1 Introduction

1.1 Business Problem Summary

Abodes by Abhineet, aka Ab^2, is a company that buys, renovates and resells homes. Ab^2 has seen an exponential take off over the past year and no longer has the staff to sort through all of the home listings in their area of operations - King's County in Seattle, Washington. With their extra capital they have decided to bring on a data scientist whose first responsibility is to create a tool that will identify sale listings worth investigating. Ab^2 has access to a large dataset of homes sold in the county in 2014 and 2015, and will provide it to the analyst.

1.2 Workflow Outline

This notebook will follow the standard machine learning workflow, OSEMIN:

1. Obtain the data by reading in our data file.
2. Scrub the data by dealing with null values, incorrect information, identifying and removing useless information.
3. Explore the data by looking at scatter plots, histograms, and pair plots.
4. Model data, starting with a baseline model and then improving upon it based on transformations.
5. Interpret results by discussing coefficients and features of the model.

2 OSEMIN

2.1 Obtain!

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

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

Out[2]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0

5 rows × 21 columns

2.1.1 Column Names and descriptions for Kings County Data Set

- **id** - unique identified for a house
- **date** - house was sold
- **price** - is prediction target
- **bedrooms** - of Bedrooms/House
- **bathrooms** - of bathrooms/bedrooms
- **sqft_living** - footage of the home
- **sqft_lot** - footage of the lot
- **floors** - floors (levels) in house
- **waterfront** - House which has a view to a waterfront
- **view** - Has been viewed (supposedly, but we think it actually means 'Has a view' and is rated 1-4, with a 0 being no view)
- **condition** - How good the condition is (Overall)
- **grade** - overall grade given to the housing unit, based on King County grading system
- **sqft_above** - square footage of house apart from basement
- **sqft_basement** - square footage of the basement
- **yr_built** - Built Year
- **yr_renovated** - Year when house was renovated
- **zipcode** - zip
- **lat** - Latitude coordinate
- **long** - Longitude coordinate
- **sqft_living15** - The square footage of interior housing living space for the nearest 15 neighbors
- **sqft_lot15** - The square footage of the land lots of the nearest 15 neighbors

In [3]:
```
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
```

Observations:

1. `date` is type:object so may want to recast as datetime
2. `waterfront`, `view`, `yr_renovated` are missing values
3. `sqft_basement` has a datatype of object
4. `id` will be useful for checking for duplicates, but otherwise can probably remove
5. Need to check relationship between `sqft_living`, `sqft_above` and `sqft_basement`
6. `grade` and `condition` columns seem to mean basically the same thing

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

Out[4]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	21597
mean	4.580474e+09	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	1
std	2.876736e+09	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	0
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1
25%	2.123049e+09	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	2
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3

There will be more to say about this in the 'Explore' section, but for now we should check out those houses with less than one bathroom, and the one with 33 bedrooms.

2.2 Scrub!

2.2.1 Missing Values

We observed that `waterfront` , `view` , and `yr_renovated` are missing values. Let's deal with those first.

In [5]:
```
df.waterfront.value_counts()
```

Out[5]:
```
0.0    19075
1.0      146
Name: waterfront, dtype: int64
```

Most homes are not waterfront property. It seems likely that if it was waterfront property, it would not be missing a value for this field. Therefore, I will replace all nulls in this column with zeros.

In [6]:
```
df['waterfront'] = df.waterfront.fillna(0)
```

In [7]:
```
df.waterfront.value_counts()
```

Out[7]:
```
0.0    21451
1.0      146
Name: waterfront, dtype: int64
```

In [8]:
```
df.view.value_counts()
```

Out[8]:
```
0.0    19422
2.0      957
3.0      508
1.0      330
4.0      317
Name: view, dtype: int64
```

Hmmm.. looks like a similar issue with `view` . Again, since such a large majority are marked as not having a view, and if they did have a view they would probably have included that information, and compared to waterfront their are very few NaNs (maybe 60) let's just fill with zeros.

In [9]:
```
df['view'] = df.view.fillna(0)
```

In [10]:
```
df.view.value_counts()
```

Out[10]:
```
0.0    19485
2.0      957
3.0      508
1.0      330
4.0      317
Name: view, dtype: int64
```

In [11]:
```
df.yr_renovated.value_counts()
```

Out[11]:
```
0.0       17011
2014.0       73
2003.0       31
2013.0       31
2007.0       30
             ...
1946.0        1
1959.0        1
1971.0        1
1951.0        1
1954.0        1
Name: yr_renovated, Length: 70, dtype: int64
```

In [12]:
```
df.yr_renovated.isna().sum()
```

Out[12]:
```
3842
```

So this column seems to have the year if it has been renovated and a zero otherwise. This one also has a pretty significant number of values missing out, nearly 20 percent at around 3800 missing records. While it does feel risky to fill these NaNs with zeros as well, it is justified by similar logic to the categories above, and in relation to our business case. Seeing as Abhineet's business runs on renovating and reselling houses at a profit, we have to assume that houses that have been renovated sold for more than they otherwise would have at least most of the time. In that way, looking at recently renovated and sold houses could actually negatively impact the model, since we assume these datapoints contribute to rightward skew relative to the rest of the dataset. Similarly to view and waterfront, it seems likely that if the home has been renovated recently, say

within the last ten years, we would have that information. We would not want to buy a home that has been remodeled so recently anyway. Therefore, either it has been renovated but we don't know since it has been a while since it has been renovated, and thus it is the same as not having been renovated, since we will need to make modern changes to appliances and decor.

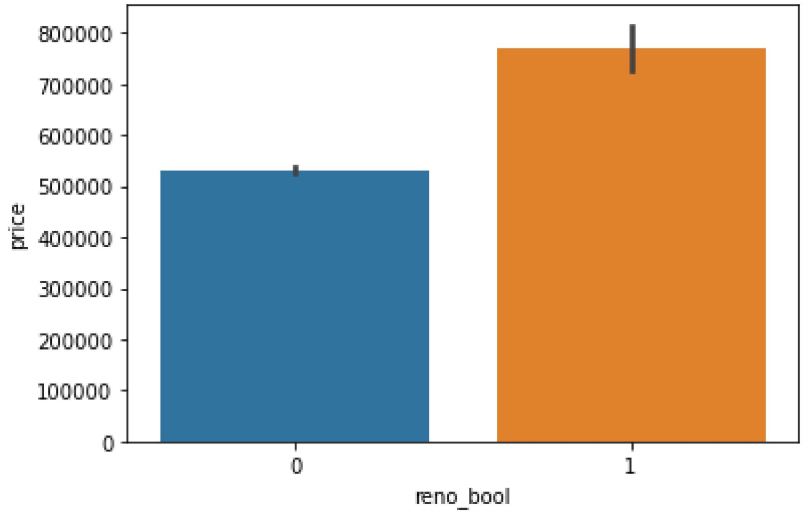
Let's look at a couple things to check these assumptions. Let's compare selling prices from for homes renovated or otherwise.

In [13]:
```
# need to first deal with date column being type object

df['date'] = pd.to_datetime(df['date'])

# create a boolean column for renovated or not, set NaNs to not renovated

df['reno_bool'] = df.yr_renovated.apply(lambda x: 0 if x == False else ( 0 if pd.
sns.barplot(x='reno_bool', y='price', data=df);
```



In [14]:
```
df.reno_bool.value_counts()
```

Out[14]:
```
0    20853
1      744
Name: reno_bool, dtype: int64
```

So, this is not perfect, because we don't know what the homes might have sold for otherwise and there are very few renovated homes sold, etc etc, but they are definitely priced higher than homes that have not been renovated. Finally, let's just look at homes that don't have information on `yr_renovated` to see if anything sticks out.

In [15]:
```
missing_reno = df[pd.isna(df.yr_renovated) == True]
```

In [16]:
```
missing_reno.head()
```

Out[16]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	vi
2	5631500400	2015-02-25	180000.0	2	1.00	770	10000	1.0	0.0	
12	114101516	2014-05-28	310000.0	3	1.00	1430	19901	1.5	0.0	
23	8091400200	2014-05-16	252700.0	2	1.50	1070	9643	1.0	0.0	
26	1794500383	2014-06-26	937000.0	3	1.75	2450	2691	2.0	0.0	
28	5101402488	2014-06-24	438000.0	3	1.75	1520	6380	1.0	0.0	

5 rows × 22 columns

In [17]:
```
missing_reno.describe()
```

Out[17]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	fl
count	3.842000e+03	3.842000e+03	3842.000000	3842.000000	3842.000000	3842.000000	3842.00
mean	4.564007e+09	5.386170e+05	3.348777	2.094157	2061.979178	14254.275377	1.48
std	2.878860e+09	3.583446e+05	0.895277	0.775021	920.365509	38026.156942	0.54
min	3.600072e+06	7.800000e+04	1.000000	0.500000	500.000000	635.000000	1.00
25%	2.099175e+09	3.229760e+05	3.000000	1.500000	1410.000000	5040.000000	1.00
50%	3.886954e+09	4.500000e+05	3.000000	2.250000	1900.000000	7665.000000	1.00
75%	7.300400e+09	6.433750e+05	4.000000	2.500000	2530.000000	10706.000000	2.00
max	9.900000e+09	5.570000e+06	8.000000	6.750000	9200.000000	881654.000000	3.50

While there are some minor differences, the quartiles and other descriptors are very similar to the dataset as a whole. There doesn't seem to be any notable differences, so since the ones with missing information seem to be more or less similar to the bulk of the main dataset, we will make them similar to the main dataset. With a large majority of the main dataset being homes that have never been renovated, will we thus fill the missing values with a zero for 'never renovated' as suggested earlier.

