SMART BRIDGE EXTERNSHIP IN

APPLIED DATA SCIENCE

ASSIGNMENT 3

NAME : JEYASRI VARTHINI B

REG.NO. : 20MID0049

BRANCH : Integrated M.Tech. Computer Science with specialization in Data Science

CAMPUS : VIT VELLORE

EMAIL : jeyasrivarthini.b2020@vitstudent.ac.in

DATE : 03-06-2023

JUPITER NOTEBOOK (.ipynb file) IN GITHUB REPOSITORY:

https://github.com/JeyasriVarthiniB/Smart-Bridge-Externship-in-Applied-Data-Science/blob/main/JEYASRI_VARTHINI_B_ADS_ASSIGNMENT_3.ipynb

JUPITER NOTEBOOK (.ipynb file) IN GOOGLE DRIVE:

https://drive.google.com/file/d/1v2pHT6q7L3Puz7TaVXkCNJqVgVL6NGQh/view?usp=sharing

HOUSE PRICE PREDICTION

1. Importing required libraries

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

2. Loading the dataset

In [2]:
```
data=pd.read_csv('Housing.csv')
```

In [3]:
```
data.head()
```

Out[3]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	furnished

In [4]:
```
data.tail()
```

Out[4]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	furnishingstatus
540	1820000	3000	2	1	1	yes	no	yes	no	no	2	unfurnished
541	1767150	2400	3	1	1	no	no	no	no	no	0	semi-furnished
542	1750000	3620	2	1	1	yes	no	no	no	no	0	unfurnished
543	1750000	2910	3	1	1	no	no	no	no	no	0	furnished
544	1750000	3850	3	1	2	yes	no	no	no	no	0	unfurnished

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   price             545 non-null    int64 
 1   area              545 non-null    int64 
 2   bedrooms          545 non-null    int64 
 3   bathrooms         545 non-null    int64 
 4   stories           545 non-null    int64 
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64 
 11  furnishingstatus  545 non-null    object
dtypes: int64(6), object(6)
memory usage: 51.2+ KB
```

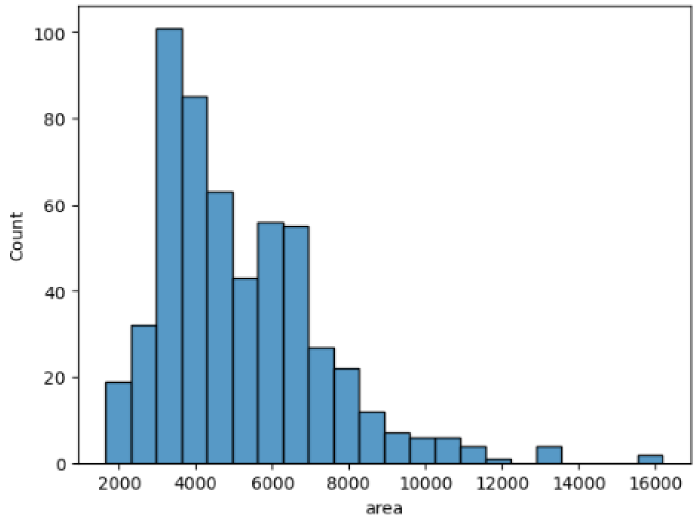
In [6]:
```
data.columns
```

Out[6]:
```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'furnishingstatus'],
      dtype='object')
```

3. Visualization - Univariate Analysis

In [7]:
```
# Histogram
sns.histplot(data['area'])
```

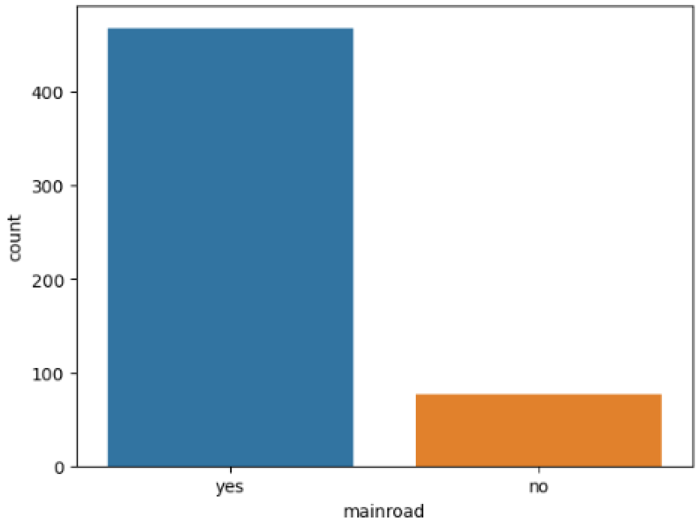
Out[7]: <AxesSubplot:xlabel='area', ylabel='Count'>



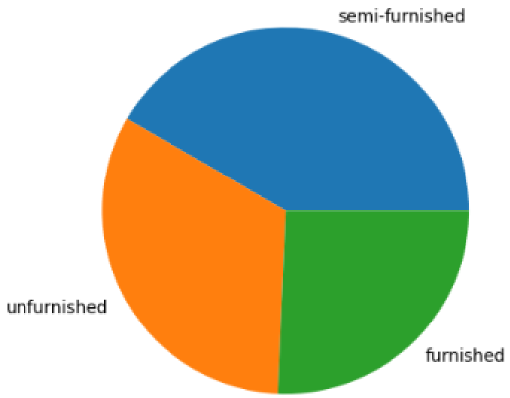
In [8]:
```
# Bar Chart
sns.countplot(data['mainroad'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword a
rg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

Out[8]: <AxesSubplot:xlabel='mainroad', ylabel='count'>



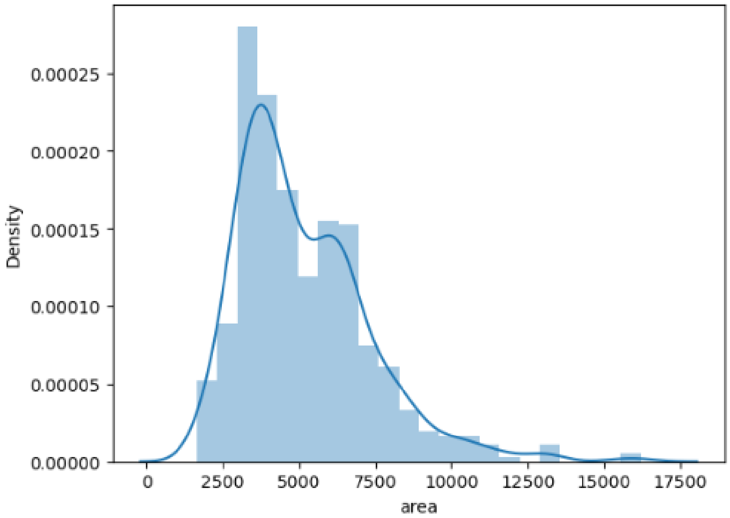
In [9]:
```python
# Pie Chart
x = data['furnishingstatus'].value_counts()
plt.pie(x.values,labels=x.index)
plt.show()
```



