Problem Statement :

A machine learning model is to be proposed to predict a house price based on data related to the house i.e. its area_type, availability, location, size, society, total_sqft, bath and balcony

Goals of the Study:

The main objectives of this case study are as follows:

1. To apply data preprocessing and preparation techniques in order to obtain clean data (EDA)
2. To build machine learning models able to predict house price based on house features
3. To analyze and compare models performance in order to choose the best model

Importing Required Libraries for EDA

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

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

Importing the Data

In [2]:
```
data = pd.read_csv("C:\\Users\\mehak\\Downloads\\Machine learning\\House_Data.csv")
```

In [3]:
```
# Top 10 records
data.head(10)
```

Out[3]:

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	Plot Area	Ready To Move	Sarjapur Road	4 Bedroom	NaN	1	4.0	NaN	120.0
1	Built-up Area	Ready To Move	Srirampuram	7 BHK	NaN	5	7.0	3.0	115.0
2	Plot Area	18-Dec	Suragajakkanahalli	3 Bedroom	PrhyaK	11	3.0	2.0	74.0
3	Carpet Area	Ready To Move	Weavers Colony	1 BHK	NaN	15	1.0	0.0	30.0
4	Built-up Area	Ready To Move	Grihalakshmi Layout	5 Bedroom	NaN	24	2.0	2.0	150.0
5	Plot Area	Ready To Move	Mysore Road	1 Bedroom	NaN	45	1.0	0.0	23.0
6	Plot Area	19-Oct	Whitefield	4 Bedroom	NVaree	60	4.0	2.0	218.0
7	Super built-up Area	Ready To Move	Tilak Nagar	1 BHK	NaN	250	2.0	2.0	40.0
8	Plot Area	Ready To Move	Hennur Road	2 Bedroom	NaN	276	3.0	3.0	23.0
9	Super built-up Area	Ready To Move	Yelahanka New Town	1 BHK	KHatsFl	284	1.0	1.0	8.0

In [4]:
```
# Last 10 records
data.tail(10)
```

Out[4]:

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
13310	Super built-up Area	Ready To Move	Rajapura	2 BHK	NaN	86.72Sq. Meter	2.0	2.0	40.000
13311	Super built-up Area	19-Jul	Sarjapur Road	2 BHK	Sasta S	870 - 1080	2.0	0.0	28.275
13312	Super built-up Area	18-Aug	Magadi Road	2 BHK	Vrenty	884 - 1116	2.0	0.0	46.500
13313	Super built-up Area	18-Oct	Electronic City Phase II	2 BHK	SRhtsa	888 - 1290	2.0	0.0	32.670
13314	Super built-up Area	Ready To Move	Hoskote	2 BHK	Soose P	929 - 1078	2.0	0.0	28.095
13315	Super built-up Area	18-Nov	Thanisandra	2 BHK	Bhe 2ko	934 - 1437	2.0	0.0	58.680
13316	Super built-up Area	18-May	Mysore Road	2 BHK	Brama P	942 - 1117	2.0	0.0	50.855
13317	Super built-up Area	Ready To Move	Hormavu	2 BHK	SKvanin	943 - 1220	2.0	0.0	38.665
13318	Super built-up Area	18-Jun	Mysore Road	2 BHK	Gopia O	980 - 1030	2.0	0.0	35.175
13319	Super built-up Area	20-Dec	Whitefield	2 BHK	Somns T	981 - 1249	2.0	0.0	34.555

Understanding and Pre-Processing of Data

In [5]:
```
data.shape
```
Out[5]:(13320, 9)

In [6]:
```
# There are 13320 rows and 9 columns.Let's have a look at all columns and their respective data types.
data.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13320 entries, 0 to 13319
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   area_type     13320 non-null  object 
 1   availability  13320 non-null  object 
 2   location      13319 non-null  object 
 3   size          13304 non-null  object 
 4   society       7818 non-null   object 
 5   total_sqft    13320 non-null  object 
 6   bath          13247 non-null  float64
 7   balcony       12711 non-null  float64
 8   price         13320 non-null  float64
dtypes: float64(3), object(6)
memory usage: 936.7+ KB
```

Null Value Check

In [7]:
```
# Gives % of null values for each column
round(data.isna().sum()*100/data.shape[0],2)
```
Out[7]:
```
area_type        0.00
availability     0.00
location         0.01
size             0.12
society         41.31
total_sqft       0.00
bath             0.55
balcony          4.57
price            0.00
dtype: float64
```

Dropping columns with excessive Null values

In [8]:
```
# Since society has 41% null values, we will drop it
data.drop(columns=['society'],inplace=True)
```

Dropping rows with Null values

In [9]:
```
# Dropping rows that have null values for size
data.dropna(subset=['size'], how='all', inplace=True)
```
In [10]:
```
data.shape
```
Out[10]:(13304, 8)

Filling of Null Values

In [11]:
```
# Location has 1 null value which we will replace with the mode of location values
data.location.fillna(data.location.mode()[0],inplace=True)
```
In [12]:
```
# Null bath values can be replaced with 1.Houses will have atleast one bathroom
```

```
data.bath.fillna(1,inplace=True)
```

In [13]:
```
# Null balcony values can be replaced with 0
data.balcony.fillna(0,inplace=True)
```

In [14]:
```
# All null values are dealt with
data.isna().sum()
```

Out[14]:
```
area_type       0
availability    0
location        0
size            0
total_sqft      0
bath            0
balcony         0
price           0
dtype: int64
```

Conversion of square feet values

In [15]:
```
# Notice that total sqft is of object data type, change that to float by using conversion() function.
# This function converts all other area units to square feet as well.

def conversion(x) :
    try :
        if 'Perch' in x :
            x = x.replace('Perch','')
            return float(x)*272.3

        elif 'Sq. Meter' in x:
            x = x.replace('Sq. Meter','')
            return float(x)*10.764

        elif 'Sq. Yards' in x:
            x = x.replace('Sq. Yards','')
            return float(x)*9

        elif 'Acres' in x:
            x = x.replace('Acres','')
            return float(x)*43560

        elif 'Cents' in x :
            x = x.replace('Cents','')
            return float(x)*435.56

        elif 'Guntha' in x:
            x = x.replace('Guntha','')
            return float(x)*1089

        elif 'Grounds' in x :
            x = x.replace('Grounds','')
            return float(x)*2400.35

        else:
            return float(x)

    except:
        # Exception occurs in cases where area range is given (for eg 2100-2800)
        list_ = x.split("-")
        arr = np.array(list_,dtype=float)
        return arr.mean()  # In this case mean is returned

data.total_sqft = data.total_sqft.apply(lambda x : conversion(x))
```

Transforming Size column

In [16]:
```
# Also size column can be converted to integer which shows 'bhk' value
data['bhk']=data['size'].str.split().str.get(0).astype(int)
```

In [17]:
```
# Now we don't know need size column so drop it
data.drop(columns=['size'],inplace=True)
```

