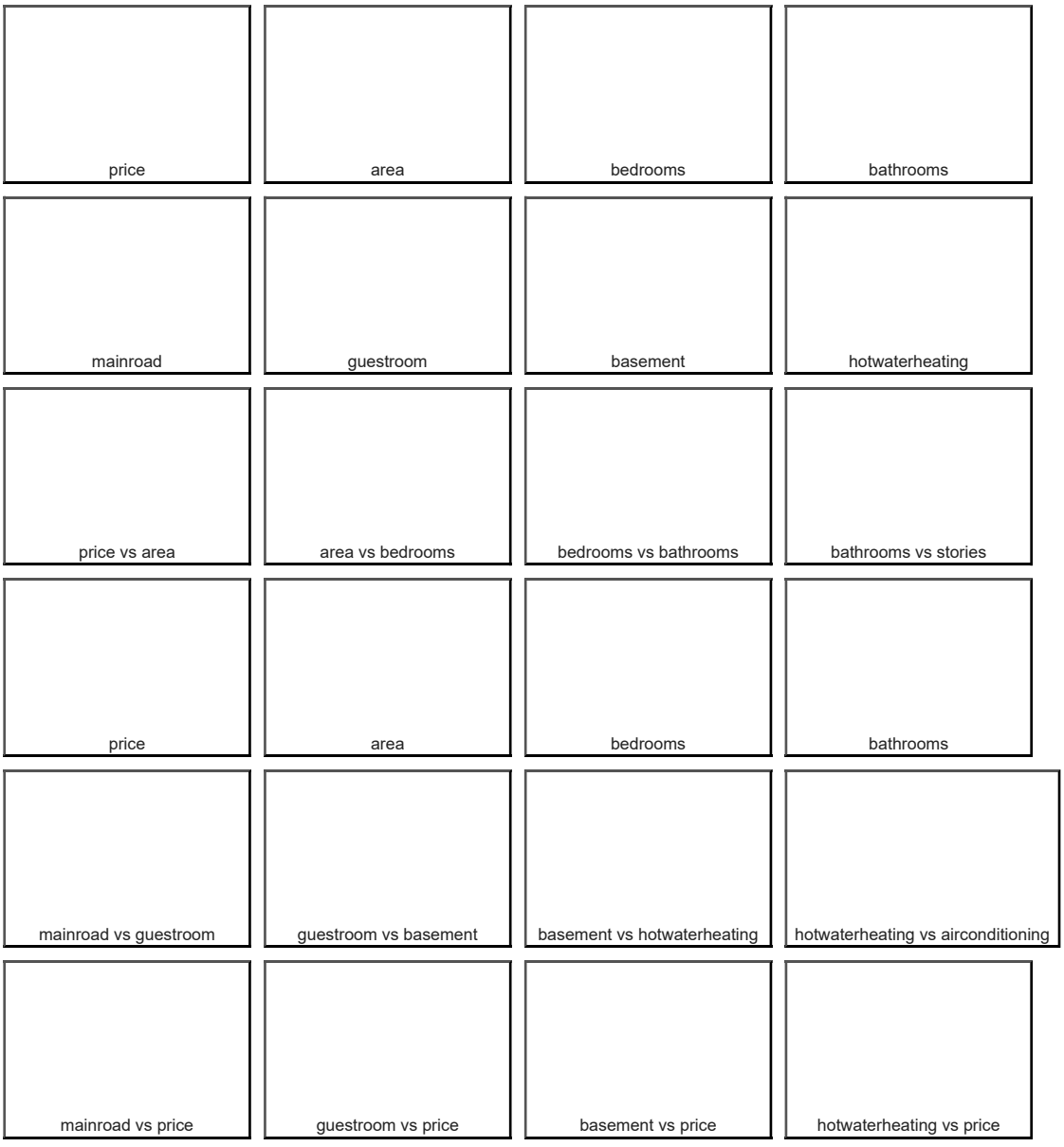
```
import numpy as np   # Importing the numpy library for numerical computations
import pandas as pd  # Importing the pandas library for data manipulation and analysis
import matplotlib.pyplot as plt  # Importing the matplotlib library for plotting graphs
import seaborn as sns  # Importing the seaborn library for enhanced data visualization
from sklearn.model_selection import train_test_split # Importing the train_test_split function from sklearn.model_selection module
from sklearn.preprocessing import MinMaxScaler # Import the MinMaxScaler from sklearn.preprocessing module
from sklearn.linear_model import LinearRegression  # Importing the LinearRegression model from scikit-learn
from sklearn.metrics import r2_score  # Importing the r2_score function from the sklearn.metrics module
```

```
data = pd.read_csv('/content/Housing (1).csv')  # Reading a CSV file and storing the data in a pandas DataFrame called 'data'
data.head()  # Di
```

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

Next steps: Generate code with `data` View recommended plots




```
data.tail(5)
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwate
10	1820000	3000	2	1	1	yes	no	yes	
11	1767150	2400	3	1	1	no	no	no	
12	1750000	3620	2	1	1	yes	no	no	
13	1750000	2910	3	1	1	no	no	no	
14	1750000	3850	3	1	2	yes	no	no	

```
print("Rows and Columns of the dataset :- ",data.shape)
```

```
Rows and Columns of the dataset :-  (545, 13)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 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  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

```
data.columns
```

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

```
data.describe(include ='all')
```

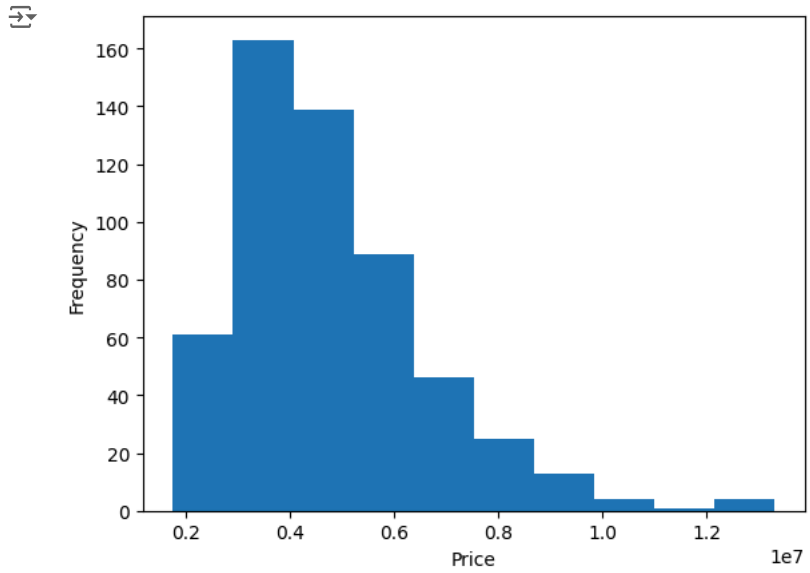
	price	area	bedrooms	bathrooms	stories	mainroad	guest
count	5.450000e+02	545.000000	545.000000	545.000000	545.000000	545	
unique	NaN	NaN	NaN	NaN	NaN	2	
top	NaN	NaN	NaN	NaN	NaN	yes	
freq	NaN	NaN	NaN	NaN	NaN	468	
mean	4.766729e+06	5150.541284	2.965138	1.286239	1.805505	NaN	
std	1.870440e+06	2170.141023	0.738064	0.502470	0.867492	NaN	
min	1.750000e+06	1650.000000	1.000000	1.000000	1.000000	NaN	
25%	3.430000e+06	3600.000000	2.000000	1.000000	1.000000	NaN	
50%	4.340000e+06	4600.000000	3.000000	1.000000	2.000000	NaN	
75%	5.740000e+06	6360.000000	3.000000	2.000000	2.000000	NaN	
max	1.330000e+07	16200.000000	6.000000	4.000000	4.000000	NaN	

```
data.isnull().sum()
```

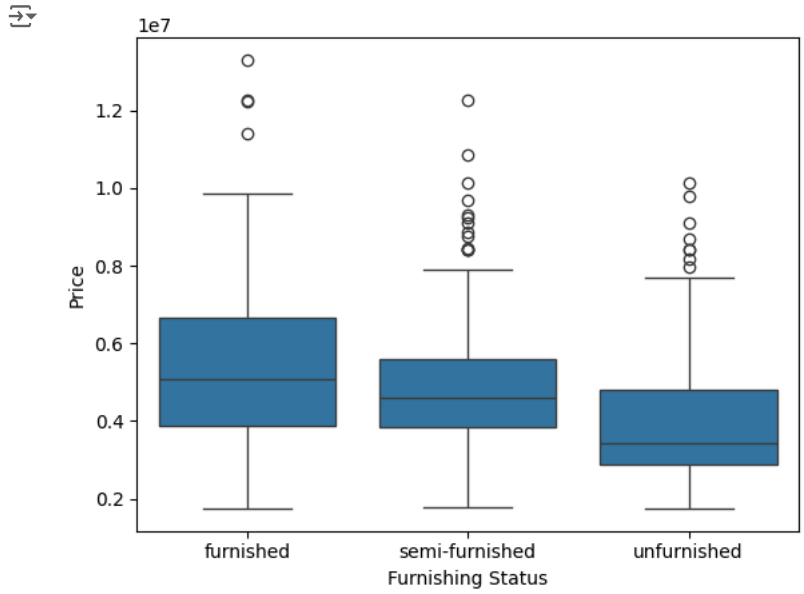
```
price              0
area               0
bedrooms           0
bathrooms          0
stories            0
mainroad           0
guestroom          0
basement           0
hotwaterheating    0
airconditioning    0
```

```
parking           0
prefarea          0
furnishingstatus  0
dtype: int64
```

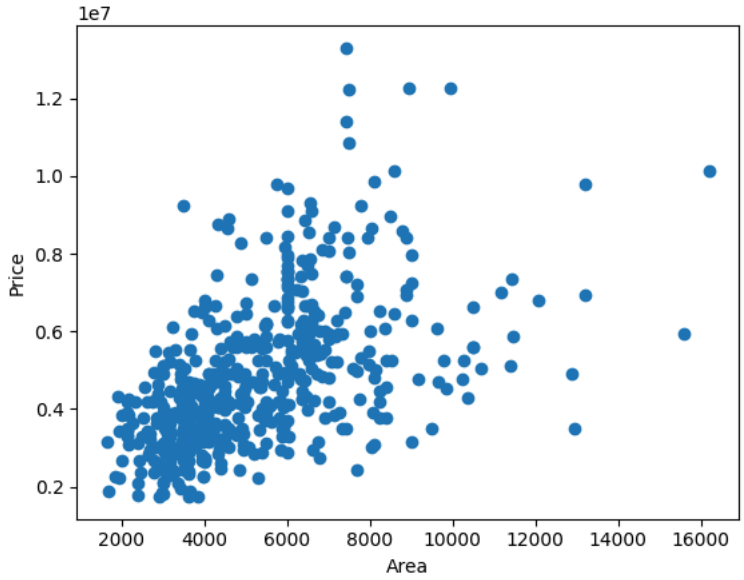
```
plt.hist(data['price'])
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```



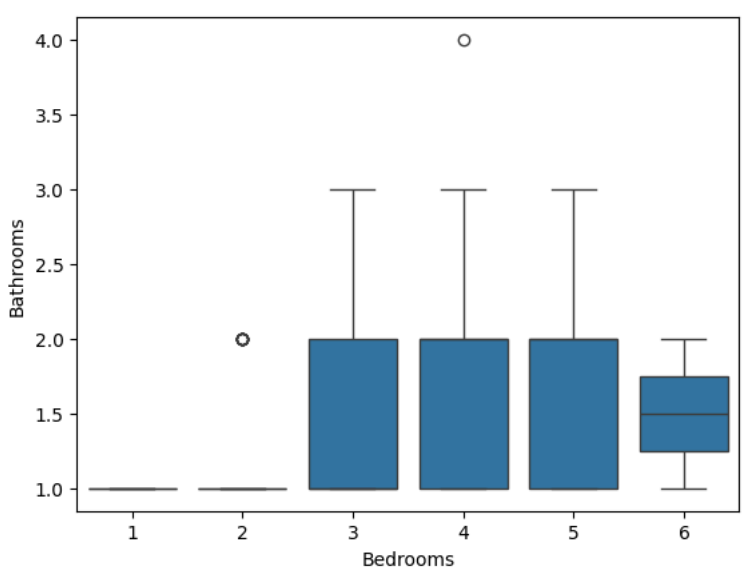

```
sns.boxplot(x='furnishingstatus', y='price', data=data)
plt.xlabel('Furnishing Status')
plt.ylabel('Price')
plt.show()
```




```
plt.scatter(data['area'], data['price'])
plt.xlabel('Area')
plt.ylabel('Price')
plt.show()
```

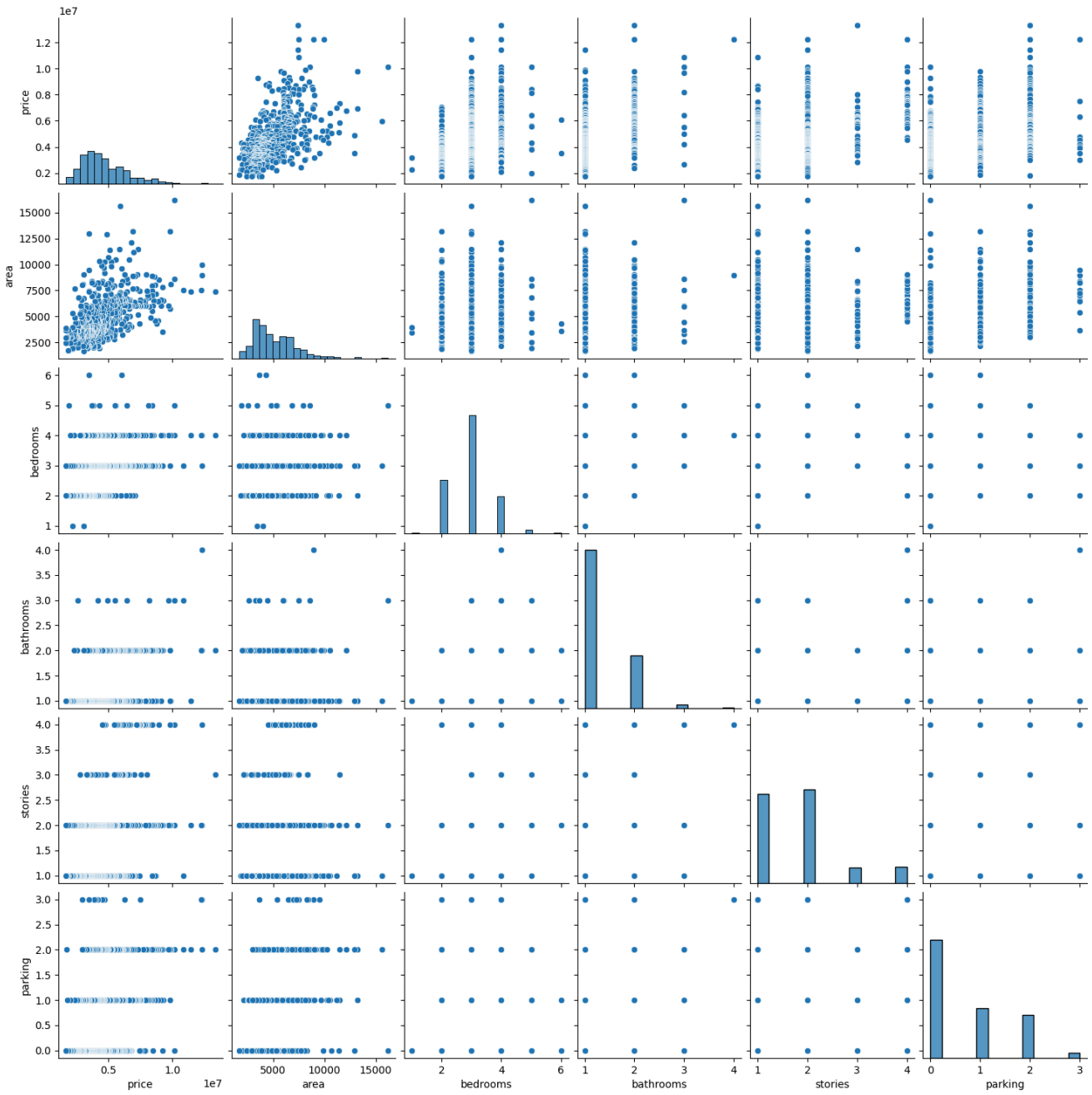



```
sns.boxplot(x='bedrooms', y='bathrooms', data=data)
plt.xlabel('Bedrooms')
plt.ylabel('Bathrooms')
plt.show()
```

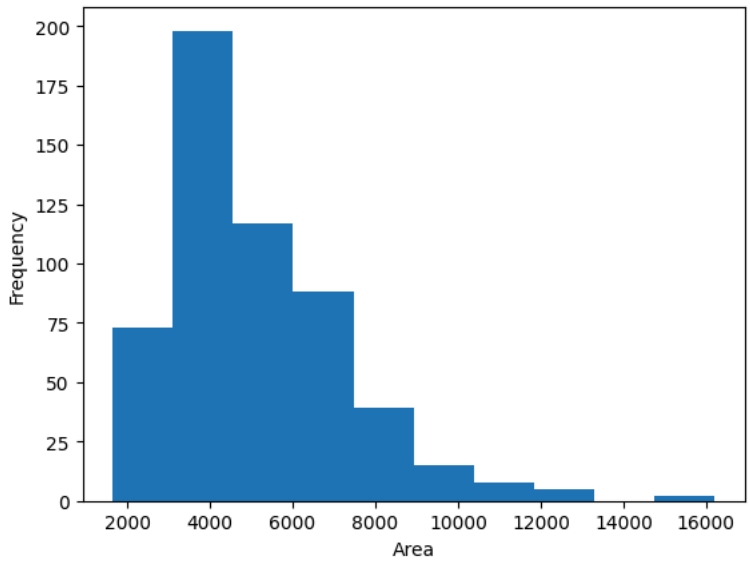



```
sns.pairplot(data)
plt.show()
```

[↓]




```
plt.hist(data['area'], bins=10)
plt.xlabel('Area')
plt.ylabel('Frequency')
plt.show()
```

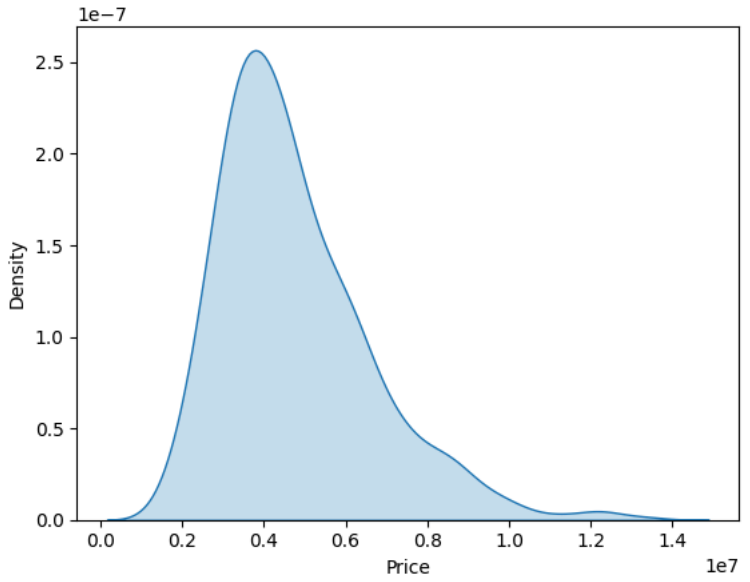



```
sns.kdeplot(data['price'], shade=True)
plt.xlabel('Price')
plt.ylabel('Density')
plt.show()
```

```
<ipython-input-22-0eff5dc43d41>:1: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data['price'], shade=True)
```




```
categorical_col =  ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea']
```

```
data[categorical_col]
```

	mainroad	guestroom	basement	hotwaterheating	airconditioning	prefarea
0	yes	no	no	no	yes	yes
1	yes	no	no	no	yes	no
2	yes	no	yes	no	no	yes
3	yes	no	yes	no	yes	yes
4	yes	yes	yes	no	yes	no
...	...	...	...	...	...	...
540	yes	no	yes	no	no	no
541	no	no	no	no	no	no
542	yes	no	no	no	no	no
543	no	no	no	no	no	no
544	yes	no	no	no	no	no

545 rows × 6 columns

```
def binary_map(x):
    """
    Function to map 'yes' and 'no' values to 1 and 0, respectively.

    Parameters:
    x (pandas Series): Input Series containing 'yes' and 'no' values.

    Returns:
    pandas Series: Mapped Series with 'yes' mapped to 1 and 'no' mapped to 0.
    """
    return x.map({'yes': 1, 'no': 0})


data[categorical_col] = data[categorical_col].apply(binary_map)

# Display the updated values of the categorical columns
data[categorical_col]
```

	mainroad	guestroom	basement	hotwaterheating	airconditioning	prefarea
0	1	0	0	0	1	1
1	1	0	0	0	1	0
2	1	0	1	0	0	1
3	1	0	1	0	1	1
4	1	1	1	0	1	0
...	...	...	...	...	...	...
540	1	0	1	0	0	0
541	0	0	0	0	0	0
542	1	0	0	0	0	0
543	0	0	0	0	0	0
544	1	0	0	0	0	0

545 rows × 6 columns

```
data.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterh
13300000	7420	4	2	3	1	0	0		
12250000	8960	4	4	4	1	0	0		
12250000	9960	3	2	2	1	0	1		
12215000	7500	4	2	2	1	0	1		
11410000	7420	4	1	2	1	1	1		


```
dummy_col = pd.get_dummies(data['furnishingstatus'])

# Display the first few rows of the dummy variables DataFrame
dummy_col.head()
```

	furnished	semi-furnished	unfurnished
0	True	False	False
1	True	False	False
2	False	True	False
3	True	False	False
4	True	False	False

Next steps: Generate code with dummy_col View recommended plots

```
dummy_col = pd.get_dummies(data['furnishingstatus'], drop_first=True)

# Display the first few rows of the dummy variables DataFrame
dummy_col.head()
```

	semi-furnished	unfurnished
0	False	False
1	False	False
2	True	False
3	False	False
4	False	False

Next steps: Generate code with dummy_col View recommended plots

```
data = pd.concat([data, dummy_col], axis=1)

# Display the first few rows of the updated DataFrame
data.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea
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	0
2	12250000	9960	3	2	2	1	0	1	0	0	2	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	0

Next steps: Generate code with `data` View recommended plots

```
data.drop(['furnishingstatus'], axis=1, inplace=True)

# Display the first few rows of the updated DataFrame
data.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwa
0	13300000	7420	4	2	3	1	0	0	
1	12250000	8960	4	4	4	1	0	0	
2	12250000	9960	3	2	2	1	0	1	
3	12215000	7500	4	2	2	1	0	1	
4	11410000	7420	4	1	2	1	1	1	

Next steps: Generate code with `data` View recommended plots

```
data.columns
```

```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'semi-furnished', 'unfurnished'],
      dtype='object')
```

```
np.random.seed(0)

# Split the data into training and testing subsets
# df_train: Training subset
# df_test: Testing subset
df_train, df_test = train_test_split(data, train_size=0.7, test_size=0.3, random_state=100)
```

```
df_train.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotw
359	3710000	3600	3	1	1	1	0	0	
19	8855000	6420	3	2	2	1	0	0	
159	5460000	3150	3	2	1	1	1	1	
35	8080940	7000	3	2	4	1	0	0	
28	8400000	7950	5	2	2	1	0	1	

Next steps: Generate code with `df_train` View recommended plots

```
df_train.shape
```

```
(381, 14)
```

```
df_test.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hot
265	4403000	2880	3	1	2	1	0	0	
54	7350000	6000	3	2	2	1	1	0	
171	5250000	10269	3	1	1	1	0	0	
244	4550000	5320	3	1	2	1	1	1	
268	4382000	4950	4	1	2	1	0	0	

Next steps: Generate code with `df_test` View recommended plots

```
df_test.shape
```

```
(164, 14)
```

```
scaler = MinMaxScaler()


col_to_scale = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking', 'price']
# Scaling the specified columns in the training subset using the MinMaxScaler
df_train[col_to_scale] = scaler.fit_transform(df_train[col_to_scale])
# Displaying the training subset
df_train.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement
359	0.169697	0.155227	0.4	0.0	0.000000	1	0	0
19	0.615152	0.403379	0.4	0.5	0.333333	1	0	0
159	0.321212	0.115628	0.4	0.5	0.000000	1	1	1
35	0.548133	0.454417	0.4	0.5	1.000000	1	0	0
28	0.575758	0.538015	0.8	0.5	0.333333	1	0	1

Next steps: Generate code with `df_train` View recommended plots

```
y_train = df_train.pop('price')

# Extract the remaining features as the training data
x_train = df_train
# To display the first few rows of the target variable in the training subset
y_train.head()
```

```
359    0.169697
19     0.615152
159    0.321212
35     0.548133
28     0.575758
Name: price, dtype: float64
```

```
linear_regression = LinearRegression()


linear_regression.fit(x_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
coefficients = linear_regression.coef_

# Print the coefficients
print(coefficients)
```

```
[ 0.23466354  0.04673453  0.19082319  0.10851563  0.05044144  0.03042826
  0.02159488  0.08486327  0.06688093  0.06073533  0.05942788  0.00092052
 -0.03100561]
```

```
score = linear_regression.score(x_train, y_train)

# Print the coefficient of determination (R²)
col_to_scale = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking','price']
0.6814892088451202
y_test = df_test.pop('price')

# Extract the remaining features as the testing data
x_test = df_test


prediction = linear_regression.predict(x_test)


r2 = r2_score(y_test, prediction)


y_test.shape

# Reshape y_test to a matrix with a single column
y_test_matrix = y_test.values.reshape(-1, 1)
# Creating a DataFrame with actual and predicted values
data_fname = pd.DataFrame({'actual': y_test_matrix.flatten(), 'predicted': prediction.flatten()})
```