In [10]:
```python
# Distance Plot
sns.distplot(data.area)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[10]: `<AxesSubplot:xlabel='area', ylabel='Density'>`

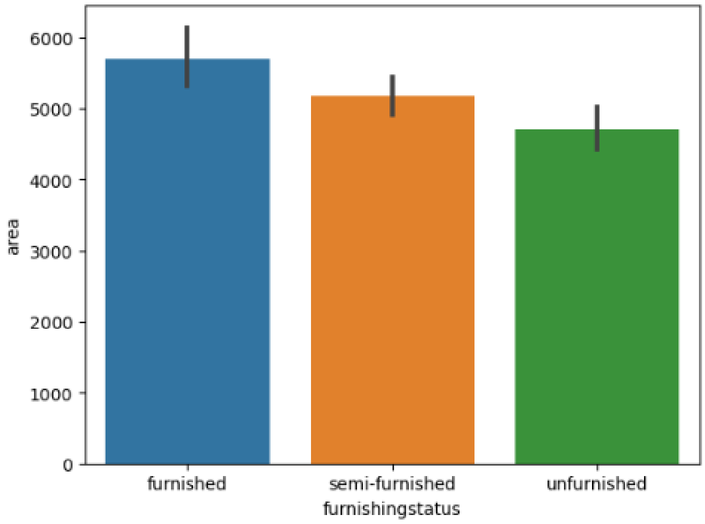


3. Visualization - Bivariate Analysis

Categorical VS Numerical

In [11]:
```
# Bar Plot
sns.barplot(x=data['furnishingstatus'], y=data['area'])
```

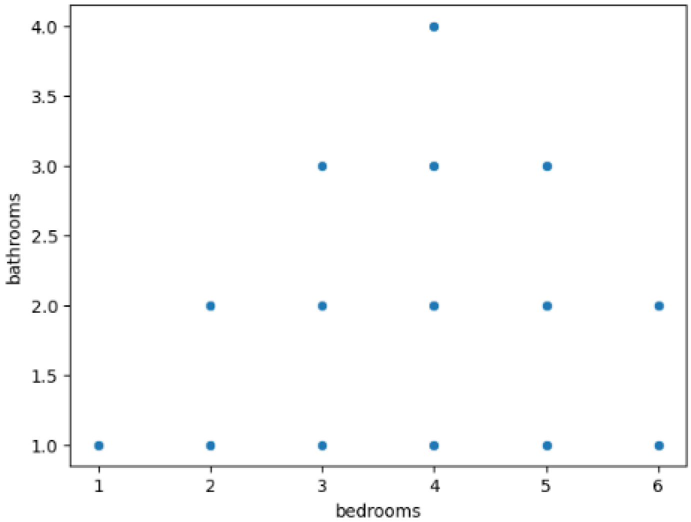
Out[11]: <AxesSubplot:xlabel='furnishingstatus', ylabel='area'>



Numerical VS Numerical

In [12]:
```
# Scatter Plot
sns.scatterplot(x=data['bedrooms'],y=data['bathrooms'])
```

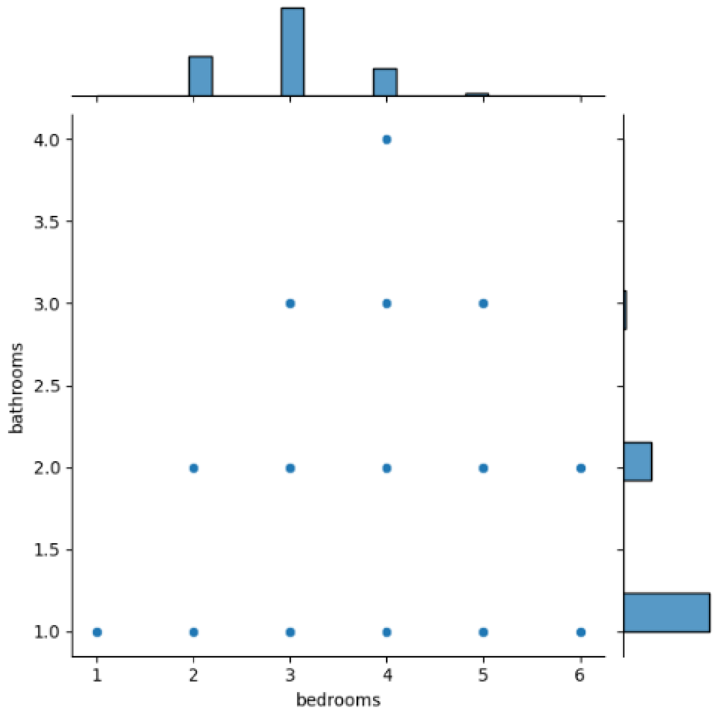
Out[12]: <AxesSubplot:xlabel='bedrooms', ylabel='bathrooms'>



In [13]:
```
# Join PLot
sns.jointplot(data.bedrooms,data.bathrooms)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword ar
gs: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit
keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[13]: `<seaborn.axisgrid.JointGrid at 0x1e982becf70>`