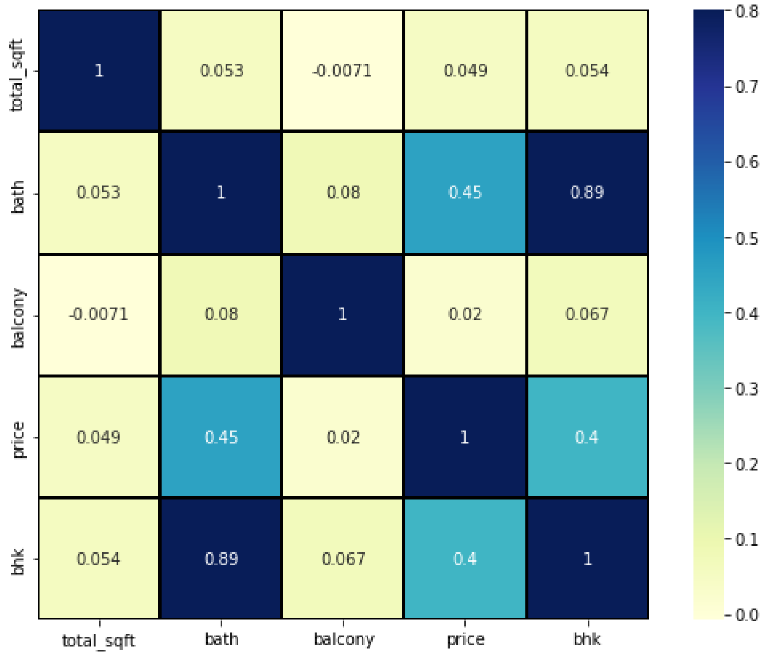
Correlation Check

In [18]:
```
data.corr()
```

Out[18]:

	total_sqft	bath	balcony	price	bhk
total_sqft	1.000000	0.052777	-0.007114	0.048976	0.054270
bath	0.052777	1.000000	0.079789	0.452434	0.892177
balcony	-0.007114	0.079789	1.000000	0.019501	0.067480
price	0.048976	0.452434	0.019501	1.000000	0.398292
bhk	0.054270	0.892177	0.067480	0.398292	1.000000

In [19]:
```
plt.figure(figsize=(10,7))
sns.heatmap(data.corr(),vmax=.8,linewidth=.01, square = True, annot = True,cmap='YlGnBu',linecolor ='black')
plt.show()
```



In [20]:
```
# We see bath and bhk are highly correlated, so I will drop one of them
# Here I am dropping bhk column

data.drop(columns=['bhk'],inplace=True)
```

Distinct Groups in Each Column

In [21]:
```
for i in data.columns:
    print(i,"->")
    print(data[i].value_counts())
    print()
    print("*"*40)
    print()
```

```
area_type ->
Super built-up  Area    8790
Built-up  Area          2418
Plot  Area              2009
Carpet  Area              87
Name: area_type, dtype: int64

****************************************

availability ->
Ready To Move    10581
18-Dec             307
18-May             295
18-Apr             271
18-Aug             200
                 ...
15-Aug               1
15-Jun               1
15-Dec               1
16-Nov               1
17-Jan               1
Name: availability, Length: 80, dtype: int64

****************************************

location ->
Whitefield             540
Sarjapur  Road         397
```

```
Sarjapur  Road        397
Electronic City       302
Kanakpura Road        273
Thanisandra           234
                      ...
Kamdhenu Nagar          1
Jagajyothi layout       1
1Channasandra           1
Chowdeshwari Layout     1
arudi                   1
Name: location, Length: 1304, dtype: int64

****************************************

total_sqft ->
1200.0    843
1100.0    221
1500.0    204
2400.0    195
600.0     180
         ...
2008.0      1
2015.0      1
2019.0      1
2023.0      1
1081.5      1
Name: total_sqft, Length: 2033, dtype: int64

****************************************

bath ->
2.0     6908
3.0     3286
4.0     1226
1.0      845
5.0      524
6.0      273
7.0      102
8.0       64
9.0       43
10.0      13
12.0       7
11.0       3
13.0       3
16.0       2
18.0       1
15.0       1
40.0       1
14.0       1
27.0       1
Name: bath, dtype: int64

****************************************

balcony ->
2.0    5113
1.0    4897
3.0    1672
0.0    1622
Name: balcony, dtype: int64

****************************************

price ->
75.000    310
65.000    302
55.000    275
60.000    270
45.000    240
         ...
56.900      1
64.150      1
77.930      1
45.980      1
34.555      1
Name: price, Length: 1985, dtype: int64

****************************************
```

Transforming location Column

In [22]: data.location.nunique() # 1304 unique values

Out[22]:1304

In [23]:
```
#location column is a categorical column , this needs to be converted to numerical column .
#When we convert to numerical column we will get too many columns using One hot encoder
#All those locations which have really less count, mark it to Others in order to limit the features
```

In [24]:
```
location_count= data['location'].value_counts()

location_count_less_20= location_count[location_count<=20]
location_count_less_20
```

Out[24]:
```
HBR Layout              20
Poorna Pragna Layout    20
Sanjay nagar            20
Yelachenahalli          20
HRBR Layout             19
                        ..
Kamdhenu Nagar           1
Jagajyothi layout        1
1Channasandra            1
Chowdeshwari Layout      1
arudi                    1
Name: location, Length: 1160, dtype: int64
```

In [25]:
```
# So if the area is having less than or equal to 20 houses I will mark them as "Others"

data['location']=data['location'].apply(lambda x:'Others' if x in location_count_less_20 else x)
```

In [26]: data.location.value_counts() # 145 unique values left out of 1304

Out[26]:
```
Others                4296
Whitefield             540
Sarjapur  Road         397
Electronic City        302
Kanakpura Road         273
                      ...
Domlur                  22
Ulsoor                  21
Hoskote                 21
Binny Pete              21
Basaveshwara Nagar      21
Name: location, Length: 145, dtype: int64
```

Dropping Unnecessary Columns

In [27]:
```
data.drop(columns=['availability','balcony'],inplace=True)
```

Numerical Columns

In [28]:
```
# Statistical information for numerical columns
data.describe()
```

Out[28]:

	total_sqft	bath	price
count	1.330400e+04	13304.000000	13304.000000
mean	1.911220e+03	2.685358	112.582035
std	1.728808e+04	1.343139	148.988398
min	1.000000e+00	1.000000	8.000000
25%	1.100000e+03	2.000000	50.000000
50%	1.276000e+03	2.000000	72.000000
75%	1.680000e+03	3.000000	120.000000
max	1.306800e+06	40.000000	3600.000000

Categorical Columns

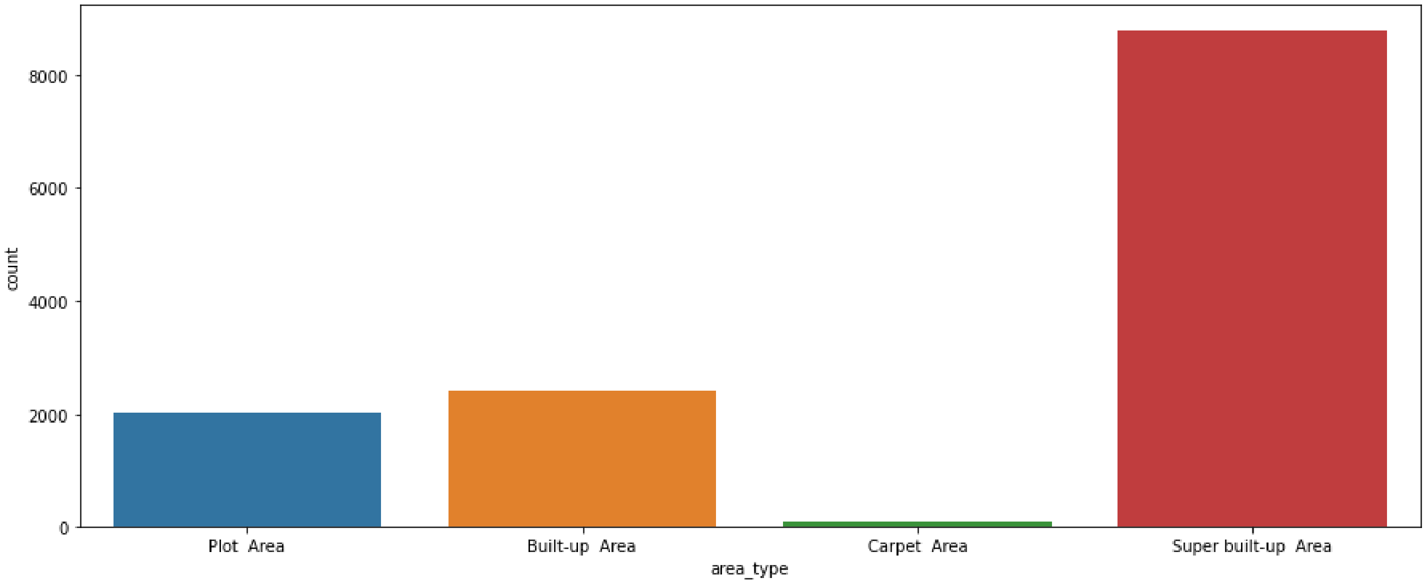
In [29]:
```
data.describe(include = 'object')
```

