1 Phase 2 Project : King County Housing Sales Data Set -Linear Regression

Author: Jhonathan David Herrera-Shaikh

- Student pace: Flex
- Scheduled project review date/time: October, 2022
- Instructor name:
- Blog post URL:[www.jhonathanddavid.com (http://www.jhonathanddavid.com)](http://www.jhonathanddavid.com)

2 Background

In this notebook, an analysis of King County sales data in the United States for years 2014-2015 will be conducted. The purpose of the analysis is to derive conclusions for business decision making purposes, affecting current homeowners and prospective buyers of this specific area. King county, is one of three Washington state counties that include Seattle, Bellevue and Tacoma area. It covers an area of of approximately 39 towns and cities. U.S Census Bureau stats indicate the county has a population of approximately 2.2 million people as of 2020.

3 Business Understanding & Business Problem

Understanding that my business stakeholder can be a real estate agency, who would want to advice both buyers and sellers on this market, it is important to note that in this type of business, both buyers and sellers are interested in price. Therefore, it is important to understand the database first, navigage its features, identify what other categories besides price are availble to try to define and predict what exactly is the best correlation to price.

4 Database Analysis

Downloading and conducting an exploration of the data in this section

4.1 Download data bases and libraries

In [1]:
```
#importing libraries
import pandas as pd
# setting pandas display to avoid scientific notation in my dataframes
pd.options.display.float_format = '{:.2f}'.format
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn

from bs4 import BeautifulSoup
import json
import requests

import folium

import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats import diagnostic as diag
from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import NearestNeighbors
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_err
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_absolute_error, mean_squared_error,r2_scor
from sklearn.preprocessing import PolynomialFeatures, StandardScaler, OneHo


import scipy.stats as stats

import pylab

%matplotlib inline
```

I'll take a look at the database by reading it

In [2]:
```
#loading database
df= pd.read_csv('data/kc_house_data.csv')
df
```

Out[2]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	water
0	7129300520	10/13/2014	221900.00	3	1.00	1180	5650	1.00	
1	6414100192	12/9/2014	538000.00	3	2.25	2570	7242	2.00	
2	5631500400	2/25/2015	180000.00	2	1.00	770	10000	1.00	
3	2487200875	12/9/2014	604000.00	4	3.00	1960	5000	1.00	
4	1954400510	2/18/2015	510000.00	3	2.00	1680	8080	1.00	
...	...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.00	3	2.50	1530	1131	3.00	
21593	6600060120	2/23/2015	400000.00	4	2.50	2310	5813	2.00	
21594	1523300141	6/23/2014	402101.00	2	0.75	1020	1350	2.00	
21595	291310100	1/16/2015	400000.00	3	2.50	1600	2388	2.00	
21596	1523300157	10/15/2014	325000.00	2	0.75	1020	1076	2.00	

21597 rows × 21 columns

4.2 Exploring and cleaning the database

Continuing to explore and clean the database

In [3]:
```
#exploring the head and tail
df.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
0	7129300520	10/13/2014	221900.00	3	1.00	1180	5650	1.00	NaN
1	6414100192	12/9/2014	538000.00	3	2.25	2570	7242	2.00	NO
2	5631500400	2/25/2015	180000.00	2	1.00	770	10000	1.00	NO
3	2487200875	12/9/2014	604000.00	4	3.00	1960	5000	1.00	NO
4	1954400510	2/18/2015	510000.00	3	2.00	1680	8080	1.00	NO

5 rows × 21 columns

In [4]:
```
#looking at tail
df.tail()
```

Out[4]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	water
21592	263000018	5/21/2014	360000.00	3	2.50	1530	1131	3.00	
21593	6600060120	2/23/2015	400000.00	4	2.50	2310	5813	2.00	
21594	1523300141	6/23/2014	402101.00	2	0.75	1020	1350	2.00	
21595	291310100	1/16/2015	400000.00	3	2.50	1600	2388	2.00	
21596	1523300157	10/15/2014	325000.00	2	0.75	1020	1076	2.00	

5 rows × 21 columns

In [5]:
```
#understanding the shaple
df.shape
```

Out[5]:
```
(21597, 21)
```

In [6]:
```
#understanding columns and data types
df.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  object 
 9   view           21534 non-null  object 
 10  condition      21597 non-null  object 
 11  grade          21597 non-null  object 
 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(6), int64(9), object(6)
memory usage: 3.5+ MB
```

In [17]:
```
#checking the null values
df.isnull().sum()
```

Out[17]:
```
id                  0
date                0
price               0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront       2376
view               63
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_built            0
yr_renovated     3842
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
dtype: int64
```

In [7]:
```
#a statistical view
df.describe()
```

Out[7]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_abc
count	21597.00	21597.00	21597.00	21597.00	21597.00	21597.00	21597.00	21597
mean	4580474287.77	540296.57	3.37	2.12	2080.32	15099.41	1.49	1788
std	2876735715.75	367368.14	0.93	0.77	918.11	41412.64	0.54	827
min	1000102.00	78000.00	1.00	0.50	370.00	520.00	1.00	370
25%	2123049175.00	322000.00	3.00	1.75	1430.00	5040.00	1.00	1190
50%	3904930410.00	450000.00	3.00	2.25	1910.00	7618.00	1.50	1560
75%	7308900490.00	645000.00	4.00	2.50	2550.00	10685.00	2.00	2210
max	9900000190.00	7700000.00	33.00	8.00	13540.00	1651359.00	3.50	9410

Now that I know what the database looks like, how big is it, the number of rows and columns, the classification of columns, the kind of data in it overall including a brief os statiscal values and null values, my next step is to clean the data base.

4.2.1 Data initial cleaning of null values, and dropping columns

In [8]:
```
#starting with dropping rows with null values
df= df.dropna(axis=0, how='any')
df.isnull().sum()
```

Out[8]:
```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

In [9]:
```
#dropped columns: df= df.drop(columns = ["id", "lat","long","sqft_living15"
```

In [10]:
```
df
```

Out[10]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wate
1	6414100192	12/9/2014	538000.00	3	2.25	2570	7242	2.00	
3	2487200875	12/9/2014	604000.00	4	3.00	1960	5000	1.00	
4	1954400510	2/18/2015	510000.00	3	2.00	1680	8080	1.00	
5	7237550310	5/12/2014	1230000.00	4	4.50	5420	101930	1.00	
6	1321400060	6/27/2014	257500.00	3	2.25	1715	6819	2.00	
...	...	...	...	...	...	...	...	...	
21591	2997800021	2/19/2015	475000.00	3	2.50	1310	1294	2.00	
21592	263000018	5/21/2014	360000.00	3	2.50	1530	1131	3.00	
21593	6600060120	2/23/2015	400000.00	4	2.50	2310	5813	2.00	
21594	1523300141	6/23/2014	402101.00	2	0.75	1020	1350	2.00	
21596	1523300157	10/15/2014	325000.00	2	0.75	1020	1076	2.00	

15762 rows × 21 columns

In [12]:
```
#confirming
df.duplicated().sum()
```

Out[12]: 0

4.2.2 Replacing strings with integers

At this time, I'm also replacing strings with integers because by doing so, I could assign values to the strings and have a numerical hierarchy, that can account for analysis and modeling in the future much better. Doing is a better fit and data management technique.