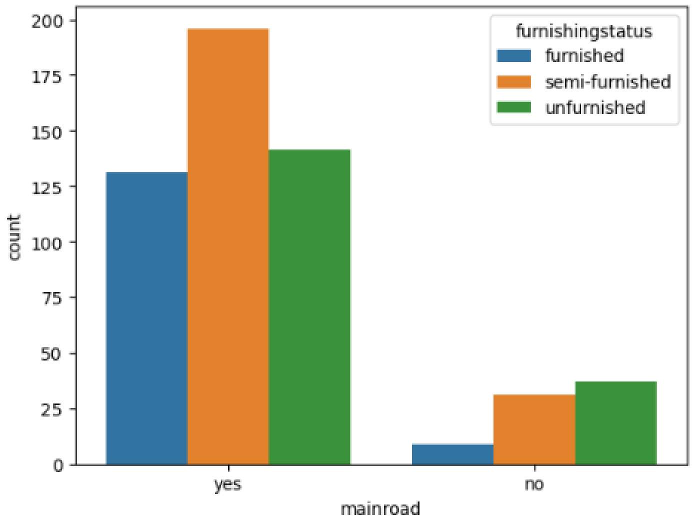


Categorical VS Categorical

In [14]:
```
# Count PLot
sns.countplot(data['mainroad'],hue=data['furnishingstatus'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword a
rg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

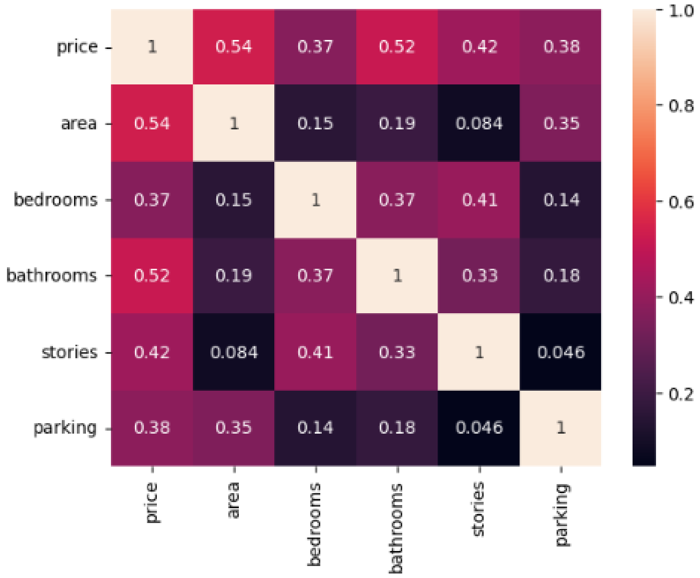
Out[14]: `<AxesSubplot:xlabel='mainroad', ylabel='count'>`



3. Visualization - Multivariate Analysis

In [15]:
```
# Heat Map
sns.heatmap(data.corr(), annot=True)
```

Out[15]: `<AxesSubplot:>`



4. Descriptive Statistics

In [16]:
```
data.describe()
```

Out[16]:

	price	area	bedrooms	bathrooms	stories	parking
count	5.450000e+02	545.000000	545.000000	545.000000	545.000000	545.000000
mean	4.766729e+06	5150.541284	2.965138	1.286239	1.805505	0.693578
std	1.870440e+06	2170.141023	0.738064	0.502470	0.867492	0.861586
min	1.750000e+06	1650.000000	1.000000	1.000000	1.000000	0.000000
25%	3.430000e+06	3600.000000	2.000000	1.000000	1.000000	0.000000
50%	4.340000e+06	4600.000000	3.000000	1.000000	2.000000	0.000000
75%	5.740000e+06	6360.000000	3.000000	2.000000	2.000000	1.000000
max	1.330000e+07	16200.000000	6.000000	4.000000	4.000000	3.000000

In [17]:
```
data.mean()
```

```
C:\Users\JEYASRI VARTHINI\AppData\Local\Temp\ipykernel_3480\531903386.py:1: FutureWarning: Dropping of nuisance columns in Data
Frame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid co
lumns before calling the reduction.
  data.mean()
```

Out[17]:
```
price        4.766729e+06
area         5.150541e+03
bedrooms     2.965138e+00
bathrooms    1.286239e+00
stories      1.805505e+00
parking      6.935780e-01
dtype: float64
```

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

```
C:\Users\JEYASRI VARTHINI\AppData\Local\Temp\ipykernel_3480\4184645713.py:1: FutureWarning: Dropping of nuisance columns in Dat
aFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid c
olumns before calling the reduction.
  data.median()
```

Out[18]:
```
price        4340000.0
area            4600.0
bedrooms           3.0
bathrooms          1.0
stories            2.0
parking            0.0
dtype: float64
```

In [19]:
```
data.mode()
```
Out[19]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	furnishingstatus
0	3500000	6000.0	3.0	1.0	2.0	yes	no	no	no	no	0.0	semi-furnished
1	4200000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [20]:
```
data.var()
```
```
C:\Users\JEYASRI VARTHINI\AppData\Local\Temp\ipykernel_3480\445316826.py:1: FutureWarning: Dropping of nuisance columns in Data
Frame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid co
lumns before calling the reduction.
  data.var()
```
Out[20]:
```
price        3.498544e+12
area         4.709512e+06
bedrooms     5.447383e-01
bathrooms    2.524757e-01
stories      7.525432e-01
parking      7.423300e-01
dtype: float64
```

In [21]:
```
data.std()
```
```
C:\Users\JEYASRI VARTHINI\AppData\Local\Temp\ipykernel_3480\2723740006.py:1: FutureWarning: Dropping of nuisance columns in Dat
aFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid c
olumns before calling the reduction.
  data.std()
```
Out[21]:
```
price        1.870440e+06
area         2.170141e+03
bedrooms     7.380639e-01
bathrooms    5.024696e-01
stories      8.674925e-01
parking      8.615858e-01
dtype: float64
```

5. Handling Missing Values

In [22]:
```
data.isna()
```
Out[22]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	furnishingstatus
0	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...
540	False	False	False	False	False	False	False	False	False	False	False	False
541	False	False	False	False	False	False	False	False	False	False	False	False
542	False	False	False	False	False	False	False	False	False	False	False	False
543	False	False	False	False	False	False	False	False	False	False	False	False
544	False	False	False	False	False	False	False	False	False	False	False	False

545 rows × 12 columns

In [23]:
```
data.isnull().any()
```
Out[23]:
```
price               False
area                False
bedrooms            False
bathrooms           False
stories             False
mainroad            False
guestroom           False
basement            False
hotwaterheating     False
airconditioning     False
parking             False
furnishingstatus    False
dtype: bool
```

In [24]:
```
data.isnull().sum()
```

Out[24]:
```
price               0
area                0
bedrooms            0
bathrooms           0
stories             0
mainroad            0
guestroom           0
basement            0
hotwaterheating     0
airconditioning     0
parking             0
furnishingstatus    0
dtype: int64
```

In [25]:
```
# Missing Values of Numerical attribute must be replaced with its Mean
# Missing Values of Categorical attribute must be replaced with frequently occuring category
# But here there are no missing values
print("No Missing Values!")
```

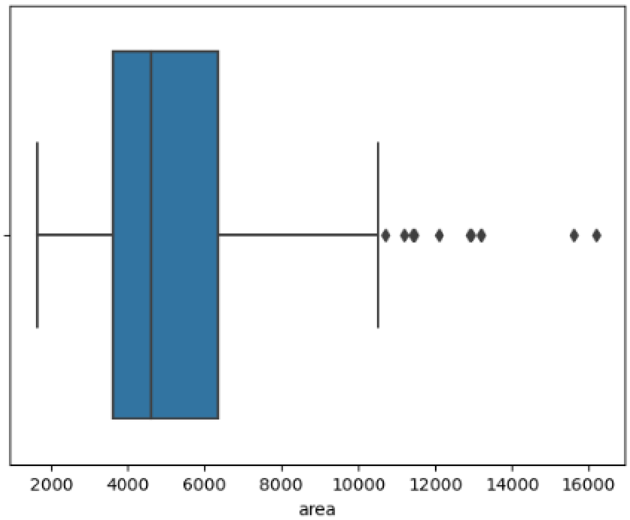
```
No Missing Values!
```

6. Finding and replacing the outliers

In [26]:
```
# area (Numerical attribute)
sns.boxplot(data.area)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword a
rg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

Out[26]:
```
<AxesSubplot:xlabel='area'>
```



In [27]:
```
max_threshold = data.area.quantile(0.95)
max_threshold
```

Out[27]:
```
9000.0
```

In [28]:
```
min_threshold = data.area.quantile(0.05)
min_threshold
```

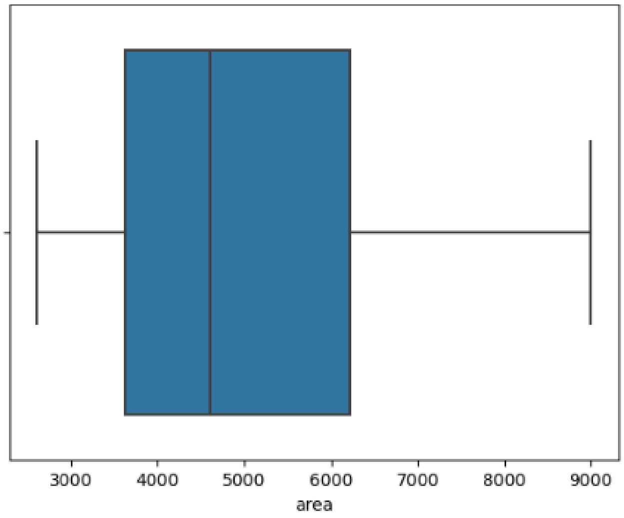
Out[28]:
```
2562.0
```

In [29]:
```
data = data[data.area<=max_threshold]
data = data[data.area>=min_threshold]
```

In [30]:
```
sns.boxplot(data.area)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword a
rg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

Out[30]: <AxesSubplot:xlabel='area'>



7. Encoding - Label Encoding

In [31]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data.mainroad = le.fit_transform(data.mainroad)
data.guestroom = le.fit_transform(data.guestroom)
data.basement = le.fit_transform(data.basement)
data.hotwaterheating = le.fit_transform(data.hotwaterheating)
data.airconditioning = le.fit_transform(data.airconditioning)
```

In [32]:
```
data.head()
```

Out[32]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	furnishingstatus
0	13300000	7420	4	2	3	1	0	0	0	1	2	furnished
1	12250000	8960	4	4	4	1	0	0	0	1	3	furnished
3	12215000	7500	4	2	2	1	0	1	0	1	3	furnished
4	11410000	7420	4	1	2	1	1	1	0	1	2	furnished
5	10850000	7500	3	3	1	1	0	1	0	1	2	semi-furnished

7. Encoding - One Hot Encoding

In [33]:
```
data = pd.get_dummies(data,columns=['furnishingstatus'])
```

In [34]:
```
data.head()
```

Out[34]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	furnishingstatus_furnished	furnis
0	13300000	7420	4	2	3	1	0	0	0	1	2	1	
1	12250000	8960	4	4	4	1	0	0	0	1	3	1	
3	12215000	7500	4	2	2	1	0	1	0	1	3	1	
4	11410000	7420	4	1	2	1	1	1	0	1	2	1	
5	10850000	7500	3	3	1	1	0	1	0	1	2	0	

8. Splitting data into dependent and independent variables

In [35]:
```
# Dependent/Target variable y
y = data['price']
y.head()
```

Out[35]:
```
0    13300000
1    12250000
3    12215000
4    11410000
5    10850000
Name: price, dtype: int64
```

In [36]:
```
# Independent/Predictor variable x
x = data.drop(columns=['price'],axis=1)
x.head()
```

Out[36]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	furnishingstatus_furnished	furnishingstatus_ furi
0	7420	4	2	3	1	0	0	0	1	2	1	
1	8960	4	4	4	1	0	0	0	1	3	1	
3	7500	4	2	2	1	0	1	0	1	3	1	
4	7420	4	1	2	1	1	1	0	1	2	1	
5	7500	3	3	1	1	0	1	0	1	2	0	

9. Scaling the independent variables

In [37]:
```
# Minmax Scaling (Scaling values between 0 and 1)
name = x.columns
name
```

Out[37]:
```
Index(['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom',
       'basement', 'hotwaterheating', 'airconditioning', 'parking',
       'furnishingstatus_furnished', 'furnishingstatus_semi-furnished',
       'furnishingstatus_unfurnished'],
      dtype='object')
```

In [38]:
```
from sklearn.preprocessing import MinMaxScaler
scale = MinMaxScaler()
x_scaled = scale.fit_transform(x)
x_scaled
```

Out[38]:
```
array([[0.75273865, 0.6       , 0.33333333, ..., 1.        , 0.        ,
        0.        ],
       [0.99374022, 0.6       , 1.        , ..., 1.        , 0.        ,
        0.        ],
       [0.76525822, 0.6       , 0.33333333, ..., 1.        , 0.        ,
        0.        ],
       ...,
       [0.15805947, 0.2       , 0.        , ..., 0.        , 0.        ,
        1.        ],
       [0.04694836, 0.4       , 0.        , ..., 1.        , 0.        ,
        0.        ],
       [0.19405321, 0.4       , 0.        , ..., 0.        , 0.        ,
        1.        ]])
```

In [39]:
```
x = pd.DataFrame(x_scaled,columns=name)
x
```

Out[39]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	furnishingstatus_furnished	furnishi
0	0.752739	0.6	0.333333	0.666667	1.0	0.0	0.0	0.0	1.0	0.666667	1.0	
1	0.993740	0.6	1.000000	1.000000	1.0	0.0	0.0	0.0	1.0	1.000000	1.0	
2	0.765258	0.6	0.333333	0.333333	1.0	0.0	1.0	0.0	1.0	1.000000	1.0	
3	0.752739	0.6	0.000000	0.333333	1.0	1.0	1.0	0.0	1.0	0.666667	1.0	
4	0.765258	0.4	0.666667	0.000000	1.0	0.0	1.0	0.0	1.0	0.666667	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	
487	0.059468	0.2	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.333333	0.0	
488	0.061033	0.2	0.000000	0.000000	1.0	0.0	1.0	0.0	0.0	0.666667	0.0	
489	0.158059	0.2	0.000000	0.000000	1.0	0.0	0.0	0.0	0.0	0.000000	0.0	
490	0.046948	0.4	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	1.0	
491	0.194053	0.4	0.000000	0.333333	1.0	0.0	0.0	0.0	0.0	0.000000	0.0	

492 rows × 13 columns

10. Splitting data into Training and Testing data

In [40]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=0)
```

In [41]:
```
x_train.head()
```

Out[41]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	furnishingstatus_furnished	furnishi
8	0.530516	0.6	0.666667	0.333333	1.0	1.0	1.0	1.0	0.0	0.666667	0.0	
488	0.061033	0.2	0.000000	0.000000	1.0	0.0	1.0	0.0	0.0	0.666667	0.0	
426	0.178404	0.4	0.000000	0.333333	1.0	0.0	0.0	0.0	0.0	0.000000	0.0	
224	0.142410	0.4	0.000000	0.000000	1.0	0.0	0.0	0.0	0.0	0.000000	0.0	
157	0.452269	0.4	0.333333	0.000000	1.0	0.0	1.0	0.0	0.0	0.000000	0.0	

In [42]:
```
x_test.head()
```

Out[42]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	furnishingstatus_furnished	furnishi
362	0.161189	0.2	0.000000	0.000000	1.0	0.0	0.0	0.0	0.0	0.333333	0.0	
96	0.452269	0.4	0.333333	0.000000	1.0	1.0	1.0	0.0	0.0	0.666667	1.0	
320	0.154930	0.2	0.000000	0.000000	1.0	0.0	0.0	0.0	0.0	0.000000	0.0	
144	0.032394	0.6	0.333333	0.333333	0.0	1.0	1.0	0.0	0.0	0.333333	1.0	
15	0.311424	0.4	0.333333	0.333333	1.0	1.0	0.0	0.0	1.0	0.666667	1.0	

In [43]:
```
y_train
```

Out[43]:
```
11     9681000
540    1820000
471    3010000
246    4550000
170    5250000
        ...
354    3780000
209    4900000
127    5880000
50     7420000
188    5075000
Name: price, Length: 393, dtype: int64
```

In [44]:
```
y_test
```

Out[44]:
```
396    3500000
104    6195000
351    3780000
157    5495000
18     8890000
        ...
442    3220000
247    4550000
214    4865000
9      9800000
299    4200000
Name: price, Length: 99, dtype: int64
```

MODEL 1 - MULTILINEAR REGRESSION

In [45]:
```
# 11. Building the Model
from sklearn.linear_model import LinearRegression
model1=LinearRegression()
model1
```

Out[45]:
```
LinearRegression()
```

In [46]:
```
# 12. Training the Model
model1.fit(x_train,y_train)
```

Out[46]:
```
LinearRegression()
```

In [47]:
```
# 13. Testing the Model
pred1=model1.predict(x_test)
pred1
```

Out[47]:
```
array([ 3328818.76048527,  5763895.7176806 ,  3081902.04922748,
        4503608.02876151,  6421105.36629327,  2058972.50736273,
        4984367.03059062,  2514102.0303344 ,  7710301.40816266,
        2767980.93799451,  3097588.23335861,  3119207.21764157,
        5386112.09601322,  4254840.23378505,  5421179.20407663,
        3352268.1259543 ,  4851252.88912829,  4777511.33100125,
        4135466.85407371,  3662855.856973  ,  3988968.03978978,
        4475700.8914891 ,  3795491.88479888,  1879099.38795548,
        2948114.31564679,  5951762.08625314,  6036259.35636536,
        4173047.58761786,  7136145.37613913,  6250970.63661024,
        3866649.2507309 ,  5924925.01142682,  3121759.74669283,
        5447391.19354001,  4812819.24778435,  4803051.13513427,
        3656103.88414531,  5457211.73407273,  2093457.58089213,
        5765104.84552782,  7046474.23369589,  5445172.61030276,
        3499056.59805479,  3743968.20354876,  3603757.3596415 ,
        3268536.89459244,  2963693.59846905,  5794637.78379926,
        5866442.13328873,  5746936.12487649,  5857244.45978014,
        3940601.32894725,  4157505.93691058,  6362247.78582878,
        6539448.19821126,  4650103.84503661,  3829773.41103846,
       10710371.73840569,  2283063.38530379,  4387405.78055451,
        3471596.7311487 ,  5369107.00307802,  5029349.70549483,
        5760901.80385473,  4117931.60154501,  3172342.37789729,
        5103991.72394619,  3851015.55968428,  2818989.07259735,
        2201066.95802002,  2674284.54672564,  3229104.32211483,
        4093010.40559483,  4083812.73208624,  3342927.6625247 ,
        2638108.41525772,  2201549.34086609,  4968559.48171642,
        8335778.65860652,  3367702.37033429,  5489246.77634093,
        5327373.72770443,  3090310.05860679,  4670953.2379453 ,
        5832261.34119973,  4534656.7214529 ,  7884396.5320023 ,
        5494845.74279105,  5074088.10584069,  5906385.42684057,
        5225718.13616348,  7309975.58165652,  3725866.41733494,
        6186397.60247015,  3529616.28607638,  6584940.09660297,
        3323206.010071  ,  7100735.28342157,  5047060.5925485 ])
```

In [48]:
```
# 14. Evaluating the Model
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
mse1=mean_squared_error(y_test,pred1)
mae1=mean_absolute_error(y_test,pred1)
rmse1=np.sqrt(mse1)
rsquare1=r2_score(y_test,pred1)
print("Mean Squared Error: ",mse1)
print("Mean Absolute Error: ",mae1)
print("Root Mean Squared Error: ",rmse1)
print("R Squared Error: ",rsquare1)
```