In [18]:
```
df['yr_renovated'] = df.yr_renovated.fillna(0)
```

In [19]:
```
df.yr_renovated.value_counts()
```

Out[19]:
```
0.0       20853
2014.0       73
2003.0       31
2013.0       31
2007.0       30
            ...
1946.0        1
1959.0        1
1971.0        1
1951.0        1
1954.0        1
Name: yr_renovated, Length: 70, dtype: int64
```

Keeping in mind our business problem, we should remember at the end of the scrubbing process that we will have to think about whether or not to simply drop rows for homes that have been renovated, say, after 2010 (within the last ten years and so already modernized), and similarly for homes built after 2010 or perhaps even earlier.

2.2.2 Dealing with similar columns

We saw that `grade` and `condition` seem similar, as well as there is likely some relationship between `sqft_living`, `sqft_above` and `sqft_basement`. `sqft_basement` is also an object data type so let's fix that.

In [20]:
```
df.sqft_basement.value_counts()
```

Out[20]:
```
0.0       12826
?           454
600.0       217
500.0       209
700.0       208
            ...
2196.0        1
172.0         1
518.0         1
2610.0        1
2850.0        1
Name: sqft_basement, Length: 304, dtype: int64
```

Hmm, so really it has some missing values and all are cast as strs since they used ? to denote basement or not. Again, here I am just going to quickly replace the '?' with 0 for no basement since it seems like it would be noted if it was there.

In [21]:
```
df['sqft_basement'] = df.sqft_basement.replace(to_replace = '?',value = '0.0')
```

In [22]:
```
df.sqft_basement.value_counts()
```

Out[22]:
```
0.0       13280
600.0       217
500.0       209
700.0       208
800.0       201
           ...
2850.0        1
172.0         1
518.0         1
2610.0        1
2196.0        1
Name: sqft_basement, Length: 303, dtype: int64
```

In [23]:
```
df['sqft_basement'] = df.sqft_basement.astype(float)
```

In [24]:
```
df.sqft_basement.value_counts()
```

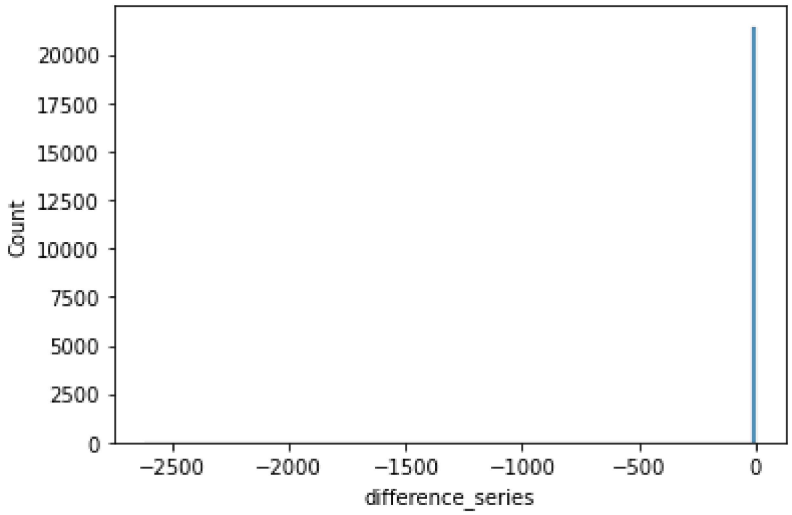
Out[24]:
```
0.0       13280
600.0       217
500.0       209
700.0       208
800.0       201
           ...
915.0         1
295.0         1
1281.0        1
2130.0        1
906.0         1
Name: sqft_basement, Length: 303, dtype: int64
```

In [25]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21597 non-null  int64
 1   date           21597 non-null  datetime64[ns]
 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     21597 non-null  float64
 9   view           21597 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   21597 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
 21  reno_bool      21597 non-null  int64
dtypes: datetime64[ns](1), float64(9), int64(12)
memory usage: 3.6 MB
```

Sweet, that is fixed. Let's quickly investigate whether we can get rid of any of these columns. Let's plot a distribution of the difference between `sqft_above` and the quantity `sqft_living - sqft_basement`.

In [26]:
```
df['difference_series'] = pd.Series( df.sqft_above - (df.sqft_living - df.sqft_ba
sns.histplot(data = df, x = 'difference_series', binwidth = 10);
```



Okay, so this is telling me, except for a couple outliers potentially that are impossible to see on the graph and are possibly because of one or two mistakes in filling in zeros for '?' as we did steps ago, that `sqft_above` is just equal to the square footage of the home without the basement. I would guess that, since `sqft_living` is an aggregate of the other two columns, we can drop that column and not lose any information about the data. By keeping above and basement, we indirectly retain what percent of livable sq footage is above and below, whereas if we kept only `sqft_living` we would lose that. Before dropping that column, let's see how many there are with a difference greater than 10 sqft.

In [27]:
```
df[df['difference_series'] <= -10].difference_series.count()
```

Out[27]:
```
170
```

Okay, so for most of them the supposition seems to be true. For the ones that it isn't, there is few enough of them that I feel fine removing those rows. So let's first remove the rows then drop the sqft_living column altogether.

In [28]:
```
df2 = df[df['difference_series'] >= -10]
```

In [29]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21427 entries, 0 to 21596
Data columns (total 23 columns):
 #   Column             Non-Null Count  Dtype         
---  ------             --------------  -----         
 0   id                 21427 non-null  int64         
 1   date               21427 non-null  datetime64[ns]
 2   price              21427 non-null  float64       
 3   bedrooms           21427 non-null  int64         
 4   bathrooms          21427 non-null  float64       
 5   sqft_living        21427 non-null  int64         
 6   sqft_lot           21427 non-null  int64         
 7   floors             21427 non-null  float64       
 8   waterfront         21427 non-null  float64       
 9   view               21427 non-null  float64       
 10  condition          21427 non-null  int64         
 11  grade              21427 non-null  int64         
 12  sqft_above         21427 non-null  int64         
 13  sqft_basement      21427 non-null  float64       
 14  yr_built           21427 non-null  int64         
 15  yr_renovated       21427 non-null  float64       
 16  zipcode            21427 non-null  int64         
 17  lat                21427 non-null  float64       
 18  long               21427 non-null  float64       
 19  sqft_living15      21427 non-null  int64         
 20  sqft_lot15         21427 non-null  int64         
 21  reno_bool          21427 non-null  int64         
 22  difference_series  21427 non-null  float64       
dtypes: datetime64[ns](1), float64(10), int64(12)
memory usage: 3.9 MB
```

Cool, so now we can drop `difference_series` and `sqft_living`.

In [30]:
```
df2 = df2.drop(columns = ['difference_series','sqft_living']).reset_index()
```

In [31]:
```
df2 = df2.drop(columns = 'index')
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21427 entries, 0 to 21426
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   id             21427 non-null  int64         
 1   date           21427 non-null  datetime64[ns]
 2   price          21427 non-null  float64       
 3   bedrooms       21427 non-null  int64         
 4   bathrooms      21427 non-null  float64       
 5   sqft_lot       21427 non-null  int64         
 6   floors         21427 non-null  float64       
 7   waterfront     21427 non-null  float64       
 8   view           21427 non-null  float64       
 9   condition      21427 non-null  int64         
 10  grade          21427 non-null  int64         
 11  sqft_above     21427 non-null  int64         
 12  sqft_basement  21427 non-null  float64       
 13  yr_built       21427 non-null  int64         
 14  yr_renovated   21427 non-null  float64       
 15  zipcode        21427 non-null  int64         
 16  lat            21427 non-null  float64       
 17  long           21427 non-null  float64       
 18  sqft_living15  21427 non-null  int64         
 19  sqft_lot15     21427 non-null  int64         
 20  reno_bool      21427 non-null  int64         
dtypes: datetime64[ns](1), float64(9), int64(11)
memory usage: 3.4 MB
```

Okay, so now between the `grade` and `condition` columns, the `condition` seems user defined while `grade` is based on a King County system and may be evaluated by an appraiser or some such, though we don't really know for sure. At any rate, `grade` is better since it has more tiers, is associated with a rubric of some kind, and does seem to have at least somewhat of a linear relationship with price, so let's keep that column and drop `condition`.

In [32]:
```
df2 = df2.drop(columns = 'condition')
```

2.2.3 Checking on duplicate entries

Since we have an `id` column with unique ID numbers for each house, let's check for duplicates.

In [33]:
```
df2.duplicated("id").sum()
```

Out[33]:
```
176
```

In [34]:
```
duplicates = df2[df2.duplicated(subset = "id", keep = False) == True]
```

In [35]:
```
duplicates.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 351 entries, 93 to 21395
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   id             351 non-null    int64         
 1   date           351 non-null    datetime64[ns]
 2   price          351 non-null    float64       
 3   bedrooms       351 non-null    int64         
 4   bathrooms      351 non-null    float64       
 5   sqft_lot       351 non-null    int64         
 6   floors         351 non-null    float64       
 7   waterfront     351 non-null    float64       
 8   view           351 non-null    float64       
 9   grade          351 non-null    int64         
 10  sqft_above     351 non-null    int64         
 11  sqft_basement  351 non-null    float64       
 12  yr_built       351 non-null    int64         
 13  yr_renovated   351 non-null    float64       
 14  zipcode        351 non-null    int64         
 15  lat            351 non-null    float64       
 16  long           351 non-null    float64       
 17  sqft_living15  351 non-null    int64         
 18  sqft_lot15     351 non-null    int64         
 19  reno_bool      351 non-null    int64         
dtypes: datetime64[ns](1), float64(9), int64(10)
memory usage: 57.6 KB
```

