In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:

```
data=pd.read_csv('housing.csv')
data
```

Out[3]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwa
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

In [4]:

```
data.head()
```

Out[4]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwate
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 [5]:

```
data.tail()
```

Out[5]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwat
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	

In [6]:

```
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 [7]:

```
data['price']
```

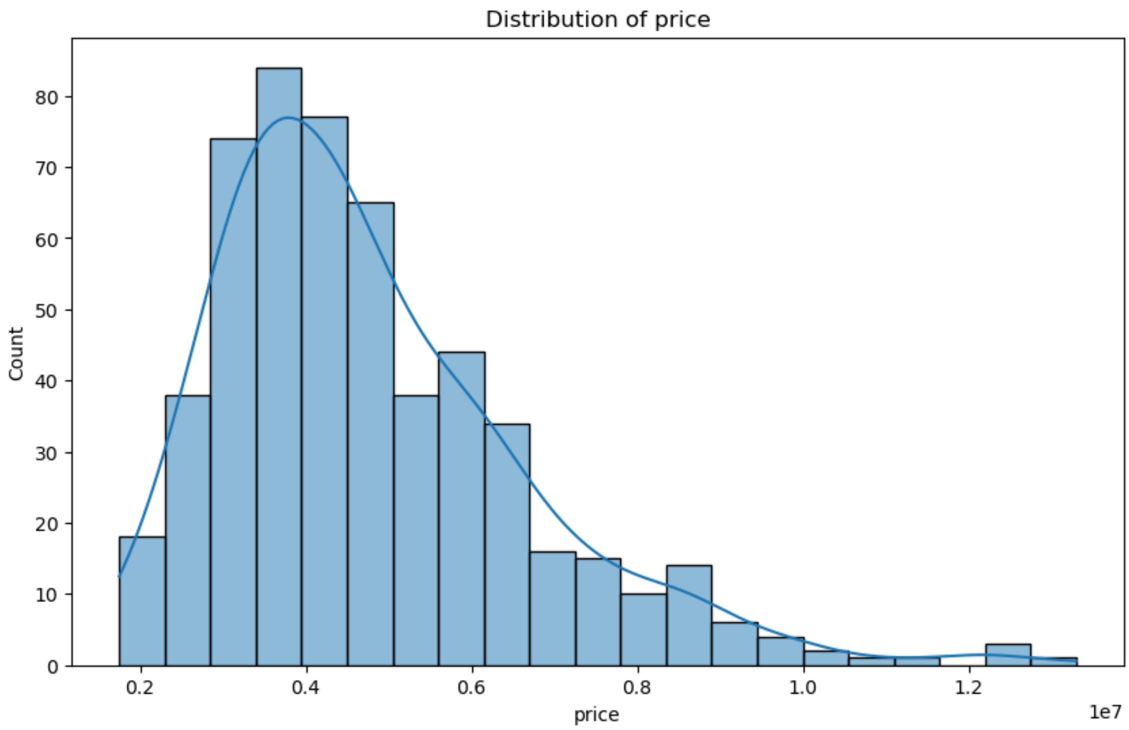
Out[7]:

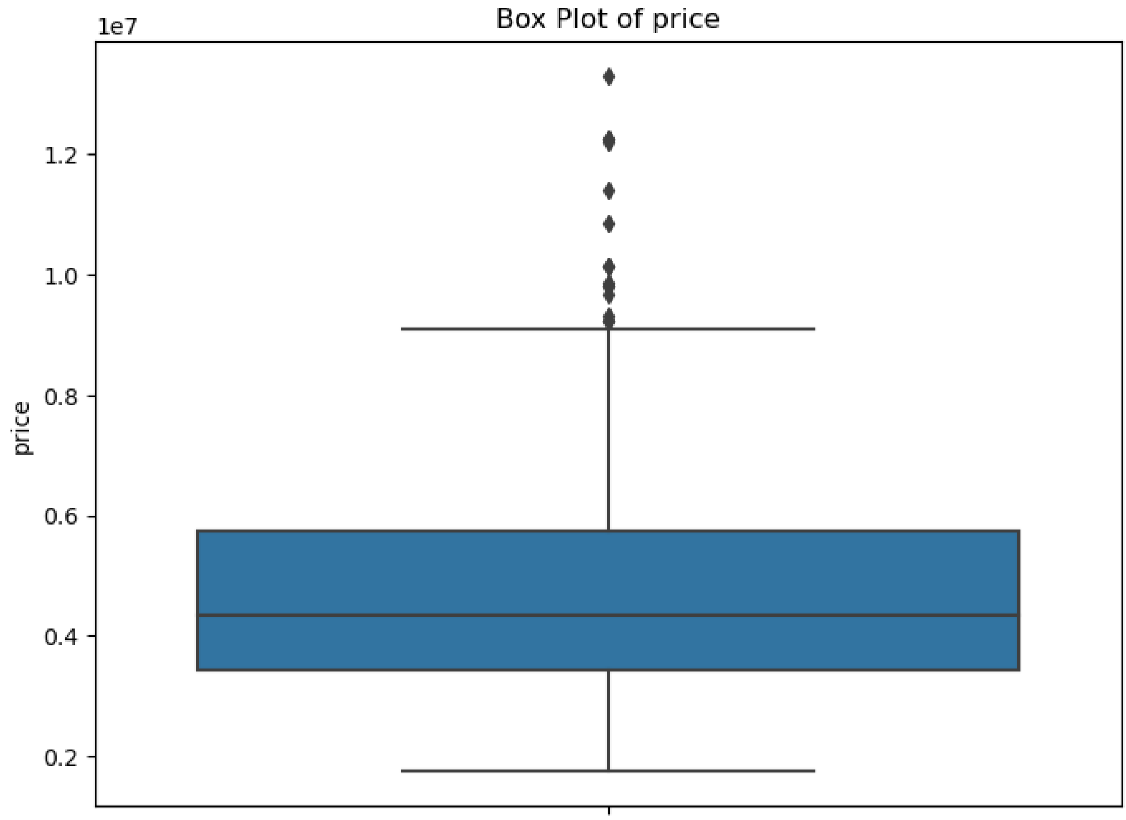
```
0      13300000
1      12250000
2      12250000
3      12215000
4      11410000
         ...   
540     1820000
541     1767150
542     1750000
543     1750000
544     1750000
Name: price, Length: 545, dtype: int64
```

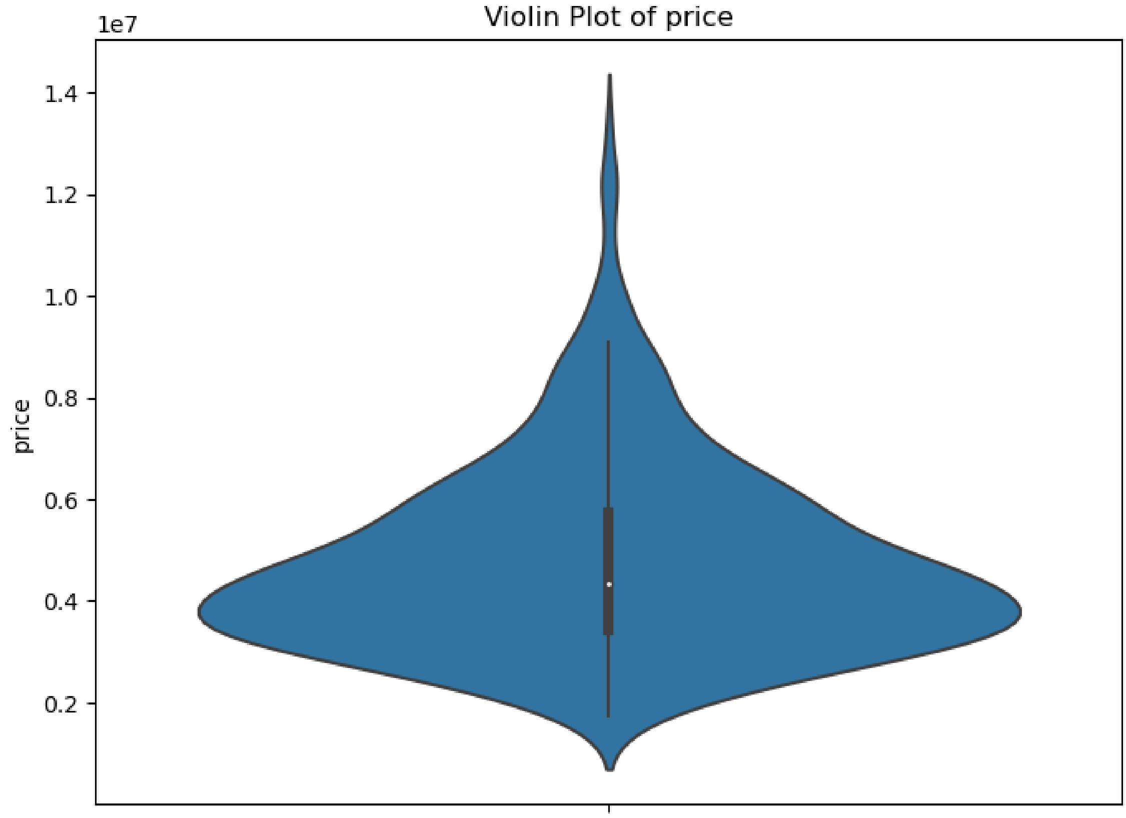
In [8]:

```
#Univariate analysis
variable = 'price'
print(data[variable].describe())
plt.figure(figsize=(10, 6))
sns.histplot(data=data, x=variable, kde=True)
plt.title('Distribution of ' + variable)
plt.show()
plt.figure(figsize=(8, 6))
sns.boxplot(data=data, y=variable)
plt.title('Box Plot of ' + variable)
plt.show()
plt.figure(figsize=(8, 6))
sns.violinplot(data=data, y=variable)
plt.title('Violin Plot of ' + variable)
plt.show()
```

```
count    5.450000e+02
mean     4.766729e+06
std      1.870440e+06
min      1.750000e+06
25%      3.430000e+06
50%      4.340000e+06
75%      5.740000e+06
max      1.330000e+07
Name: price, dtype: float64
```

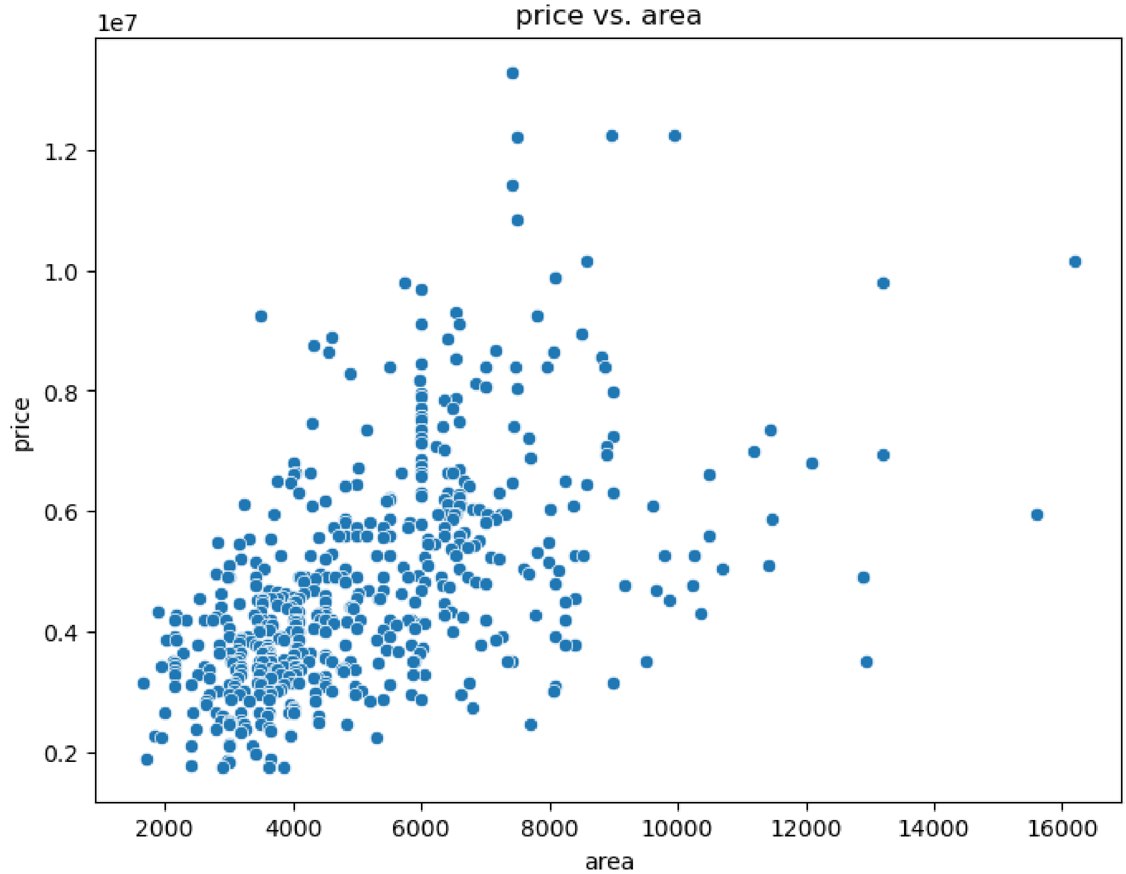



Box Plot of price
1e7
1.2
1.0
0.8
0.6
0.4
0.2
price


Violin Plot of price
1e7
1.4
1.2
1.0
0.8
0.6
0.4
0.2
price

In [9]:

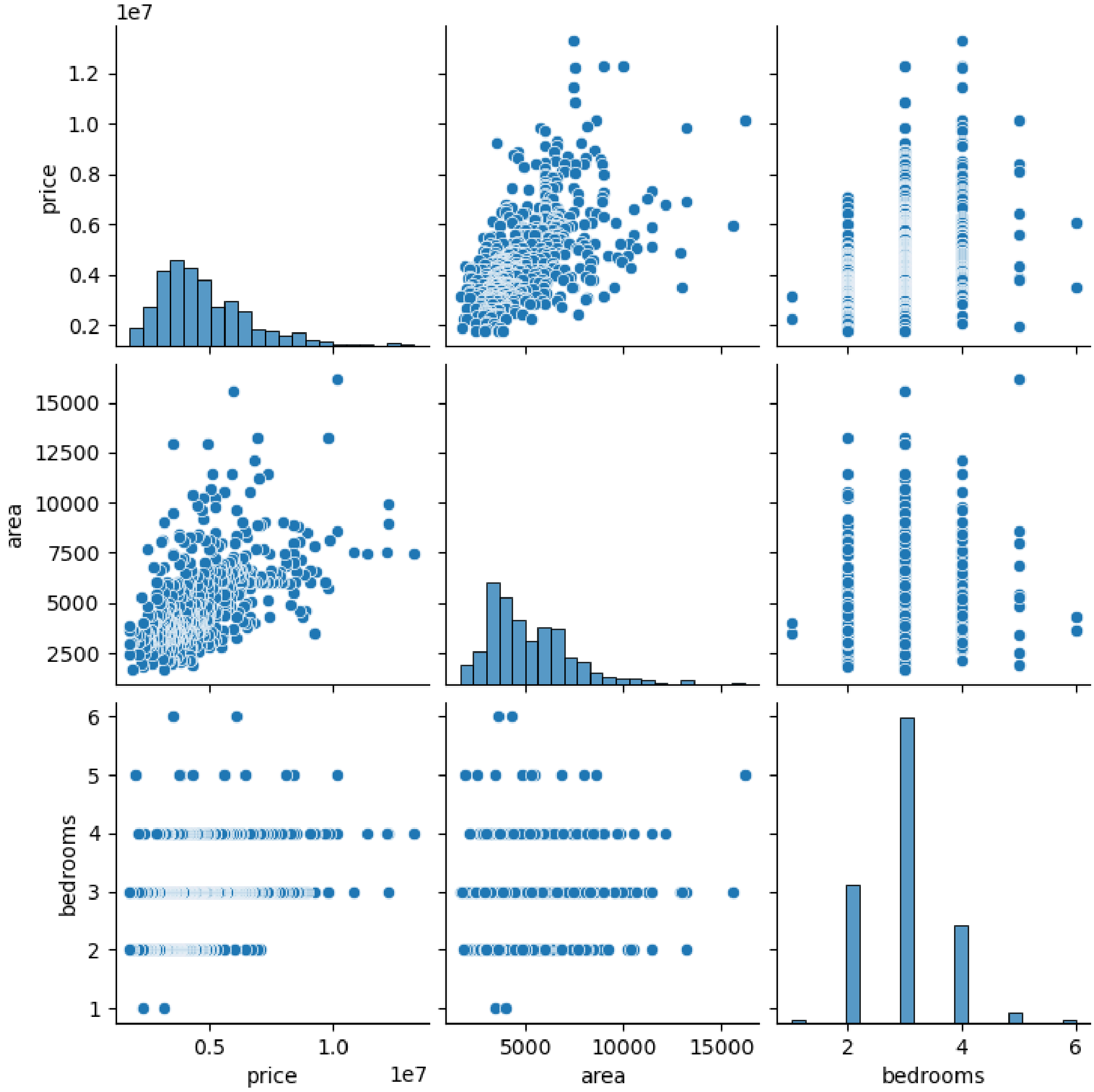
```
#Bivariate Analysis
variable1 = 'price'
variable2 = 'area'
plt.figure(figsize=(8, 6))
sns.scatterplot(data=data, x=variable2, y=variable1)
plt.title(variable1 + ' vs. ' + variable2)
plt.show()
correlation_coefficient = data[variable1].corr(data[variable2])
print('Correlation coefficient:', correlation_coefficient)
```



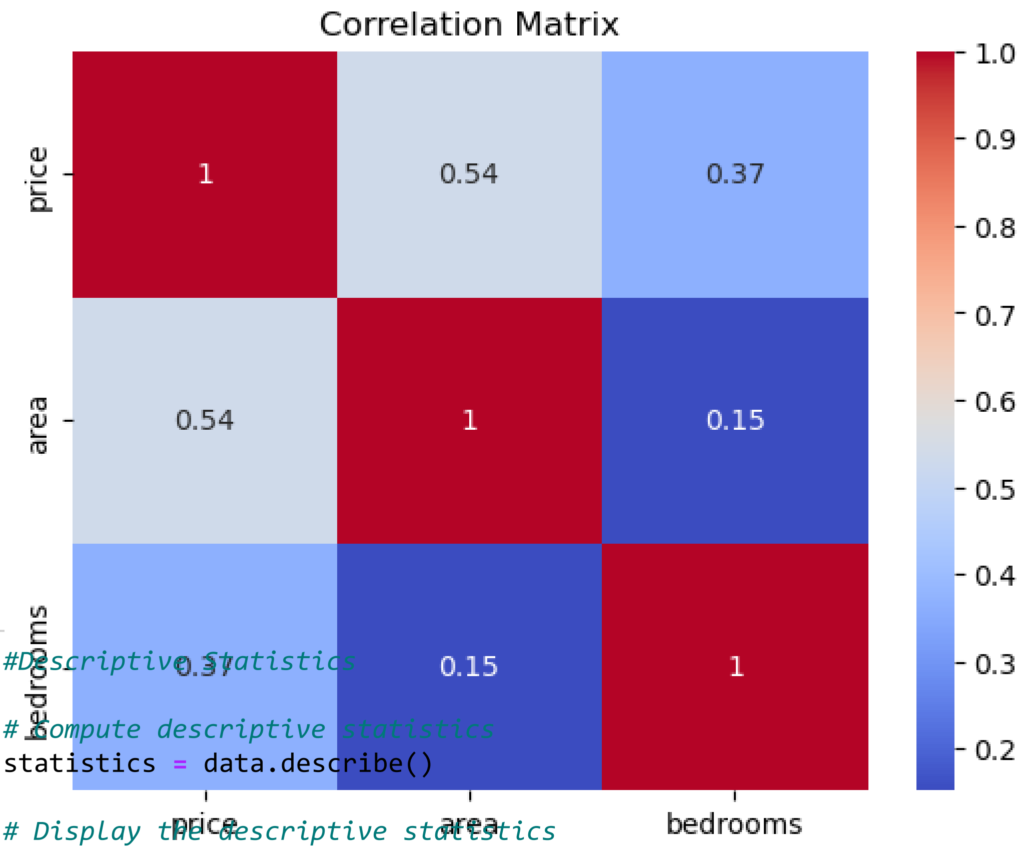

