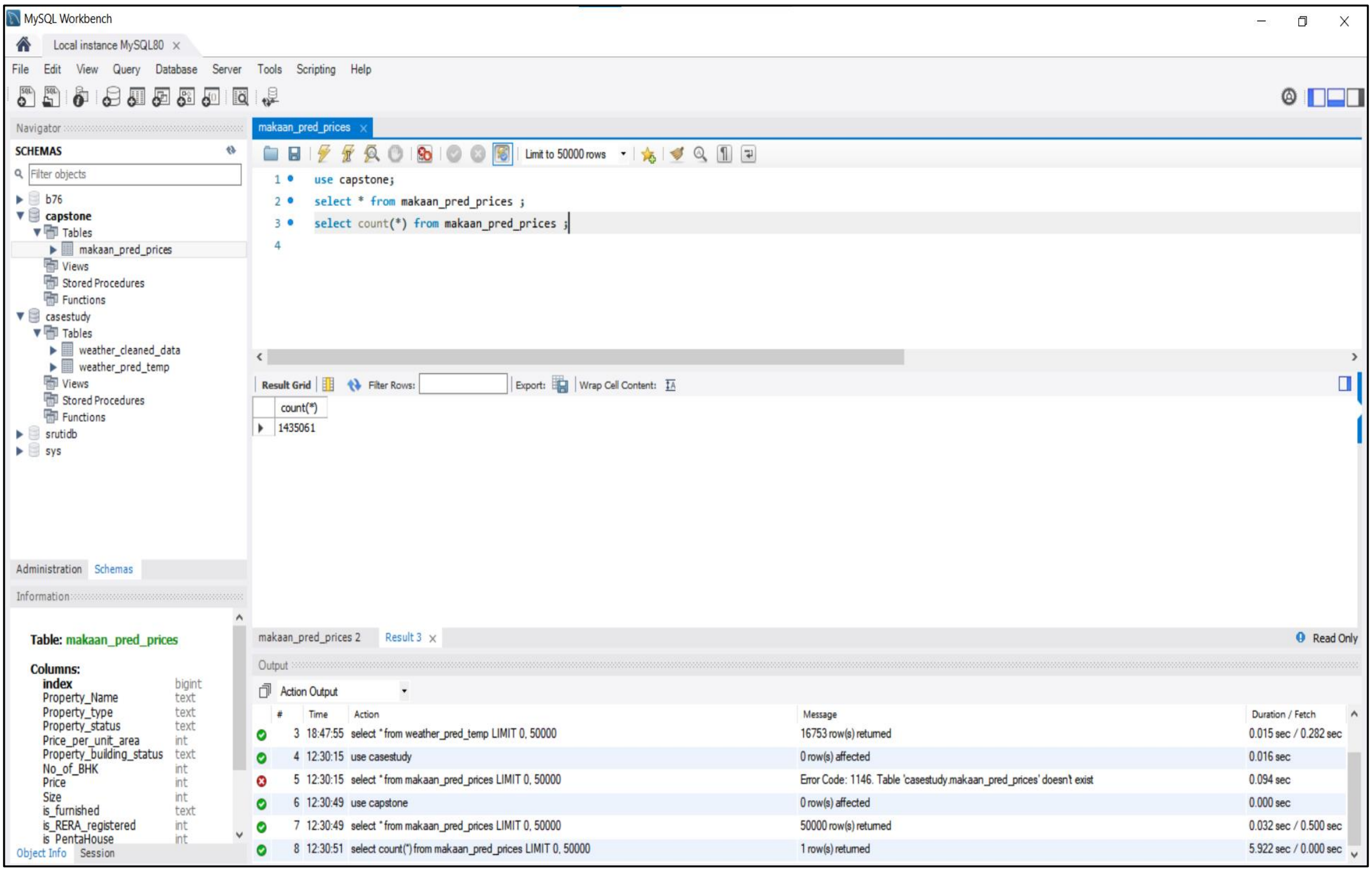
```
array([8.57850647, 1.14803314, 1.98999023, ..., 1.5760498 , 1.5760498 ,
       1.5760498 ])
```

We save our model using joblib. Besides we test the model to predict property prices.

