Vellore Institute of Technology

School of Computer Science and Engineering

M.tech Data Science

Name: Suriya Narayanan S

Reg.no: 20MID0059

Campus : Vellore

Date: 05-06-2023

Smart Bridge internship training

ADS Assignment -1

1. Download the dataset: Dataset

```
The dataset (housing.csv) is download from the link provided and saved in the
appropriate repository
```

2. Load the dataset into the tool.

In [1]:
```
# importing appropriate packages
import pandas as pd
import numpy as np
```

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

In [3]:
```
house
```

Out[3]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterl
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	
...	...	...	...	...	...	...	...	...	
540	1820000	3000	2	1	1	yes	no	yes	
541	1767150	2400	3	1	1	no	no	no	
542	1750000	3620	2	1	1	yes	no	no	
543	1750000	2910	3	1	1	no	no	no	
544	1750000	3850	3	1	2	yes	no	no	

545 rows × 12 columns

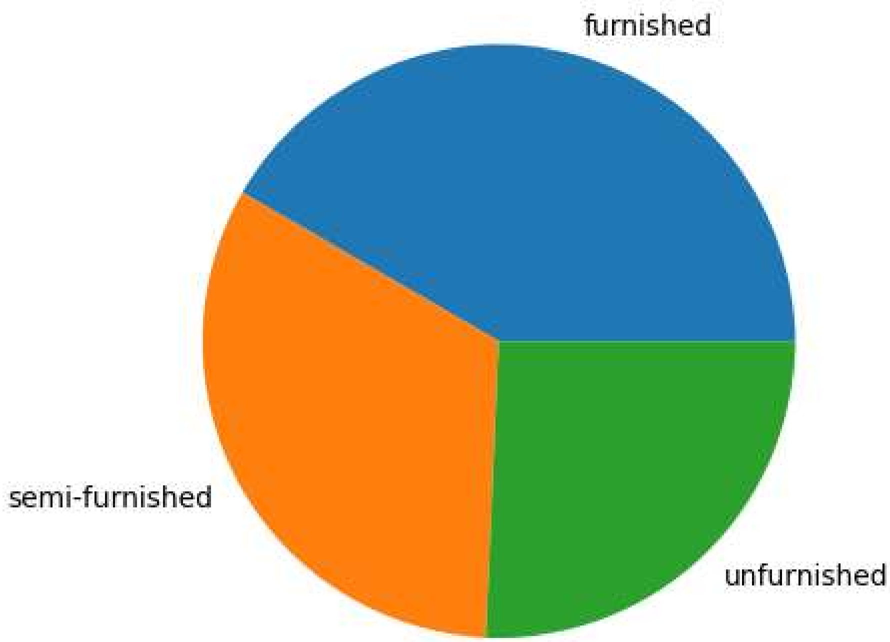
3. Perform Below Visualizations.

In [4]:
```
# importing the required package
import matplotlib.pyplot as plt
```

Univariate Analysis

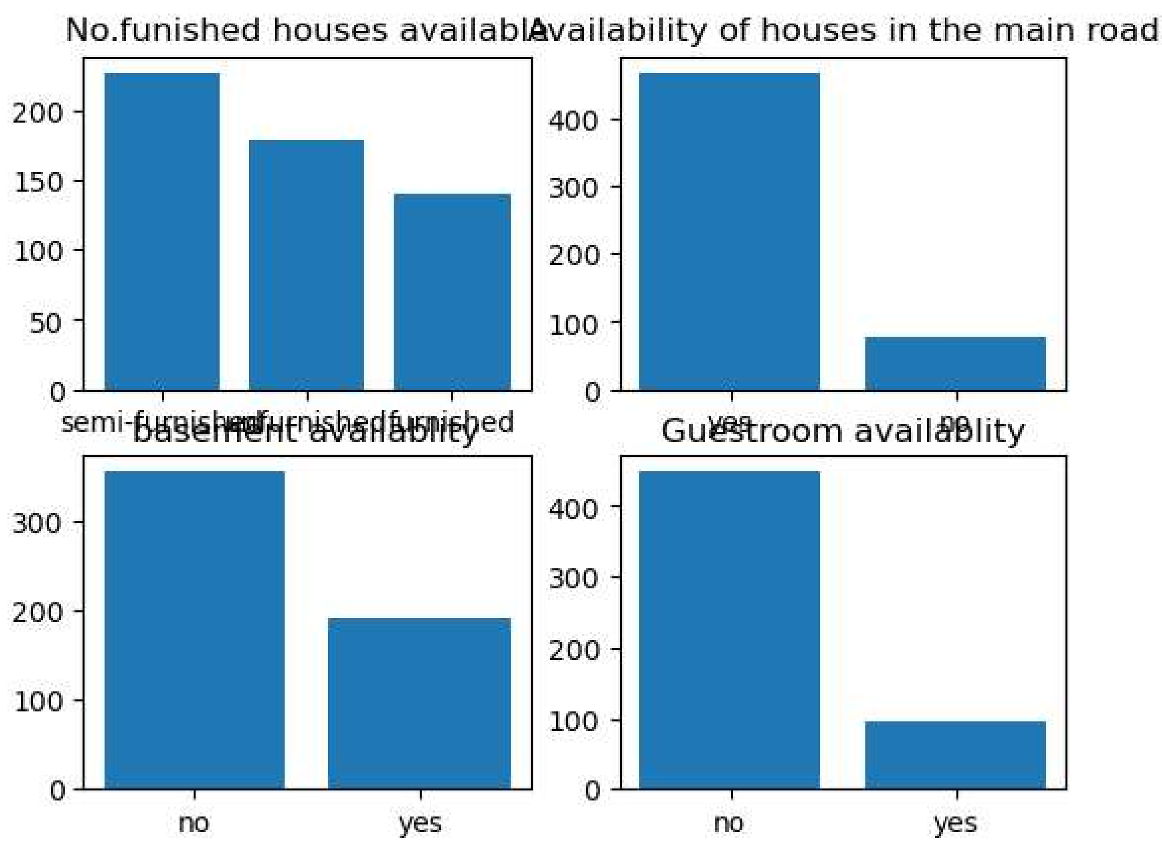
In [5]:

```
#PIE chart
q3a=house['furnishingstatus'].value_counts()
label=['furnished','semi-furnished','unfurnished']
q3a
plt.pie(q3a,labels=label)
plt.show()
```




```
Report:
    the graph details about the count of furnished, semifurnished and
unfurnished houses available to buy
```