In [ ]:
```
#replacing view1 strings to integers
df['view1']= df['view'].replace({'NONE': 0, 'FAIR': 1, 'Average':2, 'Good':
```

In [37]:
```
#replacing waterfront string to integers
df['waterfront1'] = df['waterfront'].replace({'YES':0, 'NO':1})
```

In [39]:
```
#replacing condition string to integers
df['condition1'] = df ['condition'].replace({'Poor':0, 'FAIR':1, 'Average':
```

4.2.3 Modifying to columnumerical

In [ ]:
```
#splitting and going numerical for 'grade' column will allow better stat an
df["Grade1"]= df ["grade"].str.split().apply(lambda x:x[0])
df["Grade1"]= pd.to_numeric(df["Grade1"])
```

4.2.4 Dropping unnecessary columns

In [14]:
```
df = df.drop(columns=['view','waterfront','grade'])
```

4.2.5 Statiscal findings

In [15]:
```
#square footage understanding overall
df['sqft_living'].describe()
```

Out[15]:
```
count      835.00
mean      2917.57
std       1608.54
min        370.00
25%       1785.00
50%       2570.00
75%       3756.50
max      13540.00
Name: sqft_living, dtype: float64
```

On average, houses are 2,900 square feet (SF). But there is a house as small as 370 SF and as big as 13,540 SF

In [16]:
```
#looking at the zipcodes in King County
df['zipcode'].value_counts()
```

Out[16]:
```
98001    43
98092    41
98030    37
98006    33
98053    29
         ..
98007     2
98108     2
98155     2
98148     1
98188     1
Name: zipcode, Length: 70, dtype: int64
```

I'd like to run a statistical analysis now that the data is clean, and find out correlations as per below:

In [17]:
```
#statistical correlations to price
df.corr()['price'].sort_values(ascending=False)
```

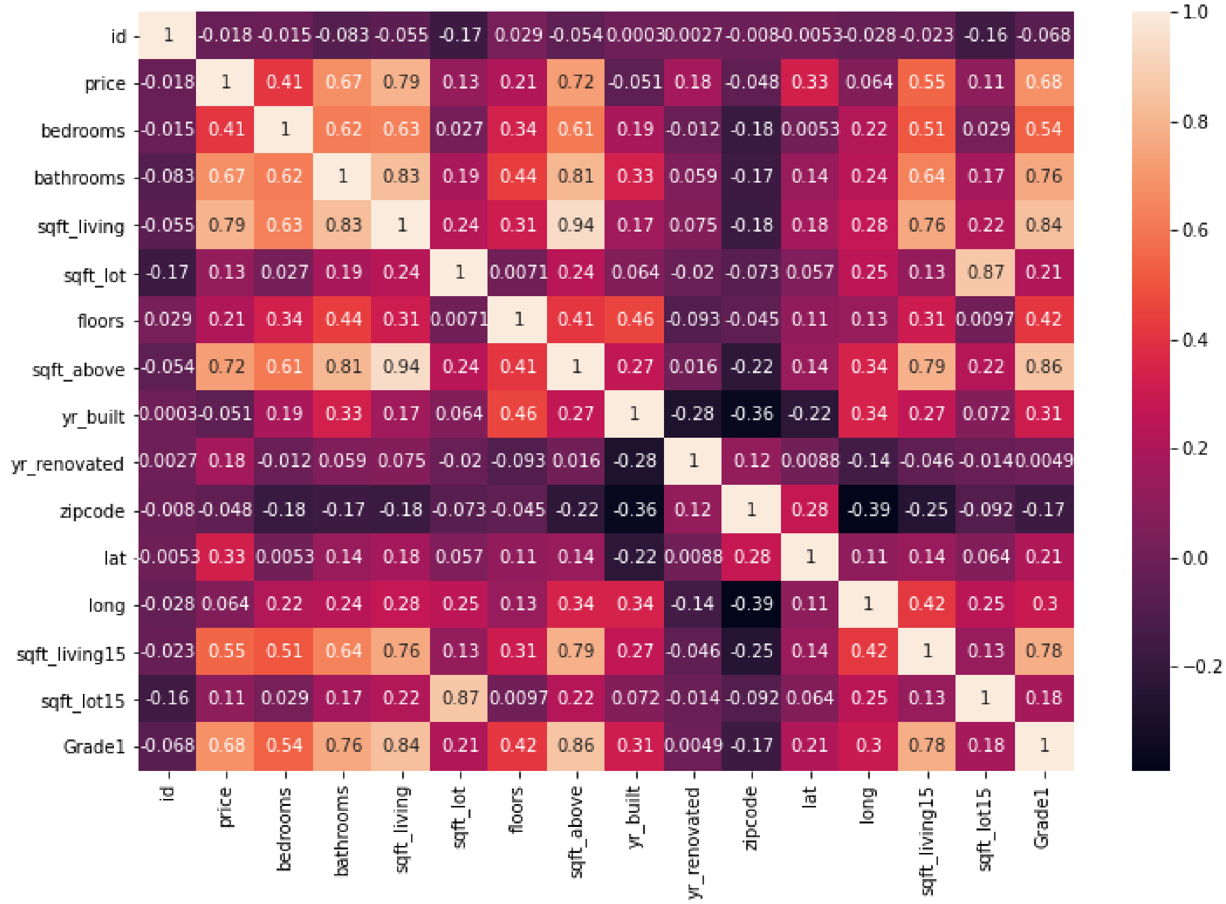
Out[17]:
```
price            1.00
sqft_living      0.79
sqft_above       0.72
Grade1           0.68
bathrooms        0.67
sqft_living15    0.55
bedrooms         0.41
lat              0.33
floors           0.21
yr_renovated     0.18
sqft_lot         0.13
sqft_lot15       0.11
long             0.06
id              -0.02
zipcode         -0.05
yr_built        -0.05
Name: price, dtype: float64
```

The highest correlation to price can be found in square feet, grade and bathrooms.

4.2.6 Observing correlations on a heat map

In addition, visualizing statistics is useful in a heat map:

In [18]:
```
# generate heatmap to display correlations
corr = df.corr()
f, ax = plt.subplots(figsize=(12, 8))
sns.heatmap(corr, annot=True);
```



```
The heatmap above, creates a colored grade scheme where attributes are
compared and assigned a color based on the level of correlation. So, for
example if you see price on the Y column names to the left and compare it
to itself looking at the X columns names, you'll see where they intersect
a value of 1 and a light color filling the box. Meaning the correlation
between price and price is perfectly correlated obviously. However, the
heatmap is a tool that compare different attributes, and comes handy then,
and you can easily see how the darker the colored boxes get the less
correlated the two attributes are and viceversa.
```

Now that I've completed cleaning and visualization of correlations in the database, I'll proceed to conduct regression analysis. Regarding price, you can see that highest correlators to price are square footage, grade, and bathrooms.

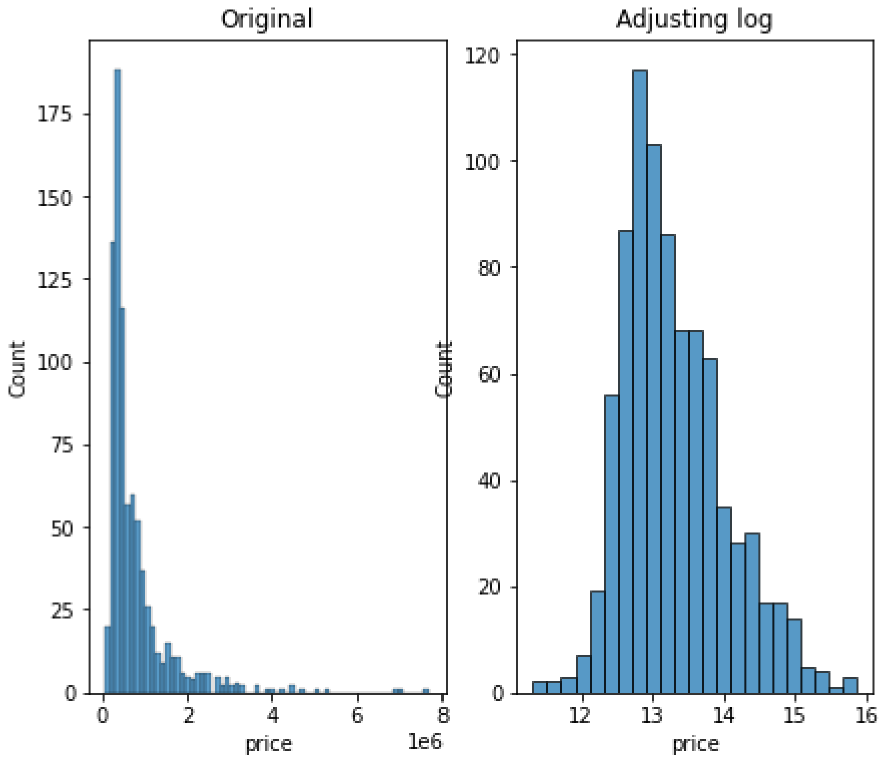
5 Regression analysis and visualizations

I'm going to log normalize the price first. The purpose of log normalization is that is a method of standirizing your data. Below you'll see what this means for price with and without price data being log normalize. The original prices data are skewed to the left, but once I'll log normalize it, you'll see the price standarized data more like a normal distribution. Normalizing is basically a quality data management technique.

Log normalizing price below:

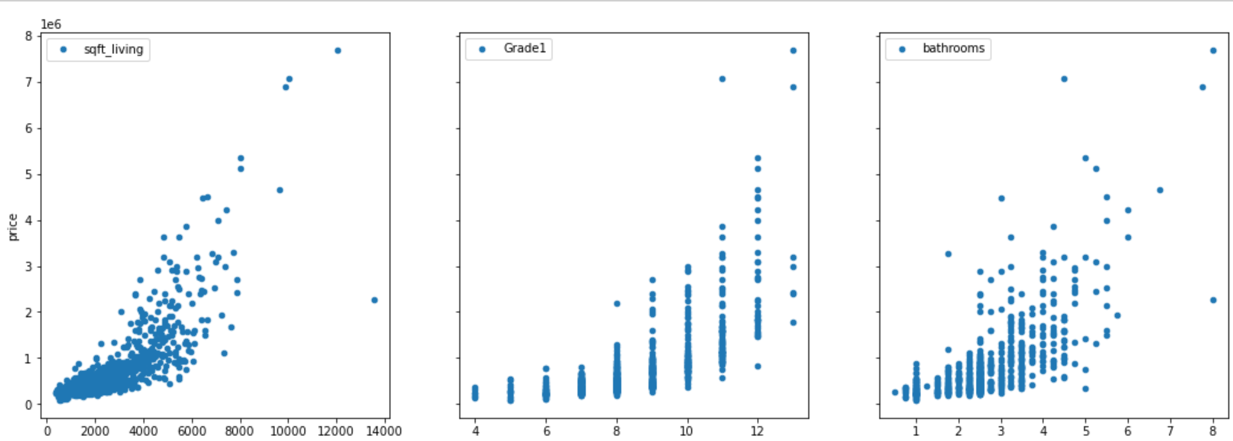
In [19]:
```
fig, ax = plt.subplots(1, 2,figsize=(7,6))

sns.histplot(df['price'], ax=ax[0])
ax[0].set_title('Original')
sns.histplot(np.log(df['price']), ax=ax[1])
ax[1].set_title('Adjusting log')
plt.show()
```



5.0.0.1 Looking at the linearity of the highest correlators to price, namely square footage, grade and bathrooms

In [23]:
```
# visualize the relationship between the predictors and the target (price)
fig, axs= plt.subplots (1,3, sharey= True, figsize=(18,6))
for idx, channel in enumerate (['sqft_living', 'Grade1', 'bathrooms']):
    df.plot(kind= 'scatter', x=channel, y='price', ax=axs[idx], label=chann
plt.legend()
plt.show()
```

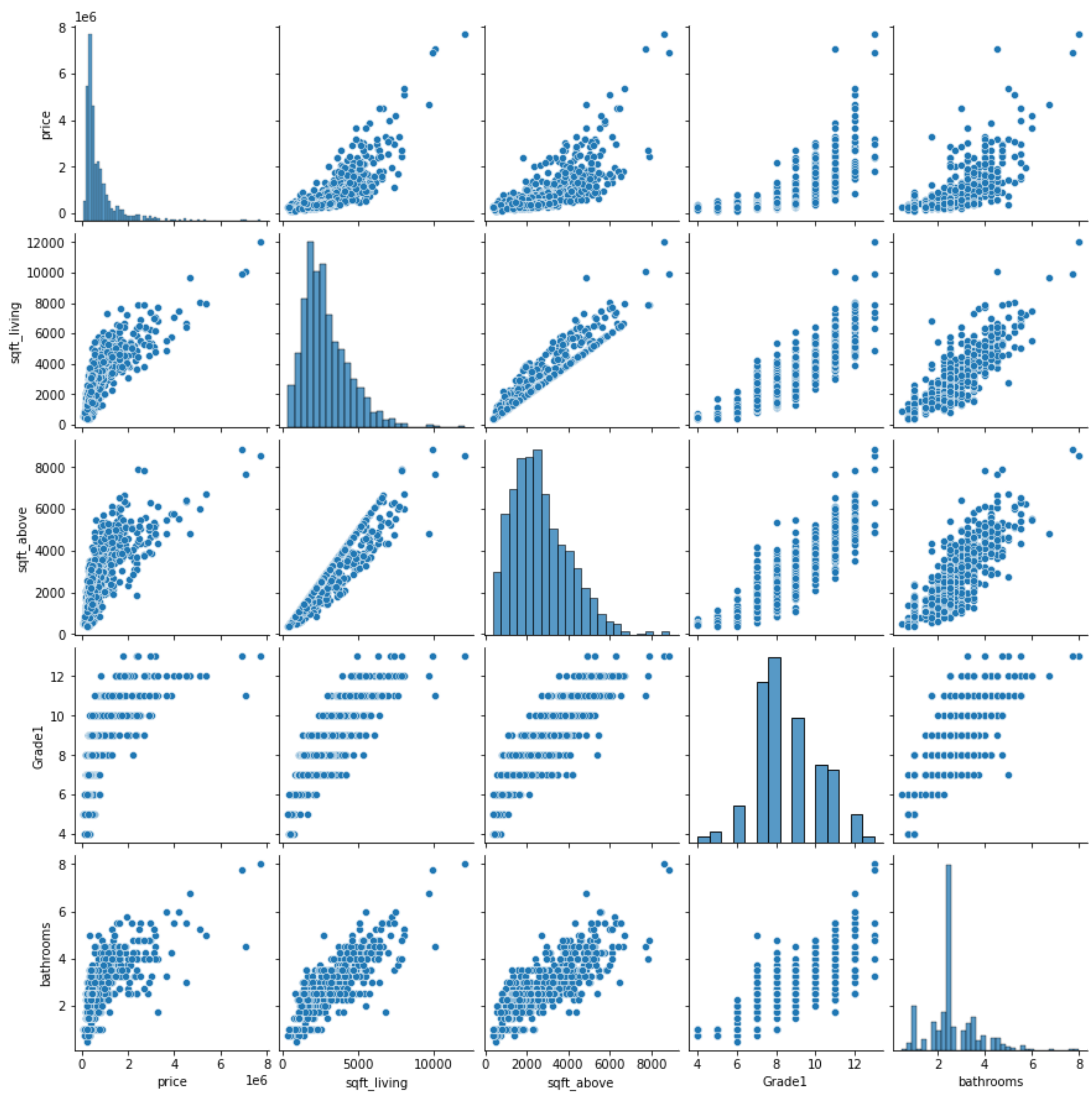


I observe that the most linear is square foot living area, and that there are some outliers (I could to take out outliers from the regression calculation)

In [37]:
```
# drop this record by using the record the index
df.drop(12764, inplace=True)
```

Python also allows me to look at other columns, and their linearity, I'm running it for visualization purposes

In [48]:
```
#exploring other correlations
df_pairplot = df[['price','sqft_living','sqft_above','Grade1','bathrooms']]
sns.pairplot(df_pairplot)
plt.show()
```



Visualizing these relationships we can see the linearity. We can take these relationships and run the model. the first one I would like to pick is highly correlated to price and has good linearity, that is sqft_living.

5.0.1 Running a simple regression in Stats model with SF as a predictor independent variable, and price as a dependent variable

In [30]:
```
# import libraries
import statsmodels.api as sm
import statsmodels.formula.api as smf
# build the formula
f = 'price~sqft_living'

# create a fitted model in one line
model=smf.ols(formula=f, data=df ).fit()
```

The model has been created and next I'll fun a regression diagnostic summary.

5.0.2 Regression Diagnostics Summary

In [31]:
```
#coding for the model summary
model.summary()
```

Out[31]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.631
Model:	OLS	**Adj. R-squared:**	0.630
Method:	Least Squares	**F-statistic:**	1423.
Date:	Sat, 29 Oct 2022	**Prob (F-statistic):**	2.21e-182
Time:	16:08:37	**Log-Likelihood:**	-12127.
No. Observations:	835	**AIC:**	2.426e+04
Df Residuals:	833	**BIC:**	2.427e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-3.755e+05	3.53e+04	-10.648	0.000	-4.45e+05	-3.06e+05
sqft_living	399.3236	10.587	37.719	0.000	378.544	420.103

Omnibus:	395.468	**Durbin-Watson:**	1.918
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	4905.344
Skew:	1.820	**Prob(JB):**	0.00
Kurtosis:	14.302	**Cond. No.**	6.90e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.9e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

The important factors to observe in the model and how to think of them from the above model summary are as follows,

The Dependent Variable: Price.

R-Squared: 63%. An important measure that compares to the baseline model, its a fitness test and with this value I'm not as confident that this model works.

R-Squared Coefficient determination , is a "goodness of fit" of the regression model. R-squared is also called the baseline model, and in this cases it indicates almost 63% can be explained by the model and how the data fit the model.

