In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import re
```

Business Promblem :

- Find the price of house based on the given features

Data Mining or Data Understanding :

In [2]:
```
df = pd.read_csv("Bengaluru_House_Data.csv")
df.head(3)
```

Out[2]:

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.07
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.00
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	62.00

observations:

- They are Null Values in the dataset
- They are 9 Features in the DataSet and 3 are Float and 6 are Object [Categorial]

In [3]:
```
df.isnull().sum()
```

Out[3]:
```
area_type          0
availability       0
location           1
size              16
society         5502
total_sqft         0
bath              73
balcony          609
price              0
dtype: int64
```

Note

- To Fill the Null Values Let's Check for Outliers Wheter present or not

Data PreProcessing :

To Find Outliers Extenstions

In [4]:
```
1 df.dtypes
```

Out[4]:
```
area_type        object
availability     object
location         object
size             object
society          object
total_sqft       object
bath            float64
balcony         float64
price           float64
dtype: object
```

In [5]:
```
1 df["total_sqft"].unique()
```

Out[5]:
```
array(['1056', '2600', '1440', ..., '1133 - 1384', '774', '4689'],
      dtype=object)
```

In [6]:
```
1 def cleaning(sento):
2     try:
3         u = int(sento)
4         return u
5     except:
6         u = sento.split(" ")
7         u = u[0]
8         u = re.sub('[^0-9]',"",u)
9         return u
```

In [7]:
```
1 df["total_sqft"]=df["total_sqft"].apply(lambda x:cleaning(x))
```

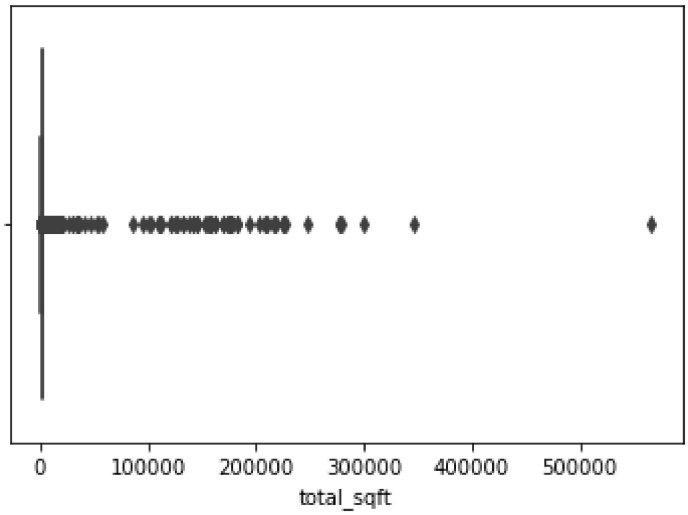
In [8]:
```
1 df.dtypes
```

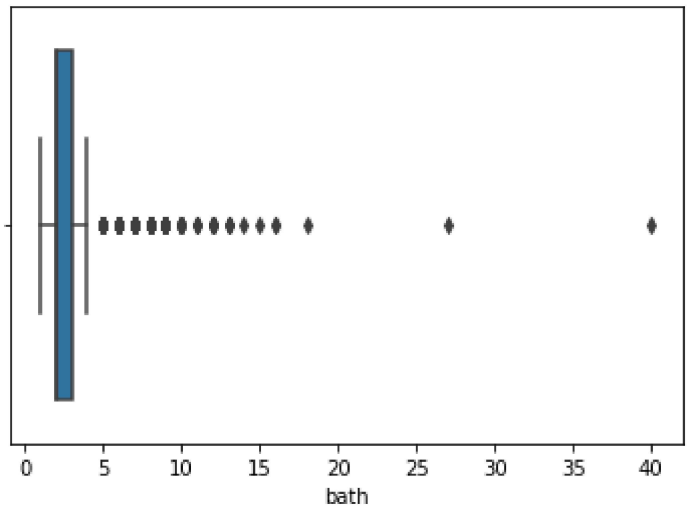
Out[8]:
```
area_type        object
availability     object
location         object
size             object
society          object
total_sqft       object
bath            float64
balcony         float64
price           float64
dtype: object
```

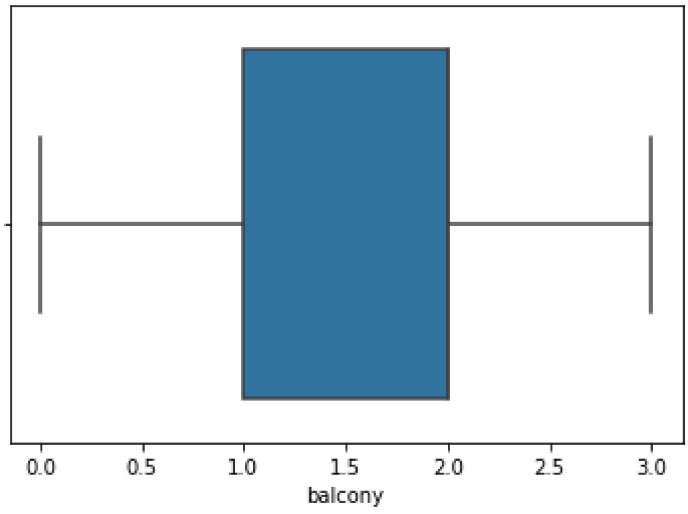
In [9]:
```
1 df["total_sqft"]=df["total_sqft"].astype("float64")
```

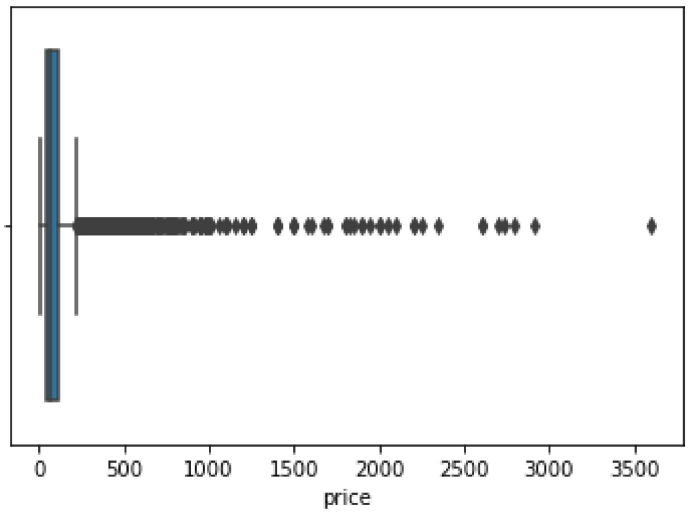
In [10]:
```
nume=[]
cate=[]
for i in df.columns:
    try:
        sns.boxplot(df[i])
        plt.show()
        nume.append(i)
    except:
        print("This Categorial Data :{}".format(i))
        cate.append(i)
```