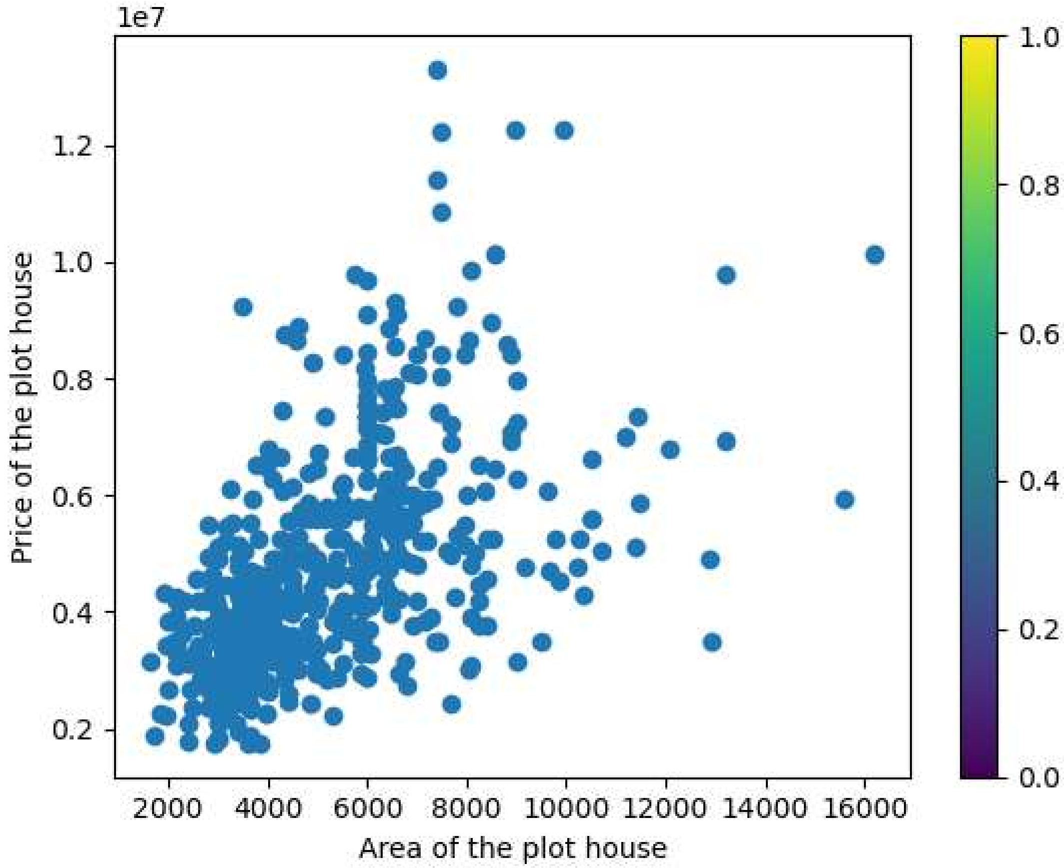
In [6]:
```
# BAR CHART
q3b1=house['mainroad'].value_counts()
q3b2=house['basement'].value_counts()
q3b3=house['guestroom'].value_counts()
fig,a =  plt.subplots(2,2)
a[0][0].bar(q3a.index,q3a.values)
a[0][0].set_title('No.funished houses available')
a[0][1].bar(q3b1.index,q3b1.values)
a[0][1].set_title('Availability of houses in the main road')
a[1][0].bar(q3b2.index,q3b2.values)
a[1][0].set_title('basement availablity')
a[1][1].bar(q3b3.index,q3b3.values)
a[1][1].set_title('Guestroom availablity')
plt.show()
```




```
Report:
    In these graph we find the number of available houses that are furnished,
with guest rooms, with basements,
    availability of main roads
```

Bi-Variate Analysis

In [7]:
```
#Scatter plot
plt.scatter(house['area'], house['price'])
plt.xlabel('Area of the plot house')
plt.ylabel('Price of the plot house')
plt.colorbar()
plt.show()
```