Out[29]:

	area_type	location
count	13304	13304
unique	4	145
top	Super built-up Area	Others
freq	8790	4296

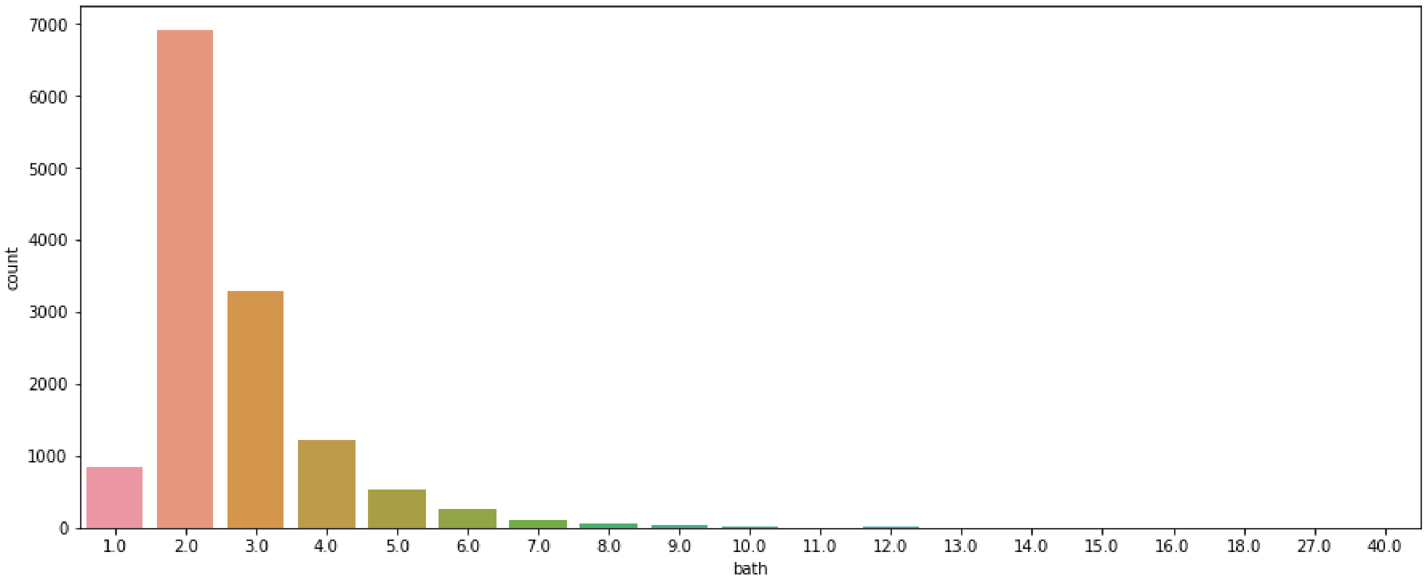
UNIVARIATE ANALYSIS

In [30]:
```
plt.figure(figsize=(15,6))
sns.countplot(data['area_type'])
plt.show()
```



Majority of the houses are in super built-up area and very few are in carpet area

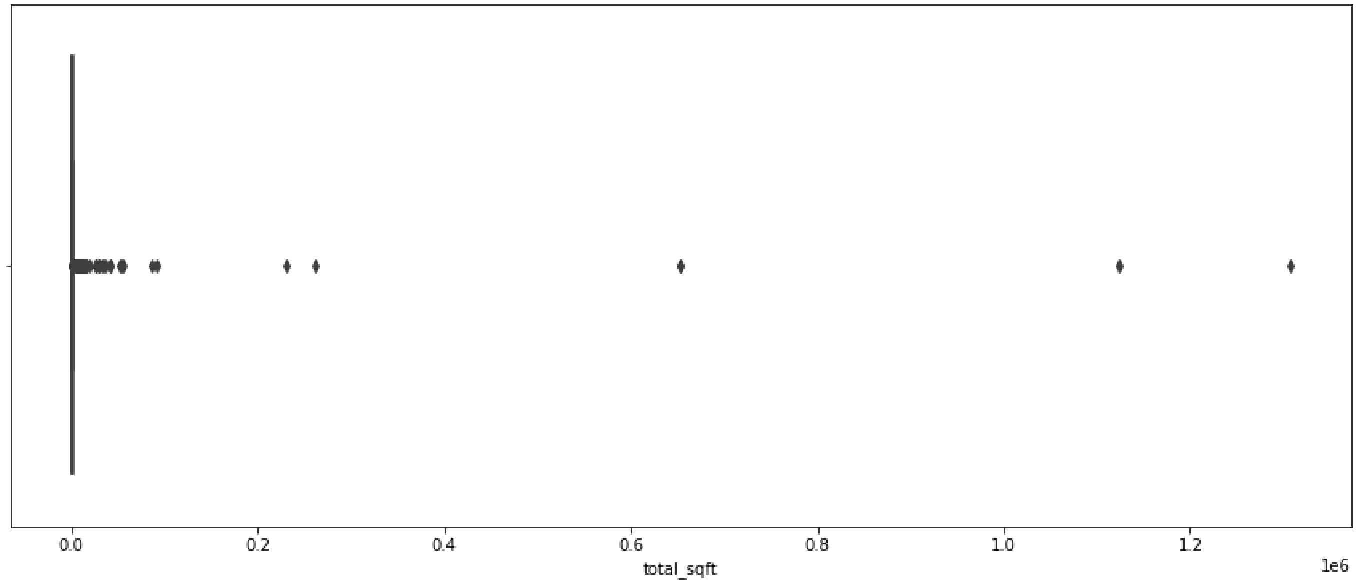
In [31]:
```
plt.figure(figsize=(15,6))
sns.countplot(data['bath'])
plt.show()
```



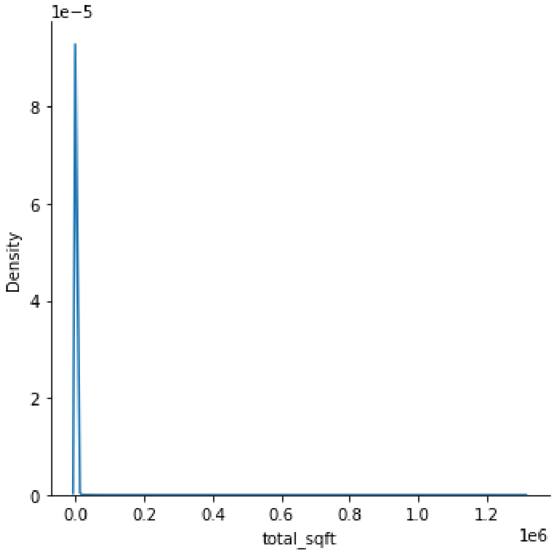
More than 50% houses have 2 baths and as the no. of baths keep on increasing the count of houses fall drastically.

OUTLIER ANALYSIS

In [32]:
```
# Outlier Analysis for 'total square feet' column

plt.figure(figsize=(15,6))
sns.boxplot(data['total_sqft'])  # A lot of outliers present
plt.show()
```

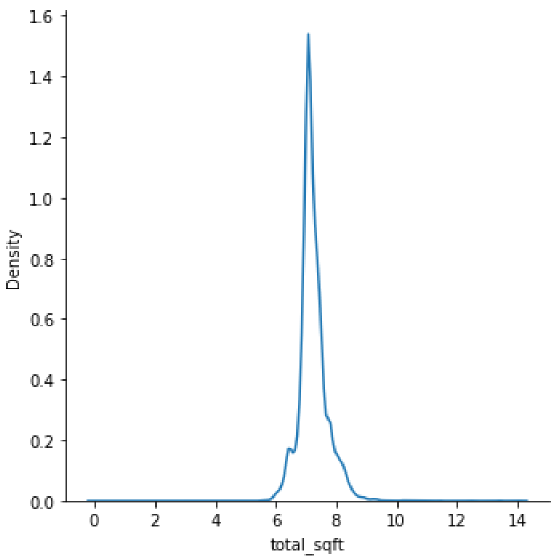


In [33]:
```
# Data is highly skewed
sns.displot(data['total_sqft'],kind='kde')
plt.show()
```



In [34]:
```
# To remove right skewness
data['total_sqft'] = np.log(data['total_sqft'])
```

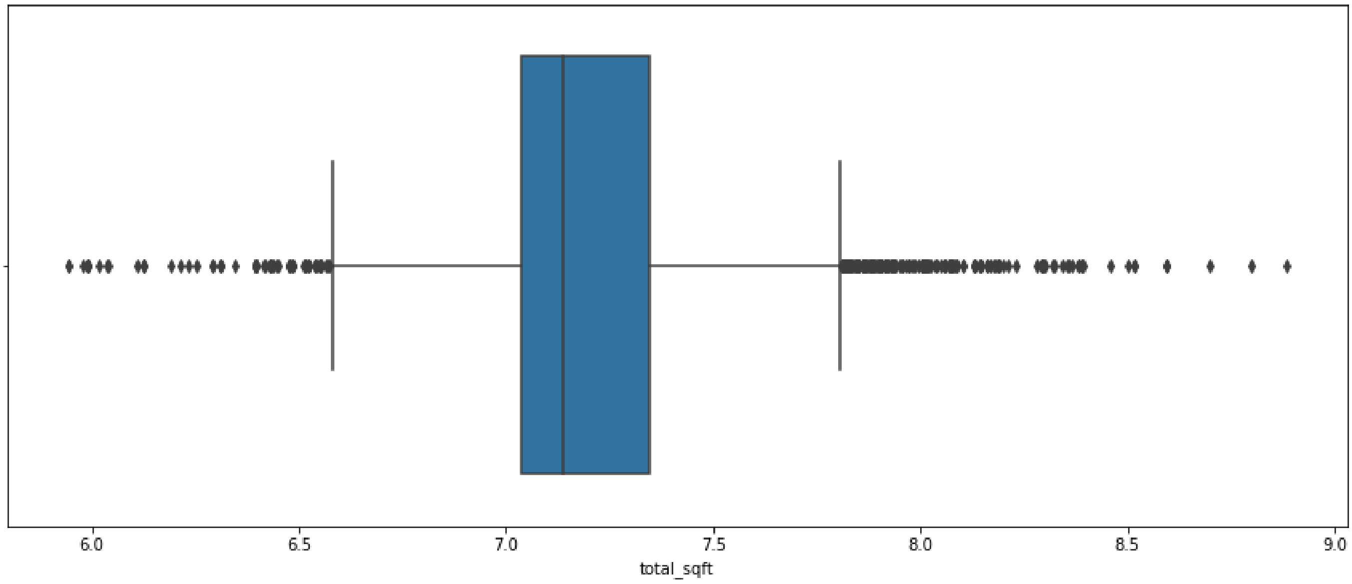
In [35]:
```
sns.displot(data['total_sqft'],kind='kde')
plt.show()
```



In [36]:
```
def remove_total_sqft_outliers(data):

    data_out = pd.DataFrame()
    for i,j in data.groupby(by=['location','area_type']):

        m = np.mean(j['total_sqft'])
        sd = np.std(j['total_sqft'])
        final_data = j[((j['total_sqft']) > (m-sd)) & ((j['total_sqft']) <= (m+sd))]
```

```python
        data_out = pd.concat([data_out, final_data], ignore_index=True)
    return data_out

data = remove_total_sqft_outliers(data)
```

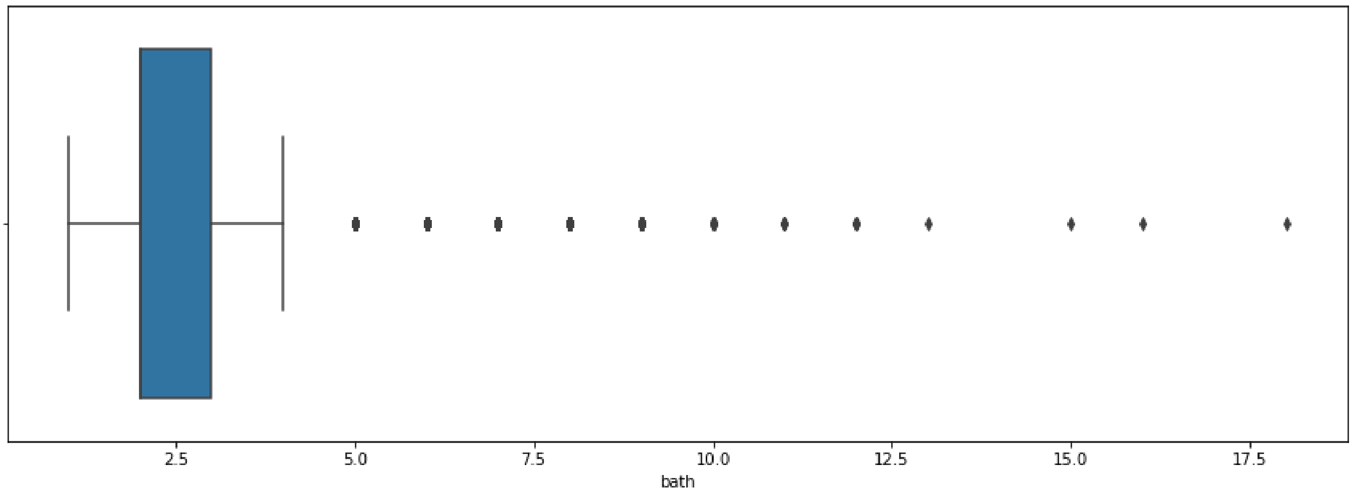
In [37]:
```python
# After removing outliers

plt.figure(figsize=(15,6))
sns.boxplot(data['total_sqft'])
plt.show()
```