```
This Categorial Data :area_type
This Categorial Data :availability
This Categorial Data :location
This Categorial Data :size
This Categorial Data :society
```

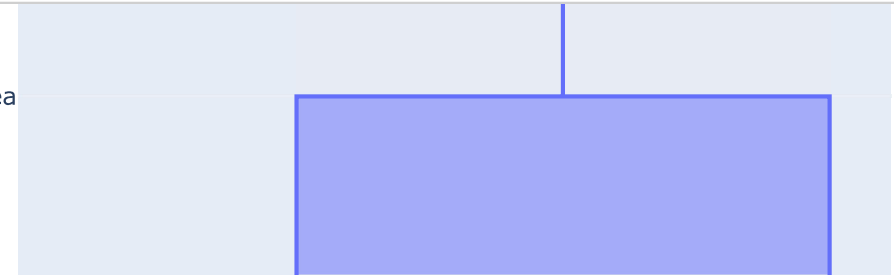





0.0
0.5
1.0
1.5
2.0
2.5
3.0
balcony


0
500
1000
1500
2000
2500
3000
3500
price

In [11]:
```
for i in df.columns:
    try:
        fig=px.box(df[i])
        fig.show()
    except:
        print("This Categorial Data :{}".format(i))

```



Note :

- There is Outliers in Bath Fill Bath Value with Median and Remaing Catgorial Fill with Mode

In [12]:
```
from sklearn.impute import SimpleImputer
```

In [13]:
```
Meen = SimpleImputer(missing_values=np.nan,strategy='mean')
```

In [14]:
```
Moode =SimpleImputer(missing_values=np.nan,strategy='most_frequent')
```

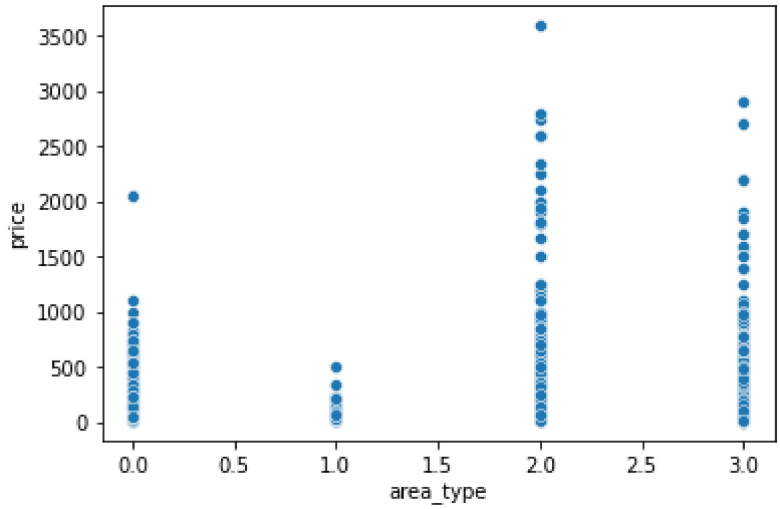
In [15]:
```
for i in df.columns:
    if type(df[i][0])== str:
        if df[i].isnull().sum()==0:
            pass
        else:
            df[i]=Moode.fit_transform(df[[i]])