In [36]:
```
duplicates.describe()
```

Out[36]:

	id	price	bedrooms	bathrooms	sqft_lot	floors	waterfront
count	3.510000e+02	3.510000e+02	351.00000	351.000000	351.000000	351.000000	351.0
mean	4.504499e+09	4.192978e+05	3.28490	1.807692	11694.336182	1.264957	0.0
std	2.854431e+09	2.953285e+05	1.03027	0.735613	22856.037950	0.433784	0.0
min	1.000102e+06	8.200000e+04	1.00000	0.750000	1092.000000	1.000000	0.0
25%	1.974300e+09	2.339500e+05	3.00000	1.000000	5573.000000	1.000000	0.0
50%	4.031001e+09	3.400000e+05	3.00000	1.750000	8043.000000	1.000000	0.0
75%	7.398600e+09	5.250000e+05	4.00000	2.250000	10140.000000	1.500000	0.0
max	9.834201e+09	1.940000e+06	6.00000	4.500000	224442.000000	3.000000	0.0

In [37]:
```
duplicates.head(6)
```

Out[37]:

	id	date	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	view	grade
93	6021501535	2014-07-25	430000.0	3	1.50	5000	1.0	0.0	0.0	8
94	6021501535	2014-12-23	700000.0	3	1.50	5000	1.0	0.0	0.0	8
310	4139480200	2014-06-18	1380000.0	4	3.25	12103	1.0	0.0	3.0	11
311	4139480200	2014-12-09	1400000.0	4	3.25	12103	1.0	0.0	3.0	11
321	7520000520	2014-09-05	232000.0	2	1.00	12092	1.0	0.0	0.0	6
322	7520000520	2015-03-11	240500.0	2	1.00	12092	1.0	0.0	0.0	6

In [38]:
```
duplicates.reno_bool.value_counts()
```

Out[38]:
```
0    343
1      8
Name: reno_bool, dtype: int64
```

Hmm.. based on this, the houses weren't necessarily bought and then resold because they were renovated. We would have to backtrack, however, to be sure that these were houses we didn't fill in zeros for when we were filling in missing values for `yr_renovated`. It seems that, for a house to be bought and then sold again, within the same year (or two, since we have some 2015 data) it was either bought and renovated and then resold or it was a mistake and ended up in the system twice somehow. Let's quickly look at the couple that were definitely renovated.

In [39]:
```
duplicates[duplicates.reno_bool == True].id
```

Out[39]:
```
321     7520000520
322     7520000520
709     8820903380
710     8820903380
3917    1825069031
3918    1825069031
8353    1721801010
8354    1721801010
Name: id, dtype: int64
```

In [40]:
```
for idx, duplicate_id in enumerate(duplicates[duplicates.reno_bool == True].id):
    if idx % 2:
        print('====================================')
        print(df2[df2['id'] == duplicate_id])
        print('====================================')
```

```
====================================
            id       date     price  bedrooms  bathrooms  sqft_lot  floors  \
321  7520000520 2014-09-05  232000.0         2        1.0     12092     1.0
322  7520000520 2015-03-11  240500.0         2        1.0     12092     1.0

     waterfront  view  grade  sqft_above  sqft_basement  yr_built  \
321         0.0   0.0      6         960          280.0      1922
322         0.0   0.0      6         960          280.0      1922

     yr_renovated  zipcode      lat     long  sqft_living15  sqft_lot15  \
321        1984.0    98146  47.4957 -122.352           1820        7460
322        1984.0    98146  47.4957 -122.352           1820        7460

     reno_bool
321          1
322          1
====================================
====================================
            id       date     price  bedrooms  bathrooms  sqft_lot  floors  \
709  8820903380 2014-07-28  452000.0         6       2.25     13579     2.0
710  8820903380 2015-01-02  730000.0         6       2.25     13579     2.0

     waterfront  view  grade  sqft_above  sqft_basement  yr_built  \
709         0.0   0.0      7        2660            0.0      1937
710         0.0   0.0      7        2660            0.0      1937

     yr_renovated  zipcode      lat     long  sqft_living15  sqft_lot15  \
709        1990.0    98125  47.7142 -122.286           1120        8242
710        1990.0    98125  47.7142 -122.286           1120        8242

     reno_bool
709          1
710          1
====================================
====================================
             id       date     price  bedrooms  bathrooms  sqft_lot  floors  \
3917  1825069031 2014-08-14  550000.0         4       1.75      8447     2.0
3918  1825069031 2014-10-16  550000.0         4       1.75      8447     2.0

      waterfront  view  grade  sqft_above  sqft_basement  yr_built  \
3917         0.0   3.0      8        2060          350.0      1936
3918         0.0   3.0      8        2060          350.0      1936

      yr_renovated  zipcode      lat     long  sqft_living15  sqft_lot15  \
3917        1980.0    98074  47.6499 -122.088           2520       14789
3918        1980.0    98074  47.6499 -122.088           2520       14789

      reno_bool
3917          1
3918          1
====================================
```

```
====================================
             id       date     price  bedrooms  bathrooms  sqft_lot  floors  \
8353  1721801010 2014-09-03  225000.0         3        1.0      6120     1.0
8354  1721801010 2015-04-24  302100.0         3        1.0      6120     1.0

      waterfront  view  grade  sqft_above  sqft_basement  yr_built  \
8353         0.0   0.0      6        1790            0.0      1937
8354         0.0   0.0      6        1790            0.0      1937

      yr_renovated  zipcode     lat     long  sqft_living15  sqft_lot15  \
8353        1964.0    98146  47.508 -122.337            830        6120
8354        1964.0    98146  47.508 -122.337            830        6120

      reno_bool
8353          1
8354          1
====================================
```

Okay, at least for those, they may actually be the same house that sold twice for different prices. Could it be that they were renovated a second time and this isn't recorded? Anyway, let's see if it looks the same for a few of the ones that didn't say they were renovated. One of them did sell for exactly the same price, but some time apart. It is not clear why the others sold for more or less; it could be that someone realized they could flip the house for more even without renovating it again.

In [41]:
```
for idx,duplicate_id in enumerate(duplicates[duplicates.reno_bool == False].id):
    if idx > 11:
        break
    elif idx % 2:
        print('====================================')
        print(df2[df2['id'] == duplicate_id])
        print('====================================')
```

```
====================================
            id       date     price  bedrooms  bathrooms  sqft_lot  floors  \
93  6021501535 2014-07-25  430000.0         3        1.5      5000     1.0
94  6021501535 2014-12-23  700000.0         3        1.5      5000     1.0

    waterfront  view  grade  sqft_above  sqft_basement  yr_built  \
93         0.0   0.0      8        1290          290.0      1939
94         0.0   0.0      8        1290          290.0      1939

    yr_renovated  zipcode     lat     long  sqft_living15  sqft_lot15  \
93           0.0    98117  47.687 -122.386           1570        4500
94           0.0    98117  47.687 -122.386           1570        4500

    reno_bool
93          0
94          0
====================================
====================================
             id       date      price  bedrooms  bathrooms  sqft_lot  floors  \
310  4139480200 2014-06-18  1380000.0         4       3.25     12103     1.0
311  4139480200 2014-12-09  1400000.0         4       3.25     12103     1.0

     waterfront  view  grade  sqft_above  sqft_basement  yr_built  \
310         0.0   3.0     11        2690         1600.0      1997
311         0.0   3.0     11        2690         1600.0      1997

     yr_renovated  zipcode      lat     long  sqft_living15  sqft_lot15  \
310           0.0    98006  47.5503 -122.102           3860       11244
311           0.0    98006  47.5503 -122.102           3860       11244

     reno_bool
310          0
311          0
====================================
====================================
             id       date     price  bedrooms  bathrooms  sqft_lot  floors  \
342  3969300030 2014-07-23  165000.0         4        1.0      7134     1.0
343  3969300030 2014-12-29  239900.0         4        1.0      7134     1.0

     waterfront  view  grade  sqft_above  sqft_basement  yr_built  \
342         0.0   0.0      6        1000            0.0      1943
343         0.0   0.0      6        1000            0.0      1943

     yr_renovated  zipcode      lat    long  sqft_living15  sqft_lot15  \
342           0.0    98178  47.4897 -122.24           1020        7138
343           0.0    98178  47.4897 -122.24           1020        7138

     reno_bool
342          0
```

```
343          0
====================================
====================================
             id       date     price  bedrooms  bathrooms  sqft_lot  floors  \
368  2231500030 2014-10-01  315000.0         4       2.25     10754     1.0
369  2231500030 2015-03-24  530000.0         4       2.25     10754     1.0

     waterfront  view  grade  sqft_above  sqft_basement  yr_built  \
368         0.0   0.0      7        1100         1080.0      1954
369         0.0   0.0      7        1100         1080.0      1954

     yr_renovated  zipcode      lat     long  sqft_living15  sqft_lot15  \
368           0.0    98133  47.7711 -122.341           1810        6929
369           0.0    98133  47.7711 -122.341           1810        6929

     reno_bool
368          0
369          0
====================================
====================================
            id       date     price  bedrooms  bathrooms  sqft_lot  floors  \
814  726049190 2014-10-02  287500.0         3        1.0      7200     1.0
815  726049190 2015-02-18  431000.0         3        1.0      7200     1.0

     waterfront  view  grade  sqft_above  sqft_basement  yr_built  \
814         0.0   0.0      7        1130          680.0      1954
815         0.0   0.0      7        1130          680.0      1954

     yr_renovated  zipcode      lat     long  sqft_living15  sqft_lot15  \
814           0.0    98133  47.7493 -122.351           1810        8100
815           0.0    98133  47.7493 -122.351           1810        8100

     reno_bool
814          0
815          0
====================================
====================================
             id       date     price  bedrooms  bathrooms  sqft_lot  floors  \
827  8682262400 2014-07-18  430000.0         2       1.75      4003     1.0
828  8682262400 2015-05-13  419950.0         2       1.75      4003     1.0

     waterfront  view  grade  sqft_above  sqft_basement  yr_built  \
827         0.0   0.0      8        1350            0.0      2004
828         0.0   0.0      8        1350            0.0      2004

     yr_renovated  zipcode      lat     long  sqft_living15  sqft_lot15  \
827           0.0    98053  47.7176 -122.033           1350        4479
828           0.0    98053  47.7176 -122.033           1350        4479

     reno_bool
827          0
828          0
====================================
```

Hmm... It's hard to know what to do with these. Some of them sold within the same year for similar prices, some of them sold in the same year for radically different prices, and one of them even sold

for less, but all the other information is always exactly the same. They will both impact a linear fit more or less equally and seem like legitimate data and there is not that many of them so I'm just going to leave them in. We should no longer have any need for the `id` column so I'm just going to drop it.