In [38]:
```python
data.shape  # 9855 rows left
```

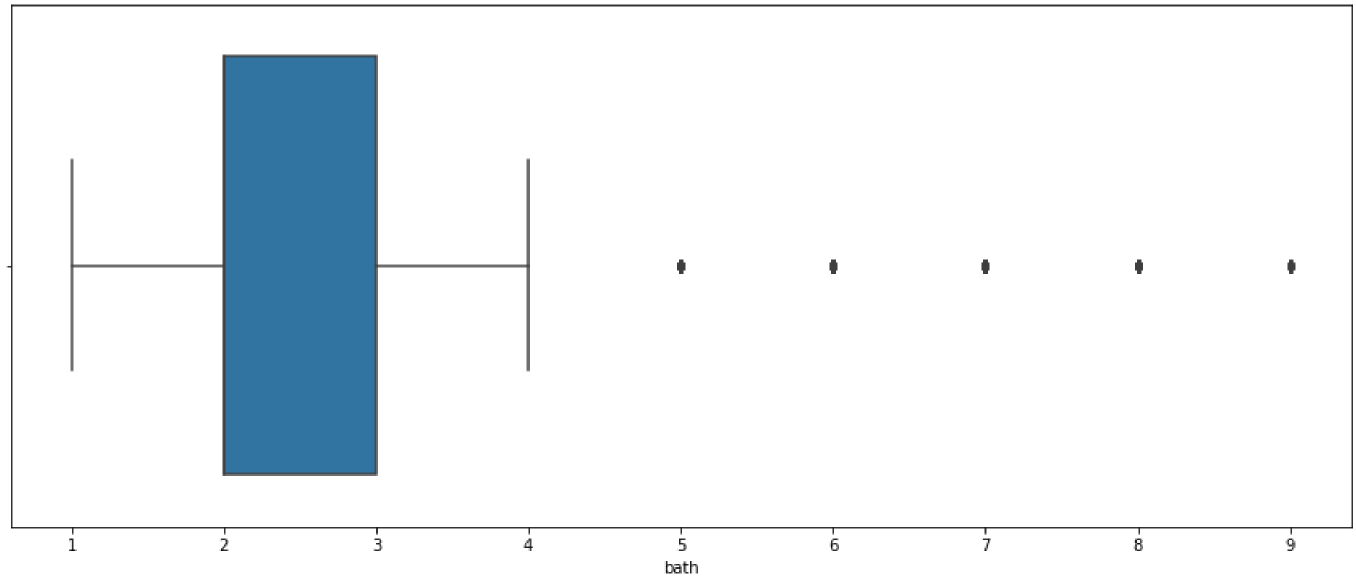
Out[38]:(9855, 5)

In [39]:
```python
# Outlier Analysis for 'bath' column

plt.figure(figsize=(15,5))
sns.boxplot(data['bath'])
plt.show()
```



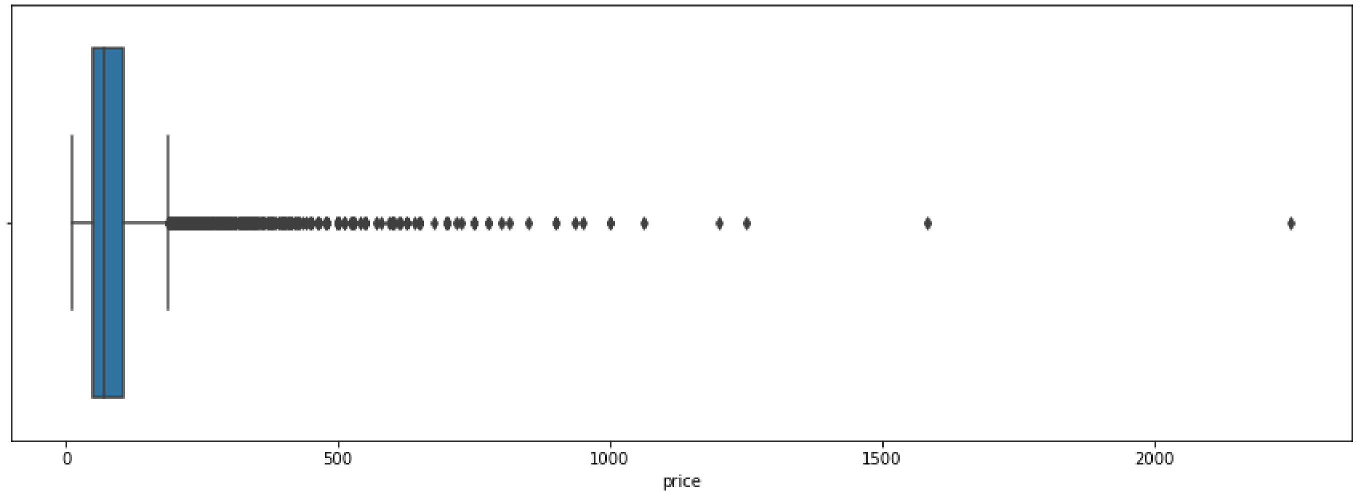
In [40]:
```python
# To remove outliers

data = data[(data['bath'] < 10)]
```

In [41]:
```python
# After removing outliers

plt.figure(figsize=(15,6))
sns.boxplot(data['bath'])
plt.show()
```

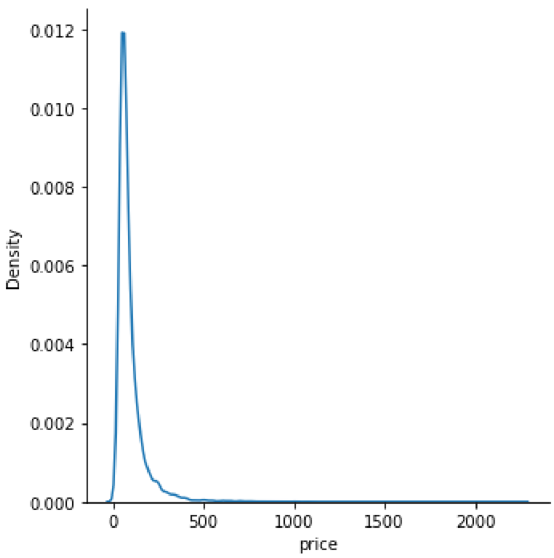


In [42]:
```
data.shape   # 9835 rows left
```
Out[42]:(9835, 5)

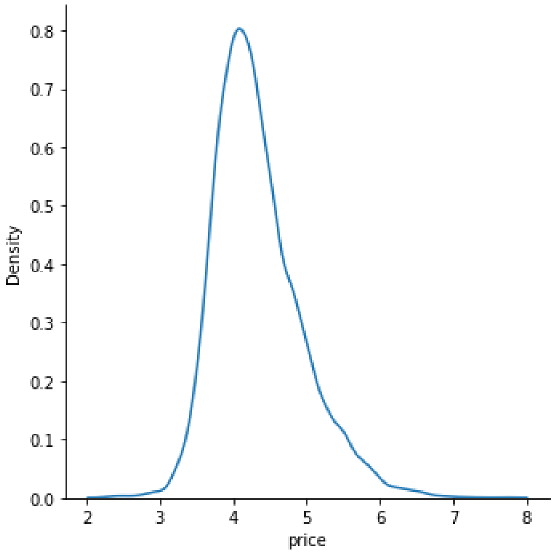
In [43]:
```
# Outlier Analysis for 'price' column

plt.figure(figsize=(15,5))   # Lots of outliers present
sns.boxplot(data['price'])
plt.show()
```