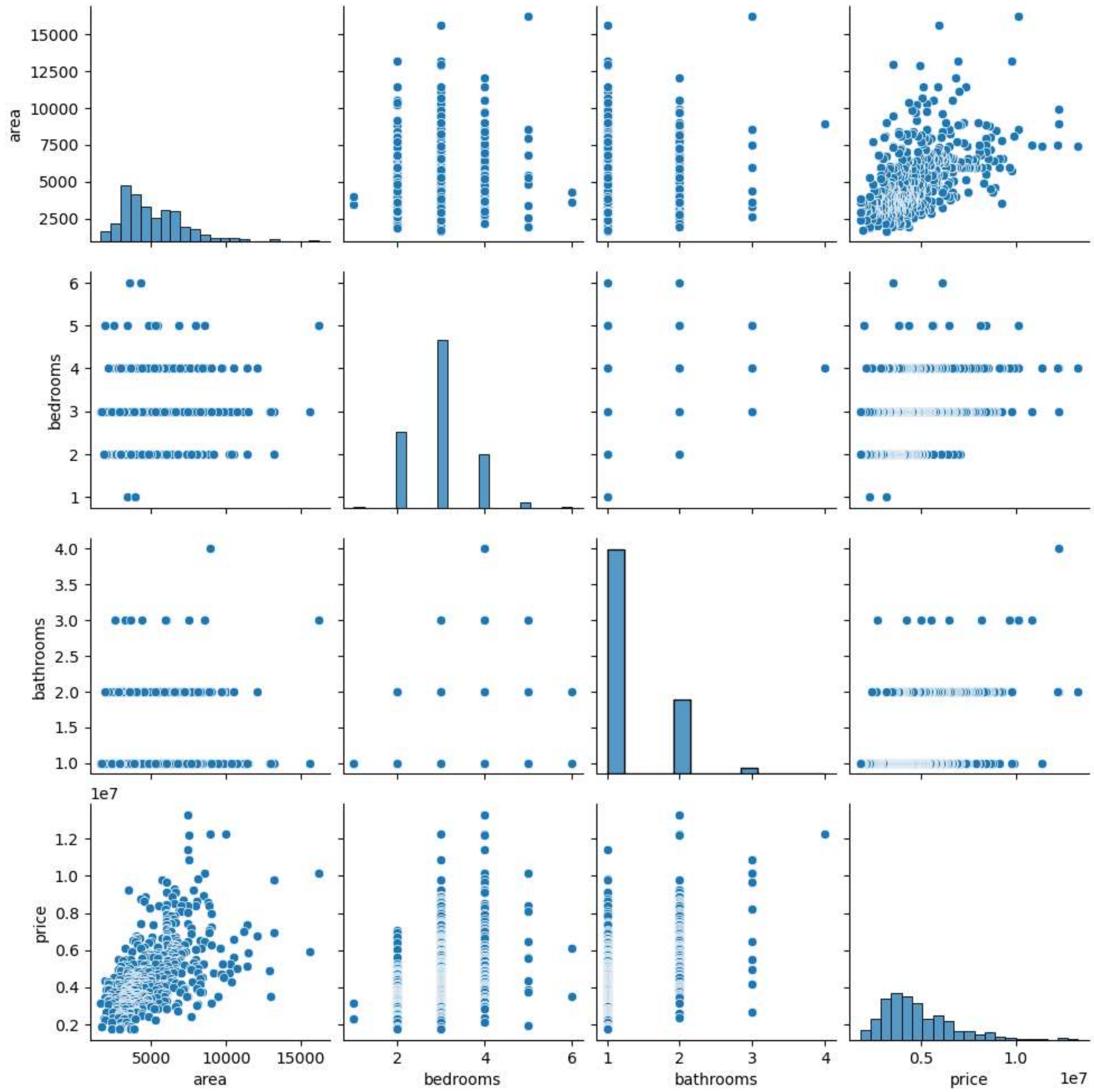

```
Report:
    the graph is lightly positively corelated and shows as the area increases
the price of the plot increases
```