In [42]:
```
df2 = df2.drop(columns = 'id')
```

In [43]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21427 entries, 0 to 21426
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   date           21427 non-null  datetime64[ns]
 1   price          21427 non-null  float64       
 2   bedrooms       21427 non-null  int64         
 3   bathrooms      21427 non-null  float64       
 4   sqft_lot       21427 non-null  int64         
 5   floors         21427 non-null  float64       
 6   waterfront     21427 non-null  float64       
 7   view           21427 non-null  float64       
 8   grade          21427 non-null  int64         
 9   sqft_above     21427 non-null  int64         
 10  sqft_basement  21427 non-null  float64       
 11  yr_built       21427 non-null  int64         
 12  yr_renovated   21427 non-null  float64       
 13  zipcode        21427 non-null  int64         
 14  lat            21427 non-null  float64       
 15  long           21427 non-null  float64       
 16  sqft_living15  21427 non-null  int64         
 17  sqft_lot15     21427 non-null  int64         
 18  reno_bool      21427 non-null  int64         
dtypes: datetime64[ns](1), float64(9), int64(9)
memory usage: 3.1 MB
```

2.2.4 Checking on weird values

In [44]:
```
no_baths = df2[df2.bathrooms < 1]
no_baths.head()
```

Out[44]:

	date	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	view	grade	sqft_above
206	2014-11-23	180250.0	2	0.75	9600	1.0	0.0	0.0	6	900
262	2014-10-27	369900.0	1	0.75	10079	1.0	1.0	4.0	5	760
347	2014-06-04	299000.0	1	0.75	12120	1.0	0.0	0.0	4	560
461	2014-05-23	80000.0	1	0.75	5050	1.0	0.0	0.0	4	430
564	2014-12-18	405000.0	2	0.75	15029	1.0	0.0	0.0	6	870

In [45]:
```
no_baths.bathrooms.value_counts()
```

Out[45]:
```
0.75    71
0.50     4
Name: bathrooms, dtype: int64
```

The internet tells me that three quarters of a bathroom is like a sink and toilet and small standup shower, no tub. These could be replaced with a full bathroom if we like, probably. Half a bathroom, however, would probably mean no installed shower at all. There are only 4, so it might not be worth doing anything, and maybe they are like little vacation cottages or something, but it is odd so let's look at them.

In [46]:
```
half_baths = df2[df2.bathrooms < .75]
half_baths.head()
```

Out[46]:

	date	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	view	grade	sqft_abov
2241	2014-10-03	273000.0	2	0.5	7750	1.0	0.0	0.0	6	59
10324	2015-01-14	109000.0	2	0.5	6900	1.0	0.0	0.0	5	58
11564	2014-08-14	255000.0	1	0.5	1642	1.0	0.0	0.0	6	50
11929	2014-12-12	312500.0	4	0.5	5570	2.0	0.0	0.0	8	230

Only half a bathroom seems very strange. They are on the lower end of the price spectrum, and the first three are very small and old, but it would be strange for the last one to actually have four bedrooms, 2300 sqft of living space and only half a bathroom. We'll leave the other three as

outliers for now but get rid of the last one.

In [47]:
```
df3 = df2.drop(11929)
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21426 entries, 0 to 21426
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   date           21426 non-null  datetime64[ns]
 1   price          21426 non-null  float64       
 2   bedrooms       21426 non-null  int64         
 3   bathrooms      21426 non-null  float64       
 4   sqft_lot       21426 non-null  int64         
 5   floors         21426 non-null  float64       
 6   waterfront     21426 non-null  float64       
 7   view           21426 non-null  float64       
 8   grade          21426 non-null  int64         
 9   sqft_above     21426 non-null  int64         
 10  sqft_basement  21426 non-null  float64       
 11  yr_built       21426 non-null  int64         
 12  yr_renovated   21426 non-null  float64       
 13  zipcode        21426 non-null  int64         
 14  lat            21426 non-null  float64       
 15  long           21426 non-null  float64       
 16  sqft_living15  21426 non-null  int64         
 17  sqft_lot15     21426 non-null  int64         
 18  reno_bool      21426 non-null  int64         
dtypes: datetime64[ns](1), float64(9), int64(9)
memory usage: 3.3 MB
```

In [48]:
```
df3[df3.bathrooms < .75].head()
```

Out[48]:

	date	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	view	grade	sqft_abov
2241	2014-10-03	273000.0	2	0.5	7750	1.0	0.0	0.0	6	59
10324	2015-01-14	109000.0	2	0.5	6900	1.0	0.0	0.0	5	58
11564	2014-08-14	255000.0	1	0.5	1642	1.0	0.0	0.0	6	50

In [49]:
```
df3 = df3.reset_index()
df3 = df3.drop(columns = 'index')
```

In [50]:
```
many_beds = df3[df3.bedrooms > 10]
many_beds.head()
```

Out[50]:

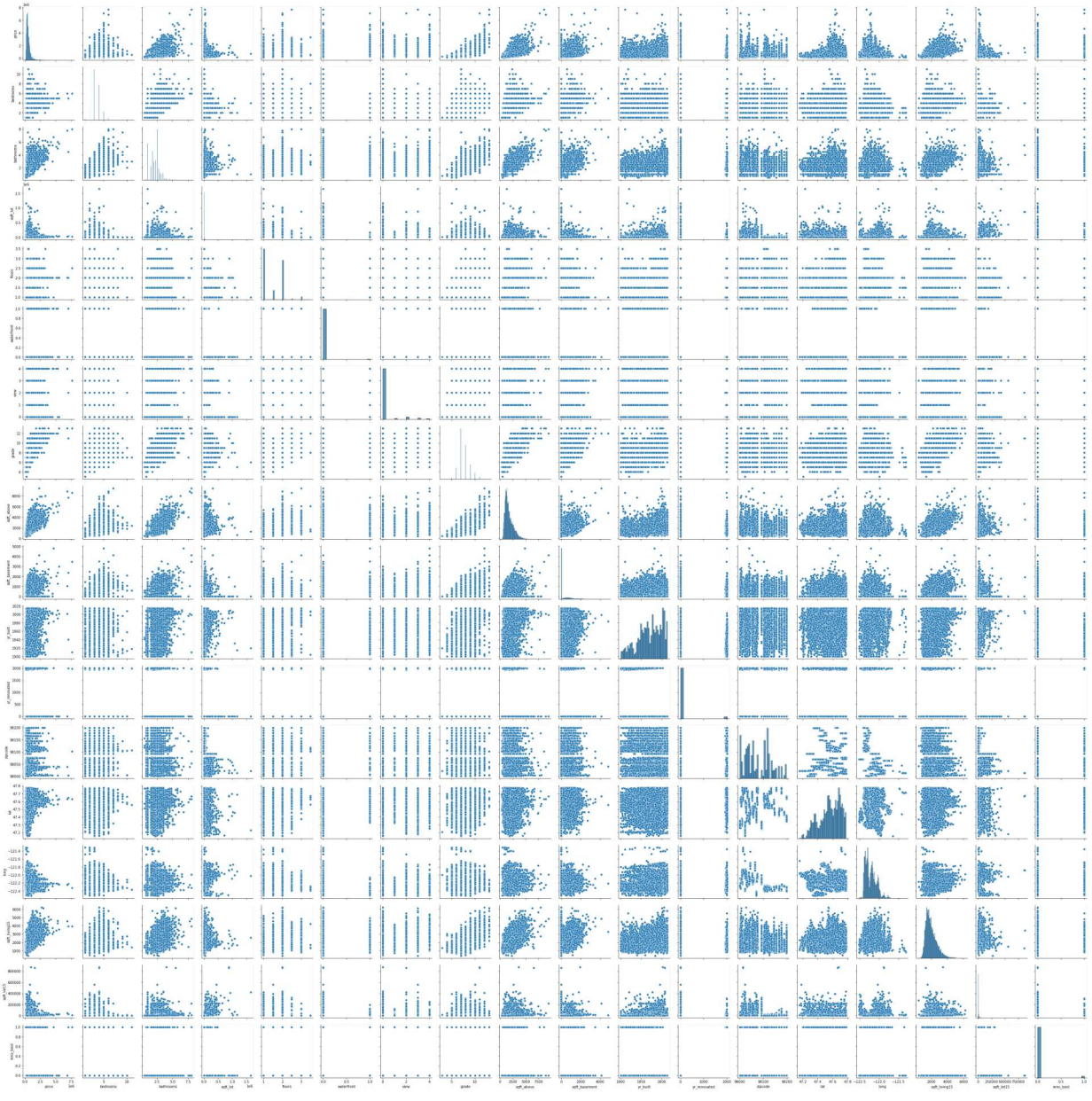
	date	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	view	grade	sqft_abov
8671	2014-08-21	520000.0	11	3.00	4960	2.0	0.0	0.0	7	240
15722	2014-06-25	640000.0	33	1.75	6000	1.0	0.0	0.0	7	104

The first one could be real, the second one definitely not based on the liveable area, so let's remove that one. We could guess that it has 3 bedrooms instead, but it's only one row.

In [51]:
```
df4 = df3.drop(15722)
df4 = df4.reset_index()
df4 = df4.drop(columns = 'index')
```

2.3 Explore!

In [52]:
```
sns.pairplot(data = df4);
```



Observations:

1. `price` looks like it may have linear relationships with respect to `bedrooms`, `bathrooms`, `sqft_lot`, `grade`, `sqft_above`, `sqft_basement`, `sqft_living15`, `sqft_lot15`, `lat`
2. `floors`, `waterfront`, `view`, `grade`, `condition`, `zipcode`, and `reno_bool` are all likely best interpreted as categorical columns, and maybe `bedrooms` and `bathrooms`
3. A lot of the histograms look right-skewed, so some log transformations may be useful.
4. There are some obvious outliers in some of the scatter plots; will have to deal with these later.
5. There are some IVs that look like they are linearly related, mostly between the IVs related to space or number of rooms.

In [53]:
```
df4.describe()
```

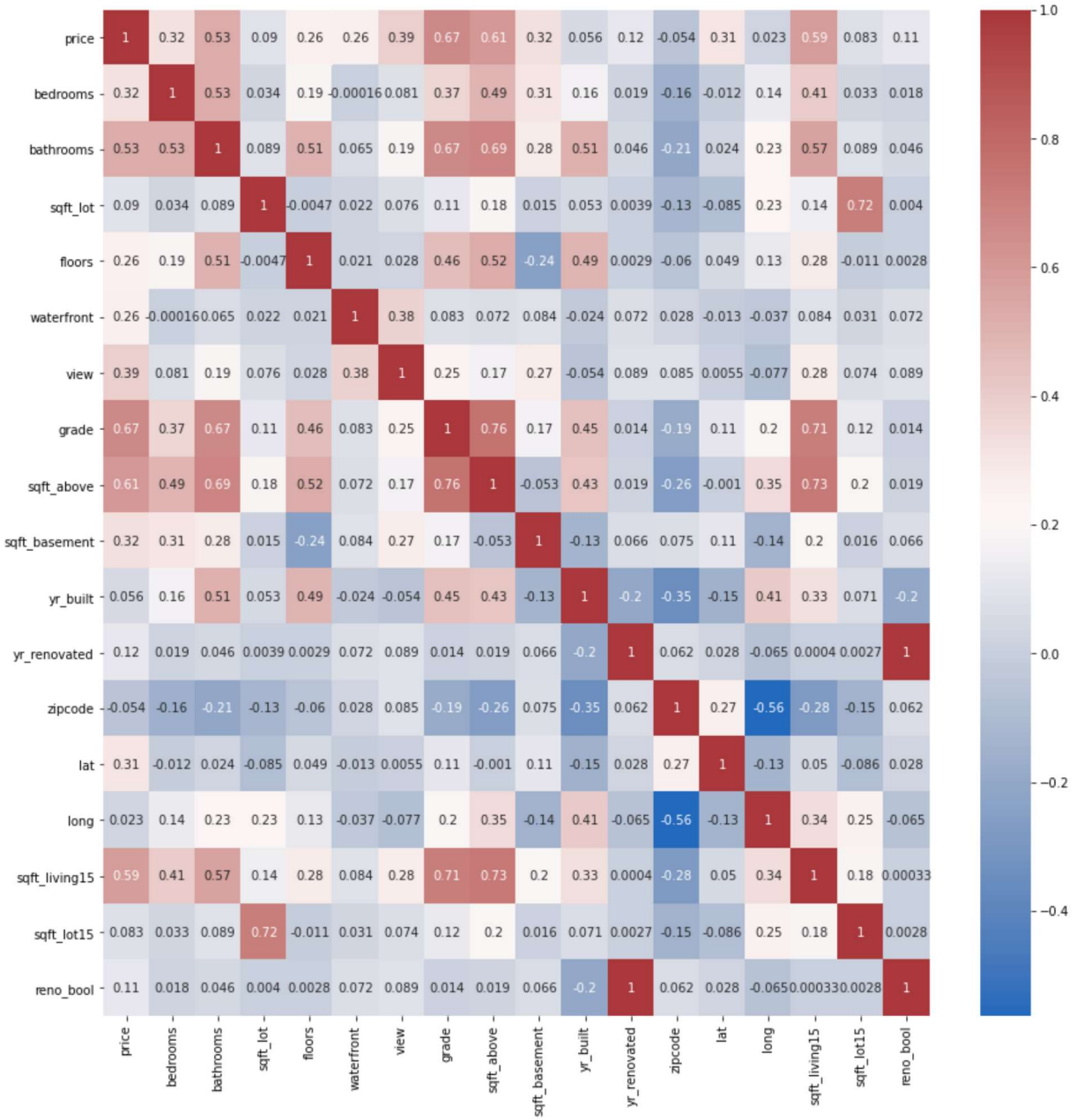
Out[53]:

	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	
count	2.142500e+04	21425.000000	21425.000000	2.142500e+04	21425.000000	21425.000000	21425
mean	5.398554e+05	3.369802	2.114597	1.510979e+04	1.495426	0.006721	0
std	3.674319e+05	0.902528	0.768867	4.149682e+04	0.539884	0.081708	0
min	7.800000e+04	1.000000	0.500000	5.200000e+02	1.000000	0.000000	0
25%	3.210000e+05	3.000000	1.750000	5.043000e+03	1.000000	0.000000	0
50%	4.500000e+05	3.000000	2.250000	7.620000e+03	1.500000	0.000000	0
75%	6.450000e+05	4.000000	2.500000	1.067500e+04	2.000000	0.000000	0
max	7.700000e+06	11.000000	8.000000	1.651359e+06	3.500000	1.000000	4

More observations:

1. The lowest and highest price may be outside what Abhineet is generally looking for in his business. The median price is ~450k while the mean is around 540k, again indicating a right skew. It may be worth limiting the dataset to a certain price range if it improves our model.
2. There are buildings with half floors, but that could be a split level type of plan?
3. For the numerical columns, there are different scales in magnitude. It may be worth scaling things between 1 and 0 to force things to be on the same magnitude.

In [54]:
```
plt.figure(figsize = (15,15))
sns.heatmap(data = df4.corr(),annot = True, cmap = 'vlag');
```



Observations:

1. Price has at least decent correlation with 8-10 different columns
2. There is some definite collinearity between columns; it will be worth trying to implement interactions in the modeling step. Interaction groupings (corr > 0.3): (bedrooms, bathrooms), (bedrooms, grade), (bedrooms, sqft_above), (bedrooms, sqft_living15), (bathrooms, floors), (bathrooms, grade), (bathrooms, sqft_above), (bathrooms, yr_built), (bathrooms, sqft_living15), (sqft_lot, sqft_lot15), (view, waterfront), (grade, floors), (sqft_above,floors), (sqft_above,grade), (yr_built, floors), -(zipcode, yr_built), -(long, zipcode), (sqft_living15,grade), (sqft_living15,sqft_above), (sqft_living15, yr_built), (sqft_living15, long), (long, sqft_above), (long, yr_built)

2.4 Model!

In [55]:
```
from statsmodels.formula.api import ols
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
import statsmodels.api as sm
import scipy.stats as stats
```

2.4.1 A baseline model

In [56]:
```
outcome = 'price'
predictors = df4.drop('price', axis = 1)
predictor_variables = "+".join(predictors.columns)
formula = outcome + '~' + predictor_variables
```

In [57]:
```
baseline_model = ols(formula = formula, data = df4).fit()
baseline_model.summary()
```

Out[57]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.707
Model:	OLS	**Adj. R-squared:**	0.701
Method:	Least Squares	**F-statistic:**	130.7
Date:	Wed, 09 Jun 2021	**Prob (F-statistic):**	0.00
Time:	11:41:30	**Log-Likelihood:**	-2.9180e+05
No. Observations:	21425	**AIC:**	5.844e+05
Df Residuals:	21036	**BIC:**	5.875e+05
Df Model:	388		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.262e+07	2.92e+06	4.317	0.000	6.89e+06	1.84e+07

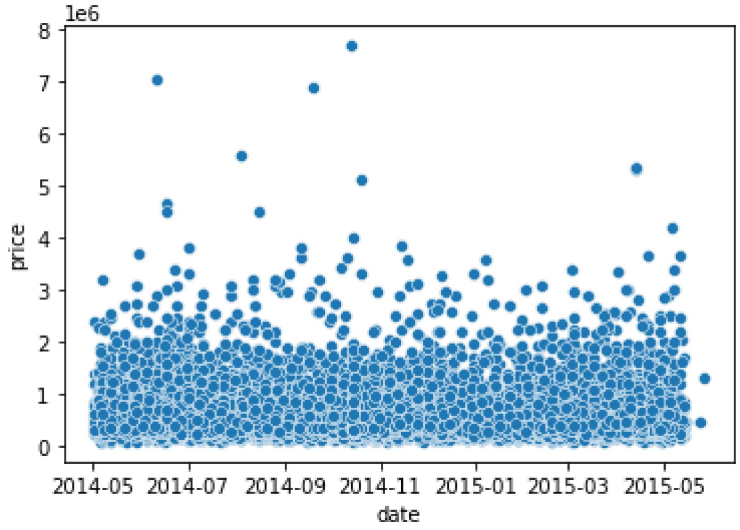
Observations:

1. The algorithm is hadnling the date column as a categorical variable, will have to deal with that.
2. The coeff for bedrooms is negative; that seems wrong.
3. `floors` has a high p-value, maybe not a good column to consider in the model, or needs to be considered as a category.

Let's quickly look at a scatter of date and price to see if it's worth trying to keep in the model.

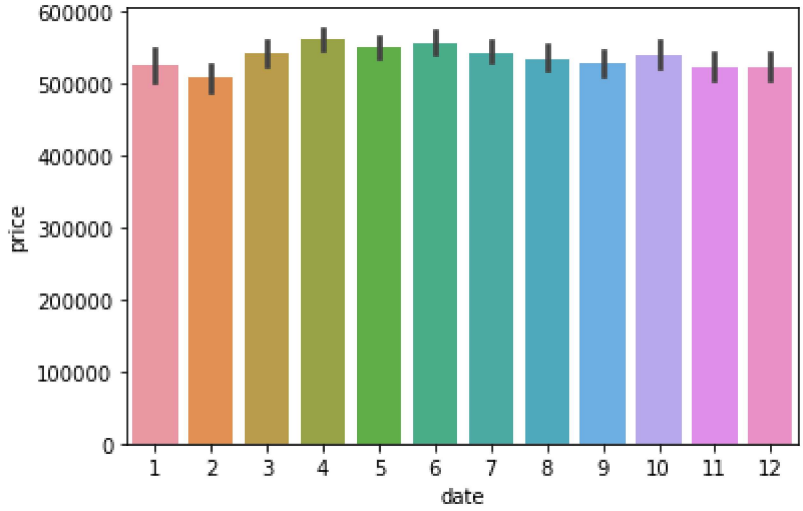
In [58]:
```
sns.scatterplot(data = df4, x = "date", y = "price")
```

Out[58]:
```
<AxesSubplot:xlabel='date', ylabel='price'>
```



In [59]:
```
sns.barplot(data = df4, x = df4.date.dt.month, y = 'price')
```

Out[59]:
```
<AxesSubplot:xlabel='date', ylabel='price'>
```



It doesn't look like there is any extremely important relationship between price and selling date, especially not a linear one. Let's remove that column from the predictors dataframe.

In [60]:
```
predictors = predictors.drop(columns = 'date')
```

In [61]:
```
predictor_variables = "+".join(predictors.columns)
formula = outcome + '~' + predictor_variables
```

Let's rerun the model to see if it has changed any (and to get a cleaner baseline summary to reference).

In [62]:
```
baseline_model = ols(formula = formula, data = df4).fit()
baseline_model.summary()
```

Out[62]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.699
Model:	OLS	**Adj. R-squared:**	0.699
Method:	Least Squares	**F-statistic:**	2931.
Date:	Wed, 09 Jun 2021	**Prob (F-statistic):**	0.00
Time:	11:41:31	**Log-Likelihood:**	-2.9207e+05
No. Observations:	21425	**AIC:**	5.842e+05
Df Residuals:	21407	**BIC:**	5.843e+05
Df Model:	17		
Covariance Type:	nonrobust		

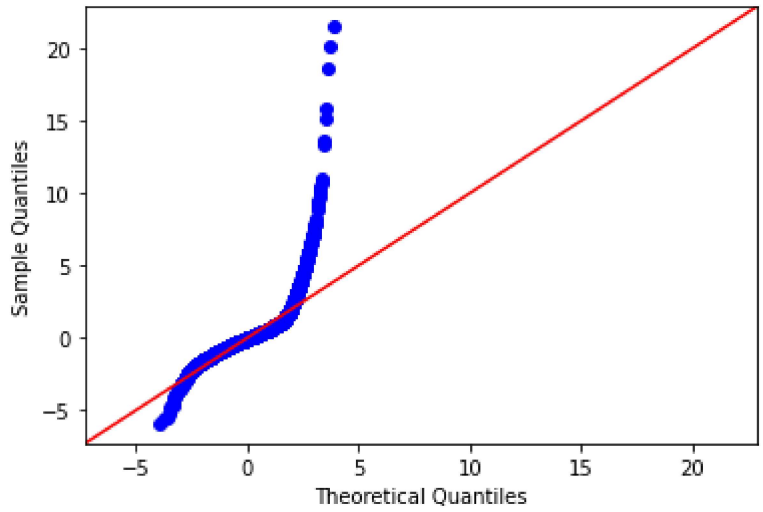
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.244e+07	2.91e+06	4.269	0.000	6.73e+06	1.81e+07
bedrooms	-3.894e+04	1994.687	-19.521	0.000	-4.28e+04	-3.5e+04
bathrooms	4.426e+04	3284.047	13.478	0.000	3.78e+04	5.07e+04
sqft_lot	0.1183	0.048	2.462	0.014	0.024	0.212
floors	4794.8839	3614.260	1.327	0.185	-2289.336	1.19e+04
waterfront	6.247e+05	1.83e+04	34.155	0.000	5.89e+05	6.61e+05
view	5.403e+04	2137.522	25.279	0.000	4.98e+04	5.82e+04
grade	9.59e+04	2172.819	44.137	0.000	9.16e+04	1e+05
sqft_above	183.2996	3.715	49.343	0.000	176.018	190.581
sqft_basement	155.6182	4.437	35.071	0.000	146.921	164.315
yr_built	-2911.1837	68.770	-42.332	0.000	-3045.977	-2776.390
yr_renovated	3248.4588	478.660	6.787	0.000	2310.249	4186.669
zipcode	-632.9610	32.958	-19.205	0.000	-697.562	-568.360
lat	5.939e+05	1.08e+04	55.113	0.000	5.73e+05	6.15e+05
long	-2.172e+05	1.32e+04	-16.447	0.000	-2.43e+05	-1.91e+05
sqft_living15	20.5763	3.465	5.939	0.000	13.785	27.367
sqft_lot15	-0.3838	0.074	-5.184	0.000	-0.529	-0.239
reno_bool	-6.453e+06	9.55e+05	-6.754	0.000	-8.33e+06	-4.58e+06

Omnibus:	17993.713	**Durbin-Watson:**	1.991
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1759766.128
Skew:	3.503	**Prob(JB):**	0.00
Kurtosis:	46.843	**Cond. No.**	2.13e+08

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.13e+08. This might indicate that there are
strong multicollinearity or other numerical problems.

Interestingly enough, the model R^2 got a little worse, but I think that was the right thing to do, since the date data did not really meet any requirements for a linear regression. Let's get a base qq plot and RMSE as well.

In [63]:
```
fig = sm.graphics.qqplot(baseline_model.resid, dist=stats.norm, line='45', fit=Tr
```



In [64]:
```
X_train, X_test, y_train, y_test = train_test_split(predictors, df4['price'], tes
baseline_linreg = LinearRegression()
baseline_linreg.fit(X_train, y_train)
y_pred = baseline_linreg.predict(X_test)
mse_train = mean_squared_error(y_train, baseline_linreg.predict(X_train))
mse_test = mean_squared_error(y_test, y_pred)
print("Train RMSE:", np.sqrt(mse_train))
print("Test RMSE:", np.sqrt(mse_test))
```