    else:
        df[i]=Meen.fit_transform(df[[i]])
```

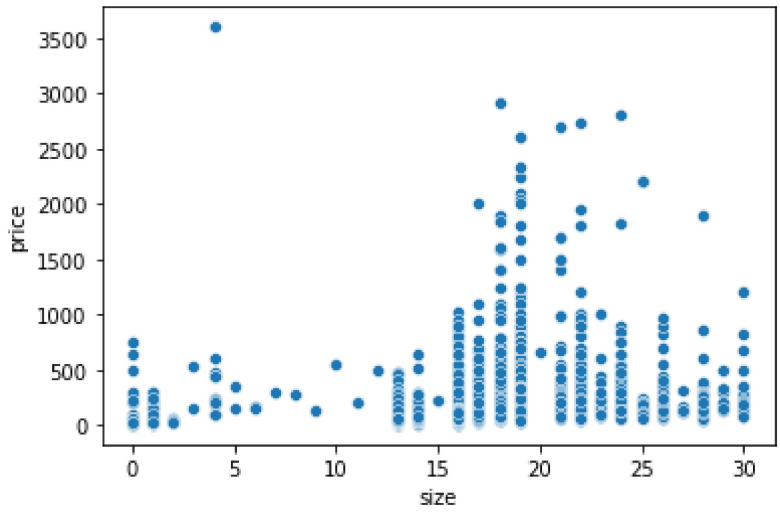
In [16]:
```
1 df.isnull().sum()
```

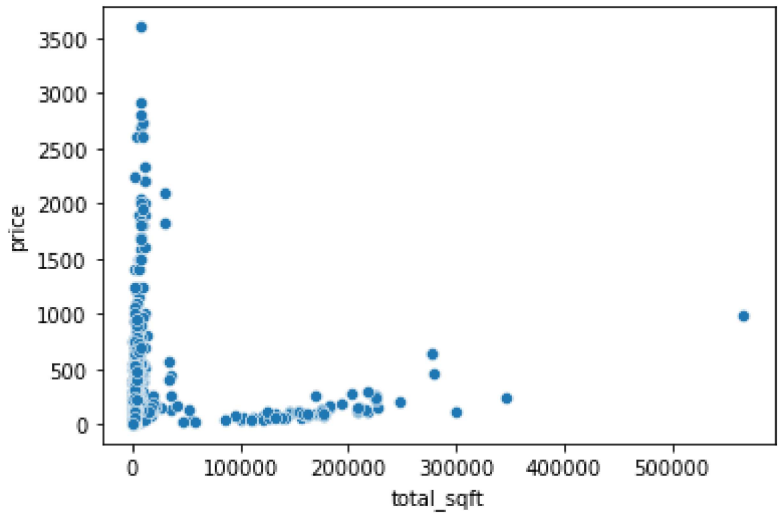
Out[16]:
```
area_type       0
availability    0
location        0
size            0
society         0
total_sqft      0
bath            0
balcony         0
price           0
dtype: int64
```

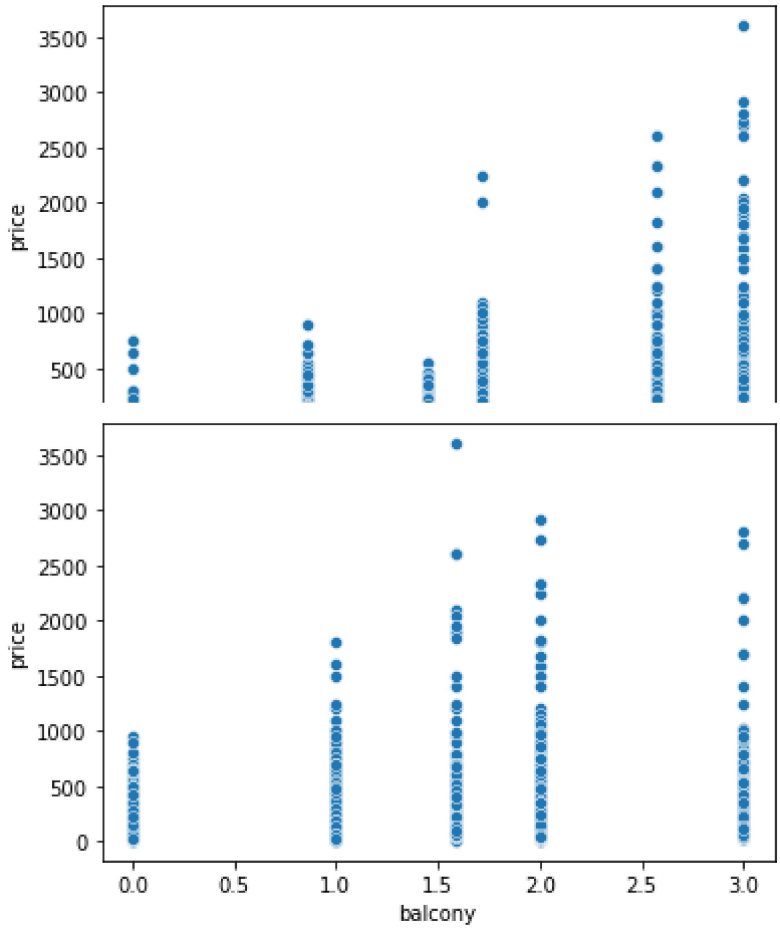
In [57]:
```
1 l=['area_type', 'size', 'total_sqft', 'bath', 'balcony']
```

In [59]:
```
for i in l:
    sns.scatterplot(df[i],df["price"])
    plt.show()
```









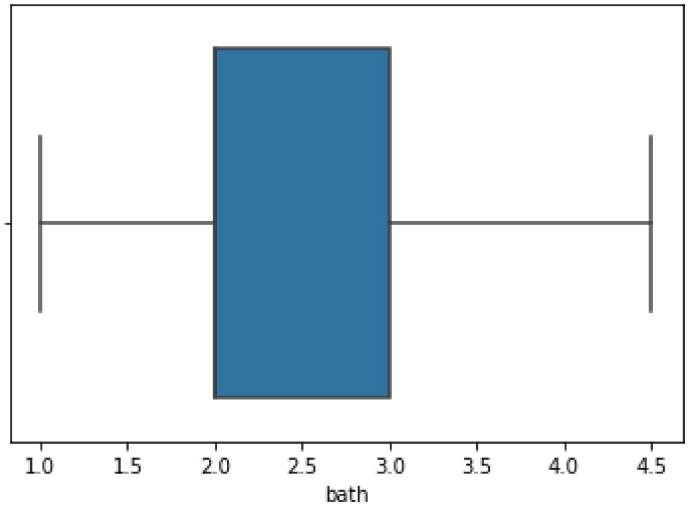
Treating Outlies