```
Correlation coefficient: 0.5359973457780794
```

In [10]:

```
#Multivariate analysis
variables = ['price', 'area', 'bedrooms']
sns.pairplot(data[variables])
plt.show()
correlation_matrix = data[variables].corr()
print(correlation_matrix)
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```




```
          price     area      bedrooms
price     1.000000  0.535997  0.366494
area      0.535997  1.000000  0.151858
bedrooms  0.366494  0.151858  1.000000
```




```python
#Descriptive Statistics

# Compute descriptive statistics
statistics = data.describe()

# Display the descriptive statistics
print(statistics)
```

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

          parking
count  545.000000
mean     0.693578
std      0.861586
min      0.000000
25%      0.000000
50%      0.000000
75%      1.000000
max      3.000000
```

In [12]:

```
#Descriptive Statistics
#Measures of central tendency
#Mean

data.mean()
```

```
C:\Users\vijay\AppData\Local\Temp\ipykernel_4932\3103449379.py:5: FutureWa
rning: The default value of numeric_only in DataFrame.mean is deprecated.
In a future version, it will default to False. In addition, specifying 'nu
meric_only=None' is deprecated. Select only valid columns or specify the v
alue of numeric_only to silence this warning.
  data.mean()
```

Out[12]:

```
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 [13]:

```
#Median
data.median()
```

```
C:\Users\vijay\AppData\Local\Temp\ipykernel_4932\1182471035.py:2: FutureWa
rning: The default value of numeric_only in DataFrame.median is deprecate
d. In a future version, it will default to False. In addition, specifying
'numeric_only=None' is deprecated. Select only valid columns or specify th
e value of numeric_only to silence this warning.
  data.median()
```

Out[13]:

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

In [14]:

```
#Median
data.median()
```

```
C:\Users\vijay\AppData\Local\Temp\ipykernel_4932\1182471035.py:2: FutureWa
rning: The default value of numeric_only in DataFrame.median is deprecate
d. In a future version, it will default to False. In addition, specifying
'numeric_only=None' is deprecated. Select only valid columns or specify th
e value of numeric_only to silence this warning.
  data.median()
```

Out[14]:

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

In [15]:

```
#Skewness
data.skew()
```

```
C:\Users\vijay\AppData\Local\Temp\ipykernel_4932\187154189.py:2: FutureWar
ning: The default value of numeric_only in DataFrame.skew is deprecated. I
n a future version, it will default to False. In addition, specifying 'num
eric_only=None' is deprecated. Select only valid columns or specify the va
lue of numeric_only to silence this warning.
  data.skew()
```

Out[15]:

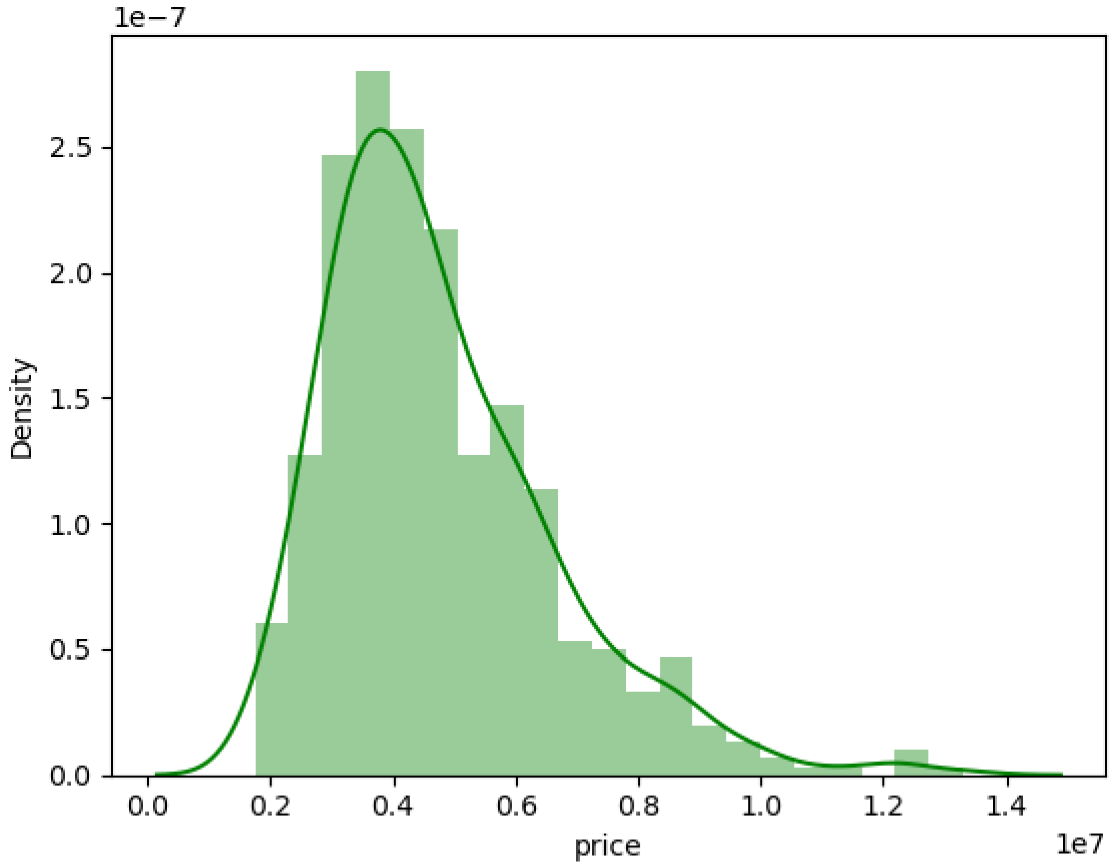
```
price        1.212239
area         1.321188
bedrooms     0.495684
bathrooms    1.589264
stories      1.082088
parking      0.842062
dtype: float64
```

In [16]:

```
# Distplot
print(sns.distplot(data['price'],color='green'))
```

```
C:\Users\vijay\AppData\Local\Temp\ipykernel_4932\3860952351.py:2: UserWarn
ing:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://
gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  print(sns.distplot(data['price'],color='green'))
```

```
Axes(0.125,0.11;0.775x0.77)
```



In [17]:

```
print(sns.distplot(df['area'],color='blue'))
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
Cell In[17], line 1
----> 1 print(sns.distplot(df['area'],color='blue'))

NameError: name 'df' is not defined
```

In [18]:

```
# kurtosis
data.kurt()
```

```
C:\Users\vijay\AppData\Local\Temp\ipykernel_4932\2667652193.py:2: FutureWa
rning: The default value of numeric_only in DataFrame.kurt is deprecated.
In a future version, it will default to False. In addition, specifying 'nu
meric_only=None' is deprecated. Select only valid columns or specify the v
alue of numeric_only to silence this warning.
  data.kurt()
```

Out[18]:

```
price        1.960130
area         2.751480
bedrooms     0.728323
bathrooms    2.164856
stories      0.679404
parking     -0.573063
dtype: float64
```

In [19]:

```
#Range

data.max()
```

Out[19]:

```
price                  13300000
area                      16200
bedrooms                      6
bathrooms                     4
stories                       4
mainroad                    yes
guestroom                   yes
basement                    yes
hotwaterheating             yes
airconditioning             yes
parking                       3
furnishingstatus    unfurnished
dtype: object
```

In [20]:

```
data.min()
```

Out[20]:

```
price                 1750000
area                     1650
bedrooms                    1
bathrooms                   1
stories                     1
mainroad                   no
guestroom                  no
basement                   no
hotwaterheating            no
airconditioning            no
parking                     0
furnishingstatus    furnished
dtype: object
```

In [21]:

```
column = 'price'

# Find the range
column_range = df[column].max() - df[column].min()

# Print the range
print(f"The range for '{column}' is: {column_range}")
```

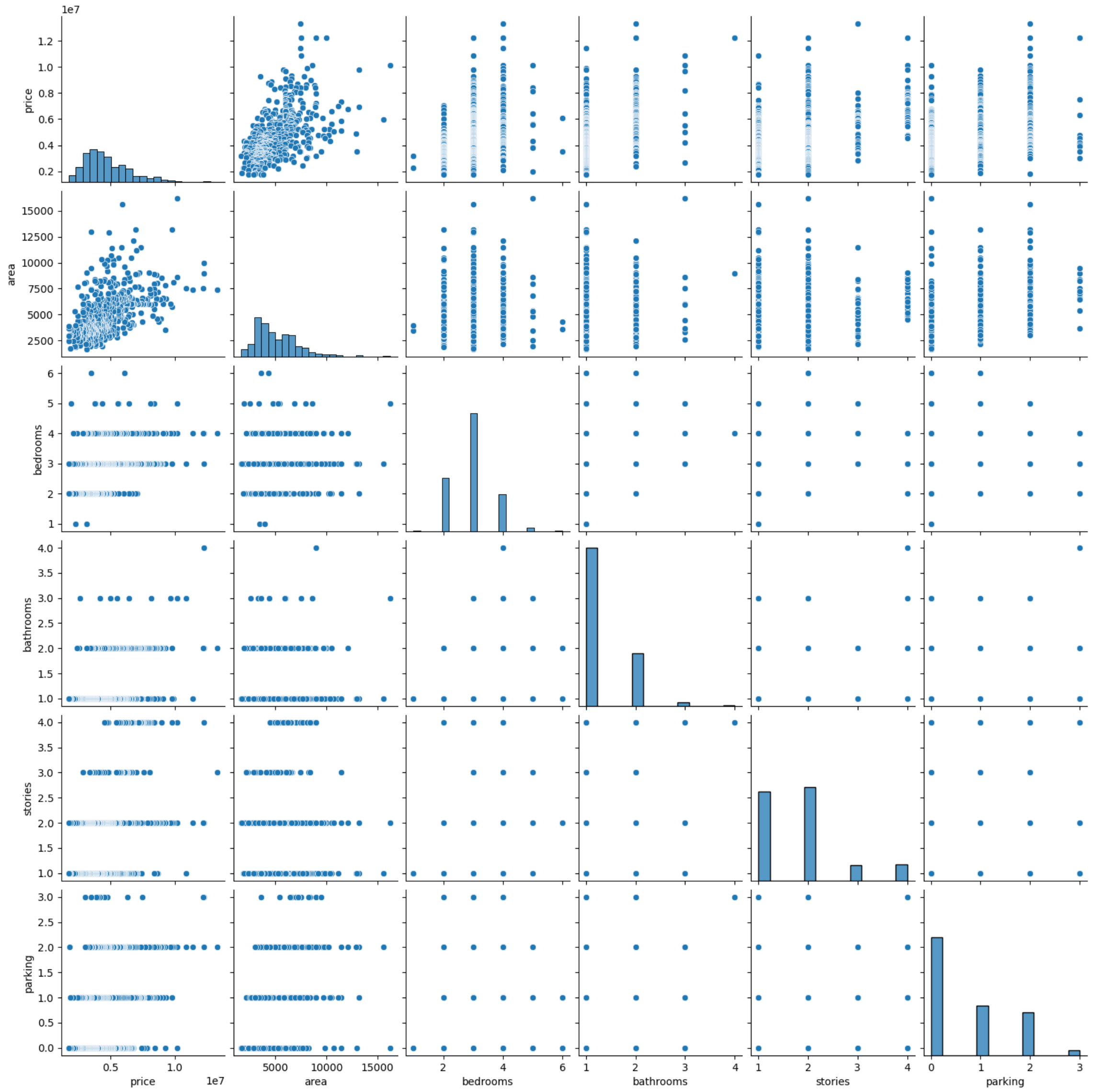
```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
Cell In[21], line 4
      1 column = 'price'
      3 # Find the range
----> 4 column_range = df[column].max() - df[column].min()
      6 # Print the range
      7 print(f"The range for '{column}' is: {column_range}")

NameError: name 'df' is not defined
```

In [22]:

```
sns.pairplot(data)
```

Out[22]:

<seaborn.axisgrid.PairGrid at 0x26e44f0a980>



In [23]:

```
#Handling the Missing values

# Check for missing values
print(df.isnull().sum())

# Handling missing values for numerical columns
df['price'].fillna(df['price'].median(), inplace=True)
df['area'].fillna(df['area'].mean(), inplace=True)


# Dropping rows with missing values
#data.dropna(inplace=True)

# Verify if missing values are handled
print(df.isnull().sum())
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
Cell In[23], line 4
      1 #Handling the Missing values
      2 
      3 # Check for missing values
----> 4 print(df.isnull().sum())
      6 # Handling missing values for numerical columns
      7 df['price'].fillna(df['price'].median(), inplace=True)

NameError: name 'df' is not defined
```

In [ ]:

```
# Find the outliers and replace the outliers

# Identify outliers in numerical columns
numeric_vars = ['price', 'area']

for var in numeric_vars:
    # Calculate the IQR (Interquartile Range)
    Q1 = df[var].quantile(0.25)
    Q3 = df[var].quantile(0.75)
    IQR = Q3 - Q1

    # Determine the upper and lower bounds for outliers
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identify outliers
    outliers = df[(df[var] < lower_bound) | (df[var] > upper_bound)]

    # Replace outliers with appropriate values
    df[var] = np.where((df[var] < lower_bound) | (df[var] > upper_bound), df[var].median

# Verify if outliers are replaced
for var in numeric_vars:
    # Calculate the IQR (Interquartile Range)
    Q1 = df[var].quantile(0.25)
    Q3 = df[var].quantile(0.75)
    IQR = Q3 - Q1

    # Determine the upper and lower bounds for outliers
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identify outliers
    outliers = df[(df[var] < lower_bound) | (df[var] > upper_bound)]

    # Print the outliers (should be empty if outliers are replaced)
    print(f"Outliers in '{var}':")
    print(outliers)
```

In [ ]:

```
#Check for Categorical columns and perform encoding.


# Check for categorical columns
categorical_cols = data.select_dtypes(include=['object']).columns
print("Categorical columns:")
print(categorical_cols)

# Perform categorical encoding
#Label Encoding
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
for col in categorical_cols:
    data[col] = label_encoder.fit_transform(df[col])

#One-Hot Encoding
df = pd.get_dummies(data, columns=categorical_cols, drop_first=True)


# Display the encoded dataset
print(df.head())
```

In [ ]:

```
dependent_variable = 'price'
independent_variables = ['area', 'bedrooms', 'bathrooms']
x = data[independent_variables]  # Independent variables
y = data[dependent_variable]  # Dependent variable
print(x.head())
print(y.head())
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=42
```

In [ ]:

```
#Scale the independent variables

from sklearn.preprocessing import StandardScaler

# Split the data into dependent and independent variables
X = df.drop('bedrooms', axis=1)  # Independent variables (features)
y = df['bedrooms']  # Dependent variable (target)

# Create a StandardScaler object
scaler = StandardScaler()

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

# Convert the scaled variables back to a DataFrame (optional)
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

# Display the scaled variables
print(X_scaled.head())
```

In [ ]:

```
from sklearn.model_selection import train_test_split

# Split the data into dependent and independent variables
X = df.drop('bedrooms', axis=1)  # Independent variables (features)
y = df['bedrooms']  # Dependent variable (target)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42

# Display the shapes of the training and testing sets
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)
```

In [ ]:

```
#build the model
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

dependent_variable = 'price'
independent_variables = ['area', 'bedrooms', 'bathrooms']
X = data[independent_variables]
y = data[dependent_variable]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error:', mse)
coefficients = model.coef_
intercept = model.intercept_
print('Coefficients:', coefficients)
print('Intercept:', intercept)
```

In [ ]:

```
# Train the Model
dependent_variable = 'price'
independent_variables = ['area', 'bedrooms', 'bathrooms']
X = data[independent_variables]
y = data[dependent_variable]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
model = LinearRegression()
model.fit(X_train, y_train)
y_train_pred = model.predict(X_train)
mse_train = mean_squared_error(y_train, y_train_pred)
print('Training Mean Squared Error:', mse_train)

y_test_pred = model.predict(X_test)
mse_test = mean_squared_error(y_test, y_test_pred)
print('Testing Mean Squared Error:', mse_test)
```

In [ ]:

```
#Test the Model
dependent_variable = 'price'
independent_variables = ['area', 'bedrooms', 'bathrooms']
X_train = data[independent_variables]
y_train = data[dependent_variable]
model = LinearRegression()
model.fit(X_train, y_train)
test_data = pd.read_csv('housing.csv')  # Load the test dataset
X_test = test_data[independent_variables]
y_test = test_data[dependent_variable]
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error:', mse)
```

In [ ]:

```
#Measure the performance using Metrics.

#Mean squared error
from sklearn.metrics import mean_squared_error

mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error:', mse)

#Root mean squared error
from sklearn.metrics import mean_squared_error

rmse = mean_squared_error(y_test, y_pred, squared=False)
print('Root Mean Squared Error:', rmse)

#Mean absolute error
from sklearn.metrics import mean_absolute_error

mae = mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error:', mae)


#R-Squared error
from sklearn.metrics import r2_score

r2 = r2_score(y_test, y_pred)
print('R-squared Score:', r2)
```

In [ ]:

```

```