House price prediction

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

In [2]:
```
Data = pd.read_csv("C:/Users/hp/OneDrive/Desktop/himadhruthi/data.csv")
Data.head()
```

Out[2]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
0	2014-05-02 00:00:00	313000.0	3.0	1.50	1340	7912	1.5	0	0
1	2014-05-02 00:00:00	2384000.0	5.0	2.50	3650	9050	2.0	0	4
2	2014-05-02 00:00:00	342000.0	3.0	2.00	1930	11947	1.0	0	0
3	2014-05-02 00:00:00	420000.0	3.0	2.25	2000	8030	1.0	0	0
4	2014-05-02 00:00:00	550000.0	4.0	2.50	1940	10500	1.0	0	0

In [3]:
```
Data.columns
```

Out[3]:
```
Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot
',
       'floors', 'waterfront', 'view', 'condition', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'street', 'city',
       'statezip', 'country'],
      dtype='object')
```

In [4]:
```
Data.describe()
```

Out[4]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	v
count	4.600000e+03	4600.000000	4600.000000	4600.000000	4.600000e+03	4600.000000	46(
mean	5.519630e+05	3.400870	2.160815	2139.346957	1.485252e+04	1.512065	
std	5.638347e+05	0.908848	0.783781	963.206916	3.588444e+04	0.538288	
min	0.000000e+00	0.000000	0.000000	370.000000	6.380000e+02	1.000000	
25%	3.228750e+05	3.000000	1.750000	1460.000000	5.000750e+03	1.000000	
50%	4.609435e+05	3.000000	2.250000	1980.000000	7.683000e+03	1.500000	
75%	6.549625e+05	4.000000	2.500000	2620.000000	1.100125e+04	2.000000	
max	2.659000e+07	9.000000	8.000000	13540.000000	1.074218e+06	3.500000	

In [5]:
```
Data.isnull().sum()
```

Out[5]:
```
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
street           0
city             0
statezip         0
country          0
dtype: int64
```

In [6]:
```
x= Data.drop(['date','street','street','city','statezip','country','price']
x.head()
```

Out[6]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above
0	3.0	1.50	1340	7912	1.5	0	0	3	1340
1	5.0	2.50	3650	9050	2.0	0	4	5	3370
2	3.0	2.00	1930	11947	1.0	0	0	4	1930
3	3.0	2.25	2000	8030	1.0	0	0	4	1000
4	4.0	2.50	1940	10500	1.0	0	0	4	1140

In [7]:
```
y = Data["price"]
y.head()
```

Out[7]:
```
0     313000.0
1    2384000.0
2     342000.0
3     420000.0
4     550000.0
Name: price, dtype: float64
```

In [8]:
```
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   bedrooms       4600 non-null   float64
 1   bathrooms      4600 non-null   float64
 2   sqft_living    4600 non-null   int64  
 3   sqft_lot       4600 non-null   int64  
 4   floors         4600 non-null   float64
 5   waterfront     4600 non-null   int64  
 6   view           4600 non-null   int64  
 7   condition      4600 non-null   int64  
 8   sqft_above     4600 non-null   int64  
 9   sqft_basement  4600 non-null   int64  
 10  yr_built       4600 non-null   int64  
 11  yr_renovated   4600 non-null   int64  
dtypes: float64(3), int64(9)
memory usage: 431.4 KB
```

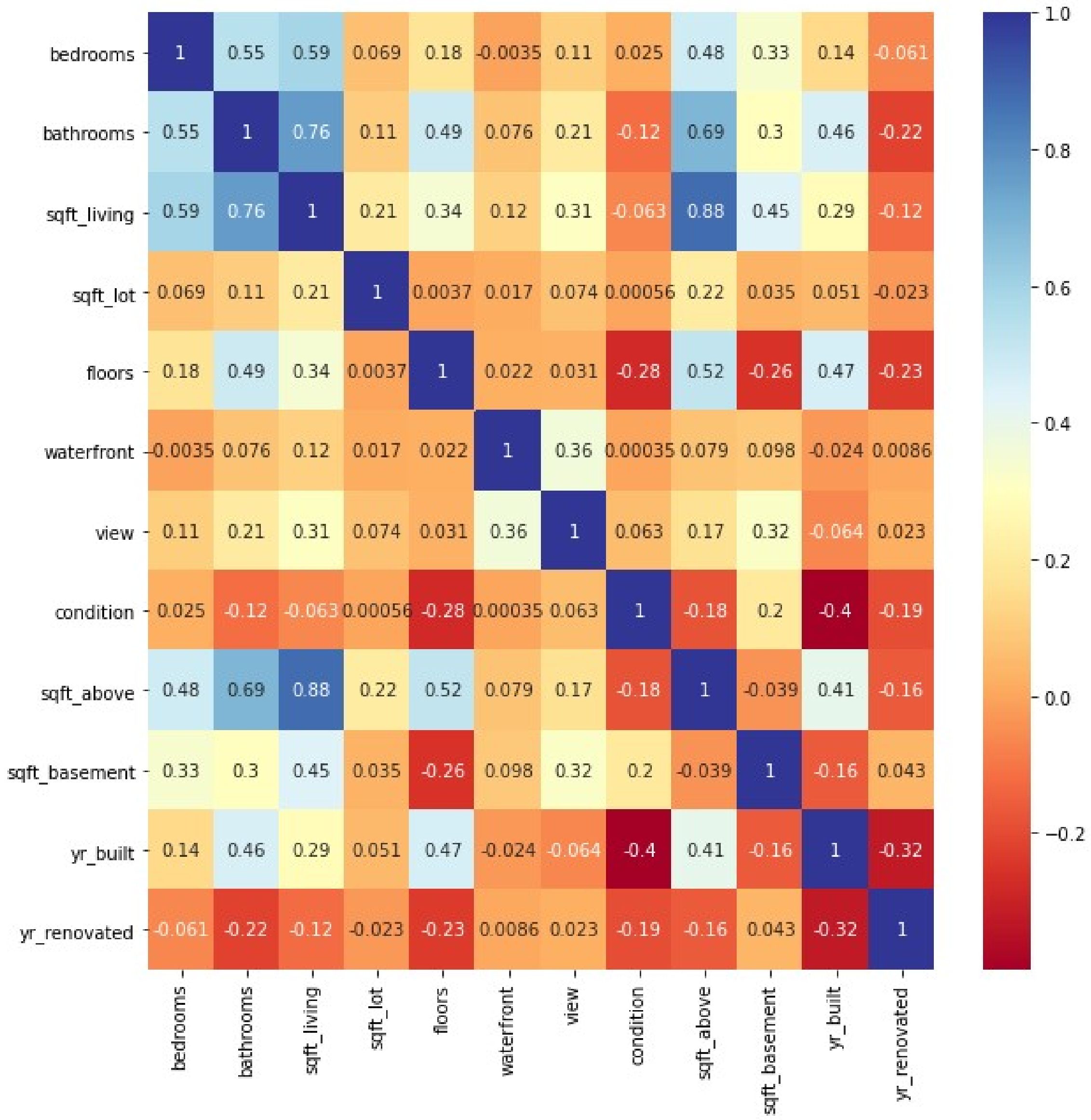
In [9]:
```
x.corr()
```

Out[9]:

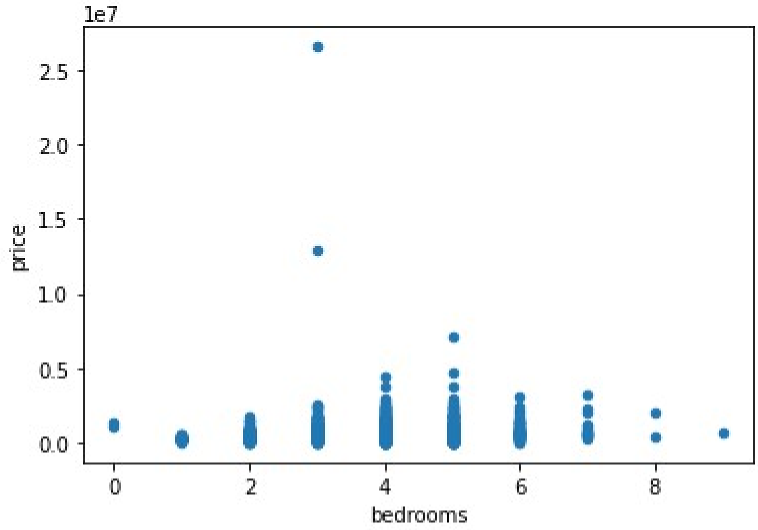
	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
bedrooms	1.000000	0.545920	0.594884	0.068819	0.177895	-0.003483	0.111028
bathrooms	0.545920	1.000000	0.761154	0.107837	0.486428	0.076232	0.211960
sqft_living	0.594884	0.761154	1.000000	0.210538	0.344850	0.117616	0.311009
sqft_lot	0.068819	0.107837	0.210538	1.000000	0.003750	0.017241	0.073907
floors	0.177895	0.486428	0.344850	0.003750	1.000000	0.022024	0.031211
waterfront	-0.003483	0.076232	0.117616	0.017241	0.022024	1.000000	0.360935
view	0.111028	0.211960	0.311009	0.073907	0.031211	0.360935	1.000000
condition	0.025080	-0.119994	-0.062826	0.000558	-0.275013	0.000352	0.063077
sqft_above	0.484705	0.689918	0.876443	0.216455	0.522814	0.078911	0.174327
sqft_basement	0.334165	0.298020	0.447206	0.034842	-0.255510	0.097501	0.321602
yr_built	0.142461	0.463498	0.287775	0.050706	0.467481	-0.023563	-0.064465
yr_renovated	-0.061082	-0.215886	-0.122817	-0.022730	-0.233996	0.008625	0.022967

In [10]:
```
plt.figure(figsize=(10,10))
sns.heatmap(x.corr(),annot =True,cmap='RdYlBu')
```

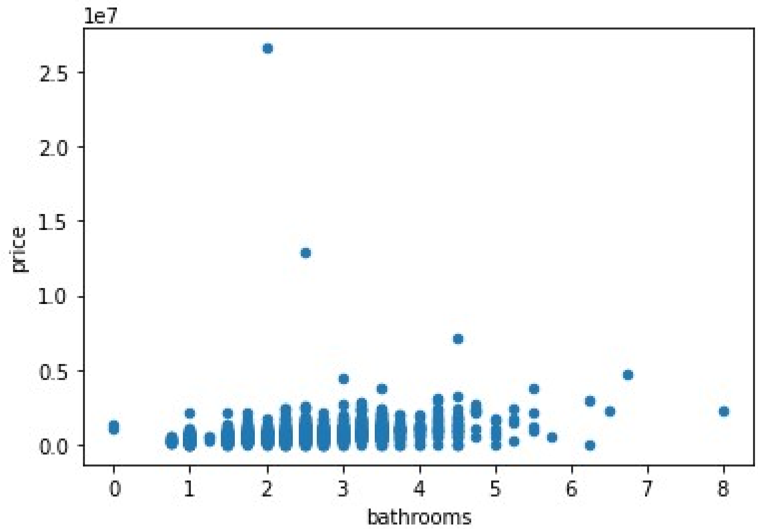
Out[10]: <AxesSubplot:>



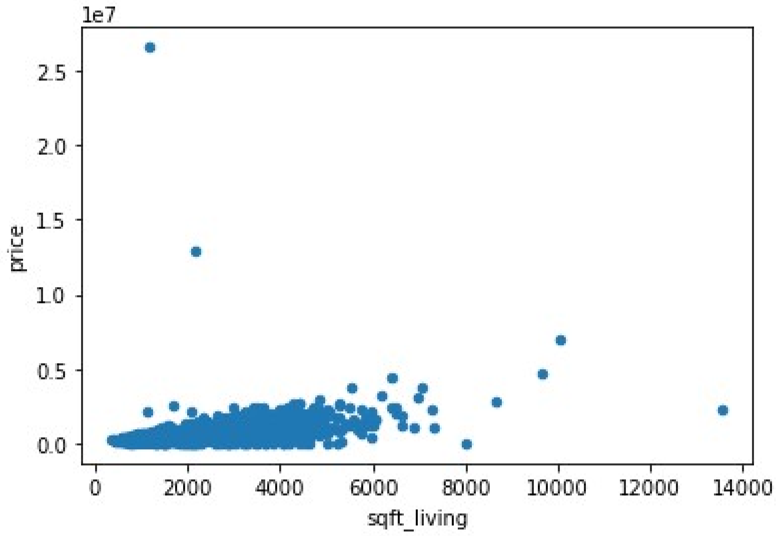
In [11]:
```
def Scatter(a):
    Data.plot(kind = "scatter" , x=a,y='price')
    plt.show()
Scatter('bedrooms')
```



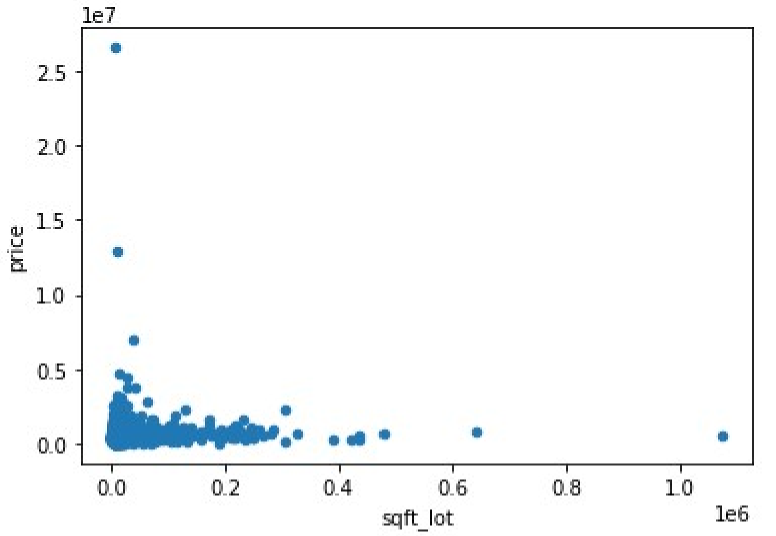
In [12]:
```
Scatter('bathrooms')
```



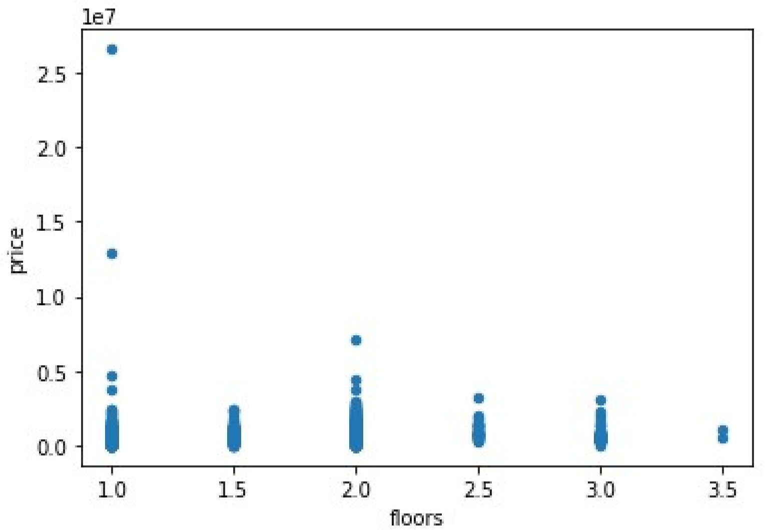
In [13]:
```python
Scatter('sqft_living')
```



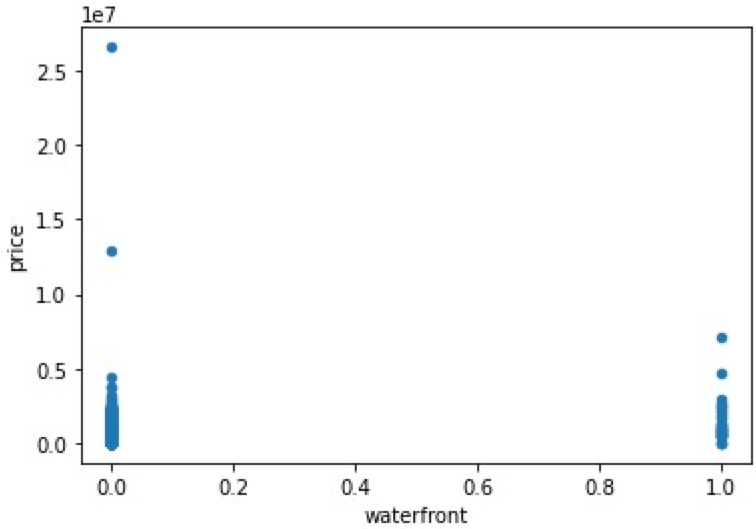
In [14]:
```python
Scatter('sqft_lot')  data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAYIAAAES(
```



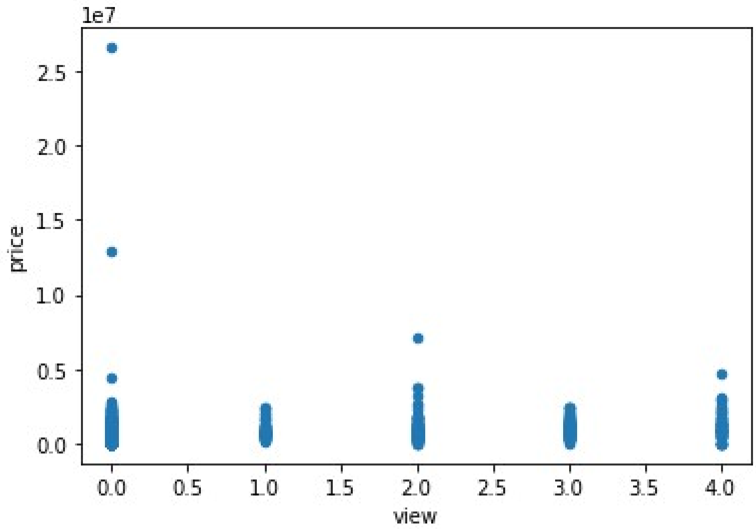
In [15]:
```python
Scatter('floors')
```



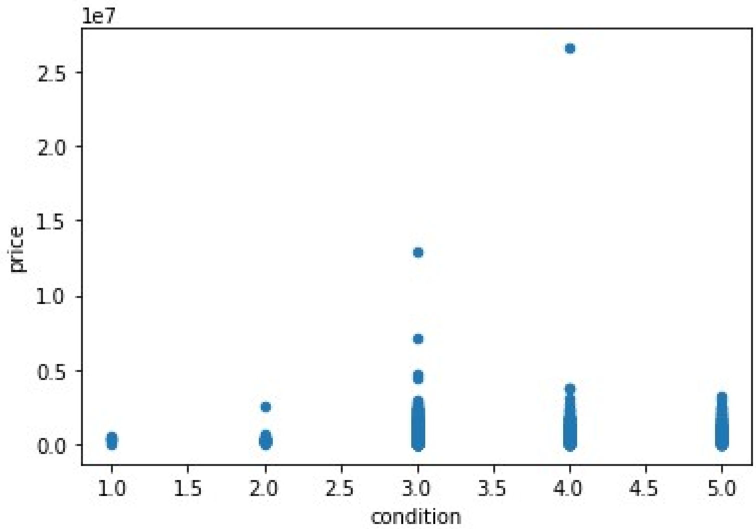
In [16]:
```
Scatter('waterfront')
```



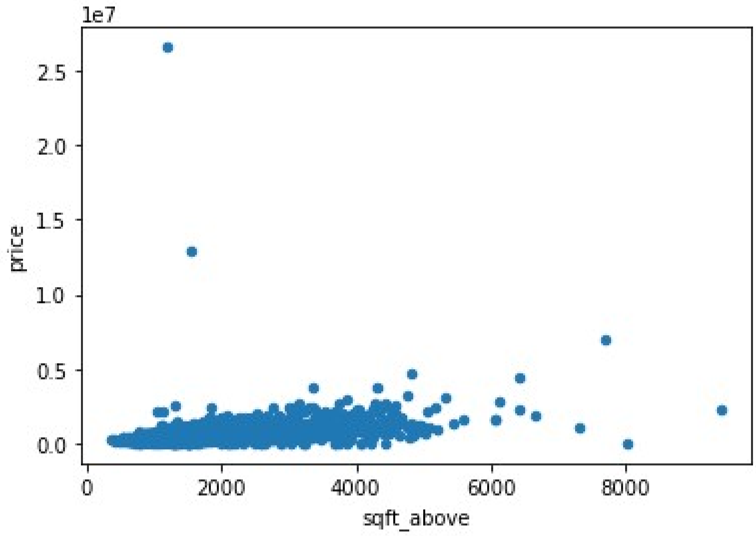
In [17]:
```
Scatter('view')
```



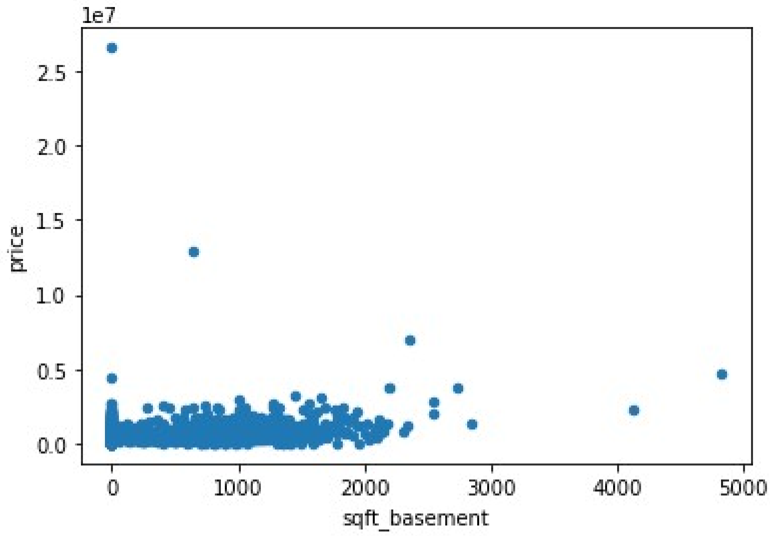
In [18]:
```
Scatter('condition')
```



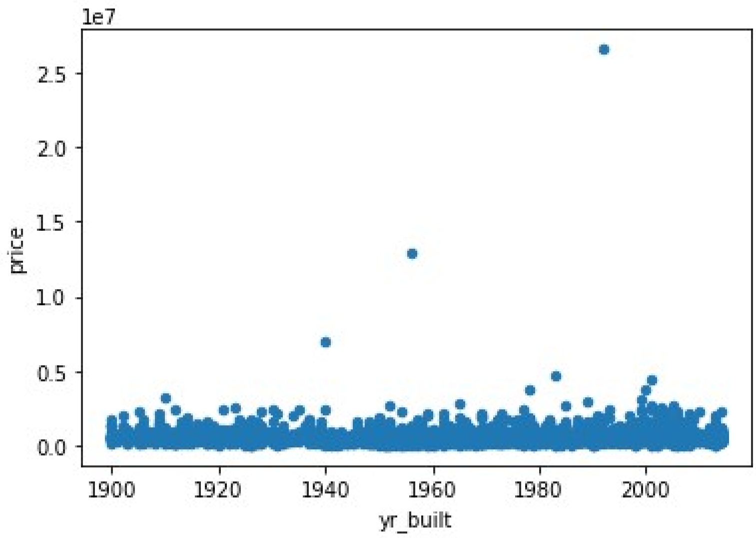
In [19]:
```
Scatter('sqft_above')
```



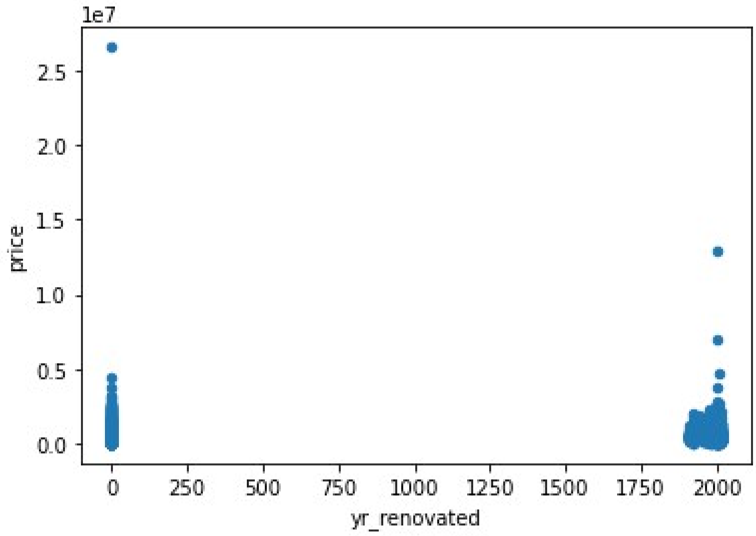
In [20]:
```
Scatter('sqft_basement')
```



In [21]:
```
Scatter('yr_built')
```



In [22]:
```
Scatter('yr_renovated')
```



In [23]:
```
X_mean = x.mean()

# Standard deviation
X_std = x.std()

# Standardization
Z = (x - X_mean) / X_std
print(Z)
```

```
      bedrooms  bathrooms  sqft_living  sqft_lot    floors  waterfront  \
0    -0.441074  -0.843112    -0.829881 -0.193413 -0.022414   -0.084995
1     1.759513   0.432754     1.568358 -0.161700  0.906456   -0.084995
2    -0.441074  -0.205179    -0.217344 -0.080969 -0.951284   -0.084995
3    -0.441074   0.113788    -0.144670 -0.190125 -0.951284   -0.084995
4     0.659220   0.432754    -0.206962 -0.121293 -0.951284   -0.084995
...        ...        ...          ...       ...       ...         ...
4595 -0.441074  -0.524145    -0.653387 -0.236663 -0.951284   -0.084995
4596 -0.441074   0.432754    -0.705297 -0.202860  0.906456   -0.084995
4597 -0.441074   0.432754     0.903911 -0.218438  0.906456   -0.084995
4598  0.659220  -0.205179    -0.051232 -0.229139 -0.951284   -0.084995
4599 -0.441074   0.432754    -0.674151 -0.188118  0.906456   -0.084995

          view  condition  sqft_above  sqft_basement  yr_built  yr_renovat
ed
0    -0.309161  -0.667040   -0.565162      -0.672391 -0.530956      1.2215
38
1     4.829554   2.286168    1.789365      -0.069121 -1.674511     -0.8256
04
2    -0.309161   0.809564    0.119158      -0.672391 -0.160982     -0.8256
04
3    -0.309161   0.809564   -0.959517       1.482145 -0.261884     -0.8256
04
4    -0.309161   0.809564   -0.797135       1.051238  0.175357      1.2082
64
...        ...        ...         ...            ...       ...
...
4595 -0.309161   0.809564   -0.367985      -0.672391 -0.564590      1.1949
91
4596 -0.309161  -0.667040   -0.425978      -0.672391  0.410795      1.2256
22
4597 -0.309161  -0.667040    1.371813      -0.672391  1.285278     -0.8256
04
4598 -0.309161  -0.667040   -0.878326       1.525235  0.108089     -0.8256
04
4599 -0.309161   0.809564   -0.391183      -0.672391  0.646233     -0.8256
04

[4600 rows x 12 columns]
```

In [24]:
```
scalar=StandardScaler()
#scalar.fit(x)
X=scalar.fit_transform(x)
print(X)
```

```
[[-0.44112227 -0.84320364 -0.82997105 ... -0.67246372 -0.53101376
   1.22167046]
 [ 1.75970468  0.43280154  1.56852826 ... -0.06912828 -1.67469295
  -0.82569345]
 [-0.44112227 -0.20520105 -0.21736733 ... -0.67246372 -0.1609999
  -0.82569345]
 ...
 [-0.44112227  0.43280154  0.90400897 ... -0.67246372  1.2854179
  -0.82569345]
 [ 0.6592912  -0.20520105 -0.05123751 ...  1.5254011   0.10810108
  -0.82569345]
 [-0.44112227  0.43280154 -0.67422434 ... -0.67246372  0.64630305
  -0.82569345]]
```

In [26]:
```
Ss = [w for w in x]
Ss
```

Out[26]:
```
['bedrooms',
 'bathrooms',
 'sqft_living',
 'sqft_lot',
 'floors',
 'waterfront',
 'view',
 'condition',
 'sqft_above',
 'sqft_basement',
 'yr_built',
 'yr_renovated']
```

In [27]:
```
X = sm.add_constant( Data[Ss] )
X.head(5)
```

Out[27]:

	const	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqf
0	1.0	3.0	1.50	1340	7912	1.5	0	0	3	
1	1.0	5.0	2.50	3650	9050	2.0	0	4	5	
2	1.0	3.0	2.00	1930	11947	1.0	0	0	4	
3	1.0	3.0	2.25	2000	8030	1.0	0	0	4	
4	1.0	4.0	2.50	1940	10500	1.0	0	0	4	

In [28]:
```
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y= train_test_split(X, y , train_size=0.7)
```

In [29]:
```
test_x.shape
```

Out[29]: (1380, 13)

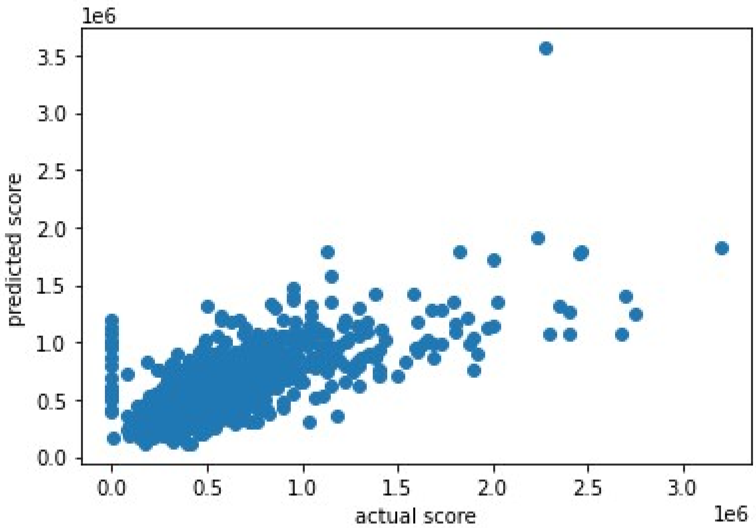
In [30]:
```
train_x.shape
```
Out[30]:
```
(3220, 13)
```

In [31]:
```
model = LinearRegression()
model.fit(train_x , train_y)
```
Out[31]:
```
▾ LinearRegression
LinearRegression()
```

In [32]:
```
pred_y = model.predict(test_x)
c = model.intercept_
c
```
Out[32]:
```
4596507.247459341
```

In [33]:
```
m = model.coef_
m
```
Out[33]:
```
array([ 0.00000000e+00, -6.35169549e+04,  6.07634606e+04,  1.70610648e+02,
       -5.91969678e-01,  3.62503877e+04,  4.05496194e+05,  3.27418491e+04,
        3.85748516e+04,  9.82494855e+01,  7.23611620e+01, -2.39190732e+03,
        4.04863974e+00])
```

In [34]:
```
print(pred_y)
```
```
[ 175336.38902299  521055.75656389  270315.74839067 ...  644250.23673443
  344889.21358581 1161871.14279958]
```

In [35]:
```
plt.scatter(test_y,pred_y)
plt.xlabel('actual score')
plt.ylabel('predicted score')
plt.show()
```


In [40]:
```
r2= r2_score(test_y,pred_y)
r2
```
Out[40]:
```
0.5024517349159334
```

In [37]:
```
model.predict([[1,3,1.50,1340,7912,1.5,0,0,3,1340,0,1955,2005]])
```

Out[37]:
```
array([354729.33696997])
```

In [38]:
```
error = 367194.60858801 - 313000.0
error
```

Out[38]:
```
54194.608588010015
```

In [41]:
```
if r2 >= 0.7:
    print("The model has a strong fit and performs well.")
elif r2 >= 0.5:
    print("The model has a moderate fit and gives decent predictions.")
else:
    print("The model may need further improvement as it has a weak fit.")

print("You can further refine the model by feature engineering and hyperpara
```

```
The model has a moderate fit and gives decent predictions.
You can further refine the model by feature engineering and hyperparameter
tuning to improve its performance.
```

This Jupyter Notebook code begins by loading the dataset, preprocessing the data (which includes handling missing values, encoding categorical variables, and selecting relevant features), and splitting the data into training and testing sets. It then creates a Linear Regression model, fits it to the training data, makes predictions on the test data, and evaluates the model's performance using metrics.

The conclusion section interprets the results and provides guidance on potential improvements to the model. This allows you to assess how well the model is performing and what actions can be taken to enhance its accuracy.