In [17]:
```
1 from feature_engine.outliers import Winsorizer
```

In [18]:
```
1 win = Winsorizer(capping_method = 'iqr',tail="both",fold= 1.5,variables=["ba
```

In [19]:
```
1 df["bath"] = win.fit_transform(df[["bath"]])
```

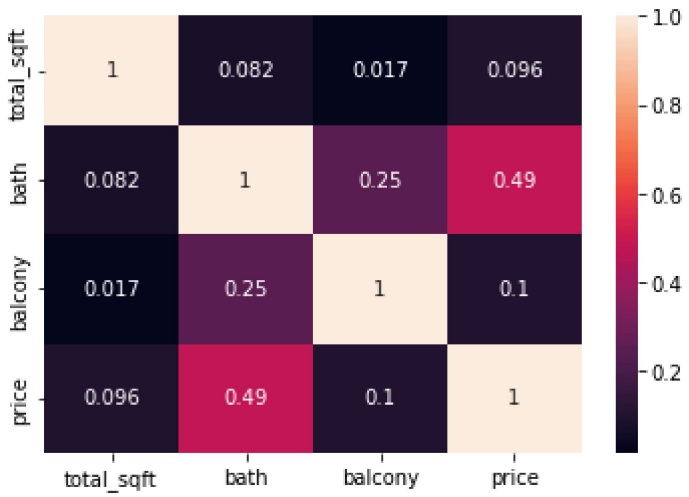
In [20]:
```
1 sns.boxplot(df["bath"])
2 plt.show()
```



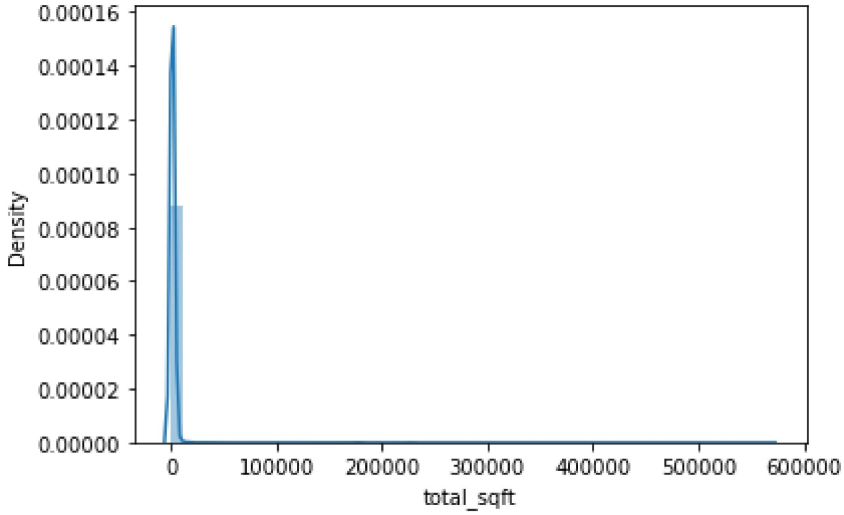
Note :

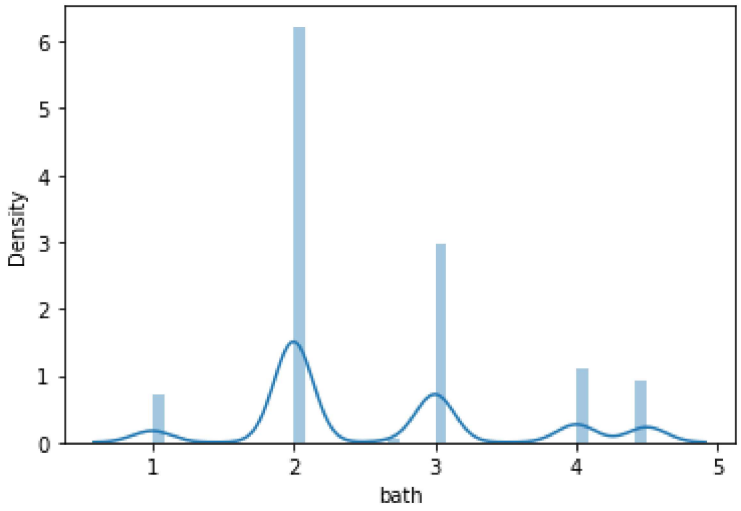
- Outliers removed let's check for Skewness to remove skewness check for co-relation

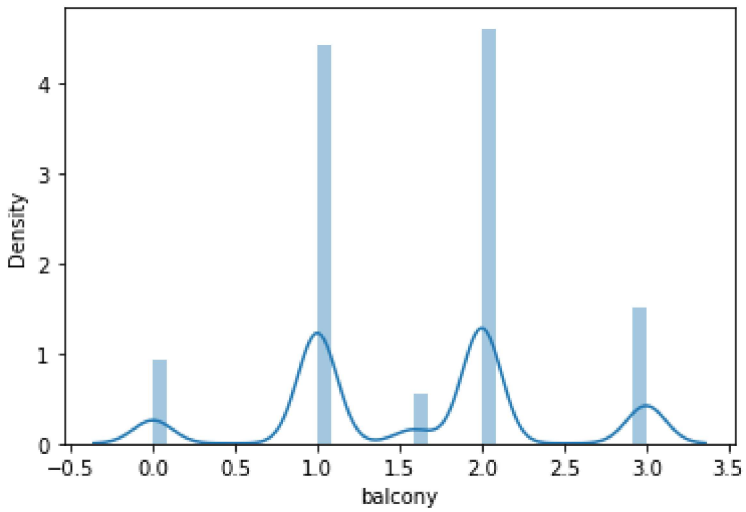
In [21]:
```
sns.heatmap(df.corr(),annot=True)
plt.show()
```

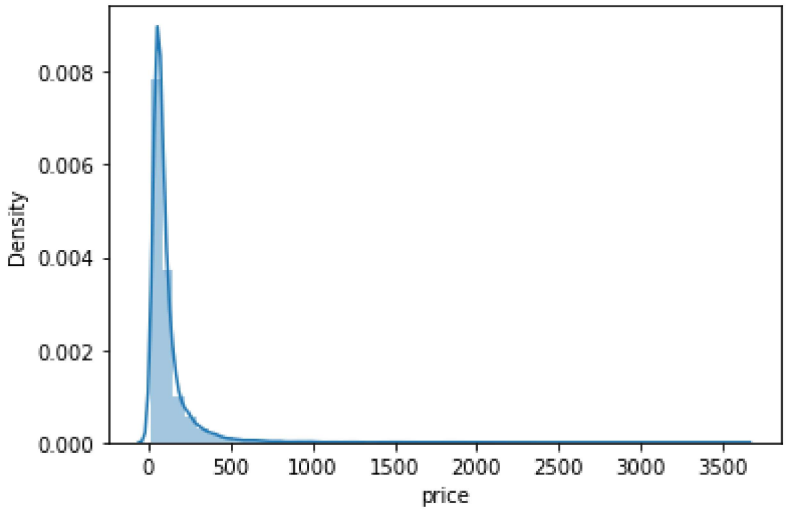


In [22]:
```
for i in df.columns:
    try:
        sns.distplot(df[i])
        plt.show()
    except:
        pass

```









In [23]:
```
cate
```

Out[23]:
```
['area_type', 'availability', 'location', 'size', 'society']
```

In [24]:
```
df.head(3)
```

Out[24]:

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056.0	2.0	1.0	39.07
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600.0	4.5	3.0	120.00
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	GrrvaGr	1440.0	2.0	3.0	62.00

In [25]:
```
for i in cate:
    x = df[i].nunique()
    print("{} The Unique values in {}".format(x,i))
```