Multi-Variate Analysis

In [8]:
```
import seaborn as sns
```

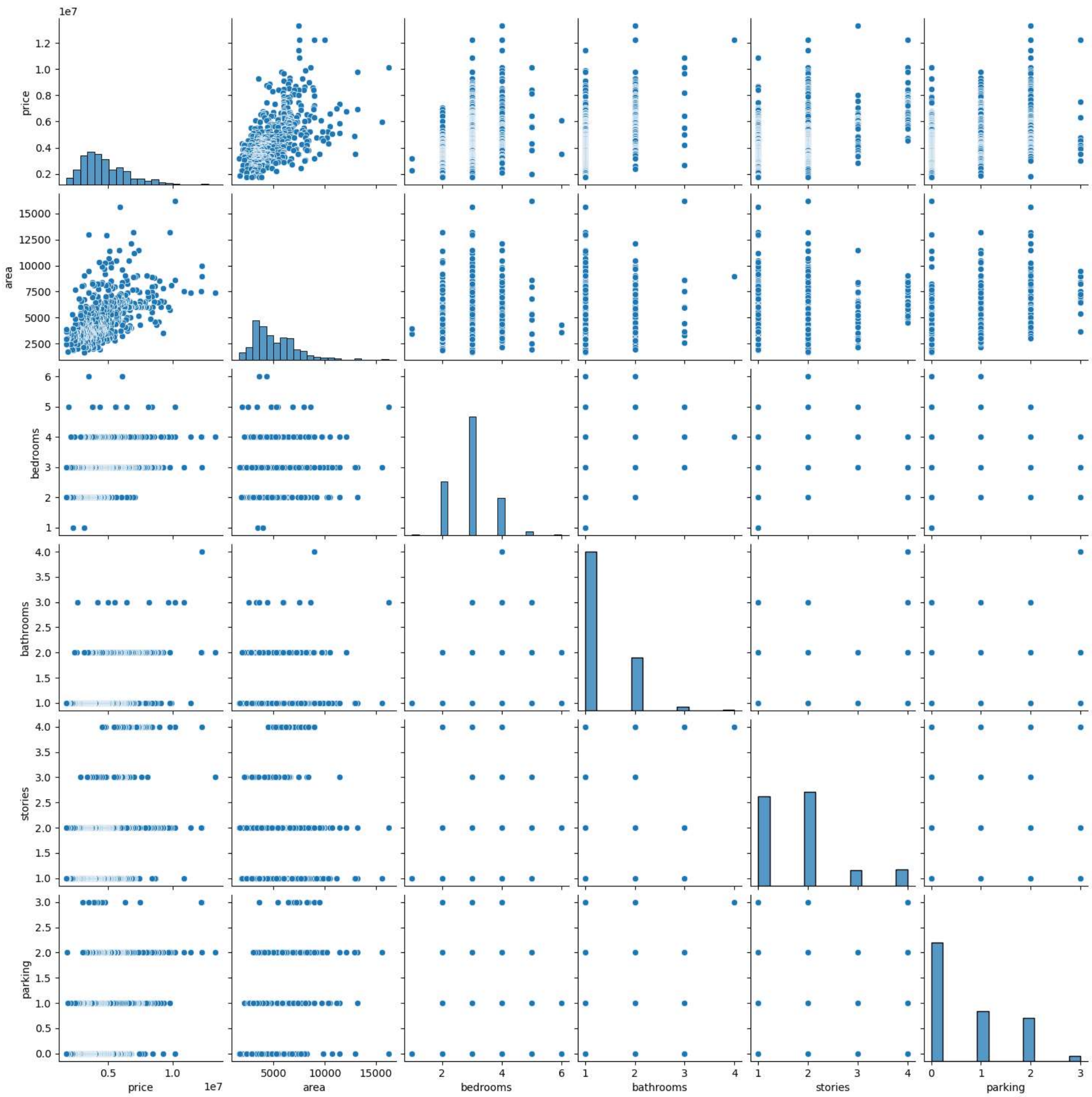
In [9]:
```
sns.pairplot(house[['area', 'bedrooms', 'bathrooms','price']])
```

Out[9]: <seaborn.axisgrid.PairGrid at 0x1fa0d3e9870>



In [10]:
```
sns.pairplot(house)
```

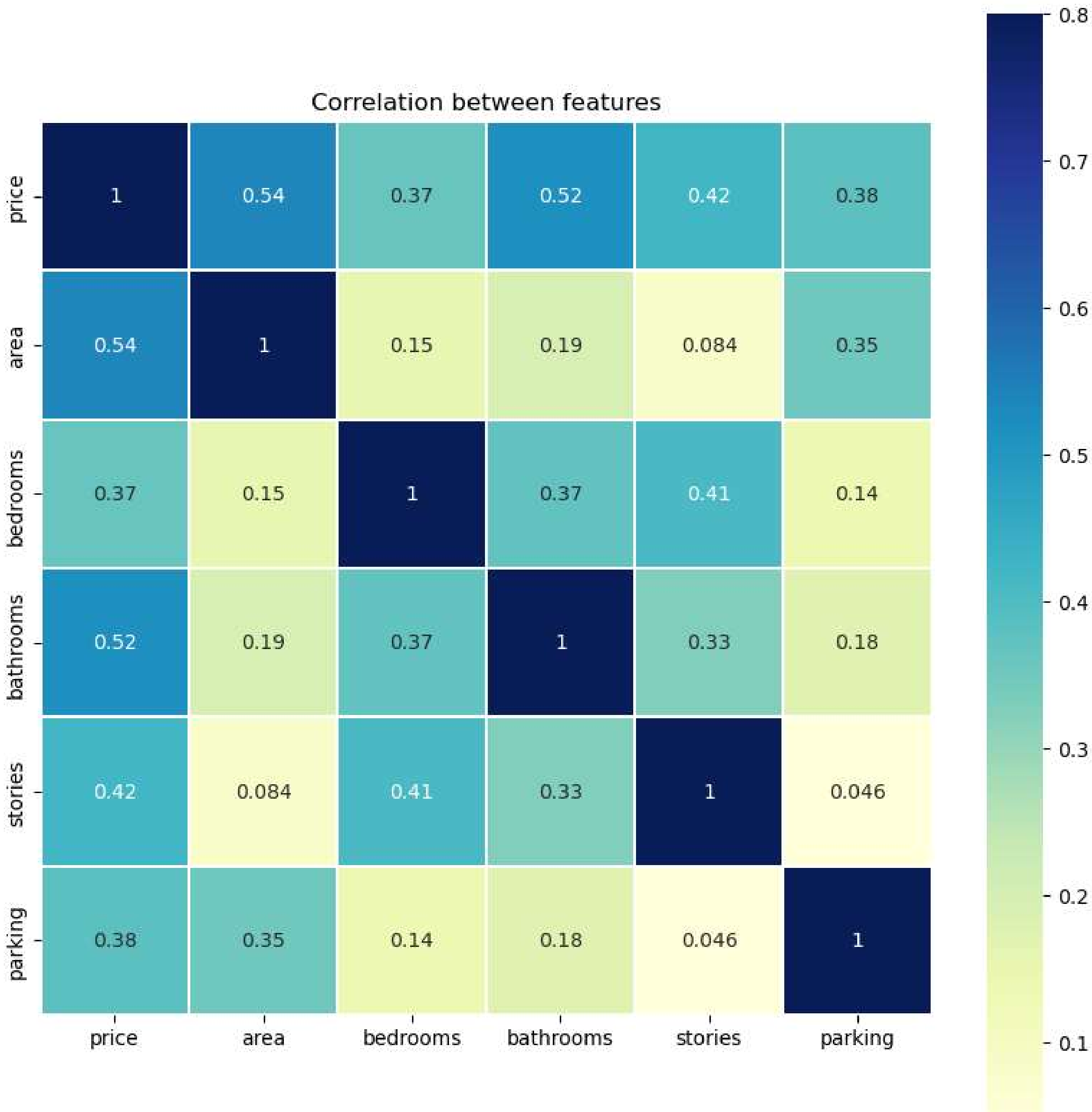
Out[10]: <seaborn.axisgrid.PairGrid at 0x1fa0e031810>