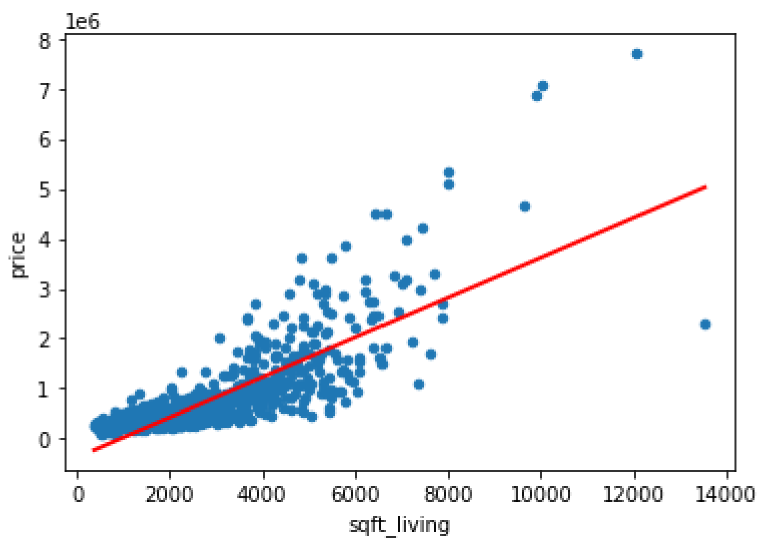
The F-statistic or P-Value: is the provability that a sample like this would yield the above results, and wether or not the model's veridict on the null hypthesis will consistently represent the population. Since this model yielded a p-value less than 0.05 we can reject the null hypothesis and know that this test is statistically significant.

5.0.3 Drawing a prediction line X(square feet living) and Y(price)

In [32]:
```
# create a DataFrame with the minimum and maximum values of sf
X_new=pd.DataFrame({'sqft_living': [df.sqft_living.min(), df.sqft_living.ma
print(X_new.head())
# make predictions for those x values and store them
preds= model.predict(X_new)
print(preds)

# first, plot the observed data and the least squares line
df.plot(kind= 'scatter', x='sqft_living', y='price')
plt.plot(X_new, preds, c='red', linewidth =2)
plt.show()
```

```
   sqft_living
0          370
1        13540
0   -227744.63
1   5031347.07
dtype: float64
```

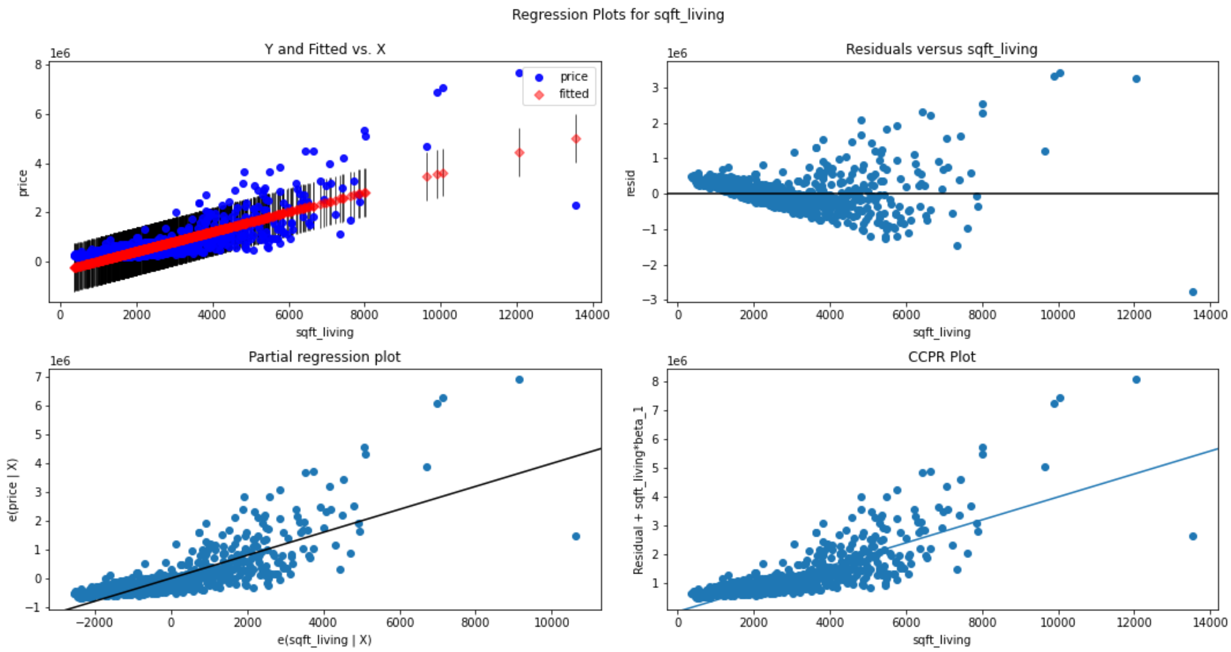


Now we have created a prediction line. However we also have to test assumptions. We have tested linearity. But in addition there are other assumptions I'll be testing for such as independence and homoscedeasticy.

5.0.4 Plots visualize error term for variance and heteroscedasticy

Q-Q Plots allow us to check that errors are indedpented , knowing that error from one point doesnt tell you anything about another error at another point, the q-q plots scatter plot of residuals vs. sqft living does show. if a pattern is shown it can indicate that errors have the same variance(funnel for living does show, if a pattern is shown it can indicate that errors have the same variance(funnel for example), and therefore not homoscedastic, rather heterocedastic.

In [34]:
```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_living", fig=fig)
plt.show()
```

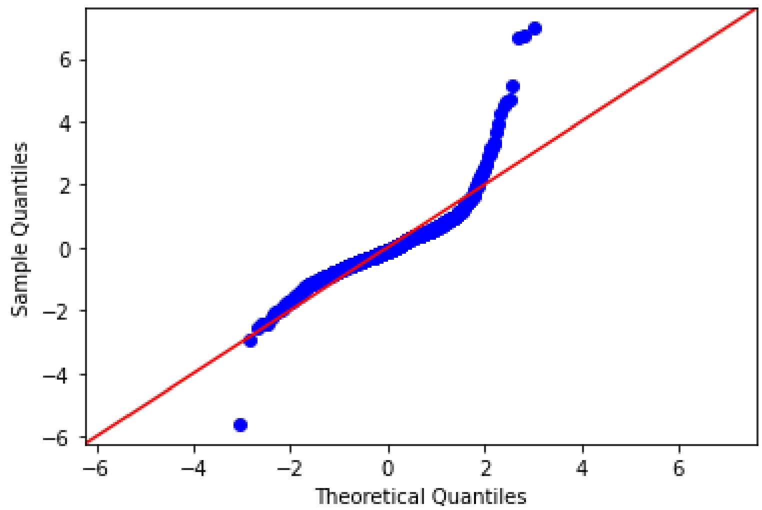


The residuals vs. sqft living graph do look like a funnel, and the assumption of homoscedaticity is not passed.

5.0.5 Checking for normality assumptions by creating QQ plots

The normality assumption in this model is a bit off according to the Q-Q plot below. If the two distributions which we are comparing are exactly equal then the points on the Q-Q plot will perfectly lie on a straight line y = x.

In [35]:
```
# Code for QQ-plot here
import scipy.stats as stats
residuals =model.resid
sm.graphics.qqplot (residuals, dist=stats.norm, line='45', fit=True)
plt.show()
```



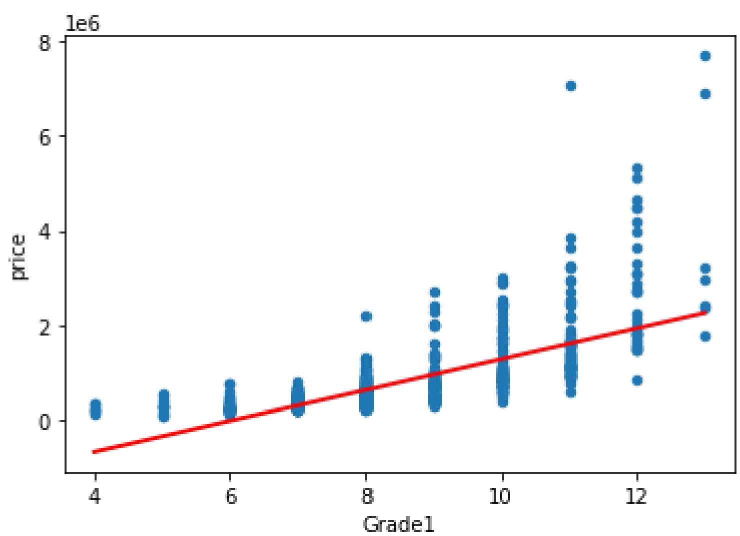
We can see above that the normality assumptions is not met.

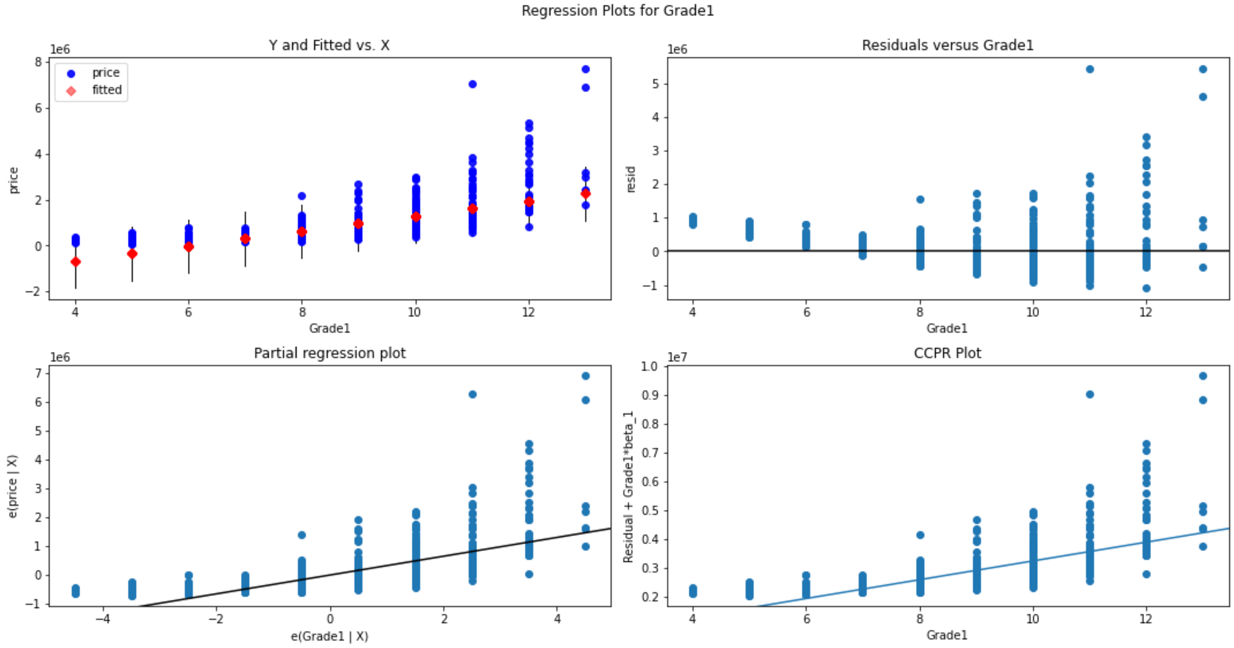
5.0.6 Repeating the above also for Grade as a predictor

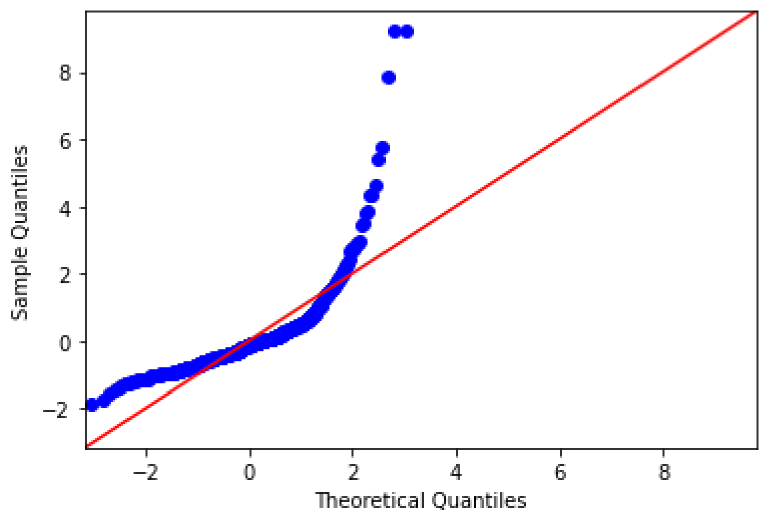
I'm repeating the process above for sqf_living for grade instead

In [46]:
```
# code for model, prediction line plot, heteroscedasticity check and QQ nor
#step 1 through 3 is looking at database as a whole.
#Step 4 run Simple regression on radio only, just we did on TV only
f='price~Grade1'
model= smf.ols(formula=f, data=df).fit()
print ('R-Squared',model.rsquared)
print (model.params)
#get regression diagnostics
model.summary()
#Step 6 Draw a prediction line on scatter plot
X_new= pd.DataFrame({'Grade1':[df.Grade1.min(),df.Grade1.max()]});
preds= model.predict(X_new)
df.plot(kind='scatter', x='Grade1', y='price');
plt.plot(X_new,preds, c='red', linewidth=2);
plt.show()
#Visualize error term for variance Heteroscedasticity
fig= plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "Grade1", fig=fig)
plt.show()
#Normality check with QQ Plot
import scipy.stats as stats
residuals= model.resid
fig=sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True)
```

```
R-Squared 0.4652008564849478
Intercept   -1981572.05
Grade1        325851.16
dtype: float64
```





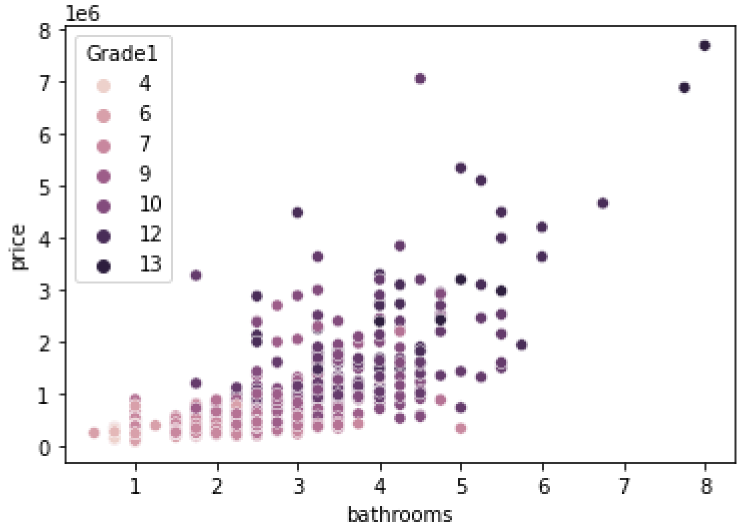


Same as before the assumptions are not quite all met.

5.0.7 Exploring more Correlations & Regression

I'll be exploring more correlations, and will try to improve model on sqft living.

In [56]:
```
sns.scatterplot(data =df,x = 'bathrooms',y= 'price',hue ='Grade1');
```



The number of bathrooms is positively correlated to price and it also helps us to conclude that the better the grade of a house, the more expensive it is

5.0.8 Creating X in a train and test models

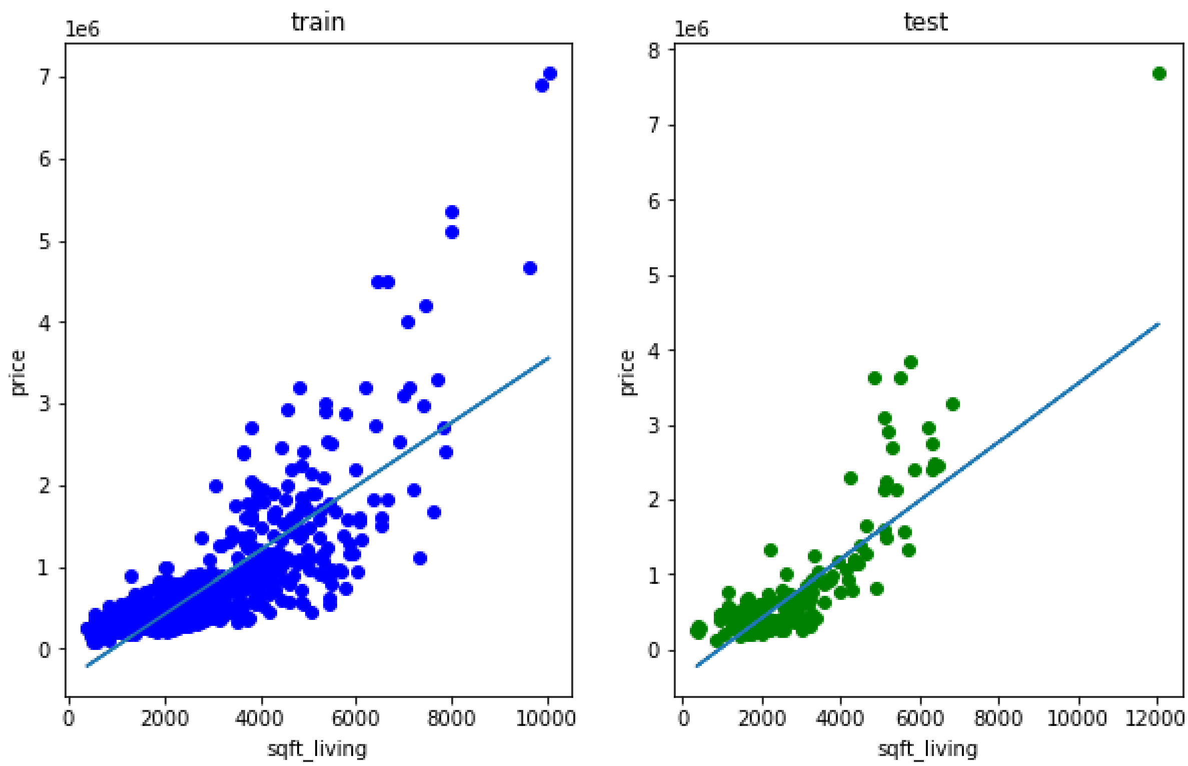
In [58]:
```
X = df[['sqft_living','sqft_above','bathrooms','bedrooms','Grade1']]
y = df['price']
```

In [59]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

In [60]:
```
sqft = LinearRegression()
sqft.fit(X_train[['sqft_living']], y_train)
sqft.score(X_train[['sqft_living']], y_train)
y_hat_train = sqft.predict(X_train[['sqft_living']])
y_hat_test = sqft.predict(X_test[['sqft_living']])
```

In [61]:
```
plt.figure(figsize=(10,6))
plt.subplot(1,2,1)
plt.scatter(X_train[['sqft_living']], y_train, color = "blue")
plt.plot(X_train[['sqft_living']] ,y_hat_train)
plt.xlabel('sqft_living')
plt.ylabel('price')
plt.title('train')

plt.subplot(1,2,2)
plt.scatter(X_test[['sqft_living']], y_test, color = "green")
plt.plot(X_test[['sqft_living']] ,y_hat_test)
plt.xlabel('sqft_living')
plt.ylabel('price')
plt.title('test');
```



Both train and set with linear correlation

In [75]:
```
reg = sm.add_constant(X, has_constant='add')
model = sm.OLS(y, X)
result1 = model.fit()
result1.summary()
```

Out[75]:

OLS Regression Results

Dep. Variable:	price	**R-squared (uncentered):**	0.829
Model:	OLS	**Adj. R-squared (uncentered):**	0.828
Method:	Least Squares	**F-statistic:**	803.4
Date:	Sat, 29 Oct 2022	**Prob (F-statistic):**	6.86e-315
Time:	17:54:14	**Log-Likelihood:**	-12070.
No. Observations:	834	**AIC:**	2.415e+04
Df Residuals:	829	**BIC:**	2.417e+04
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
sqft_living	563.3322	32.620	17.270	0.000	499.305	627.359
sqft_above	-151.8250	36.106	-4.205	0.000	-222.695	-80.955
bathrooms	7.487e+04	3.03e+04	2.470	0.014	1.54e+04	1.34e+05
bedrooms	-1.476e+05	1.96e+04	-7.521	0.000	-1.86e+05	-1.09e+05
Grade1	-1.164e+04	1.01e+04	-1.147	0.252	-3.16e+04	8280.837

Omnibus:	368.835	**Durbin-Watson:**	1.910
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3688.194
Skew:	1.736	**Prob(JB):**	0.00
Kurtosis:	12.699	**Cond. No.**	8.51e+03

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 8.51e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

5.0.9 Coefficients Determination, P-Values & Model Explained

The important factors to observe in the model above are as follows, The Dependent Variable: Price. R-Squared: 83%, an important measure that compares to the baseline model, its a fitness test and with this value I'm confident that this model works. The sum of squares is divided by Total Sum

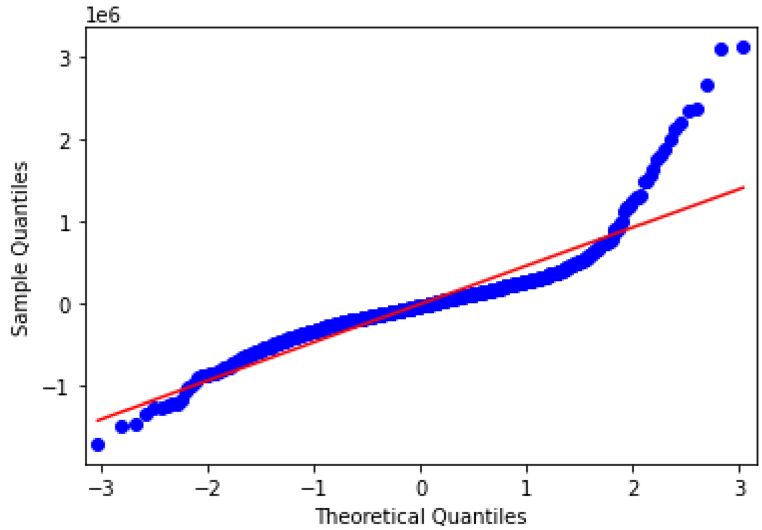
Squares. R-Squared Coefficient determination , in this model is a btter "goodness of fit". This value of 83%, is a better interpretation of how well the regression model fits the observed data values.

The F-statistic o P-Value: is the provability that. asample like this would yield the above results, and wether or not the model's veridict on the null hypthesis will consistently represent the population. Since this model yielded a p-value less than 0.05 we can reject the null hypothesis and know that this test is statistically significant

Next I'll check of the Normality of this test...

5.0.10 Checking the Normality Residual pattern

In [76]:
```
qqplot = sm.qqplot(result1.resid,line ='s',dist=stats.norm)
```



Although not perfectly normal, this normality is good enough for me and much better in this model.

5.0.11 Distributing the Data Normally

When we modify the data and distribute it normally, the model can be further improved.

In [79]:
```
h = np.log(df['price'])
reg = sm.add_constant(X, has_constant='add')
model = sm.OLS(h, X)
result2 = model.fit()
result2.summary()
```

Out[79]:

OLS Regression Results

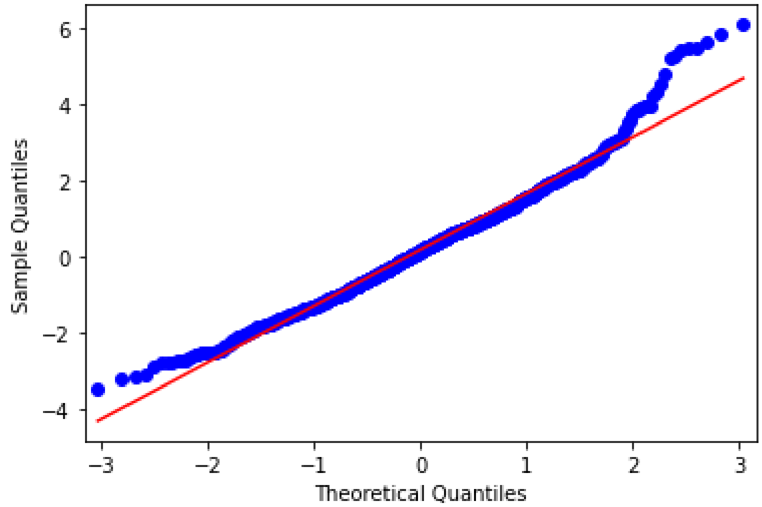
Dep. Variable:	price	**R-squared (uncentered):**	0.987
Model:	OLS	**Adj. R-squared (uncentered):**	0.987
Method:	Least Squares	**F-statistic:**	1.295e+04
Date:	Sat, 29 Oct 2022	**Prob (F-statistic):**	0.00
Time:	18:01:20	**Log-Likelihood:**	-1518.7
No. Observations:	834	**AIC:**	3047.
Df Residuals:	829	**BIC:**	3071.
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
sqft_living	-0.0004	0.000	-3.871	0.000	-0.001	-0.000
sqft_above	-0.0010	0.000	-8.527	0.000	-0.001	-0.001
bathrooms	0.0531	0.097	0.547	0.585	-0.138	0.244
bedrooms	0.7180	0.063	11.415	0.000	0.595	0.841
Grade1	1.6621	0.033	51.100	0.000	1.598	1.726

Omnibus:	55.755	**Durbin-Watson:**	1.920
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	75.390
Skew:	0.562	**Prob(JB):**	4.26e-17
Kurtosis:	3.953	**Cond. No.**	8.51e+03

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 8.51e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [80]:
```
qqplot = sm.qqplot(result2.resid,line ='s',dist=stats.norm)
```



The R-Squared at 98% when the data is normally distributed

6 Conclusion & Results

6.0.1 People in King County who would like to buy or sell should be aware that there are attributes in their homes that contribute directly to the price of the home in this market

6.0.2 People who own homes or would like to buy homes King County should know that Square Footage, Grade, and Bathrooms, are the highest and the most directly correlated to home prices

6.0.3 Prices of Homes in King county are highly and mostly influenced by the Square Footage, the most important factor and highest contributor to the price of a home in King County

6.0.4 The better built homes, meaning homes with higher grades, increase the price of a King County Home. So Grade level matters when selling a home in this market