```
4 The Unique values in area_type
81 The Unique values in availability
1305 The Unique values in location
31 The Unique values in size
2688 The Unique values in society
```

Encoding

In [26]:
```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
```

In [27]:
```
Le = LabelEncoder()
Oe = OneHotEncoder(sparse=False,handle_unknown='error')
```

In [28]:
```
df["area_type"] = Le.fit_transform(df[["area_type"]])
```

In [29]:
```
df["size"]=Le.fit_transform(df[["size"]])
```

In [30]:
```
df.head(3)
```

Out[30]:

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	3	19-Dec	Electronic City Phase II	13	Coomee	1056.0	2.0	1.0	39.07
1	2	Ready To Move	Chikka Tirupathi	19	Theanmp	2600.0	4.5	3.0	120.00
2	0	Ready To Move	Uttarahalli	16	GrrvaGr	1440.0	2.0	3.0	62.00

In [31]:
```
df.drop(columns=["availability","location","society"],inplace=True)
```

In [32]:
```
x = df.drop("price",axis=1)
```

In [ ]:
```

```

In [33]:
```
y = df["price"]
```

In [34]:
```
from sklearn.model_selection import train_test_split
```

In [35]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_st
```

In [36]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [37]:
```
ModelRRR = RandomForestRegressor()
```

In [38]:
```
ModelRRR.fit(x_train,y_train)
```

Out[38]:
```
RandomForestRegressor
RandomForestRegressor()
```

In [39]:
```
ModelRRR.score(x_train,y_train)
```

Out[39]:
```
0.8917220139606602
```

In [40]:
```
ModelRRR.score(x_test,y_test)
```

Out[40]:
```
0.5794796295643545
```

Note:

- Model is Overfitted

In [41]:
```
y_pred=ModelRRR.predict(x_test)
```

In [42]:
```
y_pred
```

Out[42]:
```
array([ 43.60001675, 310.96      ,  69.20772871, ...,  78.84898885,
        75.80111139, 402.06666667])
```

Evalution

In [43]:
```
from sklearn.metrics import mean_squared_error,r2_score
from math import sqrt
```

In [44]:
```
mse = mean_squared_error(y_test,y_pred)
mse
```

Out[44]:
```
10248.067733049638
```

In [45]:
```
rmse = sqrt(mean_squared_error(y_test, y_pred))
rmse
```

Out[45]:
```
101.2327404205262
```

In [46]:
```
r2_score(y_test, y_pred)
```

Out[46]:
```
0.5794796295643545
```

Multi-Linear_Regression

In [49]:
```
from sklearn.linear_model import LinearRegression
```

In [50]:
```
le =LinearRegression()
```

In [52]:
```
le.fit(x_train,y_train)
```

Out[52]:
```
LinearRegression
LinearRegression()
```

In [53]:
```
le.score(x_train,y_train)
```

Out[53]:
```
0.2469168113767335
```

In [54]:
```
le.score(x_test,y_test)
```

Out[54]:
```
0.23597915494676758
```

Result : Worst Model[Under Fitting]

Regularization

In [60]:
```
1 from sklearn import datasets
```

In [61]:
```
1 bos = datasets.load_boston()
```

In [62]:
```
1 bos
```

Out[62]:
```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e
+02,
        4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
        9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
        4.0300e+00],
       ...,
       [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        5.6400e+00],
       [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
        6.4800e+00],
       [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        7.8800e+00]]),
 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9,
15. ,
       18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
       15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
       13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
       21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
       35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
       19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
       20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
       23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
       33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
       21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
       20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
       23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
       15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
       17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
       25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
       23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
       32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
       34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
       20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
       26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
       31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
       22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
       42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
       36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
       32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
       20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
       20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
       22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
       21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
       19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
       32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
       18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
       16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
       13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
        7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
       12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
       27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
        8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
```

```
         9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
        10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
        15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
        19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
        29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
        20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
        23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9]),
 'feature_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'),
 'DESCR': ".. _boston_dataset:\n\nBoston house prices dataset\n---------------
------------\n\n**Data Set Characteristics:**  \n\n    :Number of Instances:
506 \n\n    :Number of Attributes: 13 numeric/categorical predictive. Median
Value (attribute 14) is usually the target.\n\n    :Attribute Information (in
order):\n        - CRIM     per capita crime rate by town\n        - ZN
proportion of residential land zoned for lots over 25,000 sq.ft.\n        - I
NDUS    proportion of non-retail business acres per town\n        - CHAS
Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)\n
- NOX      nitric oxides concentration (parts per 10 million)\n        - RM
average number of rooms per dwelling\n        - AGE      proportion of owner-
occupied units built prior to 1940\n        - DIS      weighted distances to
five Boston employment centres\n        - RAD      index of accessibility to
radial highways\n        - TAX      full-value property-tax rate per $10,000
\n        - PTRATIO  pupil-teacher ratio by town\n        - B        1000(Bk
- 0.63)^2 where Bk is the proportion of black people by town\n        - LSTAT
% lower status of the population\n        - MEDV     Median value of owner-oc
cupied homes in $1000's\n\n    :Missing Attribute Values: None\n\n    :Creato
r: Harrison, D. and Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing datas
et.\nhttps://archive.ics.uci.edu/ml/machine-learning-databases/housing/\n\n\n
This dataset was taken from the StatLib library which is maintained at Carneg
ie Mellon University.\n\nThe Boston house-price data of Harrison, D. and Rubi
nfeld, D.L. 'Hedonic\nprices and the demand for clean air', J. Environ. Econo
mics & Management,\nvol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Re
gression diagnostics\n...', Wiley, 1980.   N.B. Various transformations are u
sed in the table on\npages 244-261 of the latter.\n\nThe Boston house-price d
ata has been used in many machine learning papers that address regression\npr
oblems.   \n     \n.. topic:: References\n\n   - Belsley, Kuh & Welsch, 'Regr
ession diagnostics: Identifying Influential Data and Sources of Collinearit
y', Wiley, 1980. 244-261.\n   - Quinlan,R. (1993). Combining Instance-Based a
nd Model-Based Learning. In Proceedings on the Tenth International Conference
of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Ka
ufmann.\n",
 'filename': 'boston_house_prices.csv',
 'data_module': 'sklearn.datasets.data'}
```