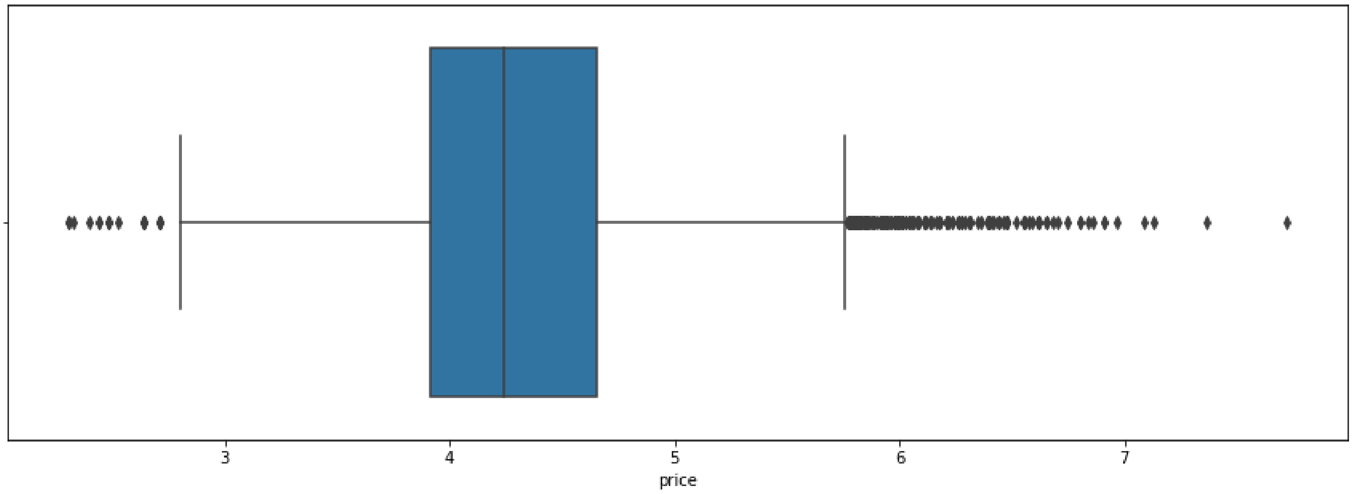
In [44]:
```
# Data is highly skewed
sns.displot(data['price'],kind='kde')
plt.show()
```



In [45]:
```
# To remove right skewness
data['price'] = np.log(data['price'])
```
In [46]:
```
sns.displot(data['price'],kind='kde')
plt.show()
```

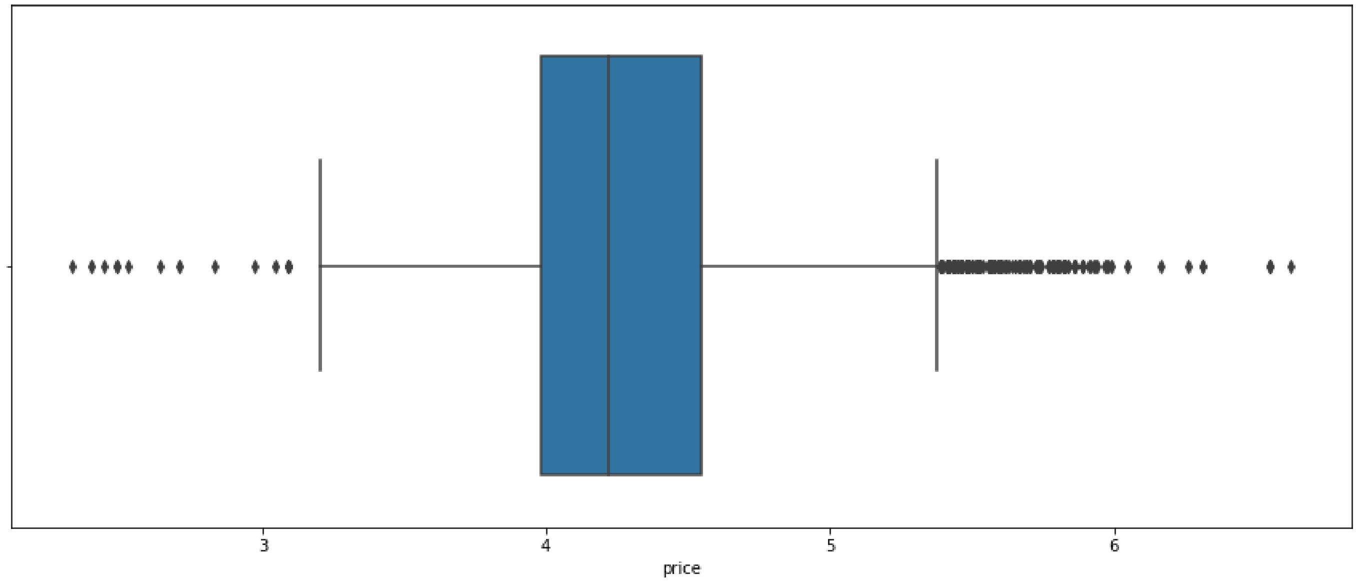


In [47]:
```
plt.figure(figsize=(15,5))   # Lots of outliers present
sns.boxplot(data['price'])
plt.show()
```



In [48]:
```
def remove_prices_outliers(data):

    data_out = pd.DataFrame()
    for i,j in data.groupby(by=['location','area_type']):

        m = np.mean(j['price'])
        sd = np.std(j['price'])
        final_data = j[((j['price']) > (m-sd)) & ((j['price']) <= (m+sd))]

        data_out = pd.concat([data_out, final_data], ignore_index=True)
    return data_out

data = remove_prices_outliers(data)
```

In [49]:
```
# After removing outliers

plt.figure(figsize=(15,6))
sns.boxplot(data['price'])
plt.show()
```

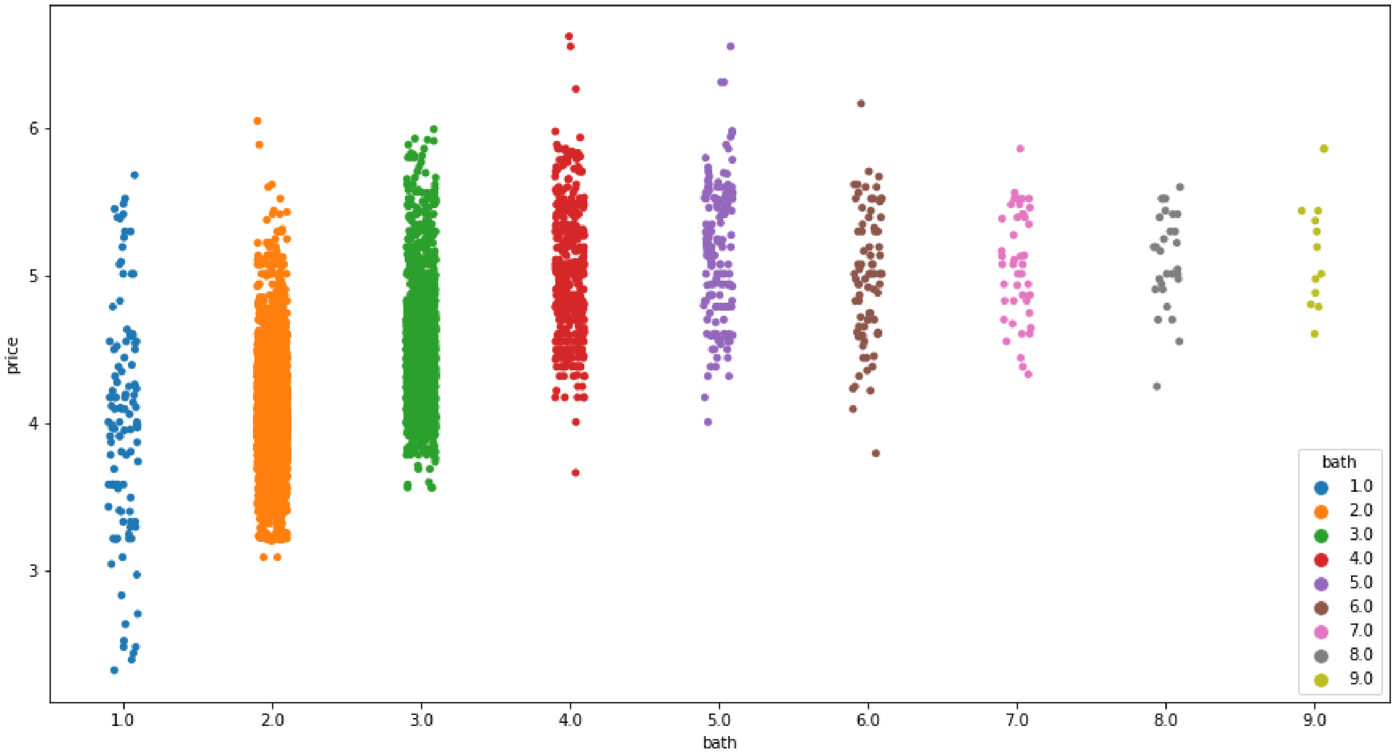


In [50]:
```
data.shape   # After removing outliers 6674 rows left out of 13,320
```