In [11]:
```
# heat plot
corr=house.corr()
plt.figure(figsize=(10, 10))
sns.heatmap(corr, vmax=.8, linewidths=0.01,
 square=True,annot=True,cmap='YlGnBu',linecolor="white")
plt.title('Correlation between features')
```

```
C:\Users\SURIYA\AppData\Local\Temp\ipykernel_1692\283143464.py:2: FutureWarni
ng: The default value of numeric_only in DataFrame.corr is deprecated. In a f
uture version, it will default to False. Select only valid columns or specify
the value of numeric_only to silence this warning.
  corr=house.corr()
```

Out[11]: Text(0.5, 1.0, 'Correlation between features')



4. Perform descriptive statistics on the dataset.

In [12]:
```
house.describe()
```

Out[12]:

	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

Finding correlation between area and price

In [13]:
```
house[['area','price']].corr()
```

Out[13]:

	area	price
area	1.000000	0.535997
price	0.535997	1.000000

```
Report:
    the two arugments as correaltion of 0.53
```

Finding correlation between bedrooms bathrooms stories price

In [14]:
```
print(house[['price', 'bedrooms','bathrooms','stories','parking']].corr())
```

```
              price  bedrooms  bathrooms   stories   parking
price      1.000000  0.366494   0.517545  0.420712  0.384394
bedrooms   0.366494  1.000000   0.373930  0.408564  0.139270
bathrooms  0.517545  0.373930   1.000000  0.326165  0.177496
stories    0.420712  0.408564   0.326165  1.000000  0.045547
parking    0.384394  0.139270   0.177496  0.045547  1.000000
```

5. Check for Missing values and deal with them.

In [15]:
```
house.isnull().sum()
```

Out[15]:
```
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 [16]:
```
print(house.interpolate(inplace=True))
```

```
None
```

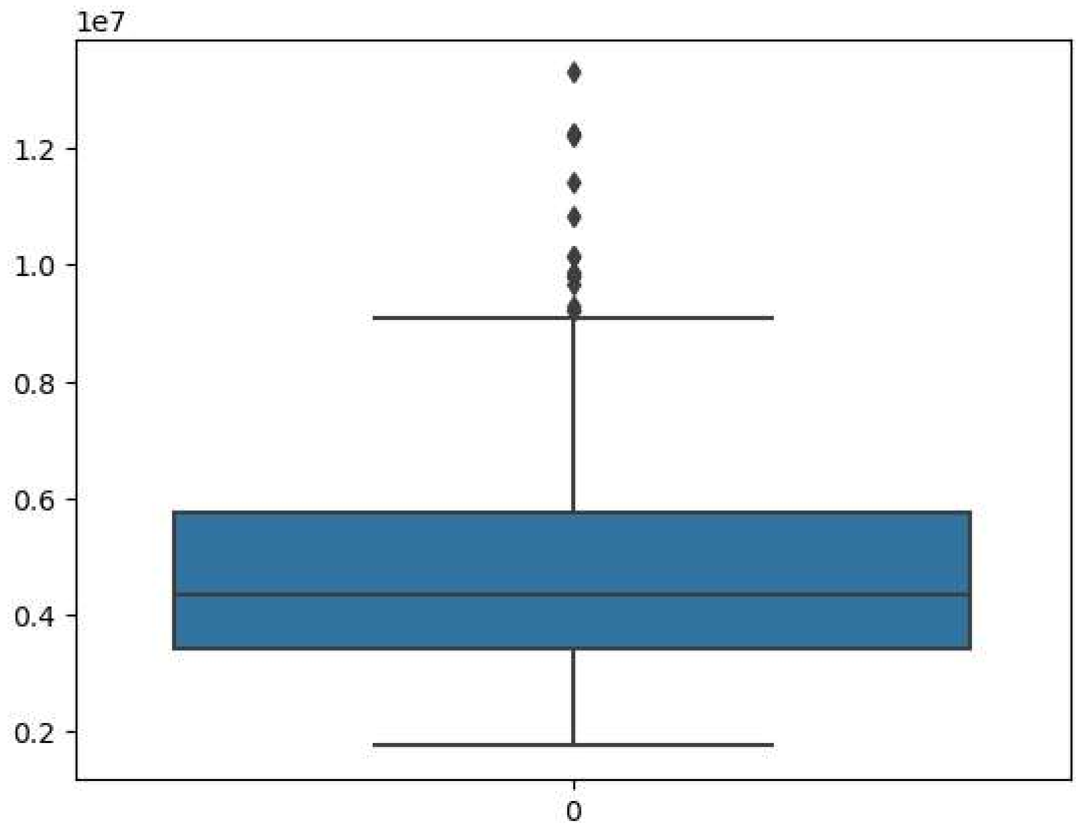
```
Report:
    The dataset has no null values
```

6. Find the outliers and replace them outliers