In [68]:

```
1 bos_df = pd.DataFrame(bos.data)
```

In [69]:
```
1 bos_df
```

Out[69]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

506 rows × 13 columns

In [70]:
```
1 bos_c =bos.feature_names
```

In [79]:
```
1 di={i:j for i,j in enumerate(bos_c)}
2 di
```

Out[79]:
```
{0: 'CRIM',
 1: 'ZN',
 2: 'INDUS',
 3: 'CHAS',
 4: 'NOX',
 5: 'RM',
 6: 'AGE',
 7: 'DIS',
 8: 'RAD',
 9: 'TAX',
 10: 'PTRATIO',
 11: 'B',
 12: 'LSTAT'}
```

In [97]:
```
x=bos_df.rename(columns=di)
x
```

Out[97]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.

506 rows × 13 columns

Traget

In [98]:
```
y = pd.Series(np.array(bos.target))
y
```

Out[98]:
```
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Length: 506, dtype: float64
```

In [100]:
```
from sklearn.model_selection import train_test_split
```

In [101]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_st
```

In [102]:
```
x_train.shape,y_train.shape
```

Out[102]: ((354, 13), (354,))

In [103]:
```
x_test.shape,y_test.shape
```

Out[103]: ((152, 13), (152,))

In [92]:
```
from sklearn.linear_model import LinearRegression
```

In [93]:
```
le1 = LinearRegression()
```

In [104]:
```
le1.fit(x_train,y_train)
```

Out[104]:
```
LinearRegression
LinearRegression()
```

In [105]:
```
le1.score(x_train,y_train)
```

Out[105]: 0.7540596363316301

In [106]:
```
le1.score(x_test,y_test)
```

Out[106]: 0.6845848781125314

Result :Overfitted

Ridge

In [108]:
```
from sklearn.linear_model import Ridge
```

In [117]:
```
ri = Ridge(alpha=2)
```

In [118]:
```
ri.fit(x_train,y_train)
```

Out[118]:
```
Ridge
Ridge(alpha=2)
```

In [119]:
```
ri.score(x_train,y_train)
```

Out[119]: 0.750952617647177

In [120]:
```
ri.score(x_test,y_test)
```

Out[120]: 0.6718372632972254

`

Result :Overfitted

Lasso

In [121]:
```
1 from sklearn.linear_model import Lasso
```

In [142]:
```
1 la = Lasso(alpha=1.2)
```

In [143]:
```
1 la.fit(x_train,y_train)
```

Out[143]:
```
Lasso
Lasso(alpha=1.2)
```

In [144]:
```
1 la.score(x_train,y_train)
```

Out[144]: 0.6863845753506665

In [145]:
```
1 la.score(x_test,y_test)
```

Out[145]: 0.65427293735519

Result :Generalized