Recommendations for Improving Property valuation for Kings County Real Estate Agency

Author: Shimnaz Fathima

Welcome to the analysis done by Shimnaz Fathima! You will find a brief overview of the project, the findings and the recommendations derived after the completion of this project here.

Project Overview

This project involves using the data of housing sales in the Northwestern county of King County and recommending the client, King County Real Estate, a real estate agent in the region on aspects related to houses that will improve the value of the property.

Business Problem

This project uses Multilinear Correlation theory to assess the features that improve the price of the property and recommend it to our client King County Real Estate. These recommendations will used by them to target houses with high value and also to recommend their potential house selling clients on increasing their value of the property. After the analysis, the factor by which the value of the property will increase for each feature will be discussed.

The code starts here:

Importing relevant libraries

In [123…

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
import scipy.stats as stats
```

Inputting raw data and exploring the data

In [123…

```
df1=pd.read_csv('data/kc_house_data.csv')
df1
```

Out[1234]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basemer
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN	0.0	...	7	1180	0
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	...	7	2170	400
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	...	6	770	0
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	...	7	1050	910
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	...	8	1680	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	0.0	0.0	...	8	1530	0
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	0.0	0.0	...	8	2310	0
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	0.0	0.0	...	7	1020	0
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	NaN	0.0	...	8	1600	0
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	2.0	0.0	0.0	...	7	1020	0

21597 rows × 21 columns

In [123…

```
#checking info of df1
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 1   date           21597 non-null  object 
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64  
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64  
 6   sqft_lot       21597 non-null  int64  
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  float64
 9   view           21534 non-null  float64
 10  condition      21597 non-null  int64  
 11  grade          21597 non-null  int64  
 12  sqft_above     21597 non-null  int64  
 13  sqft_basement  21597 non-null  object 
 14  yr_built       21597 non-null  int64  
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64  
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64  
 20  sqft_lot15     21597 non-null  int64  
dtypes: float64(8), int64(11), object(2)
memory usage: 3.5+ MB
```

Having first glance look into the data. It looks like the columns, price, sqft_living, sqft_lot, sqft_above, sqft_basement are continous variables.

In [123…

```
#removing character symbols from the sqft_basement column as it will affect any furtehr arithmetic operations
df1['sqft_basement']=df1['sqft_basement'].str.extract(pat='(\d+)', expand=False)
df1['sqft_basement']=df1['sqft_basement'].astype(float)
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 1   date           21597 non-null  object 
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64  
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64  
 6   sqft_lot       21597 non-null  int64  
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  float64
 9   view           21534 non-null  float64
 10  condition      21597 non-null  int64  
 11  grade          21597 non-null  int64  
 12  sqft_above     21597 non-null  int64  
 13  sqft_basement  21143 non-null  float64
 14  yr_built       21597 non-null  int64  
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64  
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64  
 20  sqft_lot15     21597 non-null  int64  
dtypes: float64(9), int64(11), object(1)
memory usage: 3.5+ MB
```

Removing the outliers in the data

The data which is more than 2 standard deviations will be considered as outliers

In [123…

```
#finding mean and standard deviation of continous variables
price_mean=df1['price'].mean()
price_std=df1['price'].std()
sqft_living_mean=df1['sqft_living'].mean()
sqft_living_std=df1['sqft_living'].std()
sqft_lot_mean=df1['sqft_lot'].mean()
sqft_lot_std=df1['sqft_lot'].std()
sqft_above_mean=df1['sqft_above'].mean()
sqft_above_std=df1['sqft_above'].std()
sqft_basement_mean=df1['sqft_basement'].mean()
sqft_basement_std=df1['sqft_basement'].std()
sqft_lot15_mean=df1['sqft_lot15'].mean()
sqft_lot15_std=df1['sqft_lot15'].std()
```

In [123…
```
#dropping data which is more than 2 x standard deviation
sd_criteria=2
df1=df1.drop(df1[abs(df1['price'])> price_mean + (sd_criteria*price_std)].index)
df1=df1.drop(df1[abs(df1['sqft_living'])> sqft_living_mean + (sd_criteria*sqft_living_std)].index)
df1=df1.drop(df1[abs(df1['sqft_lot'])> sqft_lot_mean + (sd_criteria*sqft_lot_std)].index)
df1=df1.drop(df1[abs(df1['sqft_above'])> sqft_above_mean + (sd_criteria*sqft_above_std)].index)
df1=df1.drop(df1[abs(df1['sqft_basement'])> sqft_basement_mean + (sd_criteria*sqft_basement_std)].index)
df1=df1.drop(df1[abs(df1['sqft_lot15'])> sqft_lot15_mean + (sd_criteria*sqft_lot15_std)].index)
df1= df1.drop(df1[df1['bedrooms'] > 11].index)
df1= df1.drop(df1[df1['bathrooms'] > 6].index)
df1.describe()
```

Out[1238]:

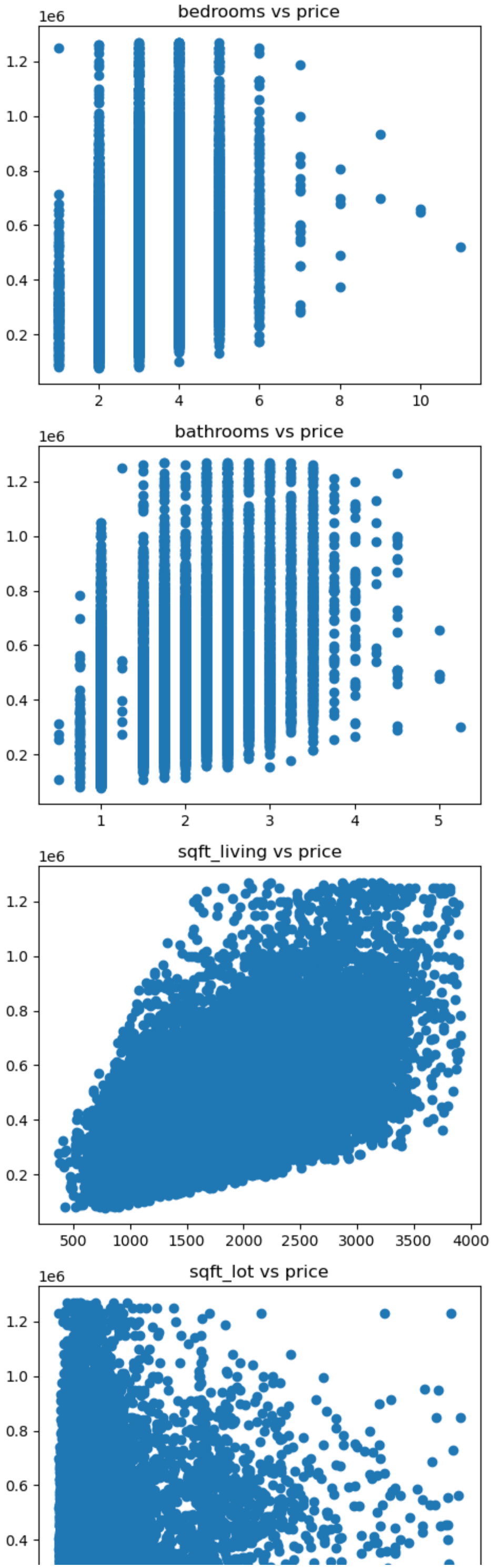
	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition
count	1.875700e+04	1.875700e+04	18757.000000	18757.000000	18757.000000	18757.000000	18757.000000	16675.000000	18705.000000	18757.000000
mean	4.672442e+09	4.647717e+05	3.271739	1.999094	1865.080503	9225.498054	1.476862	0.002939	0.151991	3.407101
std	2.875862e+09	2.077706e+05	0.849709	0.677657	654.331274	9372.248641	0.540479	0.054130	0.603618	0.648068
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	520.000000	1.000000	0.000000	0.000000	1.000000
25%	2.212200e+09	3.071500e+05	3.000000	1.500000	1370.000000	4920.000000	1.000000	0.000000	0.000000	3.000000
50%	4.023500e+09	4.250000e+05	3.000000	2.000000	1790.000000	7260.000000	1.000000	0.000000	0.000000	3.000000
75%	7.436300e+09	5.800000e+05	4.000000	2.500000	2300.000000	9748.000000	2.000000	0.000000	0.000000	4.000000
max	9.900000e+09	1.270000e+06	11.000000	5.250000	3910.000000	97661.000000	3.500000	1.000000	4.000000	5.000000

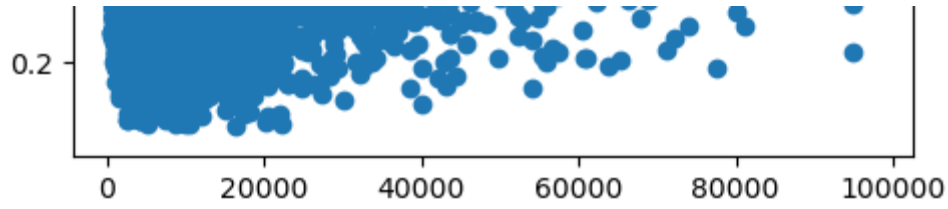
By looking at the 5 point data, we can understand that variables like 'bedrooms', 'bathrooms', 'floor', 'waterfront', 'view', 'condition' and 'grade' are categorical variables and may need to implement dummy variables to handle these.

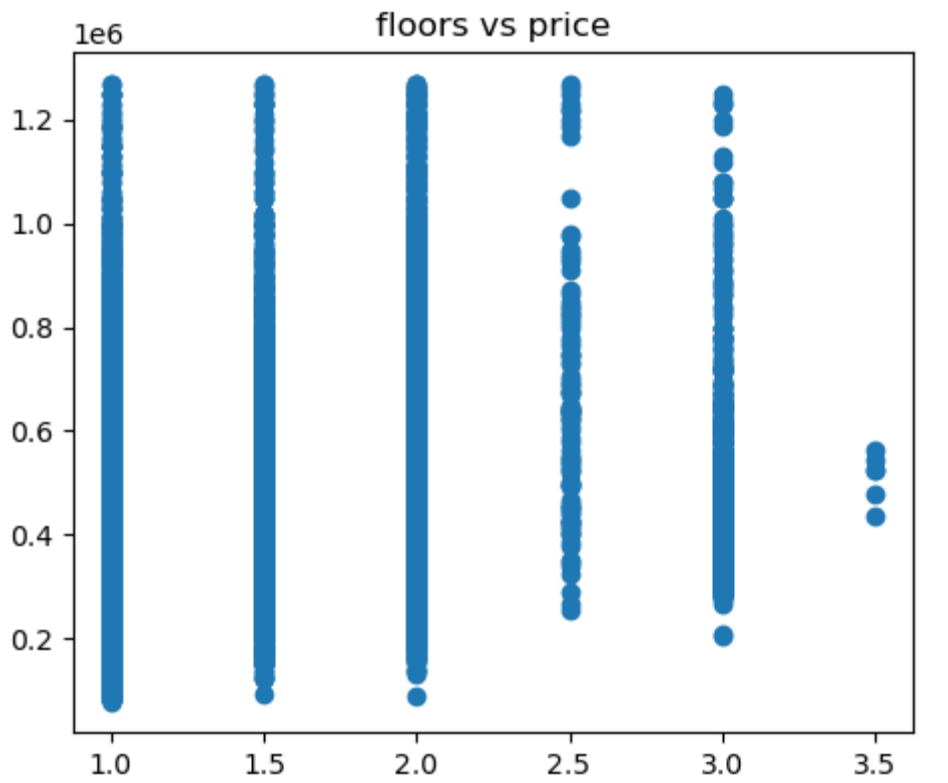
In [123…
```
#creating a new datatframe df2 and dropping the columns zipcode,lat and long
df2=df1.iloc[:,2:21]
df2
df2=df2.drop(['zipcode','lat','long'],axis=1)
```

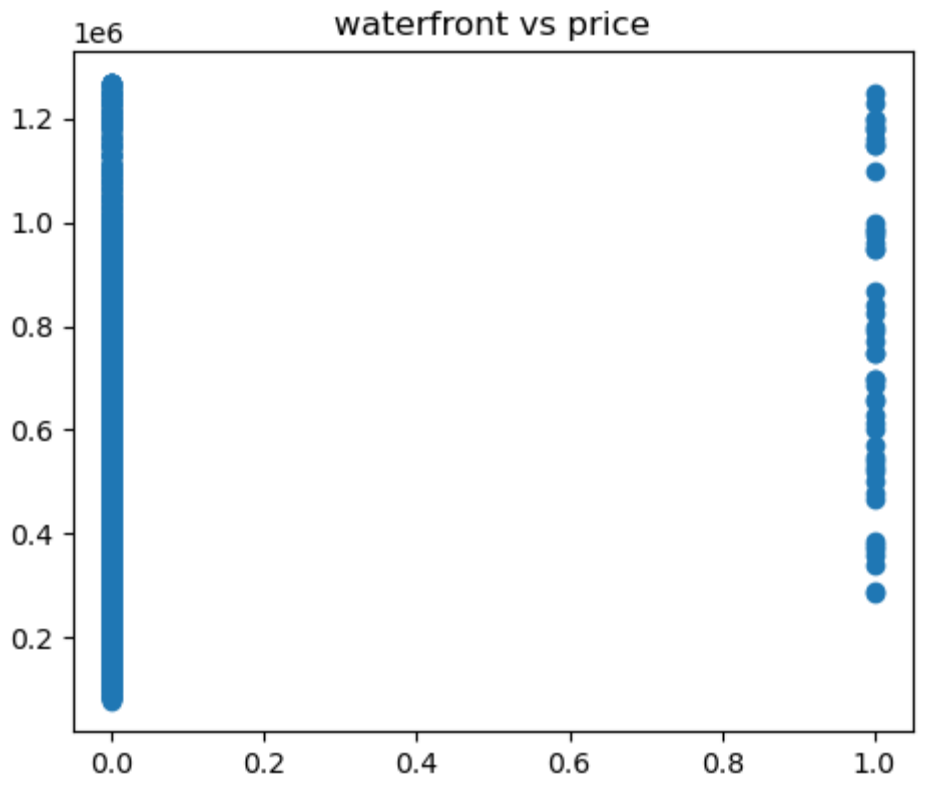
Plotting the scatter plot for price with each variable to understand the trends

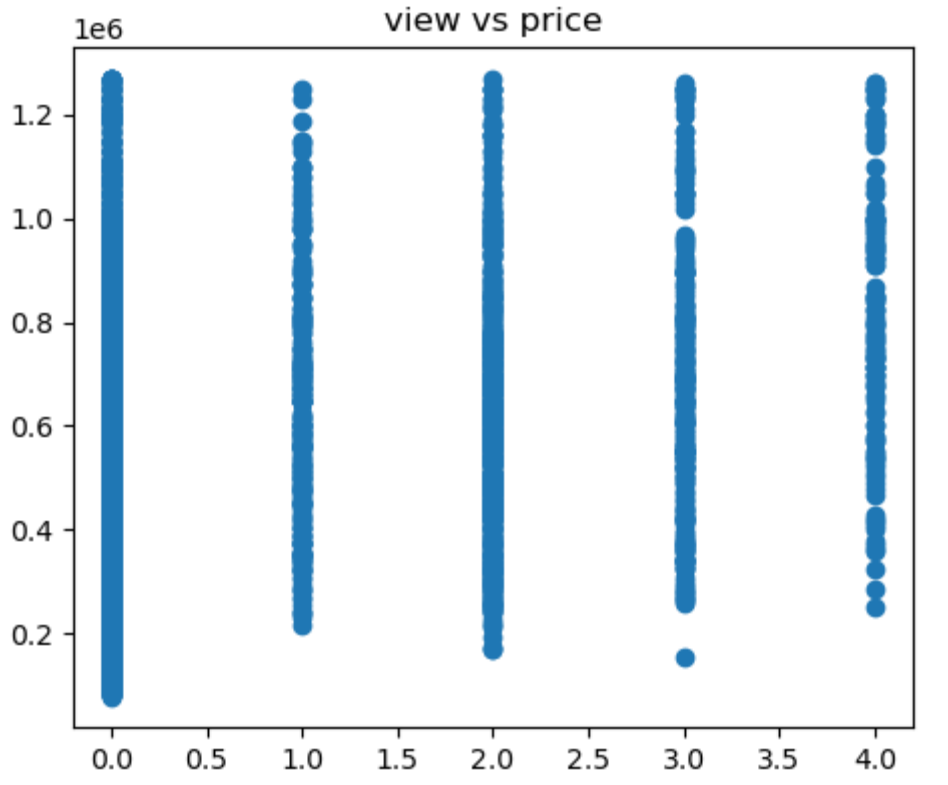
In [124…
```
fig, axes = plt.subplots(nrows=len(df2.columns)-1, ncols=1, figsize=(5, 60))

# Plot scatter plots of price (on Y axis) with each columns (on the X axis)
for i, column in enumerate(df2.columns[1:20]):
    axes[i].scatter(y=df2['price'], x=df2[column])
    axes[i].set_title(f'{column} vs price')


plt.tight_layout()
plt.show()
```


bedrooms vs price
1e6
1.2
1.0
0.8
0.6
0.4
0.2
2
4
6
8
10
bathrooms vs price
1e6
1.2
1.0
0.8
0.6
0.4
0.2
1
2
3
4
5
sqft_living vs price
1e6
1.2
1.0
0.8
0.6
0.4
0.2
500
1000
1500
2000
2500
3000
3500
4000
sqft_lot vs price
1e6
1.2
1.0
0.8
0.6
0.4

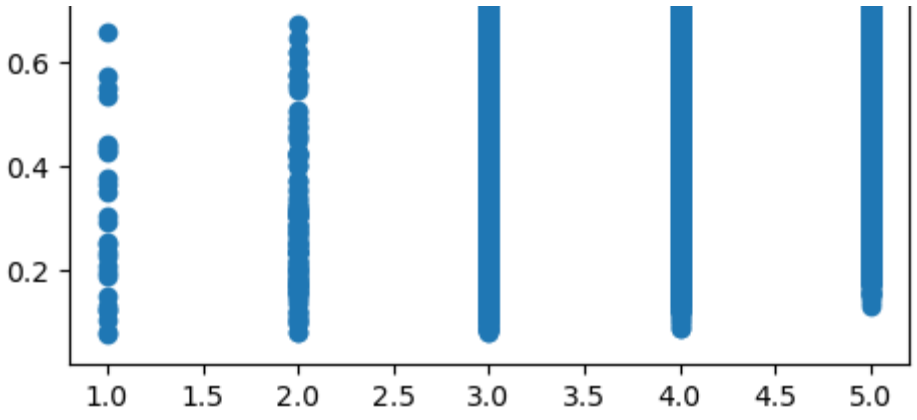

0.2
0
20000
40000
60000
80000
100000

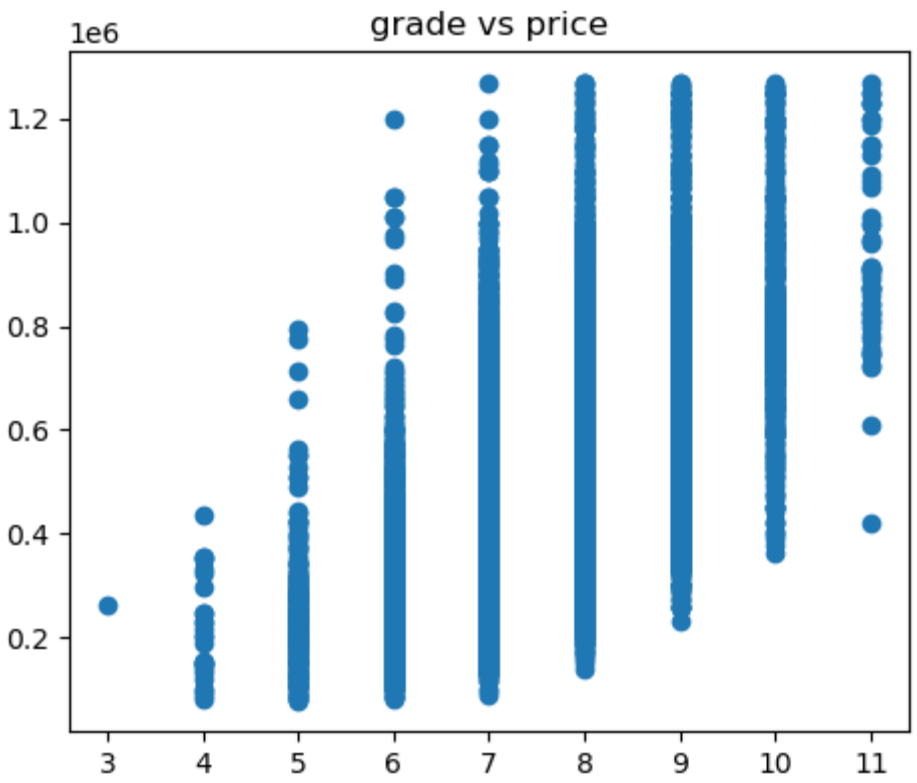

floors vs price
1e6
1.2
1.0
0.8
0.6
0.4
0.2
1.0
1.5
2.0
2.5
3.0
3.5

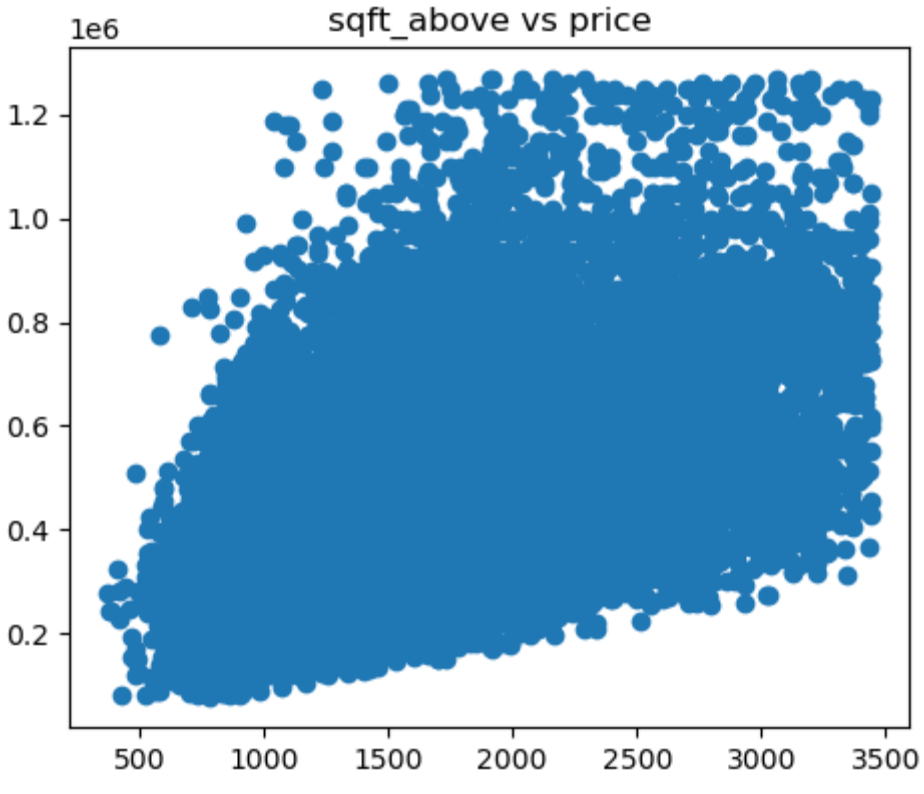

waterfront vs price
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0

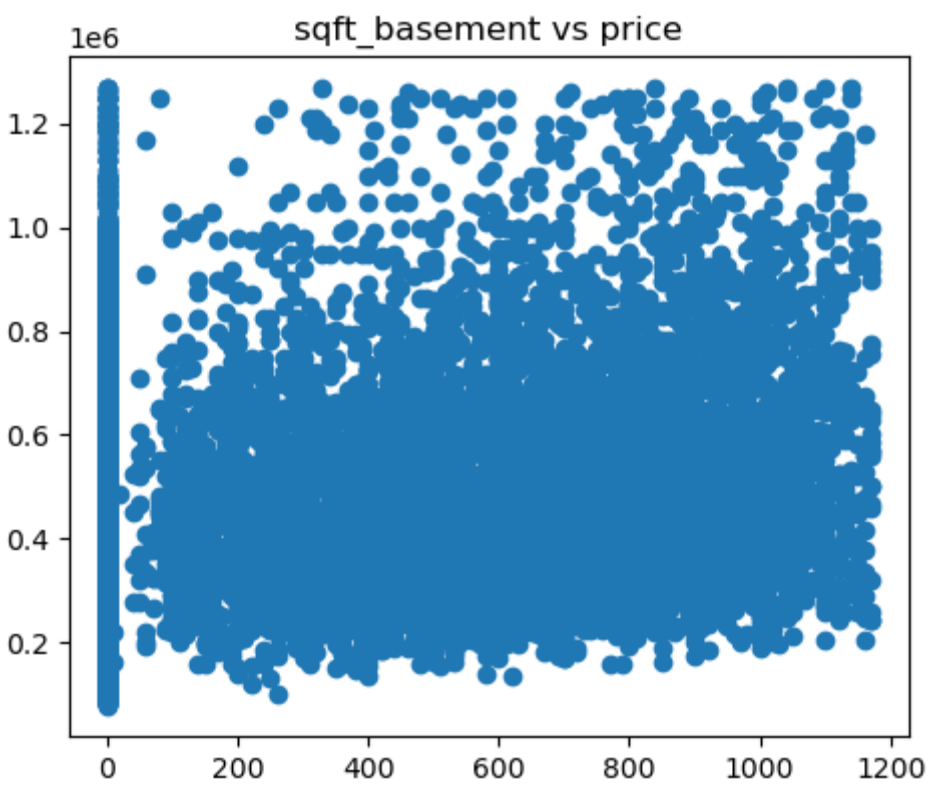

view vs price
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

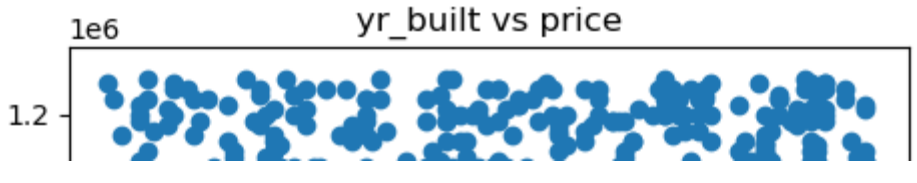

condition vs price
1e6
1.2
1.0
0.8

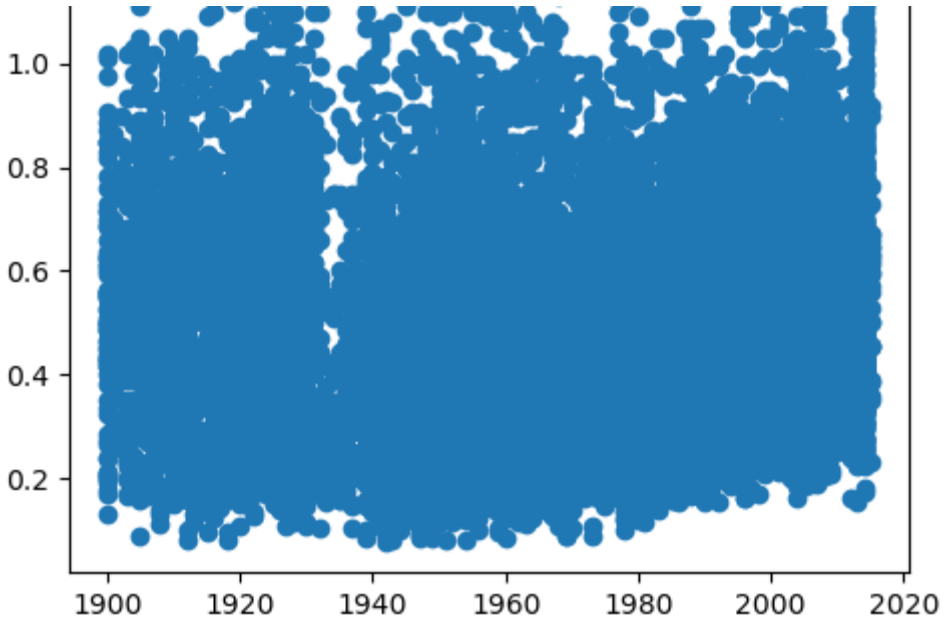

0.6
0.4
0.2
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

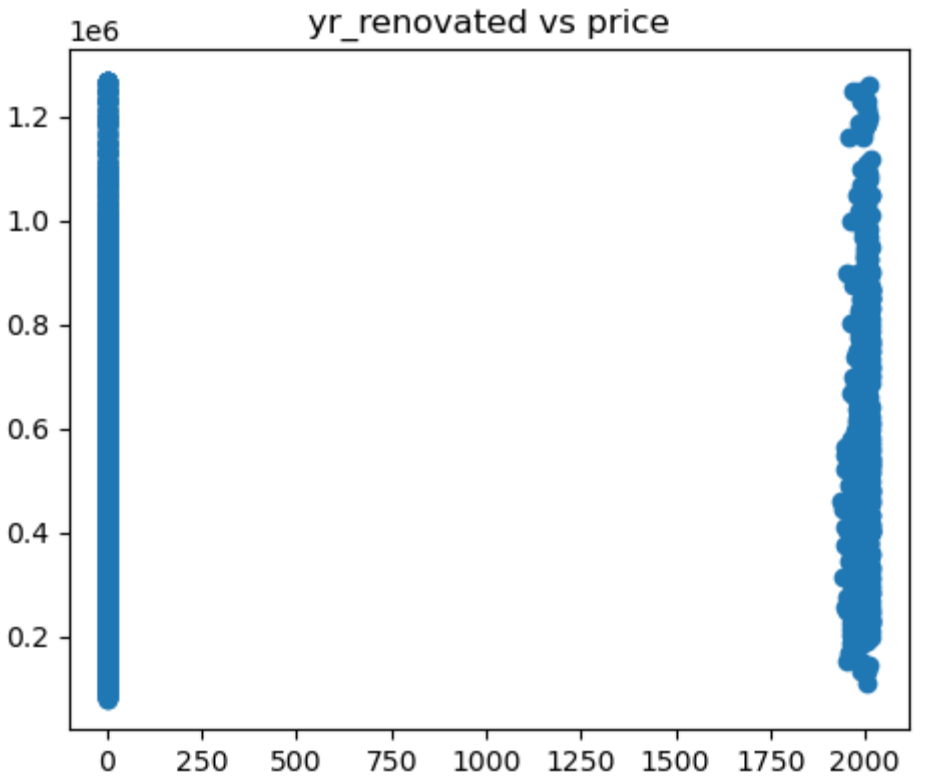

grade vs price
1e6
1.2
1.0
0.8
0.6
0.4
0.2
3
4
5
6
7
8
9
10
11

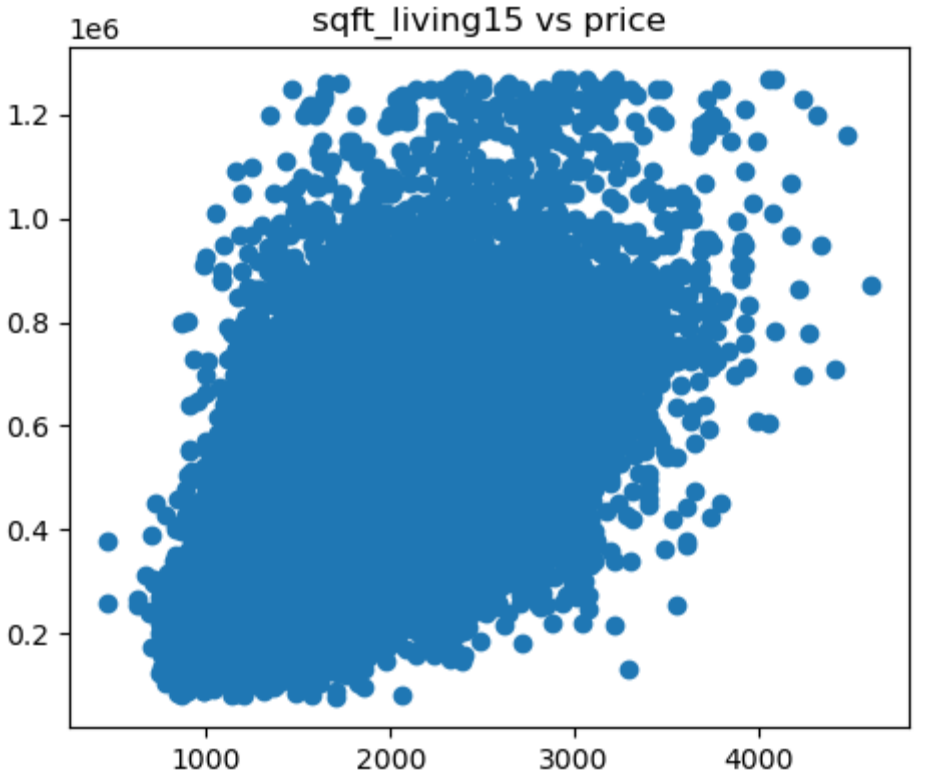

sqft_above vs price
1e6
1.2
1.0
0.8
0.6
0.4
0.2
500
1000
1500
2000
2500
3000
3500

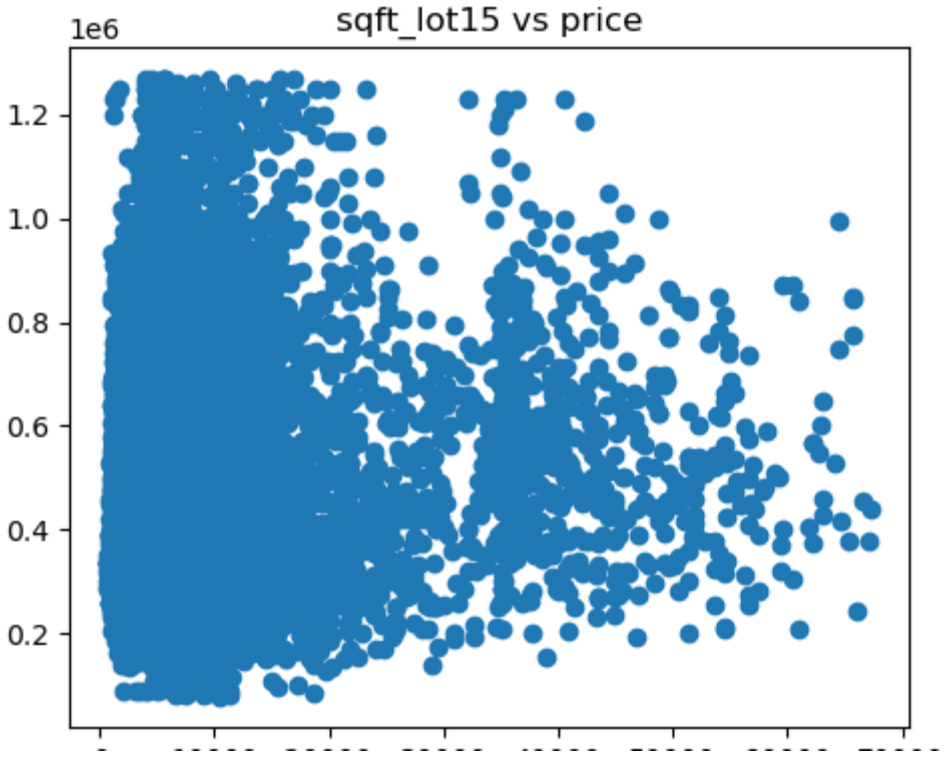

sqft_basement vs price
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0
200
400
600
800
1000
1200


yr_built vs price
1e6
1.2


1.0
0.8
0.6
0.4
0.2
1900
1920
1940
1960
1980
2000
2020


yr_renovated vs price
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0
250
500
750
1000
1250
1500
1750
2000


sqft_living15 vs price
1e6
1.2
1.0
0.8
0.6
0.4
0.2
1000
2000
3000
4000


sqft_lot15 vs price
1e6
1.2
1.0
0.8
0.6
0.4
0.2

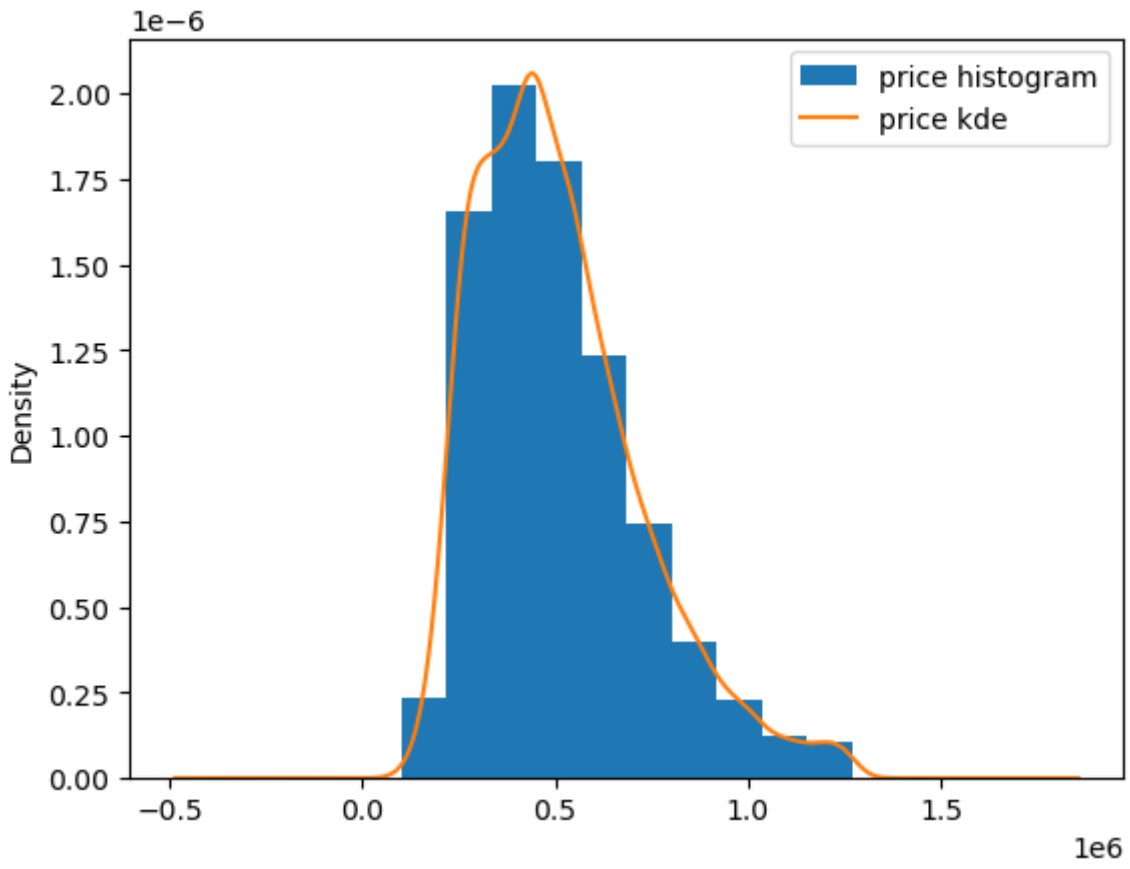
0 10000 20000 30000 40000 50000 60000 70000

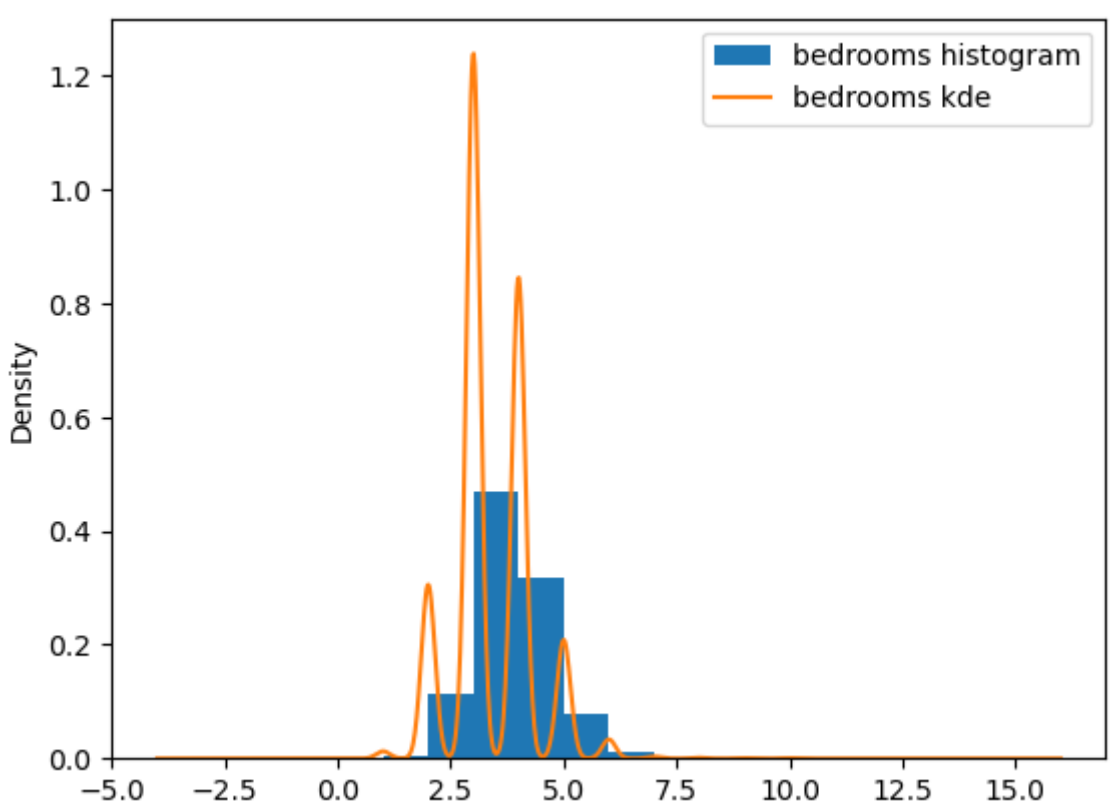
Linear trend can be observed in the variables "sqft_living", "sqft_above", "sqft_living15". Also "sqft_basement" has significant number of 0 values which is used to signify that the house does no have a basement. However the presence of large of 0 will make the data skewed and therefore the 0 values will be removed.

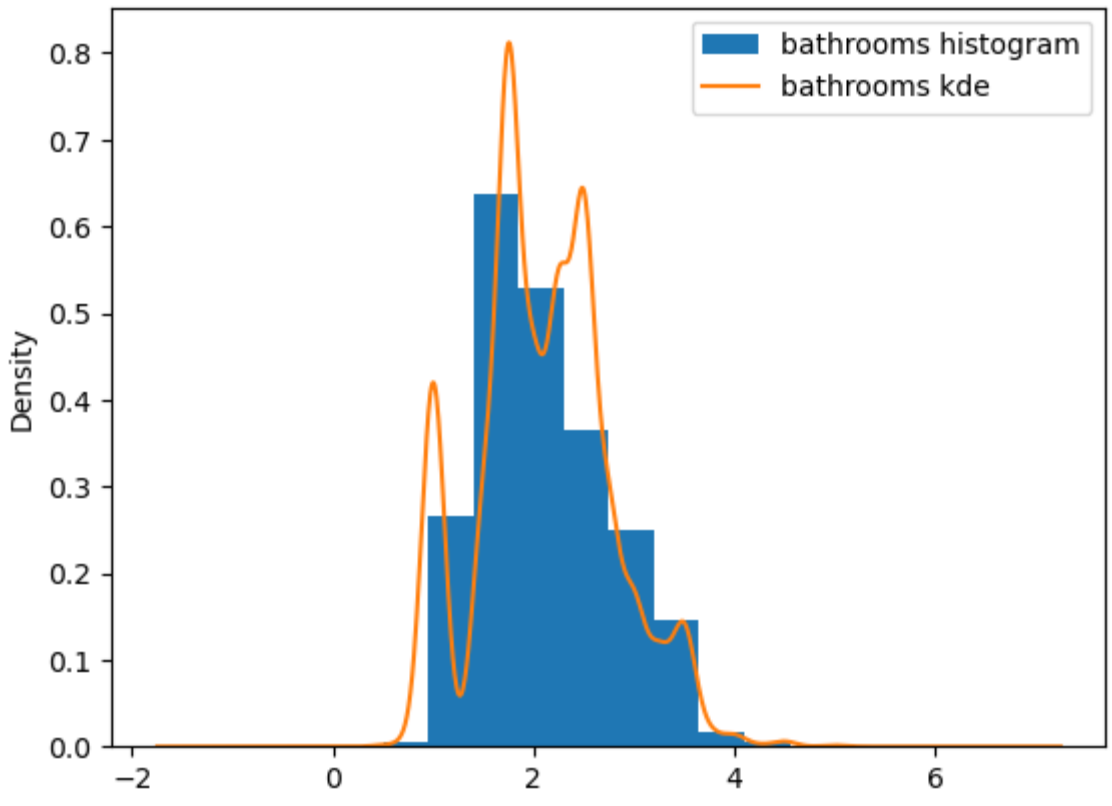
In [108…
```
#removing 0 values from sqft_basement
df2=df2.drop(df2[(df2['sqft_basement']==0)].index)
```

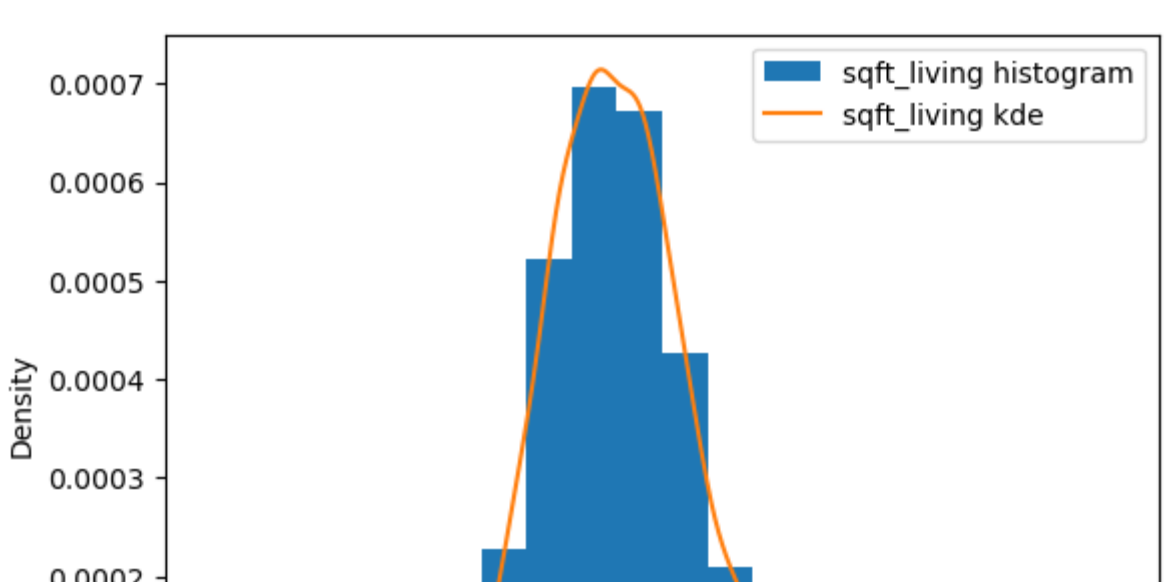
Plottting the histogram and the Kernel Density Estimate for each parameters to understand the distribution of data

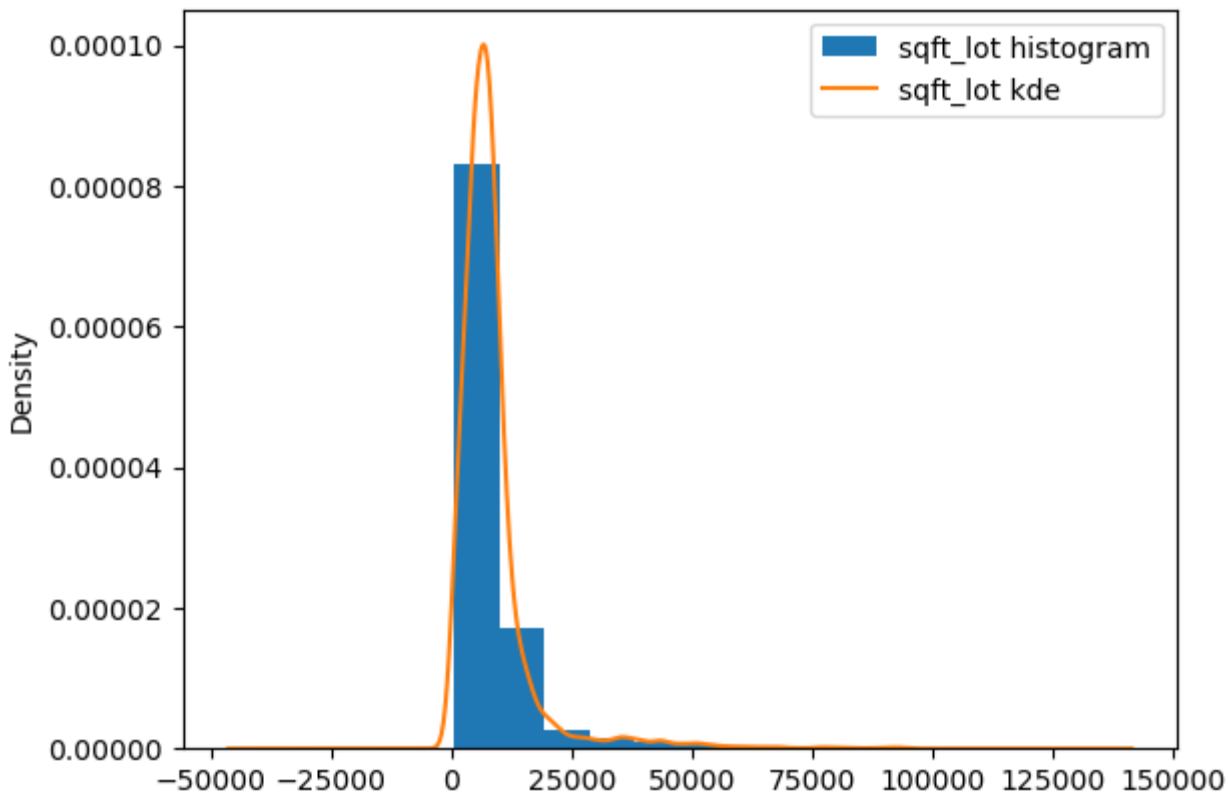
In [108…
```
for column in df2:
    df2[column].plot.hist(density=True, label = column+' histogram')
    df2[column].plot.kde(label =column+' kde')
    plt.legend()
    plt.show()
```

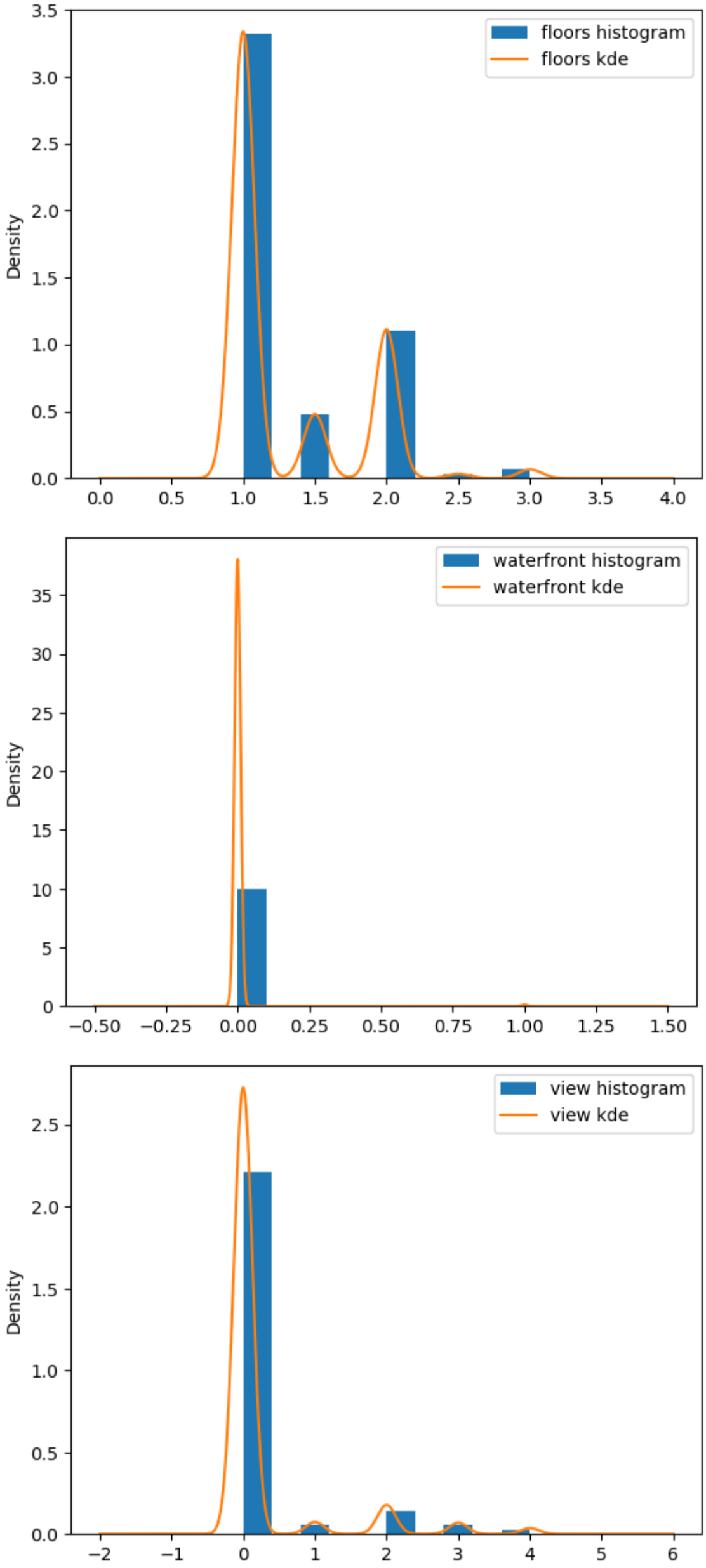


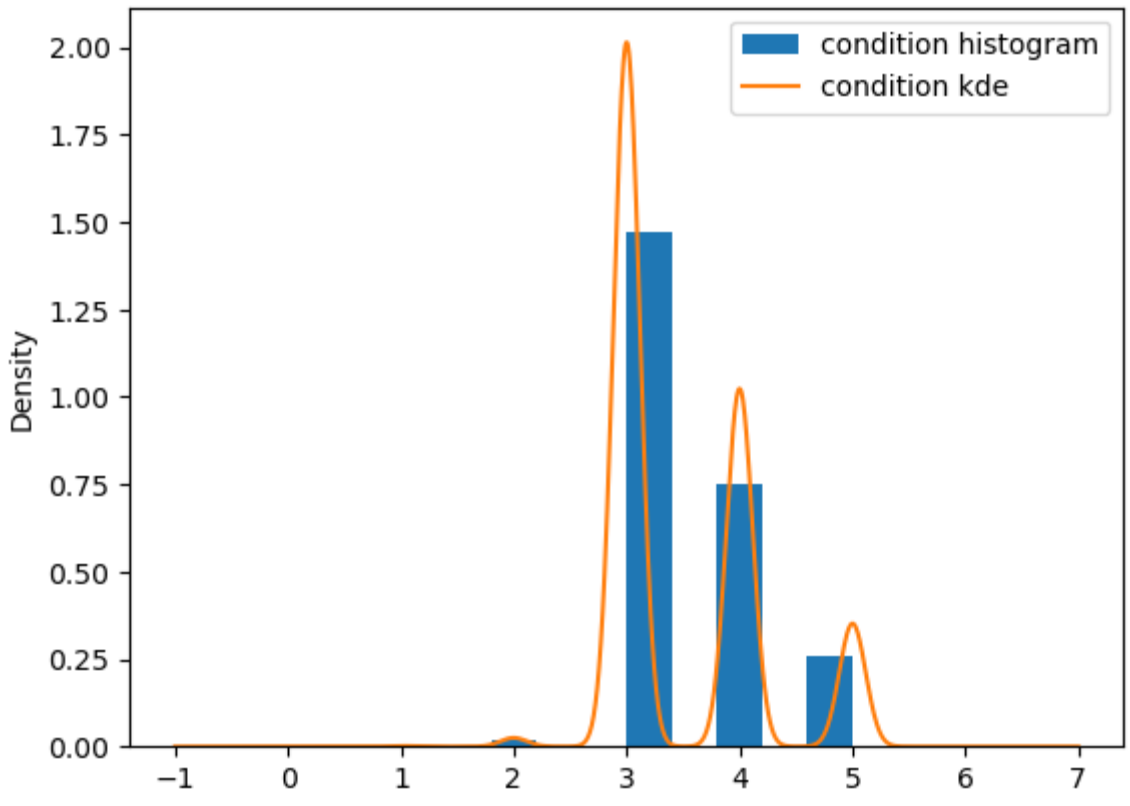


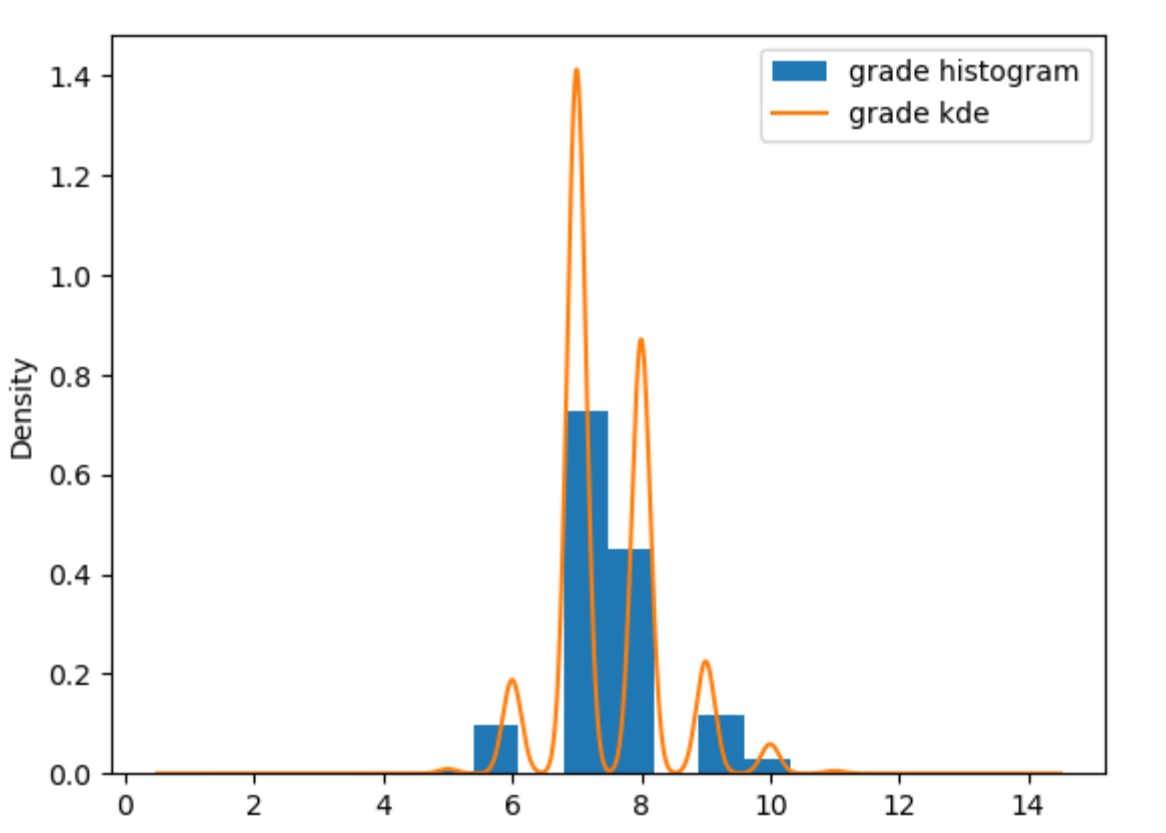

bathrooms histogram
bathrooms kde
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
−2
0
2
4
6

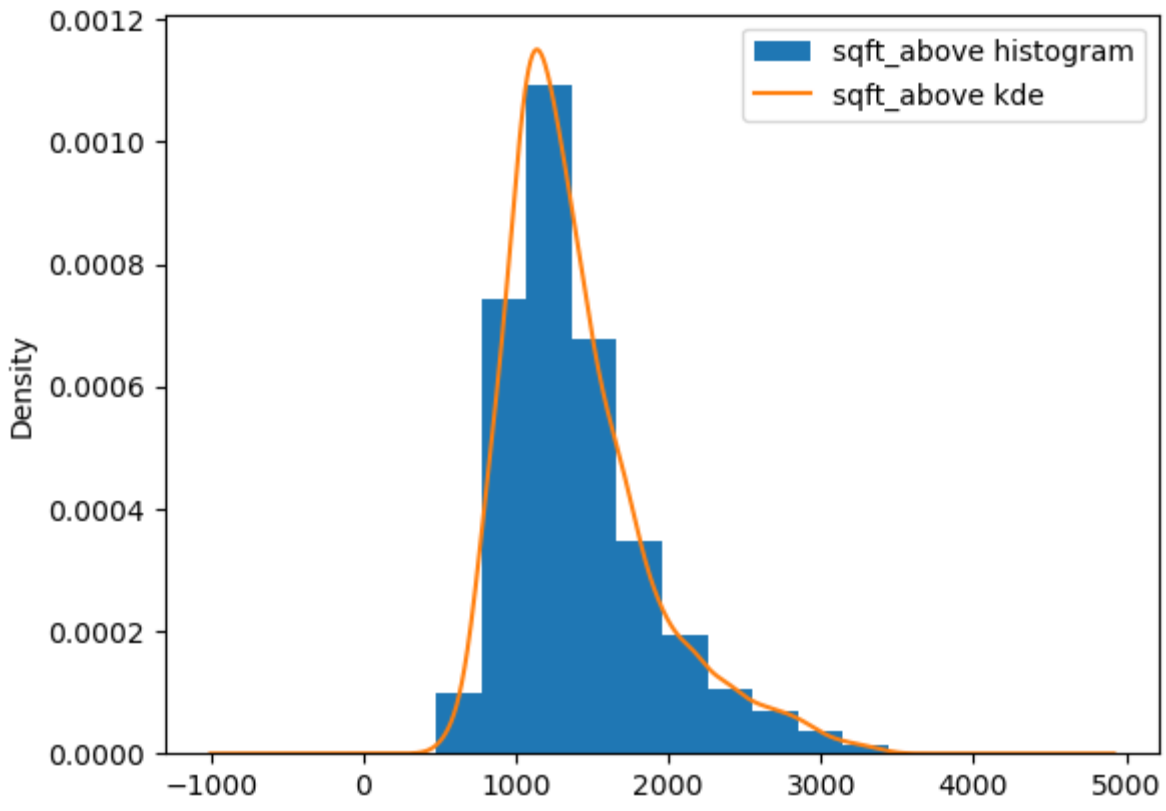

sqft_living histogram
sqft_living kde
Density
0.0000
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006
0.0007
−1000
0
1000
2000
3000
4000
5000

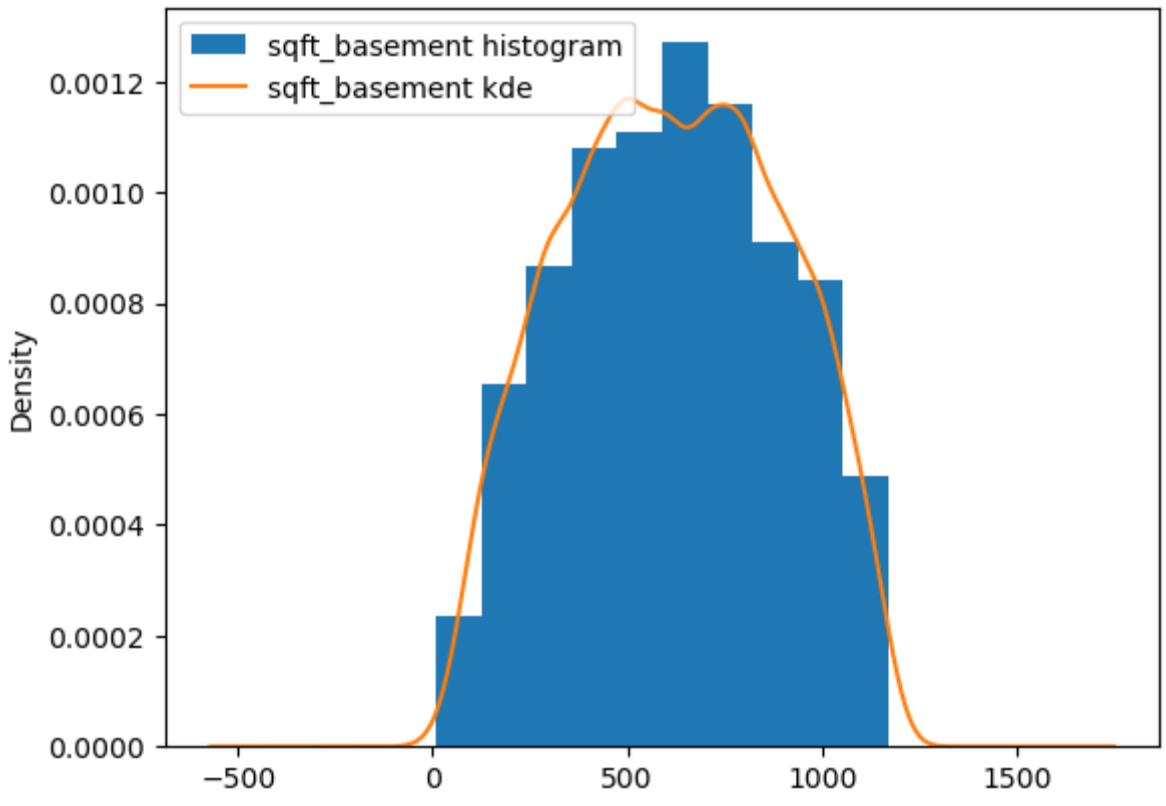

sqft_lot histogram
sqft_lot kde
Density
0.00000
0.00002
0.00004
0.00006
0.00008
0.00010
−50000
−25000
0
25000
50000
75000
100000
125000
150000

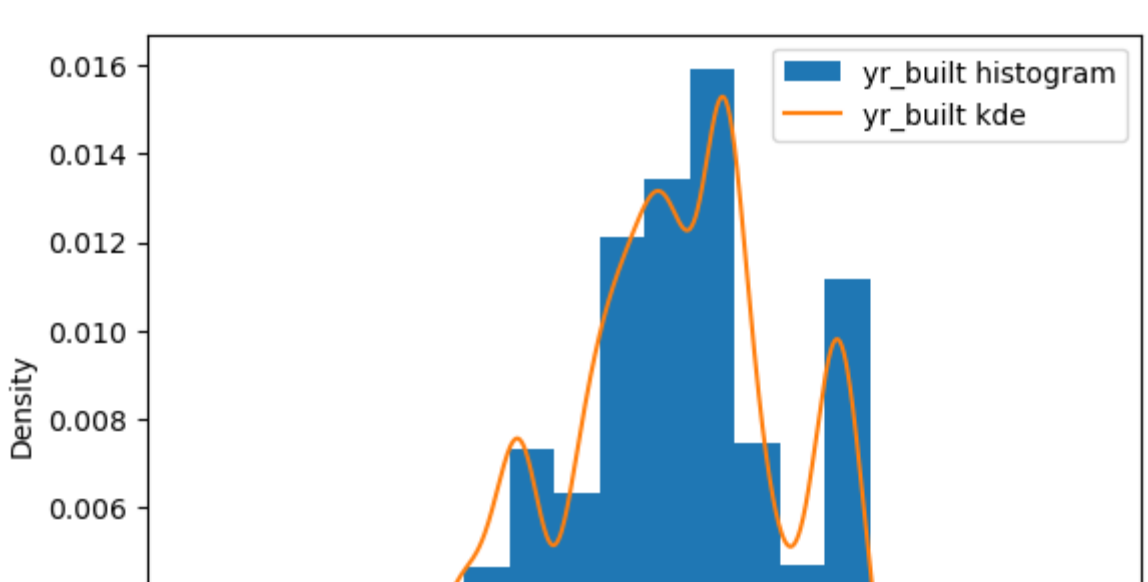

floors histogram
floors kde
Density
waterfront histogram
waterfront kde
Density
view histogram
view kde
Density

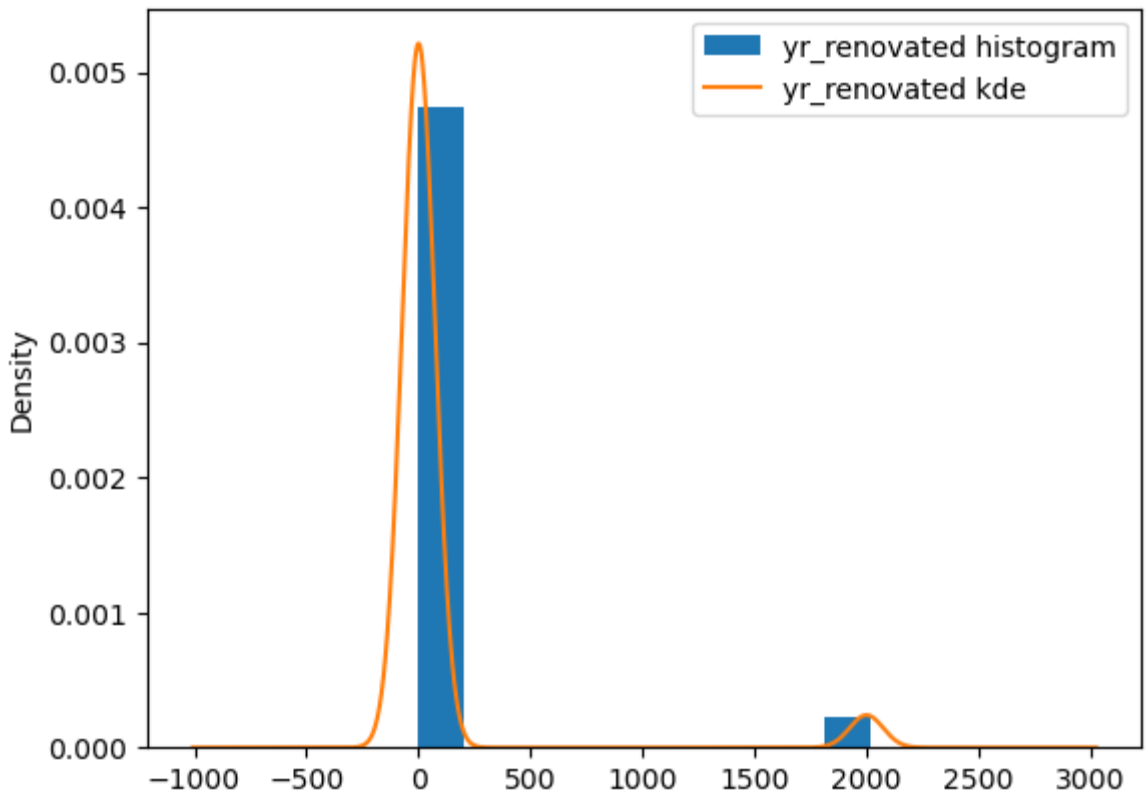

condition histogram
condition kde
Density
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−1
0
1
2
3
4
5
6
7

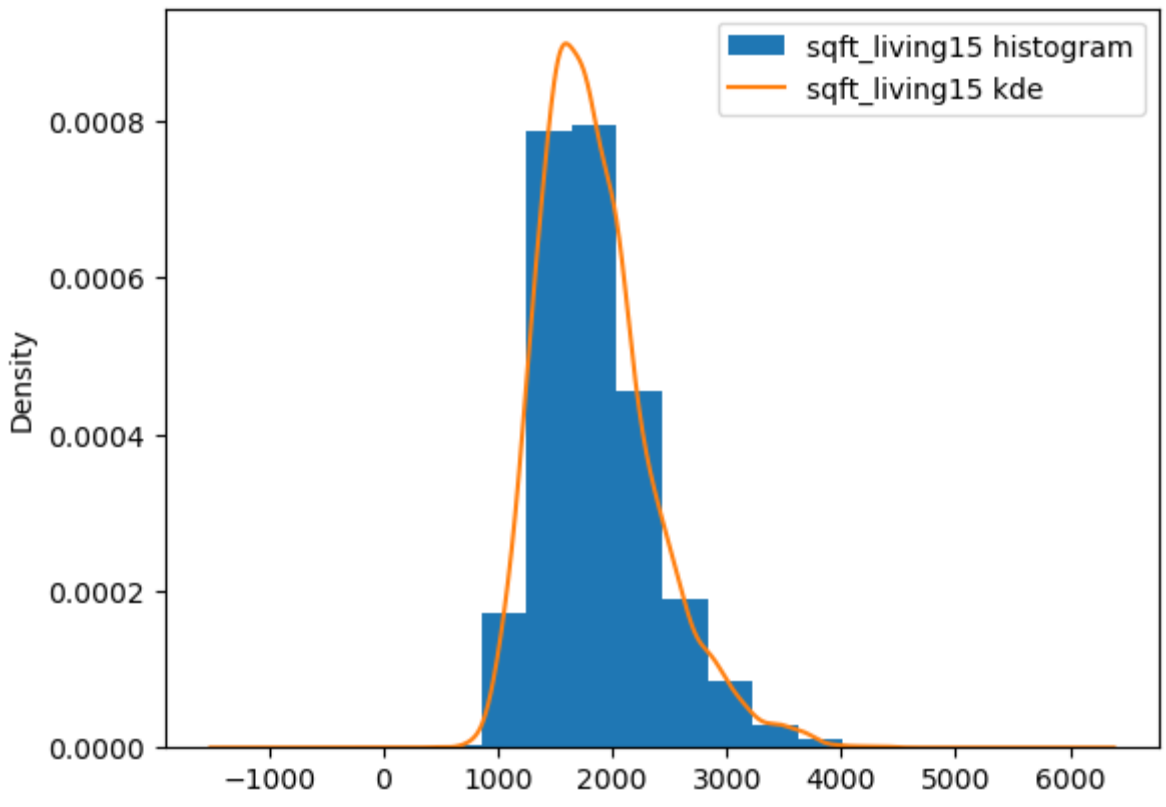

grade histogram
grade kde
Density
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
14

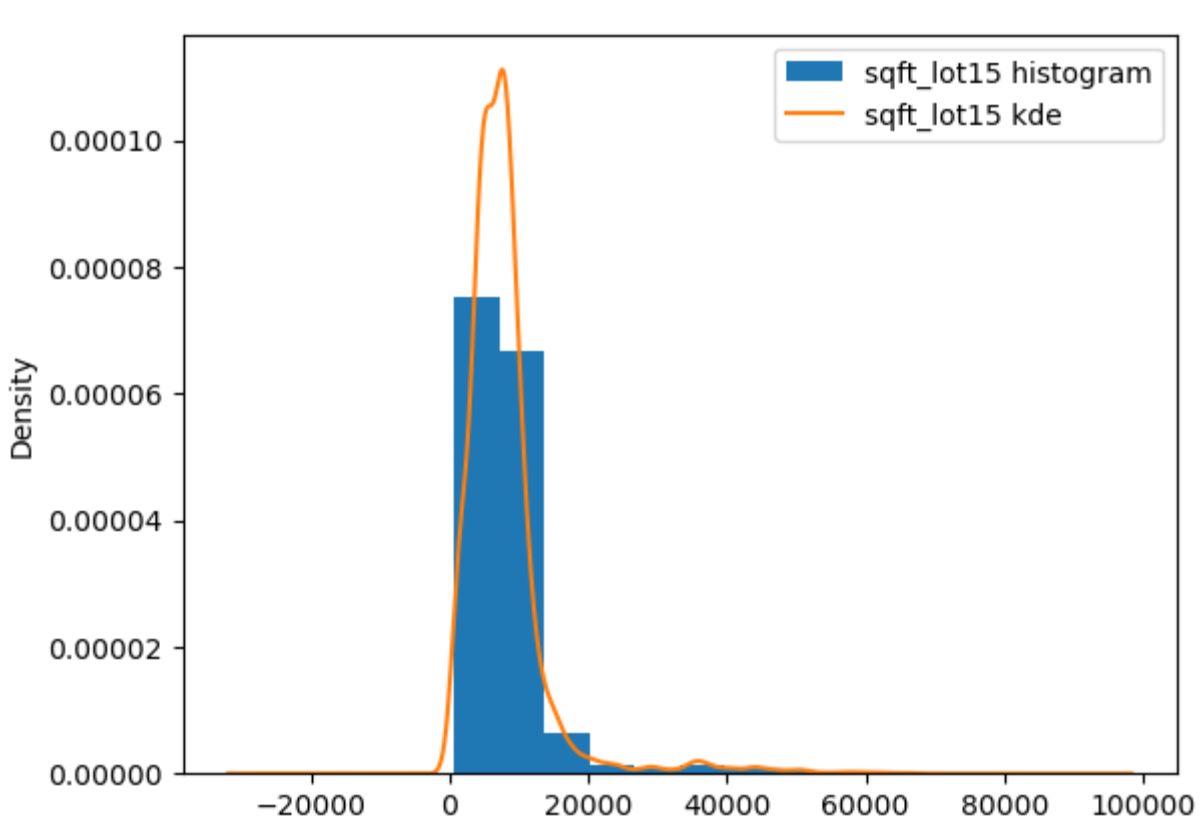

sqft_above histogram
sqft_above kde
Density
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
−1000
0
1000
2000
3000
4000
5000


sqft_basement histogram
sqft_basement kde
Density
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
−500
0
500
1000
1500


yr_built histogram
yr_built kde
Density
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
1850
1900
1950
2000
2050


yr_renovated histogram
yr_renovated kde
Density
0.005
0.004
0.003
0.002
0.001
0.000
−1000
−500
0
500
1000
1500
2000
2500
3000





The distribution of the continuous variables, price, sqft_living, sqft_above, sqft_lot, sqft_living15 and sqft_lot15 looks close to normal distribution however some of the variables are skewed to the the left. The parameter sqft_basement after removal of the 0s, is close to normal distribution.

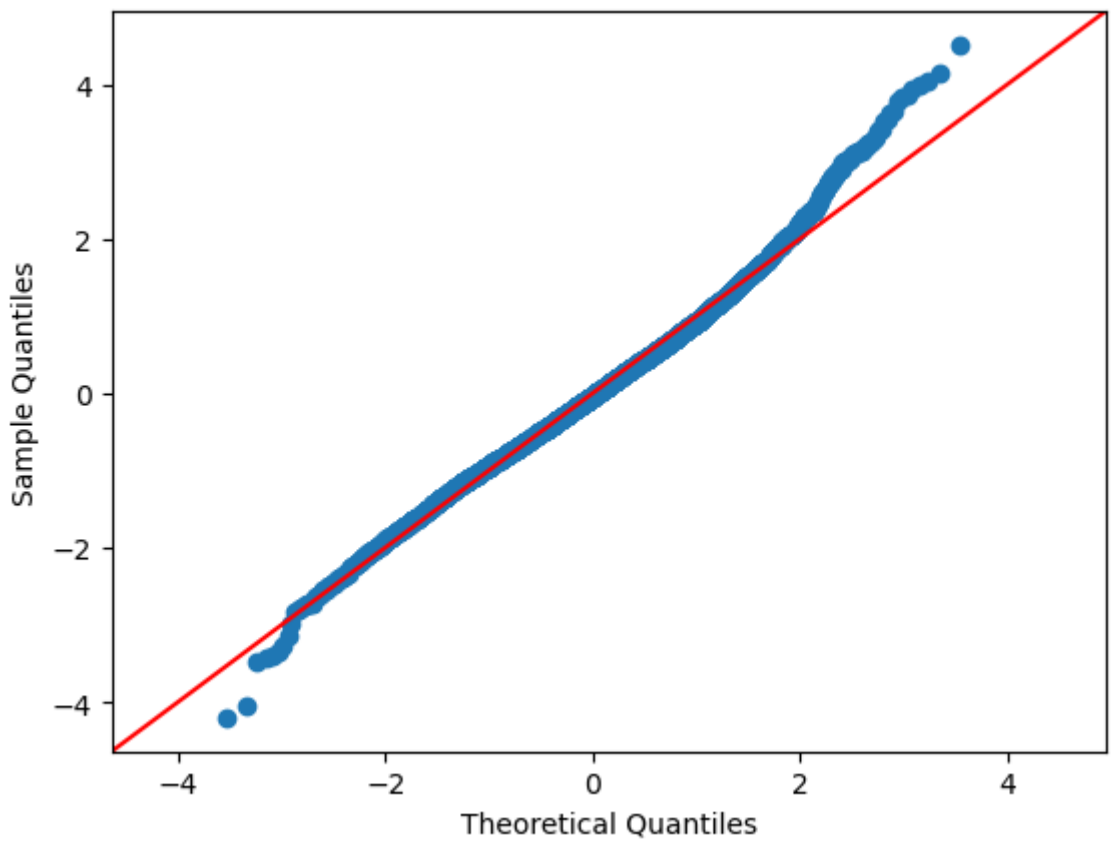
Iteration 1

Running the statsmodel as part of first iteration, plotting the Q-Q plot and the residual plots. The significance of the model and if the models satisfies the assumptions of the linear regression theory will also be checked.

In [108…

```
outcome='price'
x_cols=df2.iloc[:,1:20]
predictors='+'.join(x_cols)
formula=outcome+'~'+predictors
model_itr1=smf.ols(formula=formula,data=df2).fit()
print(model_itr1.summary())

residuals = model_itr1.resid
sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True)
plt.show()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.563
Model:                            OLS   Adj. R-squared:                  0.562
Method:                 Least Squares   F-statistic:                     449.0
Date:                Sat, 02 Dec 2023   Prob (F-statistic):               0.00
Time:                        08:59:39   Log-Likelihood:                -64908.
No. Observations:                4889   AIC:                         1.298e+05
Df Residuals:                    4874   BIC:                         1.299e+05
Df Model:                          14
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    4.654e+06   1.85e+05     25.159      0.000    4.29e+06    5.02e+06
bedrooms    -1.232e+04   2831.520     -4.352      0.000   -1.79e+04   -6772.395
bathrooms    2.201e+04   4556.691      4.830      0.000    1.31e+04    3.09e+04
sqft_living    45.5252      4.076     11.168      0.000      37.534      53.516
sqft_lot        0.0584      0.384      0.152      0.879      -0.695       0.812
floors       7.623e+04   6186.661     12.321      0.000    6.41e+04    8.84e+04
waterfront   1.384e+05   3.35e+04      4.126      0.000    7.26e+04    2.04e+05
view         2.354e+04   2965.739      7.937      0.000    1.77e+04    2.94e+04
condition     2.19e+04   3283.019      6.671      0.000    1.55e+04    2.83e+04
grade        9.873e+04   3607.760     27.365      0.000    9.17e+04    1.06e+05
sqft_above     30.5714      5.591      5.468      0.000      19.610      41.533
sqft_basement  14.9538      6.322      2.365      0.018       2.559      27.349
yr_built    -2713.8090     95.362    -28.458      0.000   -2900.761   -2526.858
yr_renovated   12.2441      5.359      2.285      0.022       1.737      22.751
sqft_living15  77.3434      5.881     13.152      0.000      65.815      88.872
sqft_lot15     -3.3516      0.531     -6.315      0.000      -4.392      -2.311
==============================================================================
Omnibus:                      146.730   Durbin-Watson:                   2.007
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              217.362
Skew:                           0.304   Prob(JB):                     6.31e-48
Kurtosis:                       3.835   Cond. No.                     6.08e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.48e-22. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```



From the results of the multivariate regression analysis, the adjusted R square value value was found to be 0.562, which is a good value as we can say that 56% of the variance of the price can be explained by the model. Alos the model is also significant as the F statistic probablity is less than 0.05. All the variables except sqft_lot is also significant as the P<|t| values are less than 0.05. The JB values is quite high denoting non-normality in the residuals. This is also confirmed from the Q-Q plot above which shows deviations from the straight line.

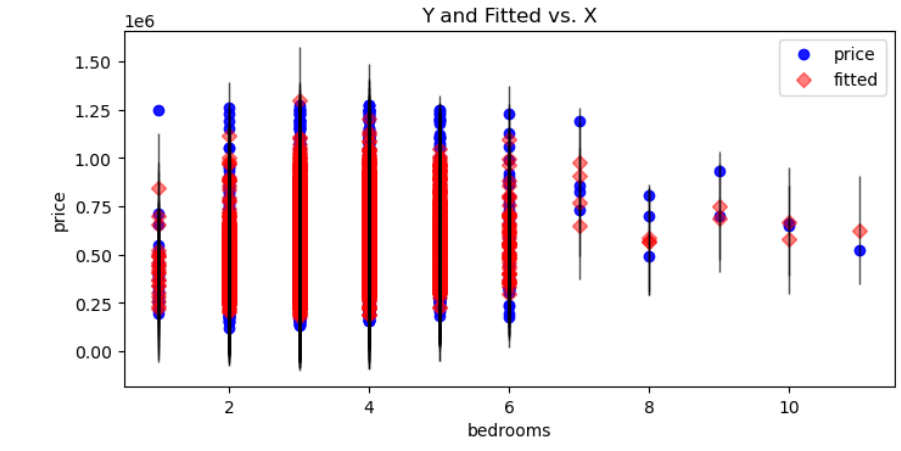
The notes of the results suggest that there could be strong multicollinearity above the variables. This will be checked in the next iteration.

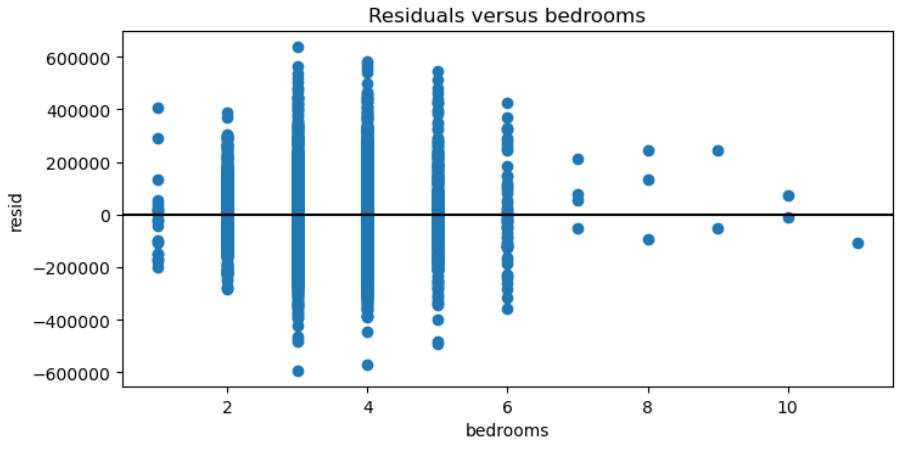
The residuals plots for each variables will now be generated.

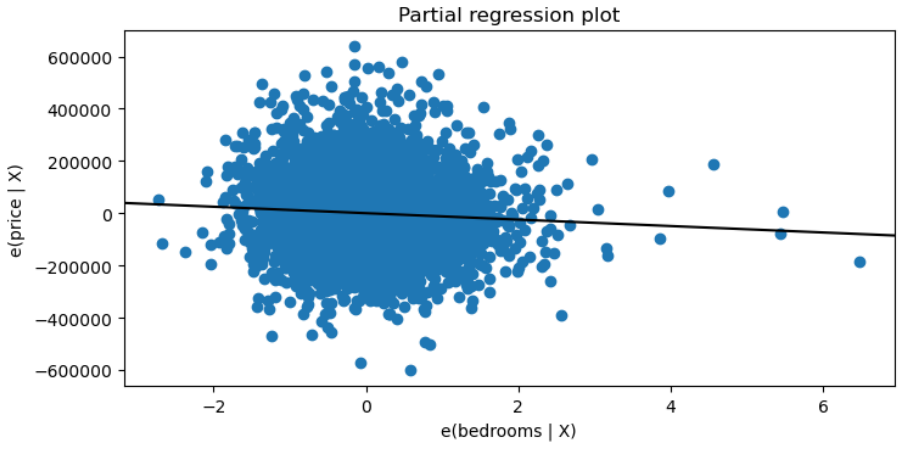
In [108…

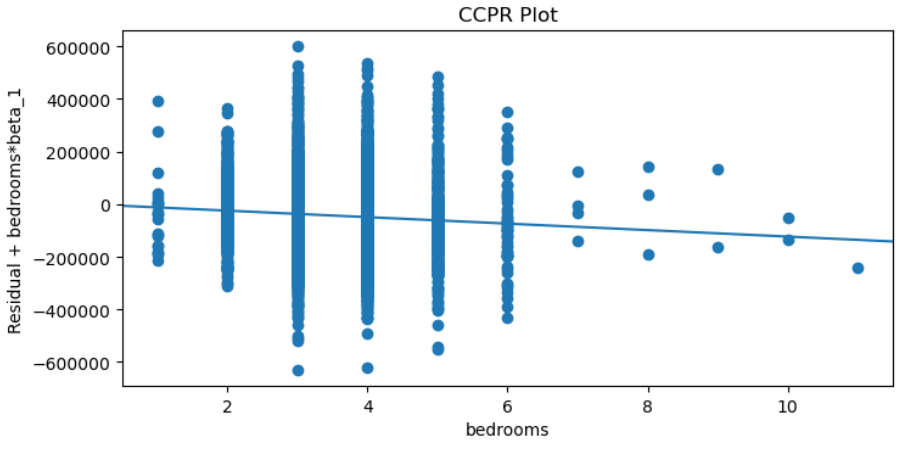
```
#checking for homoscedasticity/heteroscedasticity assumptions
for i, column in enumerate(df2.columns[1:20]):
    fig = plt.figure(figsize=(15,8))
    fig = sm.graphics.plot_regress_exog(model_itr1, column, fig=fig)
```

Regression Plots for bedrooms

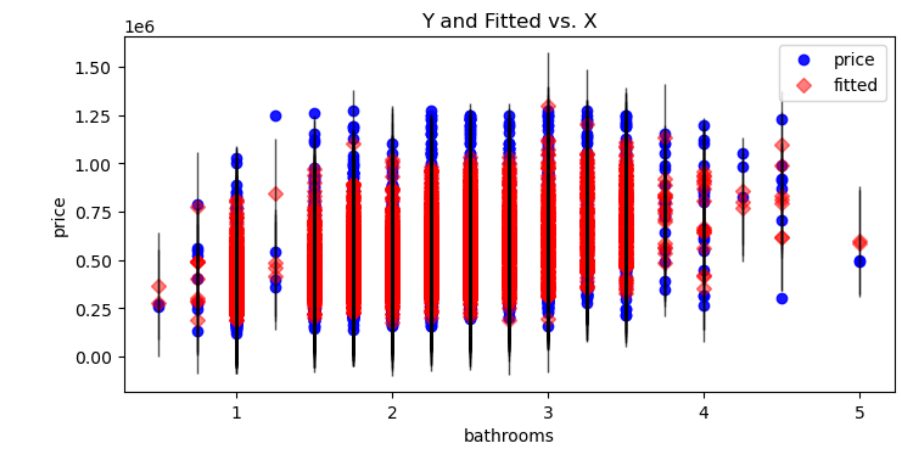


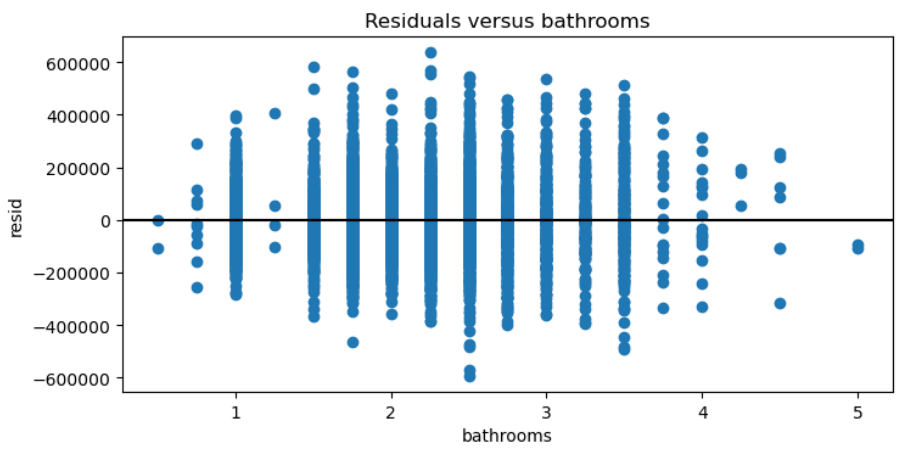


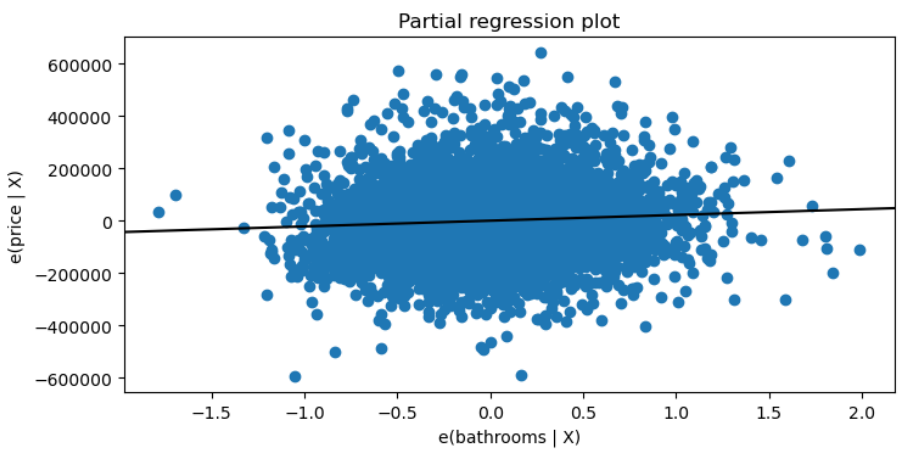


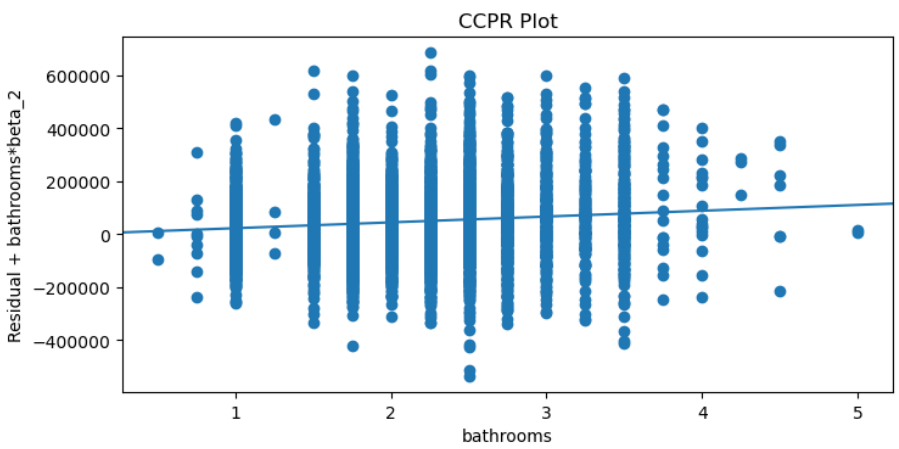


Regression Plots for bathrooms

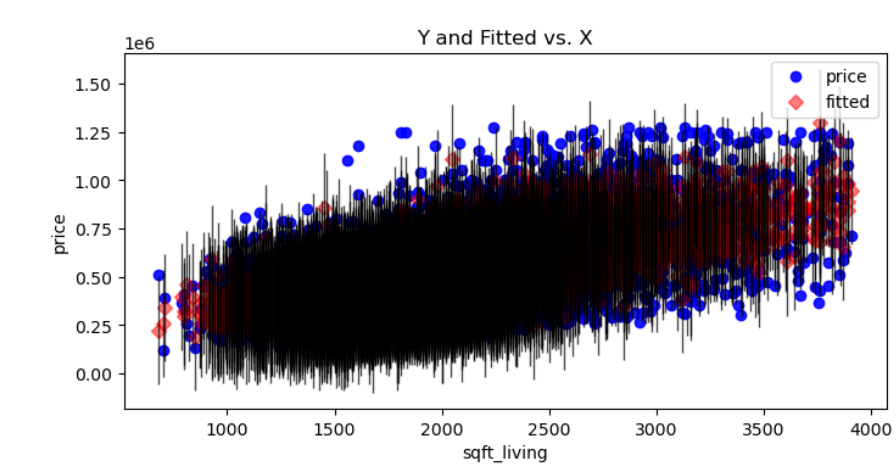


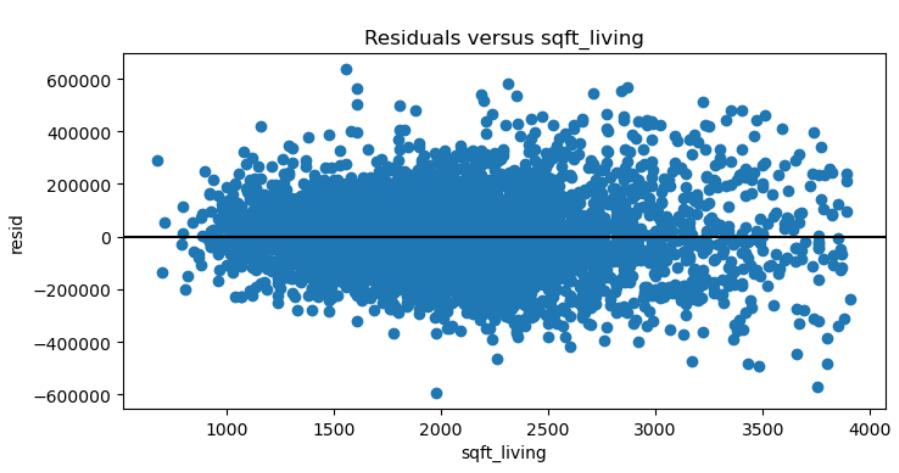


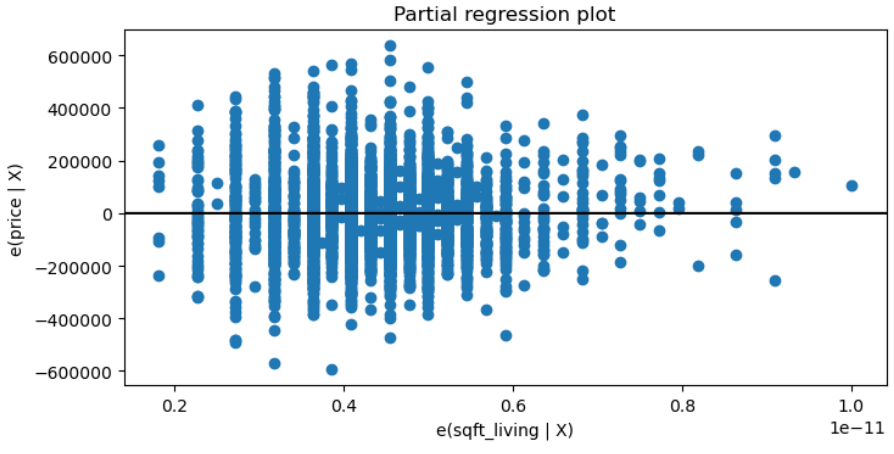


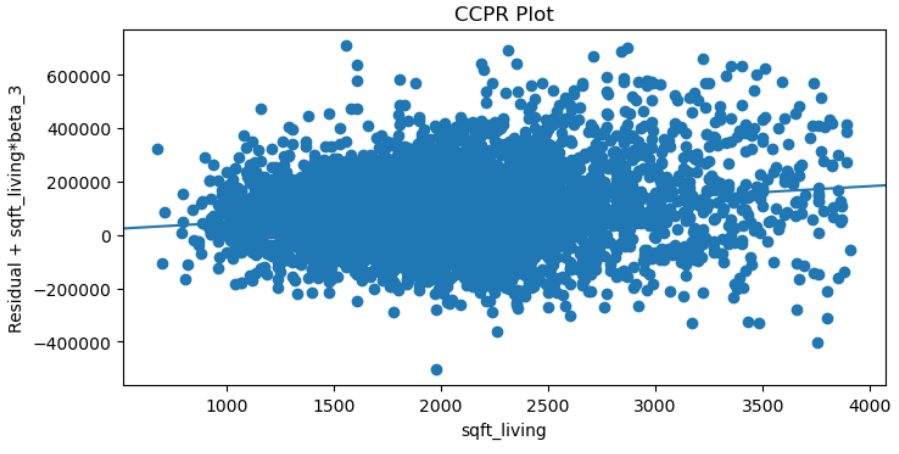


Regression Plots for sqft_living

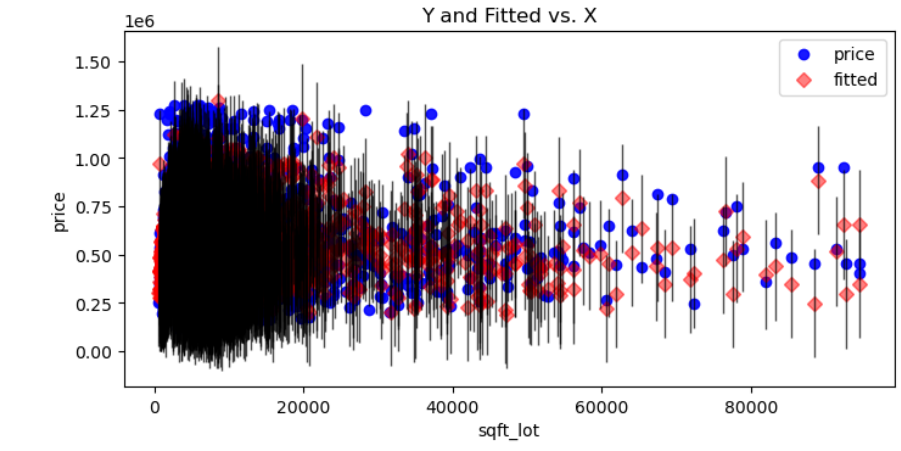


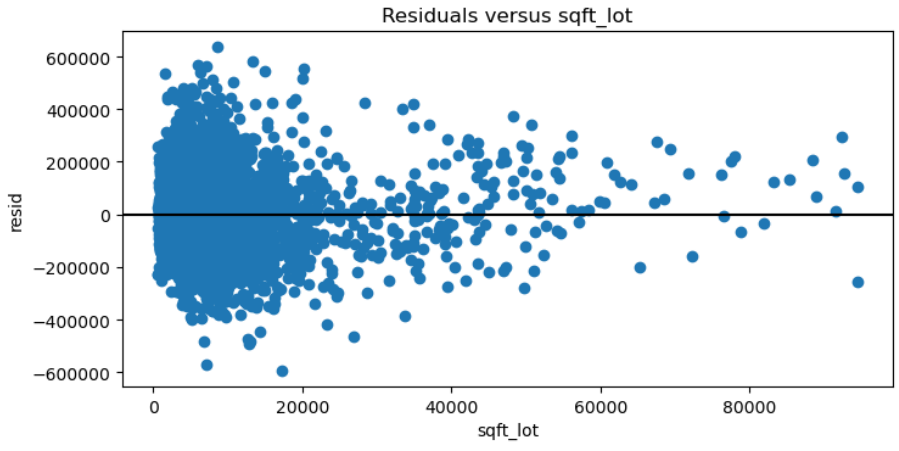


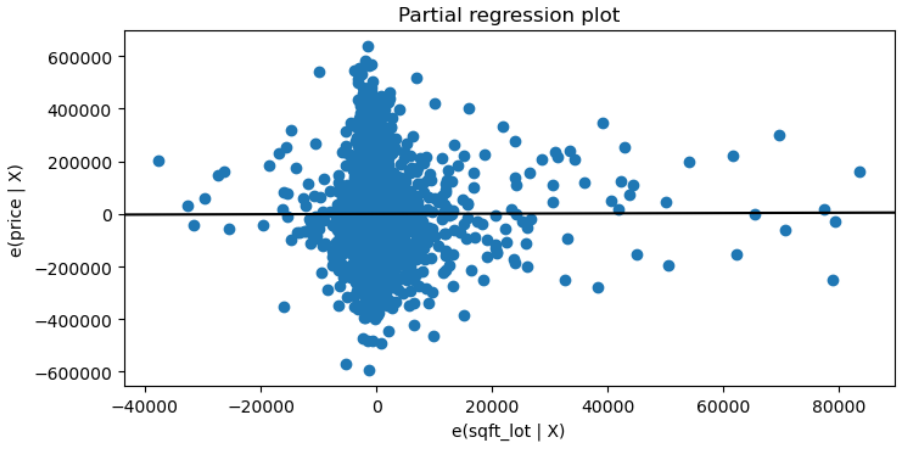


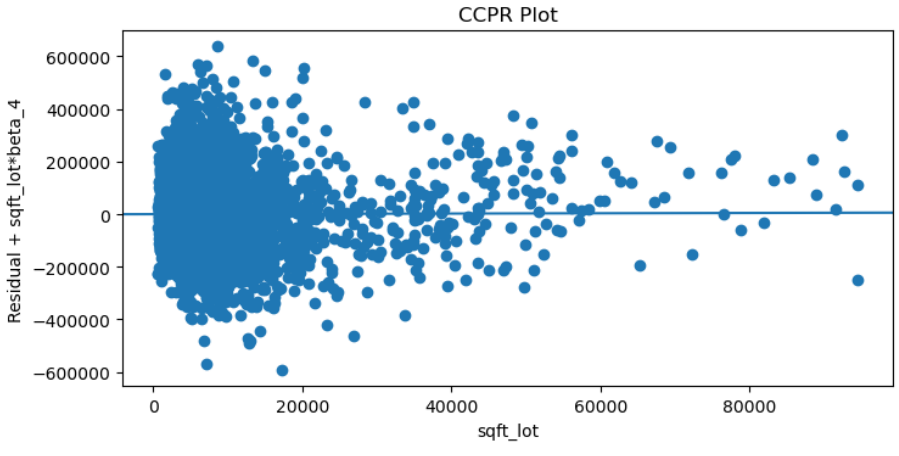


Regression Plots for sqft_lot

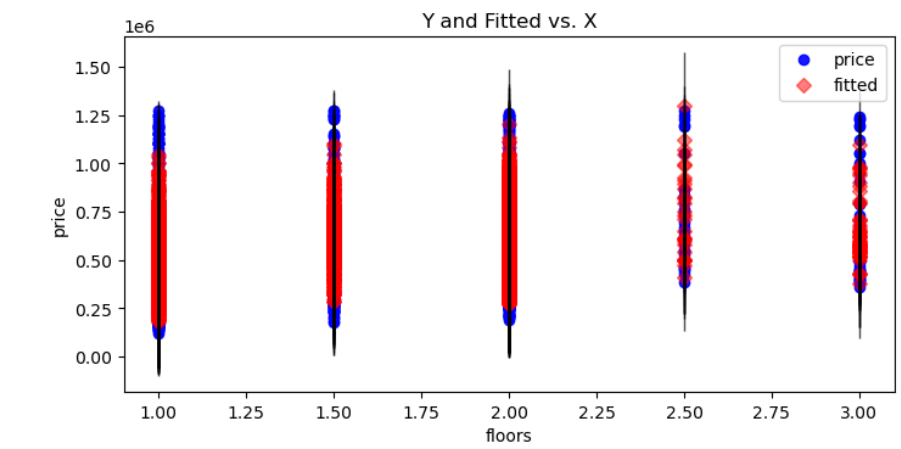


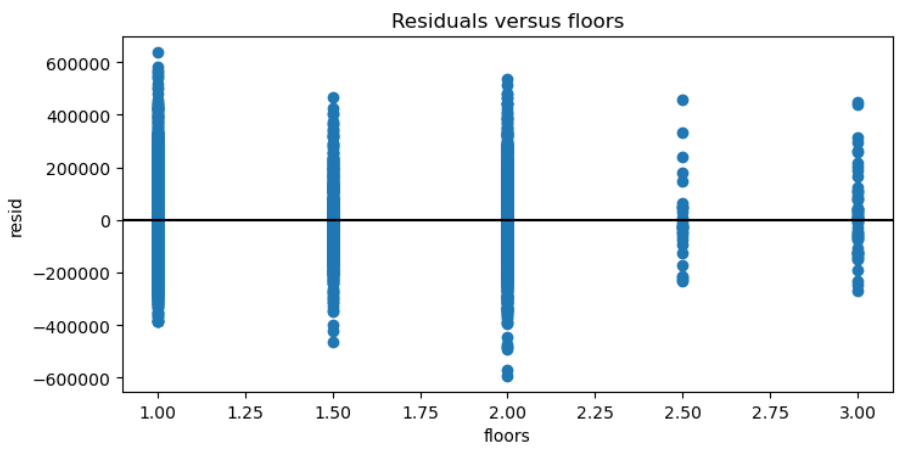


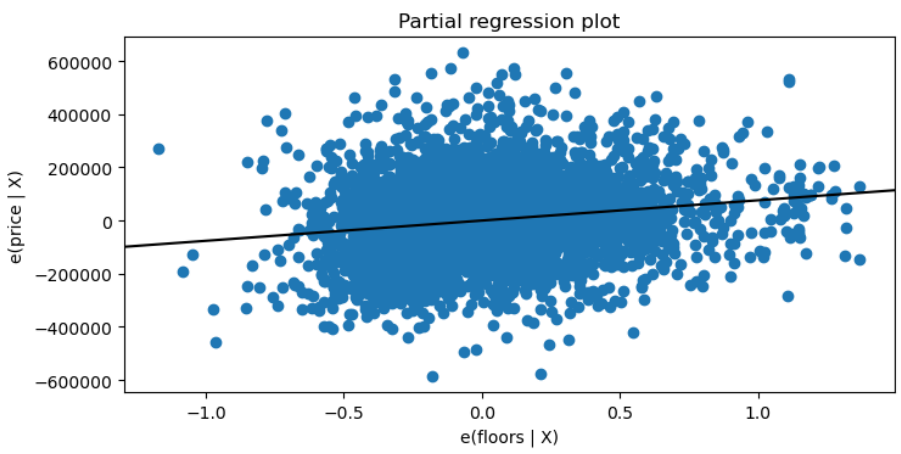


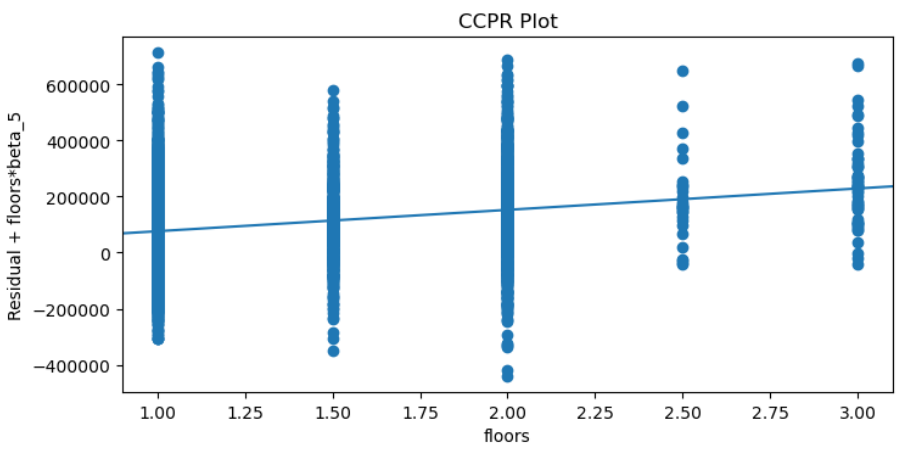


Regression Plots for floors

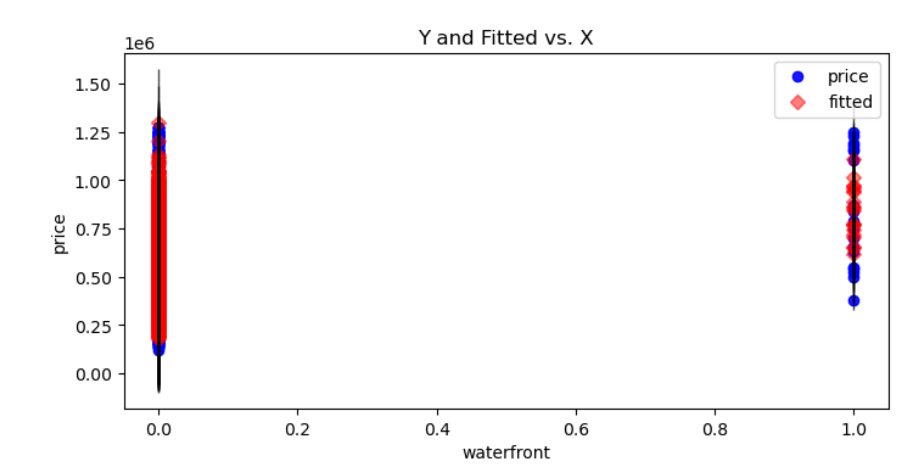


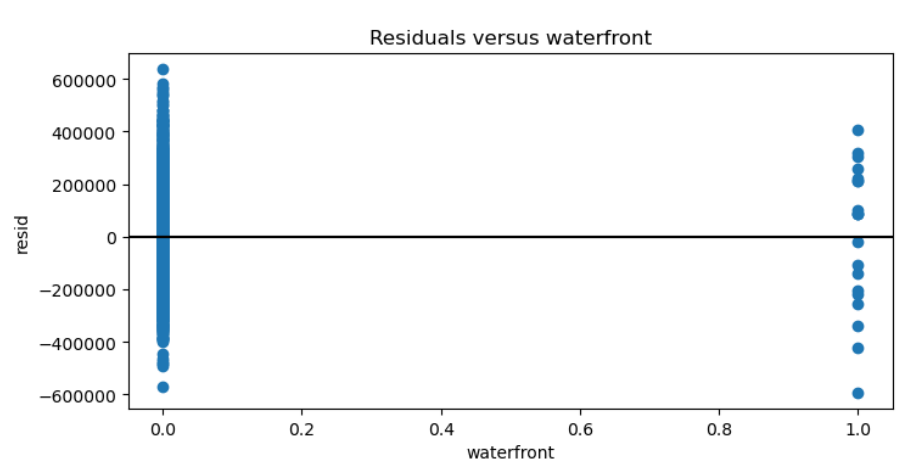


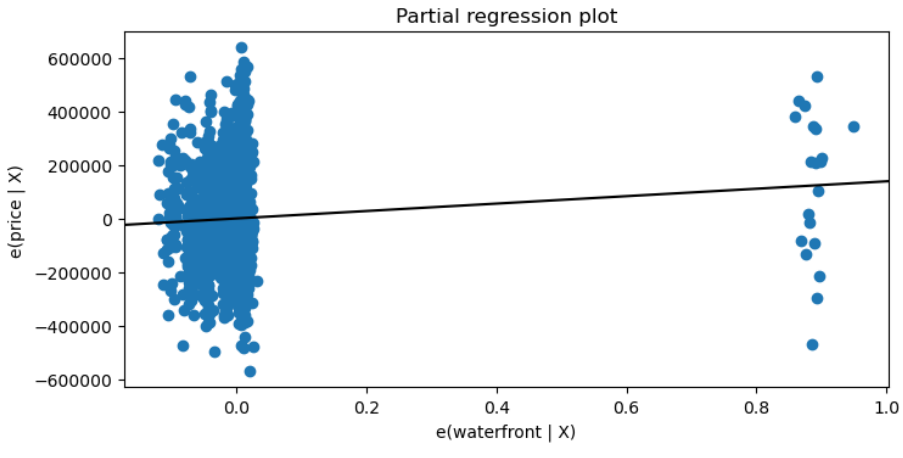


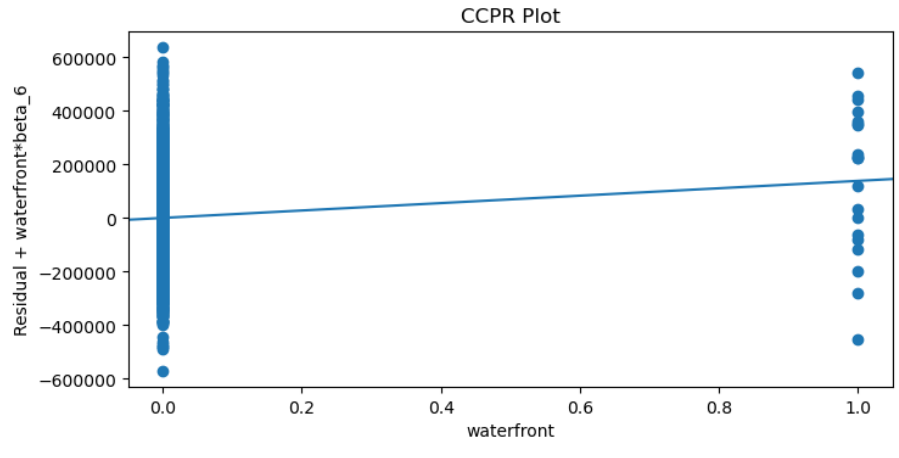


Regression Plots for waterfront

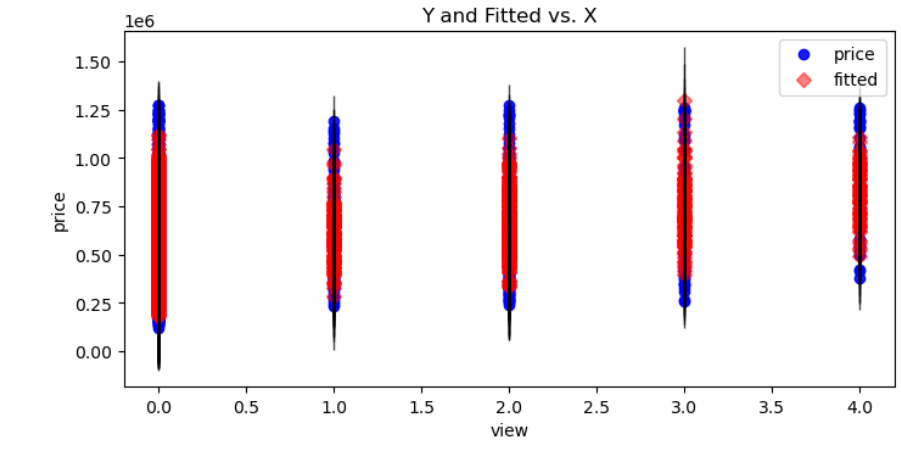


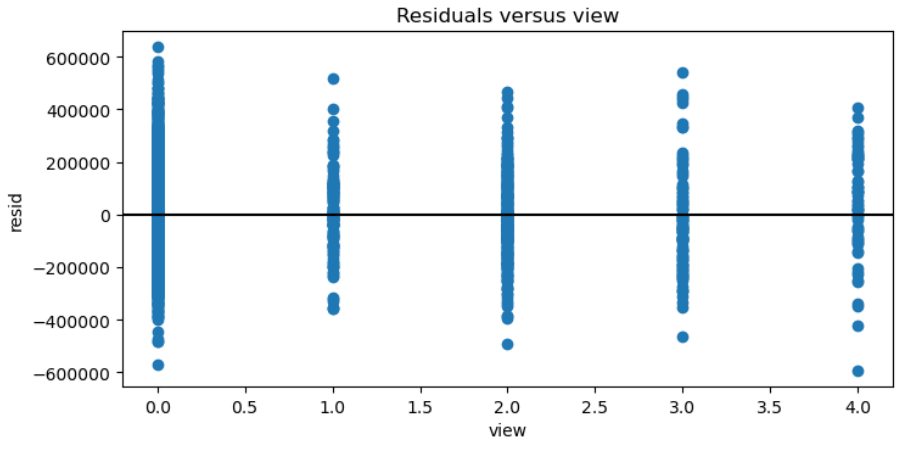


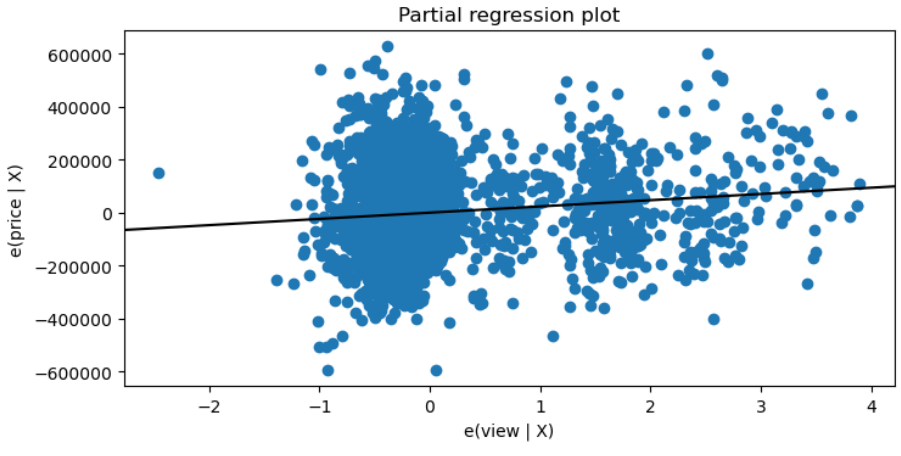


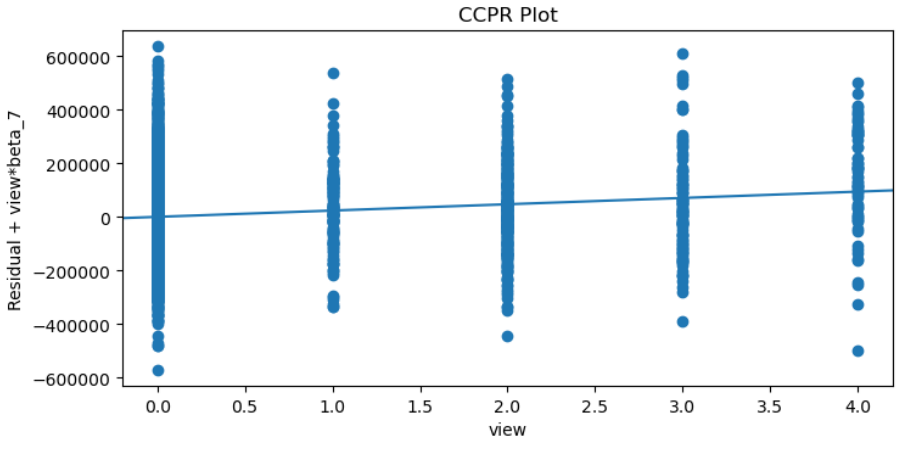


Regression Plots for view

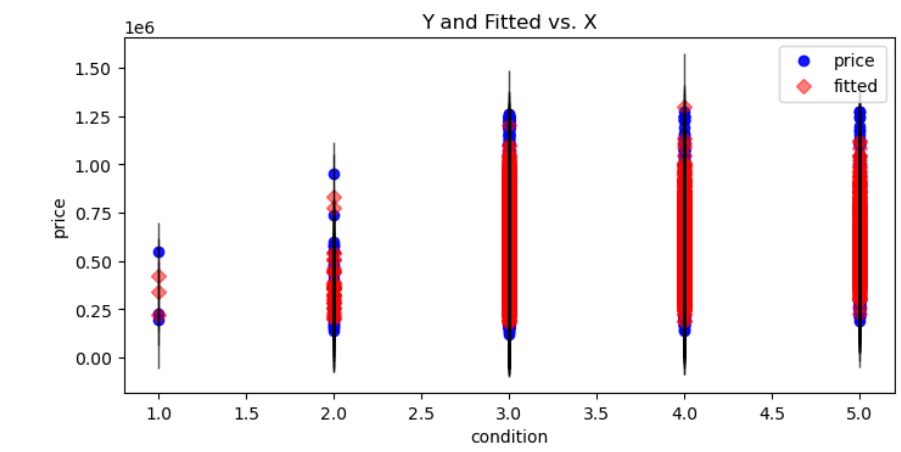


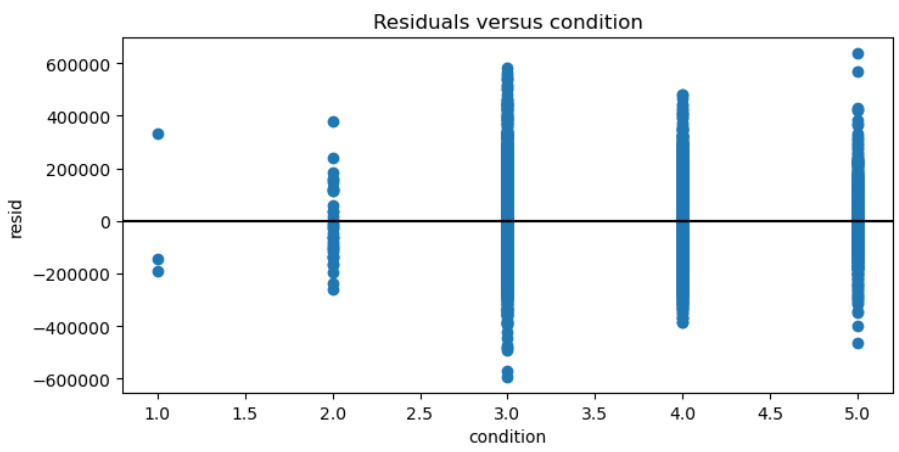


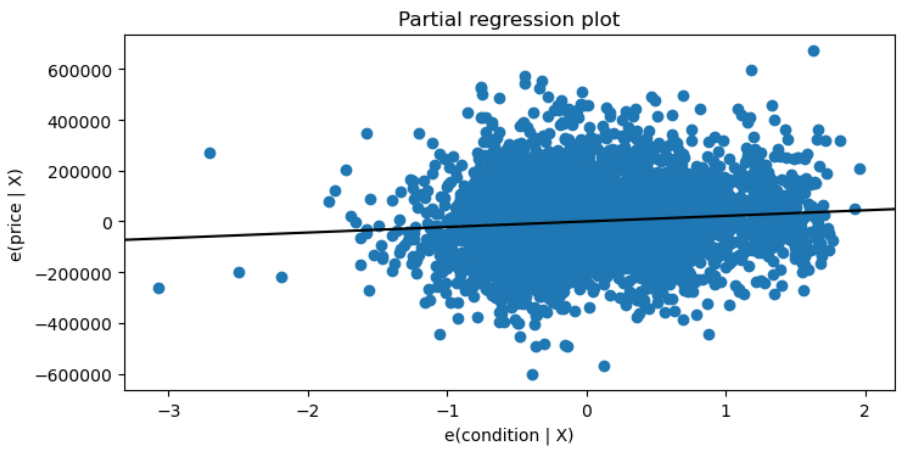


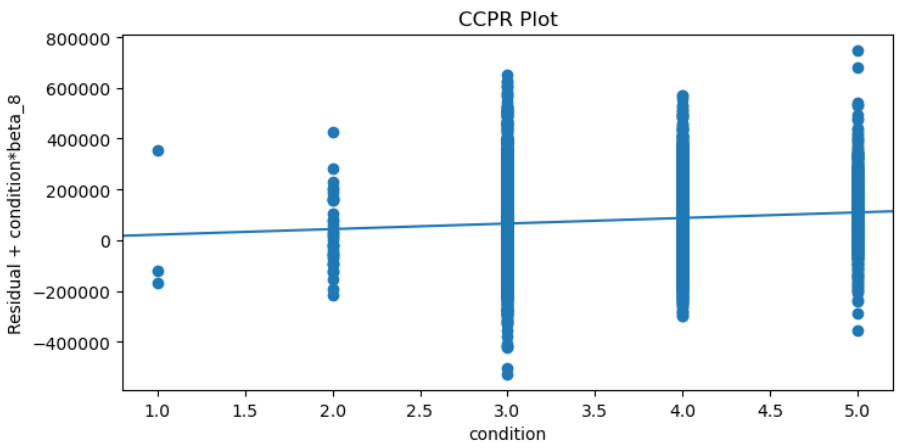


Regression Plots for condition

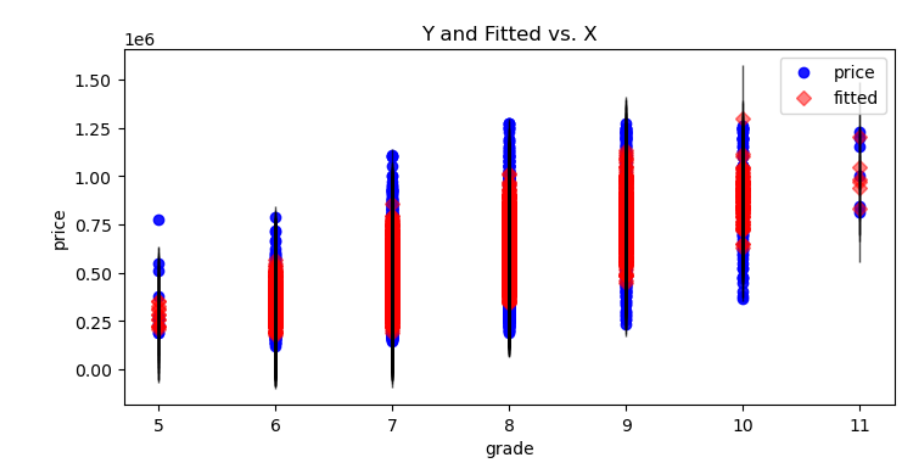


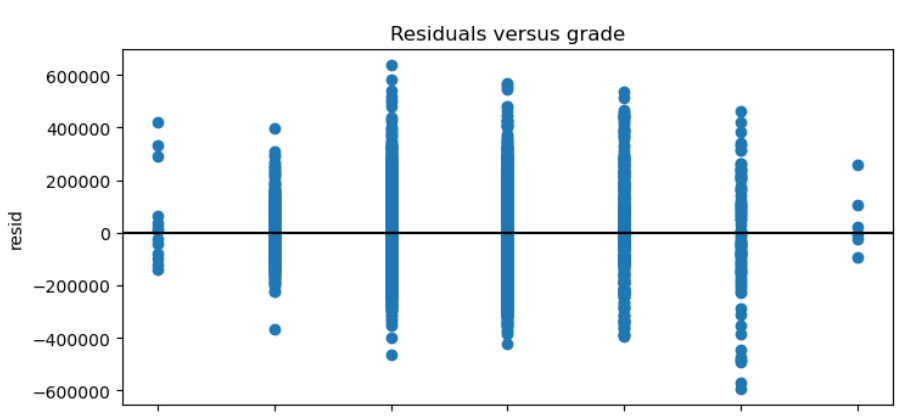


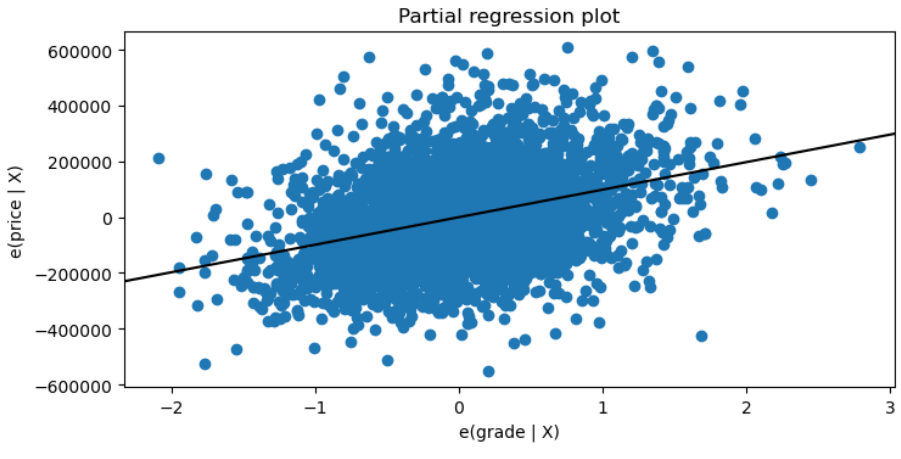


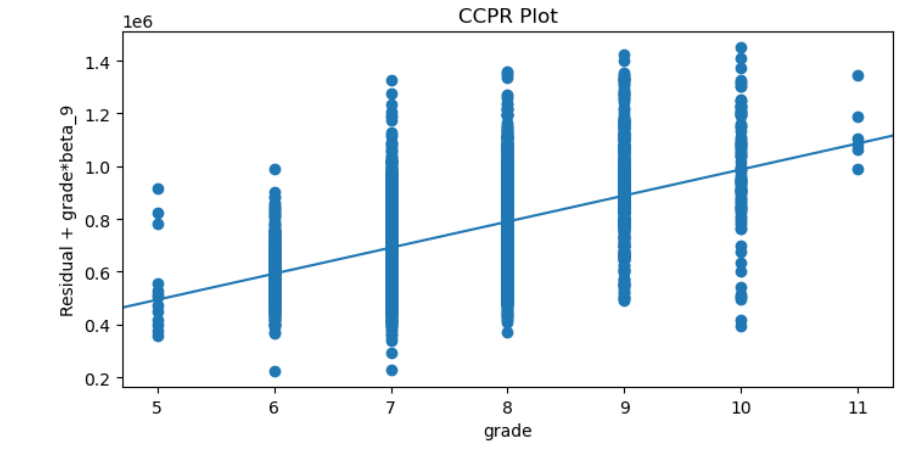


Regression Plots for grade

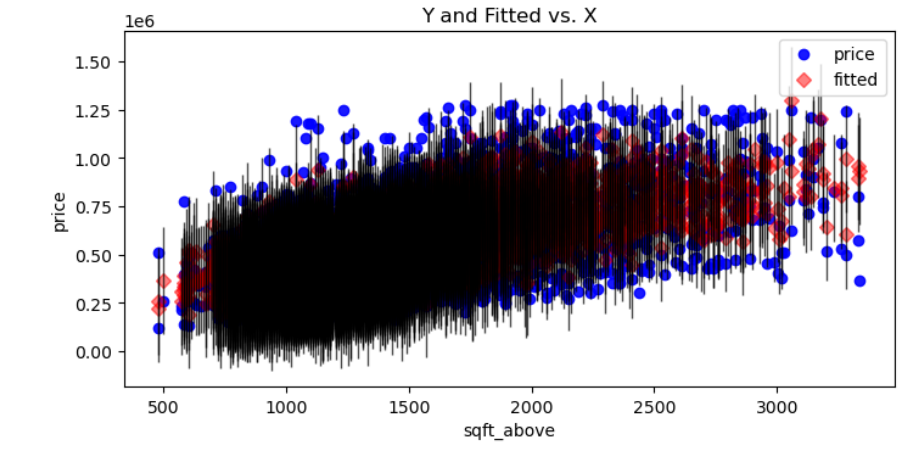


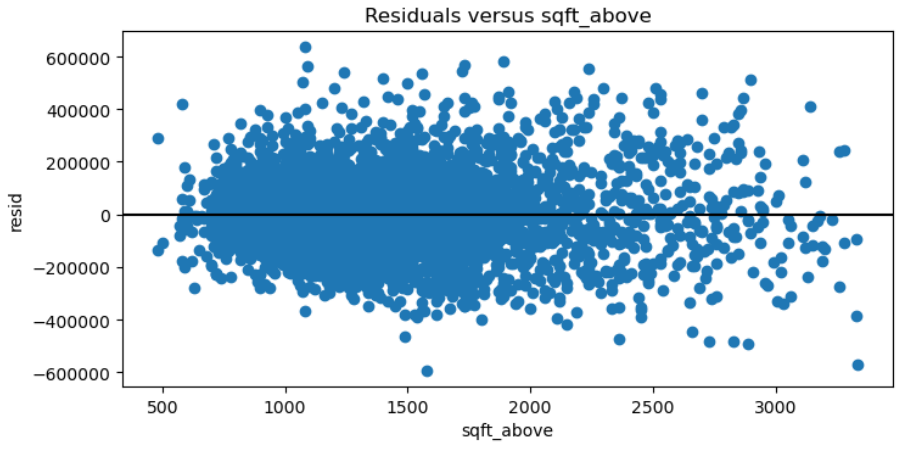


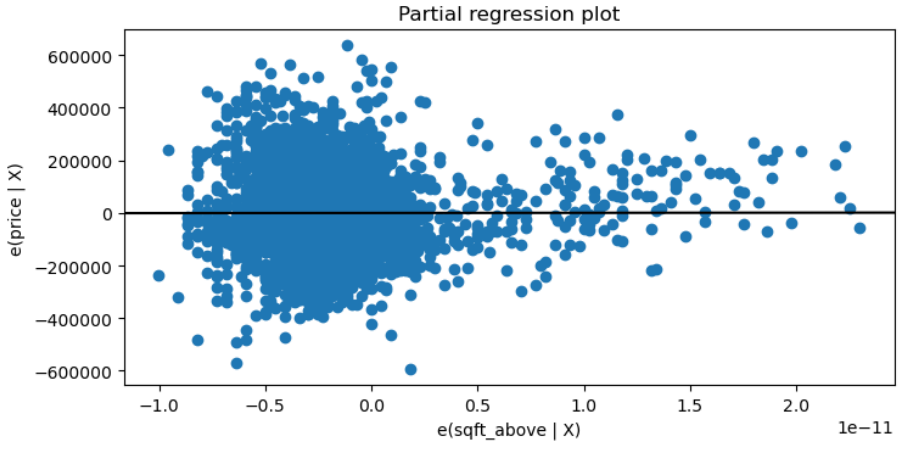


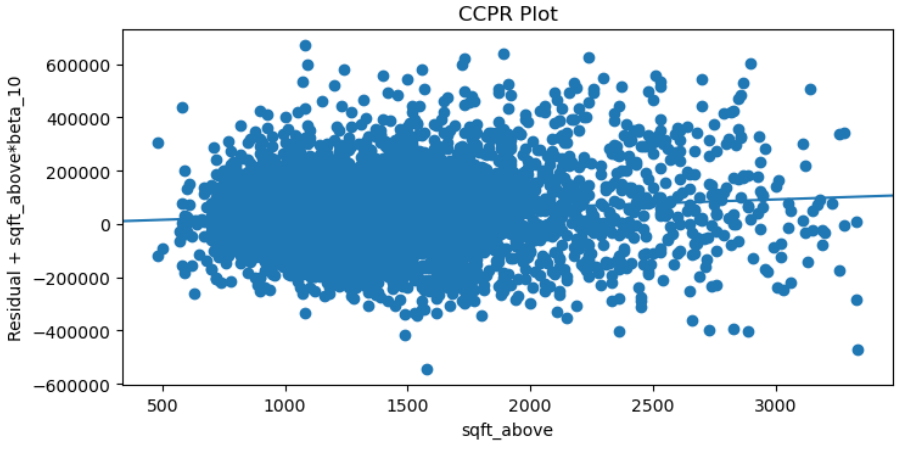


Regression Plots for sqft_above

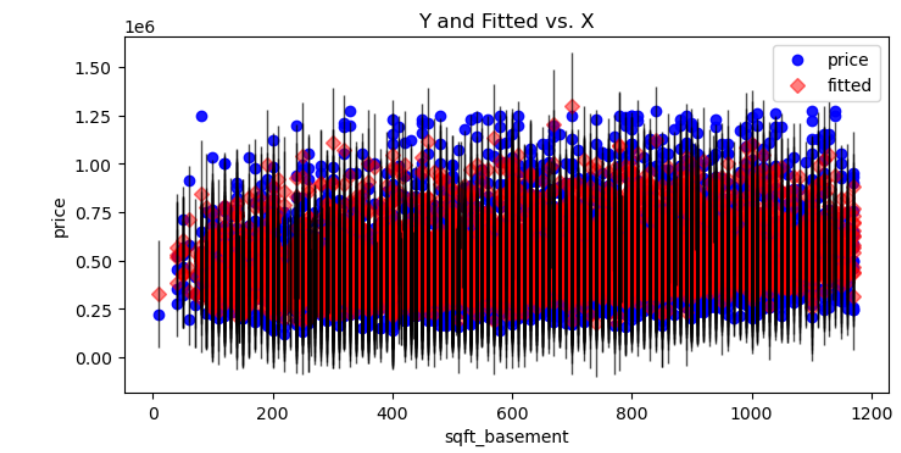


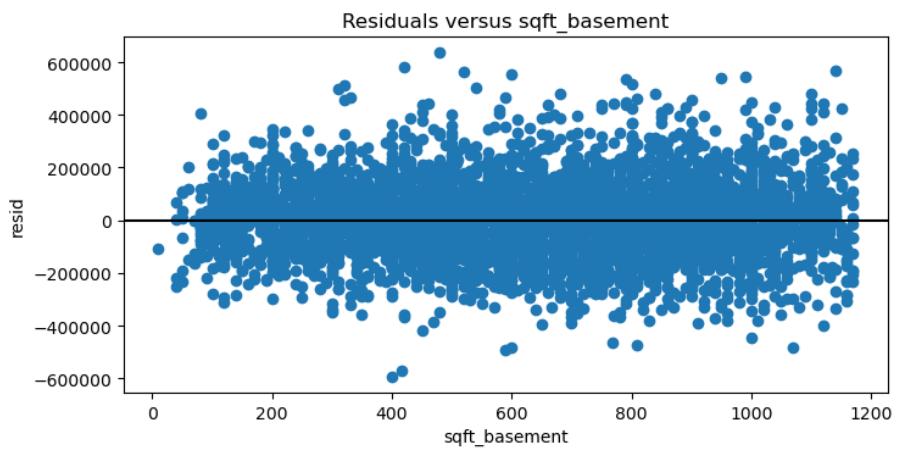


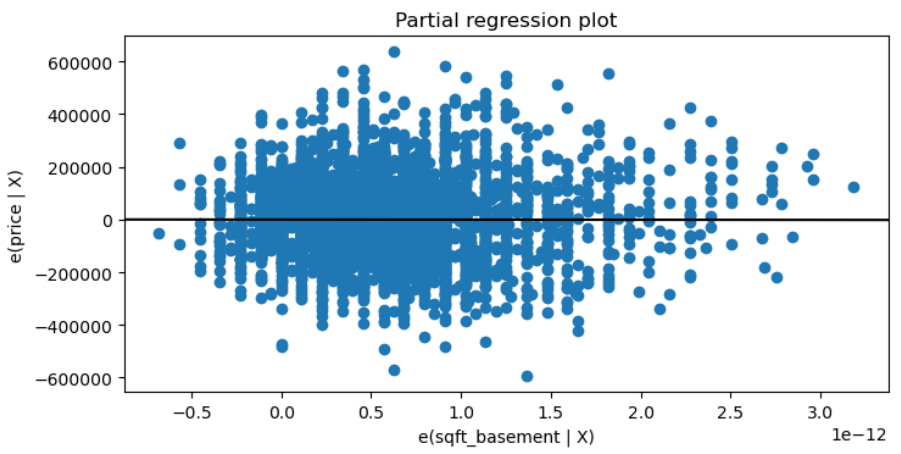


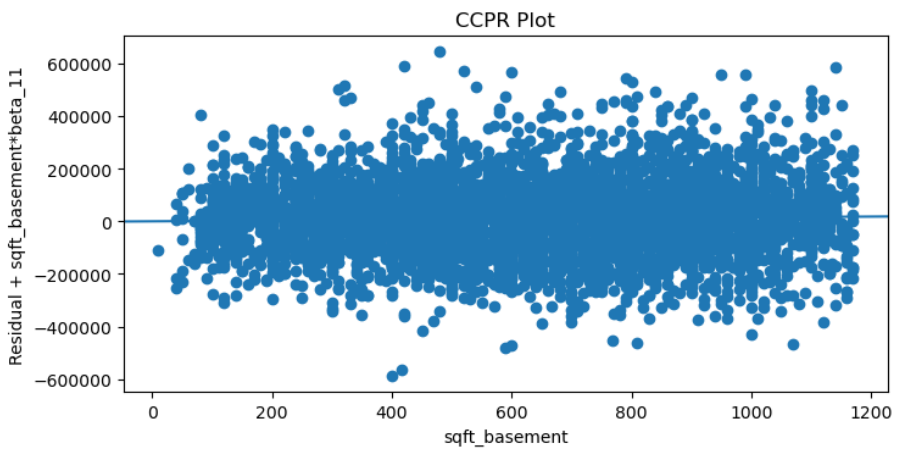


Regression Plots for sqft_basement

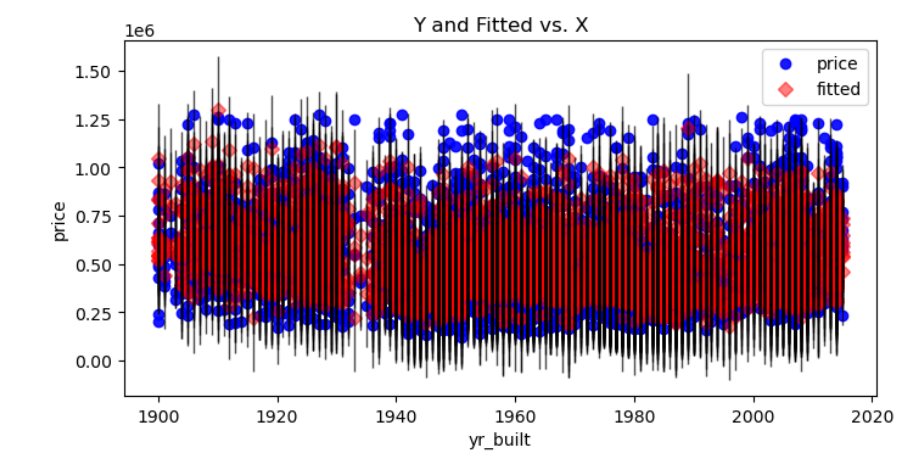


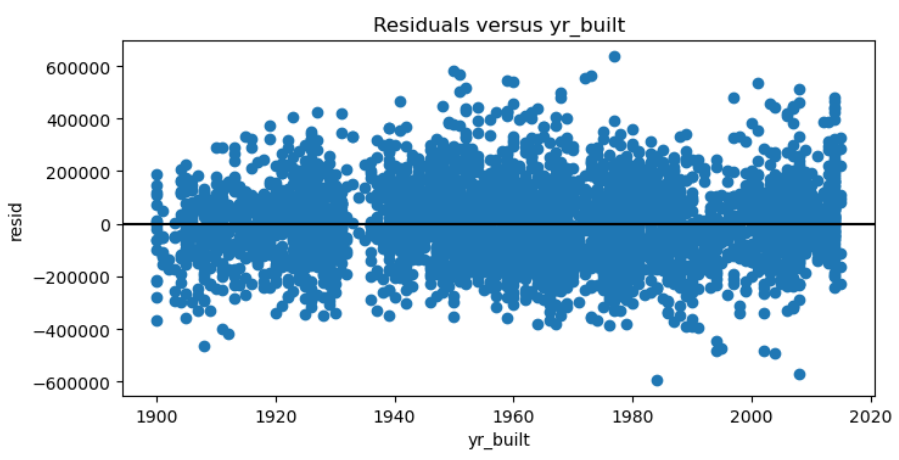


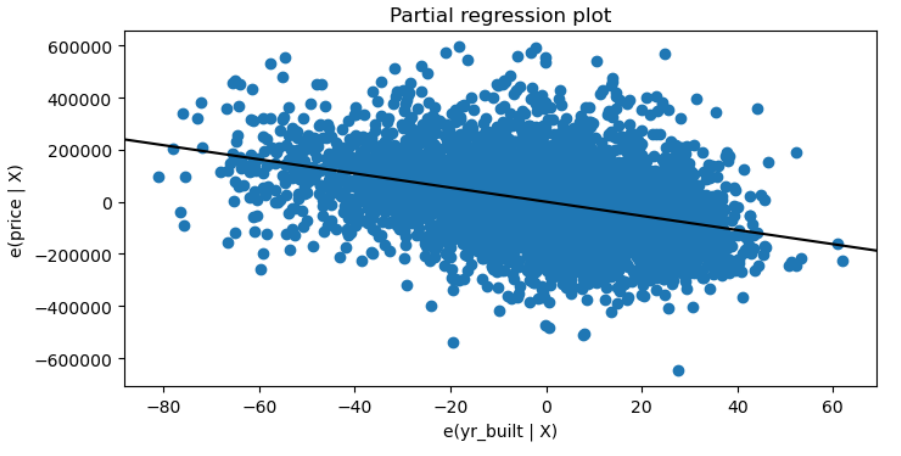


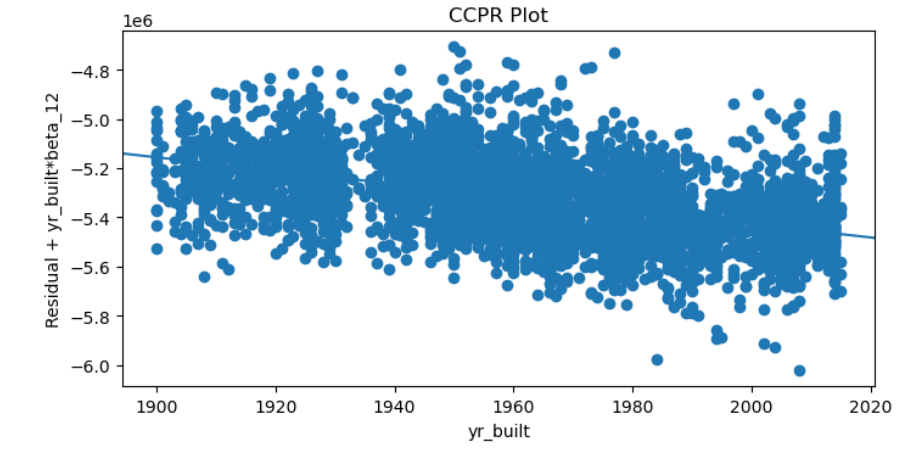


Regression Plots for yr_built

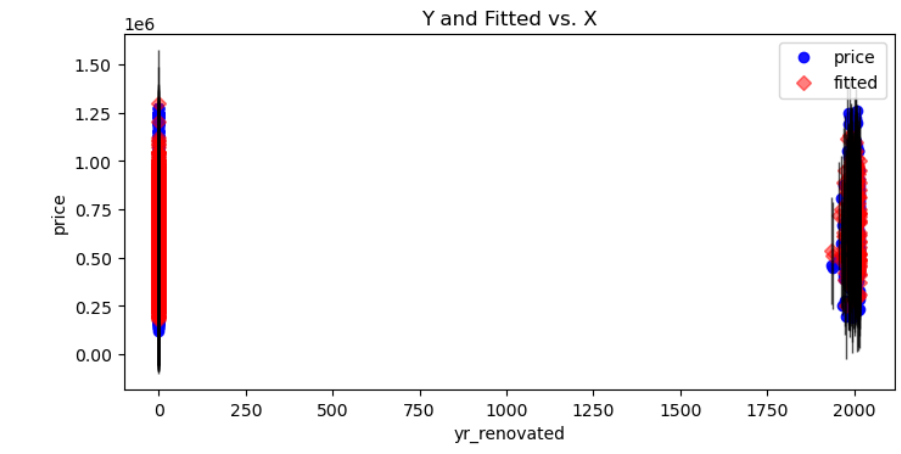


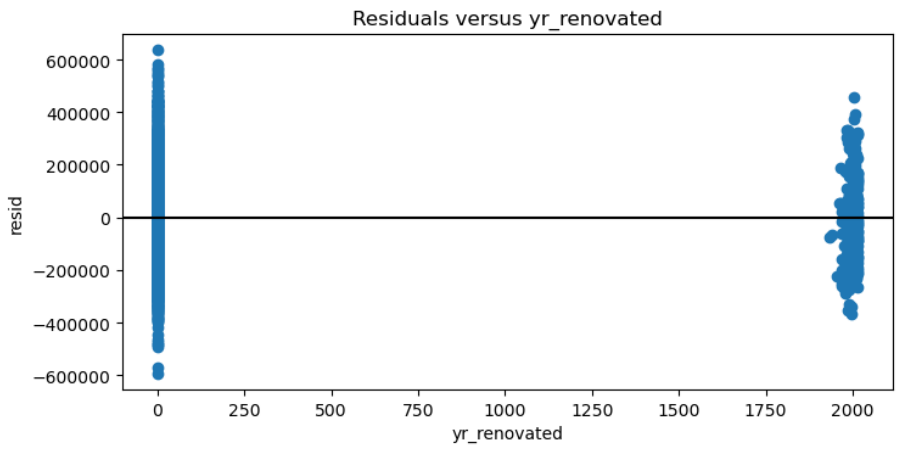


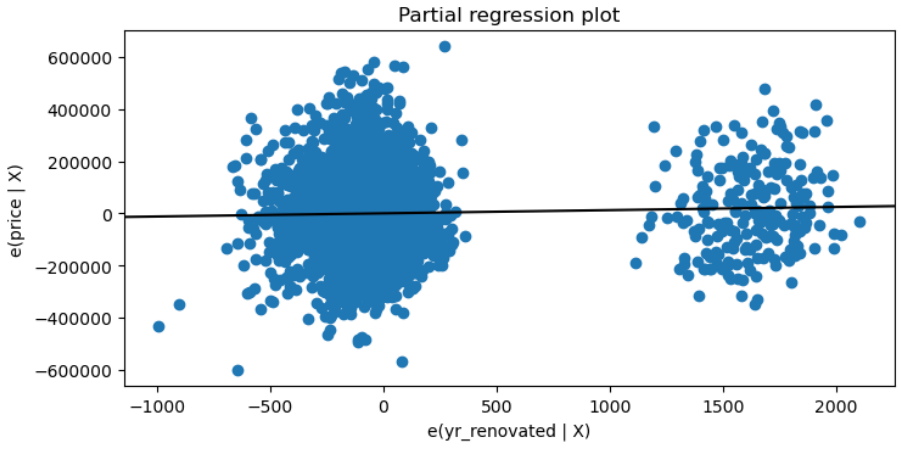


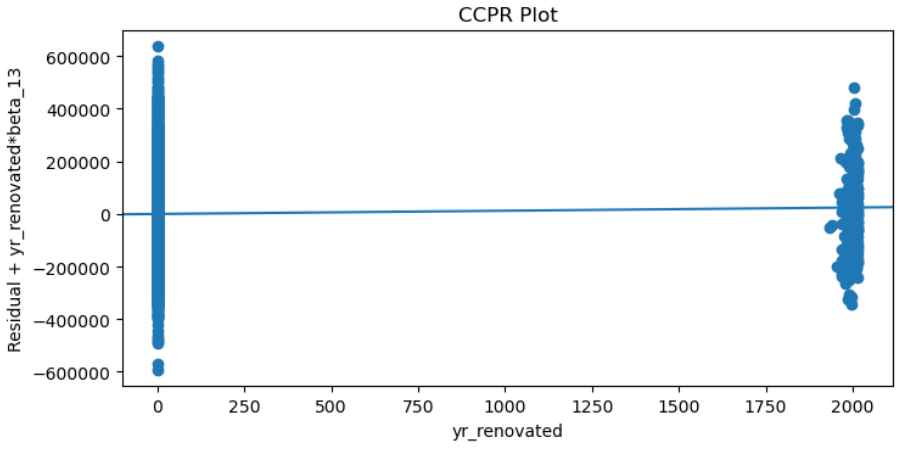


Regression Plots for yr_renovated

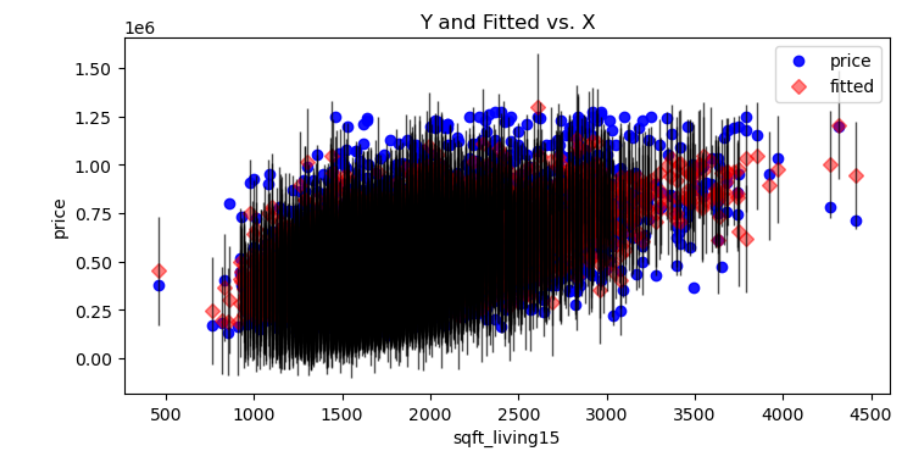


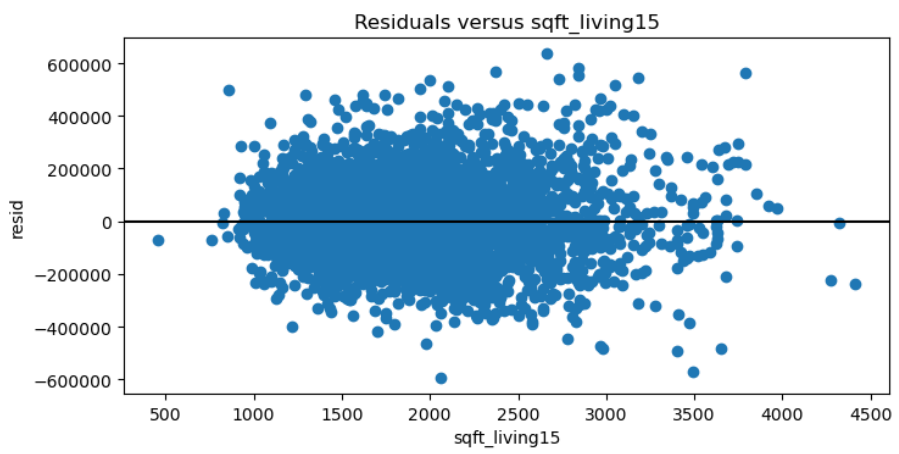


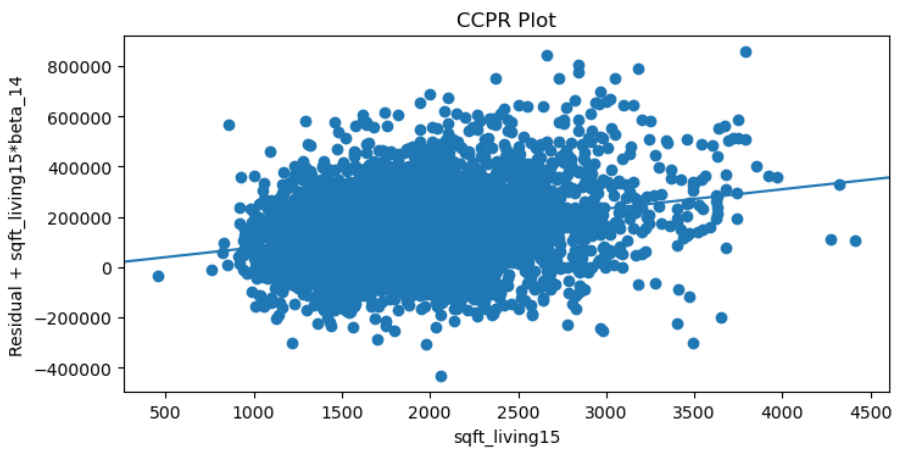




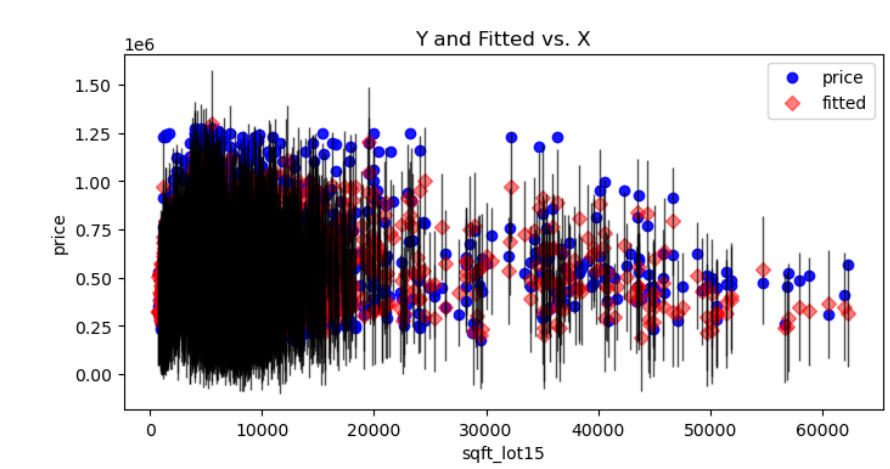
Regression Plots for sqft_living15

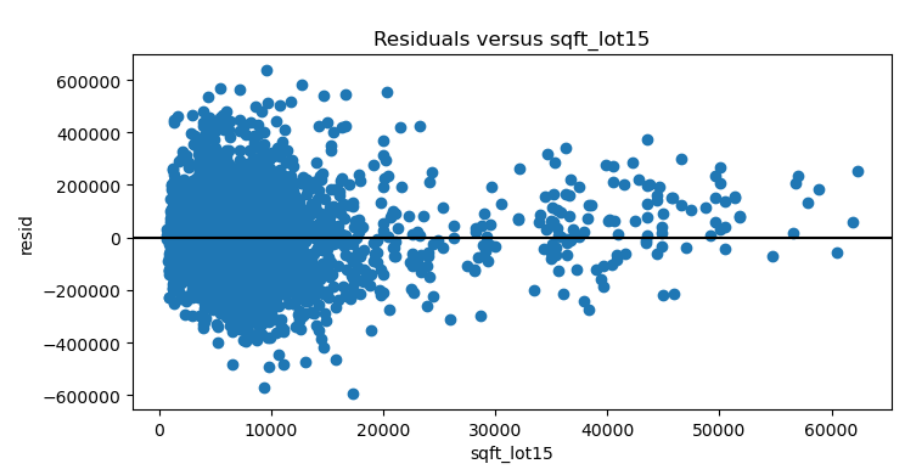


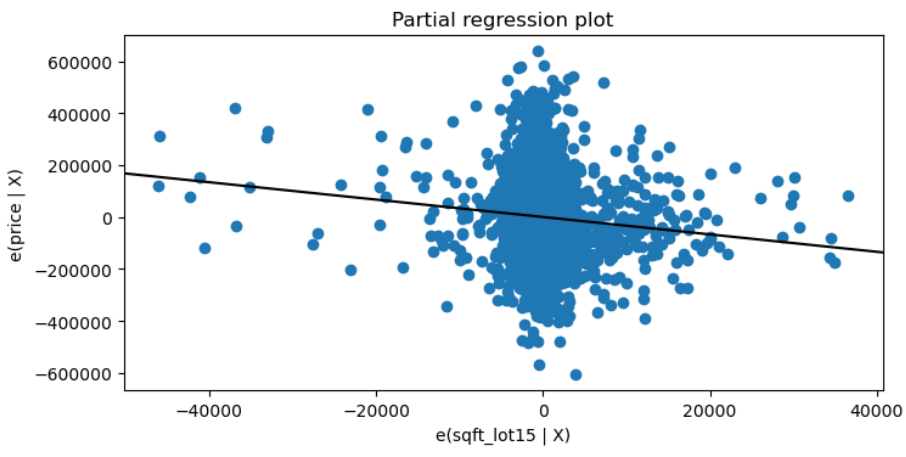


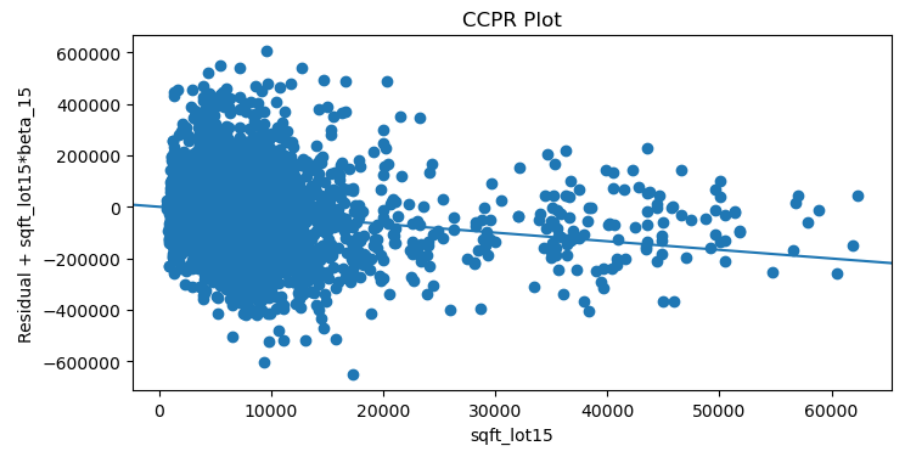


Regression Plots for sqft_lot15









From the above regression plots, we can see conical shape in case of sqft_lot and sqft_lot15. Therefore the homoscedasticity assumption seems to be violated for these variables. All other variables looks fine.

Iteration 2

In this iteration multicollinearity among the variables are determined and the dummy variables are introduced for categorical variables.

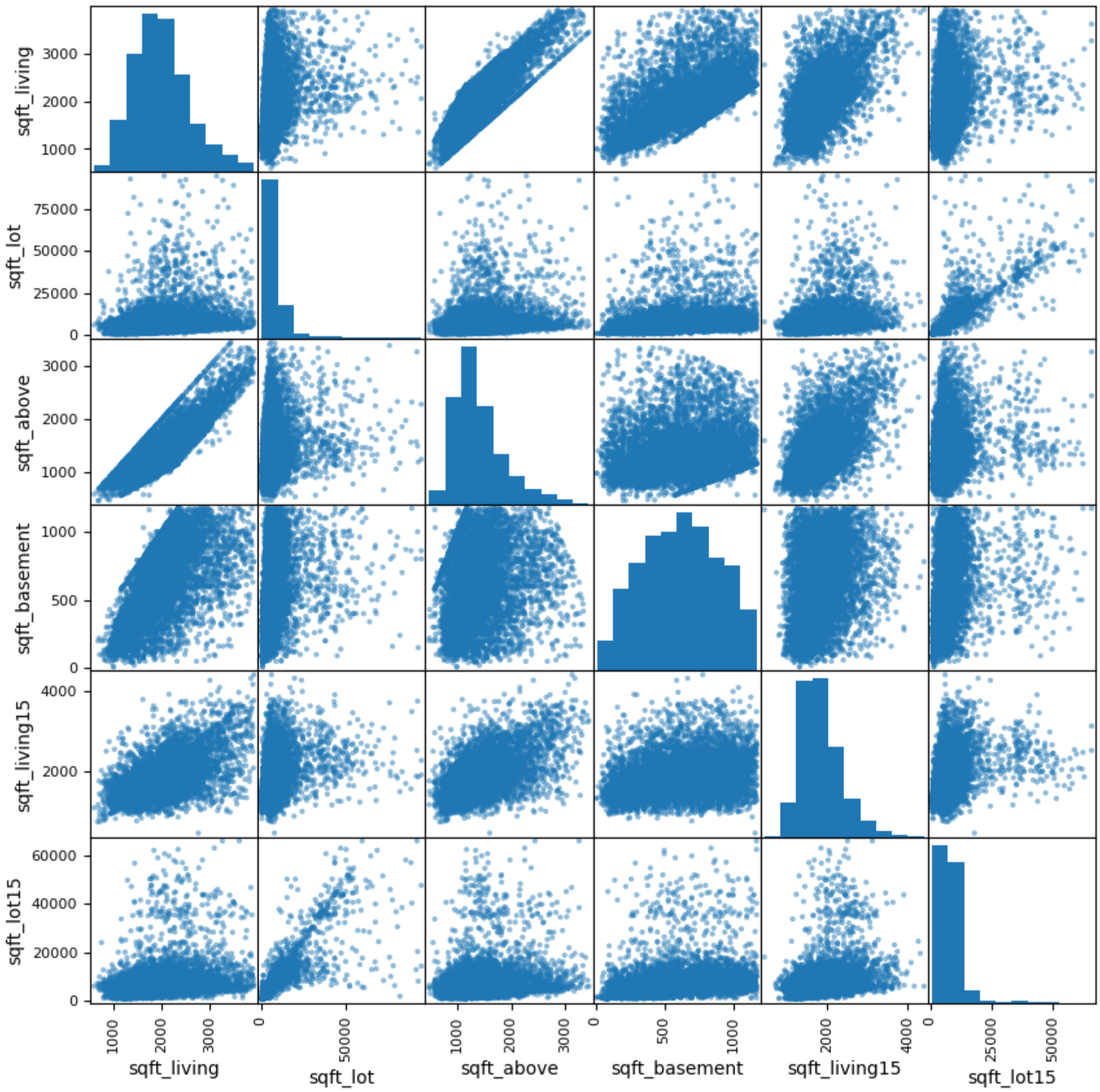
In [108…
```
#creating a new variable for iteration 2
df_it2=df2
#creating an array of continous variable columns
cont_var=['sqft_living','sqft_lot','sqft_above','sqft_basement','sqft_living15','sqft_lot15']
dpred=df_it2[cont_var]
dpred
```

Out[1086]:

	sqft_living	sqft_lot	sqft_above	sqft_basement	sqft_living15	sqft_lot15
1	2570	7242	2170	400.0	1690	7639
3	1960	5000	1050	910.0	1360	5000
6	1715	6819	1715	NaN	2238	6819
8	1780	7470	1050	730.0	1780	8113
11	1160	6000	860	300.0	1330	6000
...	...	...	...	...	...	...
21568	1260	900	940	320.0	1310	1415
21572	1210	1278	1020	190.0	1210	1118
21579	1530	981	1480	50.0	1530	1282
21590	3510	7200	2600	910.0	2050	6200
21591	1310	1294	1180	130.0	1330	1265

7127 rows × 6 columns

In [108…
```
#creating a scatter matrix of the continous matrix
pd.plotting.scatter_matrix(dpred,figsize=(10,10));
plt.show()
```



In the scatter matrix above, we can see that there exists a linear correlation between "sqft_above" and "sqft_living" and also between "sqft_lot" and "sqft_lot15". Lets further calculate the correlation between the variables.

In [108…
```
dpred.corr()
```

Out[1088]:

	sqft_living	sqft_lot	sqft_above	sqft_basement	sqft_living15	sqft_lot15
sqft_living	1.000000	0.236563	0.861465	0.604035	0.615284	0.233063
sqft_lot	0.236563	1.000000	0.191054	0.170261	0.313465	0.815394
sqft_above	0.861465	0.191054	1.000000	0.152776	0.595943	0.185961
sqft_basement	0.604035	0.170261	0.152776	1.000000	0.269897	0.176685
sqft_living15	0.615284	0.313465	0.595943	0.269897	1.000000	0.339652
sqft_lot15	0.233063	0.815394	0.185961	0.176685	0.339652	1.000000

In [108…
```
#checking if the correlation between the variables is greater than 0.75
abs(dpred.corr())>0.75
```

Out[1089]:

	sqft_living	sqft_lot	sqft_above	sqft_basement	sqft_living15	sqft_lot15
sqft_living	True	False	True	False	False	False
sqft_lot	False	True	False	False	False	True
sqft_above	True	False	True	False	False	False
sqft_basement	False	False	False	True	False	False
sqft_living15	False	False	False	False	True	False
sqft_lot15	False	True	False	False	False	True

In [109…
```
#creating pairs of variables and their corresponding correlation factor and sorting them in the descending order
df3= dpred.corr().abs().stack().reset_index().sort_values(0,ascending=False)
df3['pairs']=list(zip(df3.level_0,df3.level_1))
df3.set_index(['pairs'],inplace=True)
df3.drop(columns=['level_0','level_1'],inplace=True)
df3.columns = ['cc']
df3.drop_duplicates(inplace=True)
df3
```

Out[1090]:

pairs	cc
(sqft_living, sqft_living)	1.000000
(sqft_living, sqft_above)	0.861465
(sqft_lot15, sqft_lot)	0.815394
(sqft_living, sqft_living15)	0.615284
(sqft_living, sqft_basement)	0.604035
(sqft_above, sqft_living15)	0.595943
(sqft_lot15, sqft_living15)	0.339652
(sqft_living15, sqft_lot)	0.313465
(sqft_basement, sqft_living15)	0.269897
(sqft_lot, sqft_living)	0.236563
(sqft_living, sqft_lot15)	0.233063
(sqft_lot, sqft_above)	0.191054
(sqft_lot15, sqft_above)	0.185961
(sqft_basement, sqft_lot15)	0.176685
(sqft_basement, sqft_lot)	0.170261
(sqft_above, sqft_basement)	0.152776

In [109…
```
#selecting the pairs with correlation factor more than 0.75
df3[(df3.cc>0.75)&(df3.cc<1)]
```

Out[1091]:

pairs	cc
(sqft_living, sqft_above)	0.861465
(sqft_lot15, sqft_lot)	0.815394

So we can see that 4 variables are highly correlated among each other, namely: 'sqft_above', 'sqft_living', 'sqft_lot15', 'sqft_lot'. So we would need to remove 3 of them to remove collinear features.

We can remove the variables 'sqft_above', 'sqft_lot15' and 'sqft_lot'.

This will also help in avoiding the violation of th homoscedasticity assumption by the variables 'sqft_lot15' and 'sqft_lot' as it will be removed due to the multicollinearity.

In [109…
```
#dropping the columns with multicollinearity
df_it2 = df_it2.drop(['sqft_above','sqft_lot15','sqft_lot'],axis=1)
```

Dummy variables will be created for the categorical variables now. To avoid the dummy vraiable trap, the one of the columns of the dummy variable will be removed.

In [109…
```
#creating dummy variables for waterfront, view, condition and grade after removing the null values
df_it2.dropna(subset=['waterfront','view','condition','grade'],inplace=True)
wtr_dummy=pd.get_dummies(df_it2['waterfront'].astype(int),prefix='wtr',drop_first=True)
view_dummy=pd.get_dummies(df_it2['view'].astype(int),prefix='view',drop_first=True)
cond_dummy=pd.get_dummies(df_it2['condition'],prefix='cond',drop_first=True)
grad_dummy=pd.get_dummies(df_it2['grade'],prefix='grad',drop_first=True)
```

Dummy variables for each unique value in the columns of the above parameters were created as they did not have a large number of unique variables.

However in the case of bedrooms, bathrooms, year built and year renovated, the unique variables are many in number and hence having a dummy vraible for each unique value in the column will increase the number of dummy varaiables significantly. For example, year built has years ranging from 1900 to 2015 and therefore creating a dummy variable for each unique year in the dataset will not a good solution.

Therefore a dummy variable will be introduced for a range of values. Dummy variables will be created for number of floors between 1-2, 2-3 and greater than 3. For year built and year renovated, a dummy variable will be created for each decade of years.

In [109…

```
#Distributing the number of floors data into [1,2),[2,3) and >=3 categories.
flr=pd.cut(df_it2['floors'],[0,1.5,2.5,3.5],labels=[1,2,3])
#creating a dummy variable for each of the category mentioned above and dropping the first column
flr_dummy=pd.get_dummies(flr,prefix='flr',drop_first=True)

#Distributing the number of bedrooms data into [1,3),[3,6) and >=6 categories.
bed=pd.cut(df_it2['bedrooms'],[0,3,6,max(df_it2['bedrooms'])],labels=[1,4,7])
#creating a dummy variable for each of the category mentioned above
bed_dummy=pd.get_dummies(bed,prefix='bed')
# dropping the last column of the dummy variables for bedroom
bed_dummy= bed_dummy.drop(bed_dummy.columns[2], axis=1)

#Distributing the number of bedrooms data into [1,2),[2,4) and >=4 categories.
bath=pd.cut(df_it2['bathrooms'],[0,2,4,max(df2['bathrooms'])],labels=[1,3,5])
#creating a dummy variable for each of the category mentioned above
bath_dummy=pd.get_dummies(bath,prefix='bath')
# dropping the last column of the dummy variables for bathroom
bath_dummy= bath_dummy.drop(bath_dummy.columns[2], axis=1)

#creating an array to input into the categorizing function for each deacde starting from 1900 until 2020
a=np.arange(1900,2021,10)
#Distributing the yr_built data into different decades
y_built=pd.cut(df_it2['yr_built'],a,right=False,labels=a[0:-1],include_lowest=True)
#creating a dummy variable for each of the decade and dropping the first column
ybuilt_dummy=pd.get_dummies(y_built,prefix='y_built',drop_first=True)

#creating an array to input into the categorizing function for each deacde starting from 1900 until 2020
b=np.arange(1930,2021,10)
#including 0 in the beginning of the array to capture the data for the house where no renovation was done
b=np.insert(b,0,0)
#Distributing the yr_renovated data into different decades
y_reno=pd.cut(df_it2['yr_renovated'],b,right=False,labels=b[0:-1], include_lowest=True)
#creating a dummy variable for each of the decade and dropping the first column
yreno_dummy= pd.get_dummies(y_reno,prefix='y_reno',drop_first=True)
```

The original columns for the categoroical variables are removed and the dummy variables are inserted into the dataframe. Also the data in the dummy variable columns are boolean data. This will be converted to numeric 0s and 1s.

In [109…

```
df_it2 = df_it2.drop(['floors','waterfront','view','condition','grade','bedrooms','bathrooms','yr_renovated','yr_built'], axis
df_it2 = pd.concat([df_it2, flr_dummy, wtr_dummy, view_dummy,cond_dummy,grad_dummy,bed_dummy,bath_dummy,yreno_dummy,ybuilt_dum
```

In [109…

```
df_it2 = df_it2.replace({True: 1, False: 0})
df_it2.head()
```

Out[1096]:

	price	sqft_living	sqft_basement	sqft_living15	flr_2	flr_3	wtr_1	view_1	view_2	view_3	...	y_built_1920	y_built_1930	y_built_1940	y_bu
1	538000.0	2570	400.0	1690	1	0	0	0	0	0	...	0	0	0	
3	604000.0	1960	910.0	1360	0	0	0	0	0	0	...	0	0	0	
6	257500.0	1715	NaN	2238	1	0	0	0	0	0	...	0	0	0	
8	229500.0	1780	730.0	1780	0	0	0	0	0	0	...	0	0	0	
11	468000.0	1160	300.0	1330	0	0	0	0	0	0	...	0	0	1	

5 rows × 46 columns

The statsmodel was run after removing the multicollinearities and the adding the dummy variables

In [109…

```
outcome='price'
x_cols=df_it2.iloc[:,1:46]
predictors='+'.join(x_cols)
formula=outcome+'~'+predictors
model_itr2=smf.ols(formula=formula,data=df_it2).fit()
model_itr2.summary()
```

Out[1097]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.579
Model:	OLS	**Adj. R-squared:**	0.576
Method:	Least Squares	**F-statistic:**	180.3
Date:	Sat, 02 Dec 2023	**Prob (F-statistic):**	0.00
Time:	09:02:07	**Log-Likelihood:**	-78919.
No. Observations:	5950	**AIC:**	1.579e+05
Df Residuals:	5904	**BIC:**	1.582e+05
Df Model:	45		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.431e+05	1.71e+05	0.839	0.401	-1.91e+05	4.77e+05
sqft_living	77.9715	6.357	12.265	0.000	65.509	90.434
sqft_basement	-16.8894	9.866	-1.712	0.087	-36.230	2.452
sqft_living15	73.7357	5.237	14.079	0.000	63.469	84.003
flr_2	5.307e+04	7244.024	7.325	0.000	3.89e+04	6.73e+04
flr_3	1.322e+05	1.87e+04	7.062	0.000	9.55e+04	1.69e+05
wtr_1	8.433e+04	3.47e+04	2.430	0.015	1.63e+04	1.52e+05
view_1	3.375e+04	1.19e+04	2.847	0.004	1.05e+04	5.7e+04
view_2	3.408e+04	8248.884	4.131	0.000	1.79e+04	5.02e+04
view_3	4.743e+04	1.28e+04	3.697	0.000	2.23e+04	7.26e+04
view_4	1.54e+05	1.95e+04	7.917	0.000	1.16e+05	1.92e+05
cond_2	3.649e+04	8.45e+04	0.432	0.666	-1.29e+05	2.02e+05
cond_3	7.323e+04	8.18e+04	0.895	0.371	-8.72e+04	2.34e+05
cond_4	9.135e+04	8.18e+04	1.116	0.264	-6.91e+04	2.52e+05
cond_5	1.321e+05	8.19e+04	1.612	0.107	-2.86e+04	2.93e+05
grad_5	-7.133e+04	1.45e+05	-0.494	0.622	-3.55e+05	2.12e+05
grad_6	-1.596e+04	1.4e+05	-0.114	0.909	-2.91e+05	2.59e+05
grad_7	1.032e+05	1.4e+05	0.736	0.461	-1.72e+05	3.78e+05
grad_8	1.91e+05	1.4e+05	1.362	0.173	-8.4e+04	4.66e+05
grad_9	3.203e+05	1.4e+05	2.280	0.023	4.49e+04	5.96e+05
grad_10	3.909e+05	1.41e+05	2.772	0.006	1.14e+05	6.67e+05
grad_11	5.209e+05	1.47e+05	3.546	0.000	2.33e+05	8.09e+05
bed_1	-6741.2665	3.72e+04	-0.181	0.856	-7.97e+04	6.62e+04
bed_4	-9698.4242	3.7e+04	-0.262	0.793	-8.23e+04	6.29e+04
bath_1	-3.536e+04	3.87e+04	-0.913	0.361	-1.11e+05	4.06e+04
bath_3	-1.319e+04	3.84e+04	-0.344	0.731	-8.84e+04	6.2e+04
y_reno_1930	-7.883e+04	1.41e+05	-0.561	0.575	-3.55e+05	1.97e+05
y_reno_1940	-7.263e+04	1.4e+05	-0.518	0.604	-3.48e+05	2.02e+05
y_reno_1950	-2.265e+05	1.4e+05	-1.615	0.106	-5.01e+05	4.85e+04
y_reno_1960	-5.452e+04	6.32e+04	-0.863	0.388	-1.78e+05	6.93e+04
y_reno_1970	-1.023e+05	3.78e+04	-2.706	0.007	-1.76e+05	-2.82e+04
y_reno_1980	-2035.7843	2.57e+04	-0.079	0.937	-5.24e+04	4.83e+04
y_reno_1990	144.5890	2.23e+04	0.006	0.995	-4.36e+04	4.39e+04
y_reno_2000	7.884e+04	1.83e+04	4.316	0.000	4.3e+04	1.15e+05
y_reno_2010	3.818e+04	1.89e+04	2.021	0.043	1151.673	7.52e+04
y_built_1910	1.965e+04	1.32e+04	1.485	0.138	-6289.620	4.56e+04
y_built_1920	3763.0518	1.2e+04	0.314	0.754	-1.98e+04	2.73e+04
y_built_1930	-2.867e+04	1.44e+04	-1.984	0.047	-5.7e+04	-341.767
y_built_1940	-4.137e+04	1.19e+04	-3.463	0.001	-6.48e+04	-1.8e+04
y_built_1950	-1.064e+05	1.17e+04	-9.053	0.000	-1.29e+05	-8.33e+04
y_built_1960	-1.765e+05	1.17e+04	-15.068	0.000	-1.99e+05	-1.54e+05
y_built_1970	-2.049e+05	1.17e+04	-17.540	0.000	-2.28e+05	-1.82e+05

y_built_1980	-2.046e+05	1.23e+04	-16.584	0.000	-2.29e+05	-1.8e+05
y_built_1990	-2.527e+05	1.43e+04	-17.677	0.000	-2.81e+05	-2.25e+05
y_built_2000	-2.047e+05	1.32e+04	-15.517	0.000	-2.31e+05	-1.79e+05
y_built_2010	-1.93e+05	1.53e+04	-12.575	0.000	-2.23e+05	-1.63e+05

Omnibus:	196.331	**Durbin-Watson:**	2.005
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	294.371
Skew:	0.325	**Prob(JB):**	1.20e-64
Kurtosis:	3.875	**Cond. No.**	6.34e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.34e+05. This might indicate that there are

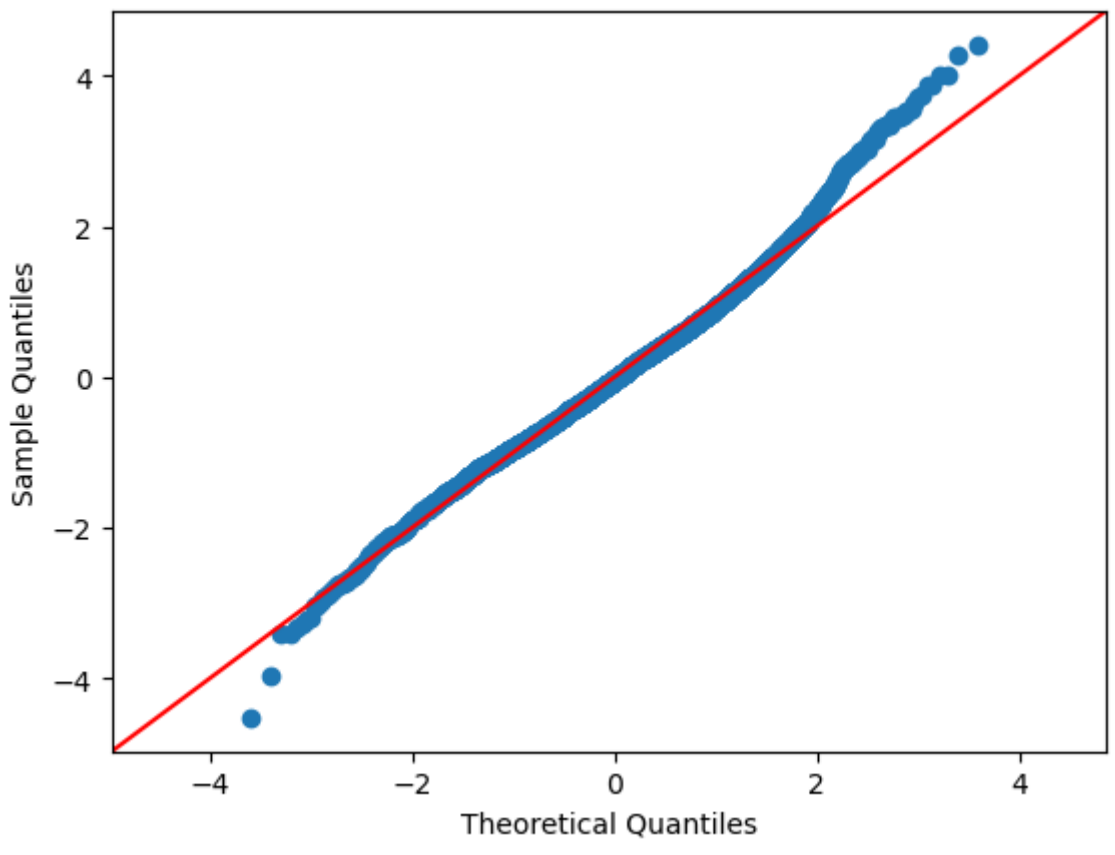
strong multicollinearity or other numerical problems.

The results show an imporvement in the Adjusted R square value after doing the above changes in Iteration 2. The model significance is high however there are few features with low significance as the P>|t| value is higher than 0.05.

The JB value is still high at around 294. Lets see how the Q-Q plot looks like with this model.

In [105…

```
residuals = model_itr2.resid
sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True)
plt.show()
```



The Q-Q plots shows deviation from the straight line and hence we can conclude that the residuals are still not normally distributed.

Iteration 3

In this iteration, log transformation and feature scaling will be applied to see any improvements to the model

In [123…

```
#copying the values in the dataframe to a new dataframe for iteration 3
df_it3=df_it2
df_it3
```

Out[1232]:

	price	sqft_living	sqft_basement	sqft_living15	flr_2	flr_3	wtr_1	view_1	view_2	view_3	...	y_built_1920	y_built_1930	y_built_1940
1	538000.0	2570	400.0	1690	1	0	0	0	0	0	...	0	0	0
3	604000.0	1960	910.0	1360	0	0	0	0	0	0	...	0	0	0
6	257500.0	1715	NaN	2238	1	0	0	0	0	0	...	0	0	0
8	229500.0	1780	730.0	1780	0	0	0	0	0	0	...	0	0	0
11	468000.0	1160	300.0	1330	0	0	0	0	0	0	...	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21568	380000.0	1260	320.0	1310	1	0	0	0	0	0	...	0	0	0
21572	414500.0	1210	190.0	1210	1	0	0	0	0	0	...	0	0	0
21579	520000.0	1530	50.0	1530	0	1	0	0	0	0	...	0	0	0
21590	1010000.0	3510	910.0	2050	1	0	0	0	0	0	...	0	0	0
21591	475000.0	1310	130.0	1330	1	0	0	0	0	0	...	0	0	0

6317 rows × 46 columns

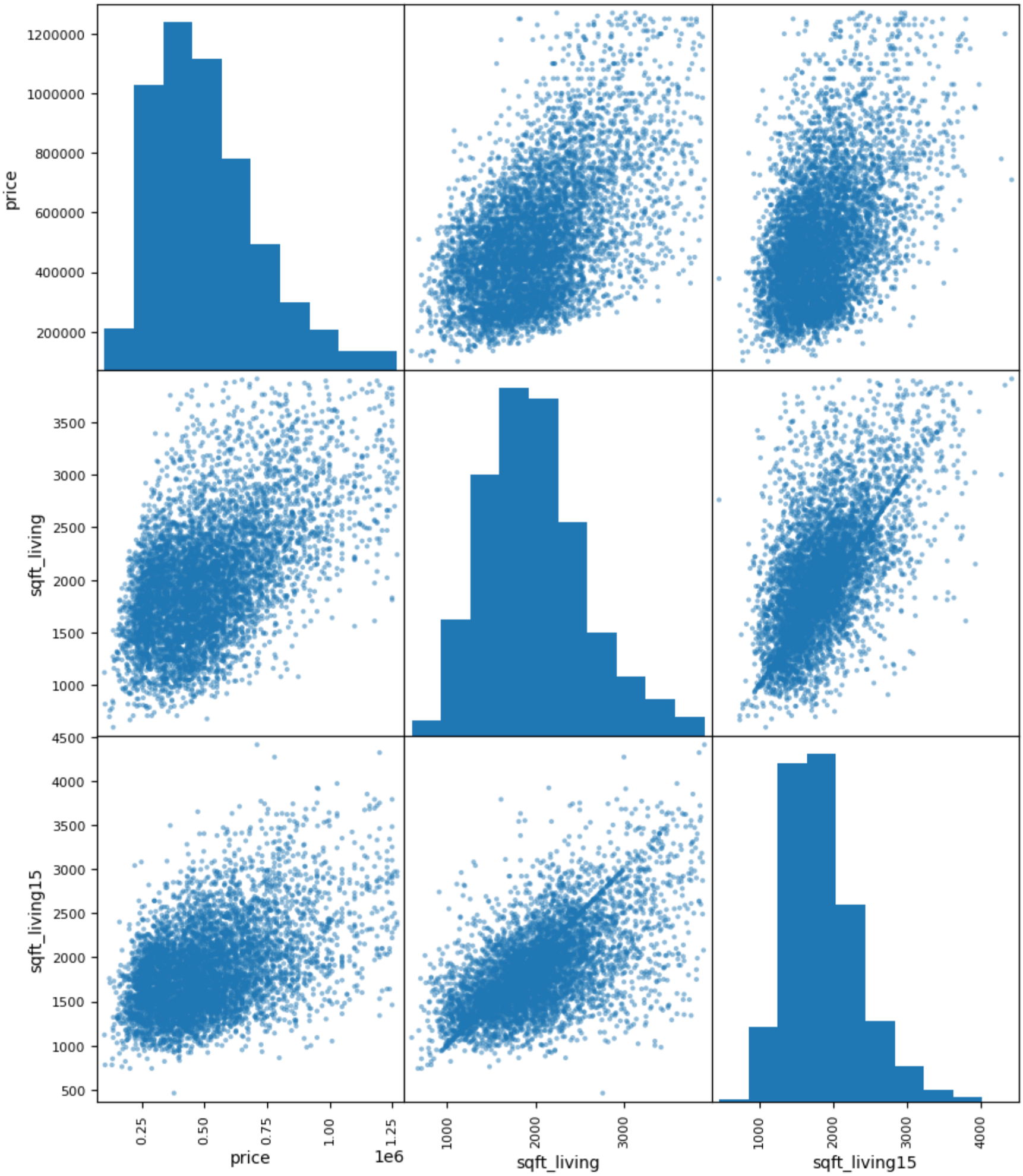
Plotting the scatter matrix for the continuous variables to see the trend in data

In [110…

```
Logtr_columns=['price','sqft_living','sqft_living15']
pd.plotting.scatter_matrix(df_it3[Logtr_columns],figsize=(10,12))
```

Out[1101]:

```
array([[<Axes: xlabel='price', ylabel='price'>,
        <Axes: xlabel='sqft_living', ylabel='price'>,
        <Axes: xlabel='sqft_living15', ylabel='price'>],
       [<Axes: xlabel='price', ylabel='sqft_living'>,
        <Axes: xlabel='sqft_living', ylabel='sqft_living'>,
        <Axes: xlabel='sqft_living15', ylabel='sqft_living'>],
       [<Axes: xlabel='price', ylabel='sqft_living15'>,
        <Axes: xlabel='sqft_living', ylabel='sqft_living15'>,
        <Axes: xlabel='sqft_living15', ylabel='sqft_living15'>]],
      dtype=object)
```



Log transformation will be done to the continuous variables: price, sqft_living, sqft_living15 as they have leftward skew.

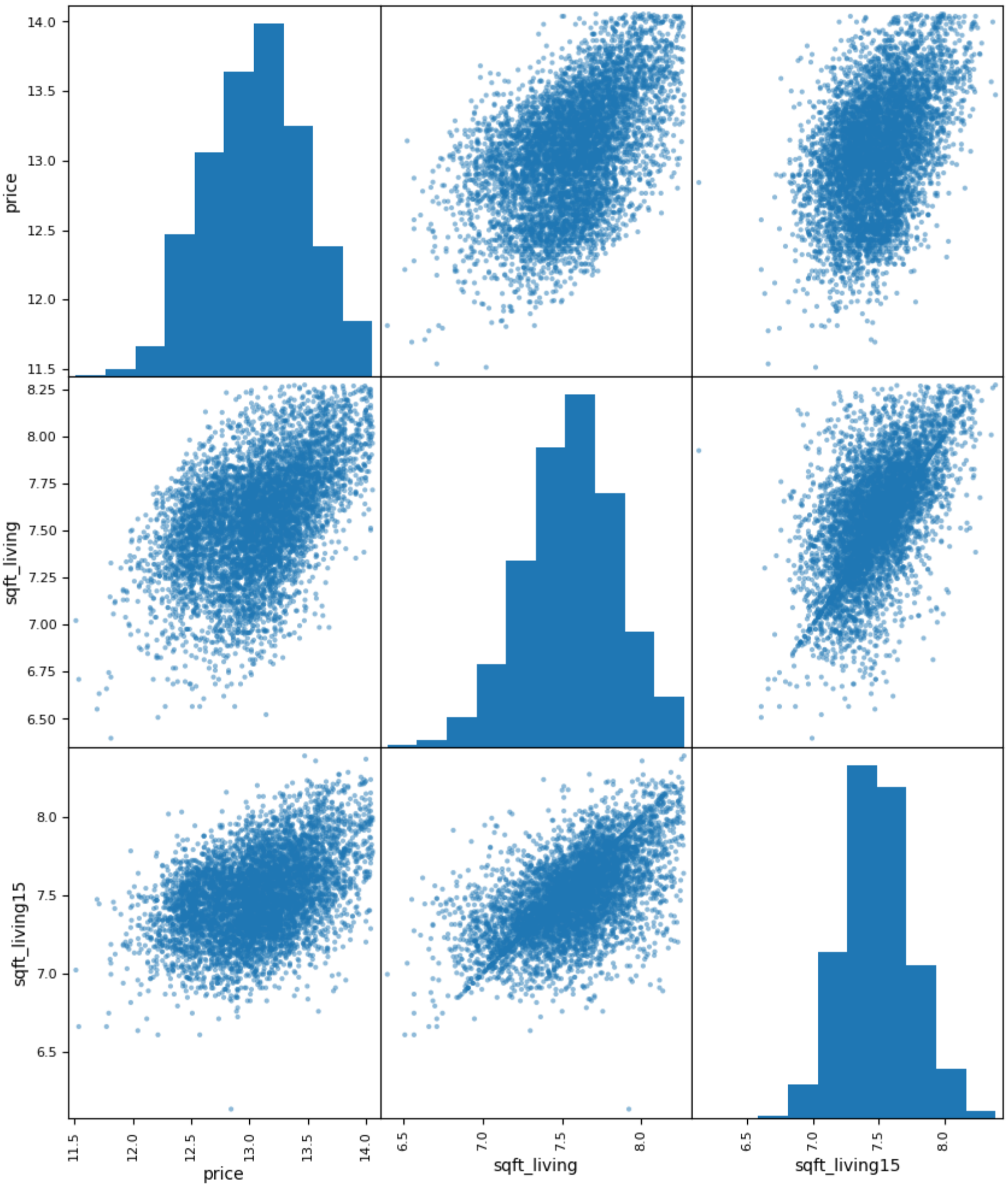
sqft_basement, another continuous variable is avoided from the log transformation as it looks to normally distributed currently and after doing log transformation, the skewness was found to be increased.

In [110…

```
#applying log transformation
log_transform=np.log(df_it3[Logtr_columns])
pd.plotting.scatter_matrix(log_transform,figsize=(10,12))
```

Out[1103]:

```
array([[<Axes: xlabel='price', ylabel='price'>,
        <Axes: xlabel='sqft_living', ylabel='price'>,
        <Axes: xlabel='sqft_living15', ylabel='price'>],
       [<Axes: xlabel='price', ylabel='sqft_living'>,
        <Axes: xlabel='sqft_living', ylabel='sqft_living'>,
        <Axes: xlabel='sqft_living15', ylabel='sqft_living'>],
       [<Axes: xlabel='price', ylabel='sqft_living15'>,
        <Axes: xlabel='sqft_living', ylabel='sqft_living15'>,
        <Axes: xlabel='sqft_living15', ylabel='sqft_living15'>]],
      dtype=object)
```



The scatter matrix shows a normal trend for the three continuous variables after the log transformation.

In [110…
```
log_transform.head()
```

Out[1104]:

	price	sqft_living	sqft_living15
1	13.195614	7.851661	7.432484
3	13.311329	7.580700	7.215240
6	12.458775	7.447168	7.713338
8	12.343658	7.484369	7.484369
11	13.056224	7.056175	7.192934

Feature Scaling on the continuous variables will done to bring all the variables to a uniform scale. The mean and standard deviation will be used to do this scaling. The values after doing the scaling will be between -1 and 1.

In [110…
```
log_price=log_transform['price']
log_sqliv=log_transform['sqft_living']
log_sqbas=df_it3['sqft_basement']
log_sqliv15=log_transform['sqft_living15']


scaled_price = (log_price - np.mean(log_price)) / np.sqrt(np.var(log_price))
scaled_sqliv = (log_sqliv - np.mean(log_sqliv)) / np.sqrt(np.var(log_sqliv))
scaled_sqbas = (log_sqbas - np.mean(log_sqbas)) / np.sqrt(np.var(log_sqbas))
scaled_sqliv15 = (log_sqliv15 - np.mean(log_sqliv15)) / np.sqrt(np.var(log_sqliv15))
```

The feature scaling was also done to sqft-basement as seen above, so as to bring it within the same scale as other area parameters.

The original columns corresponding to these parameters will be removed from the dataframe and the lof transformed and feature scaled values will be inserted in to the dataframe.

In [110…
```
#removing the old columns for the continuous variables
df_it3=df_it3.drop(['sqft_living','sqft_basement','sqft_living15','price'], axis=1)
df_it3.head()
```

Out[1106]:

	flr_2	flr_3	wtr_1	view_1	view_2	view_3	view_4	cond_2	cond_3	cond_4	...	y_built_1920	y_built_1930	y_built_1940	y_built_1950	y_built_
1	1	0	0	0	0	0	0	0	1	0	...	0	0	0	1	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
6	1	0	0	0	0	0	0	0	1	0	...	0	0	0	0	
8	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	
11	0	0	0	0	0	0	0	0	0	1	...	0	0	1	0	

5 rows × 42 columns

In [110…
```
#adding the log transformed and feature scaled values into the dataframe
df_it3 = pd.concat([scaled_price,scaled_sqliv,scaled_sqbas,scaled_sqliv15, df_it3],axis=1)
df_it3
```

Out[1107]:

	price	sqft_living	sqft_basement	sqft_living15	flr_2	flr_3	wtr_1	view_1	view_2	view_3	...	y_built_1920	y_built_1930	y_built_1940
1	0.337097	1.004379	-0.778401	-0.226905	1	0	0	0	0	0	...	0	0	0
3	0.614840	0.081984	1.066219	-1.068292	0	0	0	0	0	0	...	0	0	0
6	-1.431478	-0.372578	NaN	0.860844	1	0	0	0	0	0	...	0	0	0
8	-1.707784	-0.245942	0.415177	-0.025955	0	0	0	0	0	0	...	0	0	0
11	0.002529	-1.703580	-1.140091	-1.154682	0	0	0	0	0	0	...	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21568	-0.497429	-1.422084	-1.067753	-1.213365	1	0	0	0	0	0	...	0	0	0
21572	-0.288846	-1.559923	-1.537950	-1.520908	1	0	0	0	0	0	...	0	0	0
21579	0.255418	-0.761147	-2.044317	-0.612118	0	1	0	0	0	0	...	0	0	0
21590	1.848868	2.065490	1.066219	0.521016	1	0	0	0	0	0	...	0	0	0
21591	0.038164	-1.289610	-1.754964	-1.154682	1	0	0	0	0	0	...	0	0	0

6317 rows × 46 columns

The statsmodel will be run after performing the above changes

In [110…
```
outcome='price'
x_cols=df_it3.iloc[:,1:46]
predictors='+'.join(x_cols)
formula=outcome+'~'+predictors
model_itr3=smf.ols(formula=formula,data=df_it3).fit()
model_itr3.summary()
```

Out[1108]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.533
Model:	OLS	**Adj. R-squared:**	0.529
Method:	Least Squares	**F-statistic:**	149.5
Date:	Sat, 02 Dec 2023	**Prob (F-statistic):**	0.00
Time:	09:15:58	**Log-Likelihood:**	-6145.1
No. Observations:	5950	**AIC:**	1.238e+04
Df Residuals:	5904	**BIC:**	1.269e+04
Df Model:	45		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.3922	0.829	-0.473	0.636	-2.017	1.233
sqft_living	0.1758	0.018	9.672	0.000	0.140	0.211
sqft_basement	-0.0039	0.014	-0.286	0.775	-0.031	0.023
sqft_living15	0.1804	0.012	14.496	0.000	0.156	0.205
flr_2	0.2233	0.035	6.377	0.000	0.155	0.292
flr_3	0.5679	0.091	6.215	0.000	0.389	0.747
wtr_1	0.3447	0.169	2.037	0.042	0.013	0.676
view_1	0.1792	0.058	3.100	0.002	0.066	0.293
view_2	0.1535	0.040	3.816	0.000	0.075	0.232
view_3	0.1749	0.063	2.797	0.005	0.052	0.297
view_4	0.5135	0.095	5.420	0.000	0.328	0.699
cond_2	0.1873	0.412	0.454	0.650	-0.621	0.995
cond_3	0.5335	0.399	1.337	0.181	-0.249	1.316
cond_4	0.6267	0.399	1.570	0.116	-0.156	1.409
cond_5	0.7948	0.400	1.989	0.047	0.011	1.578
grad_5	-0.7507	0.705	-1.064	0.287	-2.134	0.632
grad_6	-0.2883	0.685	-0.421	0.674	-1.631	1.054
grad_7	0.3692	0.685	0.539	0.590	-0.973	1.711
grad_8	0.8192	0.685	1.196	0.232	-0.524	2.162
grad_9	1.2775	0.686	1.861	0.063	-0.068	2.623
grad_10	1.5372	0.689	2.231	0.026	0.187	2.888
grad_11	1.9227	0.717	2.680	0.007	0.516	3.329
bed_1	-0.1107	0.181	-0.611	0.541	-0.466	0.244
bed_4	-0.1232	0.180	-0.683	0.494	-0.477	0.230
bath_1	-0.1299	0.188	-0.690	0.490	-0.499	0.239
bath_3	-0.0314	0.186	-0.169	0.866	-0.397	0.334
y_reno_1930	-0.1577	0.686	-0.230	0.818	-1.503	1.187
y_reno_1940	-0.1988	0.684	-0.291	0.771	-1.539	1.142
y_reno_1950	-0.7756	0.684	-1.134	0.257	-2.116	0.565
y_reno_1960	-0.2345	0.308	-0.761	0.447	-0.839	0.370
y_reno_1970	-0.4299	0.184	-2.333	0.020	-0.791	-0.069
y_reno_1980	-0.0450	0.125	-0.360	0.719	-0.290	0.200
y_reno_1990	-0.0704	0.109	-0.647	0.518	-0.284	0.143
y_reno_2000	0.2344	0.089	2.631	0.009	0.060	0.409
y_reno_2010	0.1829	0.092	1.985	0.047	0.002	0.363
y_built_1910	0.0912	0.065	1.412	0.158	-0.035	0.218
y_built_1920	0.0305	0.059	0.522	0.602	-0.084	0.145
y_built_1930	-0.1702	0.070	-2.415	0.016	-0.308	-0.032
y_built_1940	-0.1837	0.058	-3.153	0.002	-0.298	-0.070
y_built_1950	-0.5392	0.057	-9.413	0.000	-0.651	-0.427
y_built_1960	-0.9008	0.057	-15.778	0.000	-1.013	-0.789
y_built_1970	-1.0006	0.057	-17.568	0.000	-1.112	-0.889

y_built_1980	-1.0063	0.060	-16.728	0.000	-1.124	-0.888
y_built_1990	-1.1687	0.070	-16.760	0.000	-1.305	-1.032
y_built_2000	-0.8817	0.064	-13.675	0.000	-1.008	-0.755
y_built_2010	-0.8016	0.075	-10.711	0.000	-0.948	-0.655

Omnibus:	65.209	**Durbin-Watson:**	2.028
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	67.107
Skew:	-0.259	**Prob(JB):**	2.68e-15
Kurtosis:	3.052	**Cond. No.**	383.

Notes:

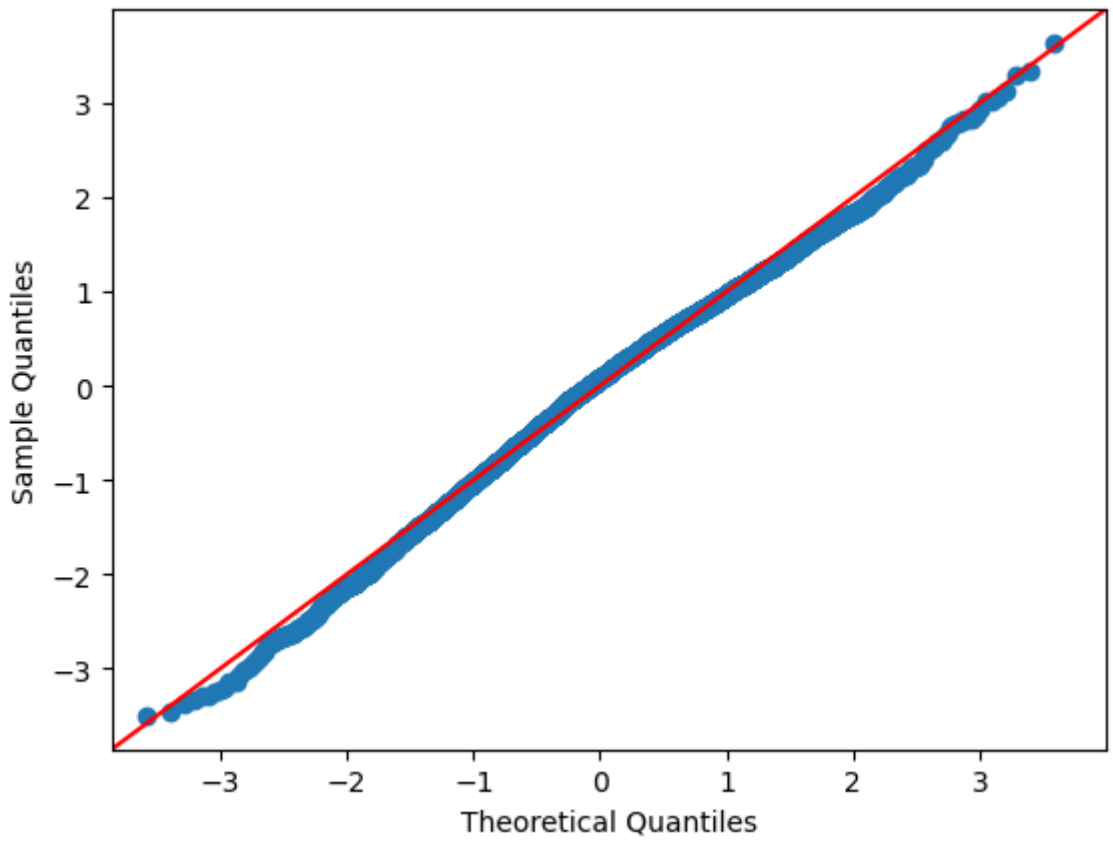
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

The result from the statsmodel run, show a decrease in the Adjusted R square value to 0.529. The model remains significant as the probability (F-statistics) is less than 0.05. There are number of features with less significance as the P>|t| value was greater than 0.05. In the next iteration, this will be attempted to be removed. Also it should be noted that the the dependable variable in the model is the log of price after doing the log transformation and feature scaling.

Eventhough there was a drop in the Adjusted R square value, there is significant improvement in the JB value from 294 to 67, meaning that the model residuals are more closer to normality. Lets verify this by plotting the Q-Q plot.

In [110…

```
residuals = model_itr3.resid
sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True)
plt.show()
```



From the Q-Q plot, it can be seen that the devaition from the straight line is minimal and hence the model residuals are closer to normality.

Iteration 4

In this iteration the stepwise forward and backward regression will be performed and the least significant features will be removed and the statemodel will be run again.

In [111…

```
#wrtitng the function definition of stepwise forward regression
def forward_regression(X, y,
                       threshold_in,
                       verbose=False):
    initial_list = []
    included = list(initial_list)
    while True:
        changed=False
        excluded = list(set(X.columns)-set(included))
        new_pval = pd.Series(index=excluded)
        for new_column in excluded:
            model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included+[new_column]]))).fit()
            new_pval[new_column] = model.pvalues[new_column]
        best_pval = new_pval.min()
        if best_pval < threshold_in:
            best_feature = new_pval.idxmin()
            included.append(best_feature)
            changed=True
            if verbose:
                print('Add  {:30} with p-value {:.6}'.format(best_feature, best_pval))

        if not changed:
            break

    return included

#writing the function definition of the stepwise backward regression
def backward_regression(X, y,
                        threshold_out,
                        verbose=False):
    included=list(X.columns)
    while True:
        changed=False
        model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).fit()
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max() # null if pvalues is empty
        if worst_pval > threshold_out:
            changed=True
            worst_feature = pvalues.idxmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feature, worst_pval))
        if not changed:
            break
    return included
```

In [111…

```
#running the stepwise forward regrssion
df_it3=df_it3.dropna()
result = forward_regression(df_it3.iloc[:,1:46], df_it3['price'], 0.01,verbose=True)
print('resulting features:')
print(result)
```

```
Add  sqft_living                    with p-value 0.0
Add  grad_7                         with p-value 7.25557e-75
Add  grad_6                         with p-value 7.45733e-67
Add  y_built_1920                   with p-value 1.10891e-71
Add  y_built_1910                   with p-value 2.38179e-50
Add  grad_8                         with p-value 4.11809e-54
Add  y_built_1940                   with p-value 1.12541e-53
Add  sqft_living15                  with p-value 2.20662e-32
Add  y_built_1930                   with p-value 4.40886e-31
Add  grad_5                         with p-value 5.01661e-31
Add  y_built_1950                   with p-value 1.83815e-27
Add  flr_2                          with p-value 5.80296e-22
Add  cond_5                         with p-value 6.51326e-17
Add  flr_3                          with p-value 6.43385e-14
Add  view_4                         with p-value 3.30238e-11
Add  y_built_1990                   with p-value 8.84719e-09
Add  y_built_1970                   with p-value 3.22337e-07
Add  y_built_1980                   with p-value 2.12855e-08
Add  y_built_1960                   with p-value 1.35579e-14
Add  y_built_2000                   with p-value 3.77847e-19
Add  y_built_2010                   with p-value 5.56411e-31
Add  bath_1                         with p-value 2.7883e-05
Add  cond_4                         with p-value 4.66418e-05
Add  grad_9                         with p-value 4.50156e-05
Add  cond_3                         with p-value 0.000189039
Add  view_2                         with p-value 0.000556785
Add  view_1                         with p-value 0.00230703
Add  view_3                         with p-value 0.00446081
resulting features:
['sqft_living', 'grad_7', 'grad_6', 'y_built_1920', 'y_built_1910', 'grad_8', 'y_built_1940', 'sqft_living15', 'y_built_1930
', 'grad_5', 'y_built_1950', 'flr_2', 'cond_5', 'flr_3', 'view_4', 'y_built_1990', 'y_built_1970', 'y_built_1980', 'y_built_1
960', 'y_built_2000', 'y_built_2010', 'bath_1', 'cond_4', 'grad_9', 'cond_3', 'view_2', 'view_1', 'view_3']
```

Above the results show the most significant features in the model. Now the least significant features will be determined by stepwise backward regression.

In [111…
```
result2 = backward_regression(df_it3.iloc[:,1:46], df_it3['price'], 0.05,verbose=True)
print('resulting features:')
print(result2)
```

```
Drop bath_3                         with p-value 0.866114
Drop y_reno_1930                    with p-value 0.818228
Drop sqft_basement                  with p-value 0.773393
Drop y_reno_1940                    with p-value 0.771992
Drop y_reno_1980                    with p-value 0.722807
Drop grad_6                         with p-value 0.676347
Drop cond_2                         with p-value 0.646516
Drop y_built_1920                   with p-value 0.591555
Drop bed_1                          with p-value 0.523026
Drop y_reno_1990                    with p-value 0.525562
Drop bed_4                          with p-value 0.512159
Drop y_reno_1960                    with p-value 0.451464
Drop y_reno_1950                    with p-value 0.263149
Drop y_built_1910                   with p-value 0.146231
resulting features:
['sqft_living', 'sqft_living15', 'flr_2', 'flr_3', 'wtr_1', 'view_1', 'view_2', 'view_3', 'view_4', 'cond_3', 'cond_4', 'cond
_5', 'grad_5', 'grad_7', 'grad_8', 'grad_9', 'grad_10', 'grad_11', 'bath_1', 'y_reno_1970', 'y_reno_2000', 'y_reno_2010', 'y_
built_1930', 'y_built_1940', 'y_built_1950', 'y_built_1960', 'y_built_1970', 'y_built_1980', 'y_built_1990', 'y_built_2000',
'y_built_2010']
```

From the above result, the least significant features are obtained. This will be removed from the dataframe to run the statsmodel again.

In [111…
```
#removing the least significant features from the dataframe
df_it4=df_it3.drop(['bath_3','y_reno_1930','sqft_basement','y_reno_1940','y_reno_1980','grad_6','cond_2','y_built_1920','bed_1
```

In [111…
```
df_it4.head()
```

Out[1116]:

	price	sqft_living	sqft_living15	flr_2	flr_3	wtr_1	view_1	view_2	view_3	view_4	...	y_reno_2010	y_built_1930	y_built_1940	y_built_195
1	0.337097	1.004379	-0.226905	1	0	0	0	0	0	0	...	0	0	0	
3	0.614840	0.081984	-1.068292	0	0	0	0	0	0	0	...	0	0	0	
8	-1.707784	-0.245942	-0.025955	0	0	0	0	0	0	0	...	0	0	0	
11	0.002529	-1.703580	-1.154682	0	0	0	0	0	0	0	...	0	0	1	
15	0.791011	1.473810	0.687423	1	0	0	0	0	1	0	...	0	0	0	

5 rows × 32 columns

In [111…
```
#Running the statsmodel after removign the least significant features
outcome='price'
x_cols=df_it4.iloc[:,1:32]
predictors='+'.join(x_cols)
formula=outcome+'~'+predictors
model_itr3=smf.ols(formula=formula,data=df_it4).fit()
model_itr3.summary()
```

Out[1117]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.532
Model:	OLS	**Adj. R-squared:**	0.530
Method:	Least Squares	**F-statistic:**	217.2
Date:	Sat, 02 Dec 2023	**Prob (F-statistic):**	0.00
Time:	09:28:40	**Log-Likelihood:**	-6148.3
No. Observations:	5950	**AIC:**	1.236e+04
Df Residuals:	5918	**BIC:**	1.257e+04
Df Model:	31		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.6147	0.108	-5.684	0.000	-0.827	-0.403
sqft_living	0.1697	0.013	13.061	0.000	0.144	0.195
sqft_living15	0.1804	0.012	14.700	0.000	0.156	0.204
flr_2	0.2261	0.033	6.780	0.000	0.161	0.291
flr_3	0.5771	0.090	6.447	0.000	0.402	0.753
wtr_1	0.3390	0.168	2.013	0.044	0.009	0.669
view_1	0.1791	0.058	3.102	0.002	0.066	0.292
view_2	0.1525	0.040	3.804	0.000	0.074	0.231
view_3	0.1730	0.062	2.782	0.005	0.051	0.295
view_4	0.5154	0.094	5.455	0.000	0.330	0.701
cond_3	0.3594	0.101	3.567	0.000	0.162	0.557
cond_4	0.4533	0.101	4.480	0.000	0.255	0.652
cond_5	0.6196	0.104	5.958	0.000	0.416	0.824
grad_5	-0.4832	0.174	-2.770	0.006	-0.825	-0.141
grad_7	0.6551	0.039	16.780	0.000	0.579	0.732
grad_8	1.1078	0.045	24.778	0.000	1.020	1.195
grad_9	1.5666	0.058	26.943	0.000	1.453	1.681
grad_10	1.8306	0.081	22.498	0.000	1.671	1.990
grad_11	2.2129	0.215	10.286	0.000	1.791	2.635
bath_1	-0.0969	0.023	-4.255	0.000	-0.142	-0.052
y_reno_1970	-0.4240	0.184	-2.304	0.021	-0.785	-0.063
y_reno_2000	0.2353	0.089	2.655	0.008	0.062	0.409
y_reno_2010	0.1803	0.092	1.965	0.050	0.000	0.360
y_built_1930	-0.2148	0.053	-4.016	0.000	-0.320	-0.110
y_built_1940	-0.2238	0.036	-6.168	0.000	-0.295	-0.153
y_built_1950	-0.5800	0.034	-16.831	0.000	-0.648	-0.512
y_built_1960	-0.9415	0.034	-27.663	0.000	-1.008	-0.875
y_built_1970	-1.0399	0.034	-30.718	0.000	-1.106	-0.974
y_built_1980	-1.0431	0.039	-26.771	0.000	-1.120	-0.967
y_built_1990	-1.2063	0.053	-22.699	0.000	-1.310	-1.102
y_built_2000	-0.9202	0.047	-19.680	0.000	-1.012	-0.828
y_built_2010	-0.8407	0.060	-13.973	0.000	-0.959	-0.723

Omnibus:	66.968	**Durbin-Watson:**	2.027
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	68.972
Skew:	-0.262	**Prob(JB):**	1.05e-15
Kurtosis:	3.056	**Cond. No.**	37.4

Notes:

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

The above results show a small increase in the adjusted R square value. Also Kurtosis value has come slightly closer to 3. The JB value has slightly increased to 68. However all the features in the model are now significant.

Now after obtaining the results of the statsmodel, we will perform the model validation. Also it should be noted that the the dependable variable in the model is the log of price after doing the log transformation and feature scaling.

Model Validation

This involves splitting the dataset randomly to two groups: trainig and testing datasets and fitting the model based on the training dataset.

In [121…
```
from sklearn.model_selection import train_test_split

data=df1

#dropping the irrelevant parameters
data=data.drop(['id','date','zipcode','lat','long','sqft_basement','sqft_lot','sqft_above','bedrooms','sqft_lot15','yr_built',

data=data.dropna()
X=data.drop('price',axis=1)
y=data[['price']]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
X_train
```

Out[1214]:

	bathrooms	sqft_living	floors	waterfront	view	condition	grade	sqft_living15
3151	2.25	1870	2.0	0.0	0.0	3	7	1620
14969	2.50	2290	2.0	0.0	0.0	3	9	2290
3994	1.00	1280	1.5	0.0	0.0	4	5	1410
15660	1.00	1400	1.0	0.0	0.0	3	7	1280
6785	3.00	1550	2.0	0.0	0.0	3	8	1940
...	...	...	...	...	...	...	...	...
14623	2.50	1710	2.0	0.0	0.0	3	7	1500
15509	1.75	1090	1.0	0.0	0.0	5	8	1160
7064	1.75	1670	1.0	0.0	0.0	5	8	2200
1128	1.50	2550	1.0	0.0	0.0	4	7	1380
20493	3.50	2530	2.0	0.0	0.0	3	8	1470

12471 rows × 8 columns

In [121…
```
print(len(X_train), len(X_test), len(y_train), len(y_test))
```

```
12471 4157 12471 4157
```

It cna be seen from above that the training and testing datasets are divided with the ratio 0.75:0.25.

The continuous variables will lof transformed and the One hot encorder will be applied to categorical variables.

In [121…
```
from sklearn.preprocessing import FunctionTransformer

# Instantiate a custom transformer for log transformation
log_transformer = FunctionTransformer(np.log, validate=True)

# Columns to be log transformed
log_cols = ['sqft_living', 'sqft_living15']

# New names for columns after transformation
new_log_cols = ['log_sqft_living', 'log_sqft_living15']

# Log transform the training columns and convert them into a DataFrame
X_train_log = pd.DataFrame(log_transformer.fit_transform(X_train[log_cols]), columns=new_log_cols, index=X_train.index)

X_train
```

Out[1216]:

	bathrooms	sqft_living	floors	waterfront	view	condition	grade	sqft_living15
3151	2.25	1870	2.0	0.0	0.0	3	7	1620
14969	2.50	2290	2.0	0.0	0.0	3	9	2290
3994	1.00	1280	1.5	0.0	0.0	4	5	1410
15660	1.00	1400	1.0	0.0	0.0	3	7	1280
6785	3.00	1550	2.0	0.0	0.0	3	8	1940
...	...	...	...	...	...	...	...	...
14623	2.50	1710	2.0	0.0	0.0	3	7	1500
15509	1.75	1090	1.0	0.0	0.0	5	8	1160
7064	1.75	1670	1.0	0.0	0.0	5	8	2200
1128	1.50	2550	1.0	0.0	0.0	4	7	1380
20493	3.50	2530	2.0	0.0	0.0	3	8	1470

12471 rows × 8 columns

In [121…

```
#Similiar log transformation will be for the testing dataset
X_test_log = pd.DataFrame(log_transformer.transform(X_test[log_cols]), columns=new_log_cols, index=X_test.index)

X_test
```

Out[1217]:

	bathrooms	sqft_living	floors	waterfront	view	condition	grade	sqft_living15
5927	1.50	1270	1.0	0.0	0.0	4	6	1270
15488	2.50	2390	2.0	0.0	0.0	3	7	2180
9709	2.50	2110	2.0	0.0	0.0	3	9	2540
16819	1.50	900	1.5	0.0	0.0	5	6	1000
6304	1.75	1510	1.0	0.0	0.0	4	7	1480
...	...	...	...	...	...	...	...	...
1336	2.50	3370	2.0	0.0	0.0	3	9	3370
3894	2.50	1639	2.0	0.0	0.0	3	8	1580
7916	2.00	2180	1.0	0.0	0.0	5	7	1470
20344	2.50	1530	2.5	0.0	0.0	3	7	1530
4453	2.50	2650	2.0	0.0	0.0	3	8	2230

4157 rows × 8 columns

Log transformation will be done to the y values of training and test datset as the acceptable model from the 4th iteration had the dependable variable as log of price

In [121…

```
#doing log transformation to y price columns of training and test datasets
y_train_log = np.log(y_train)
y_test_log=np.log(y_test)
```

In [121…

```
#Applying One Hote Encorder to the categorical variables of the training dataset
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder(drop='first', sparse_output=False)

cat_columns=[col for col in X_train.drop(['sqft_living','sqft_living15'],axis=1)]


ohe.fit(X_train[cat_columns])

#concatanating the log transformed continuous varaibles and the one hot encoded categorical variables of training datset
X_train = pd.concat([X_train_log,pd.DataFrame(ohe.transform(X_train[cat_columns]), index=X_train.index)], axis=1)


#concatanating the log transformed continuous varaibles and the one hot encoded categorical variables of test dataset
X_test = pd.concat([X_test_log,pd.DataFrame(ohe.transform(X_test[cat_columns]), index=X_test.index)], axis=1)
```

Out[1219]:

	log_sqft_living	log_sqft_living15	0	1	2	3	4	5	6	7	...	30	31	32	33	34	35	36	37	38	39
5927	7.146772	7.146772	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
15488	7.779049	7.687080	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
9709	7.654443	7.839919	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
16819	6.802395	6.907755	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
6304	7.319865	7.299797	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1336	8.122668	8.122668	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
3894	7.401842	7.365180	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
7916	7.687080	7.293018	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
20344	7.333023	7.333023	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
4453	7.882315	7.709757	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0

4157 rows × 42 columns

In [122…
```
#converting the datatypes to string to perform the fitting
X_train.columns = X_train.columns.astype(str)
X_test.columns = X_test.columns.astype(str)
```

In [122…
```
#performing the model fitting on the training datset and predicitng the y value for the training and test dataset
from sklearn.linear_model import LinearRegression
linreg = LinearRegression()

linreg.fit(X_train, y_train_log)

y_hat_train = linreg.predict(X_train)
y_hat_test = linreg.predict(X_test)
```

In [122…
```
#Calculatign the Mean square error for both the training and test data set
from sklearn.metrics import mean_squared_error
train_mse = mean_squared_error(y_train_log, y_hat_train)
test_mse = mean_squared_error(y_test_log, y_hat_test)
print('Train Mean Squared Error:', train_mse)
print('Test Mean Squared Error: ', test_mse)
```

```
Train Mean Squared Error: 0.103364518331006
Test Mean Squared Error:  0.10338294980578394
```

As it can be seen that the test MSE is slightly higher but not having very high deviation from the training MSE showing that the model is not overfitting however is slightly underfitting.

Inferences from the model

The following Inferences can be derived from the results: based on the coefficients obtained form the results of the statsmodel in iteration 4, the features with high-impact on the price are square feet living, number of floors, if the property has a water frontage, the condition and the grading of the house, and if the house was renovated or not.

- For targeting high price properties, the real estate agency can be recommended to look for the following aspects:
  - The number of floors of the house can influence the pricing. the log of the price will increase by 0.23 to 0.57 by having two to three floors which means that there will be an increase in price by 1.3 to 1.8 times the average price of the property of that kind.
  - Having a Waterfront property can improve the price by 1.4 times.
  - Also, better the condition and the grade, better will be the price of the property.
- The real estate agency could provide the following recommendations to potential sellers to improve the value of the property:
  - Doing a renovation can increase the value of the house by around 1.25 times the value of the house.
  - Properties with just one bathroom depreciated the value of the house by 0.91 times. therefore, increasing the number of bathrooms while doing the renovations would be recommended.