```
Finding outliers in the dataset
```

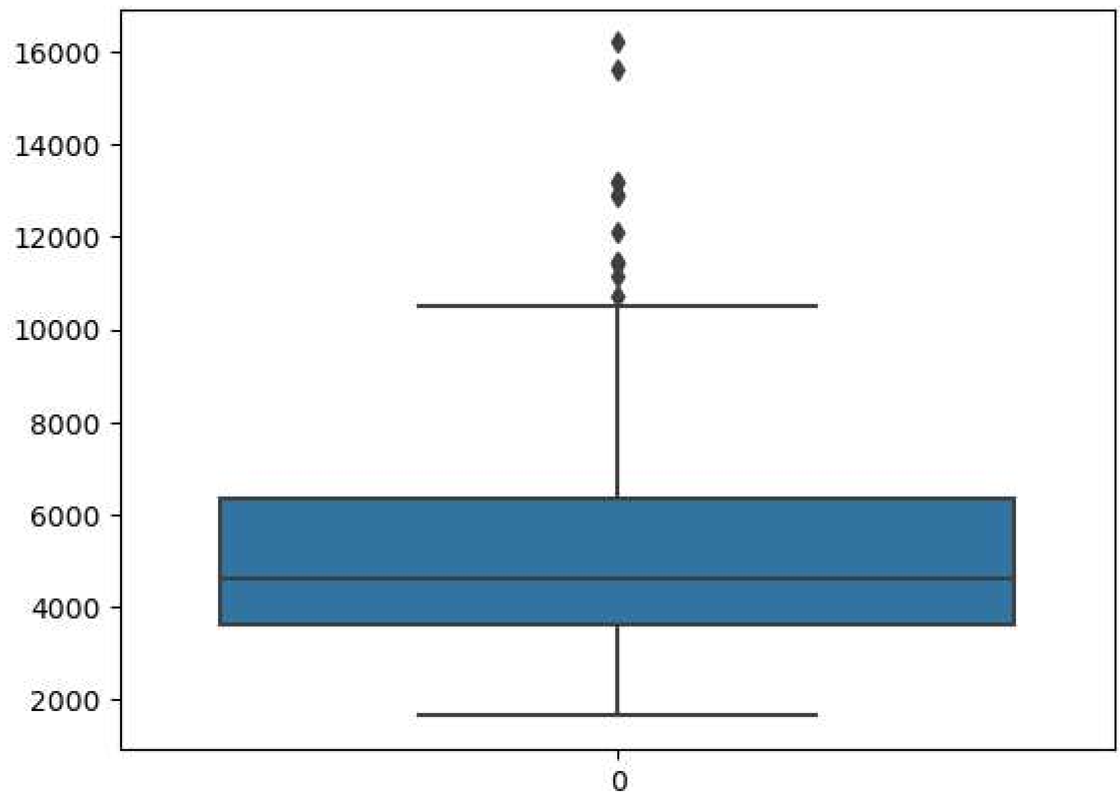
In [17]:
```
sns.boxplot(house['price'])
```

Out[17]: <Axes: >



In [18]:
```
sns.boxplot(house['area'])
```

Out[18]: <Axes: >




```
Report:
    So we have outliers on two numerical columns
```

In [19]:
```
# Fixing outliers on the price column
Q1 = house['price'].quantile(0.25)
Q3 = house['price'].quantile(0.75)
IQR = Q3 - Q1
whisker_width = 1.5
Fare_outliers = house[(house['price'] < Q1 - whisker_width*IQR) | (house['pric
Fare_outliers.head()
```

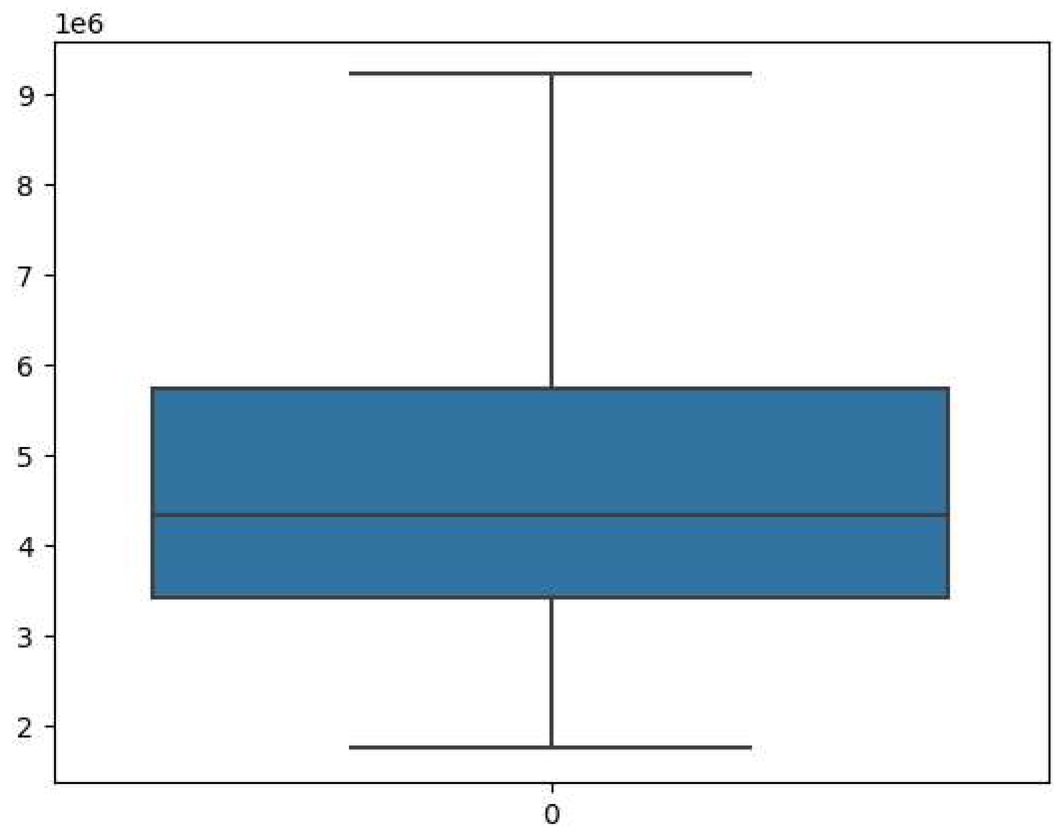
Out[19]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterhea
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	

In [20]:
```
lower_whisker = Q1 -(whisker_width*IQR)
upper_whisker = Q3 + (whisker_width*IQR)
house['price']=np.where(house['price']>upper_whisker,upper_whisker,np.where(hc
```

In [21]:
```
sns.boxplot(house['price'])
```

Out[21]: <Axes: >