Out[50]:(6674, 5)

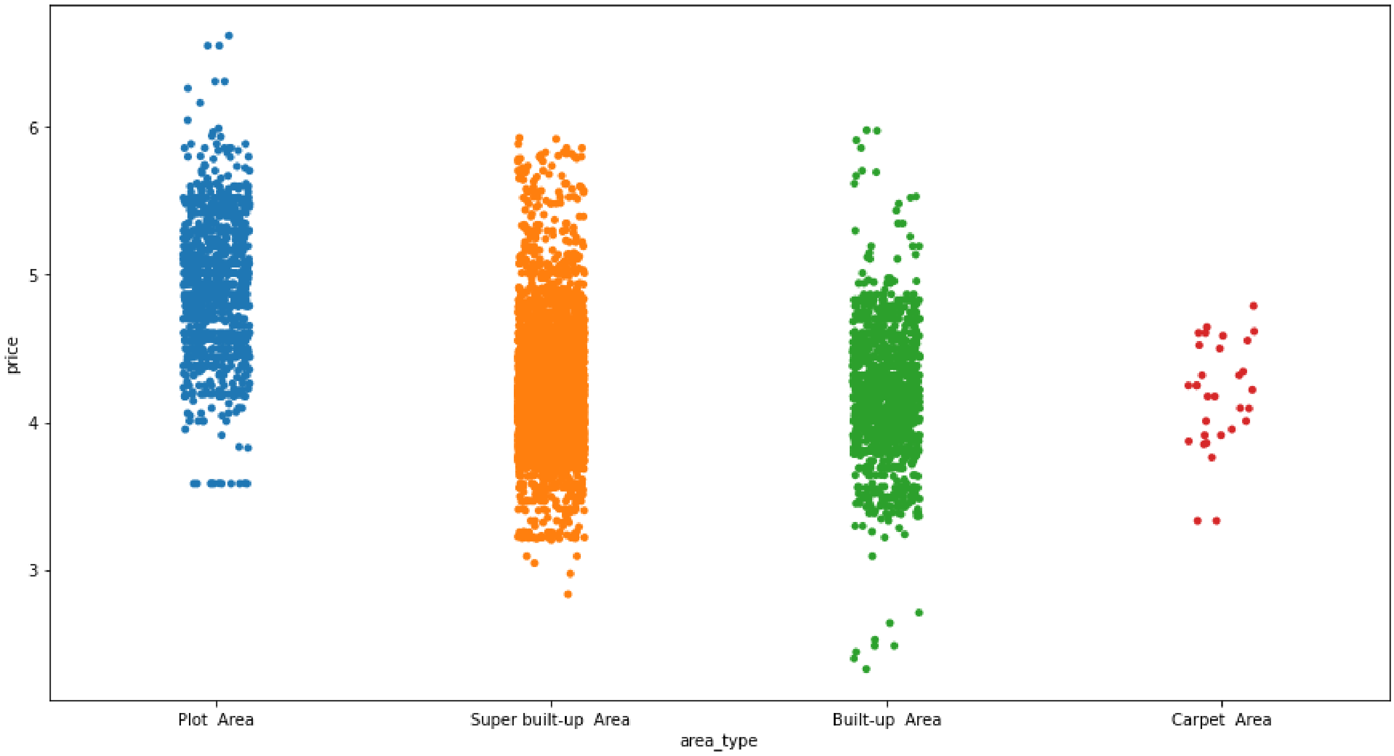
BIVARIATE ANALYSIS

In [51]:
```
plt.figure(figsize=(15,8))
sns.stripplot(x=data['bath'],y=data['price'],hue=data['bath'])
plt.show()
```



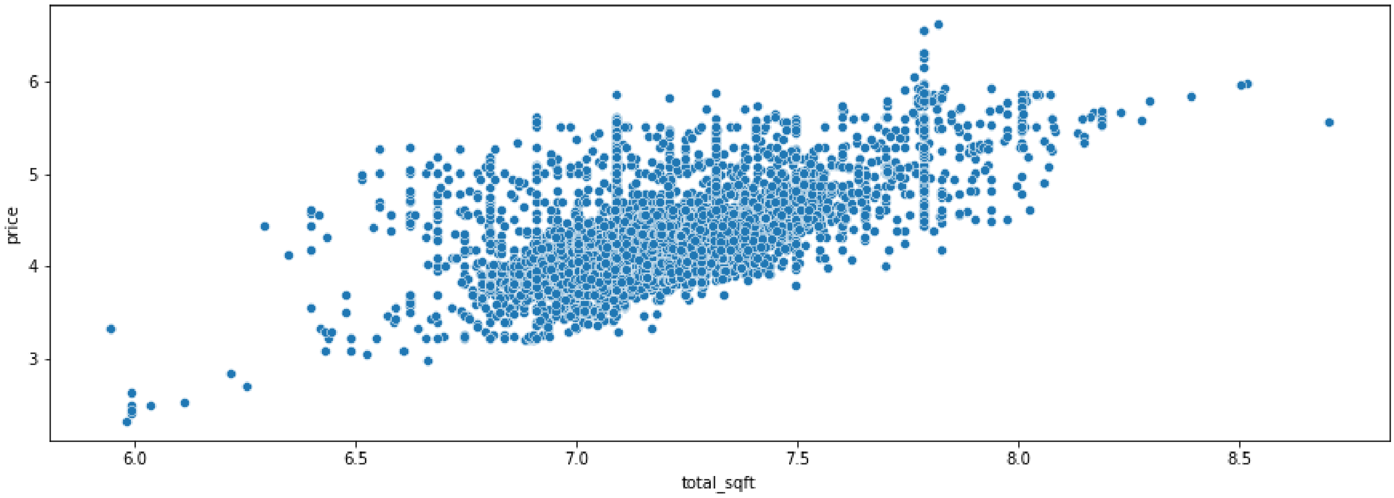
As the number of bathroom increases, price range goes up.

In [52]:
```
plt.figure(figsize=(15,8))
sns.stripplot(data=data,y='price',x='area_type')
plt.show()
```



Price range is lower for built up areas and highest for plot areas.

In [53]:
```
plt.figure(figsize=(15,5))
sns.scatterplot(data=data,y='price',x='total_sqft')
plt.show()
```



With increase in total square feet area, price is also increasing.

Clean Data

In [54]:
```
data
```

Out[54]:

	area_type	location	total_sqft	bath	price
0	Plot Area	1st Phase JP Nagar	7.090077	7.0	5.480639
1	Super built-up Area	1st Phase JP Nagar	7.239933	2.0	4.605170
2	Super built-up Area	1st Phase JP Nagar	7.371489	3.0	4.875197
3	Super built-up Area	1st Phase JP Nagar	7.536364	3.0	5.117994
4	Super built-up Area	1st Phase JP Nagar	7.612831	3.0	5.056246
...	...	...	...	...	...
6669	Super built-up Area	Yeshwanthpur	7.434257	3.0	4.682131
6670	Super built-up Area	Yeshwanthpur	7.446001	3.0	4.700480
6671	Super built-up Area	Yeshwanthpur	7.522941	3.0	4.605170
6672	Super built-up Area	Yeshwanthpur	7.575585	4.0	4.867534
6673	Super built-up Area	Yeshwanthpur	7.152660	2.0	4.562419

6674 rows × 5 columns

In [55]:
```
data.shape
```
Out[55]:(6674, 5)

In [56]:
```
data.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6674 entries, 0 to 6673
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   area_type   6674 non-null   object
 1   location    6674 non-null   object
 2   total_sqft  6674 non-null   float64
 3   bath        6674 non-null   float64
 4   price       6674 non-null   float64
dtypes: float64(3), object(2)
memory usage: 260.8+ KB
```
In [57]:
```
data.describe(include='all')
```
Out[57]:

	area_type	location	total_sqft	bath	price
count	6674	6674	6674.000000	6674.000000	6674.000000
unique	4	145	NaN	NaN	NaN
top	Super built-up Area	Others	NaN	NaN	NaN
freq	4527	2386	NaN	NaN	NaN
mean	NaN	NaN	7.186386	2.536410	4.303345
std	NaN	NaN	0.244725	1.016104	0.485201
min	NaN	NaN	5.942799	1.000000	2.327278
25%	NaN	NaN	7.047517	2.000000	3.984530
50%	NaN	NaN	7.138867	2.000000	4.219508
75%	NaN	NaN	7.313220	3.000000	4.543295
max	NaN	NaN	8.699515	9.000000	6.620073

Linear Regression

In [58]:
```
x=data.drop(columns=['price'])
```
In [59]:
```
y=data['price']
```
In [60]:
```
# Independent Variables (Features)
x
```

Out[60]:

	area_type	location	total_sqft	bath
0	Plot Area	1st Phase JP Nagar	7.090077	7.0
1	Super built-up Area	1st Phase JP Nagar	7.239933	2.0
2	Super built-up Area	1st Phase JP Nagar	7.371489	3.0
3	Super built-up Area	1st Phase JP Nagar	7.536364	3.0
4	Super built-up Area	1st Phase JP Nagar	7.612831	3.0
...	...	...	...	...
6669	Super built-up Area	Yeshwanthpur	7.434257	3.0
6670	Super built-up Area	Yeshwanthpur	7.446001	3.0
6671	Super built-up Area	Yeshwanthpur	7.522941	3.0
6672	Super built-up Area	Yeshwanthpur	7.575585	4.0
6673	Super built-up Area	Yeshwanthpur	7.152660	2.0

6674 rows × 4 columns

In [61]:
```
# Dependent Variable (Target Column)
y
```

Out[61]:
```
0       5.480639
1       4.605170
2       4.875197
3       5.117994
4       5.056246
          ...
6669    4.682131
6670    4.700480
6671    4.605170
6672    4.867534
6673    4.562419
Name: price, Length: 6674, dtype: float64
```

Importing Required Libraries for Regression

In [62]:
```
from sklearn.model_selection import  train_test_split

from sklearn.compose import make_column_transformer

from sklearn.preprocessing import MinMaxScaler,StandardScaler,OneHotEncoder

from sklearn.linear_model import LinearRegression,Lasso,Ridge
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score
```

In [63]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y, test_size=0.3,random_state=22)
```

In [64]:
```
print(x_train.shape)
print(x_test.shape)
```

```
(4671, 4)
(2003, 4)
```

In [65]:
```
# one hot encoding
col_trans = make_column_transformer((OneHotEncoder(sparse=False), ['location','area_type']),
                                    remainder='passthrough')
```

In [66]:
```
scaled = StandardScaler()
```

In [67]:
```
lr=LinearRegression(normalize=True)
```

In [68]:
```
pipe = make_pipeline(col_trans,scaled,lr)
```

In [69]:
```
pipe.fit(x_train,y_train)
```

Out[69]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
                                                  OneHotEncoder(sparse=False),
                                                  ['location', 'area_type'])])),
                ('standardscaler', StandardScaler()),
                ('linearregression', LinearRegression(normalize=True))])
```

In [70]:
```
y_predicted = pipe.predict(x_test)
```

In [71]:
```
# R2 value

r2_score (y_test,y_predicted)
```

Out[71]:
```
0.8101442283301818
```

In [72]:
```
# Adjusted R-squared
```

```
1 - (1-pipe.score(x, y))*(len(y)-1)/(len(y)-x.shape[1]-1)
```

Out[72]:0.8074694874094117

Evaluation Matrices for the Model

In [73]:

```
from sklearn.metrics import mean_squared_error as MSE

accuracy0 = pipe.score(x_test,y_test)
print("Testing Accuracy:",accuracy0)
print()

print("Training Accuracy:",pipe.score(x_train,y_train))
print()

MSE_score = MSE(y_test,y_predicted)
print("Mean Squared Error:",MSE_score.mean())
print()

import math
print("Root Mean Squared Error:",math.sqrt(MSE_score.mean()))
print()

from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error:",mean_absolute_error(y_test, y_predicted))
print()
```

Testing Accuracy: 0.8101442283301818

Training Accuracy: 0.8064618757621531

Mean Squared Error: 0.04523465350207817

Root Mean Squared Error: 0.21268439882153597

Mean Absolute Error: 0.1630027651419351

Comparison of Models

Lasso Regression

In [74]:

```
from sklearn.linear_model import Lasso
```

In [75]:

```
lassoreg=Lasso(alpha=0.001,normalize=True)


pipe1 = make_pipeline(col_trans,scaled,lassoreg)
pipe1.fit(x_train,y_train)

y_predicted = pipe1.predict(x_test)
print("R2 value :",r2_score (y_test,y_predicted))
print()
print("Adjusted R2:",1 - (1-pipe1.score(x, y))*(len(y)-1)/(len(y)-x.shape[1]-1))
print()

accuracy1 = pipe1.score(x_test,y_test)
MSE_score1 = MSE(y_test,y_predicted)

print("Training Accuracy :",pipe1.score(x_train,y_train))
print()
print("Testing Accuracy :",accuracy1)
print()
print("Mean Squared Error :",MSE_score1.mean())
print()

print("Root Mean Squared Error :",math.sqrt(MSE_score1.mean()))
print()

print("Mean Absolute Error :",mean_absolute_error(y_test, y_predicted))
```

R2 value : 0.5910492191582313

Adjusted R2: 0.5821138377685939

Training Accuracy : 0.5785608621419338

Testing Accuracy : 0.5910492191582313

Mean Squared Error : 0.09743578879947482

Root Mean Squared Error : 0.3121470627756648

Mean Absolute Error : 0.24301491730908661

Ridge Regression

In [76]:
```
from sklearn.linear_model import Ridge

ridgereg=Ridge(alpha=0.001,normalize=True)

pipe2 = make_pipeline(col_trans,scaled,ridgereg)
pipe2.fit(x_train,y_train)
y_predicted = pipe2.predict(x_test)
print("R2 value :",r2_score (y_test,y_predicted))
print()
print("Adjusted R2 :",1 - (1-pipe2.score(x, y))*(len(y)-1)/(len(y)-x.shape[1]-1))
print()

accuracy1 = pipe2.score(x_test,y_test)
MSE_score1 = MSE(y_test,y_predicted)

print("Training Accuracy :",pipe2.score(x_train,y_train))
print()
print("Testing Accuracy :",accuracy1)
print()
print("Mean Squared Error :",MSE_score1.mean())
print()

print("Root Mean Squared Error :",math.sqrt(MSE_score1.mean()))
print()


print("Mean Absolute Error :",mean_absolute_error(y_test, y_predicted))
```

R2 value : 0.8108034737272101

Adjusted R2 : 0.807941944361105

Training Accuracy : 0.80685232558839

Testing Accuracy : 0.8108034737272101

Mean Squared Error : 0.04507758302249711

Root Mean Squared Error : 0.21231482054368486

Mean Absolute Error : 0.16278511440575158