```
Train RMSE: 202496.35972326662
Test RMSE: 198429.40636895597
```

2.4.1.1 Defining a function for ease of evaluation

Based on the study groups, we will repeating a lot of these steps every time we want to try a new model. Let's see if we can write it as a function to save ourselves some time.

In [65]:
```
def evaluate_model(dataframe,log_model_bool):
    outcome = 'price'
    predictors = dataframe.drop('price', axis = 1)
    predictor_variables = "+".join(predictors.columns)
    formula = outcome + '~' + predictor_variables
    model = ols(formula = formula, data = dataframe).fit()
    print(model.summary())
    fig = sm.graphics.qqplot(model.resid, dist=stats.norm, line='45', fit=True)
    print('=========================================================================
    plt.show()
    X_train, X_test, y_train, y_test = train_test_split(predictors, dataframe[out
    linreg = LinearRegression()
    linreg.fit(X_train, y_train)
    y_test_pred = linreg.predict(X_test)
    y_train_pred = linreg.predict(X_train)
    if log_model_bool:
        y_train = np.exp(y_train)
        y_train_pred = np.exp(y_train_pred)
        y_test = np.exp(y_test)
        y_test_pred = np.exp(y_test_pred)
    mse_train = mean_squared_error(y_train, y_train_pred)
    mse_test = mean_squared_error(y_test, y_test_pred)
    print('=========================================================================
    print("Train RMSE:", np.sqrt(mse_train))
    print("Test RMSE:", np.sqrt(mse_test))
    print('=========================================================================
    return model
```

Let's see if it recreates the baseline model properly.

In [66]:
```
df5 = df4.drop(columns = 'date')
evaluate_model(df5,0);
```

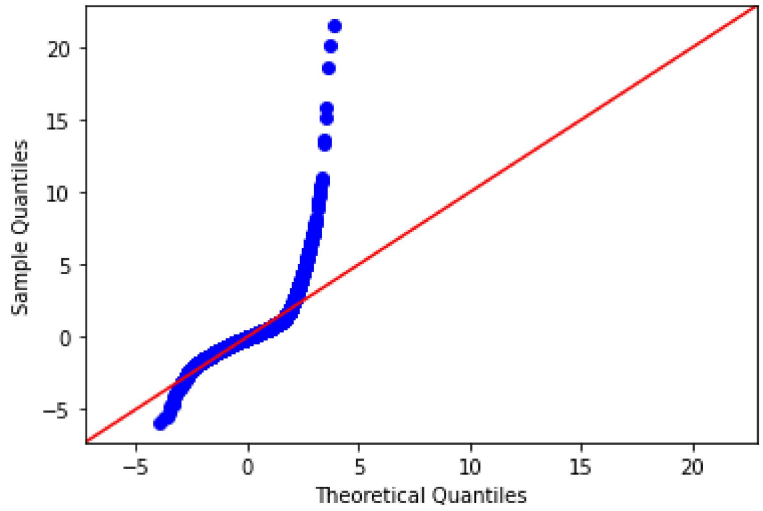
OLS Regression Results

Dep. Variable:	price	R-squared:	0.699
Model:	OLS	Adj. R-squared:	0.699
Method:	Least Squares	F-statistic:	2931.
Date:	Wed, 09 Jun 2021	Prob (F-statistic):	0.00
Time:	11:41:31	Log-Likelihood:	-2.9207e+05
No. Observations:	21425	AIC:	5.842e+05
Df Residuals:	21407	BIC:	5.843e+05
Df Model:	17		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.244e+07	2.91e+06	4.269	0.000	6.73e+06	1.81e+07
bedrooms	-3.894e+04	1994.687	-19.521	0.000	-4.28e+04	-3.5e+04
bathrooms	4.426e+04	3284.047	13.478	0.000	3.78e+04	5.07e+04
sqft_lot	0.1183	0.048	2.462	0.014	0.024	0.212
floors	4794.8839	3614.260	1.327	0.185	-2289.336	1.19e+04
waterfront	6.247e+05	1.83e+04	34.155	0.000	5.89e+05	6.61e+05
view	5.403e+04	2137.522	25.279	0.000	4.98e+04	5.82e+04
grade	9.59e+04	2172.819	44.137	0.000	9.16e+04	1e+05
sqft_above	183.2996	3.715	49.343	0.000	176.018	190.581
sqft_basement	155.6182	4.437	35.071	0.000	146.921	164.315
yr_built	-2911.1837	68.770	-42.332	0.000	-3045.977	-2776.390
yr_renovated	3248.4588	478.660	6.787	0.000	2310.249	4186.669
zipcode	-632.9610	32.958	-19.205	0.000	-697.562	-568.360
lat	5.939e+05	1.08e+04	55.113	0.000	5.73e+05	6.15e+05
long	-2.172e+05	1.32e+04	-16.447	0.000	-2.43e+05	-1.91e+05
sqft_living15	20.5763	3.465	5.939	0.000	13.785	27.367
sqft_lot15	-0.3838	0.074	-5.184	0.000	-0.529	-0.239
reno_bool	-6.453e+06	9.55e+05	-6.754	0.000	-8.33e+06	-4.58e+06