```
Report:
    Now we have no outliers on the price column
```

In [22]:
```
# Fixing outliers on the price column
Q1 = house['area'].quantile(0.25)
Q3 = house['area'].quantile(0.75)
IQR = Q3 - Q1
whisker_width = 1.5
Fare_outliers = house[(house['area'] < Q1 - whisker_width*IQR) | (house['area'
Fare_outliers.head()
```

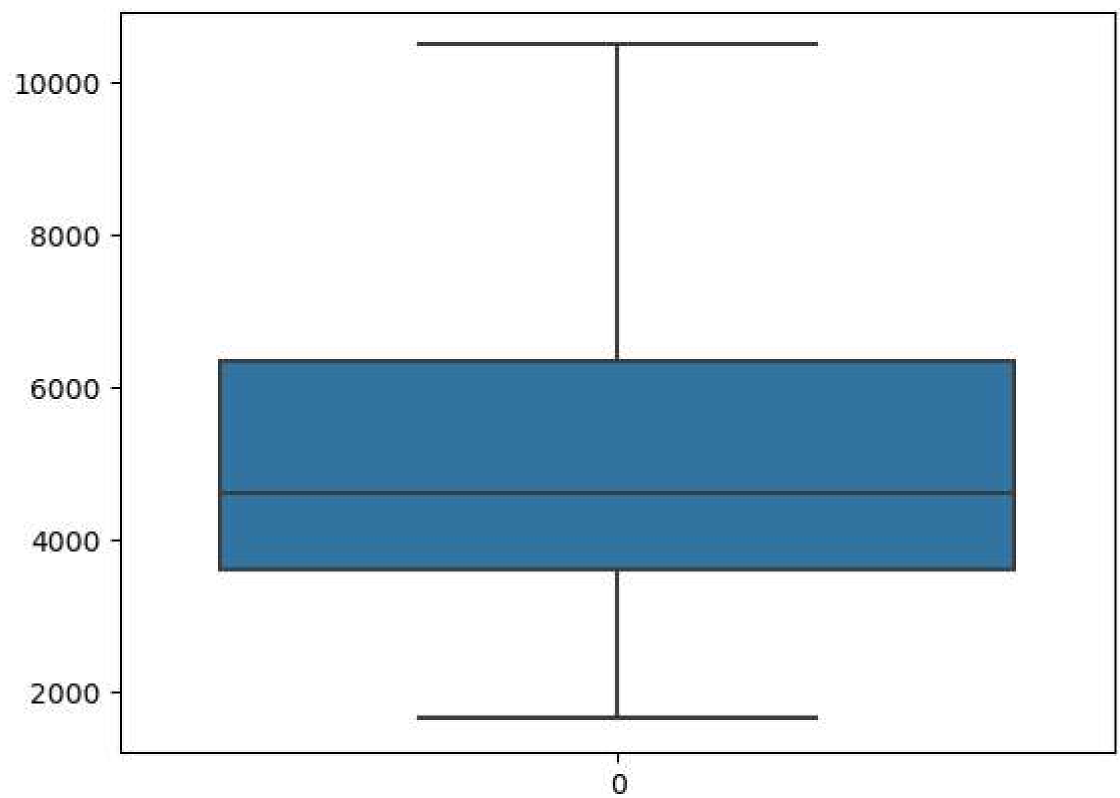
Out[22]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterhe
0	9205000.0	7420	4	2	3	yes	no	no	
1	9205000.0	8960	4	4	4	yes	no	no	
2	9205000.0	9960	3	2	2	yes	no	yes	
3	9205000.0	7500	4	2	2	yes	no	yes	
4	9205000.0	7420	4	1	2	yes	yes	yes	

In [23]:
```
lower_whisker = Q1 -(whisker_width*IQR)
upper_whisker = Q3 + (whisker_width*IQR)
house['area']=np.where(house['area']>upper_whisker,upper_whisker,np.where(hous
```

In [24]:
```
sns.boxplot(house['area'])
```

Out[24]: <Axes: >



Report:

```
Now we have no outliers on the area column
```

7. Check for Categorical columns and perform encoding.

In [25]:
```
cat_features=[i for i in house.columns if house.dtypes[i]=='object']
```

In [26]:
```
## one-hot encoding
house = pd.get_dummies(house, columns=cat_features, drop_first=True)
```

In [27]:
```
house
```

Out[27]:

	price	area	bedrooms	bathrooms	stories	parking	mainroad_yes	guestroom_yes
0	9205000.0	7420.0	4	2	3	2	1	0
1	9205000.0	8960.0	4	4	4	3	1	0
2	9205000.0	9960.0	3	2	2	2	1	0
3	9205000.0	7500.0	4	2	2	3	1	0
4	9205000.0	7420.0	4	1	2	2	1	1
...	...	...	...	...	...	...	...	...
540	1820000.0	3000.0	2	1	1	2	1	0
541	1767150.0	2400.0	3	1	1	0	0	0
542	1750000.0	3620.0	2	1	1	0	1	0
543	1750000.0	2910.0	3	1	1	0	0	0
544	1750000.0	3850.0	3	1	2	0	1	0

545 rows × 13 columns

```
Report:
    I treated all the categorical data with proper encoding
```

8. Split the data into dependent and independent variables.

In [28]:
```
# Dependent variable
y = house['price']

# Independent variables
X = house[['area', 'bedrooms', 'bathrooms','stories','mainroad_yes','guestroom
```

In [29]:
```
y
```

Out[29]:
```
0      9205000.0
1      9205000.0
2      9205000.0
3      9205000.0
4      9205000.0
         ...
540    1820000.0
541    1767150.0
542    1750000.0
543    1750000.0
544    1750000.0
Name: price, Length: 545, dtype: float64
```

In [30]:
```
X
```

Out[30]:

	area	bedrooms	bathrooms	stories	mainroad_yes	guestroom_yes	basement_yes	parki
0	7420.0	4	2	3	1	0	0	
1	8960.0	4	4	4	1	0	0	
2	9960.0	3	2	2	1	0	1	
3	7500.0	4	2	2	1	0	1	
4	7420.0	4	1	2	1	1	1	
...	...	...	...	...	...	...	...	
540	3000.0	2	1	1	1	0	1	
541	2400.0	3	1	1	0	0	0	
542	3620.0	2	1	1	1	0	0	
543	2910.0	3	1	1	0	0	0	
544	3850.0	3	1	2	1	0	0	

545 rows × 8 columns

9. Scale the independent variables

In [31]:
```
from sklearn.preprocessing import StandardScaler
```

In [32]:
```
# Initialize the scaler
scaler = StandardScaler()

# Scale the independent variables
X_scaled = scaler.fit_transform(X)
```

In [33]:
```
X_scaled
```

Out[33]:
```
array([[ 1.15658327,  1.40341936,  1.42181174, ..., -0.46531479,
        -0.73453933,  1.51769249],
       [ 1.92506041,  1.40341936,  5.40580863, ..., -0.46531479,
        -0.73453933,  2.67940935],
       [ 2.42407154,  0.04727831,  1.42181174, ..., -0.46531479,
         1.3613975 ,  1.51769249],
       ...,
       [-0.73965902, -1.30886273, -0.57018671, ..., -0.46531479,
        -0.73453933, -0.80574124],
       [-1.09395692,  0.04727831, -0.57018671, ..., -0.46531479,
        -0.73453933, -0.80574124],
       [-0.62488646,  0.04727831, -0.57018671, ..., -0.46531479,
        -0.73453933, -0.80574124]])
```

10. Split the data into training and testing

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

In [35]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2
```

In [36]:
```
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (436, 8)
Shape of X_test: (109, 8)
Shape of y_train: (436,)
Shape of y_test: (109,)
```

11. Build the Model

In [37]:
```
from sklearn.linear_model import LinearRegression
```

In [38]:
```
# Initialize the linear regression model
model = LinearRegression()
```

12. Train the Model

In [39]:
```
# Train the model on the training data
model.fit(X_train, y_train)
```

Out[39]:
```
▾ LinearRegression
LinearRegression()
```

13. Test the Model

In [40]:
```
# Make predictions on the testing data
y_pred = model.predict(X_test)
```

14. Measure the performance using Metrics

In [41]:
```
# mean squared error
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
mse
```

Out[41]:
```
1660592715275.1707
```

In [42]:
```
# r2_score
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
r2
```

Out[42]:
```
0.5889511982359401
```

In [43]:
```
#adjusted r2 score
n = X.shape[0]  # number of samples
p = X.shape[1]  # number of predictors
adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
adjusted_r2
```

Out[43]:
```
0.5828161414931929
```

In [44]:
```
#root mean squared error
```

In [45]:
```
import math
rmse=math.sqrt(mse)
rmse
```

Out[45]:
```
1288639.870279967
```

In [46]:
```
from sklearn.metrics import accuracy_score
```

In [48]:

```
def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    errors = abs(predictions - test_labels)
    mape = 100 * np.mean(errors / test_labels)
    accuracy = 100 - mape
    print('Model Performance')
    print('Average Error: {:0.4f} degrees.'.format(np.mean(errors)))
    print('Accuracy = {:0.2f}%.'.format(accuracy))
    return accuracy
base_accuracy = evaluate(model, X_test, y_test)
base_accuracy
```

```
Model Performance
Average Error: 1028378.7157 degrees.
Accuracy = 75.68%.
```

Out[48]:

```
75.67715682259981
```

In [ ]: