Final Project Submission

Please fill out:

- Student name: Michael Holthouser
- Student pace: self paced
- Scheduled project review date/time:
- Instructor name: Claude Fried
- Blog post URL:

Home Renovation Recommendations For King County, WA

Author: Michael Holthouser

Overview

I have been tasked with analyzing housing data for King County Washington. My goal is to provide information for a real estate agency to help homeowners with recommendation on helpful renovations to increase the selling price of their homes. With the use of multiple linear regression modeling, I am hopeful to give insight on which home renovations are best prioritized in terms of increasing the sale price.

Business Understanding

King County Realtors needs guidance on assisting clients with recommendations on which home renovations may increase the estimated value of their homes.

- Stakeholder: King County Realtors
- Business Question: Which home renovations might increase the estimated value of their homes, and by what amount?

The Data

This data comes from the King County House Sales dataset which is available on the King County Open Data King County Open Data. The data includes all data of single-family home sales from 2014-2015.

The features that I will be investigating are the following:

- `price` - sales price
- `bedrooms` - Number of bedrooms
- `bathrooms` - Number of bathrooms
- `sqft_living` - Square footbage of living space in the home
- `sqft_lot` - Square footage of the lot
- `floors` - Number of floors (levels) in house
- `waterfront` - whether the house is on a waterfront
- `grade` - Overall grade of the house. Related to the construction and design of the house
- `yr_built` - Year when house was built
- `condition` - How good the overall condition of the house is. Related to maintenance of house

Import packages and libraries

In [2]:
```
# import packages and libraries
import pandas as pd
import numpy as np
from numpy.random import randn
import seaborn as sns
sns.set(style='dark')


from matplotlib import pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from sklearn import preprocessing
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.outliers_influence import variance_inflation_factor
import scipy.stats as stats
from sklearn.preprocessing import StandardScaler
import sklearn.metrics as metrics
```

Import the data

In [3]:
```
kchd = pd.read_csv("kc_house_data.csv", usecols=["price", "bedrooms", "bath
                                                  "floors", "waterfron

kchd.head()
```

Out[3]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	grade	yr_bu
0	221900.0	3	1.00	1180	5650	1.0	NaN	Average	7 Average	19
1	538000.0	3	2.25	2570	7242	2.0	NO	Average	7 Average	19
2	180000.0	2	1.00	770	10000	1.0	NO	Average	6 Low Average	19
3	604000.0	4	3.00	1960	5000	1.0	NO	Very Good	7 Average	19
4	510000.0	3	2.00	1680	8080	1.0	NO	Average	8 Good	19

Data Cleaning

In [4]:
```
## Get info of the data frame
kchd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 10 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   19221 non-null  object
 7   condition    21597 non-null  object
 8   grade        21597 non-null  object
 9   yr_built     21597 non-null  int64
dtypes: float64(3), int64(4), object(3)
memory usage: 1.6+ MB
```

First observations are that I do appear to have some missing values in the waterfront column, and I also have three columns that I will need to change from an object data type into a numerical data type in order to run a linear regression.

In [5]:
```
## Get descriptive stats
kchd.describe()
```

Out[5]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	
count	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	21597.000000	21597.
mean	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	1.494096	1970.
std	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	0.539683	29.
min	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.000000	1900.
25%	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1.000000	1951.
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	1975.
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	2.000000	1997.
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	2015.

From this statistics table, there are a some outliers that will need to be addressed as well. For example, 33 bedrooms.

Remove unncessary columns: sqft_lot and yr_built

Seeing that we are only looking for renovations, I felt that it was appropriate to remove `sqft_lot` and `yr_built` since they are not techinically renovations you can perfom on a home.

In [6]:
```
kchd.drop(columns = ['sqft_lot', 'yr_built'], inplace=True)
```

In [7]:
```
## Check to see if dropped columns have been removed
kchd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 8 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   floors       21597 non-null  float64
 5   waterfront   19221 non-null  object 
 6   condition    21597 non-null  object 
 7   grade        21597 non-null  object 
dtypes: float64(3), int64(2), object(3)
memory usage: 1.3+ MB
```

Check for missing value

In [8]:
```
kchd.isna().sum()
```

Out[8]:
```
price             0
bedrooms          0
bathrooms         0
sqft_living       0
floors            0
waterfront     2376
condition         0
grade             0
dtype: int64
```

In [9]:
```
# Check the unique values of the waterfront column
kchd['waterfront'].unique()
```

Out[9]:
```
array([nan, 'NO', 'YES'], dtype=object)
```

In [10]:
```
# Get a value count of each entry in waterfront
kchd['waterfront'].value_counts(dropna=False)
```

Out[10]:
```
NO     19075
NaN     2376
YES      146
Name: waterfront, dtype: int64
```

Since 19,075 of 21,597 entries (about 88%) of the homes have NO for waterfront, we can assume that the 2,376 NaN columns can be filled with NO.

In [11]:
```
# Change all NaN values withe "NO"
kchd['waterfront'].fillna("NO", inplace=True)
```

In [12]:
```
# Check to see if NaN values have been changed
kchd['waterfront'].value_counts(dropna=False)
```

Out[12]:
```
NO     21451
YES      146
Name: waterfront, dtype: int64
```

In [13]:
```
# # Convert the waterfront column to the data type int, and make YES = 1 an
kchd['waterfront'] = kchd['waterfront'].map({'YES': 1, 'NO': 0})
```

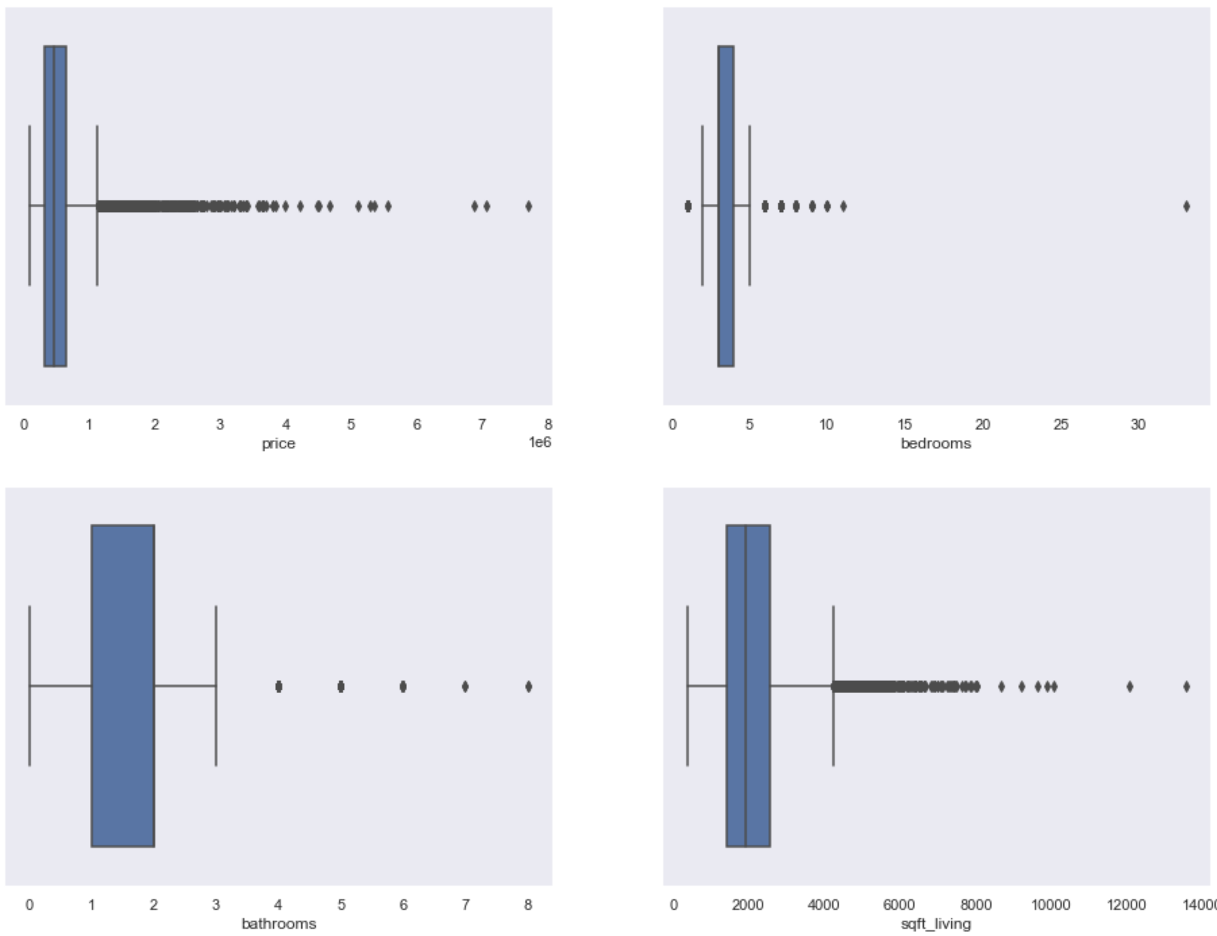
I obtain the code above from stackover flow https://stackoverflow.com/questions/40901770/is-there-a-simple-way-to-change-a-column-of-yes-no-to-1-0-in-a-pandas-dataframe (https://stackoverflow.com/questions/40901770/is-there-a-simple-way-to-change-a-column-of-yes-no-to-1-0-in-a-pandas-dataframe)

In [14]:
```
kchd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 8 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   floors       21597 non-null  float64
 5   waterfront   21597 non-null  int64  
 6   condition    21597 non-null  object 
 7   grade        21597 non-null  object 
dtypes: float64(3), int64(3), object(2)
memory usage: 1.3+ MB
```

Checking for outliers

In [15]:
```
# Box plots on features where outliers exist
plt.figure(figsize=(16, 12))
plt.subplot(221)
sns.boxplot(kchd['price'])
plt.subplot(222)
sns.boxplot(kchd['bedrooms'])
plt.subplot(223)
sns.boxplot(kchd.bathrooms.astype('int'))
plt.subplot(224)
sns.boxplot(kchd['sqft_living'])
sns.set_theme(style="whitegrid");
```



Box plots are a great way to visually see outliers in a dataset. The outlier that caught my eye the most was within the bedrooms feature. 33 bedrooms was the clear cut largest outlier, so I will remove it, in the code below.

In [16]:
```
# Value count of bedrooms
kchd['bedrooms'].value_counts()
```

Out[16]:
```
3     9824
4     6882
2     2760
5     1601
6      272
1      196
7       38
8       13
9        6
10       3
11       1
33       1
Name: bedrooms, dtype: int64
```

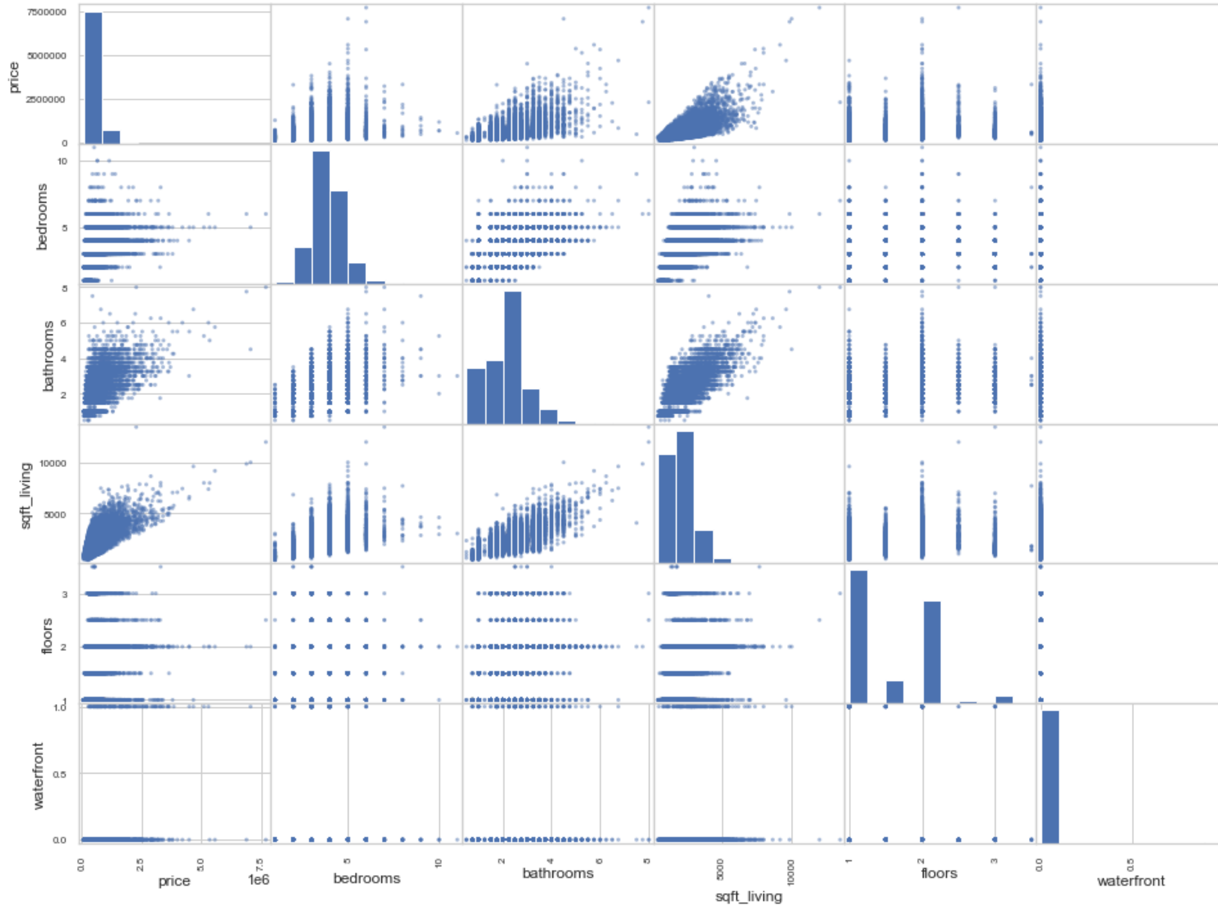
Since there is only one entry with 33 bedrooms, I will simply delete that row.

In [17]:
```
kchd = kchd[(kchd.bedrooms != 33)]

#Check if record has been removed
kchd['bedrooms'].value_counts()
```

Out[17]:
```
3     9824
4     6882
2     2760
5     1601
6      272
1      196
7       38
8       13
9        6
10       3
11       1
Name: bedrooms, dtype: int64
```

Scatter matrix to check if variables are normally distributed

In [18]:
```
# Scatter Matrix
pd.plotting.scatter_matrix(kchd, figsize=(16, 12));
```



A scatter matrix is a great way to show if the features in our model are normally distributed. From the matrix, you can tell that there is a clear linear relationship between sqft_living and price. As sqft_living increases the price increases as well.

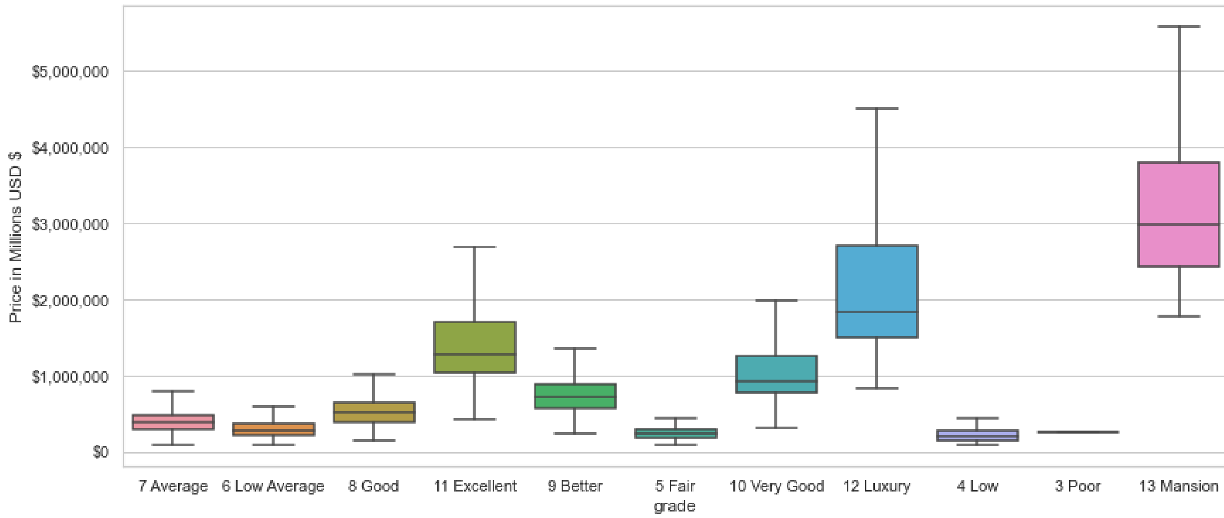
Box plots of condition and grade columns

For these particular features, a box plot visualization does a better job of showing it's relationship with price.

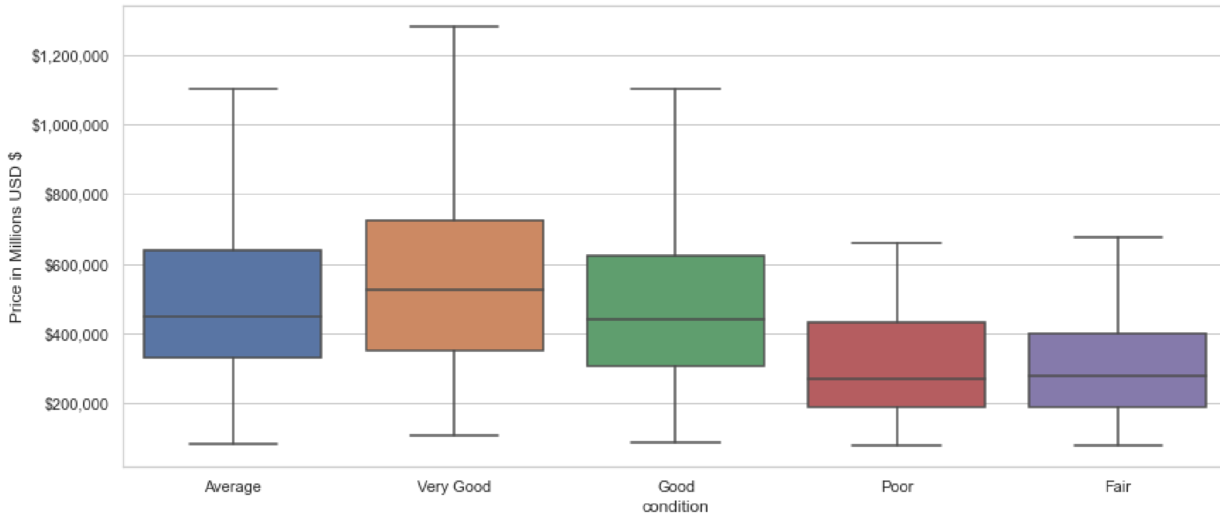
- As the quality increases, so does the grade. A higher grade will result in a higher selling price

- Similarly, the more maintenance done to a home to keep it in at least an "average" condition, will result in a higher asking price.

In [63]:
```
# Create bar chart of the grade column
plt.figure(figsize=(14,6))

ax = sns.boxplot(x='grade', y='price', showfliers=False, data=kchd)
ax.set(ylabel='Price in Millions USD $')
# Use automatic StrMethodFormatter
ax.yaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('${x:,.0f}'))


plt.show()
```



In [64]:
```
# Create bar chart of the condition column
plt.figure(figsize=(14,6))

ax = sns.boxplot(x='condition', y='price', showfliers=False, data=kchd)
ax.set(ylabel='Price in Millions USD $')
# Use automatic StrMethodFormatter
ax.yaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('${x:,.0f}'))


plt.show()
```



Change data types of grade and condition columns

The `grade` and `condition` features are currently in a string format. The code below with change their data types to integers, to be able to include them in my regression model.

In [58]:
```
kchd['grade'].unique()
```

Out[58]:
```
array(['7 Average', '6 Low Average', '8 Good', '11 Excellent', '9 Bette
r',
       '5 Fair', '10 Very Good', '12 Luxury', '4 Low', '3 Poor',
       '13 Mansion'], dtype=object)
```

In [59]:
```
## split the string from the number in order to change the the data type to
kchd['grade'] = kchd['grade'].str.split(n=1, expand=True)[0].astype(int)
kchd['grade'].unique()
```

Out[59]:
```
array([ 7,  6,  8, 11,  9,  5, 10, 12,  4,  3, 13])
```

In [60]:
```
kchd['condition'].unique()
```

Out[60]:
```
array(['Average', 'Very Good', 'Good', 'Poor', 'Fair'], dtype=object)
```

In [61]:
```
kchd['condition'] = kchd['condition'].map({'Poor': 1, 'Fair': 2, 'Average':
kchd['condition'].unique()
```

Out[61]:
```
array([3, 5, 4, 1, 2])
```

In [62]:
```
## Check if all data types are of a numeric data type
kchd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21596 entries, 0 to 21596
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   price        21596 non-null  float64
 1   bedrooms     21596 non-null  int64  
 2   bathrooms    21596 non-null  float64
 3   sqft_living  21596 non-null  int64  
 4   floors       21596 non-null  float64
 5   waterfront   21596 non-null  int64  
 6   condition    21596 non-null  int64  
 7   grade        21596 non-null  int64  
dtypes: float64(3), int64(5)
memory usage: 1.5 MB
```

Modeling

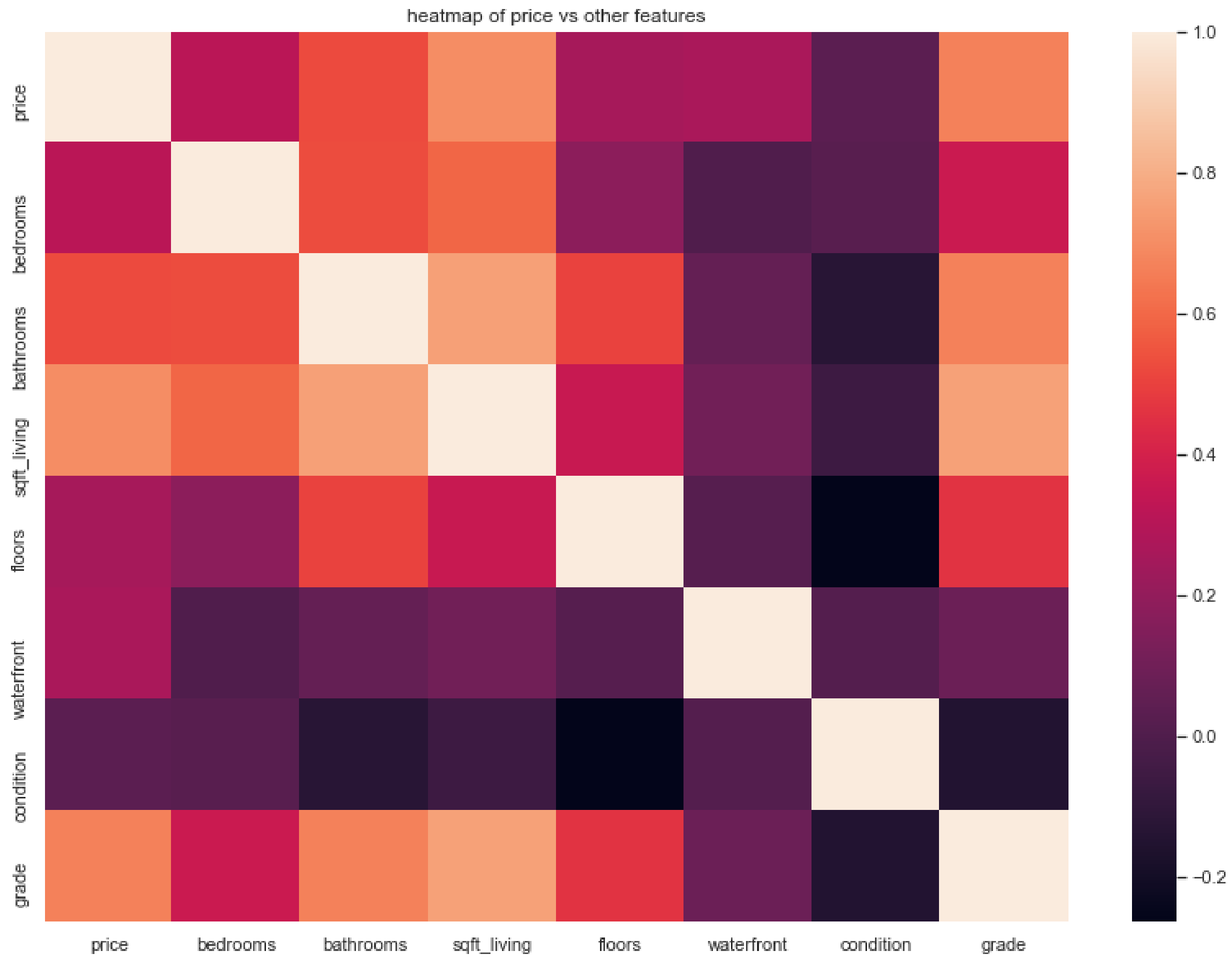
Correlation of the independent variables (features) with the dependant variable (target)

The code below shows the features and their correlation with price. The closer the value is to 1, the higher the relationship it has with price.

In [63]:
```
price_corr = kchd.corr()['price'].map(abs).sort_values(ascending=False)
price_corr
```

Out[63]:
```
price          1.000000
sqft_living    0.701929
grade          0.667964
bathrooms      0.525915
bedrooms       0.315961
waterfront     0.264308
floors         0.256820
condition      0.036031
Name: price, dtype: float64
```

In [64]:
```
## Heat map to visualize the strongest correlations
# Use the .heatmap method to depict the relationship visually
plt.figure(figsize=(14,10))
sns.heatmap(kchd.corr());
plt.title('heatmap of price vs other features')
plt.show()
```



Taking a look of the heat map, where the lighter the color means a stronger correlation. With price as the dependant variable, sqft_living, grade, and bathrooms have the strongest correlations.

In [65]:
```
kchd.head()
```

Out[65]:

	price	bedrooms	bathrooms	sqft_living	floors	waterfront	condition	grade
0	221900.0	3	1.00	1180	1.0	0	3	7
1	538000.0	3	2.25	2570	2.0	0	3	7
2	180000.0	2	1.00	770	1.0	0	3	6
3	604000.0	4	3.00	1960	1.0	0	5	7
4	510000.0	3	2.00	1680	1.0	0	3	8

Baseline Model

Since sqft_living has the strongest relationship with price, I will use sqft_living in my baseline model.

In [66]:
```
# Single Linear Regression
y = kchd['price']
X = sm.add_constant(kchd['sqft_living'])
X.head()
```

Out[66]:

	const	sqft_living
0	1.0	1180
1	1.0	2570
2	1.0	770
3	1.0	1960
4	1.0	1680

In [67]:
```
model = sm.OLS(y, X).fit()
model.summary()
```

Out[67]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.493
Model:	OLS	**Adj. R-squared:**	0.493
Method:	Least Squares	**F-statistic:**	2.097e+04
Date:	Thu, 25 Aug 2022	**Prob (F-statistic):**	0.00
Time:	13:10:46	**Log-Likelihood:**	-3.0005e+05
No. Observations:	21596	**AIC:**	6.001e+05
Df Residuals:	21594	**BIC:**	6.001e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-4.401e+04	4410.123	-9.980	0.000	-5.27e+04	-3.54e+04
sqft_living	280.8688	1.939	144.820	0.000	277.067	284.670

Omnibus:	14801.492	**Durbin-Watson:**	1.982
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	542642.481
Skew:	2.820	**Prob(JB):**	0.00
Kurtosis:	26.901	**Cond. No.**	5.63e+03

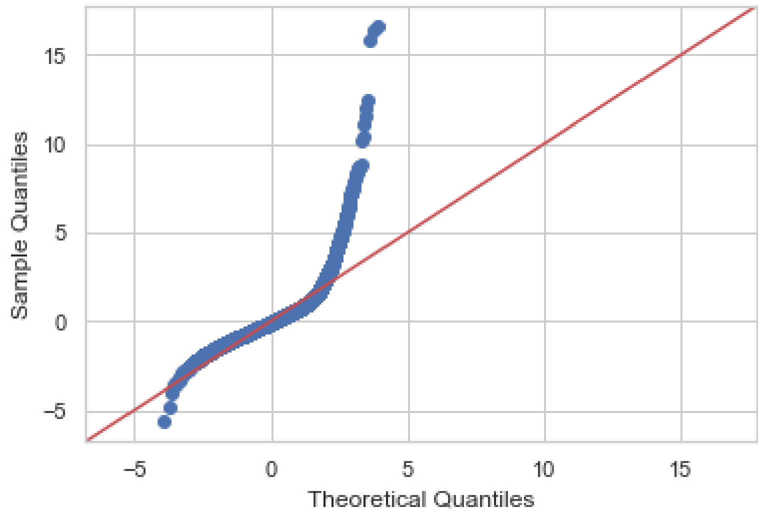
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.63e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

Model Results:

- An R-squared of 0.493. R-squared is the statistical measurement in a regression model that determines the proportion of variance in the dependant variable("price") that can be explained by the independant variable ("sqft_living"). 0.493 is pretty low goodness of fit.

Normality Check

In [68]:
```
# Check to see if model residuals follow a normal distribution using a QQ p
residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True)
```

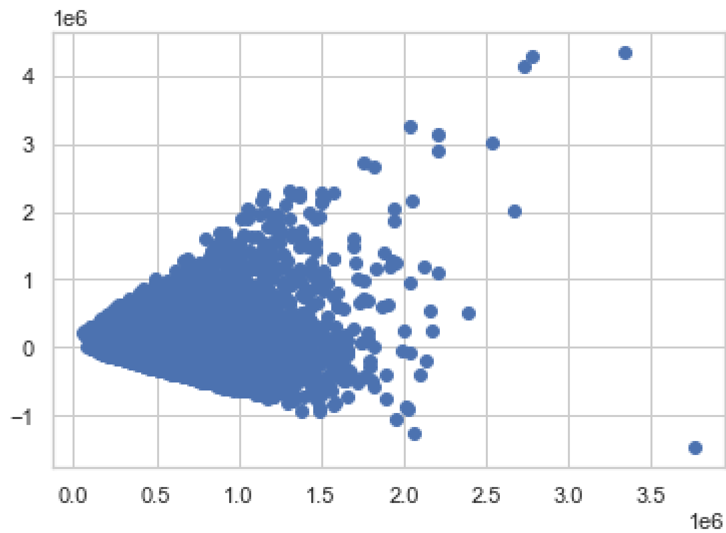


Results:

- From the QQ plot, you can see that there are some outliers skewing the visualization, however with a Durbin-Watson score of 1.981, the **model meets the normality assumption**. An acceptable range for a Durbin-Watson score is between 1.5 and 2.5

Homoskedasticity Check

In [69]:
```
## Check for Homoskedasticity
plt. scatter(x=model.fittedvalues, y=model.resid);
```



Results:

- Given the scatter plot above, this model **does not meet the homoscedasticity assumption.**

Model1

Log transform price

In [70]:
```
#Log transform price
kchd['price_log'] = np.log(kchd['price'])
```

In [71]:
```
kchd['price_log']
```

Out[71]:
```
0        12.309982
1        13.195614
2        12.100712
3        13.311329
4        13.142166
           ...
21592    12.793859
21593    12.899220
21594    12.904459
21595    12.899220
21596    12.691580
Name: price_log, Length: 21596, dtype: float64
```

In [72]:
```
kchd.head()
```

Out[72]:

	price	bedrooms	bathrooms	sqft_living	floors	waterfront	condition	grade	price_log
0	221900.0	3	1.00	1180	1.0	0	3	7	12.309982
1	538000.0	3	2.25	2570	2.0	0	3	7	13.195614
2	180000.0	2	1.00	770	1.0	0	3	6	12.100712
3	604000.0	4	3.00	1960	1.0	0	5	7	13.311329
4	510000.0	3	2.00	1680	1.0	0	3	8	13.142166

In [73]:
```
## add constant
y = kchd['price_log']
X1 = sm.add_constant(kchd['sqft_living'])
X.head()
```

Out[73]:

	const	sqft_living
0	1.0	1180
1	1.0	2570
2	1.0	770
3	1.0	1960
4	1.0	1680

In [74]:
```
model1 = sm.OLS(y, X1).fit()
model1.summary()
```

Out[74]:

OLS Regression Results

Dep. Variable:	price_log	**R-squared:**	0.483
Model:	OLS	**Adj. R-squared:**	0.483
Method:	Least Squares	**F-statistic:**	2.020e+04
Date:	Thu, 25 Aug 2022	**Prob (F-statistic):**	0.00
Time:	13:10:47	**Log-Likelihood:**	-9661.4
No. Observations:	21596	**AIC:**	1.933e+04
Df Residuals:	21594	**BIC:**	1.934e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	12.2187	0.006	1915.377	0.000	12.206	12.231
sqft_living	0.0004	2.81e-06	142.125	0.000	0.000	0.000

Omnibus:	3.543	**Durbin-Watson:**	1.977
Prob(Omnibus):	0.170	**Jarque-Bera (JB):**	3.564
Skew:	0.028	**Prob(JB):**	0.168
Kurtosis:	2.973	**Cond. No.**	5.63e+03

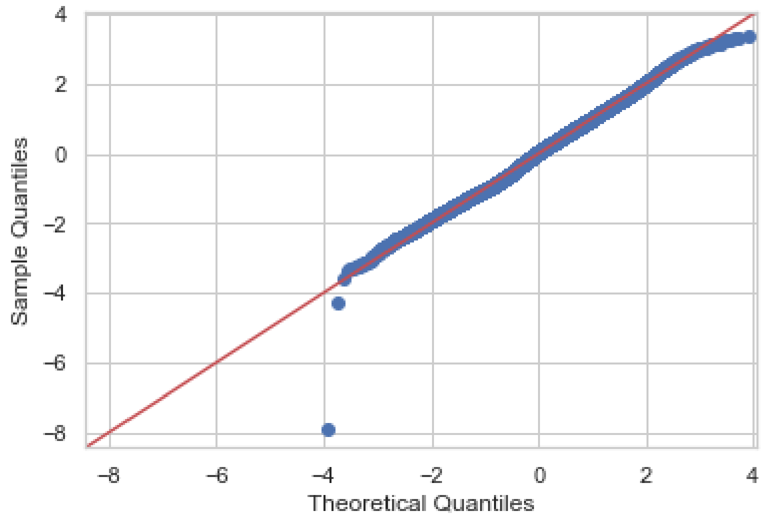
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.63e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

Model Results:

- The R-squared for our new model decreased slightly from 0.493 to 0.483
- Durbin-Watson score is in the acceptable range for normality.

Normality Check

In [75]:
```
# Check to see if model residuals follow a normal distribution using a QQ p
residuals1 = model1.resid
fig = sm.graphics.qqplot(residuals1, dist=stats.norm, line='45', fit=True)
```

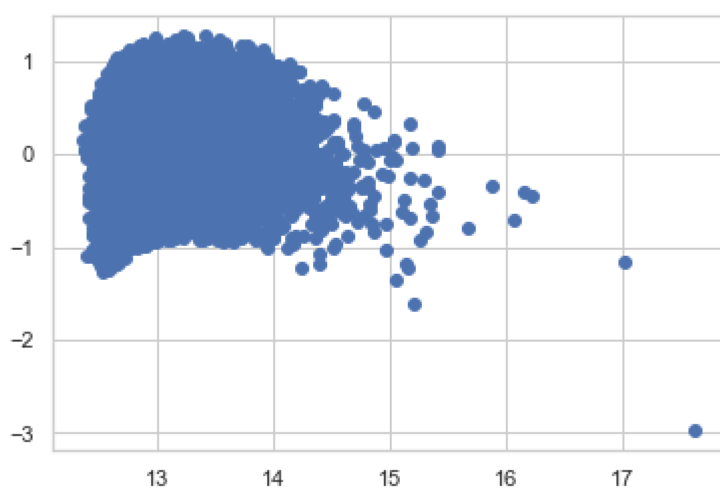


Results:

- From the QQ plot, the model is getting better and with a Durbin-Watson score of 1.977, the **model meets the normality assumption.**

Homoskedasticity Check

In [76]:
```
## Check for Homoskedasticity
plt. scatter(x=model1.fittedvalues, y=model1.resid);
```



Results:

- Given the scatter plot above, this model **does not meet the homoscedasticity assumption.**

Model 2

Add all features to OLS model

In [77]:
```
# Revisit the dataset
kchd.head()
```

Out[77]:

	price	bedrooms	bathrooms	sqft_living	floors	waterfront	condition	grade	price_log
0	221900.0	3	1.00	1180	1.0	0	3	7	12.309982
1	538000.0	3	2.25	2570	2.0	0	3	7	13.195614
2	180000.0	2	1.00	770	1.0	0	3	6	12.100712
3	604000.0	4	3.00	1960	1.0	0	5	7	13.311329
4	510000.0	3	2.00	1680	1.0	0	3	8	13.142166

In [78]:
```


#create predictors
X2 = kchd.drop(['price_log', 'price'], axis=1)
y = kchd['price_log']

#create model intercept
X_int = sm.add_constant(X2)

# fit model to data
model2 = sm.OLS(y, X_int).fit()
```

In [79]:
```
model2.summary()
```

Out[79]:

OLS Regression Results

Dep. Variable:	price_log	**R-squared:**	0.581
Model:	OLS	**Adj. R-squared:**	0.581
Method:	Least Squares	**F-statistic:**	4272.
Date:	Thu, 25 Aug 2022	**Prob (F-statistic):**	0.00
Time:	13:10:47	**Log-Likelihood:**	-7405.5
No. Observations:	21596	**AIC:**	1.483e+04
Df Residuals:	21588	**BIC:**	1.489e+04
Df Model:	7		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	10.8080	0.026	420.886	0.000	10.758	10.858
bedrooms	-0.0226	0.003	-6.827	0.000	-0.029	-0.016
bathrooms	-0.0030	0.005	-0.593	0.553	-0.013	0.007
sqft_living	0.0002	5.04e-06	43.777	0.000	0.000	0.000
floors	0.0181	0.005	3.429	0.001	0.008	0.028
waterfront	0.5944	0.029	20.797	0.000	0.538	0.650
condition	0.1041	0.004	27.958	0.000	0.097	0.111
grade	0.1930	0.003	58.436	0.000	0.186	0.199

Omnibus:	15.851	**Durbin-Watson:**	1.977
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	15.713
Skew:	0.058	**Prob(JB):**	0.000387
Kurtosis:	2.936	**Cond. No.**	2.80e+04

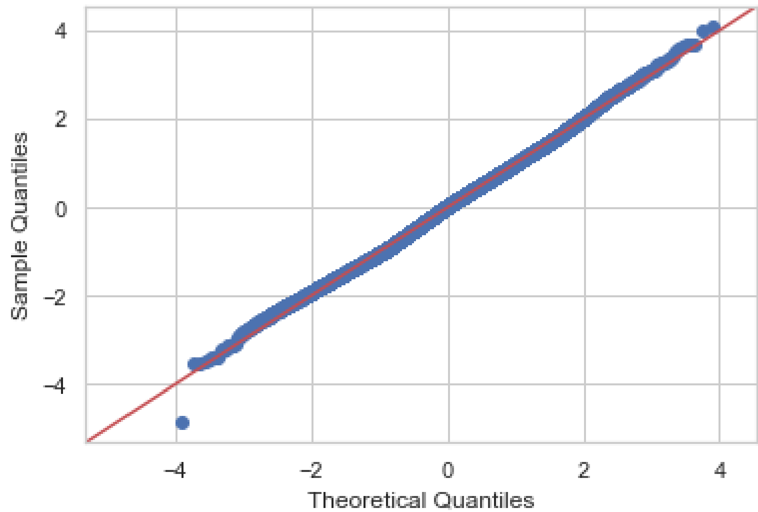
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.8e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

Model Results:

- The R-squared has improved from 0.483 to 0.581
- The Durbin-Watson score of 1.977, is still in the acceptable range for Normality.
- Bathrooms has a p-value greater than 0.05, proving to be statistically insignificant to price and should be removed from the model.

Normality Check

In [80]:
```
# Check to see if model residuals follow a normal distribution using a QQ p
residuals2 = model2.resid
fig = sm.graphics.qqplot(residuals2, dist=stats.norm, line='45', fit=True)
```

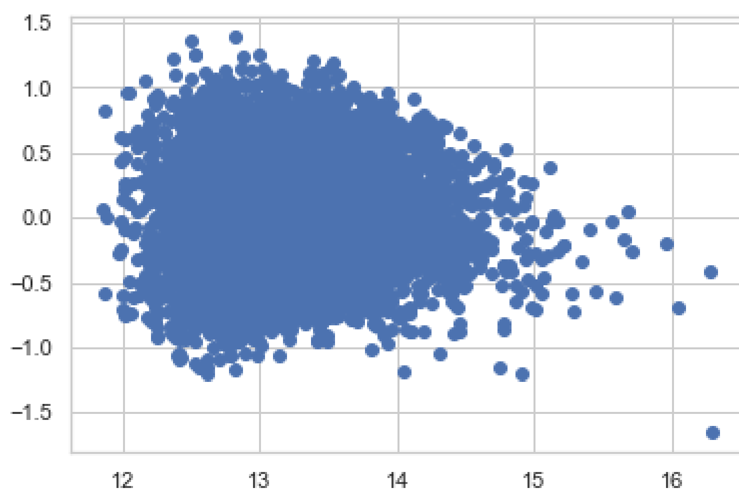


Results:

- From the QQ plot results, the model's normality assumption still holds with a Durbin-Watson score of 1.977. **model meets the normality assumption.**

Homoskedasticity Check

In [81]:
```
## Check for Homoskedasticity
plt. scatter(x=model2.fittedvalues, y=model2.resid);
```



Results:

- Given the scatter plot with its cone like shape, this model **does not meet the homoscedasticity assumption.**

Multicollinearity Check

In [82]:
```
# Writing a function to create VIF dictionary.
def create_vif_dictionary(X):
    """
    Parameters
    ----------
    X: Pandas dataframe of predictive variables only.
        Should have `.columns` and `.values` attributes.
    """

    vif_dct = {}

    # Loop through each row and set the variable name to the VIF.
    for i in range(len(X.columns)):
        # Calculate VIF
        vif = variance_inflation_factor(X.values, i)

        # Extract column name for dictionary key.
        v = X.columns[i]

        # Set value in dictionary.
        vif_dct[v] = vif

    return vif_dct
```

In [83]:
```
create_vif_dictionary(sm.add_constant(X2))
```

Out[83]:
```
{'const': 122.46066339242776,
 'bedrooms': 1.6683656381685992,
 'bathrooms': 2.8811136825903882,
 'sqft_living': 3.984057257596944,
 'floors': 1.5008510928466368,
 'waterfront': 1.0186462083218444,
 'condition': 1.0887169675013981,
 'grade': 2.7870959299699294}
```

Results:

- If the VIF is greater than 5, then the explanatory variable is highly collinear with the other explanatory variables.
- With model 2, **multicollinearity does not exist.**

Final Model

I have chose this iteration to be my final model because every predictor I have added to the model has statistical significance to price. Also, the increase in R-squared to 58% seems to be the highest it will reach with the current features in my model.

In [84]:
```
kchd.head()
```

Out[84]:

	price	bedrooms	bathrooms	sqft_living	floors	waterfront	condition	grade	price_log
0	221900.0	3	1.00	1180	1.0	0	3	7	12.309982
1	538000.0	3	2.25	2570	2.0	0	3	7	13.195614
2	180000.0	2	1.00	770	1.0	0	3	6	12.100712
3	604000.0	4	3.00	1960	1.0	0	5	7	13.311329
4	510000.0	3	2.00	1680	1.0	0	3	8	13.142166

In [85]:
```
# add predictors
X3 = kchd.drop(['price_log', 'price', 'bathrooms'], axis=1)
y = kchd['price_log']
```

In [86]:
```
X_int = sm.add_constant(X3)
model3 = sm.OLS(y, X_int).fit()
model3.summary()
```

Out[86]:

OLS Regression Results

Dep. Variable:	price_log	**R-squared:**	0.581
Model:	OLS	**Adj. R-squared:**	0.581
Method:	Least Squares	**F-statistic:**	4984.
Date:	Thu, 25 Aug 2022	**Prob (F-statistic):**	0.00
Time:	13:10:48	**Log-Likelihood:**	-7405.7
No. Observations:	21596	**AIC:**	1.483e+04
Df Residuals:	21589	**BIC:**	1.488e+04
Df Model:	6		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	10.8087	0.026	421.453	0.000	10.758	10.859
bedrooms	-0.0230	0.003	-7.087	0.000	-0.029	-0.017
sqft_living	0.0002	4.61e-06	47.675	0.000	0.000	0.000
floors	0.0170	0.005	3.426	0.001	0.007	0.027
waterfront	0.5944	0.029	20.799	0.000	0.538	0.650
condition	0.1041	0.004	27.990	0.000	0.097	0.111
grade	0.1927	0.003	58.892	0.000	0.186	0.199

Omnibus:	16.276	**Durbin-Watson:**	1.977
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	16.139
Skew:	0.059	**Prob(JB):**	0.000313
Kurtosis:	2.936	**Cond. No.**	2.80e+04

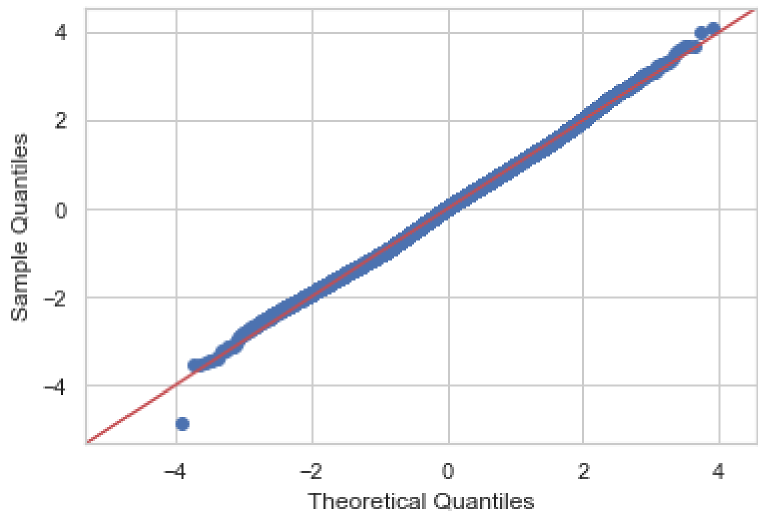
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.8e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

Model Results:

- After removing bathrooms, the R-squared remained the same at 0.581.
- Durbin-Watson score also remained the same at 1.977

Normality Check

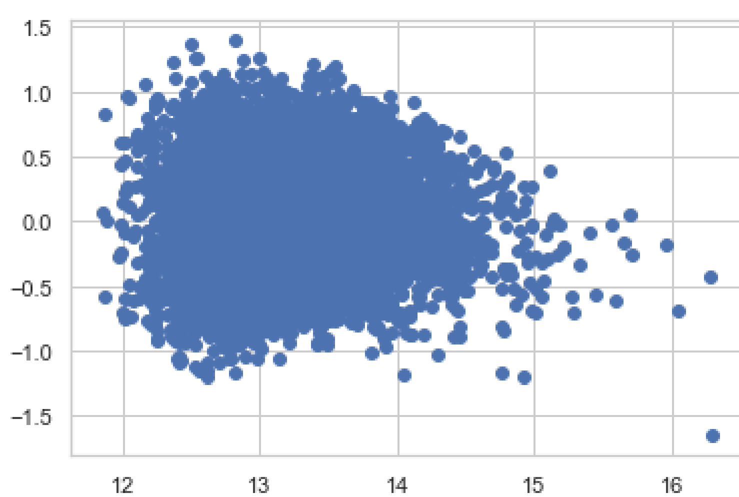
In [87]:
```
# Check to see if model residuals follow a normal distribution using a QQ p
residuals3 = model3.resid
fig = sm.graphics.qqplot(residuals3, dist=stats.norm, line='45', fit=True)
```



- From the QQ plot, the model remains the same in terms of the Durbin-Watson score of 1.977, and **model meets the normality assumption.**

Homoskedasticity Check

In [88]:
```
## Check for Homoskedasticity
plt. scatter(x=model3.fittedvalues, y=model3.resid);
```



Results:

- Given the scatter plot with its cone like shape, this model **does not meet the homoscedasticity assumption.**

Multicollinearity Check

In [89]:
```
# Check for mulitcollinearity with variance_inflation_factor
create_vif_dictionary(sm.add_constant(X3))
```

Out[89]:
```
{'const': 122.15187145883901,
 'bedrooms': 1.602146297579926,
 'sqft_living': 3.322380602933455,
 'floors': 1.3356284661252429,
 'waterfront': 1.0186365961674506,
 'condition': 1.0876932031556439,
 'grade': 2.736671884404993}
```

Results:

- **Multicollinearity does not exist**.

Recommendations

In the final model with all features excluding `bathrooms` , `yr_built` , and `sqft_lot` , our model's performance based on the adjusted R-squared improved from 0.493 to 0.581. Meaning that 41.9% of the variation of the price variable within the data is not explained by our model.

In the final model, all features included in the model have statistical significance relationship with price. All p-values are less than 0.05.

Coefficient Interpretations:

- For every bedroom added to a home, it decreases the price of the home by 2.3%
- For every square foot of living added to a home, it increases the price of a home by 0.02%
- For every floor added to a home, it increases the price of a home by 1.7%
- For every increase in condition to a home, it increases the price of a home by 10.4%
- For every increase in the grade of a home, it increases the price of a home by 19.3%

Conclusions for King County Realtors:

1. In order to maximize the price of a home, you should recommend to your clients that they should use great quality products when rennovating their home to increase the grade of their home to highest possible level.
2. If the seller is wanting to expand the size of their home, creating another floor is a great option to increase the price of their home. For a 500,000 dollar home, adding one floor would increase the price by 8,500.
3. Improving the condition of your home to a minimum, average condition, will increase your home's value by 10%.

Our model only explains 58 percent of the variation in sale price, so we must tread with caution with our predictions. Our final model does not hold true every assumption of linear regression, and violaties the rule of homoscedasticity.

Future Work:

- Add more features to our model to see the affects on adjusted R-squared.