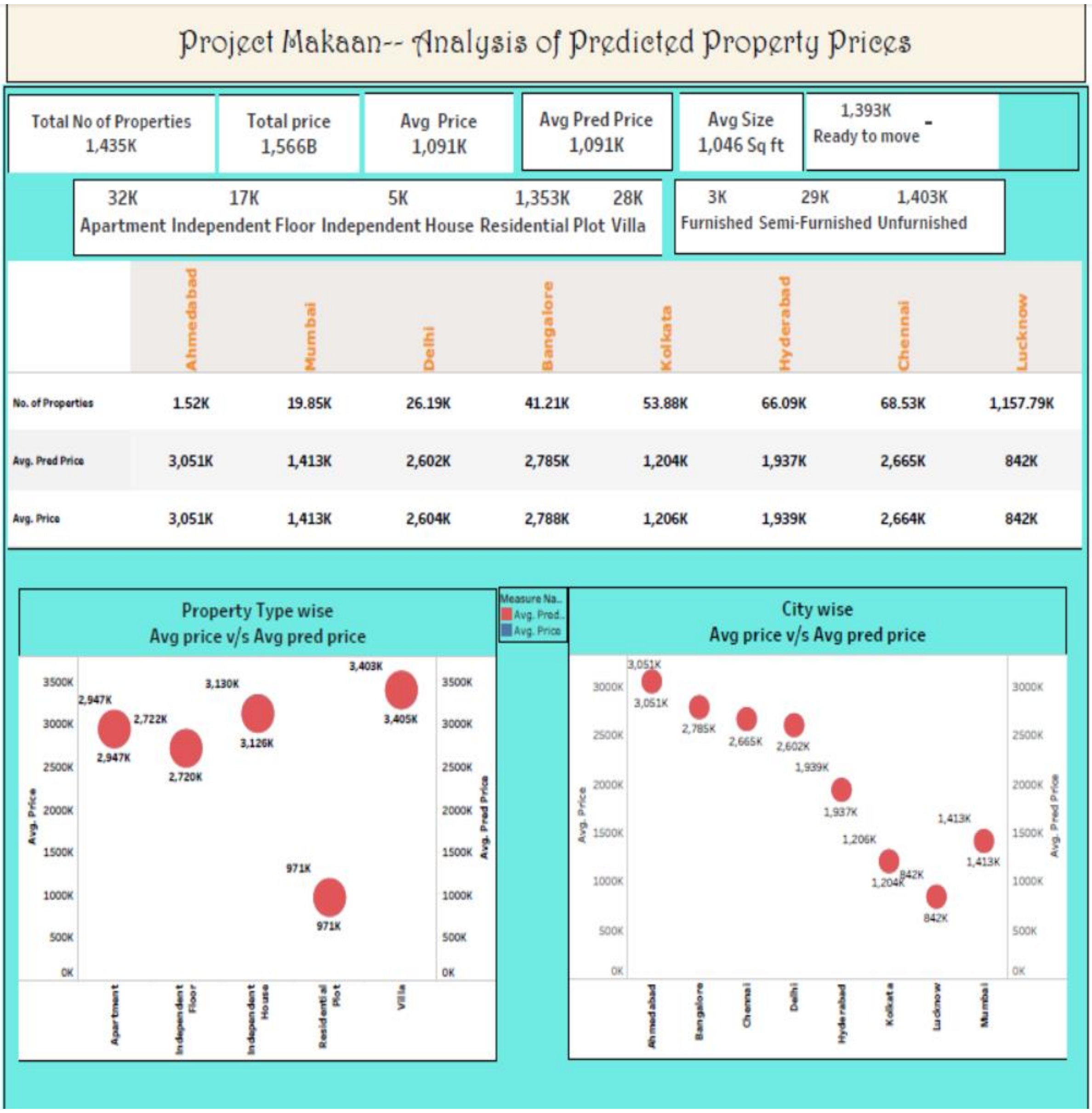
9. Importing Property prices predicted by Multiple Linear Regression model to MySQL:

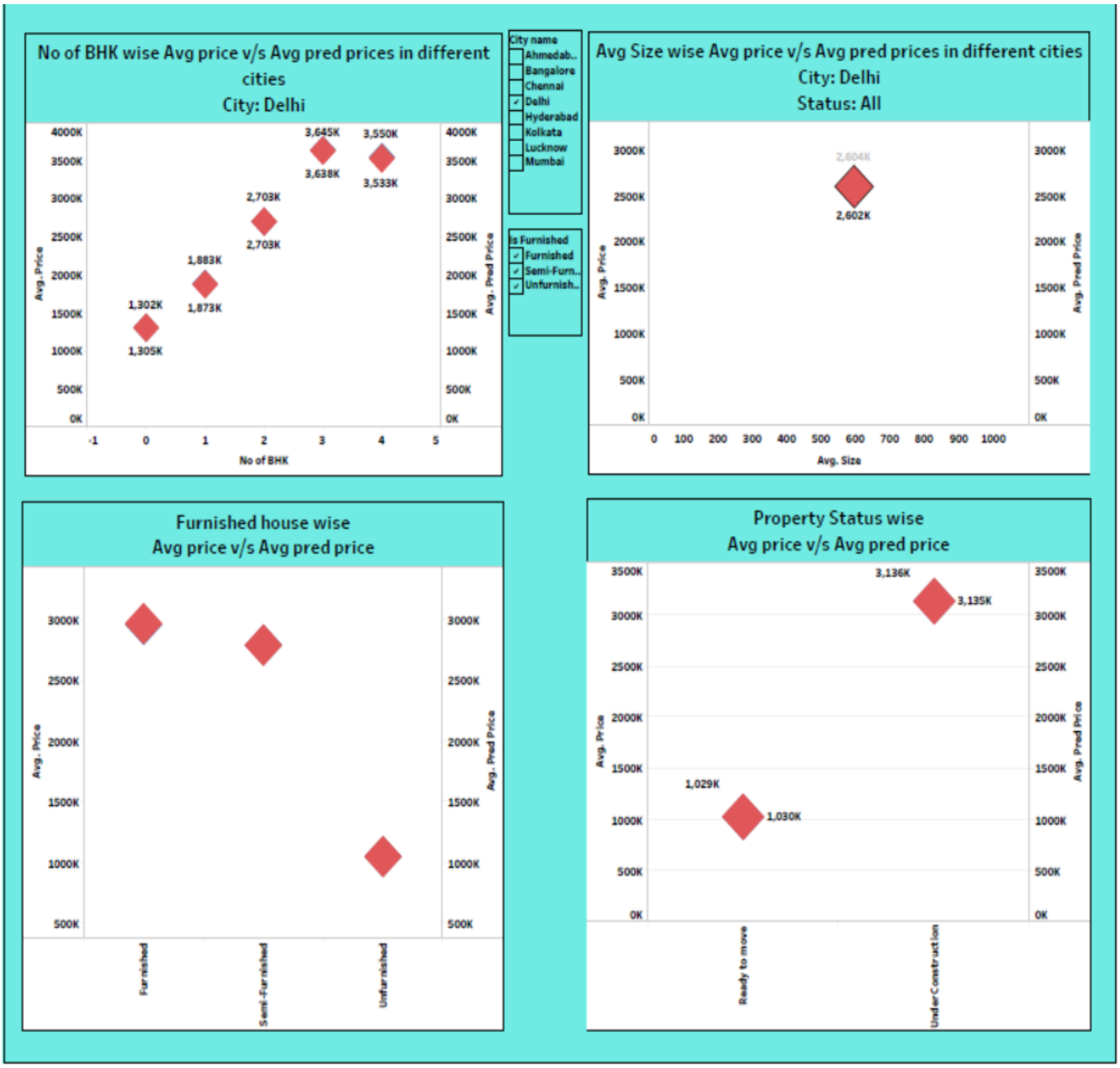
```
data_["Property_Name"].fillna('T',inplace=True)
test_=data_[data_["Property_Name"]=='T']
test_["Pred_Price"]=predL
#inverse_transform--
test_["Pred_Price"]=sc1.inverse_transform(test_["Pred_Price"].values.reshape(-1,1))

from sqlalchemy import create_engine
engine = create_engine("mysql+pymysql://root:Fuchka%40104@localhost/capstone")
con=engine.connect()
test_.to_sql(con=con,name="makaan_pred_prices",if_exists="replace")
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_12772\1285901267.py:3: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy

C:\Users\hp\AppData\Local\Temp\ipykernel_12772\1285901267.py:5: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy

1435061
```



10. Final Dashboard of predicted house prices prepared Using Tableau:




No of BHK wise Avg price v/s Avg pred prices in different cities
City: Delhi
Avg. Price
Avg. Pred Price
No of BHK
1,302K
1,305K
1,883K
1,873K
2,703K
2,703K
3,645K
3,638K
3,550K
3,533K
City name
Ahmedab..
Bangalore
Chennai
Delhi
Hyderabad
Kolkata
Lucknow
Mumbai
Is Furnished
Furnished
Semi-Furn..
Unfurnish..
Avg Size wise Avg price v/s Avg pred prices in different cities
City: Delhi
Status: All
Avg. Size
2,604K
2,602K
Furnished house wise
Avg price v/s Avg pred price
Furnished
Semi-Furnished
Unfurnished
Property Status wise
Avg price v/s Avg pred price
1,029K
1,030K
3,136K
3,135K
Ready to move
Under Construction