In [1]:
```
# Supress Warnings

import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
```

In [2]:
```
housing=pd.read_csv(r'C:\Users\mohd arhab ahmad\Downloads\csv datasets\Housing.csv')
```

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

Out[3]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishi
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	semi
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	

In [4]:
```
housing.info()
```

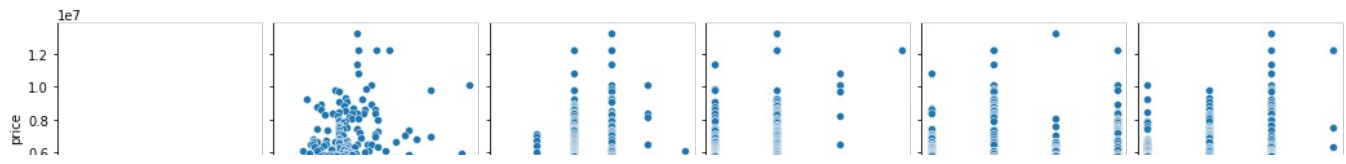
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   price             545 non-null    int64 
 1   area              545 non-null    int64 
 2   bedrooms          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
```

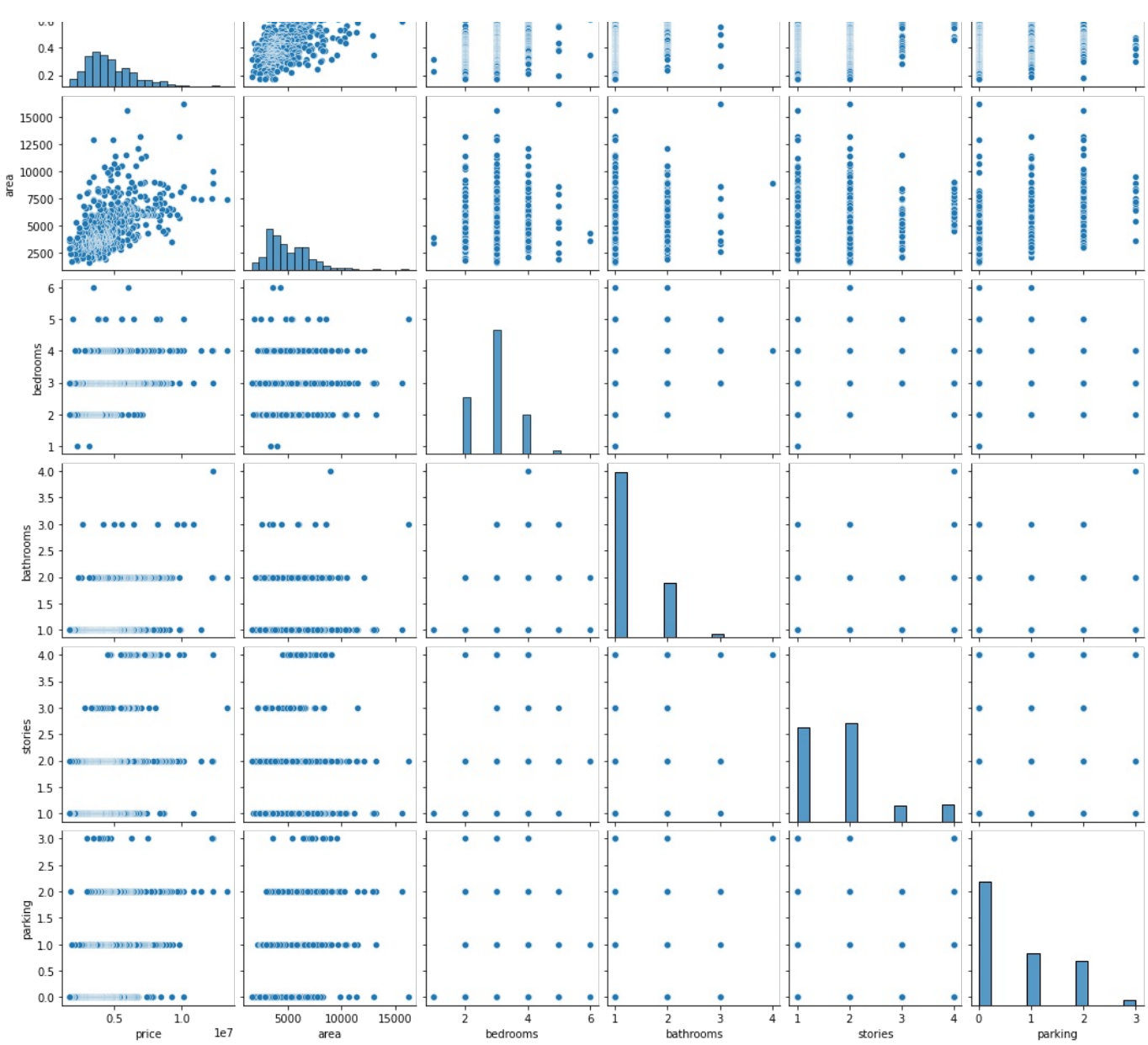
In [5]:
```
housing.describe()
```

Out[5]:

	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

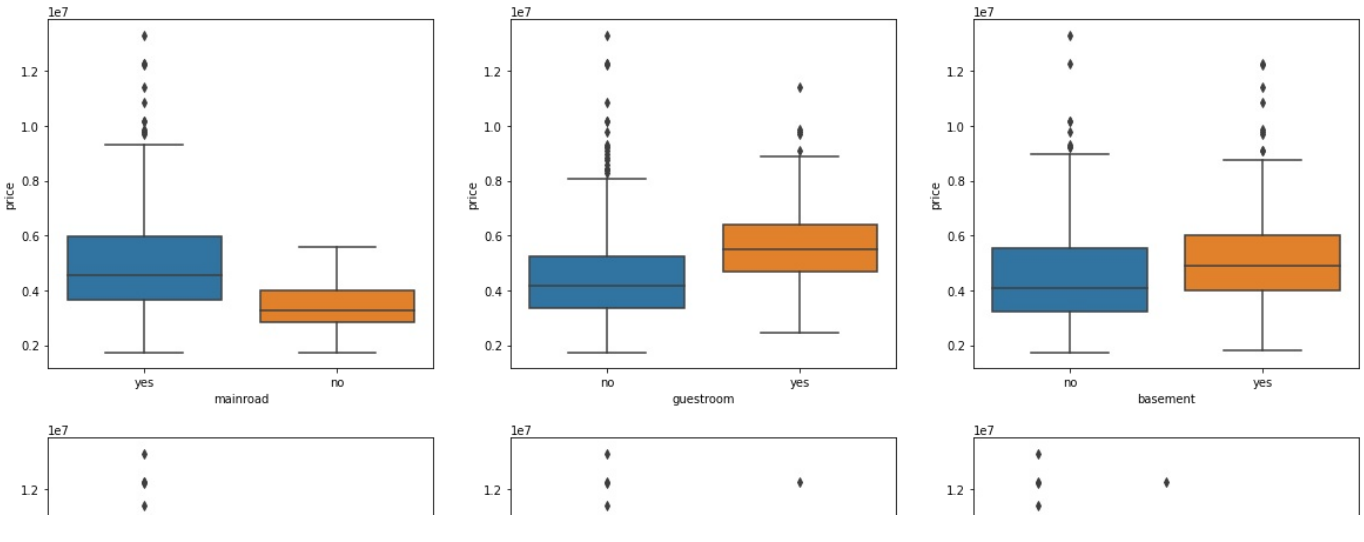
In [6]:
```
import matplotlib.pyplot as plt
import seaborn as sns

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

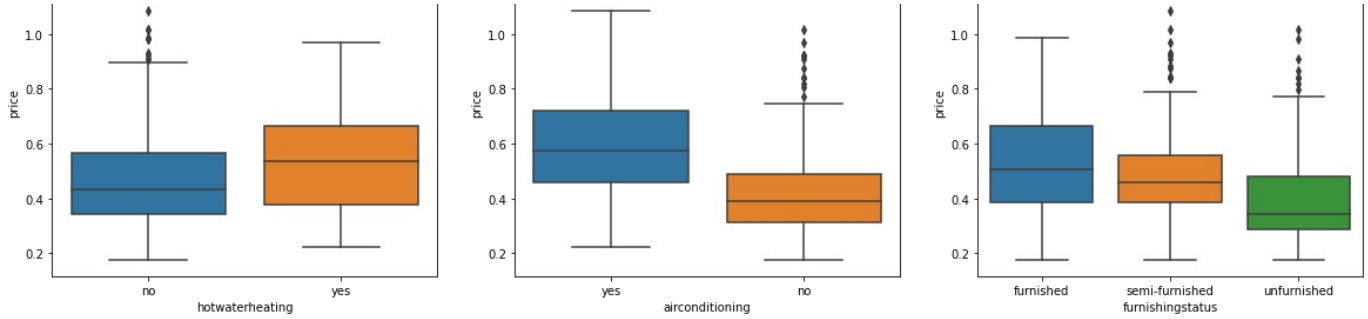




In [7]:

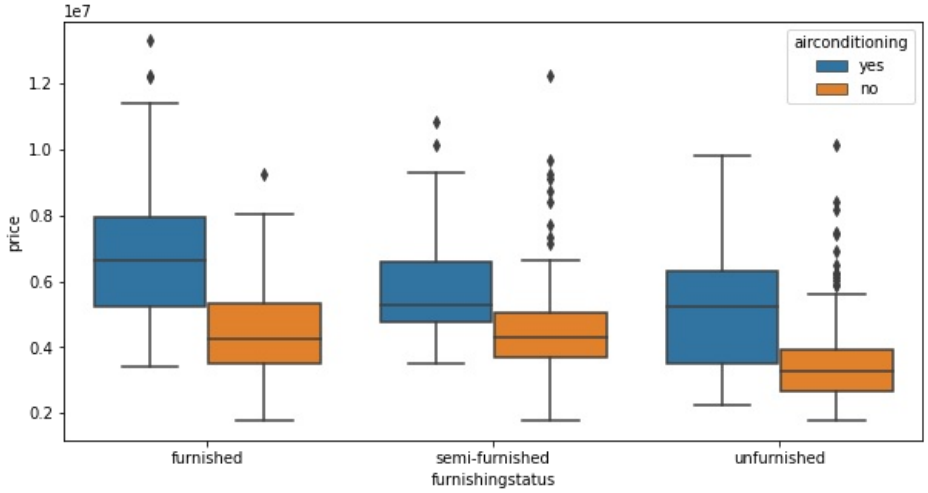
```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.boxplot(x = 'mainroad', y = 'price', data = housing)
plt.subplot(2,3,2)
sns.boxplot(x = 'guestroom', y = 'price', data = housing)
plt.subplot(2,3,3)
sns.boxplot(x = 'basement', y = 'price', data = housing)
plt.subplot(2,3,4)
sns.boxplot(x = 'hotwaterheating', y = 'price', data = housing)
plt.subplot(2,3,5)
sns.boxplot(x = 'airconditioning', y = 'price', data = housing)
plt.subplot(2,3,6)
sns.boxplot(x = 'furnishingstatus', y = 'price', data = housing)
plt.show()
```





In [8]:

```
plt.figure(figsize = (10, 5))
sns.boxplot(x = 'furnishingstatus', y = 'price', hue = 'airconditioning', data = housing)
plt.show()
```



In [9]:

```
# List of variables to map

varlist =  ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea']

# Defining the map function
def binary_map(x):
    return x.map({'yes': 1, "no": 0})

# Applying the function to the housing list
housing[varlist] = housing[varlist].apply(binary_map)
```

In [10]:

```
# Check the housing dataframe now
housing.head()
```

Out[10]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnish
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	semi-
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	

In [11]:

```
# Get the dummy variables for the feature 'furnishingstatus' and store it in a new variable - 'status'
status = pd.get_dummies(housing['furnishingstatus'])
```

In [12]:

```
# Check what the dataset 'status' looks like
status.head()
```

Out[12]:

	furnished	semi-furnished	unfurnished
0	1	0	0
1	1	0	0
2	0	1	0

3	1	0	0
4	1	0	0

In [13]:
```
# Let's drop the first column from status df using 'drop_first = True'
status = pd.get_dummies(housing['furnishingstatus'], drop_first = True)

# Add the results to the original housing dataframe
housing = pd.concat([housing, status], axis = 1)
```

In [14]:
```
# Now let's see the head of our dataframe.

housing.head()
```

Out[14]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishi
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	semi
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	

In [15]:
```
# Drop 'furnishingstatus' as we have created the dummies for it

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

In [16]:
```
housing.head()
```

Out[16]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	sem furnish
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	

In [17]:
```
from sklearn.model_selection import train_test_split

# We specify this so that the train and test data set always have the same rows, respectively
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.7, test_size = 0.3, random_state = 100)

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
```

In [18]:
```
# Apply scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking','price']

df_train[num_vars] = scaler.fit_transform(df_train[num_vars])
```

In [19]:
```
df_train.head()
```

Out[19]:

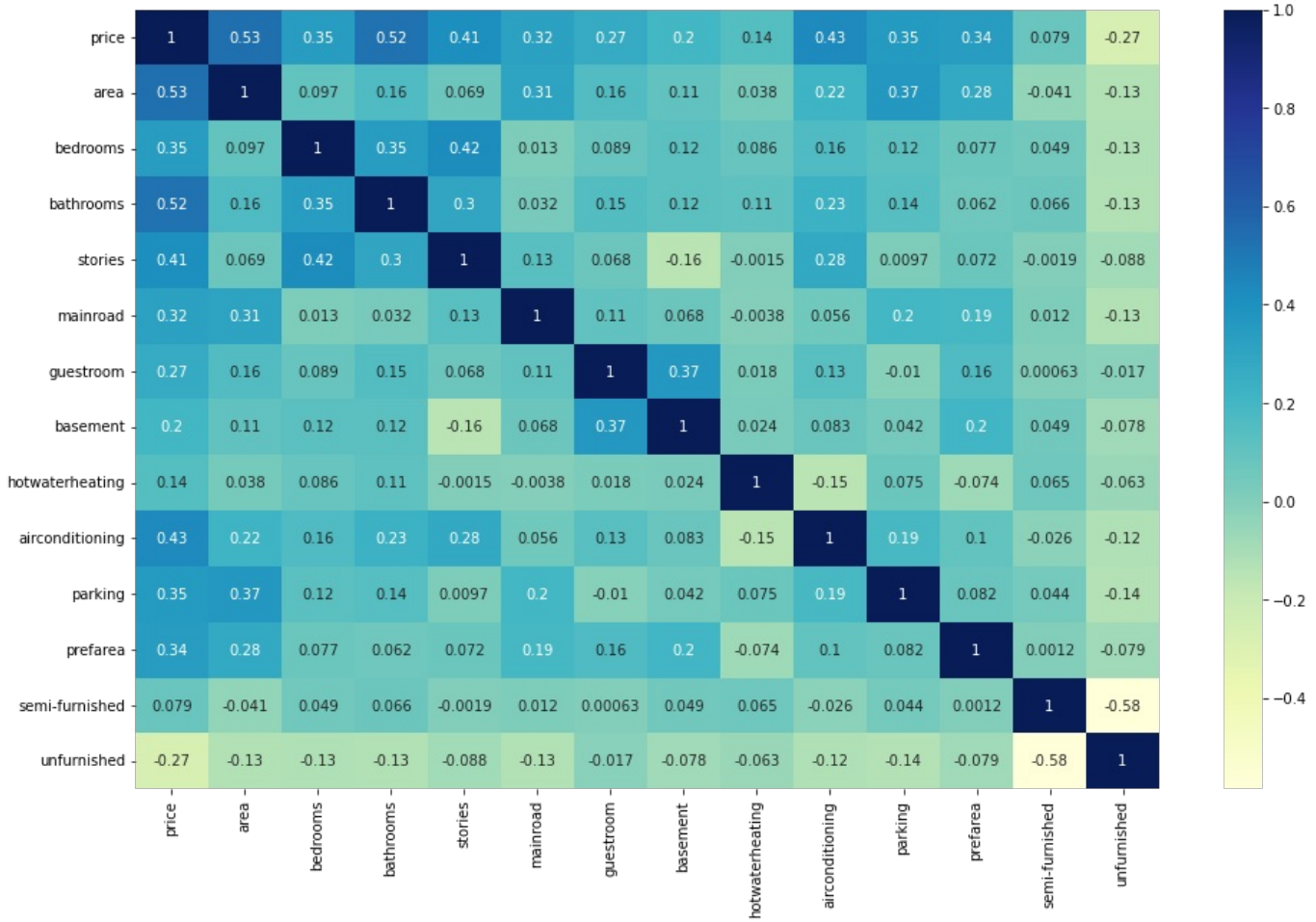
	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea
359	0.169697	0.155227	0.4	0.0	0.000000	1	0	0	0	0	0.333333	0
19	0.615152	0.403379	0.4	0.5	0.333333	1	0	0	0	1	0.333333	1
159	0.321212	0.115628	0.4	0.5	0.000000	1	1	1	0	1	0.000000	0
35	0.548133	0.454417	0.4	0.5	1.000000	1	0	0	0	1	0.666667	0
28	0.575758	0.538015	0.8	0.5	0.333333	1	0	1	1	0	0.666667	0

In [20]:
```
df_train.describe()
```

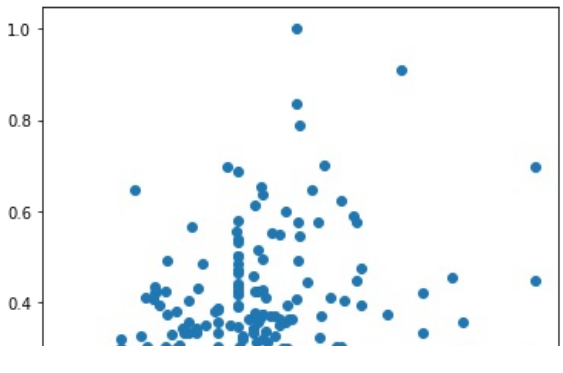
Out[20]:

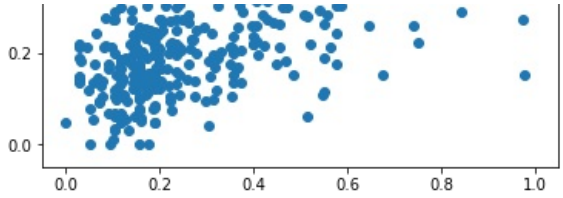
	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	pa
count	381.000000	381.000000	381.000000	381.000000	381.000000	381.000000	381.000000	381.000000	381.000000	381.000000	381.00
mean	0.260333	0.288710	0.386352	0.136483	0.268591	0.855643	0.170604	0.351706	0.052493	0.299213	0.24
std	0.157607	0.181420	0.147336	0.237325	0.295001	0.351913	0.376657	0.478131	0.223313	0.458515	0.28
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	0.151515	0.155227	0.200000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.00
50%	0.221212	0.234424	0.400000	0.000000	0.333333	1.000000	0.000000	0.000000	0.000000	0.000000	0.00
75%	0.345455	0.398099	0.400000	0.500000	0.333333	1.000000	0.000000	1.000000	0.000000	1.000000	0.33
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00

In [21]:
```
# Let's check the correlation coefficients to see which variables are highly correlated

plt.figure(figsize = (16, 10))
sns.heatmap(df_train.corr(), annot = True, cmap="YlGnBu")
plt.show()
```



In [22]:
```
plt.figure(figsize=[6,6])
plt.scatter(df_train.area, df_train.price)
plt.show()
```



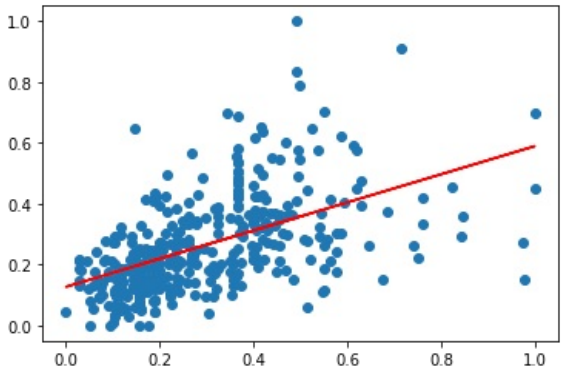


In [23]:
```
y_train = df_train.pop('price')
X_train = df_train
```

In [24]:
```
import statsmodels.api as sm

# Add a constant
X_train_lm = sm.add_constant(X_train[['area']])

# Create a first fitted model
lr = sm.OLS(y_train, X_train_lm).fit()
# Check the parameters obtained

lr.params
```

Out[24]:
```
const    0.126894
area     0.462192
dtype: float64
```

In [25]:
```
# Let's visualise the data with a scatter plot and the fitted regression line
plt.scatter(X_train_lm.iloc[:, 1], y_train)
plt.plot(X_train_lm.iloc[:, 1], 0.127 + 0.462*X_train_lm.iloc[:, 1], 'r')
plt.show()
```



In [26]:
```
# Print a summary of the linear regression model obtained
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.283
Model:                            OLS   Adj. R-squared:                  0.281
Method:                 Least Squares   F-statistic:                     149.6
Date:                Mon, 24 Jul 2023   Prob (F-statistic):           3.15e-29
Time:                        15:28:23   Log-Likelihood:                 227.23
No. Observations:                 381   AIC:                            -450.5
Df Residuals:                     379   BIC:                            -442.6
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1269      0.013      9.853      0.000       0.102       0.152
area           0.4622      0.038     12.232      0.000       0.388       0.536
==============================================================================
Omnibus:                       67.313   Durbin-Watson:                   2.018
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              143.063
Skew:                           0.925   Prob(JB):                     8.59e-32
Kurtosis:                       5.365   Cond. No.                         5.99
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [27]:
```
# Assign all the feature variables to X
X_train_lm = X_train[['area', 'bathrooms']]
# Build a linear model

import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train_lm)

lr = sm.OLS(y_train, X_train_lm).fit()

lr.params
```

Out[27]:
```
const        0.104589
area         0.398396
bathrooms    0.298374
dtype: float64
```

In [28]:
```
# Check the summary
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.480
Model:                            OLS   Adj. R-squared:                  0.477
Method:                 Least Squares   F-statistic:                     174.1
Date:                Mon, 24 Jul 2023   Prob (F-statistic):           2.51e-54
Time:                        15:28:23   Log-Likelihood:                 288.24
No. Observations:                 381   AIC:                            -570.5
Df Residuals:                     378   BIC:                            -558.6
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1046      0.011      9.384      0.000       0.083       0.127
area           0.3984      0.033     12.192      0.000       0.334       0.463
bathrooms      0.2984      0.025     11.945      0.000       0.249       0.347
==============================================================================
Omnibus:                       62.839   Durbin-Watson:                   2.157
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              168.790
Skew:                           0.784   Prob(JB):                     2.23e-37
Kurtosis:                       5.859   Cond. No.                         6.17
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [29]:
```
# Assign all the feature variables to X
X_train_lm = X_train[['area', 'bathrooms','bedrooms']]
# Build a linear model

import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train_lm)

lr = sm.OLS(y_train, X_train_lm).fit()

lr.params
```

Out[29]:
```
const        0.041352
area         0.392211
bathrooms    0.259978
bedrooms     0.181863
dtype: float64
```

In [30]:
```
# Print the summary of the model

print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.505
Model:                            OLS   Adj. R-squared:                  0.501
Method:                 Least Squares   F-statistic:                     128.2
Date:                Mon, 24 Jul 2023   Prob (F-statistic):           3.12e-57
Time:                        15:28:23   Log-Likelihood:                 297.76
No. Observations:                 381   AIC:                            -587.5
Df Residuals:                     377   BIC:                            -571.7
Df Model:                           3
```

```
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0414      0.018      2.292      0.022       0.006       0.077
area           0.3922      0.032     12.279      0.000       0.329       0.455
bathrooms      0.2600      0.026     10.033      0.000       0.209       0.311
bedrooms       0.1819      0.041      4.396      0.000       0.101       0.263
==============================================================================
Omnibus:                       50.037   Durbin-Watson:                   2.136
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              124.806
Skew:                           0.648   Prob(JB):                     7.92e-28
Kurtosis:                       5.487   Cond. No.                         8.87
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [ ]: