Phase 2 Project

- Authors: Jules Mejia
- Instructor name: Hardik

- Date : Sunday 2 July 2023

Overview

This project is an opportunity to apply the concepts and techniques I have learned in Phase 2 and apply it using real world data. Through exploratory data analysis and linear regression, this report will generate 3 key insights that will aid in the decision making of a residential builder looking to make a presence in King County. It is important to understand the relationship each variable has with price so the residential builder can have a blueprint for their designs.

Business Problem

A residential builder based in the USA has had much success on the east coast. They are looking to expand their market into the west coast of the USA, starting in King County, Washington. The residential builder would like to understand what factors influence the price of a house in that area.

The aim is to generate designs of a house based on those 5 factors and offer it as an option in King County.

The model generated will be based on inference rather than prediction. I want to identify which features have a strong relationship with house price and see their effect.

Exploratory Data Analysis

The dataset to be used for this model comes houses that were sold between 2014 and 2015 in King County, Washington. There are 21,597 entries containing a wide variety of information including features of the house (number bedrooms, bathrooms), location (zipcode, latitude, longitude) and ratings (view, grade).

In [1]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.formula.api as smf

import scipy.stats as stats

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

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

```
C:\Users\jules\anaconda3\envs\learn-env\lib\site-packages\statsmodels\tsa
\base\tsa_model.py:7: FutureWarning: pandas.Int64Index is deprecated and w
ill be removed from pandas in a future version. Use pandas.Index with the
appropriate dtype instead.
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
C:\Users\jules\anaconda3\envs\learn-env\lib\site-packages\statsmodels\tsa
\base\tsa_model.py:7: FutureWarning: pandas.Float64Index is deprecated and
will be removed from pandas in a future version. Use pandas.Index with the
appropriate dtype instead.
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
```

Out[1]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wate
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	

5 rows × 21 columns

In [2]:

```
df.describe()
```

Out[2]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04
mean	4.580474e+09	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04
std	2.876736e+09	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02
25%	2.123049e+09	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06

In [3]:

```
# Check data types, identify columns for cleaning

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  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
```

In [4]:

```
# 2 object columns, Date and sqft_basement
# Date column does not affect target variable price. Does not require cleaning
# Investigate sqft_basement column

df['sqft_basement'].unique()
```

Out[4]:

```
array(['0.0', '400.0', '910.0', '1530.0', '?', '730.0', '1700.0', '300.0',
       '970.0', '760.0', '720.0', '700.0', '820.0', '780.0', '790.0',
       '330.0', '1620.0', '360.0', '588.0', '1510.0', '410.0', '990.0',
       '600.0', '560.0', '550.0', '1000.0', '1600.0', '500.0', '1040.0',
       '880.0', '1010.0', '240.0', '265.0', '290.0', '800.0', '540.0',
       '710.0', '840.0', '380.0', '770.0', '480.0', '570.0', '1490.0',
       '620.0', '1250.0', '1270.0', '120.0', '650.0', '180.0', '1130.0',
       '450.0', '1640.0', '1460.0', '1020.0', '1030.0', '750.0', '640.0',
       '1070.0', '490.0', '1310.0', '630.0', '2000.0', '390.0', '430.0',
       '850.0', '210.0', '1430.0', '1950.0', '440.0', '220.0', '1160.0',
       '860.0', '580.0', '2060.0', '1820.0', '1180.0', '200.0', '1150.0',
       '1200.0', '680.0', '530.0', '1450.0', '1170.0', '1080.0', '960.0',
       '280.0', '870.0', '1100.0', '460.0', '1400.0', '660.0', '1220.0',
       '900.0', '420.0', '1580.0', '1380.0', '475.0', '690.0', '270.0',
       '350.0', '935.0', '1370.0', '980.0', '1470.0', '160.0', '950.0',
       '50.0', '740.0', '1780.0', '1900.0', '340.0', '470.0', '370.0',
       '140.0', '1760.0', '130.0', '520.0', '890.0', '1110.0', '150.0',
       '1720.0', '810.0', '190.0', '1290.0', '670.0', '1800.0', '1120.0',
       '1810.0', '60.0', '1050.0', '940.0', '310.0', '930.0', '1390.0',
       '610.0', '1830.0', '1300.0', '510.0', '1330.0', '1590.0', '920.0',
       '1320.0', '1420.0', '1240.0', '1960.0', '1560.0', '2020.0',
       '1190.0', '2110.0', '1280.0', '250.0', '2390.0', '1230.0', '170.0',
       '830.0', '1260.0', '1410.0', '1340.0', '590.0', '1500.0', '1140.0',
       '260.0', '100.0', '320.0', '1480.0', '1060.0', '1284.0', '1670.0',
       '1350.0', '2570.0', '1090.0', '110.0', '2500.0', '90.0', '1940.0',
       '1550.0', '2350.0', '2490.0', '1481.0', '1360.0', '1135.0',
       '1520.0', '1850.0', '1660.0', '2130.0', '2600.0', '1690.0',
       '243.0', '1210.0', '1024.0', '1798.0', '1610.0', '1440.0',
       '1570.0', '1650.0', '704.0', '1910.0', '1630.0', '2360.0',
       '1852.0', '2090.0', '2400.0', '1790.0', '2150.0', '230.0', '70.0',
       '1680.0', '2100.0', '3000.0', '1870.0', '1710.0', '2030.0',
       '875.0', '1540.0', '2850.0', '2170.0', '506.0', '906.0', '145.0',
       '2040.0', '784.0', '1750.0', '374.0', '518.0', '2720.0', '2730.0',
       '1840.0', '3480.0', '2160.0', '1920.0', '2330.0', '1860.0',
       '2050.0', '4820.0', '1913.0', '80.0', '2010.0', '3260.0', '2200.0',
       '415.0', '1730.0', '652.0', '2196.0', '1930.0', '515.0', '40.0',
       '2080.0', '2580.0', '1548.0', '1740.0', '235.0', '861.0', '1890.0',
       '2220.0', '792.0', '2070.0', '4130.0', '2250.0', '2240.0',
       '1990.0', '768.0', '2550.0', '435.0', '1008.0', '2300.0', '2610.0',
       '666.0', '3500.0', '172.0', '1816.0', '2190.0', '1245.0', '1525.0',
       '1880.0', '862.0', '946.0', '1281.0', '414.0', '2180.0', '276.0',
       '1248.0', '602.0', '516.0', '176.0', '225.0', '1275.0', '266.0',
       '283.0', '65.0', '2310.0', '10.0', '1770.0', '2120.0', '295.0',
       '207.0', '915.0', '556.0', '417.0', '143.0', '508.0', '2810.0',
       '20.0', '274.0', '248.0'], dtype=object)
```

In [5]:

```
# '?' is present. Check how many instances this occurs

df['sqft_basement'].value_counts()
```

Out[5]:

```
0.0       12826
?           454
600.0       217
500.0       209
700.0       208
           ...
1920.0        1
3480.0        1
2730.0        1
2720.0        1
248.0         1
Name: sqft_basement, Length: 304, dtype: int64
```

In [6]:

```
# '?' has 454 instances. Could be a placeholder or does not have a basement
# 0.0 has 12,826 instances out of 21,597 . This means 59% of houses do not have a basemen
# Therefore I will treat '?' as 0.0

df['sqft_basement'] = df['sqft_basement'].replace("\?", 0, regex=True)
df['sqft_basement'] = df['sqft_basement'].astype(float)
```

In [7]:

```
# Re-check 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  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  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
```

In [8]:

```
# Identify NaN values

df.isna().sum()
```

Out[8]:

```
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 [9]:

```
# Investigate view column

df['view'].unique()
```

Out[9]:

```
array([ 0., nan,  3.,  4.,  2.,  1.])
```

In [10]:

```
df['view'].value_counts()
```

Out[10]:

```
0.0    19422
2.0      957
3.0      508
1.0      330
4.0      317
Name: view, dtype: int64
```

In [11]:

```
# View looks to be categorical data (rating system)
# Majority of houses do not have a view rating plus small amount of NaN values
# Therefore I will treat NaN values as 0.

df['view'].fillna(0, inplace=True)
```

In [12]:

```
# Investigate waterfront column

df['waterfront'].unique()
```

Out[12]:

```
array([nan,  0.,  1.])
```

In [13]:

```
df['waterfront'].value_counts()
```

Out[13]:

```
0.0    19075
1.0      146
Name: waterfront, dtype: int64
```

In [14]:

```
# Waterfront looks to be categorical data (yes or no)
# Majority of houses do not have a waterfront. Makes sense in terms of real world applica
# Therefore I will treat NaN values as 0.

df['waterfront'].fillna(0, inplace=True)
```

In [15]:

```
# Investigate yr_renovated column

df['yr_renovated'].unique()
```

Out[15]:

```
array([   0., 1991.,   nan, 2002., 2010., 1992., 2013., 1994., 1978.,
       2005., 2003., 1984., 1954., 2014., 2011., 1983., 1945., 1990.,
       1988., 1977., 1981., 1995., 2000., 1999., 1998., 1970., 1989.,
       2004., 1986., 2007., 1987., 2006., 1985., 2001., 1980., 1971.,
       1979., 1997., 1950., 1969., 1948., 2009., 2015., 1974., 2008.,
       1968., 2012., 1963., 1951., 1962., 1953., 1993., 1996., 1955.,
       1982., 1956., 1940., 1976., 1946., 1975., 1964., 1973., 1957.,
       1959., 1960., 1967., 1965., 1934., 1972., 1944., 1958.])
```

In [16]:

```
df['yr_renovated'].value_counts()
```

Out[16]:

```
0.0       17011
2014.0       73
2013.0       31
2003.0       31
2007.0       30
          ...
1951.0        1
1953.0        1
1946.0        1
1976.0        1
1948.0        1
Name: yr_renovated, Length: 70, dtype: int64
```

In [17]:

```
# Again yr_renovated looks to be categorical data (years)
# Majority have not been renovated. Makes sense in terms of real world application
# Therefore I will treat NaN values as 0.

df['yr_renovated'].fillna(0, inplace=True)
```

In [18]:

```
# Check NaN values

df.isna().sum()
```

Out[18]:

```
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 [19]:

```
# Drop columns id and date because they definitely do not affect price
# Use a heatmap to visualise correlation between variables

df = df.drop(columns=['id', 'date'])
```

In [20]:

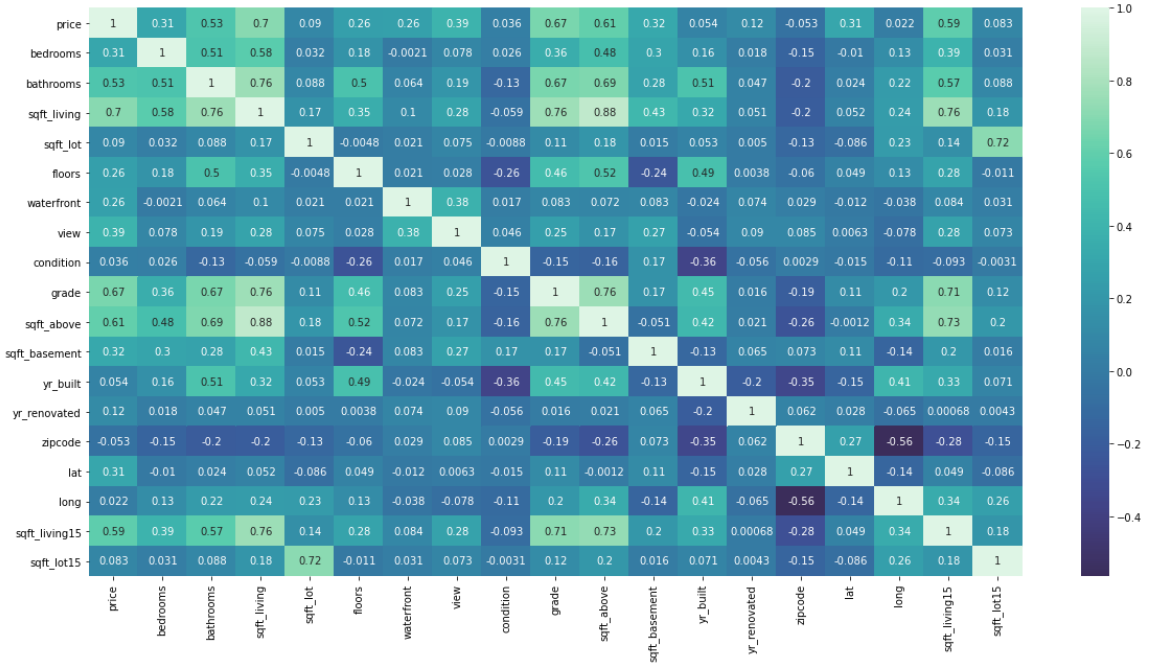
```
df.head()
```

Out[20]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	g
0	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	3	
1	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	3	
2	180000.0	2	1.00	770	10000	1.0	0.0	0.0	3	
3	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	5	
4	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	3	

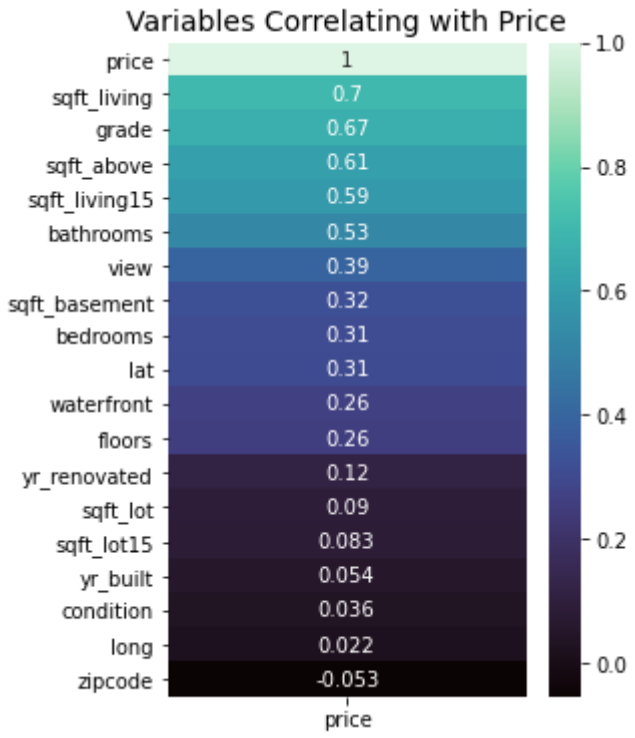
In [21]:

```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(), annot=True, center=0, cmap='mako');
```



In [22]:

```
# Now ordered in a way to see ranking of correlations
# This will help in decision making of what variables to consider during transformations

price_corr = df.corr()[['price']].sort_values(by='price', ascending=False)

plt.figure(figsize=(4, 6))
heatmap = sns.heatmap(price_corr, annot=True, cmap='mako')
heatmap.set_title('Variables Correlating with Price', fontsize=14);
plt.savefig("Images/df_price_corr.png", bbox_inches='tight');
```



Iteration 1 (Baseline Model)

Now that I am happy with my EDA, I will generate a baseline model. This model will contain all data and no transformations. It will be compared to subsequent iterations to observe the effect of the transformations.

In [23]:

```
continuous = ['sqft_living', 'sqft_lot', 'sqft_above', 'sqft_basement', 'lat', 'long', 's
categoricals = ['bedrooms', 'bathrooms', 'floors', 'waterfront', 'view', 'condition', 'gr

# Log transform and normalize
df_cont = df[continuous]

for col in categoricals:
    df_cont[col] = df[col].astype('category')

# Perform one-hot encoding
df_cat = pd.get_dummies(df_cont[categoricals], prefix=categoricals, drop_first=True)

df_baseline = pd.concat([df_cont, df_cat], axis=1)

X = df_baseline.drop('price', axis=1)
y = df_baseline['price']

X_int = sm.add_constant(X)
model = sm.OLS(y,X_int).fit()
model.summary()
```

```
<ipython-input-23-2b2b64093d2f>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  df_cont[col] = df[col].astype('category')
<ipython-input-23-2b2b64093d2f>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  df_cont[col] = df[col].astype('category')
<ipython-input-23-2b2b64093d2f>:8: SettingWithCopyWarning:
```

Iteration 1 Model Comments

So far the P-values indicate all independent variables except sqft_basement are statistically significant. That would make sense given that all the features we have chosen would have an effect on the price of the house. The adjusted R-squared is a great value however there are many categorical variables that are not statistically significant leading to high variance.

Skew = 2.235 indicates the model is positively skewed

Kurtosis = 43.117 indicates this is a leptokurtic curve. Data likely has heavy tails and many outliers

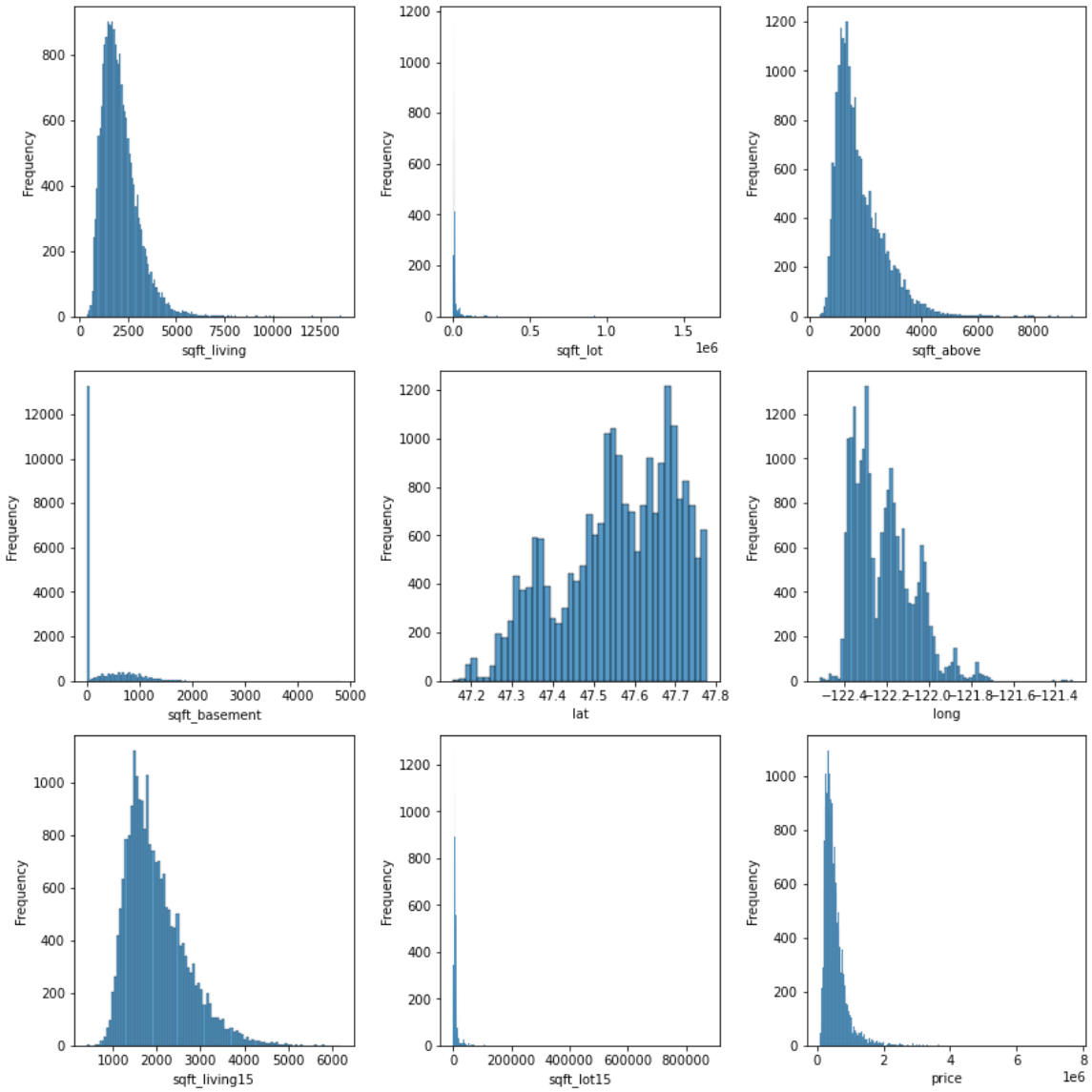
However, the aim is to narrow down which independent variables has the strongest effect on the target variable. The builder will then be able to focus their design and budget on those parameters.

Distributions and KDE

Visualise the distribution of each variable. If they are not normally distributed, determine the next step to evaluate the variable.

In [24]:

```
fig, axs = plt.subplots(3, 3, figsize=(12, 12))

columns = ['sqft_living', 'sqft_lot', 'sqft_above', 'sqft_basement', 'lat', 'long', 'sqft

for i, column in enumerate(columns):
    row = i // 3
    col = i % 3

    sns.histplot(data=df, x=column, ax=axs[row, col])
    axs[row, col].set_xlabel(column)
    axs[row, col].set_ylabel('Frequency')

plt.tight_layout()

plt.savefig("images/dist_it1", bbox_inches='tight')

plt.show()
```



Distribution and KDE comments

Normal distribution with positive skew

- price
- sqft_living
- sqft_lot
- sqft_above
- sqft_basement
- long
- sqft_living15

Normal distribution with negative skew

- lat

Not a great distribution. Maybe outliers greatly affecting it

- sqft_lot15

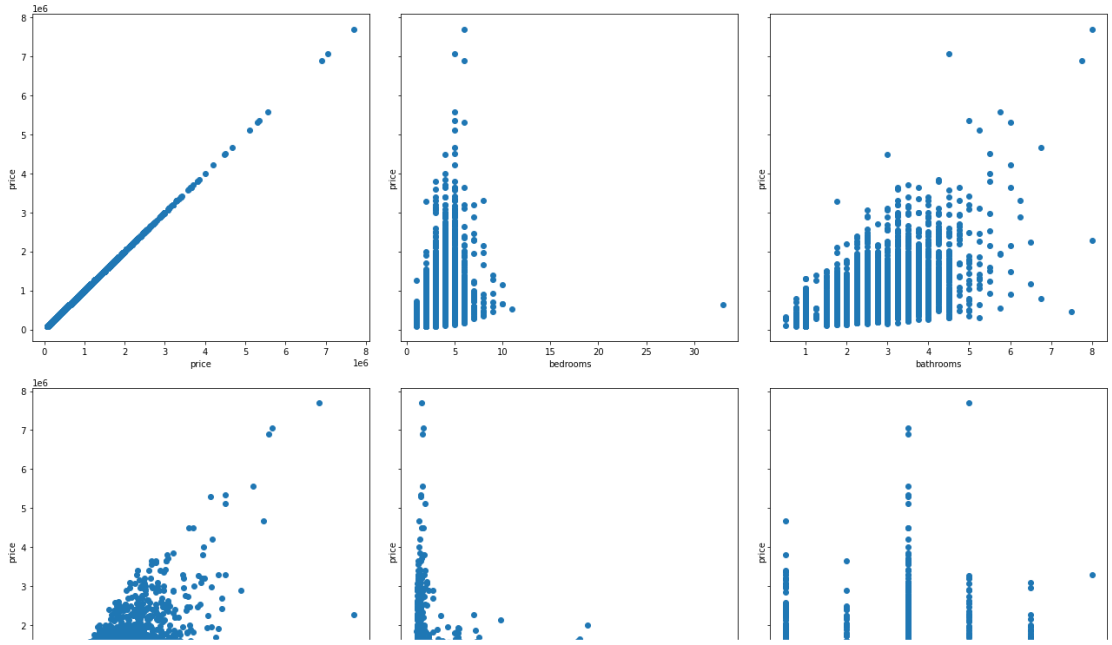
Many of the histograms also exhibit quite large tails. This is an indication of outliers.

Verify the Linearity Assumption

Identify which variables have a linear relationship with the target variable price

In [25]:

```
num_columns = len(df.columns)
num_rows = (num_columns + 2) // 3  # Calculate the number of rows needed to display the s

fig, axs = plt.subplots(num_rows, 3, sharey=True, figsize=(18, 6 * num_rows))

for idx, column in enumerate(df.columns):
    row_idx = idx // 3  # Calculate the row index for the subplot
    col_idx = idx % 3  # Calculate the column index for the subplot

    axs[row_idx, col_idx].scatter(df[column], df['price'])
    axs[row_idx, col_idx].set_xlabel(column)
    axs[row_idx, col_idx].set_ylabel('price')

# Hide empty subplots
if num_columns % 3 != 0:
    for idx in range(num_columns, num_rows * 3):
        row_idx = idx // 3
        col_idx = idx % 3
        plt.delaxes(axs[row_idx, col_idx])

plt.tight_layout()  # Adjust the spacing between subplots

plt.savefig("images/scat_lin_it1", bbox_inches='tight')

plt.show()
```



Linearity Comments

Confirmed as categorical data.

- bedroom
- bathroom
- floors
- waterfront
- view

- condition
- grade

They are clustered but the scatter plot indicate these variables are categorical data.

- yr_built
- yr_renovated
- zipcode

Variables that have a strong linear relationship with price. These variables match up well with the ranking in the correlation heat map.

- price
- sqft_living
- sqft_above
- sqft_living15

This variable has a strong linear relationship if there is a basement present. Otherwise there are many without a basement.

- sqft_basement

Variables that have a weak linear relationship with price.

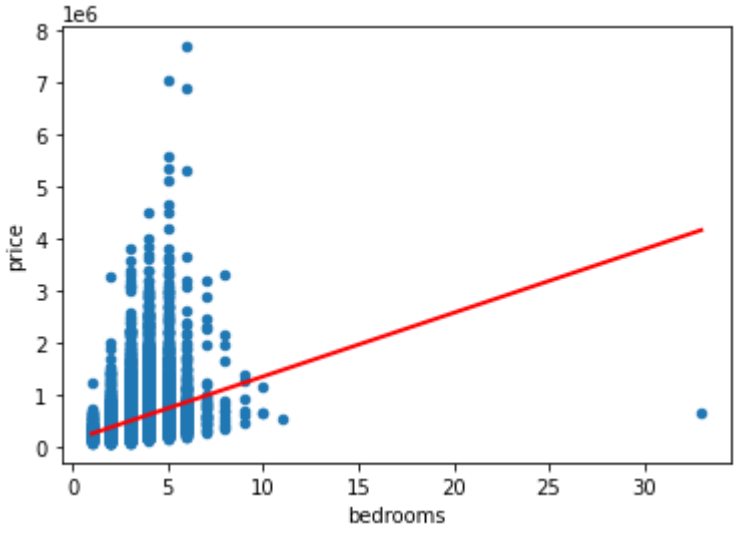
- sqft_lot
- lat
- long
- sqft_lot15

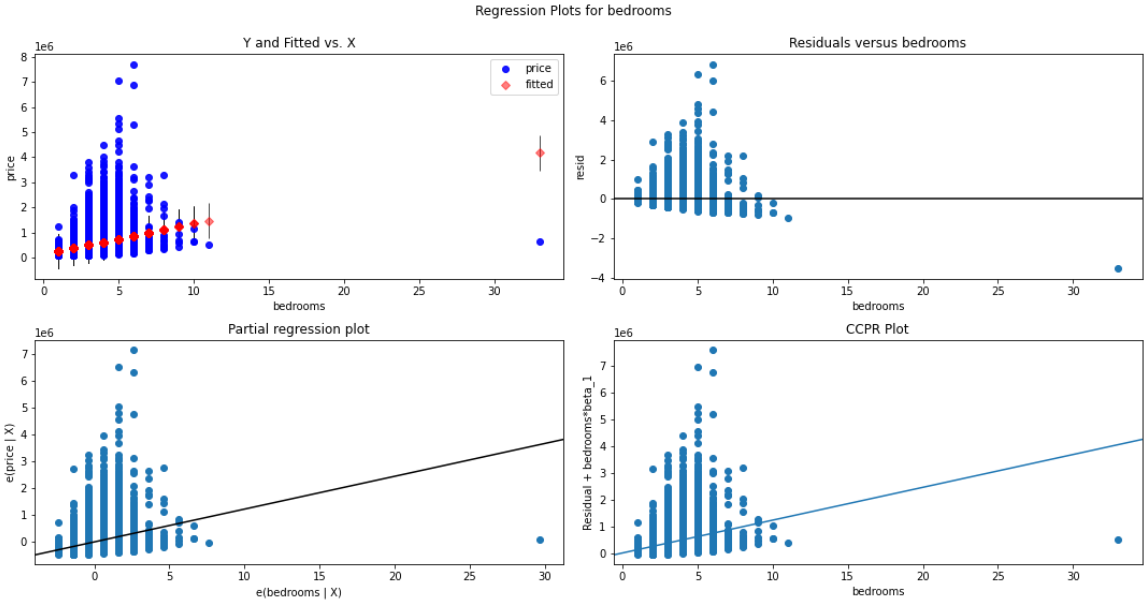
Verify the Normality and Homoscedasticity Assumptions

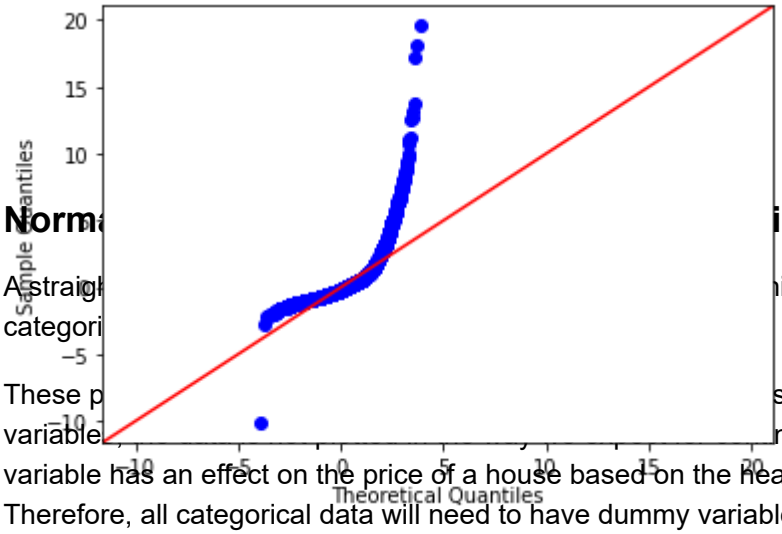
In [26]:

```
data=df
f = 'price~bedrooms'
model = smf.ols(formula=f, data=data).fit()
print ('R-Squared:',model.rsquared)
print (model.params)

X_new = pd.DataFrame({'bedrooms': [data.bedrooms.min(), data.bedrooms.max()]})
preds = model.predict(X_new)

fig, ax = plt.subplots()
data.plot(kind='scatter', x='bedrooms', y='price', ax=ax)
ax.plot(X_new['bedrooms'], preds, c='red', linewidth=2)
plt.show()

fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "bedrooms", fig=fig)
plt.show()

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

```
R-Squared: 0.0953497028373066
Intercept    127199.512685
bedrooms     122464.444174
dtype: float64
```







Norma ical variables

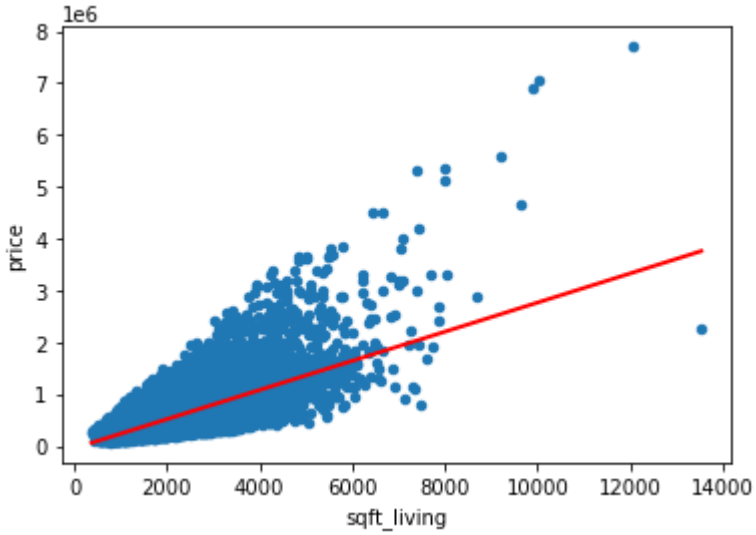
A straigh highlight the importance of transforming categori

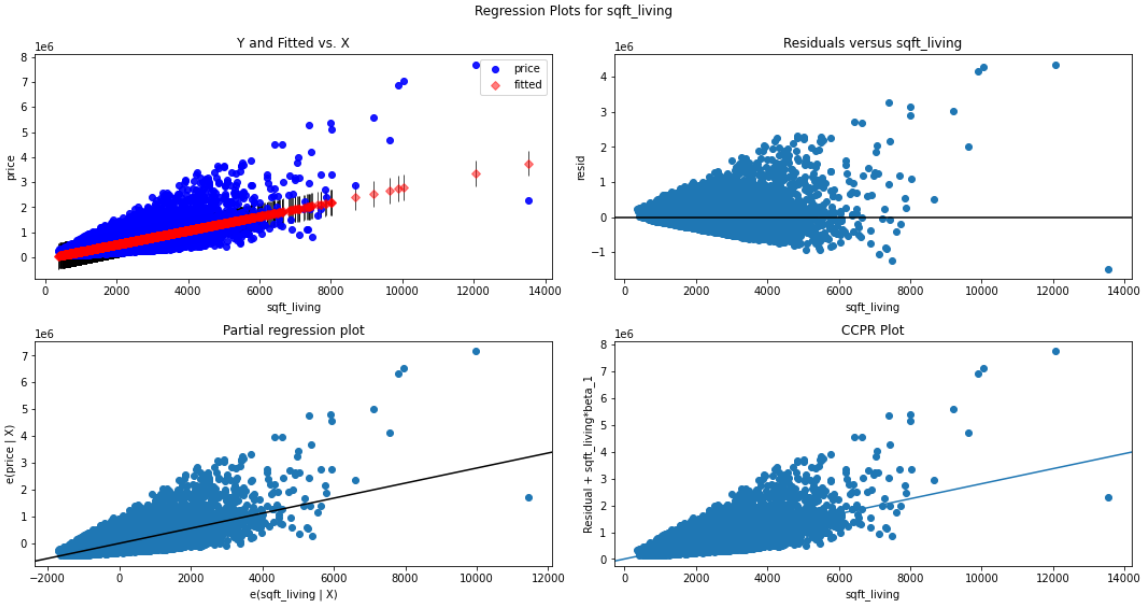
These p scedasticity and Q-Q plot. Using categorical variable nuous. However we do know the independent variable has an effect on the price of a house based on the heat map and the P-value in the baseline model. Therefore, all categorical data will need to have dummy variables created in order to observe its relationship with price

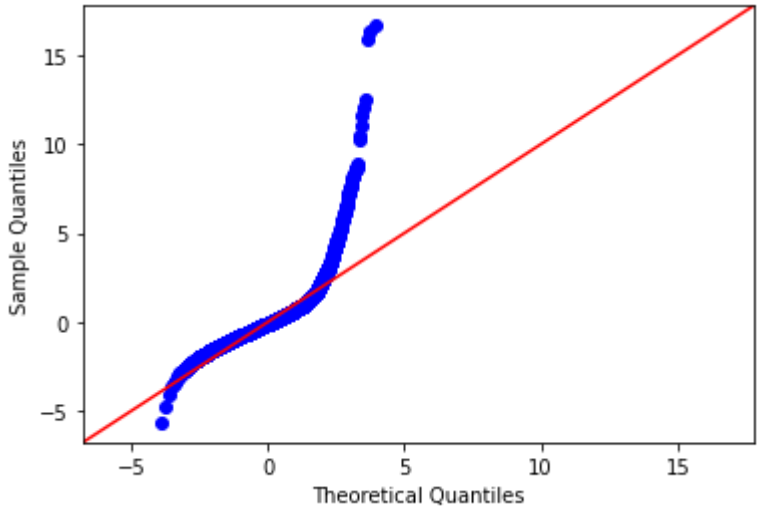
In [27]:

```
# Checking the normality and homoscedasticity assumptions on continuous variables

data=df
f = 'price~sqft_living'
model = smf.ols(formula=f, data=data).fit()
print ('R-Squared:',model.rsquared)
print (model.params)

X_new = pd.DataFrame({'sqft_living': [data.sqft_living.min(), data.sqft_living.max()]})
preds = model.predict(X_new)

fig, ax = plt.subplots()
data.plot(kind='scatter', x='sqft_living', y='price', ax=ax)
ax.plot(X_new['sqft_living'], preds, c='red', linewidth=2)
plt.show()

fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_living", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True)

plt.savefig("images/price_sqft_living_norm_homo_it1", bbox_inches='tight')

plt.show()
```

```
R-Squared: 0.49268789904035093
Intercept     -43988.892194
sqft_living      280.863014
dtype: float64
```



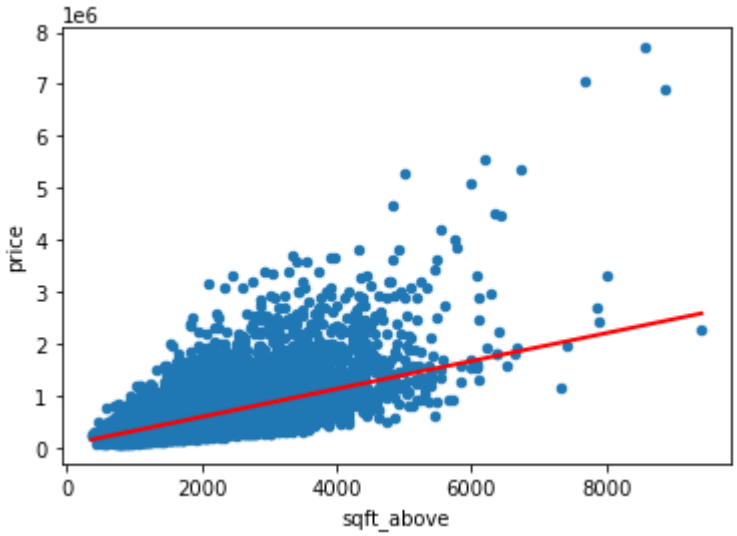

Regression Plots for sqft_living
Y and Fitted vs. X
price
fitted
sqft_living
Residuals versus sqft_living
resid
Partial regression plot
e(price | X)
e(sqft_living | X)
CCPR Plot
Residual + sqft_living*beta_1

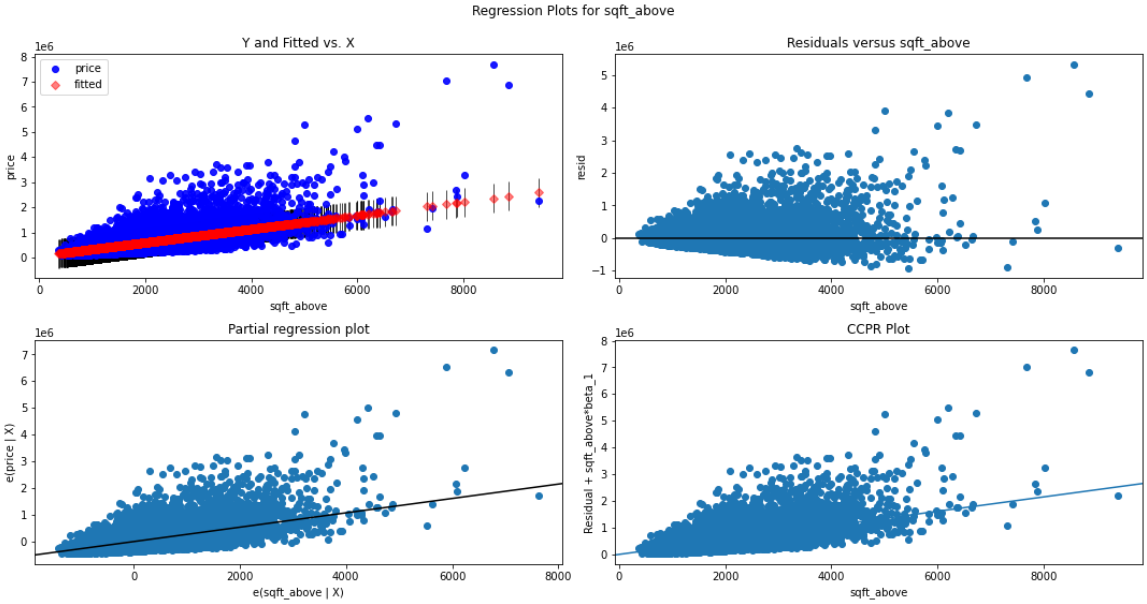

Sample Quantiles
Theoretical Quantiles

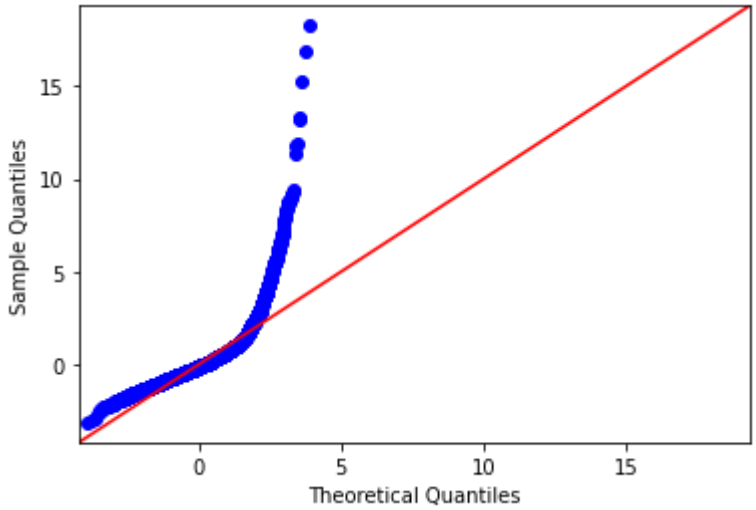
In [28]:

```
data=df
f = 'price~sqft_above'
model = smf.ols(formula=f, data=data).fit()
print ('R-Squared:',model.rsquared)
print (model.params)

X_new = pd.DataFrame({'sqft_above': [data.sqft_above.min(), data.sqft_above.max()]})
preds = model.predict(X_new)

fig, ax = plt.subplots()
data.plot(kind='scatter', x='sqft_above', y='price', ax=ax)
ax.plot(X_new['sqft_above'], preds, c='red', linewidth=2)
plt.show()

fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_above", fig=fig)
plt.show()

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

```
R-Squared: 0.36647034726583816
Intercept     59757.111006
sqft_above      268.668406
dtype: float64
```



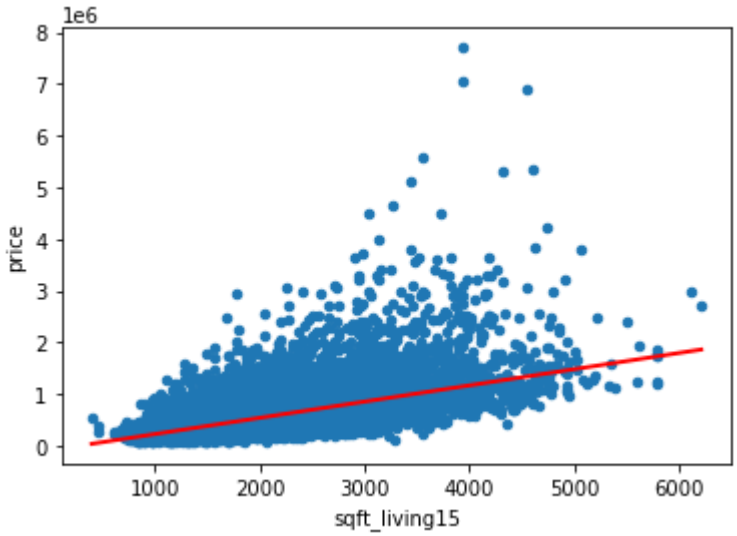

Regression Plots for sqft_above
Y and Fitted vs. X
price
fitted
1e6
price
sqft_above
Residuals versus sqft_above
resid
sqft_above
Partial regression plot
e(price | X)
e(sqft_above | X)
CCPR Plot
Residual + sqft_above*beta_1
sqft_above

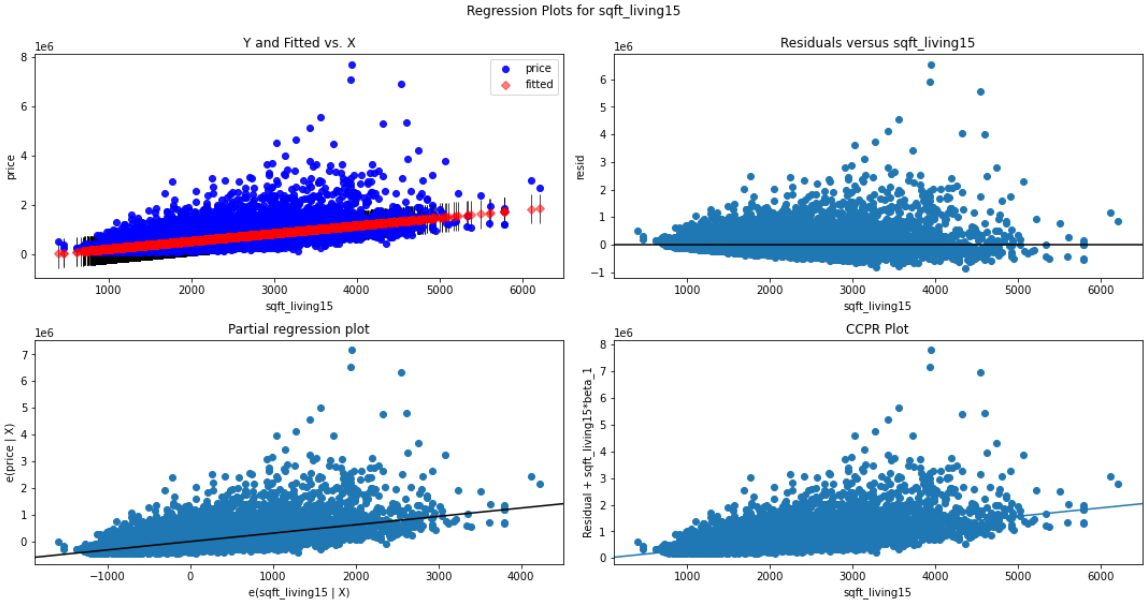

Sample Quantiles
Theoretical Quantiles

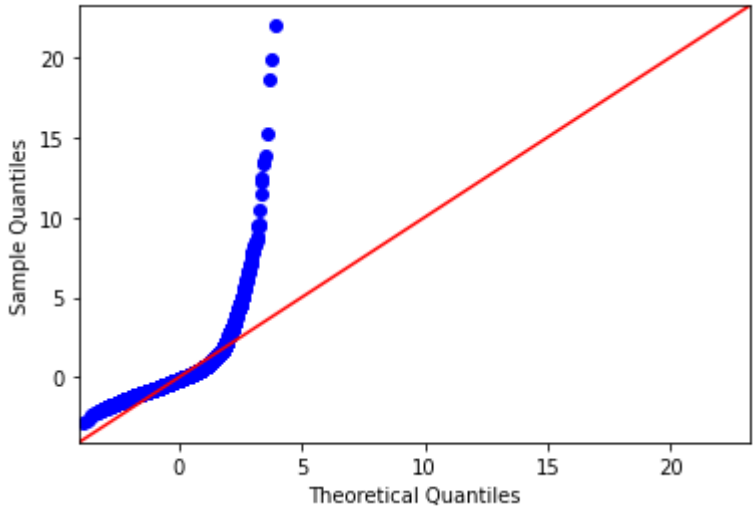
In [29]:

```
data=df
f = 'price~sqft_living15'
model = smf.ols(formula=f, data=data).fit()
print ('R-Squared:',model.rsquared)
print (model.params)

X_new = pd.DataFrame({'sqft_living15': [data.sqft_living15.min(), data.sqft_living15.max(
preds = model.predict(X_new)

fig, ax = plt.subplots()
data.plot(kind='scatter', x='sqft_living15', y='price', ax=ax)
ax.plot(X_new['sqft_living15'], preds, c='red', linewidth=2)
plt.show()

fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_living15", fig=fig)
plt.show()

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

```
R-Squared: 0.3425072641720194
Intercept       -83028.487369
sqft_living15      313.761545
dtype: float64
```



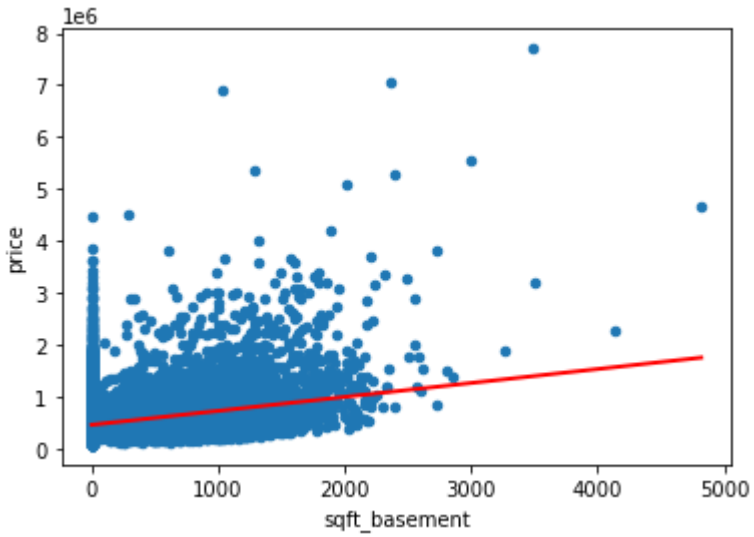

Regression Plots for sqft_living15
Y and Fitted vs. X
price
fitted
sqft_living15
Residuals versus sqft_living15
resid
sqft_living15
Partial regression plot
e(price | X)
e(sqft_living15 | X)
CCPR Plot
Residual + sqft_living15*beta_1
sqft_living15

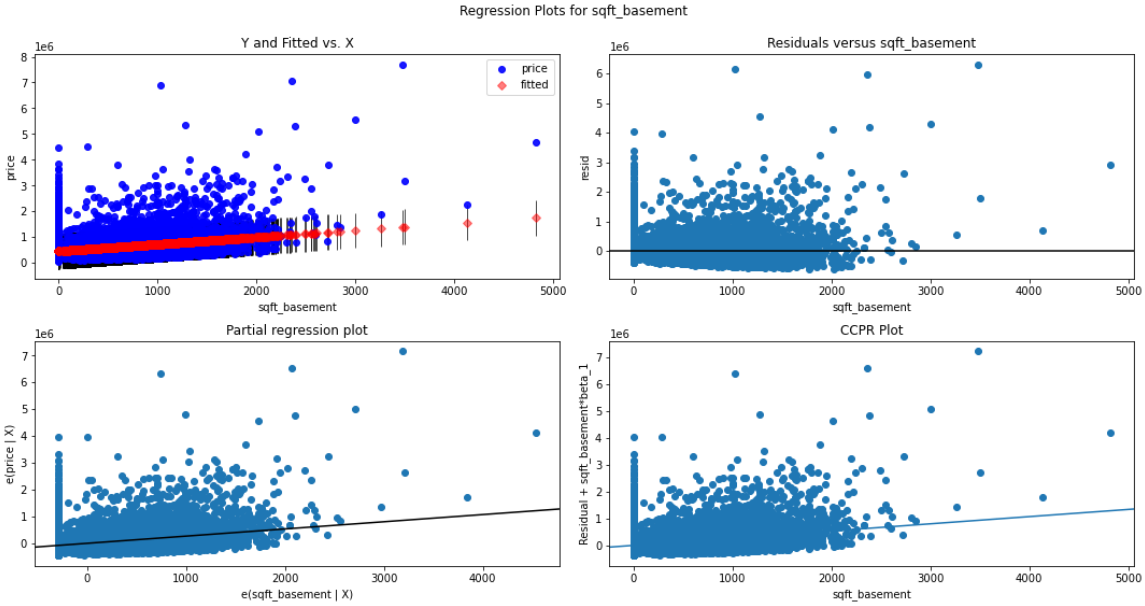

Sample Quantiles
Theoretical Quantiles

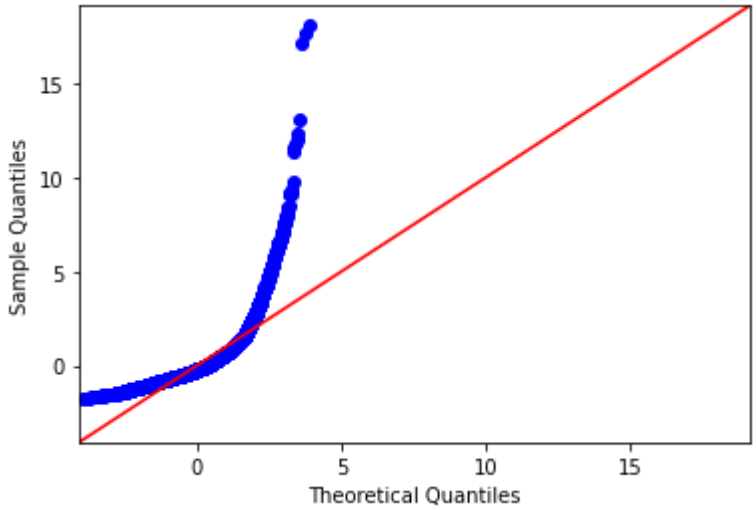
In [30]:

```
data=df
f = 'price~sqft_basement'
model = smf.ols(formula=f, data=data).fit()
print ('R-Squared:',model.rsquared)
print (model.params)

X_new = pd.DataFrame({'sqft_basement': [data.sqft_basement.min(), data.sqft_basement.max(
preds = model.predict(X_new)

fig, ax = plt.subplots()
data.plot(kind='scatter', x='sqft_basement', y='price', ax=ax)
ax.plot(X_new['sqft_basement'], preds, c='red', linewidth=2)
plt.show()

fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_basement", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True)

plt.savefig("images/price_sqft_basement_norm_homo_it1", bbox_inches='tight')

plt.show()
```

```
R-Squared: 0.10311007135689387
Intercept        463664.130330
sqft_basement       268.211396
dtype: float64
```



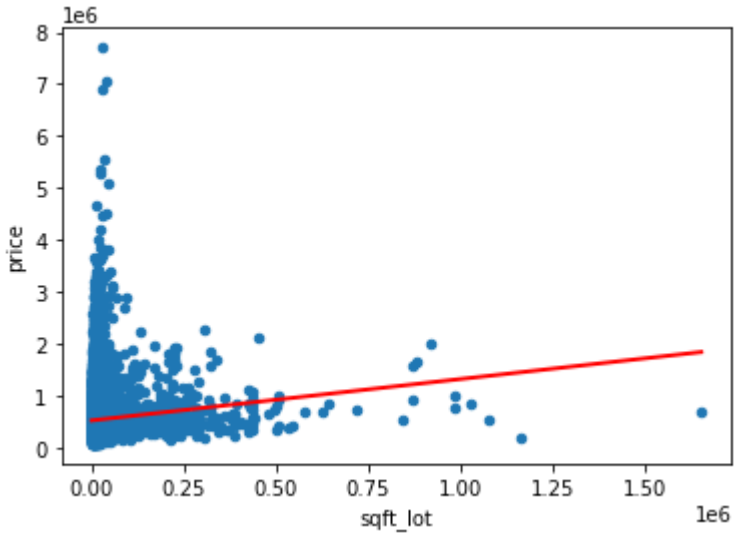

Regression Plots for sqft_basement
Y and Fitted vs. X
price
fitted
1e6
price
sqft_basement
Residuals versus sqft_basement
resid
sqft_basement
Partial regression plot
e(price | X)
e(sqft_basement | X)
CCPR Plot
Residual + sqft_basement*beta_1
sqft_basement

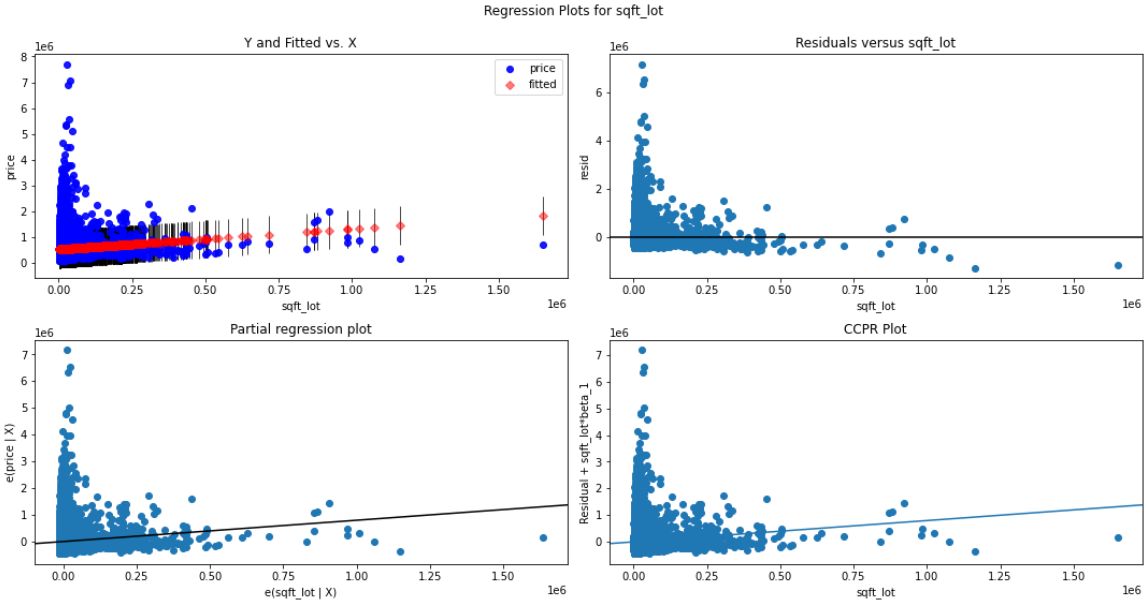

Sample Quantiles
Theoretical Quantiles

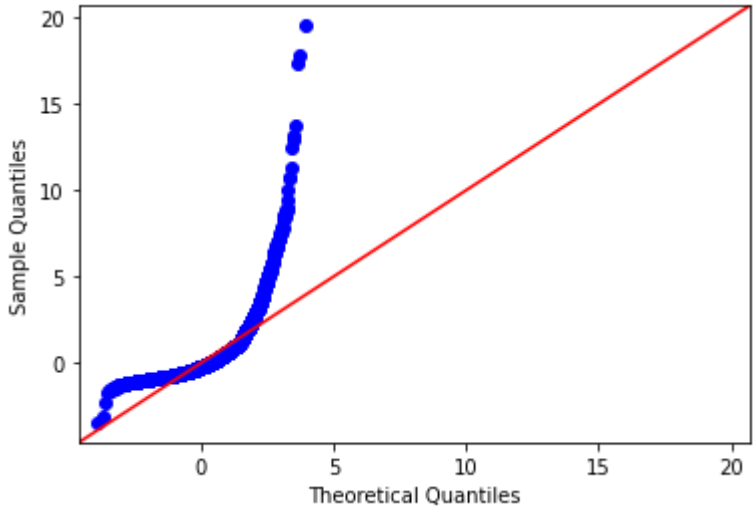
In [31]:

```
data=df
f = 'price~sqft_lot'
model = smf.ols(formula=f, data=data).fit()
print ('R-Squared:',model.rsquared)
print (model.params)

X_new = pd.DataFrame({'sqft_lot': [data.sqft_lot.min(), data.sqft_lot.max()]})
preds = model.predict(X_new)

fig, ax = plt.subplots()
data.plot(kind='scatter', x='sqft_lot', y='price', ax=ax)
ax.plot(X_new['sqft_lot'], preds, c='red', linewidth=2)
plt.show()

fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_lot", fig=fig)
plt.show()

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

```
R-Squared: 0.008077735099594197
Intercept    528258.046513
sqft_lot          0.797285
dtype: float64
```



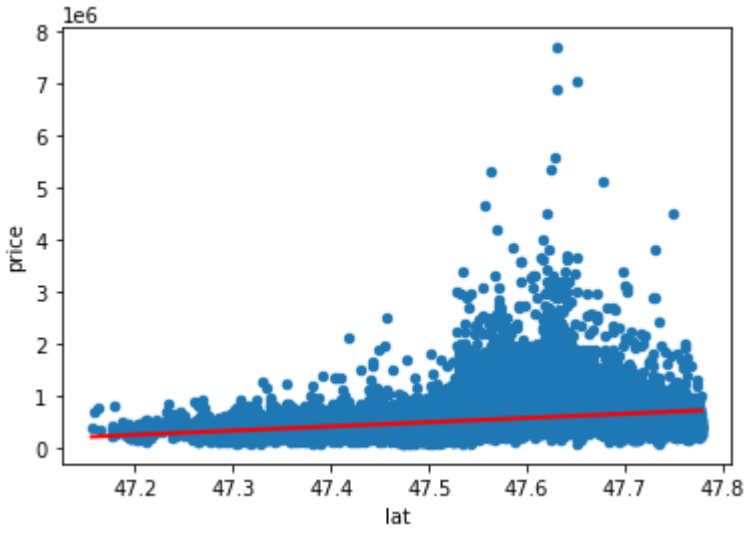

Regression Plots for sqft_lot
Y and Fitted vs. X
price
fitted
sqft_lot
Residuals versus sqft_lot
resid
Partial regression plot
e(price | X)
e(sqft_lot | X)
CCPR Plot
Residual + sqft_lot*beta_1

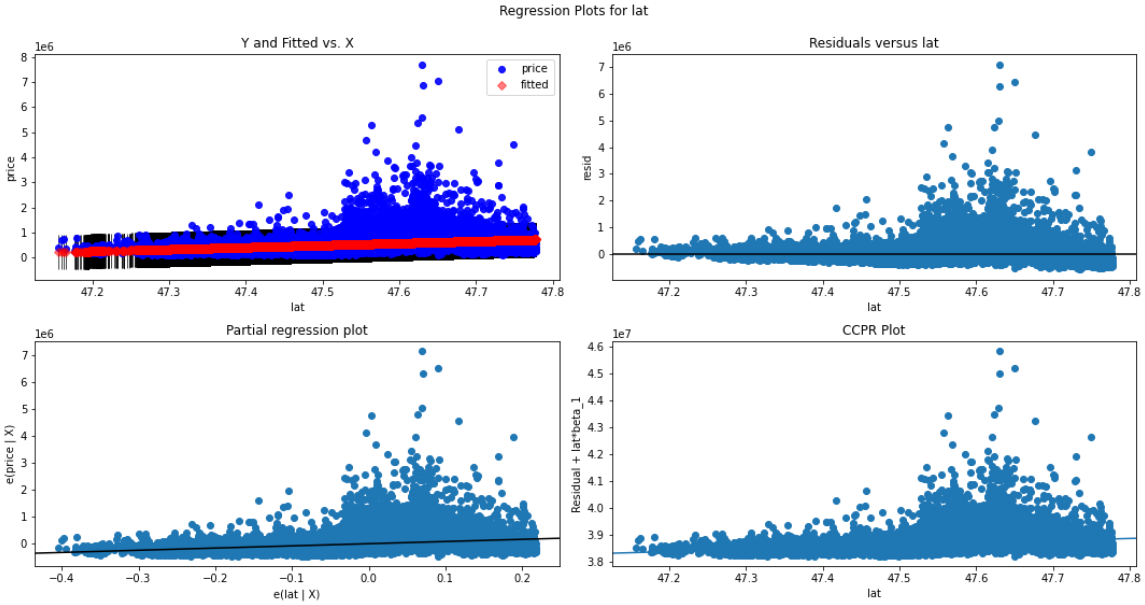

Sample Quantiles
Theoretical Quantiles

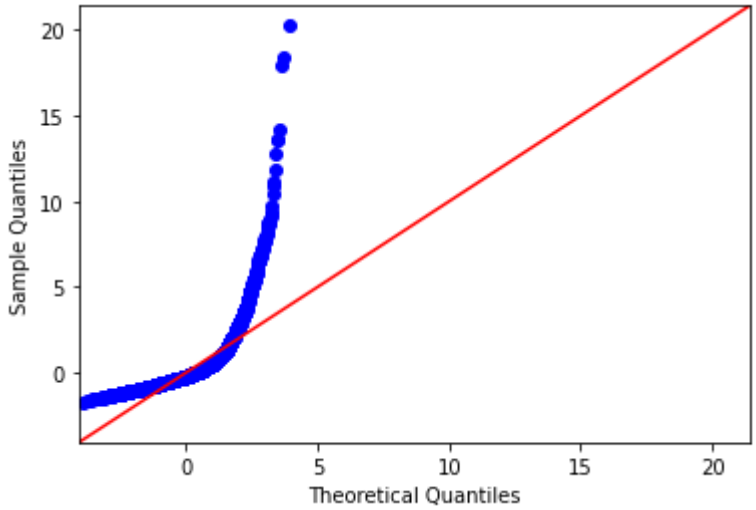
In [32]:

```
data=df
f = 'price~lat'
model = smf.ols(formula=f, data=data).fit()
print ('R-Squared:',model.rsquared)
print (model.params)

X_new = pd.DataFrame({'lat': [data.lat.min(), data.lat.max()]})
preds = model.predict(X_new)

fig, ax = plt.subplots()
data.plot(kind='scatter', x='lat', y='price', ax=ax)
ax.plot(X_new['lat'], preds, c='red', linewidth=2)
plt.show()

fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "lat", fig=fig)
plt.show()

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

```
R-Squared: 0.09406017561857005
Intercept   -3.813512e+07
lat          8.131905e+05
dtype: float64
```



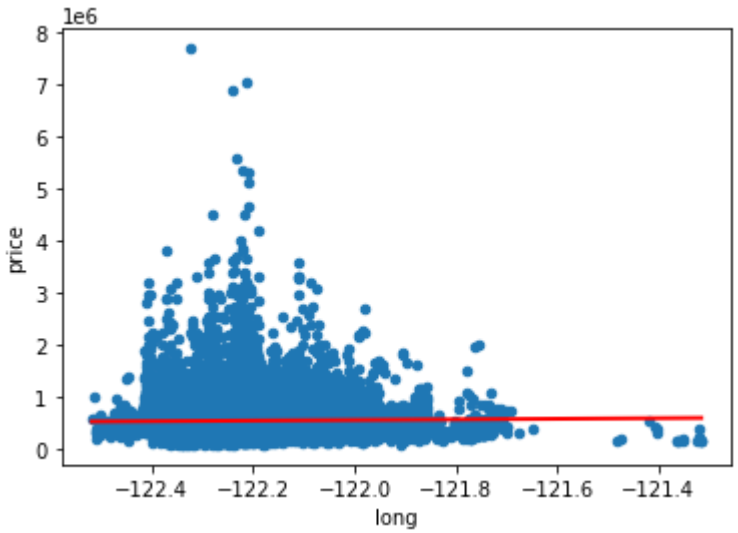

Regression Plots for lat
Y and Fitted vs. X
price
fitted
lat
Residuals versus lat
resid
Partial regression plot
e(price | X)
e(lat | X)
CCPR Plot
Residual + lat*beta_1

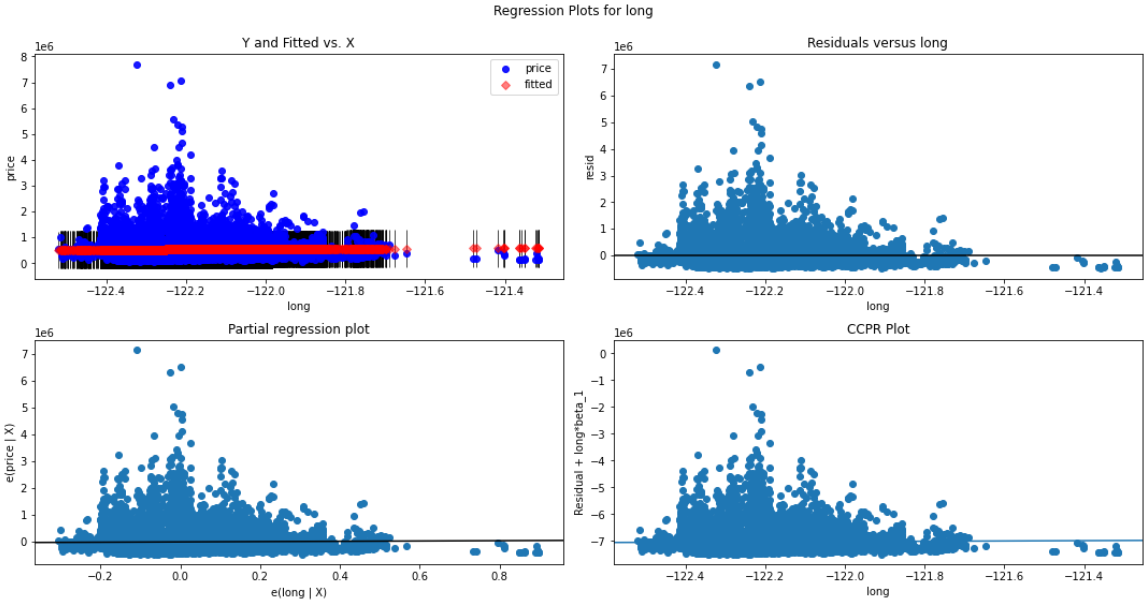

Sample Quantiles
Theoretical Quantiles

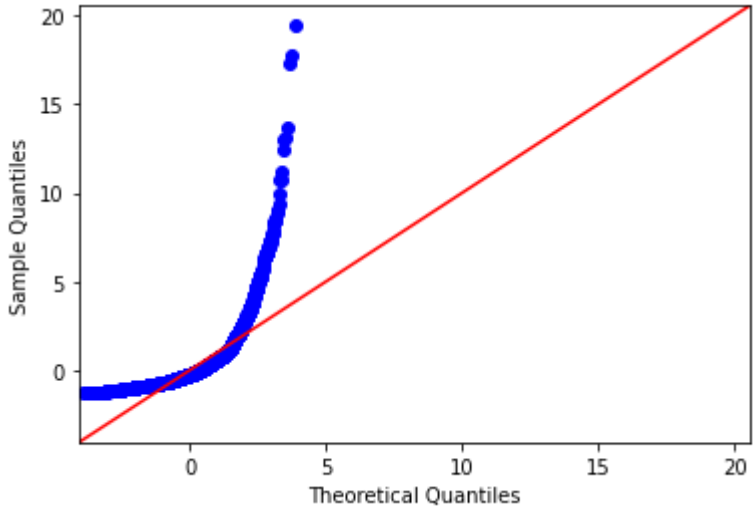
In [33]:

```
data=df
f = 'price~long'
model = smf.ols(formula=f, data=data).fit()
print ('R-Squared:',model.rsquared)
print (model.params)

X_new = pd.DataFrame({'long': [data.long.min(), data.long.max()]})
preds = model.predict(X_new)

fig, ax = plt.subplots()
data.plot(kind='scatter', x='long', y='price', ax=ax)
ax.plot(X_new['long'], preds, c='red', linewidth=2)
plt.show()

fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "long", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True)

plt.savefig("images/price_sqft_long_norm_homo_it1", bbox_inches='tight')

plt.show()
```

```
R-Squared: 0.00048559944782555764
Intercept    7.570935e+06
long         5.752728e+04
dtype: float64
```



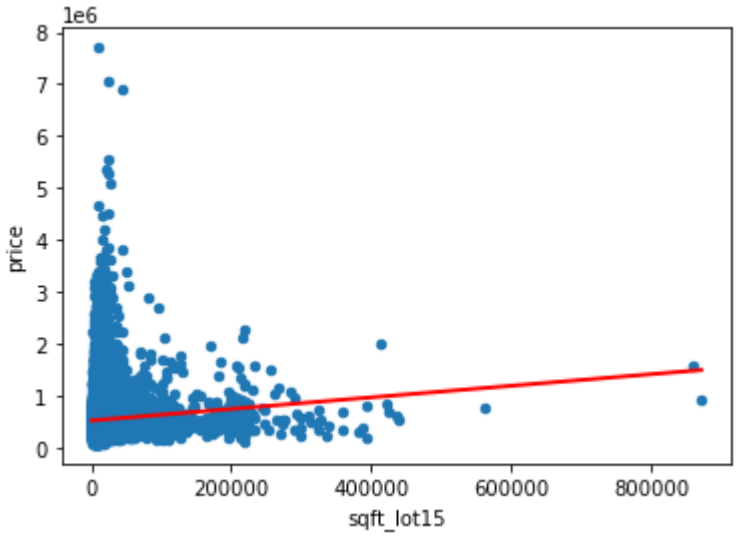

Regression Plots for long
Y and Fitted vs. X
price
fitted
1e6
price
long
Residuals versus long
resid
long
Partial regression plot
e(price | X)
e(long | X)
CCPR Plot
Residual + long*beta_1
long

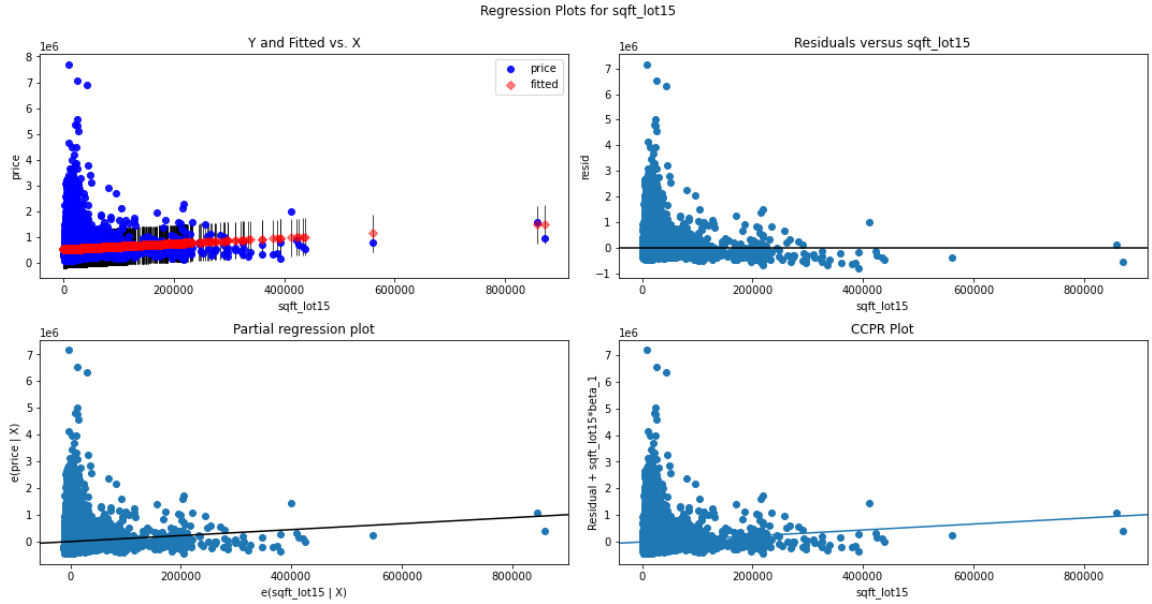

Sample Quantiles
Theoretical Quantiles

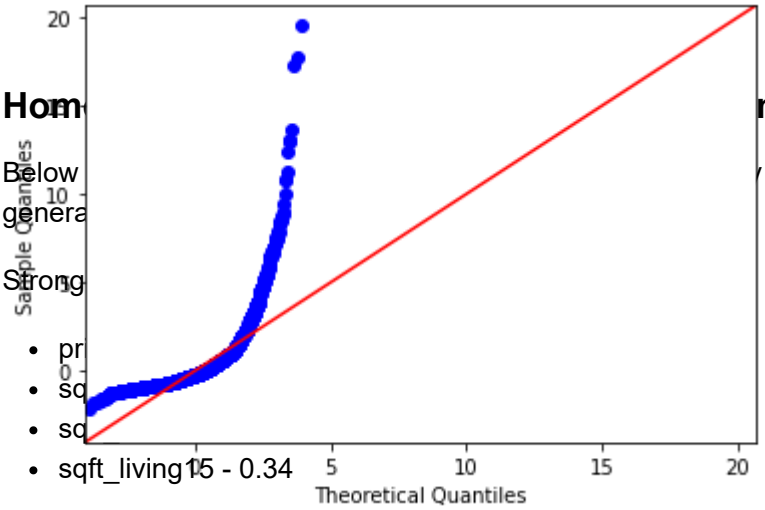
In [34]:

```
data=df
f = 'price~sqft_lot15'
model = smf.ols(formula=f, data=data).fit()
print ('R-Squared:',model.rsquared)
print (model.params)

X_new = pd.DataFrame({'sqft_lot15': [data.sqft_lot15.min(), data.sqft_lot15.max()]})
preds = model.predict(X_new)

fig, ax = plt.subplots()
data.plot(kind='scatter', x='sqft_lot15', y='price', ax=ax)
ax.plot(X_new['sqft_lot15'], preds, c='red', linewidth=2)
plt.show()

fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_lot15", fig=fig)
plt.show()

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

```
R-Squared: 0.00686328262138447
Intercept     526060.054695
sqft_lot15         1.115865
dtype: float64
```







Hom n Comments

Below y assumption with the R-squared value genera

Strong

- pr
- sq
- sq
- sqft_living15 - 0.34

Possible strong linear relationship if there is a basement present

- sqft_basement - 0.10

Weak linear relationship

- sqft_lot - 0.008
- lat - 0.09
- long - 0.0004
- sqft_lot15 - 0.006

It is clear what was observed in the linearity assumption matches with the prediction line plot. It would be safe to say that the weak linear relationship variables can be disregarded except for sqft_basement. The low R-squared value might be affected by the high volume of houses without a basement. However if the house has behaviour, the plot behaves similarly to the strong linear relationship variables. It is worth investigating the sqft_basement with the data that includes only basement houses.

Therefore, below are the variables to be considered for verifying the homoscedasticity and normality assumption

- sqft_living
- sqft_above
- sqft_living15

All variables violate the homoscedasticity assumption. For the normality assumption, all variables can be rejected based on the Q-Q plot. However I have seen previously that these variables are normally distributed. Therefore transformations are required to help normalise the distribution and pass the homoscedasticity and normality assumption

In [35]:

```
# Exploring sqft_basement without houses that do not have a basement

df_sqft_basement = df.iloc[:, [0,11]]
df_sqft_basement.head()
```

Out[35]:

	price	sqft_basement
0	221900.0	0.0
1	538000.0	400.0
2	180000.0	0.0
3	604000.0	910.0
4	510000.0	0.0

In [36]:

```
df_sqft_basement.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          21597 non-null  float64
 1   sqft_basement  21597 non-null  float64
dtypes: float64(2)
memory usage: 337.6 KB
```

In [37]:

```
df_sqft_basement = df_sqft_basement[df_sqft_basement['sqft_basement'] != 0.0]
df_sqft_basement.head()
```

Out[37]:

	price	sqft_basement
1	538000.0	400.0
3	604000.0	910.0
5	1230000.0	1530.0
8	229500.0	730.0
10	662500.0	1700.0

In [38]:

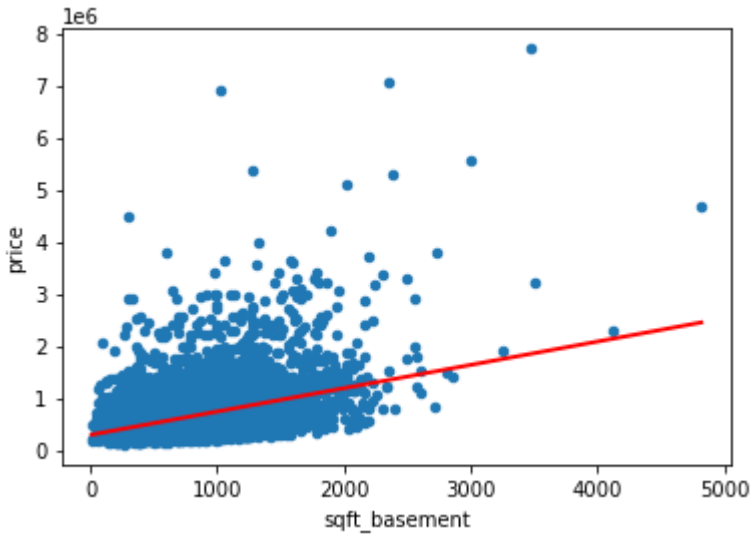
```
df_sqft_basement.info()
```

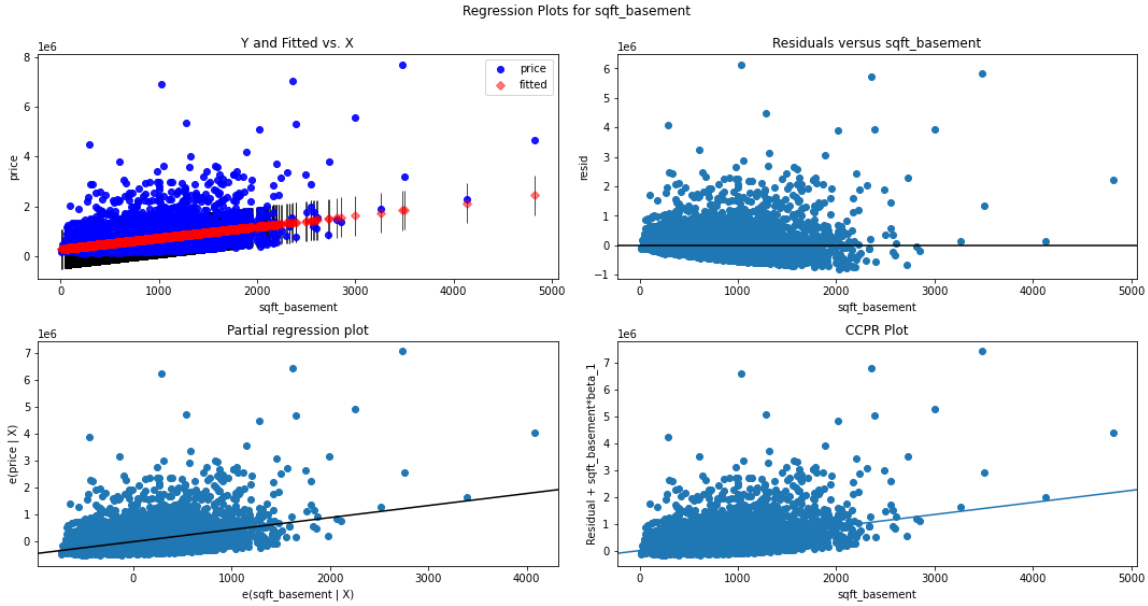
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8317 entries, 1 to 21591
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          8317 non-null   float64
 1   sqft_basement  8317 non-null   float64
dtypes: float64(2)
memory usage: 194.9 KB
```

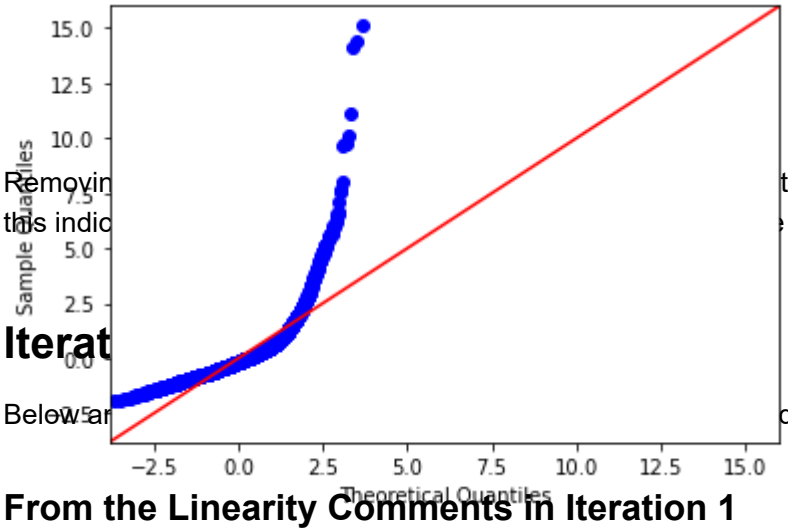
In [39]:

```
data=df_sqft_basement
f = 'price~sqft_basement'
model = smf.ols(formula=f, data=data).fit()
print ('R-Squared:',model.rsquared)
print (model.params)

X_new = pd.DataFrame({'sqft_basement': [data.sqft_basement.min(), data.sqft_basement.max(
preds = model.predict(X_new)

fig, ax = plt.subplots()
data.plot(kind='scatter', x='sqft_basement', y='price', ax=ax)
ax.plot(X_new['sqft_basement'], preds, c='red', linewidth=2)
plt.show()

fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_basement", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True)

plt.savefig("images/price_sqft_basement_norm_homo_it1_no_zero", bbox_inches='tight')

plt.show()
```

```
R-Squared: 0.16675402649464288
Intercept        290950.690144
sqft_basement       447.611997
dtype: float64
```







Removin the R-squared from 0.10 to 0.16. However this indic disregarded.

Iterat

Below ar ddress them in iteration 2

From the Linearity Comments in Iteration 1

These independent variables are categorical variables with their correlation score from the heat map.

- grade - 0.67
- bathrooms - 0.53
- view - 0.39
- bedrooms - 0.31
- floors - 0.26
- waterfront - 0.26
- yr_renovated - 0.12
- yr_built - 0.054
- condition - 0.036
- zipcode - -0.053

Based on the correlation score, I will not use the bottom 4 independent variables in this model. They do not have a strong enough correlation with the target variable. Dummy variables will be created so the remaining categorical variables can be used in the regression model.

From Linearity, Homoscedasticity and Normality Comments in Iteration 1

It has been shown through verifying the assumptions that these variables do not have a strong enough linear relationship. I will consider dropping these variables especially during the multicollinearity checking phase.

- sqft_lot
- lat
- long
- sqft_lot15
- sqft_basement

The variables below have shown to have a strong relationship with the target variable however violate the linear assumptions. I will use log transformations to improve performance.

- sqft_living
- sqft_above
- sqft_living15

Before the transformations are applied, I will address the outliers observed in the normal distributions and scatter plots.

Removing Outliers

In [40]:

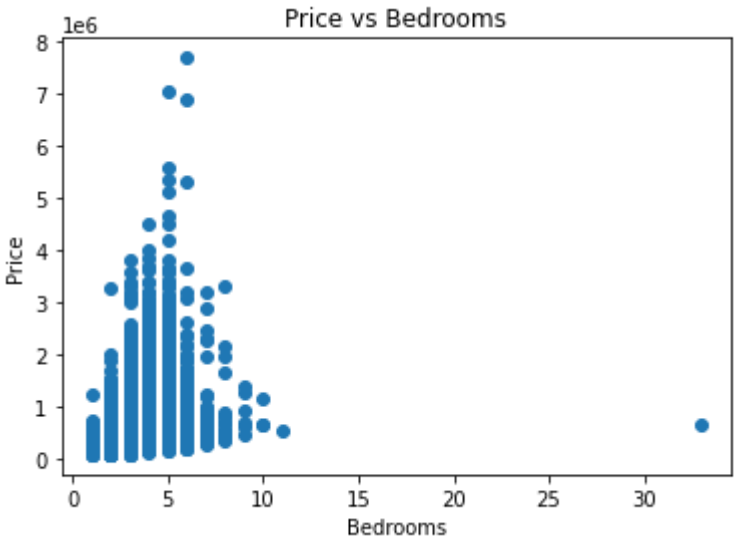
```
df.describe()
```

Out[40]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
count	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	21597.000000
mean	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	1.494096
std	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	0.539683
min	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.000000
25%	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1.000000
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	2.000000
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000

In [41]:

```
plt.scatter(df['bedrooms'], df['price'])
plt.xlabel('Bedrooms')
plt.ylabel('Price')
plt.title('Price vs Bedrooms')
plt.show()
```



In [42]:

```
# Scatter plot for bedrooms indicates an outlier to the far right

df['bedrooms'].unique()
```

Out[42]:

```
array([ 3,  2,  4,  5,  1,  6,  7,  8,  9, 11, 10, 33], dtype=int64)
```

In [43]:

```
df[df['bedrooms'] == 33]
```

Out[43]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditi
15856	640000.0	33	1.75	1620	6000	1.0	0.0	0.0	

In [44]:

```
# One house has 33 bedrooms and sqft_living of 1620. This is below the mean
# It is unlikely this house has 33 bedrooms, possibly a recording error

df['bedrooms'] = df['bedrooms'].replace(33, 3)
```

In [45]:

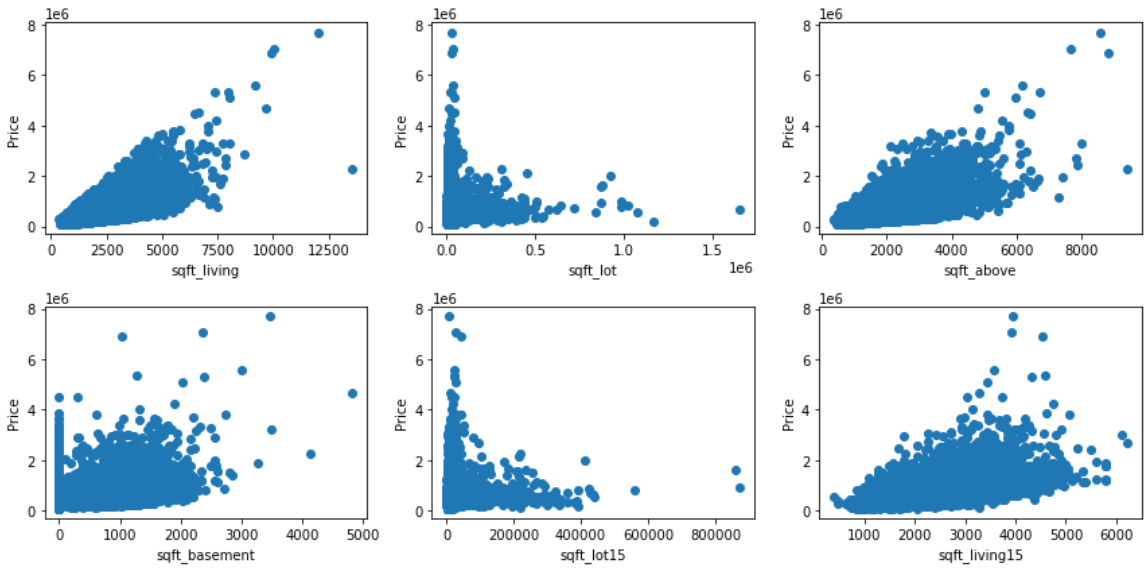
```
df['bedrooms'].unique()
```

Out[45]:

```
array([ 3,  2,  4,  5,  1,  6,  7,  8,  9, 11, 10], dtype=int64)
```

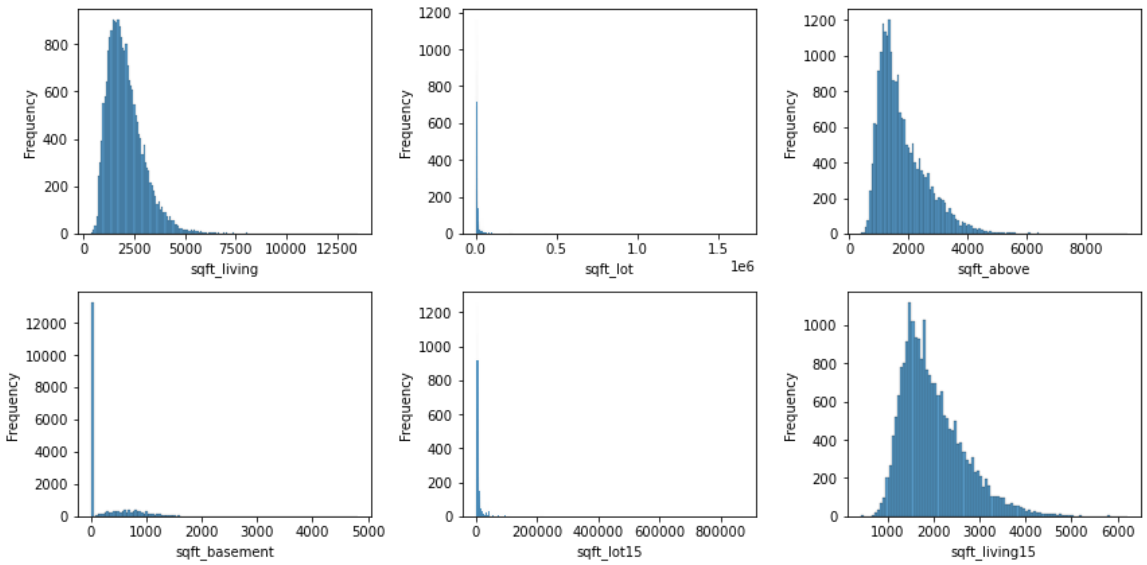
In [46]:

```
# Several sqft variables have outliers to the right. Could be related

fig, axs = plt.subplots(2, 3, figsize=(12, 6))

axs[0, 0].scatter(df['sqft_living'], df['price'])
axs[0, 0].set_xlabel('sqft_living')
axs[0, 0].set_ylabel('Price')

axs[0, 1].scatter(df['sqft_lot'], df['price'])
axs[0, 1].set_xlabel('sqft_lot')
axs[0, 1].set_ylabel('Price')

axs[0, 2].scatter(df['sqft_above'], df['price'])
axs[0, 2].set_xlabel('sqft_above')
axs[0, 2].set_ylabel('Price')

axs[1, 0].scatter(df['sqft_basement'], df['price'])
axs[1, 0].set_xlabel('sqft_basement')
axs[1, 0].set_ylabel('Price')

axs[1, 1].scatter(df['sqft_lot15'], df['price'])
axs[1, 1].set_xlabel('sqft_lot15')
axs[1, 1].set_ylabel('Price')

axs[1, 2].scatter(df['sqft_living15'], df['price'])
axs[1, 2].set_xlabel('sqft_living15')
axs[1, 2].set_ylabel('Price')

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



In [47]:

```
fig, axs = plt.subplots(2, 3, figsize=(12, 6))

sns.histplot(data=df, x='sqft_living', ax=axs[0, 0])
axs[0, 0].set_xlabel('sqft_living')
axs[0, 0].set_ylabel('Frequency')

sns.histplot(data=df, x='sqft_lot', ax=axs[0, 1])
axs[0, 1].set_xlabel('sqft_lot')
axs[0, 1].set_ylabel('Frequency')

sns.histplot(data=df, x='sqft_above', ax=axs[0, 2])
axs[0, 2].set_xlabel('sqft_above')
axs[0, 2].set_ylabel('Frequency')

sns.histplot(data=df, x='sqft_basement', ax=axs[1, 0])
axs[1, 0].set_xlabel('sqft_basement')
axs[1, 0].set_ylabel('Frequency')

sns.histplot(data=df, x='sqft_lot15', ax=axs[1, 1])
axs[1, 1].set_xlabel('sqft_lot15')
axs[1, 1].set_ylabel('Frequency')

sns.histplot(data=df, x='sqft_living15', ax=axs[1, 2])
axs[1, 2].set_xlabel('sqft_living15')
axs[1, 2].set_ylabel('Frequency')

plt.tight_layout()

plt.savefig("images/cont_var_hist_it2", bbox_inches='tight')

plt.show()
```



In [48]:

```
# Reduce outliers by reducing data size to 3 standard deviations

filter_cols = ['sqft_living', 'sqft_lot', 'sqft_above', 'sqft_basement', 'sqft_lot15', 's

df1 = df[~df[filter_cols].apply(lambda x: np.abs(x - x.mean()) > 3 * x.std()).any(axis=1)
```

In [49]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   price          21597 non-null  float64
 1   bedrooms       21597 non-null  int64
 2   bathrooms      21597 non-null  float64
 3   sqft_living    21597 non-null  int64
 4   sqft_lot       21597 non-null  int64
 5   floors         21597 non-null  float64
 6   waterfront     21597 non-null  float64
 7   view           21597 non-null  float64
 8   condition      21597 non-null  int64
 9   grade          21597 non-null  int64
 10  sqft_above     21597 non-null  int64
 11  sqft_basement  21597 non-null  float64
 12  yr_built       21597 non-null  int64
 13  yr_renovated   21597 non-null  float64
```

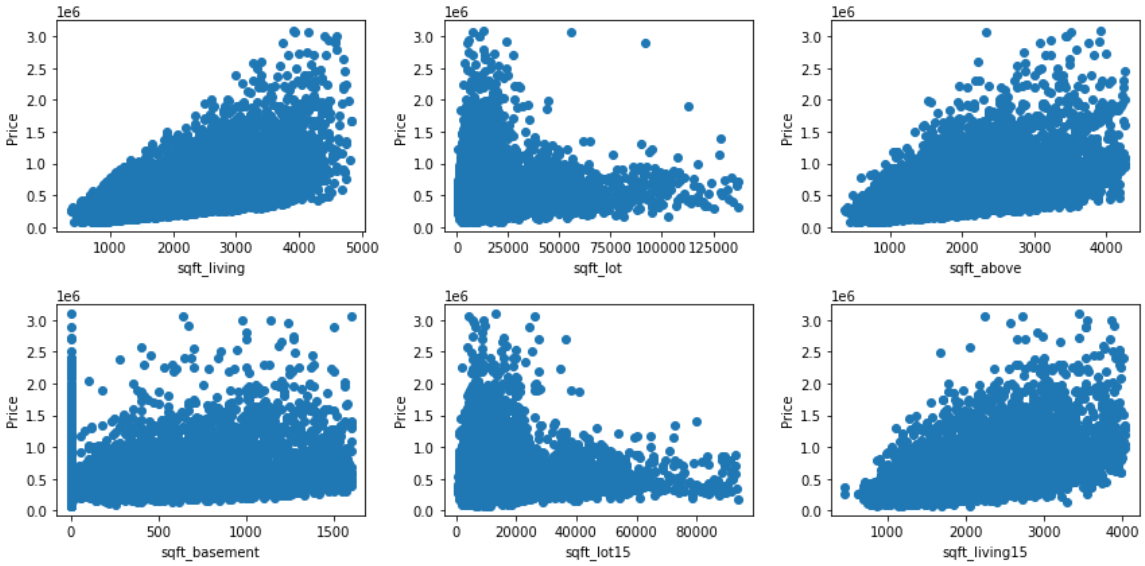
In [50]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20538 entries, 0 to 21596
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   price          20538 non-null  float64
 1   bedrooms       20538 non-null  int64
 2   bathrooms      20538 non-null  float64
 3   sqft_living    20538 non-null  int64
 4   sqft_lot       20538 non-null  int64
 5   floors         20538 non-null  float64
 6   waterfront     20538 non-null  float64
 7   view           20538 non-null  float64
 8   condition      20538 non-null  int64
 9   grade          20538 non-null  int64
 10  sqft_above     20538 non-null  int64
 11  sqft_basement  20538 non-null  float64
 12  yr_built       20538 non-null  int64
 13  yr_renovated   20538 non-null  float64
```

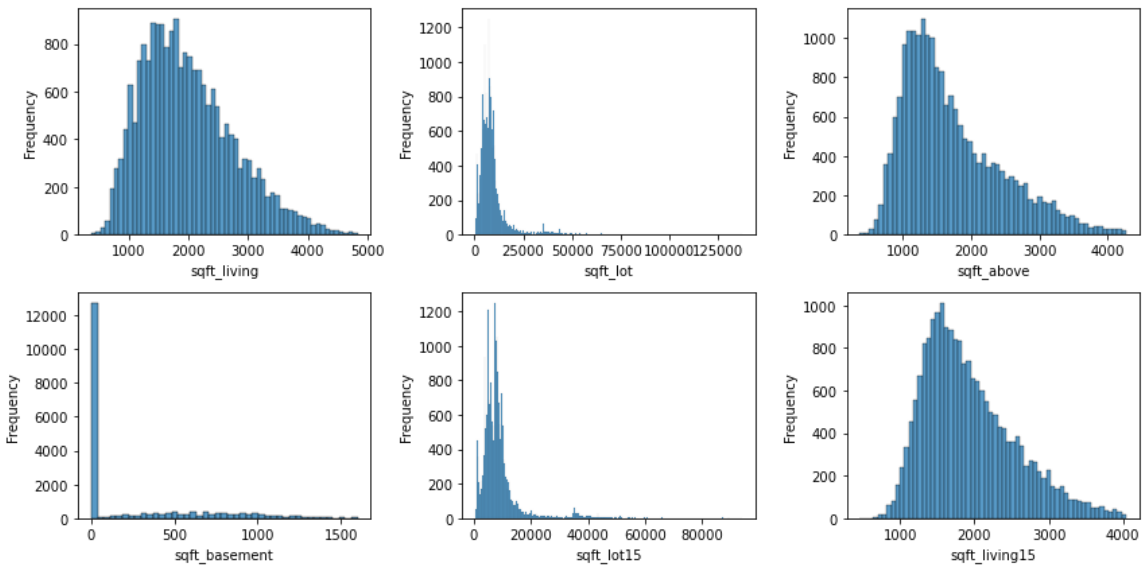
In [51]:

```
fig, axs = plt.subplots(2, 3, figsize=(12, 6))

axs[0, 0].scatter(df1['sqft_living'], df1['price'])
axs[0, 0].set_xlabel('sqft_living')
axs[0, 0].set_ylabel('Price')

axs[0, 1].scatter(df1['sqft_lot'], df1['price'])
axs[0, 1].set_xlabel('sqft_lot')
axs[0, 1].set_ylabel('Price')

axs[0, 2].scatter(df1['sqft_above'], df1['price'])
axs[0, 2].set_xlabel('sqft_above')
axs[0, 2].set_ylabel('Price')

axs[1, 0].scatter(df1['sqft_basement'], df1['price'])
axs[1, 0].set_xlabel('sqft_basement')
axs[1, 0].set_ylabel('Price')

axs[1, 1].scatter(df1['sqft_lot15'], df1['price'])
axs[1, 1].set_xlabel('sqft_lot15')
axs[1, 1].set_ylabel('Price')

axs[1, 2].scatter(df1['sqft_living15'], df1['price'])
axs[1, 2].set_xlabel('sqft_living15')
axs[1, 2].set_ylabel('Price')

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



In [52]:

```
fig, axs = plt.subplots(2, 3, figsize=(12, 6))

sns.histplot(data=df1, x='sqft_living', ax=axs[0, 0])
axs[0, 0].set_xlabel('sqft_living')
axs[0, 0].set_ylabel('Frequency')

sns.histplot(data=df1, x='sqft_lot', ax=axs[0, 1])
axs[0, 1].set_xlabel('sqft_lot')
axs[0, 1].set_ylabel('Frequency')

sns.histplot(data=df1, x='sqft_above', ax=axs[0, 2])
axs[0, 2].set_xlabel('sqft_above')
axs[0, 2].set_ylabel('Frequency')

sns.histplot(data=df1, x='sqft_basement', ax=axs[1, 0])
axs[1, 0].set_xlabel('sqft_basement')
axs[1, 0].set_ylabel('Frequency')

sns.histplot(data=df1, x='sqft_lot15', ax=axs[1, 1])
axs[1, 1].set_xlabel('sqft_lot15')
axs[1, 1].set_ylabel('Frequency')

sns.histplot(data=df1, x='sqft_living15', ax=axs[1, 2])
axs[1, 2].set_xlabel('sqft_living15')
axs[1, 2].set_ylabel('Frequency')

plt.tight_layout()

plt.savefig("images/cont_var_hist_std3_it2", bbox_inches='tight')

plt.show()
```



Scatter plots and histograms are now much improved

Multicollinearity

To improve the performance of the model and have accurate co-efficients, highly correlated variables must be removed.

In [53]:

```
df1.columns
```

Out[53]:

```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floor
s',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'lon
g',
       'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

In [54]:

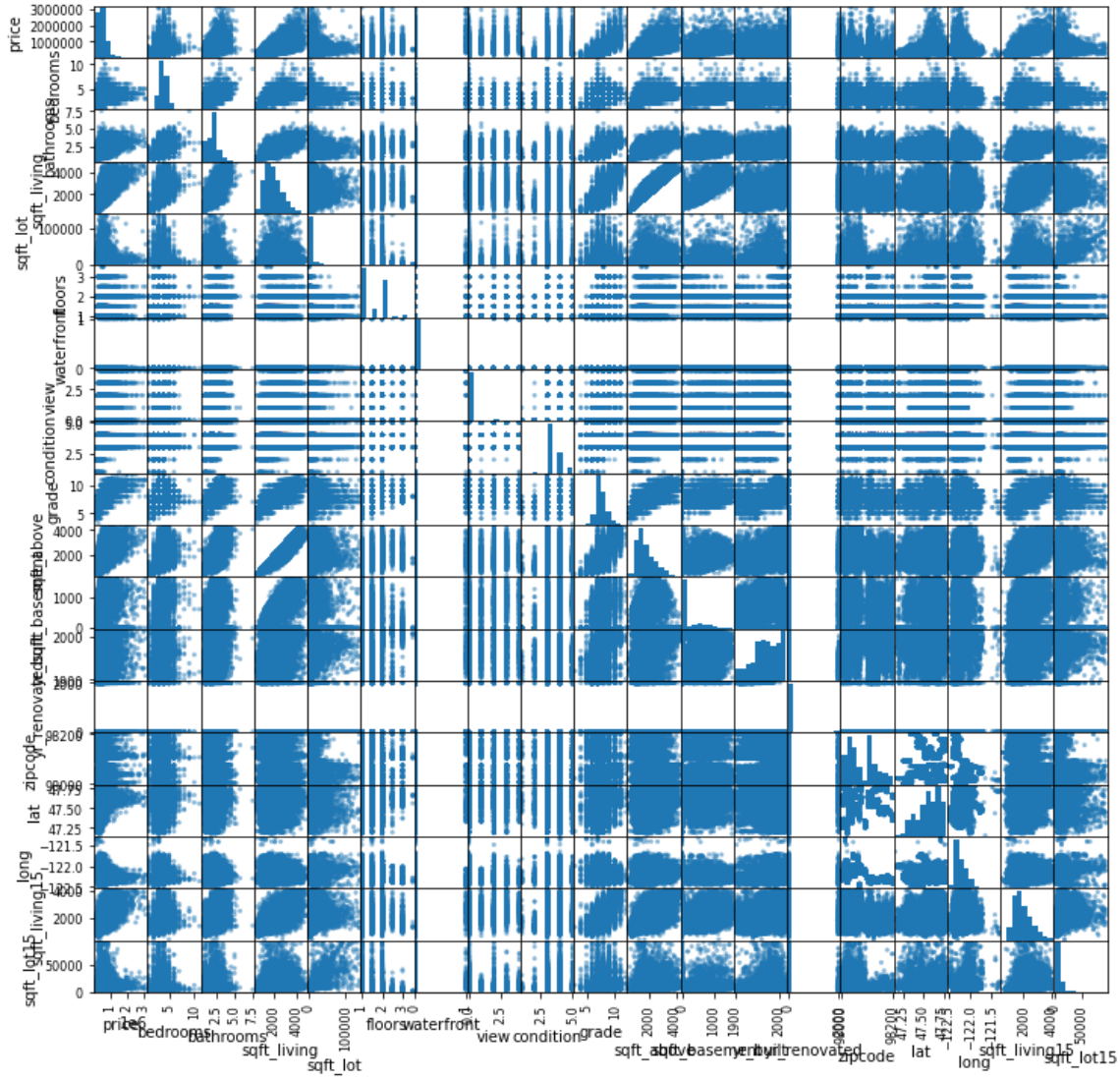
```
numeric_vars = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long',
       'sqft_living15', 'sqft_lot15']

df1_preprocessed = df1.loc[:, numeric_vars]
df1_preprocessed.head()
```

Out[54]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	g
0	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	3	
1	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	3	
2	180000.0	2	1.00	770	10000	1.0	0.0	0.0	3	
3	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	5	
4	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	3	

In [55]:

```
pd.plotting.scatter_matrix(df1_preprocessed, figsize=[12, 12]);
```



At a quick glance, the sqft variables seem to be highly correlated. In terms of real world application, you would expect the sqft_living to be related to sqft_basement. A new home builder would consider that the basement size be dictated by the living area or vice versa.

In [56]:

```
df1_preprocessed.corr()
```

Out[56]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
price	1.000000	0.285593	0.460046	0.635162	0.095391	0.265336	0.199667
bedrooms	0.285593	1.000000	0.505753	0.602363	0.091132	0.173091	-0.019747
bathrooms	0.460046	0.505753	1.000000	0.724945	0.059202	0.513247	0.028278
sqft_living	0.635162	0.602363	0.724945	1.000000	0.217353	0.364046	0.053219
sqft_lot	0.095391	0.091132	0.059202	0.217353	1.000000	-0.097669	0.064416
floors	0.265336	0.173091	0.513247	0.364046	-0.097669	1.000000	0.012401
waterfront	0.199667	-0.019747	0.028278	0.053219	0.064416	0.012401	1.000000
view	0.357803	0.045879	0.130847	0.213072	0.064616	0.016792	0.373713
condition	0.056164	0.024101	-0.128019	-0.057986	0.035406	-0.266185	0.016999
grade	0.639554	0.332976	0.623731	0.722997	0.133028	0.465129	0.050276
sqft_above	0.534189	0.486447	0.645544	0.858325	0.209234	0.540628	0.031949
sqft_basement	0.254437	0.273337	0.224374	0.367534	0.040809	-0.272359	0.044235
yr_built	0.019186	0.154726	0.519911	0.324893	0.027574	0.493352	-0.030192
yr_renovated	0.120737	0.016290	0.044375	0.049171	0.016818	0.005713	0.071672
zipcode	-0.015058	-0.153461	-0.195327	-0.190628	-0.182542	-0.056824	0.036453
lat	0.369515	-0.022727	0.013752	0.046984	-0.065493	0.044926	-0.019626
long	-0.002184	0.138094	0.218157	0.237012	0.272930	0.121787	-0.039130
sqft_living15	0.556598	0.384900	0.530424	0.740538	0.257205	0.274094	0.061009
sqft_lot15	0.087982	0.090121	0.062388	0.216218	0.811946	-0.101125	0.088295

In [57]:

```
abs(df1_preprocessed.corr()) > 0.75
```

Out[57]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	c
price	True	False	False	False	False	False	False	False	
bedrooms	False	True	False	False	False	False	False	False	
bathrooms	False	False	True	False	False	False	False	False	
sqft_living	False	False	False	True	False	False	False	False	
sqft_lot	False	False	False	False	True	False	False	False	
floors	False	False	False	False	False	True	False	False	
waterfront	False	False	False	False	False	False	True	False	
view	False	False	False	False	False	False	False	True	
condition	False	False	False	False	False	False	False	False	
grade	False	False	False	False	False	False	False	False	
sqft_above	False	False	False	True	False	False	False	False	
sqft_basement	False	False	False	False	False	False	False	False	
yr_built	False	False	False	False	False	False	False	False	
yr_renovated	False	False	False	False	False	False	False	False	
zipcode	False	False	False	False	False	False	False	False	
lat	False	False	False	False	False	False	False	False	
long	False	False	False	False	False	False	False	False	
sqft_living15	False	False	False	False	False	False	False	False	
sqft_lot15	False	False	False	False	True	False	False	False	

In [58]:

```
df1_corr_pairs = df1_preprocessed.corr().abs().stack().reset_index().sort_values(0, ascer

df1_corr_pairs['pairs'] = list(zip(df1_corr_pairs.level_0, df1_corr_pairs.level_1))

df1_corr_pairs.set_index(['pairs'], inplace = True)

df1_corr_pairs.drop(columns=['level_1', 'level_0'], inplace = True)

# cc for correlation coefficient
df1_corr_pairs.columns = ['cc']

df1_corr_pairs.drop_duplicates(inplace=True)

df1_corr_pairs[(df1_corr_pairs.cc>.75) & (df1_corr_pairs.cc<1)]
```

Out[58]:

	cc
pairs	
(sqft_living, sqft_above)	0.858325
(sqft_lot15, sqft_lot)	0.811946

Recalling the continuous variables that were found not to have a strong linear relationship with price

- sqft_lot
- lat
- long
- sqft_lot15
- sqft_basement

And the continuous variables with a strong linear relationship

- sqft_living
- sqft_above
- sqft_living15

In order to prevent multicollinearity, I will remove variables sqft_lot15 and sqft_above. Although sqft_above has a strong linear relationship with price, I value sqft_living and sqft_living15 as a better variable for affecting price. Generally speaking, the larger the area for living, the higher the price of a house. With surrounding houses having a larger area for living, generally the house price for the area is also high.

I will also drop the variables that do not have a strong linear relationship with price.

Log Transformations

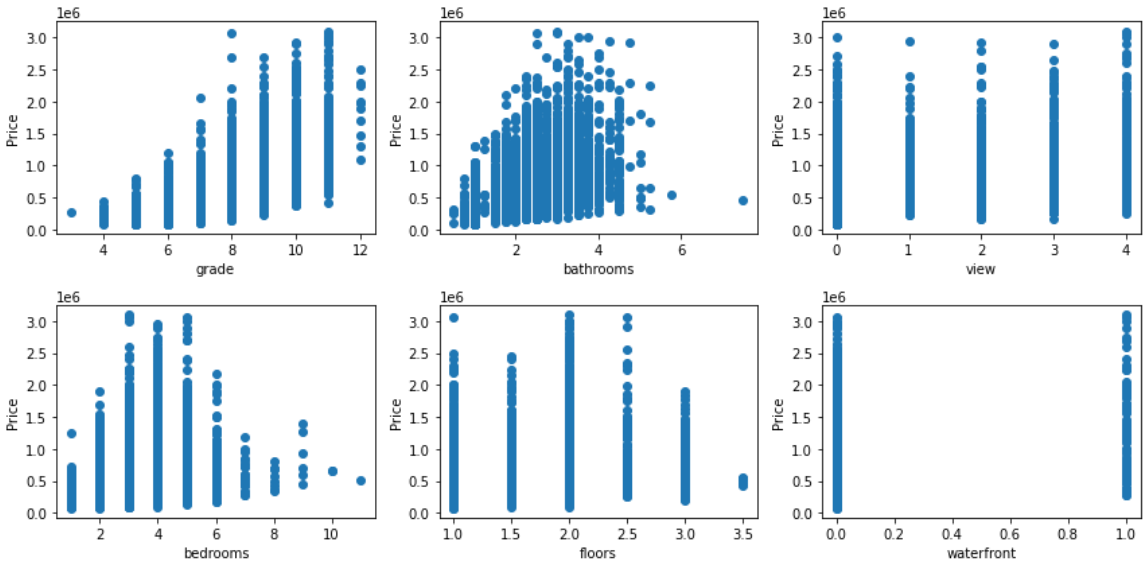
By performing and multicollinearity and using the observations in iteration 1, the continuous variables to be log transformed is finalised

In [59]:

```
continuous = ['sqft_living', 'sqft_living15','price']

# Log transform and normalize
df1_cont = df1[continuous]

# log features
log_names = [f'{column}_log' for column in df1_cont.columns]

df1_log = np.log(df1_cont)
df1_log.columns = log_names

# normalize (subract mean and divide by std)
def normalize(feature):
    return (feature - feature.mean()) / feature.std()

df1_log_norm = df1_log.apply(normalize)
```

In [60]:

```
# Verifying categorical variables

fig, axs = plt.subplots(2, 3, figsize=(12, 6))

axs[0, 0].scatter(df1['grade'], df1['price'])
axs[0, 0].set_xlabel('grade')
axs[0, 0].set_ylabel('Price')

axs[0, 1].scatter(df1['bathrooms'], df1['price'])
axs[0, 1].set_xlabel('bathrooms')
axs[0, 1].set_ylabel('Price')

axs[0, 2].scatter(df1['view'], df1['price'])
axs[0, 2].set_xlabel('view')
axs[0, 2].set_ylabel('Price')

axs[1, 0].scatter(df1['bedrooms'], df1['price'])
axs[1, 0].set_xlabel('bedrooms')
axs[1, 0].set_ylabel('Price')

axs[1, 1].scatter(df1['floors'], df1['price'])
axs[1, 1].set_xlabel('floors')
axs[1, 1].set_ylabel('Price')

axs[1, 2].scatter(df1['waterfront'], df1['price'])
axs[1, 2].set_xlabel('waterfront')
axs[1, 2].set_ylabel('Price')

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



In [61]:

```
categoricals = ['grade', 'bathrooms', 'view', 'bedrooms', 'floors', 'waterfront']

# Convert columns to category data type
for col in categoricals:
    df1[col] = df1[col].astype('category')

# Perform one-hot encoding
df1_cat = pd.get_dummies(df1[categoricals], prefix=categoricals, drop_first=True)
```

```
<ipython-input-61-f7725d793556>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1[col] = df1[col].astype('category')
<ipython-input-61-f7725d793556>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1[col] = df1[col].astype('category')
<ipython-input-61-f7725d793556>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1[col] = df1[col].astype('category')
<ipython-input-61-f7725d793556>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1[col] = df1[col].astype('category')
<ipython-input-61-f7725d793556>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1[col] = df1[col].astype('category')
<ipython-input-61-f7725d793556>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1[col] = df1[col].astype('category')
```

In [62]:

```
df2 = pd.concat([df1_log_norm, df1_cat], axis=1)
df2.head()
```

Out[62]:

	sqft_living_log	sqft_living15_log	price_log	grade_4	grade_5	grade_6	grade_7	grade_8
0	-1.121522	-1.019966	-1.424053	0	0	0	1	0
1	0.835051	-0.273574	0.368409	0	0	0	1	0
2	-2.194530	1.257118	-1.847602	0	0	1	0	0
3	0.153960	-0.972315	0.602610	0	0	0	1	0
4	-0.233515	-0.070754	0.260234	0	0	0	0	1

5 rows × 53 columns

In [63]:

```
X = df2.drop('price_log', axis=1)
y = df2['price_log']
```

In [64]:

```
X_int = sm.add_constant(X)
model = sm.OLS(y,X_int).fit()
model.summary()
```

```
C:\Users\jules\anaconda3\envs\learn-env\lib\site-packages\statsmodels\ts
a\tsatools.py:142: FutureWarning: In a future version of pandas all argu
ments of concat except for the argument 'objs' will be keyword-only.
  x = pd.concat(x[::order], 1)
```

Out[64]:

OLS Regression Results

Dep. Variable:	price_log	**R-squared:**	0.558
Model:	OLS	**Adj. R-squared:**	0.557
Method:	Least Squares	**F-statistic:**	497.5
Date:	Tue, 04 Jul 2023	**Prob (F-statistic):**	0.00
Time:	07:00:01	**Log-Likelihood:**	-20756.
No. Observations:	20538	**AIC:**	4.162e+04

In [65]:

```
# Check which variables strongly correlate with price

price_corr = df2.corr()[['price_log']].sort_values(by='price_log', ascending=False)

plt.figure(figsize=(4, 20))
heatmap = sns.heatmap(price_corr, annot=True, cmap='mako')
heatmap.set_title('Variables Correlating with Price', fontsize=14);
plt.savefig("images/df2_price_corr", bbox_inches='tight')
```

Iteration 2 Model Comments

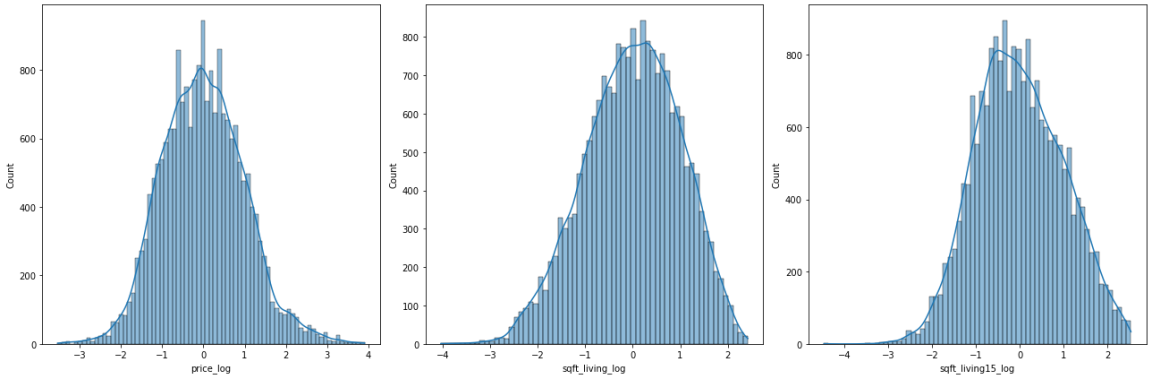
Unexpectedly, the adjusted R-squared values decreased. By only choosing the continuous and categorical variables that have a strong relationship in the model, I expected an increase in the adjusted R-squared value. However the cause of the decrease might be explained by removing sqft_above. It showed high correlation with price but also high correlation with sqft_living. By removing sqft_above because of multicollinearity, the iteration 2 model is a more reliable representation of the data.

Reviewing the P-value, there are some variables that are not statistically significant. I will remove those variables to further refine in iteration 3. Reviewing the co-efficients, there are some independent variables that large values and match up with the correlation heat map like grade 9, 10 and 11. However grade 12 and 13 with big co-efficients fall lower on the correlation heat map.

Distributions and KDE

In [66]:

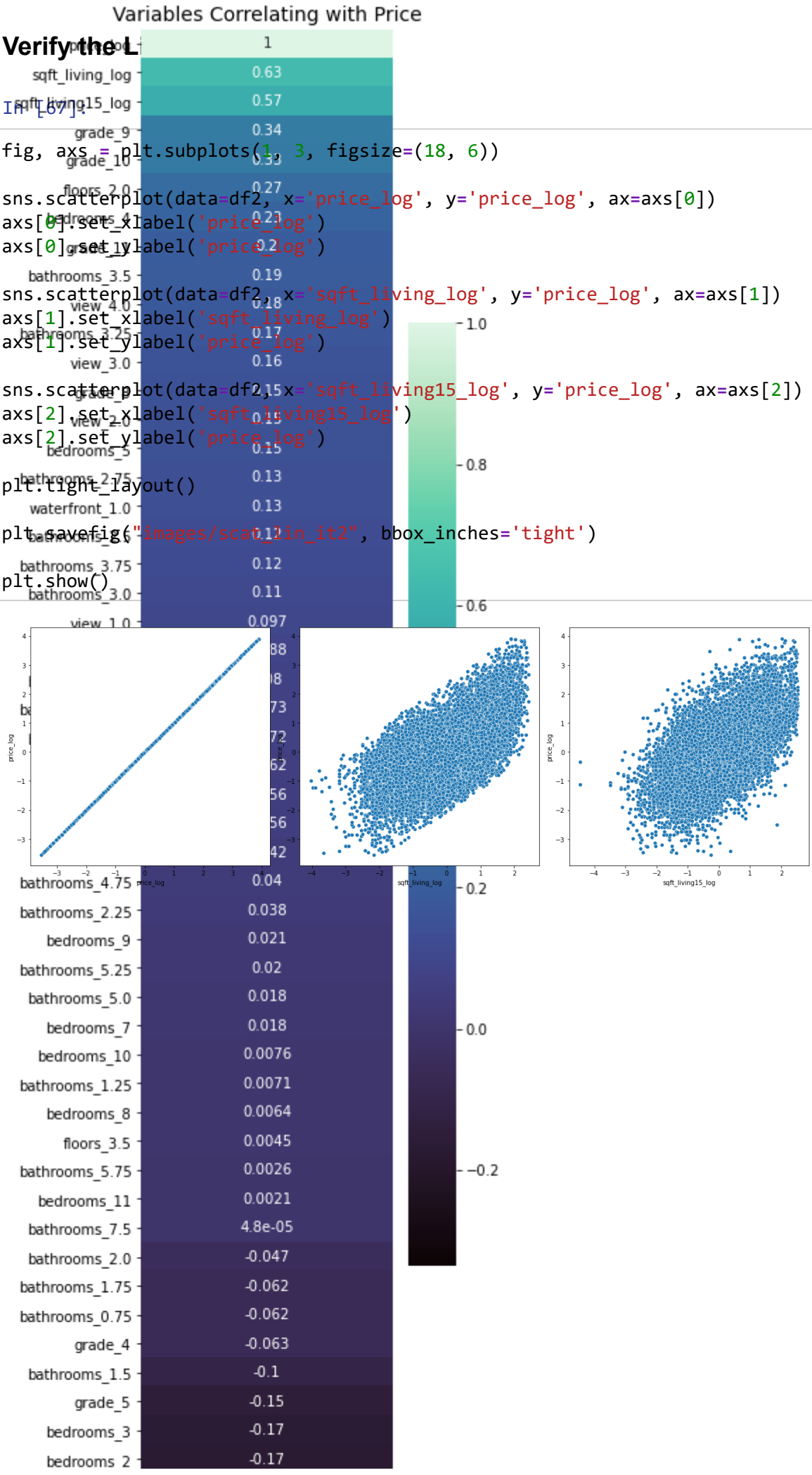
```
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

sns.histplot(data=df2['price_log'], kde=True, ax=axs[0])
axs[0].set_xlabel('price_log')

sns.histplot(data=df2['sqft_living_log'], kde=True, ax=axs[1])
axs[1].set_xlabel('sqft_living_log')

sns.histplot(data=df2['sqft_living15_log'], kde=True, ax=axs[2])
axs[2].set_xlabel('sqft_living15_log')

plt.tight_layout()

plt.savefig("images/dist_it2", bbox_inches='tight')

plt.show()
```

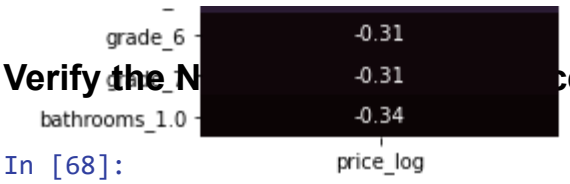


Verify the L

In [67]:

```
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

sns.scatterplot(data=df2, x='price_log', y='price_log', ax=axs[0])
axs[0].set_xlabel('price_log')
axs[0].set_ylabel('price_log')

sns.scatterplot(data=df2, x='sqft_living_log', y='price_log', ax=axs[1])
axs[1].set_xlabel('sqft_living_log')
axs[1].set_ylabel('price_log')

sns.scatterplot(data=df2, x='sqft_living15_log', y='price_log', ax=axs[2])
axs[2].set_xlabel('sqft_living15_log')
axs[2].set_ylabel('price_log')

plt.tight_layout()

plt.savefig("images/scatterin_it2", bbox_inches='tight')

plt.show()
```



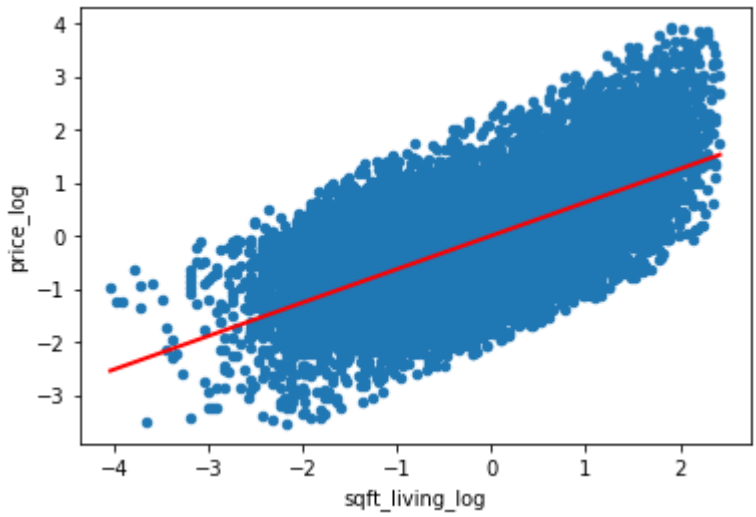


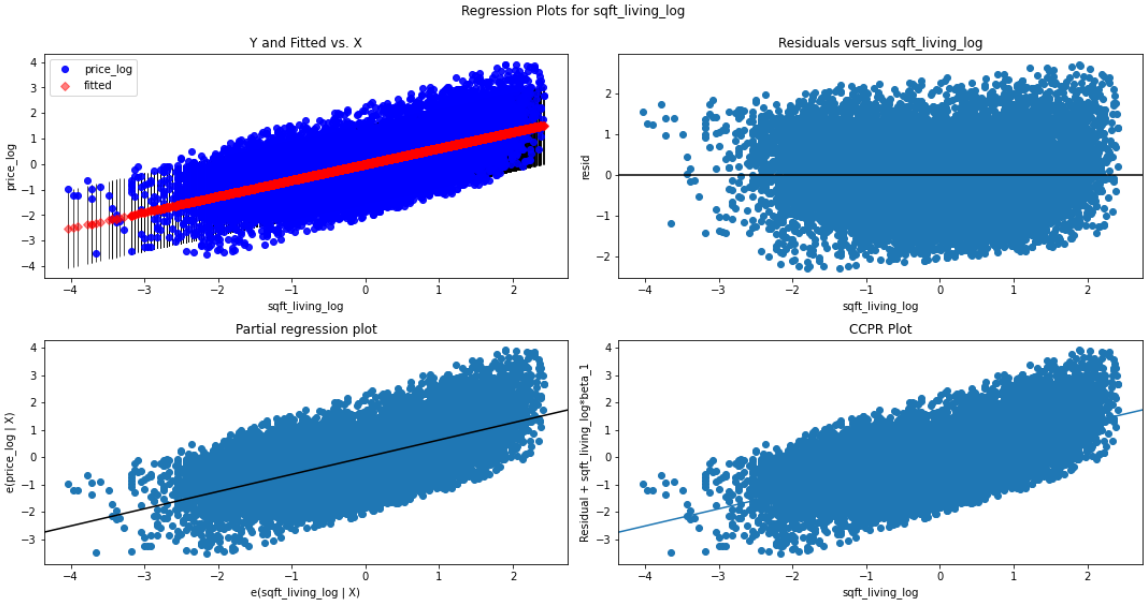
Verify the N cedasticity assumptions

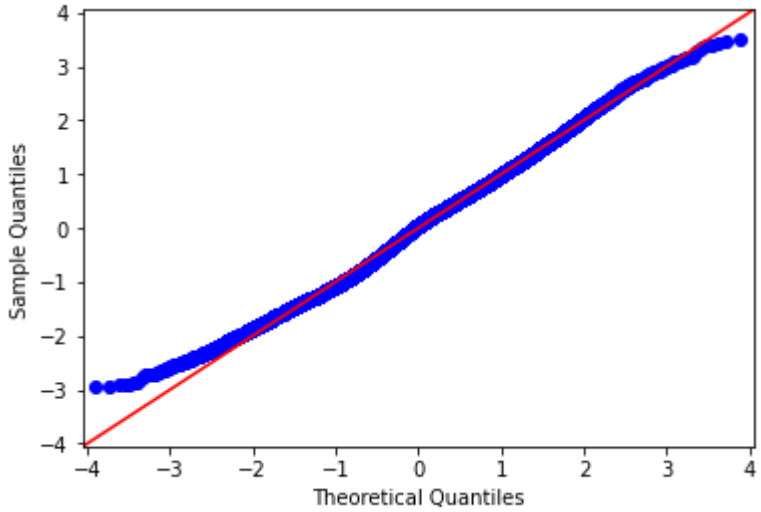
In [68]:

```
data=df2
f = 'price_log~sqft_living_log'
model = smf.ols(formula=f, data=data).fit()
print ('R-Squared:',model.rsquared)
print (model.params)

X_new = pd.DataFrame({'sqft_living_log': [data.sqft_living_log.min(), data.sqft_living_lo
preds = model.predict(X_new)

fig, ax = plt.subplots()
data.plot(kind='scatter', x='sqft_living_log', y='price_log', ax=ax)
ax.plot(X_new['sqft_living_log'], preds, c='red', linewidth=2)
plt.show()

fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_living_log", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True)

plt.savefig("images/price_log_sqft_living_log_norm_homo_it2", bbox_inches='tight')

plt.show()
```

```
R-Squared: 0.3962976141886999
Intercept          2.945560e-15
sqft_living_log    6.295217e-01
dtype: float64
```



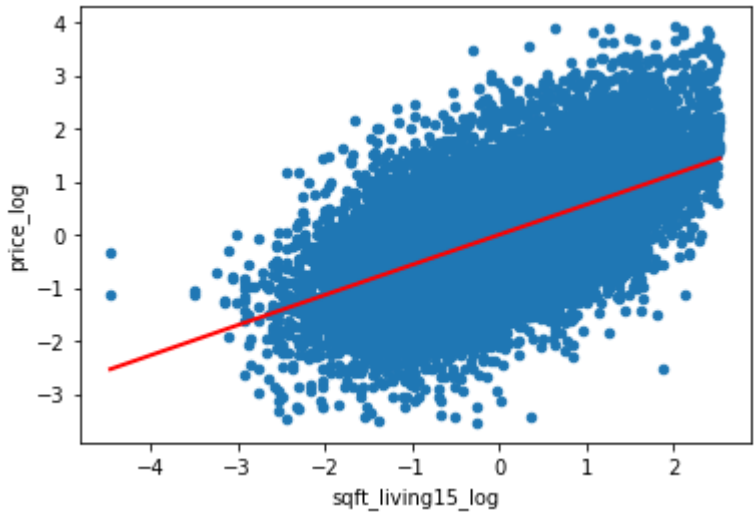

Regression Plots for sqft_living_log
Y and Fitted vs. X
price_log
fitted
price_log
sqft_living_log
Residuals versus sqft_living_log
resid
sqft_living_log
Partial regression plot
e(price_log | X)
e(sqft_living_log | X)
CCPR Plot
Residual + sqft_living_log*beta_1
sqft_living_log

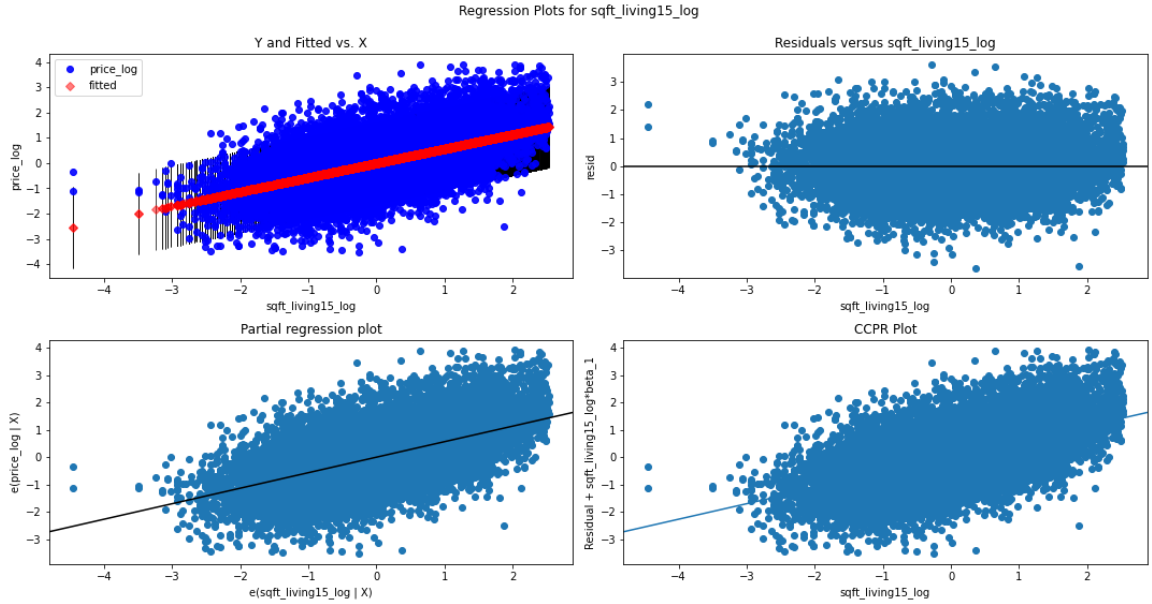

Sample Quantiles
Theoretical Quantiles

In [69]:

```
data=df2
f = 'price_log~sqft_living15_log'
model = smf.ols(formula=f, data=data).fit()
print ('R-Squared:',model.rsquared)
print (model.params)

X_new = pd.DataFrame({'sqft_living15_log': [data.sqft_living15_log.min(), data.sqft_livir
preds = model.predict(X_new)

fig, ax = plt.subplots()
data.plot(kind='scatter', x='sqft_living15_log', y='price_log', ax=ax)
ax.plot(X_new['sqft_living15_log'], preds, c='red', linewidth=2)
plt.show()

fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_living15_log", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True)

plt.savefig("images/price_log_sqft_living15_log_norm_homo_it2", bbox_inches='tight')

plt.show()
```

```
R-Squared: 0.32215433086739786
Intercept           -4.232031e-14
sqft_living15_log    5.675864e-01
dtype: float64
```





Linea

The no benefitted from being log transformed. They are no ed curve making it ideal for use in linear regress

All con ear assumption. Performing a polynomial regression is not recommended as it could result in the model overfitting.

Both sqft_living and sqft_living15 no longer violate the normality and homoscedasticity assumptions however there are some observations to note:

- The sqft_living R-squared value went down. This could be the result of outliers still being present before log transformation. As seen in the Q-Q plot, the tails are curved at the ends suggesting outliers in the data
- The sqft_living15 went up after log transformation which is expected however, like the sqft_living variable, the Q-Q plot suggests outliers are still present in the data

Iteration 3

For iteration 3, I will start by removing the outliers by reducing the data to within 2 standard deviations then observe if there has been an improvement in the R-squared value and error terms

Performing a polynomial regression may result in the model overfitting so in a final effort to increase the model's adjusted R-squared value, I will perform interactions. Iteration 3 is a good step to perform interactions because I have chosen the continuous and categorical variables that have a strong relationship with price.

I expect to see an increase in the adjusted R-squared compared to the iteration 2 model. Based on the observations of the linear assumptions, I will be able to perform model validation.

Removing Outliers

Reducing the data from 3 standard deviations to 2 in order to further remove outliers

In [70]:

```
# Check this is the data cleaned dataframe

df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20538 entries, 0 to 21596
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype   
---  ------         --------------  -----   
 0   price          20538 non-null  float64 
 1   bedrooms       20538 non-null  category
 2   bathrooms      20538 non-null  category
 3   sqft_living    20538 non-null  int64   
 4   sqft_lot       20538 non-null  int64   
 5   floors         20538 non-null  category
 6   waterfront     20538 non-null  category
 7   view           20538 non-null  category
 8   condition      20538 non-null  int64   
 9   grade          20538 non-null  category
 10  sqft_above     20538 non-null  int64   
 11  sqft_basement  20538 non-null  float64 
 12  yr_built       20538 non-null  int64   
 13  yr_renovated   20538 non-null  float64 
 14  zipcode        20538 non-null  int64   
 15  lat            20538 non-null  float64 
 16  long           20538 non-null  float64 
 17  sqft_living15  20538 non-null  int64   
 18  sqft_lot15     20538 non-null  int64   
dtypes: category(6), float64(5), int64(8)
memory usage: 2.3 MB
```

In [71]:

```
# Reduce outliers by reducing data size to within 2 standard deviations

filter_cols = ['sqft_living', 'sqft_lot', 'sqft_above', 'sqft_basement', 'sqft_lot15', 's

df3 = df[~df[filter_cols].apply(lambda x: np.abs(x - x.mean()) > 2 * x.std()).any(axis=1)
```

In [72]:

```
# Check entries

df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18737 entries, 0 to 21596
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          18737 non-null  float64
 1   bedrooms       18737 non-null  int64  
 2   bathrooms      18737 non-null  float64
 3   sqft_living    18737 non-null  int64  
 4   sqft_lot       18737 non-null  int64  
 5   floors         18737 non-null  float64
 6   waterfront     18737 non-null  float64
 7   view           18737 non-null  float64
 8   condition      18737 non-null  int64  
 9   grade          18737 non-null  int64  
 10  sqft_above     18737 non-null  int64  
 11  sqft_basement  18737 non-null  float64
 12  yr_built       18737 non-null  int64  
 13  yr_renovated   18737 non-null  float64
 14  zipcode        18737 non-null  int64  
 15  lat            18737 non-null  float64
 16  long           18737 non-null  float64
 17  sqft_living15  18737 non-null  int64  
 18  sqft_lot15     18737 non-null  int64  
dtypes: float64(9), int64(10)
memory usage: 2.9 MB
```

In [73]:

```
continuous = ['sqft_living', 'sqft_living15', 'price']

# Log transform
df3_log = np.log(df3[continuous])
df3_log.columns = [f'{column}_log' for column in df3_log.columns]

# Normalize
def normalize(feature):
    return (feature - feature.mean()) / feature.std()

df3_log_norm = df3_log.apply(normalize)

categoricals = ['grade', 'bathrooms', 'view', 'bedrooms', 'floors', 'waterfront']

# Convert columns to category data type
for col in categoricals:
    df3[col] = df[col].astype('category')

# Perform one-hot encoding
df3_cat = pd.get_dummies(df3[categoricals], prefix=categoricals, drop_first=True)

df4 = pd.concat([df3_log_norm, df3_cat], axis=1)
df4.head()
```

```
<ipython-input-73-778b36fd29db>:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df3[col] = df[col].astype('category')
<ipython-input-73-778b36fd29db>:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df3[col] = df[col].astype('category')
<ipython-input-73-778b36fd29db>:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df3[col] = df[col].astype('category')
<ipython-input-73-778b36fd29db>:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df3[col] = df[col].astype('category')
<ipython-input-73-778b36fd29db>:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df3[col] = df[col].astype('category')
<ipython-input-73-778b36fd29db>:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df3[col] = df[col].astype('category')
```

Out[73]:

	sqft_living_log	sqft_living15_log	price_log	grade_4	grade_5	grade_6	grade_7	grade_8
0	-1.067409	-0.977708	-1.412568	0	0	0	1	0
1	1.050739	-0.169396	0.511368	0	0	0	1	0
2	-2.229027	1.488281	-1.867183	0	0	1	0	0
3	0.313403	-0.926104	0.762747	0	0	0	1	0
4	-0.106070	0.050250	0.395258	0	0	0	0	1

5 rows × 61 columns

In [74]:

```
# Checking model before performing interactions
X = df4.drop('price_log', axis=1)
y = df4['price_log']

X_int = sm.add_constant(X)
model = sm.OLS(y,X_int).fit()
model.summary()
```

```
C:\Users\jules\anaconda3\envs\learn-env\lib\site-packages\statsmodels\ts
a\tsatools.py:142: FutureWarning: In a future version of pandas all argu
ments of concat except for the argument 'objs' will be keyword-only.
  x = pd.concat(x[::order], 1)
```

Out[74]:

OLS Regression Results

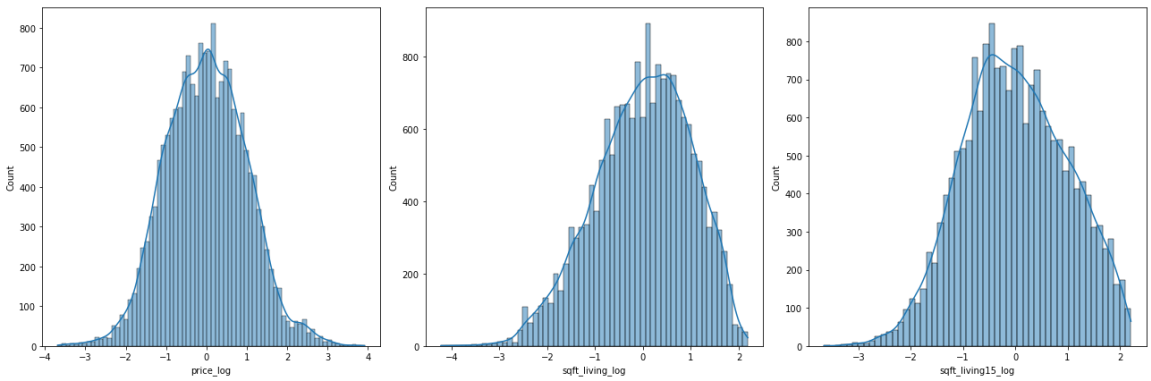
Dep. Variable:	price_log	**R-squared:**	0.492
Model:	OLS	**Adj. R-squared:**	0.491
Method:	Least Squares	**F-statistic:**	370.0
Date:	Tue, 04 Jul 2023	**Prob (F-statistic):**	0.00
Time:	07:00:12	**Log-Likelihood:**	-20233.
No. Observations:	18737	**AIC:**	4.057e+04

Interesting to note that reducing data to within 2 standard deviations instead of 3 standard drastically reduced the adjusted R-squared value. I will need to check the linearity assumptions to see how the data is distributed at 2 standard deviations.

Distributions and KDE

In [75]:

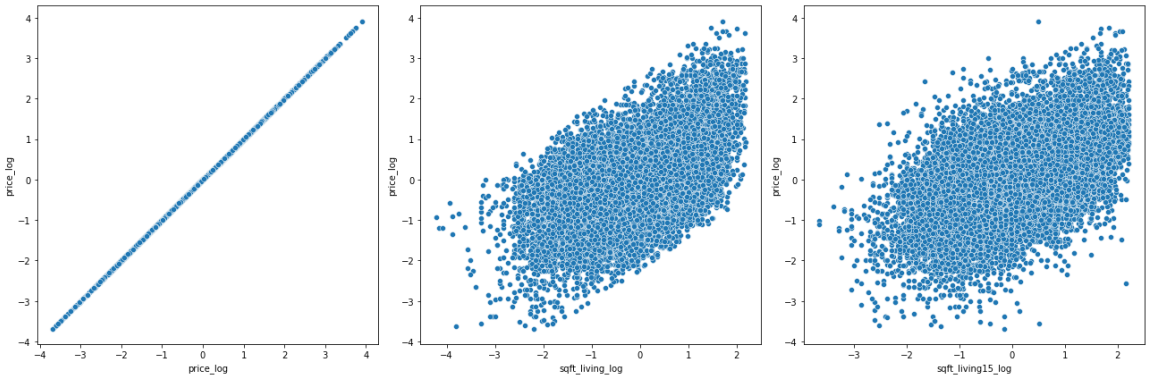
```
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

sns.histplot(data=df4['price_log'], kde=True, ax=axs[0])
axs[0].set_xlabel('price_log')

sns.histplot(data=df4['sqft_living_log'], kde=True, ax=axs[1])
axs[1].set_xlabel('sqft_living_log')

sns.histplot(data=df4['sqft_living15_log'], kde=True, ax=axs[2])
axs[2].set_xlabel('sqft_living15_log')

plt.tight_layout()

plt.savefig("images/dist_std2_it3", bbox_inches='tight')

plt.show()
```



Verify the Linearity assumption

In [76]:

```
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

sns.scatterplot(data=df4, x='price_log', y='price_log', ax=axs[0])
axs[0].set_xlabel('price_log')
axs[0].set_ylabel('price_log')

sns.scatterplot(data=df4, x='sqft_living_log', y='price_log', ax=axs[1])
axs[1].set_xlabel('sqft_living_log')
axs[1].set_ylabel('price_log')

sns.scatterplot(data=df4, x='sqft_living15_log', y='price_log', ax=axs[2])
axs[2].set_xlabel('sqft_living15_log')
axs[2].set_ylabel('price_log')

plt.tight_layout()

plt.savefig("images/scat_lin_std2_it3", bbox_inches='tight')

plt.show()
```

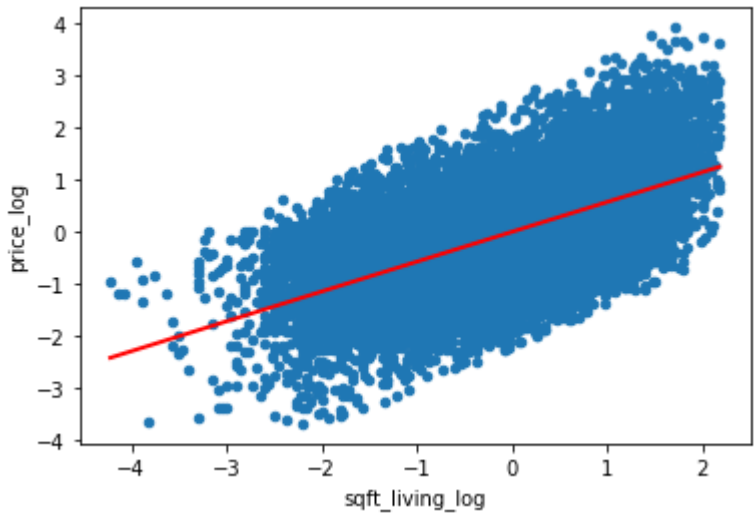


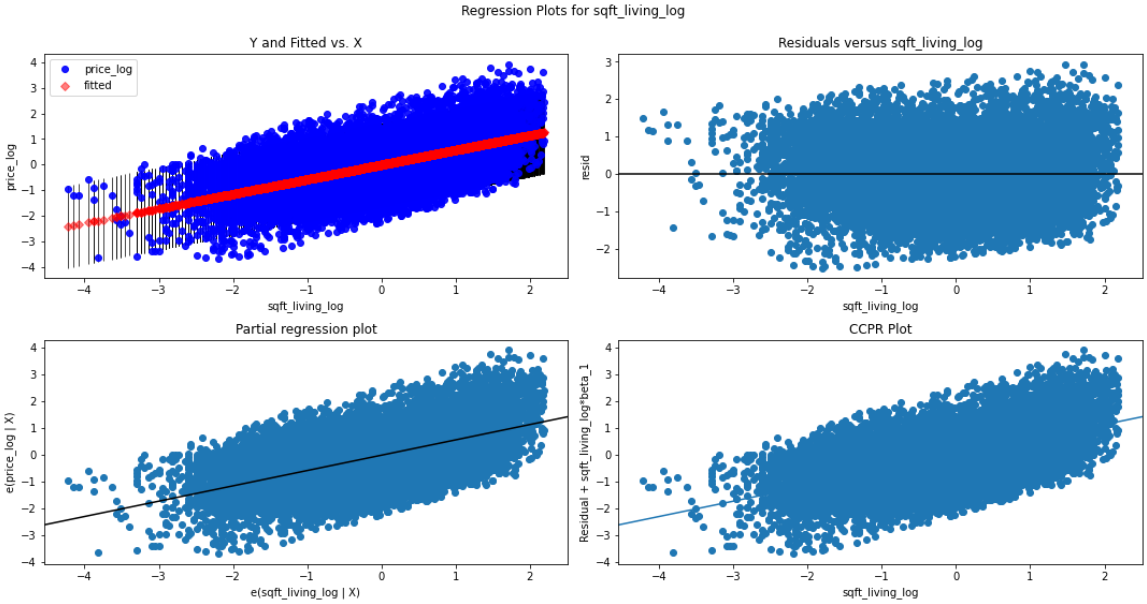
Verify the Normality and Homoscedasticity assumptions

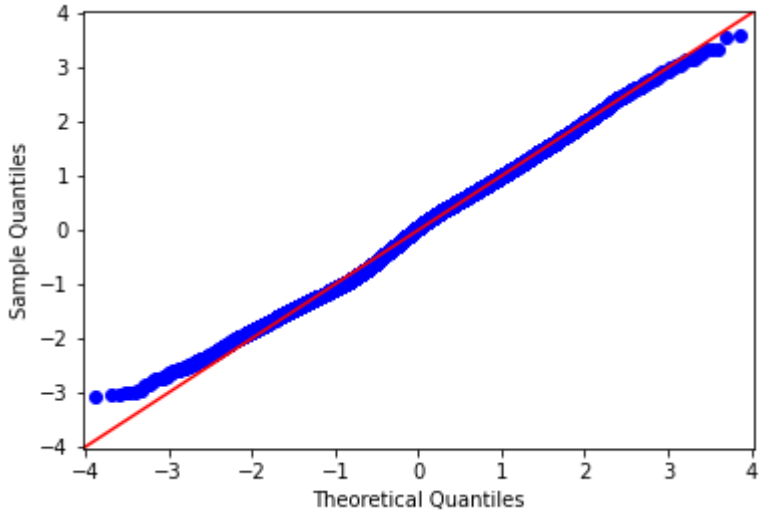
In [77]:

```
data=df4
f = 'price_log~sqft_living_log'
model = smf.ols(formula=f, data=data).fit()
print ('R-Squared:',model.rsquared)
print (model.params)

X_new = pd.DataFrame({'sqft_living_log': [data.sqft_living_log.min(), data.sqft_living_lo
preds = model.predict(X_new)

fig, ax = plt.subplots()
data.plot(kind='scatter', x='sqft_living_log', y='price_log', ax=ax)
ax.plot(X_new['sqft_living_log'], preds, c='red', linewidth=2)
plt.show()

fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_living_log", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True)

plt.savefig("images/price_log_sqft_living_std2_it3", bbox_inches='tight')

plt.show()
```

```
R-Squared: 0.32741545602699806
Intercept          1.368211e-13
sqft_living_log    5.722023e-01
dtype: float64
```



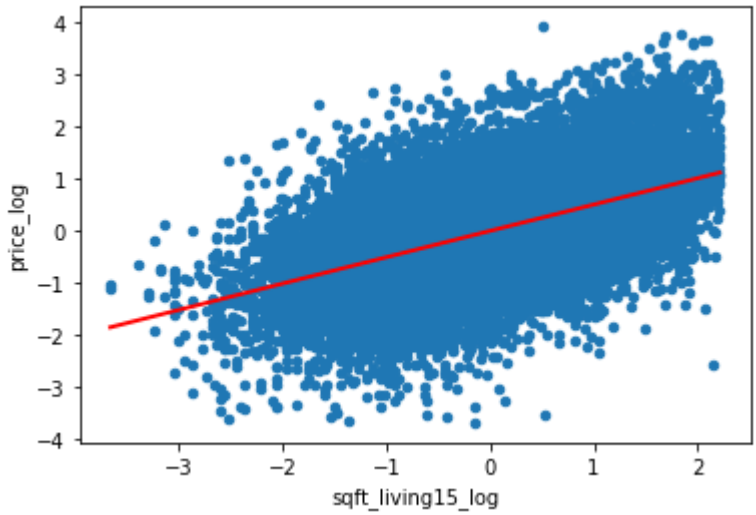

Regression Plots for sqft_living_log
Y and Fitted vs. X
price_log
fitted
price_log
sqft_living_log
Residuals versus sqft_living_log
resid
sqft_living_log
Partial regression plot
e(price_log | X)
e(sqft_living_log | X)
CCPR Plot
Residual + sqft_living_log*beta_1
sqft_living_log

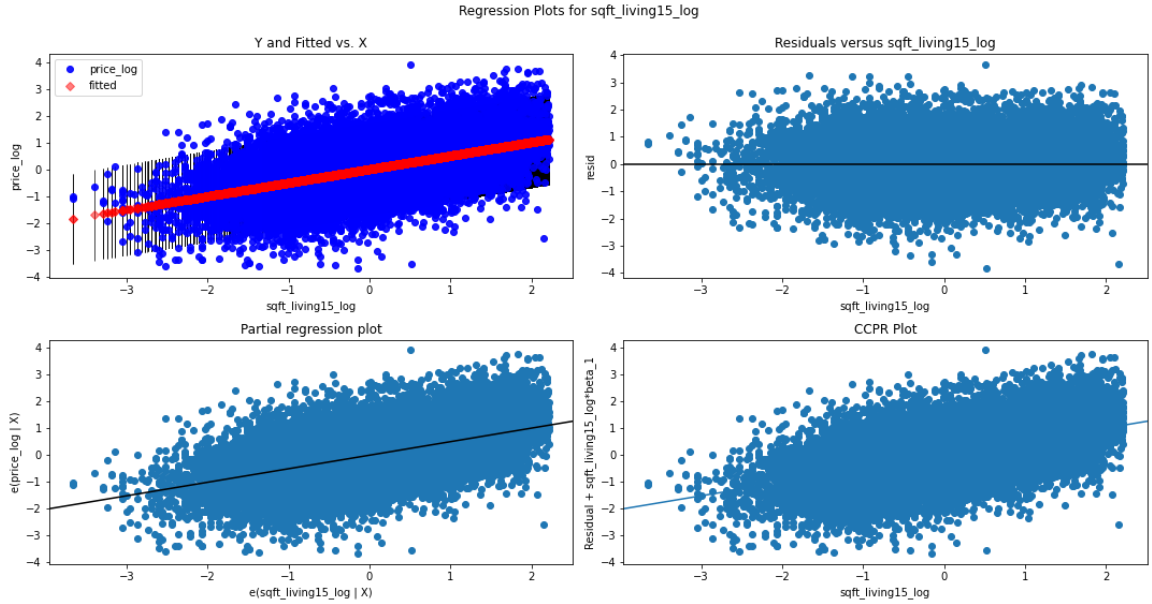

Sample Quantiles
Theoretical Quantiles

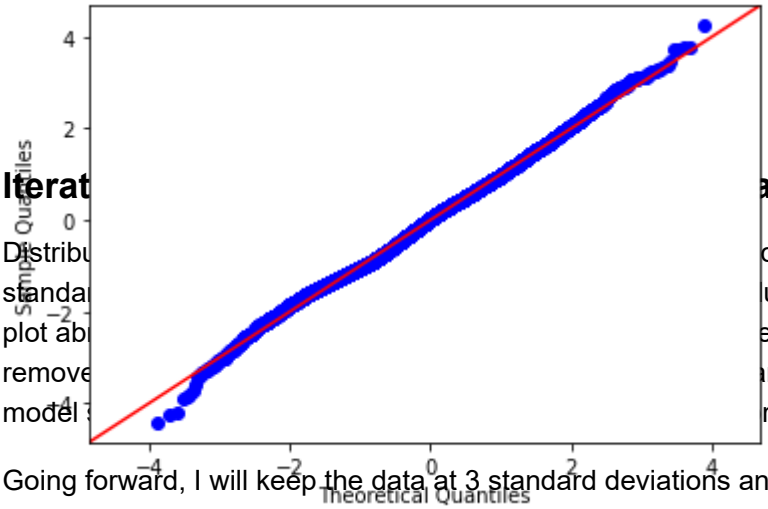
In [78]:

```
data=df4
f = 'price_log~sqft_living15_log'
model = smf.ols(formula=f, data=data).fit()
print ('R-Squared:',model.rsquared)
print (model.params)

X_new = pd.DataFrame({'sqft_living15_log': [data.sqft_living15_log.min(), data.sqft_livir
preds = model.predict(X_new)

fig, ax = plt.subplots()
data.plot(kind='scatter', x='sqft_living15_log', y='price_log', ax=ax)
ax.plot(X_new['sqft_living15_log'], preds, c='red', linewidth=2)
plt.show()

fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_living15_log", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True)

plt.savefig("images/price_log_sqft_living15_std2_it3", bbox_inches='tight')

plt.show()
```

```
R-Squared: 0.25533512453976526
Intercept            1.035630e-13
sqft_living15_log    5.053070e-01
dtype: float64
```







Iterat ations

Distribu drastically reduced with data being limited to 2 standar ues have resolved better however the scatter plot abi e skewness. It is evident that the outliers remove r from the marjority of the data, the iteration 2 model rice.

Going forward, I will keep the data at 3 standard deviations and perform interactions.

In [79]:

```
# Check this is the data cleaned dataframe

df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20538 entries, 0 to 21596
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype   
---  ------         --------------  -----   
 0   price          20538 non-null  float64 
 1   bedrooms       20538 non-null  category
 2   bathrooms      20538 non-null  category
 3   sqft_living    20538 non-null  int64   
 4   sqft_lot       20538 non-null  int64   
 5   floors         20538 non-null  category
 6   waterfront     20538 non-null  category
 7   view           20538 non-null  category
 8   condition      20538 non-null  int64   
 9   grade          20538 non-null  category
 10  sqft_above     20538 non-null  int64   
 11  sqft_basement  20538 non-null  float64 
 12  yr_built       20538 non-null  int64   
 13  yr_renovated   20538 non-null  float64 
 14  zipcode        20538 non-null  int64   
 15  lat            20538 non-null  float64 
 16  long           20538 non-null  float64 
 17  sqft_living15  20538 non-null  int64   
 18  sqft_lot15     20538 non-null  int64   
dtypes: category(6), float64(5), int64(8)
memory usage: 2.3 MB
```

In [80]:

```
# Repeating iteration 2 model as preparation for interactions and model validation

filter_cols = ['sqft_living', 'sqft_lot', 'sqft_above', 'sqft_basement', 'sqft_lot15', 's

df3 = df[~df[filter_cols].apply(lambda x: np.abs(x - x.mean()) > 3 * x.std()).any(axis=1)

continuous = ['sqft_living', 'sqft_living15','price']

# Log transform and normalize
df3_cont = df3[continuous]

# log features
log_names = [f'{column}_log' for column in df3_cont.columns]

df3_log = np.log(df3_cont)
df3_log.columns = log_names

# normalize (subract mean and divide by std)
def normalize(feature):
    return (feature - feature.mean()) / feature.std()

df3_log_norm = df3_log.apply(normalize)

categoricals = ['grade', 'bathrooms', 'view', 'bedrooms', 'floors', 'waterfront']

# Convert columns to category data type
for col in categoricals:
    df3[col] = df[col].astype('category')

# Perform one-hot encoding
df3_cat = pd.get_dummies(df3[categoricals], prefix=categoricals, drop_first=True)

df4 = pd.concat([df3_log_norm, df3_cat], axis=1)

X = df4.drop('price_log', axis=1)
y = df4['price_log']

X_int = sm.add_constant(X)
model = sm.OLS(y,X_int).fit()
model.summary()
```

```
<ipython-input-80-b0892a63b17e>:28: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  df3[col] = df[col].astype('category')
<ipython-input-80-b0892a63b17e>:28: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  df3[col] = df[col].astype('category')
<ipython-input-80-b0892a63b17e>:28: SettingWithCopyWarning:
```

Performing Interactions

In [81]:

```
regression = LinearRegression()

crossvalidation = KFold(n_splits=10, shuffle=True, random_state=1)
baseline = np.mean(cross_val_score(regression, X, y, scoring="r2", cv=crossvalidation))

baseline
```

Out[81]:

```
0.5550918109488123
```

In [82]:

```
from itertools import combinations

interactions = []

feat_combinations = combinations(X.columns, 2)

data = X.copy()
for i, (a, b) in enumerate(feat_combinations):
    data["interaction"] = data[a] * data[b]
    score = np.mean(
        cross_val_score(regression, data, y, scoring="r2", cv=crossvalidation)
    )
    if score > baseline:
        interactions.append((a, b, round(score, 3)))

    if i % 50 == 0:
        print(i)

print(
    "Top 3 interactions: %s"
    % sorted(interactions, key=lambda inter: inter[2], reverse=True)[:3]
)
```

```
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
750
800
850
900
950
1000
1050
1100
1150
1200
1250
1300
1350
1400
1450
1500
1550
1600
1650
1700
1750
Top 3 interactions: [('sqft_living_log', 'grade_4', 0.555), ('sqft_living_
log', 'bedrooms_2', 0.555), ('sqft_living_log', 'waterfront_1.0', 0.555)]
```

sqft_living_log has a relationship with 3 separate independent variables. This suggests that where other independent variables perform well, so will sqft_living. Therefore sqft_living is an essential variable to this model

Reviewing the iteration 2 model, grade_4 and bedrooms_2 are statistically insignificant. The interaction of sqft_living_log and waterfront_1.0 will be used. In terms of real world application, this interactions suggests houses with waterfront tend to be larger in square feet and price.

In [83]:

```
regression = LinearRegression()
crossvalidation = KFold(n_splits=10, shuffle=True, random_state=1)
final = X.copy()

final["sqft_living_log*waterfront_1.0"] = (
    final["sqft_living_log"] * final["waterfront_1.0"]
)

final_model = np.mean(
    cross_val_score(regression, final, y, scoring="r2", cv=crossvalidation)
)

print("Baseline Model:" )
print(baseline)
print("Baseline Model plus interaction:" )
print(final_model)
```

```
Baseline Model:
0.5550918109488123
Baseline Model plus interaction:
0.5551086194902063
```

In [84]:

```
# Adding interaction to model

df_inter_sm = sm.add_constant(final)
model = sm.OLS(y, final)
results = model.fit()

results.summary()
```

```
C:\Users\jules\anaconda3\envs\learn-env\lib\site-packages\statsmodels\ts
a\tsatools.py:142: FutureWarning: In a future version of pandas all argu
ments of concat except for the argument 'objs' will be keyword-only.
  x = pd.concat(x[::order], 1)
```

Out[84]:

OLS Regression Results

Dep. Variable:	price_log	**R-squared (uncentered):**	0.558
Model:	OLS	**Adj. R-squared (uncentered):**	0.557
Method:	Least Squares	**F-statistic:**	488.3
Date:	Tue, 04 Jul 2023	**Prob (F-statistic):**	0.00
Time:	07:09:11	**Log-Likelihood:**	-20754.
No. Observations:	20538	**AIC:**	4.161e+04

Adding the interaction did not increase the adjusted R-squared value. Since the model did not improve, it is most likely at its best fit and is ready for model validation.

Model Validation

In [85]:

```
df4.head()
```

Out[85]:

	sqft_living_log	sqft_living15_log	price_log	grade_4	grade_5	grade_6	grade_7	grade_8
0	-1.121522	-1.019966	-1.424053	0	0	0	1	0
1	0.835051	-0.273574	0.368409	0	0	0	1	0
2	-2.194530	1.257118	-1.847602	0	0	1	0	0
3	0.153960	-0.972315	0.602610	0	0	0	1	0
4	-0.233515	-0.070754	0.260234	0	0	0	0	1

5 rows × 61 columns

In [86]:

```
df4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20538 entries, 0 to 21596
Data columns (total 61 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sqft_living_log    20538 non-null  float64
 1   sqft_living15_log  20538 non-null  float64
 2   price_log          20538 non-null  float64
 3   grade_4            20538 non-null  uint8  
 4   grade_5            20538 non-null  uint8  
 5   grade_6            20538 non-null  uint8  
 6   grade_7            20538 non-null  uint8  
 7   grade_8            20538 non-null  uint8  
 8   grade_9            20538 non-null  uint8  
 9   grade_10           20538 non-null  uint8  
 10  grade_11           20538 non-null  uint8  
 11  grade_12           20538 non-null  uint8  
 12  grade_13           20538 non-null  uint8  
 13  bathrooms_0.75     20538 non-null  uint8  
 14  bathrooms_1.0      20538 non-null  uint8  
 15  bathrooms_1.25     20538 non-null  uint8  
 16  bathrooms_1.5      20538 non-null  uint8  
 17  bathrooms_1.75     20538 non-null  uint8  
 18  bathrooms_2.0      20538 non-null  uint8  
 19  bathrooms_2.25     20538 non-null  uint8  
 20  bathrooms_2.5      20538 non-null  uint8  
 21  bathrooms_2.75     20538 non-null  uint8  
 22  bathrooms_3.0      20538 non-null  uint8  
 23  bathrooms_3.25     20538 non-null  uint8  
 24  bathrooms_3.5      20538 non-null  uint8  
 25  bathrooms_3.75     20538 non-null  uint8  
 26  bathrooms_4.0      20538 non-null  uint8  
 27  bathrooms_4.25     20538 non-null  uint8  
 28  bathrooms_4.5      20538 non-null  uint8  
 29  bathrooms_4.75     20538 non-null  uint8  
 30  bathrooms_5.0      20538 non-null  uint8  
 31  bathrooms_5.25     20538 non-null  uint8  
 32  bathrooms_5.5      20538 non-null  uint8  
 33  bathrooms_5.75     20538 non-null  uint8  
 34  bathrooms_6.0      20538 non-null  uint8  
 35  bathrooms_6.25     20538 non-null  uint8  
 36  bathrooms_6.5      20538 non-null  uint8  
 37  bathrooms_6.75     20538 non-null  uint8  
 38  bathrooms_7.5      20538 non-null  uint8  
 39  bathrooms_7.75     20538 non-null  uint8  
 40  bathrooms_8.0      20538 non-null  uint8  
 41  view_1.0           20538 non-null  uint8  
 42  view_2.0           20538 non-null  uint8  
 43  view_3.0           20538 non-null  uint8  
 44  view_4.0           20538 non-null  uint8  
 45  bedrooms_2         20538 non-null  uint8  
 46  bedrooms_3         20538 non-null  uint8  
 47  bedrooms_4         20538 non-null  uint8  
 48  bedrooms_5         20538 non-null  uint8  
 49  bedrooms_6         20538 non-null  uint8  
 50  bedrooms_7         20538 non-null  uint8  
 51  bedrooms_8         20538 non-null  uint8  
 52  bedrooms_9         20538 non-null  uint8  
 53  bedrooms_10        20538 non-null  uint8  
 54  bedrooms_11        20538 non-null  uint8  
 55  floors_1.5         20538 non-null  uint8  
```

In [87]:

```
from sklearn.model_selection import train_test_split

X = df4.drop('price_log', axis=1)
y = df4['price_log']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

In [88]:

```
linreg = LinearRegression()

linreg.fit(X_train, y_train)
```

Out[88]:

```
▾ LinearRegression
LinearRegression()
```

In [89]:

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

In [90]:

```
from sklearn.metrics import mean_squared_error

train_mse = mean_squared_error(y_train, y_hat_train)
test_mse = mean_squared_error(y_test, y_hat_test)
print('Train Mean Squared Error:', train_mse)
print('Test Mean Squared Error: ', test_mse)
```

```
Train Mean Squared Error: 0.08426173699165498
Test Mean Squared Error: 1.3733220830125228e+20
```

Model Validation Comments

A huge case of overfitting. While the trainMSE is relatively low, the testMSE is through the roof indicating there is still a lot of variance in the data set being used.

```
 56  floors_2.0          20538 non-null  uint8
 57  floors_2.5          20538 non-null  uint8
 58  floors_3.0          20538 non-null  uint8
 59  floors_3.5          20538 non-null  uint8
 60  waterfront_1.0      20538 non-null  uint8
dtypes: float64(3), uint8(58)
memory usage: 1.8 MB
```

Iteration 4

Another iteration is required because the model is dramatically overfitting. It cannot be trusted that the data set is a true representation of the relationship between the independent variables and the target variable, price.

Based on iteration 3 I will drop these indepdendent variables as they are statistically insignificant

grade_4, grade_5, grade_6, grade_7, grade_8, grade_9, grade_10

bathrooms_0.75, bathrooms_1.0, bathrooms_1.25, bathrooms_1.5, bathrooms_1.75, bathrooms_2.0, bathrooms_2.25, bathrooms_2.5, bathrooms_2.75, bathrooms_3.0, bathrooms_5.0, bathrooms_5.75, bathrooms_7.5, bathrooms_6.0

bedrooms_8, bedrooms_9, bedrooms_10, bedrooms_11

floors_3.5

As an early observation only the highest of grades have a relationship with price.

The house requires more than 3 bathrooms before it starts having an effect on price.

The effect of bedrooms on price peaks at 8.

More than 3 floors has no relevance however that contradicts with sqft_living having a strong correlation with price. The data points for houses with more than 3.5 must be outliers. That would be believable in a real world application because it is rare to see houses with more than 3 floors.

Also it was observed in iteration 3 that when the data was reduced to within two standard deviations that the R-squared was reduced. Initially this was seen as negative but as we can see in the model validation, there is a lot of variance still in the data. For iteration 4, I will reduce the data to within two standard deviations.

Luckily, most of these are categorical variables. Since the linear assumptions have been proven in iteration 3, they will not needed to be checked and I can focus on the model and model validation.

In [91]:

```
filter_cols = ['sqft_living', 'sqft_lot', 'sqft_above', 'sqft_basement', 'sqft_lot15', 's

df3 = df[~df[filter_cols].apply(lambda x: np.abs(x - x.mean()) > 2 * x.std()).any(axis=1)

continuous = ['sqft_living', 'sqft_living15','price']

# Log transform and normalize
df3_cont = df3[continuous]

# log features
log_names = [f'{column}_log' for column in df3_cont.columns]

df3_log = np.log(df3_cont)
df3_log.columns = log_names

# normalize (subract mean and divide by std)
def normalize(feature):
    return (feature - feature.mean()) / feature.std()

df3_log_norm = df3_log.apply(normalize)

categoricals = ['grade', 'bathrooms', 'view', 'bedrooms', 'floors', 'waterfront']

# Convert columns to category data type
for col in categoricals:
    df3[col] = df[col].astype('category')

# Perform one-hot encoding
df3_cat = pd.get_dummies(df3[categoricals], prefix=categoricals, drop_first=True)

df4 = pd.concat([df3_log_norm, df3_cat], axis=1)

# Drop indepdendent variables
drop_grade = ['grade_4', 'grade_5', 'grade_6', 'grade_7', 'grade_8','grade_9','grade_10']
drop_bathrooms = ['bathrooms_0.75', 'bathrooms_1.0', 'bathrooms_1.25', 'bathrooms_1.5', '
drop_bedrooms = ['bedrooms_8', 'bedrooms_9', 'bedrooms_10', 'bedrooms_11']
drop_floors = ['floors_3.5']

X = df4.drop(['price_log'] + drop_grade + drop_bathrooms + drop_bedrooms + drop_floors, a
y = df4['price_log']

X_int = sm.add_constant(X)
model = sm.OLS(y,X_int).fit()
model.summary()
```

```
<ipython-input-91-2fef91563a97>:26: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  df3[col] = df[col].astype('category')
<ipython-input-91-2fef91563a97>:26: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  df3[col] = df[col].astype('category')
<ipython-input-91-2fef91563a97>:26: SettingWithCopyWarning:
```

In [92]:

```
# See highest to lowest co-efficients

# Get the summary table as HTML
summary_html = model.summary().tables[1].as_html()

# Convert the HTML table to a DataFrame
coef_df = pd.read_html(summary_html, header=0, index_col=0)[0]

# Convert coefficient values to numeric type
coef_df['coef'] = pd.to_numeric(coef_df['coef'], errors='coerce')

# Sort the coefficients by value in descending order
coef_df_sorted = coef_df.sort_values('coef', ascending=False)

# Print the sorted coefficient table
print(coef_df_sorted)
```

```
                         coef       std err       t  P>|t|        [0.025
\
grade_12             1.958800e+00  7.590000e-01   2.579  0.010  4.700000e-01
grade_11             8.378000e-01  1.190000e-01   7.020  0.000  6.040000e-01
view_4.0             8.281000e-01  7.500000e-02  11.024  0.000  6.810000e-01
bathrooms_3.75       6.839000e-01  9.800000e-02   6.965  0.000  4.910000e-01
bathrooms_4.25       6.397000e-01  2.410000e-01   2.653  0.008  1.670000e-01
floors_2.5           6.067000e-01  7.500000e-02   8.136  0.000  4.610000e-01
floors_3.0           5.829000e-01  3.300000e-02  17.700  0.000  5.180000e-01
view_3.0             4.813000e-01  4.700000e-02  10.246  0.000  3.890000e-01
sqft_living_log      4.684000e-01  1.000000e-02  47.742  0.000  4.490000e-01
waterfront_1.0       4.527000e-01  1.110000e-01   4.076  0.000  2.350000e-01
view_1.0             4.386000e-01  4.900000e-02   8.914  0.000  3.420000e-01
floors_1.5           4.275000e-01  2.000000e-02  21.290  0.000  3.880000e-01
bathrooms_3.25       4.158000e-01  4.400000e-02   9.462  0.000  3.300000e-01
view_2.0             4.082000e-01  3.000000e-02  13.433  0.000  3.490000e-01
bathrooms_3.5        4.080000e-01  4.000000e-02  10.138  0.000  3.290000e-01
bathrooms_4.0        3.899000e-01  1.280000e-01   3.057  0.002  1.400000e-01
bathrooms_4.5        2.460000e-01  1.620000e-01   1.521  0.128 -7.100000e-02
sqft_living15_log    2.131000e-01  8.000000e-03  26.717  0.000  1.970000e-01
const                1.691000e-01  5.600000e-02   2.996  0.003  5.800000e-02
bedrooms_2           4.730000e-02  5.600000e-02   0.842  0.400 -6.300000e-02
bedrooms_1           2.020000e-02  1.400000e-02   1.460  0.144 -7.000000e-03
bathrooms_6.5        2.131000e-14  1.030000e-14   2.070  0.038  1.130000e-15
bathrooms_5.5        9.599000e-15  5.350000e-15   1.793  0.073 -8.950000e-16
grade_13             3.395000e-15  1.910000e-15   1.777  0.076 -3.500000e-16
bathrooms_7.75       1.389000e-15  5.250000e-16   2.644  0.008  3.590000e-16
bathrooms_6.75       2.634000e-16  4.880000e-16   0.540  0.589 -6.930000e-16
bathrooms_8.0       -6.013000e-16  1.550000e-16  -3.882  0.000 -9.050000e-16
bathrooms_6.25      -6.974000e-15  3.480000e-15  -2.007  0.045 -1.380000e-14
bathrooms_4.75      -1.666000e-14  6.300000e-15  -2.645  0.008 -2.900000e-14
bedrooms_3          -3.191000e-01  5.600000e-02  -5.683  0.000 -4.290000e-01
bedrooms_4          -3.968000e-01  5.800000e-02  -6.826  0.000 -5.110000e-01
bedrooms_5          -4.365000e-01  6.300000e-02  -6.941  0.000 -5.600000e-01
bedrooms_6          -4.694000e-01  8.700000e-02  -5.424  0.000 -6.390000e-01
bedrooms_7          -5.075000e-01  1.830000e-01  -2.773  0.006 -8.660000e-01
bathrooms_5.25      -9.247000e-01  7.610000e-01  -1.215  0.224 -2.416000e+00
                          0.975]
grade_12             3.447000e+00
grade_11             1.072000e+00
view_4.0             9.750000e-01
bathrooms_3.75       8.760000e-01
bathrooms_4.25       1.112000e+00
floors_2.5           7.530000e-01
floors_3.0           6.470000e-01
view_3.0             5.730000e-01
sqft_living_log      4.880000e-01
waterfront_1.0       6.700000e-01
view_1.0             5.350000e-01
floors_1.5           4.670000e-01
bathrooms_3.25       5.020000e-01
view_2.0             4.680000e-01
bathrooms_3.5        4.870000e-01
bathrooms_4.0        6.400000e-01
bathrooms_4.5        5.630000e-01
sqft_living15_log    2.290000e-01
const                2.800000e-01
bedrooms_2           1.570000e-01
floors_2.0           4.700000e-02
bathrooms_6.5        4.150000e-14
```

Iteration 4 Comments

Again the adjusted R-squared value has lowered as more outliers are removed, therefore reducing the variance.

Model Validation

In [93]:

```
from sklearn.model_selection import train_test_split

drop_grade = ['grade_4', 'grade_5', 'grade_6', 'grade_7', 'grade_8', 'grade_9','grade_10']
drop_bathrooms = ['bathrooms_0.75', 'bathrooms_1.0', 'bathrooms_1.25', 'bathrooms_1.5', '
drop_bedrooms = ['bedrooms_8', 'bedrooms_9', 'bedrooms_10', 'bedrooms_11']
drop_floors = ['floors_3.5']

X = df4.drop(['price_log'] + drop_grade + drop_bathrooms + drop_bedrooms + drop_floors, a
y = df4['price_log']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

linreg = LinearRegression()
linreg.fit(X_train, y_train)
LinearRegression()
y_hat_train = linreg.predict(X_train)
y_hat_test = linreg.predict(X_test)
from sklearn.metrics import mean_squared_error
train_mse = mean_squared_error(y_train, y_hat_train)
test_mse = mean_squared_error(y_test, y_hat_test)
print('Train Mean Squared Error:', train_mse)
print('Test Mean Squared Error: ', test_mse)
```

```
Train Mean Squared Error: 0.5702729561689224
Test Mean Squared Error:  0.5853459305234379
```

```
bathrooms_5.5       2.010000e-14
grade_13            7.140000e-15
bathrooms_7.75      2.470000e-15
bathrooms_6.75      1.220000e-15
bathrooms_8.0      -2.980000e-16
bathrooms_6.25     -1.620000e-16
bathrooms_4.75     -4.310000e-15
bedrooms_3         -2.090000e-01
bedrooms_4         -2.830000e-01
bedrooms_5         -3.130000e-01
bedrooms_6         -3.000000e-01
bedrooms_7         -1.490000e-01
bathrooms_5.25      5.670000e-01
```

Slightly higher errors in trainMSE however the testMSE has greatly reduced leading to a model that fits much better. This is clear the effect of removing more variance by reducing the data set within 2 standard deviations and removing statistically insignificant variables.

In [94]:

```
# Using K-Fold Cross Validation to verify
drop_grade = ['grade_4', 'grade_5', 'grade_6', 'grade_7', 'grade_8','grade_9','grade_10']
drop_bathrooms = ['bathrooms_0.75', 'bathrooms_1.0', 'bathrooms_1.25', 'bathrooms_1.5', '
drop_bedrooms = ['bedrooms_8', 'bedrooms_9', 'bedrooms_10', 'bedrooms_11']
drop_floors = ['floors_3.5']

X = df4.drop(['price_log'] + drop_grade + drop_bathrooms + drop_bedrooms + drop_floors, a
y = df4['price_log']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

linreg = LinearRegression()

linreg.fit(X_train, y_train)

LinearRegression()

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

from sklearn.model_selection import cross_validate
from sklearn.metrics import mean_squared_error

scoring = {'mse': 'neg_mean_squared_error'}
results = cross_validate(linreg, X, y, scoring=scoring, cv=10)

test_mse_scores = -results['test_mse']

train_predictions = linreg.predict(X)
train_mse = mean_squared_error(y, train_predictions)

print("Train Mean Squared Error:", train_mse)
print("Test Mean Squared Error:", test_mse_scores.mean())
```

```
Train Mean Squared Error: 0.5740418030948392
Test Mean Squared Error: 0.5772426558303619
```

In [95]:

```
# Get top 5 variables

from sklearn.feature_selection import RFE

selector = RFE(linreg, n_features_to_select=5)
selector = selector.fit(X_train, y_train.values)

selector.support_
```

Out[95]:

```
array([False, False,  True,  True, False, False, False,  True, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False,  True,  True, False, False, False, False,
       False, False, False, False, False, False, False])
```

In [96]:

```
selected_columns = X_train.columns[selector.support_]
linreg.fit(X_train[selected_columns], y_train)
print(selected_columns)
```

```
Index(['grade_11', 'grade_12', 'bathrooms_3.75', 'view_3.0', 'view_4.0'],
dtype='object')
```

Evaluation

After 4 iterations, a collection of the top independent variables were used to create the final model. Using sklearn's feature selector the top 5 variables that have a strong relationship with the target variable price are:

- grade_11
- grade_12
- bathrooms_3.75
- view_3.0
- view_4.0

I believe this linear regression model successfully solves the business problem of identifying the five factors that have a strong relationship with price. Initially I expected the adjusted R-squared of the model to increase as more statistically significant variables were identified. However, reducing variance and outliers proved to be the more important factor which result in a lower adjusted R-squared value.

Reducing the data set to within 2 standard variations initially looked extreme as the adjusted R-squared dipped to almost half of the baseline model. However when the first model validation was performed, it showed that the model was dramatically overfitting indicating a lot of variance was still present in the data set.

During each iteration, the P-values held true with the correlation heat map. This provides me with great confidence that the correct variables were being chosen to be used in the model. It is important for data to also make sense in a real world application. Looking at the top 5 variables, they are realistic in the sense that a typical person would expect those variables when evaluating the price of a house.

Conclusion

King County has a ranking system that represents the construction quality of improvements. They are generally defined as:

1-3 Falls short of minimum building standards. Normally cabin or inferior structure.

4 Generally older, low quality construction. Does not meet code.

5 Low construction costs and workmanship. Small, simple design.

6 Lowest grade currently meeting building code. Low quality materials and simple designs.

7 Average grade of construction and design. Commonly seen in plats and older sub-divisions.

8 Just above average in construction and design. Usually better materials in both the exterior and interior finish work.

9 Better architectural design with extra interior and exterior design and quality.

10 Homes of this quality generally have high quality features. Finish work is better and more design quality is seen in the floor plans. Generally have a larger square footage.

11 Custom design and higher quality finish work with added amenities of solid woods, bathroom fixtures and more luxurious options.

12 Custom design and excellent builders. All materials are of the highest quality and all conveniences are present.

13 Generally custom designed and built. Mansion level. Large amount of highest quality cabinet work, wood trim, marble, entry ways etc.

The results Grade 11 and 12 sit on the higher end of the ranking system and is reflective of the model that has been produced. Investing in an excellent builder, high quality materials and luxurious options will yield a higher house price.

Based on the statistical significance of the number of bathrooms throughout the iterations, it was observed that the number of bathrooms only started having a strong relationship with price at 3. Again this is reflective in the results at 3.75. The results suggest that many of the higher priced homes have a minimum 3 bathrooms.

The features of the home are not the only important factors in raising the price of a house. The location is just as important and in this case, if the house has a view. King County has a variety of landmarks, ocean, lake and views. Opting to build a house within view of these natural and manmade points of interest and it will have a positive effect on the price.

In conclusion, in order for the east coast residential builder to be successful on the east coast, specifically King County, they must consider:

- Creating a custom design using high quality materials, high quality finish work and luxurious options
- Incorporating 3 or more bathrooms into their designs
- Choosing a location of the house with a great view of local points of interest