```
Mean Squared Error:  1217008374055.8647
Mean Absolute Error:  830709.4613095429
Root Mean Squared Error:  1103181.0250615557
R Squared Error:  0.6542748617098755
```

MODEL 2 - RIDGE REGRESSION

In [50]:
```
# 11. Building the Model
from sklearn.linear_model import Ridge
model2=Ridge()
model2
```

Out[50]:
```
Ridge()
```

In [51]:
```
# 12. Training the Model
model2.fit(x_train,y_train)
model2
```

Out[51]:
```
Ridge()
```

In [52]:
```
# 13. Testing the Model
pred2=model2.predict(x_test)
pred2
```

Out[52]:
```
array([ 3360099.80443938,  5731500.59591719,  3107879.71152796,
        4513719.6634283 ,  6407420.73760748,  2090884.21209372,
        4957226.15212752,  2546629.92985781,  7629408.55588469,
        2802261.11304925,  3136365.00863388,  3148589.87514598,
        5323023.73994059,  4308739.66088491,  5363884.99813509,
        3373598.49583199,  4857827.10420864,  4779289.76442721,
        4185024.27618389,  3719882.33860411,  3987157.7193756 ,
        4491163.60308308,  3842518.65374757,  1908258.48543537,
        2987852.69635149,  5979409.23050079,  6081797.65273961,
        4218175.3945718 ,  7080080.2658836 ,  6239756.69493376,
        3866412.02086666,  5937425.18307513,  3155187.0300452 ,
        5453220.96819767,  4827101.89705395,  4834753.58310763,
        3694528.61816886,  5479234.48675427,  2129103.39077922,
        5767020.79662496,  6973872.33467283,  5417676.71698012,
        3514633.73010097,  3782628.82230772,  3625045.83278775,
        3283202.15611529,  2970591.16283916,  5730606.06867023,
        5803530.70468793,  5704084.97969409,  5809136.45775237,
        3979899.82946819,  4121014.70698343,  6350853.38591425,
        6462846.39022496,  4696655.17018637,  3875478.37704146,
       10500371.65415368,  2340373.46641467,  4439020.29991564,
        3492701.09139955,  5386547.43815237,  5037528.33669254,
        5772486.74058616,  4129915.55957979,  3195033.01032677,
        5096477.90108265,  3881256.3639521 ,  2831145.88476107,
        2218524.22915913,  2691700.60668297,  3270885.64065921,
        4138931.80485285,  4144537.55791729,  3393965.4078627 ,
        2656839.28716345,  2256834.57235379,  4991267.66242493,
        8280033.479789  ,  3407318.93295193,  5409723.24502951,
        5353389.03265805,  3092605.78115749,  4653454.75861369,
        5780379.19727548,  4498735.55016513,  7799291.5763544 ,
        5539250.15534859,  5105229.83495519,  5949620.01097838,
        5272920.84540906,  7330334.24670875,  3756994.37335682,
        6141867.05439148,  3570365.34438066,  6578178.46138535,
        3333430.0330371 ,  7057211.02888619,  5058916.7389424 ])
```

In [53]:
```
# 14. Evaluating the Model
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
mse2=mean_squared_error(y_test,pred2)
mae2=mean_absolute_error(y_test,pred2)
rmse2=np.sqrt(mse2)
rsquare2=r2_score(y_test,pred2)
print("Mean Squared Error: ",mse2)
print("Mean Absolute Error: ",mae2)
print("Root Mean Squared Error: ",rmse2)
print("R Squared Error: ",rsquare2)
```

```
Mean Squared Error:  1214412307043.8303
Mean Absolute Error:  826642.0533119544
Root Mean Squared Error:  1102003.7690697026
R Squared Error:  0.6550123468791474
```

MODEL 3 - LASSO REGRESSION

In [54]:
```
# 11. Building the Model
from sklearn.linear_model import Lasso
model3=Lasso()
model3
```

Out[54]:
```
Lasso()
```

In [55]:
```
# 12. Training the Model
model3.fit(x_train,y_train)
model3
```

Out[55]:
```
Lasso()
```

In [56]:
```
# 13. Testing the Model
pred3=model3.predict(x_test)
pred3
```

Out[56]:
```
array([ 3328827.34261459,  5763887.06917493,  3081912.22161613,
        4503602.8190595 ,  6421097.24895035,  2058983.88046673,
        4984361.81454399,  2514106.96047143,  7710294.02187369,
        2767985.98411184,  3097595.87852427,  3119212.40183241,
        5386101.65192548,  4254836.83458936,  5421174.64350597,
        3352279.56957091,  4851255.62103755,  4777507.38855251,
        4135469.36391448,  3662862.59966151,  3988978.61292032,
        4475709.0740419 ,  3795496.80528129,  1879117.44596477,
        2948124.38887408,  5951765.05580497,  6036259.02361685,
        4173049.22528792,  7136141.80200454,  6250967.44769928,
        3866651.89908531,  5924927.06324853,  3121769.58251482,
        5447391.17602615,  4812819.32934906,  4803053.51249745,
        3656109.55472094,  5457203.92438954,  2093474.09315559,
        5765107.54845454,  7046464.61190222,  5445166.33164861,
        3499061.26292155,  3743974.54454715,  3603765.56587911,
        3268548.3136077 ,  2963704.35291718,  5794624.75971086,
        5866415.85127918,  5746935.1396429 ,  5857213.39222811,
        3940604.53709285,  4157515.05919732,  6362233.73012226,
        6539444.77897433,  4650099.99252131,  3829757.05411894,
       10710344.87265922,  2283080.85876263,  4387405.87247875,
        3471599.93176733,  5369102.17786931,  5029329.4923611 ,
        5760904.47126505,  4117920.52796477,  3172352.42663735,
        5103994.84331363,  3851023.25612005,  2819000.02488322,
        2201084.57584032,  2674295.69684927,  3229108.92345468,
        4093011.33261318,  4083808.87356211,  3342930.24945849,
        2638119.61484078,  2201561.73946273,  4968561.64311556,
        8335763.2590655 ,  3367714.87419351,  5489239.78826689,
        5327372.88244494,  3090320.63994689,  4670959.98512954,
        5832253.72498161,  4534664.07542635,  7884389.72777051,
        5494838.23000059,  5074083.6736608 ,  5906381.42368542,
        5225714.70494421,  7309964.30189353,  3725873.20394829,
        6186374.30522825,  3529624.9298692 ,  6584932.50816497,
        3323215.84039803,  7100728.43184472,  5047059.52204994])
```

In [57]:
```
# 14. Evaluating the Model
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
mse3=mean_squared_error(y_test,pred3)
mae3=mean_absolute_error(y_test,pred3)
rmse3=np.sqrt(mse3)
rsquare3=r2_score(y_test,pred3)
print("Mean Squared Error: ",mse3)
print("Mean Absolute Error: ",mae3)
print("Root Mean Squared Error: ",rmse3)
print("R Squared Error: ",rsquare3)
```

```
Mean Squared Error:  1217009489243.4714
Mean Absolute Error:  830708.3912323378
Root Mean Squared Error:  1103181.5305032402
R Squared Error:  0.6542745449097631
```

MODEL 4 - DECISION TREE

In [58]:
```
# 11. Building the Model
from sklearn.tree import DecisionTreeClassifier
model4=DecisionTreeClassifier(criterion='entropy',random_state=0)
model4
```

Out[58]:
```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In [59]:
```
# 12. Training the Model
model4.fit(x_train,y_train)
model4
```

Out[59]:
```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In [60]:
```
# 13. Testing the Model
pred4=model4.predict(x_test)
pred4
```

Out[60]:
```
array([ 2345000,  3703000,  3710000,  4956000,  6405000,  2310000,
        4095000,  2520000,  5250000,  3500000,  2135000,  4767000,
        5740000,  3010000,  8400000,  3325000,  5565000,  2940000,
        4585000,  2380000,  4270000,  3780000,  4403000,  2135000,
        3465000,  5810000,  5565000,  3640000,  7560000,  7070000,
        6230000,  4760000,  2380000,  5243000,  5950000,  4900000,
        3500000,  6125000,  2450000,  4830000,  5005000,  4585000,
        4200000,  4550000,  3605000,  4830000,  4200000,  5243000,
        4095000,  4200000,  3360000,  3640000,  2730000,  3640000,
        5880000,  4767000,  4193000,  7980000,  2940000,  3640000,
        4060000,  5803000,  3010000,  4900000,  3360000,  3605000,
        6125000,  3290000,  3430000,  2275000,  3710000,  4200000,
        3010000,  3360000,  1960000,  2380000,  2940000,  4200000,
        7245000,  4200000,  6230000,  4200000,  3325000,  3150000,
        6090000,  3780000, 12215000,  5565000,  3780000,  5565000,
        4620000, 11410000,  6300000,  5523000,  2450000,  6090000,
        3220000,  5740000,  3773000], dtype=int64)
```

In [61]:
```
# 14. Evaluating the Model
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
mse4=mean_squared_error(y_test,pred4)
mae4=mean_absolute_error(y_test,pred4)
rmse4=np.sqrt(mse4)
rsquare4=r2_score(y_test,pred4)
print("Mean Squared Error: ",mse4)
print("Mean Absolute Error: ",mae4)
print("Root Mean Squared Error: ",rmse4)
print("R Squared Error: ",rsquare4)
```

```
Mean Squared Error:  2691011449494.9497
Mean Absolute Error:  1161858.5858585858
Root Mean Squared Error:  1640430.2635269046
R Squared Error:  0.23554321781992626
```

MODEL 5 - RANDOM FOREST

In [62]:
```
# 11. Building the Model
from sklearn.ensemble import RandomForestClassifier
model5=RandomForestClassifier(n_estimators=10,criterion='entropy',random_state=0)
model5
```

Out[62]:
```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

In [63]:
```
# 12. Training the Model
model5.fit(x_train,y_train)
model5
```

Out[63]:
```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

In [64]:
```
# 13. Testing the Model
pred5=model5.predict(x_test)
pred5
```

Out[64]:
```
array([ 2660000,  5530000,  3710000,  2590000,  7210000,  2520000,
        5215000,  3465000,  8960000,  1960000,  2450000,  3500000,
        5740000,  4767000,  7350000,  3710000,  6720000,  2940000,
        4585000,  2520000,  3703000,  3605000,  4515000,  2135000,
        2520000,  5810000,  4620000,  3640000,  5775000,  5495000,
        3080000,  4200000,  3675000,  5425000,  3290000,  2653000,
        4550000,  4970000,  2450000,  4550000,  8463000,  3850000,
        3220000,  5040000,  3605000,  3290000,  3290000,  5600000,
        5215000,  3640000,  4340000,  3640000,  3003000,  4060000,
        5880000,  4382000,  3500000,  7980000,  3920000,  3640000,
        4060000,  5229000,  3010000,  3640000,  3360000,  3605000,
        6125000,  2275000,  3430000,  2408000,  1750000,  5110000,
        3010000,  4060000,  2940000,  1750000,  1750000,  4123000,
        5880000,  2520000,  5383000,  4620000,  2940000,  5145000,
        6090000,  3465000,  4760000,  5565000,  4382000,  3395000,
        4200000, 11410000,  3850000,  5005000,  2653000,  2852500,
        3990000,  5740000,  3773000], dtype=int64)
```

In [65]:
```
# 14. Evaluating the Model
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
mse5=mean_squared_error(y_test,pred5)
mae5=mean_absolute_error(y_test,pred5)
rmse5=np.sqrt(mse5)
rsquare5=r2_score(y_test,pred5)
print("Mean Squared Error: ",mse5)
print("Mean Absolute Error: ",mae5)
print("Root Mean Squared Error: ",rmse5)
print("R Squared Error: ",rsquare5)
```

```
Mean Squared Error:  2133376229797.9797
Mean Absolute Error:  1079378.7878787878
Root Mean Squared Error:  1460608.171207453
R Squared Error:  0.39395503942693944
```

MODEL 6 - K NEAREST NEIGHBOUR

In [66]:
```
# 11. Building the Model
from sklearn.neighbors import KNeighborsClassifier
model6=KNeighborsClassifier()
model6
```

Out[66]:
```
KNeighborsClassifier()
```

In [67]:
```
# Training the Model
model6.fit(x_train,y_train)
model6
```

Out[67]:
```
KNeighborsClassifier()
```

In [68]:
```
# Testing the Model
pred6=model6.predict(x_test)
pred6
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction func
tions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, th
is behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will b
e eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

Out[68]:
```
array([2660000, 3920000, 3010000, 3290000, 5950000, 2135000, 3010000,
       2310000, 4830000, 2940000, 3640000, 1750000, 2653000, 4193000,
       3640000, 2800000, 2653000, 2940000, 3500000, 2380000, 3500000,
       4095000, 3920000, 2135000, 2135000, 5740000, 5460000, 3500000,
       5740000, 3640000, 2275000, 4098500, 2233000, 4900000, 3290000,
       4382000, 2520000, 4123000, 2135000, 6650000, 4900000, 3850000,
       1750000, 3885000, 2660000, 1750000, 2660000, 4130000, 3080000,
       3640000, 3920000, 3500000, 2730000, 4060000, 5740000, 4382000,
       2100000, 7035000, 1750000, 3850000, 2660000, 4340000, 3080000,
       3640000, 3150000, 3605000, 2275000, 3640000, 2660000, 2135000,
       1750000, 2275000, 3010000, 3500000, 2485000, 1750000, 1750000,
       4340000, 4620000, 2520000, 3640000, 4620000, 2660000, 2870000,
       4235000, 2730000, 4473000, 3640000, 4382000, 3395000, 4025000,
       5740000, 2660000, 4900000, 2653000, 4060000, 2275000, 5740000,
       3500000], dtype=int64)
```

In [69]:
```
# 14. Evaluating the Model
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
mse6=mean_squared_error(y_test,pred6)
mae6=mean_absolute_error(y_test,pred6)
rmse6=np.sqrt(mse6)
rsquare6=r2_score(y_test,pred6)
print("Mean Squared Error: ",mse6)
print("Mean Absolute Error: ",mae6)
print("Root Mean Squared Error: ",rmse6)
print("R Squared Error: ",rsquare6)
```

```
Mean Squared Error:  3502932664141.414
Mean Absolute Error:  1432914.1414141415
Root Mean Squared Error:  1871612.3167315966
R Squared Error:  0.004894374148576297
```

MODEL 7 - NAIVE BAYES

In [70]:
```
# 11. Building the Model
from sklearn.naive_bayes import GaussianNB
model7=GaussianNB()
model7
```

Out[70]:
```
GaussianNB()
```

In [71]:
```
# 12. Training the Model
model7.fit(x_train,y_train)
model7
```

Out[71]:
```
GaussianNB()
```

In [72]:
```
# Testing the Model
pred7=model7.predict(x_test)
pred7
```

Out[72]:
```
array([4095000, 5950000, 3605000, 5215000, 6195000, 1750000, 3850000,
       3465000, 6440000, 1750000, 2450000, 2100000, 6160000, 3850000,
       4095000, 1750000, 2653000, 4410000, 3773000, 2380000, 3773000,
       4095000, 4193000, 2520000, 3990000, 5145000, 7350000, 4165000,
       7560000, 5145000, 6230000, 5565000, 3990000, 6160000, 4235000,
       5145000, 1750000, 3850000, 2450000, 4270000, 5460000, 3850000,
       3325000, 5460000, 4095000, 3325000, 3465000, 3850000, 4340000,
       4095000, 3920000, 4165000, 3465000, 6300000, 5880000, 4095000,
       2100000, 6300000, 3773000, 4165000, 4193000, 6020000, 3780000,
       4095000, 4200000, 3605000, 5880000, 2450000, 3430000, 1750000,
       3430000, 2800000, 4165000, 3773000, 3010000, 3430000, 1750000,
       6020000, 7350000, 1750000, 3150000, 4620000, 3465000, 4095000,
       6300000, 3465000, 5950000, 4095000, 4095000, 6020000, 5565000,
       5565000, 3850000, 3920000, 4270000, 2730000, 2450000, 7350000,
       3773000], dtype=int64)
```

In [73]:
```
# 14. Evaluating the Model
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
mse7=mean_squared_error(y_test,pred7)
mae7=mean_absolute_error(y_test,pred7)
rmse7=np.sqrt(mse7)
rsquare7=r2_score(y_test,pred7)
print("Mean Squared Error: ",mse7)
print("Mean Absolute Error: ",mae7)
print("Root Mean Squared Error: ",rmse7)
print("R Squared Error: ",rsquare7)
```

```
Mean Squared Error:  2600759883838.384
Mean Absolute Error:  1172111.111111111
Root Mean Squared Error:  1612687.162421275
R Squared Error:  0.2611816897340761
```

MODEL 8 - SUPPORT VECTOR MACHINE

In [74]:
```
# 11. Building the Model
from sklearn.svm import SVC
model8 = SVC(kernel='rbf')
model8
```

Out[74]: SVC()

In [75]:
```
# 12. Training the Model
model8.fit(x_train,y_train)
model8
```

Out[75]: SVC()

In [76]:
```
# 13. Testing th Model
pred8=model8.predict(x_test)
pred8
```

Out[76]:
```
array([4200000, 5950000, 4200000, 5950000, 5950000, 2940000, 4200000,
       3500000, 5950000, 2940000, 2450000, 3500000, 4200000, 4200000,
       4200000, 4270000, 3360000, 4200000, 5950000, 5950000, 5950000,
       4200000, 4200000, 2940000, 4200000, 4200000, 5950000, 4900000,
       4200000, 4900000, 4200000, 5950000, 4200000, 4900000, 5950000,
       4200000, 5950000, 4900000, 2940000, 5950000, 4900000, 4900000,
       3500000, 4900000, 4200000, 3500000, 3500000, 4900000, 4200000,
       4200000, 3920000, 4900000, 3500000, 4900000, 5740000, 4200000,
       3500000, 5950000, 2940000, 4900000, 4200000, 4900000, 4200000,
       4900000, 3640000, 4200000, 4200000, 3640000, 3500000, 2940000,
       3500000, 4200000, 4200000, 5950000, 4200000, 3500000, 2940000,
       4900000, 7350000, 5950000, 3640000, 4200000, 3500000, 4200000,
       4200000, 3500000, 5950000, 4200000, 4200000, 5950000, 5950000,
       5950000, 4200000, 4900000, 3360000, 8400000, 4200000, 5740000,
       5950000], dtype=int64)
```

In [77]:
```
# 14. Evaluating the Model
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
mse8=mean_squared_error(y_test,pred8)
mae8=mean_absolute_error(y_test,pred8)
rmse8=np.sqrt(mse8)
rsquare8=r2_score(y_test,pred8)
print("Mean Squared Error: ",mse8)
print("Mean Absolute Error: ",mae8)
print("Root Mean Squared Error: ",rmse8)
print("R Squared Error: ",rsquare8)
```

```
Mean Squared Error:  2591131631313.1313
Mean Absolute Error:  1115686.8686868686
Root Mean Squared Error:  1609699.2362901622
R Squared Error:  0.26391686313694396
```

FINDING THE BEST REGRESSION MODEL

WITH LOWEST R SQUARE ERROR VALUE

In [78]:
```
models = ["Multilinear Regression","Ridge Regression","Lasso Regression","Decision Tree","Random Forest","K Nearest Neighbour","N
rsquare_values = [rsquare1,rsquare2,rsquare3,rsquare4,rsquare5,rsquare6,rsquare7,rsquare8]
plt.figure(figsize = (15, 5))
plt.bar(models,rsquare_values)
plt.title('R SQUARE ERROR VALUES OF DIFFERENT REGRESSION MODELS')
plt.xlabel('Regression Models')
plt.ylabel('Value of R Square Error Values')
plt.show()
```

In [79]:
```
# From the plot, it can be concluded that "K Nearest Neighbour Model" has the Lowest R Square error values.
# Hence, K Nearest Neighbour is the best Regression Model for House Price Preditcion.
```

In [80]:
```
min_index = 0
i = 0
while (i<8):
    if (rsquare_values[i]==min(rsquare_values)):
        min_index = i
        break
    i += 1
print("The Best Regression Model for House Price Prediction is ",models[min_index],"with the least R Square Error Value of ",rsqu
```

```
The Best Regression Model for House Price Prediction is  K Nearest Neighbour with the least R Square Error Value of  0.00489437
4148576297
```