```
Omnibus:                    17993.713   Durbin-Watson:                   1.991
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1759766.128
Skew:                           3.503   Prob(JB):                         0.00
Kurtosis:                      46.843   Cond. No.                     2.13e+08
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 2.13e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
==============================================================================
```




```
==============================================================================
Train RMSE: 205386.46903354133
Test RMSE: 189615.9878011534
==============================================================================
```

Sick!

2.4.2 Investigating Categorical Variables

While date as a categorical variable didn't help, let's see if casting some of the others we identified as categorical candidates in our pairplot improves our fit. We want to remove the column with the highest frequency, so that we can interpret the coefficients of the remaining predictors as to how they are different from the most common.

In [67]:
```
def test_dummies(dataframe, candidates):
    for candidate in candidates:
        dummies = pd.get_dummies(dataframe[candidate], prefix = candidate, drop_f
        col_to_drop = dataframe[candidate].value_counts().index[0]
        dummies.drop(columns = f'{candidate}_{col_to_drop}',inplace = True)
        df_with_dummies = pd.concat([dataframe,dummies], axis = 1)
        df_with_dummies.drop(columns = candidate, inplace = True)
        print(f'Candidate: {candidate}')
        evaluate_model(df_with_dummies,0)
```

Have to clean up a few columns since `pd.get_dummies` doesn't work well when it tries to make floats into column names.

In [68]:
```
df5['floors'] = df5.floors.astype(int)
df5['waterfront'] = df5.waterfront.astype(int)
df5['view'] = df5.view.astype(int)
```

In [69]:
```
candidates = ['floors', 'waterfront', 'view', 'grade', 'zipcode', 'reno_bool', 'b
test_dummies(df5, candidates)
```

```
Candidate: floors
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  price   R-squared:                       0.70
0
Model:                            OLS   Adj. R-squared:                  0.70
0
Method:                 Least Squares   F-statistic:                     277
7.
Date:                Wed, 09 Jun 2021   Prob (F-statistic):               0.0
0
Time:                        11:41:31   Log-Likelihood:            -2.9204e+0
5
No. Observations:               21425   AIC:                         5.841e+0
5
Df Residuals:                   21406   BIC:                         5.843e+0
5
Df Model:                          18
```

So, for `waterfront` and `reno_bool`, the model fit did not improve. For `bedrooms`, `floors`, and `view` the model improved very slightly. However, in the case of `grade` it increased the R^2 by nearly .03, and in the case of `zipcode`, it increased it by just over .1, which is huge. The QQ plots seem to look more or less the same, however. Also, in many candidates, some of the categories do not have significance according to their p values. According to the internet, I should still keep these columns. Another thing to note is that for every one except where `floors` are categorized, it has a really high p-value. Does that indicate it is only useful when categorized? Finally, when looking at RMSE's values only decrease noticeably for `grade` and `zipcode`, as with R^2. I am thinking that for `reno_bool` knowing whether or not is has ever been renovated is perhaps not as useful as knowing how long ago it was renovated. However, since a large majority have never been renovated, I think it will be best to first drop rows that have been renovated in 2010 or after, since we will likely not want to buy them based on our business objective, and then

second drop both columns `yr_renovated` and `reno_bool` . While if more had been renovated at some point, and we could get more information about how recently a home has to have bene renovated to increase sale price, that would be useful, but since most of the values in yr_renovated are simply 0, which doesn't make sense to fit in a model, we will just remove it.

In [70]:
```
df6 = df5.drop(index = df5[df5.yr_renovated >= 2010].index)
df6 = df6.drop(columns = ['yr_renovated','reno_bool']).reset_index()
df6 = df6.drop(columns = 'index')
df6.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21278 entries, 0 to 21277
Data columns (total 16 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          21278 non-null  float64
 1   bedrooms       21278 non-null  int64  
 2   bathrooms      21278 non-null  float64
 3   sqft_lot       21278 non-null  int64  
 4   floors         21278 non-null  int32  
 5   waterfront     21278 non-null  int32  
 6   view           21278 non-null  int32  
 7   grade          21278 non-null  int64  
 8   sqft_above     21278 non-null  int64  
 9   sqft_basement  21278 non-null  float64
 10  yr_built       21278 non-null  int64  
 11  zipcode        21278 non-null  int64  
 12  lat            21278 non-null  float64
 13  long           21278 non-null  float64
 14  sqft_living15  21278 non-null  int64  
 15  sqft_lot15     21278 non-null  int64  
dtypes: float64(5), int32(3), int64(8)
memory usage: 2.4 MB
```

Finally, what if we cast all of them as categorical variables simultaneously?

In [71]:
```
cat_candidates = ['floors', 'waterfront', 'view', 'grade', 'zipcode', 'bedrooms']
df6_with_all_dummies = df6.copy()
for cat_candidate in cat_candidates:
    dummies = pd.get_dummies(df6_with_all_dummies[cat_candidate], prefix = cat_ca
    df6_with_all_dummies = pd.concat([df6_with_all_dummies,dummies], axis = 1)
    df6_with_all_dummies.drop(columns = cat_candidate, inplace = True)
evaluate_model(df6_with_all_dummies,0)
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  price   R-squared:                       0.83
4
Model:                            OLS   Adj. R-squared:                  0.83
4
Method:                 Least Squares   F-statistic:                     101
6.
Date:                Wed, 09 Jun 2021   Prob (F-statistic):               0.0
0
Time:                        11:41:33   Log-Likelihood:            -2.8369e+0
5
No. Observations:               21278   AIC:                         5.676e+0
5
Df Residuals:                   21172   BIC:                         5.684e+0
5
Df Model:                         105
Covariance Type:            nonrobust
```

Well that is pretty darn good! With some googling we could probably figure out how to turn bathrooms into a category as well, but since there are half and three quarter values I think we'll just leave it.

2.4.3 Investigating multicollinearity and reducing it's effects with interactions

Let's see if we can use things identified as collinear to improve the model by creating interaction terms. Let's bring down the groupings we made earlier for easy reference:

- (bedrooms, bathrooms), (bedrooms, grade), (bedrooms, sqft_above), (bedrooms, sqft_living15), (bathrooms, floors), (bathrooms, grade), (bathrooms, sqft_above), (bathrooms, yr_built), (bathrooms, sqft_living15), (sqft_lot, sqft_lot15), (view, waterfront), (grade, floors), (sqft_above, floors), (sqft_above, grade), (yr_built, floors), -(zipcode, yr_built), -(long, zipcode), (sqft_living15,grade), (sqft_living15,sqft_above), (sqft_living15, yr_built), (sqft_living15, long), (long, sqft_above), (long, yr_built)

That's a lot of potential options, not to mention that there are many multi-interactions that could be derived from these. To make life easier, let's see if we can create and then consider only these larger groupings. Plus, for `bedrooms`, `bathrooms`, `floors`, `grade`, `view`, and `waterfront`, `zipcode` we already know that we may want to include these as categorical columns, so it is not clear how the correlations we got previously might change, so we will investigate including those as interactions in the final model, perhaps. This is what I've come up with:

- (bedrooms, bathrooms, sqft_above, sqft_living15), (sqft_lot, sqft_lot15), (sqft_living15, yr_built, long, sqft_above)

In [72]:
```
df6_with_inter1 = df6.copy()
df6_with_inter1['bbss'] = df6.bedrooms * df6.bathrooms * df6.sqft_above * df6.sqf
evaluate_model(df6_with_inter1,0)
```

OLS Regression Results

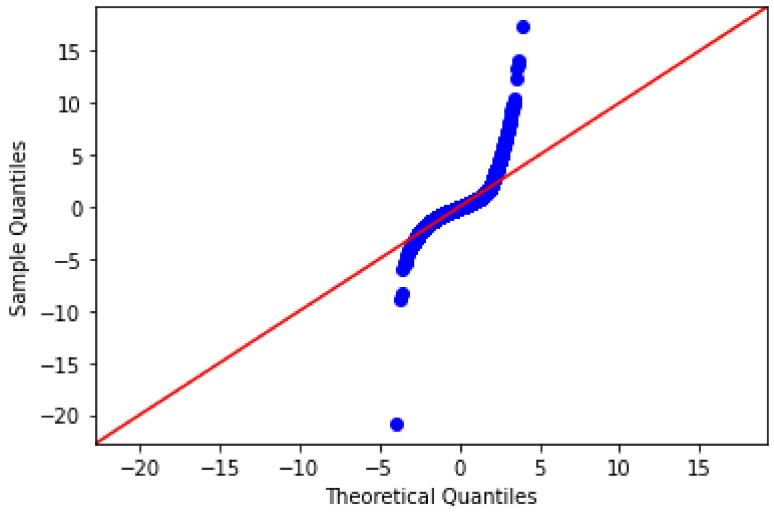
Dep. Variable:	price	R-squared:	0.719
Model:	OLS	Adj. R-squared:	0.719
Method:	Least Squares	F-statistic:	3396.
Date:	Wed, 09 Jun 2021	Prob (F-statistic):	0.00
Time:	11:41:33	Log-Likelihood:	-2.8933e+05
No. Observations:	21278	AIC:	5.787e+05
Df Residuals:	21261	BIC:	5.788e+05
Df Model:	16		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.801e+07	2.82e+06	6.377	0.000	1.25e+07	2.35e+07
bedrooms	-4.02e+04	1936.510	-20.760	0.000	-4.4e+04	-3.64e+04
bathrooms	2.107e+04	3244.304	6.494	0.000	1.47e+04	2.74e+04
sqft_lot	0.1502	0.046	3.234	0.001	0.059	0.241
floors	1.377e+04	3595.222	3.829	0.000	6719.482	2.08e+04
waterfront	6.251e+05	1.77e+04	35.402	0.000	5.91e+05	6.6e+05
view	5.382e+04	2076.034	25.923	0.000	4.97e+04	5.79e+04
grade	1.024e+05	2109.929	48.519	0.000	9.82e+04	1.07e+05
sqft_above	111.0052	4.072	27.260	0.000	103.023	118.987
sqft_basement	136.8987	4.308	31.778	0.000	128.455	145.343
yr_built	-2787.6632	67.648	-41.208	0.000	-2920.259	-2655.068
zipcode	-664.8038	31.911	-20.833	0.000	-727.351	-602.256
lat	5.907e+05	1.04e+04	56.649	0.000	5.7e+05	6.11e+05
long	-1.974e+05	1.28e+04	-15.441	0.000	-2.22e+05	-1.72e+05
sqft_living15	-7.7227	3.414	-2.262	0.024	-14.415	-1.031
sqft_lot15	-0.3940	0.072	-5.509	0.000	-0.534	-0.254
bbss	0.0015	4e-05	38.510	0.000	0.001	0.002

Omnibus:	12297.136	Durbin-Watson:	1.986

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           893620.246
Skew:                           1.969   Prob(JB):                         0.00
Kurtosis:                      34.503   Cond. No.                     1.54e+11
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 1.54e+11. This might indicate that there are
strong multicollinearity or other numerical problems.
==============================================================================
```




```
==============================================================================
Train RMSE: 196160.3595147717
Test RMSE: 189794.70048398554
==============================================================================
```

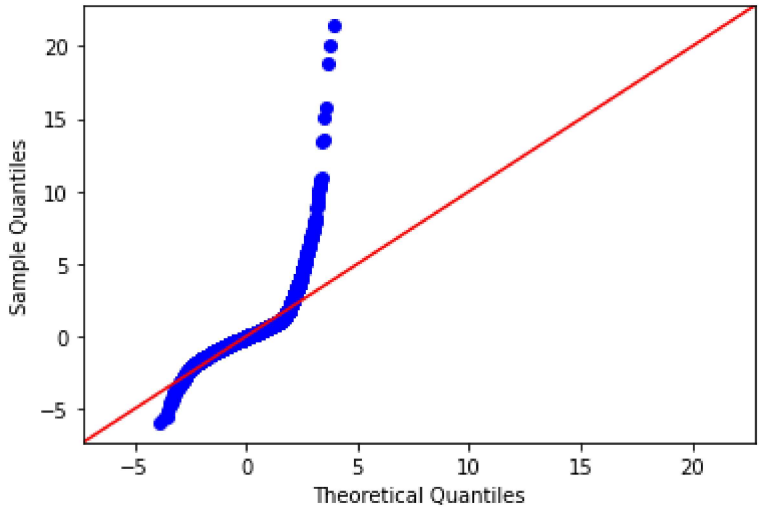
Out[72]: <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x24ece8e7eb0>

In [73]:
```
df6_with_inter2 = df6.copy()
df6_with_inter2['sl_sl15'] = df6.sqft_lot * df6.sqft_lot15
evaluate_model(df6_with_inter2,0)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.699
Model:                            OLS   Adj. R-squared:                  0.699
Method:                 Least Squares   F-statistic:                     3091.
Date:                Wed, 09 Jun 2021   Prob (F-statistic):               0.00
Time:                        11:41:34   Log-Likelihood:            -2.9004e+05
No. Observations:               21278   AIC:                         5.801e+05
Df Residuals:                   21261   BIC:                         5.802e+05
Df Model:                          16
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    1.266e+07   2.92e+06      4.336      0.000    6.94e+06    1.84e+07
bedrooms     -3.93e+04   2002.632    -19.623      0.000   -4.32e+04   -3.54e+04
bathrooms    4.647e+04   3283.619     14.151      0.000       4e+04    5.29e+04
sqft_lot        0.0299      0.053      0.566      0.572      -0.074       0.133
floors       1312.2544   3711.593      0.354      0.724   -5962.749    8587.258
waterfront   6.262e+05   1.83e+04     34.280      0.000    5.9e+05    6.62e+05
view         5.408e+04   2146.684     25.192      0.000    4.99e+04    5.83e+04
grade        9.645e+04   2175.951     44.325      0.000    9.22e+04    1.01e+05
sqft_above    185.3940      3.721     49.827      0.000     178.101     192.687
sqft_basement 155.3717      4.428     35.092      0.000     146.693     164.050
yr_built     -2956.2499     69.804    -42.351      0.000   -3093.071   -2819.429
zipcode       -626.6338     32.978    -19.001      0.000    -691.274    -561.994
lat          5.932e+05   1.08e+04     55.005      0.000    5.72e+05    6.14e+05
long        -2.113e+05   1.33e+04    -15.933      0.000   -2.37e+05   -1.85e+05
sqft_living15    18.3480      3.470      5.288      0.000      11.547      25.149
sqft_lot15       -0.5185      0.083     -6.282      0.000      -0.680      -0.357
sl_sl15       7.398e-07   1.95e-07      3.791      0.000    3.57e-07    1.12e-06
==============================================================================
Omnibus:                    17861.682   Durbin-Watson:                   1.985
```

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1747446.931
Skew:                           3.500   Prob(JB):                         0.00
Kurtosis:                      46.841   Cond. No.                     2.39e+13
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 2.39e+13. This might indicate that there are
strong multicollinearity or other numerical problems.
==============================================================================
```




```
==============================================================================
Train RMSE: 200807.98355915712
Test RMSE: 202311.7916422509
==============================================================================
```

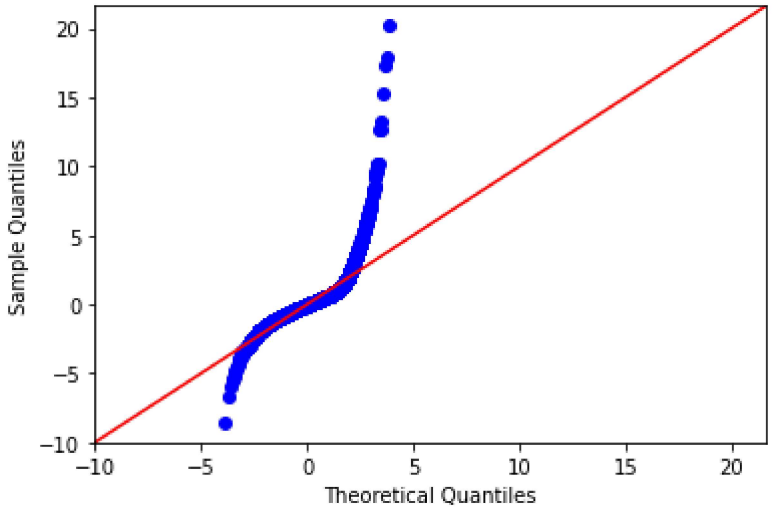
Out[73]: <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x24ed0846880>

In [74]:
```
df6_with_inter3 = df6.copy()
df6_with_inter3['y_l'] =  df6.yr_built * df6.long * df6.sqft_living15 * df6.sqft_
evaluate_model(df6_with_inter3,0)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.707
Model:                            OLS   Adj. R-squared:                  0.707
Method:                 Least Squares   F-statistic:                     3420.
Date:                Wed, 09 Jun 2021   Prob (F-statistic):               0.00
Time:                        11:41:34   Log-Likelihood:            -2.8976e+05
No. Observations:               21278   AIC:                         5.796e+05
Df Residuals:                   21262   BIC:                         5.797e+05
Df Model:                          15
Covariance Type:            nonrobust
==============================================================================
==
                   coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
Intercept        71.3894      6.696     10.661      0.000      58.264      84.5
15
bedrooms      -2.867e+04   2015.024    -14.228      0.000   -3.26e+04   -2.47e+
04
bathrooms      5.245e+04   3249.458     16.142      0.000    4.61e+04    5.88e+
04
sqft_lot          0.1739      0.047      3.670      0.000       0.081       0.2
67
floors         4386.4210   3602.782      1.218      0.223   -2675.304    1.14e+
04
waterfront       6.2e+05    1.8e+04     34.398      0.000    5.85e+05    6.55e+
05
view           5.571e+04   2110.416     26.400      0.000    5.16e+04    5.99e+
04
grade          9.592e+04   2141.229     44.796      0.000    9.17e+04       1e+
05
sqft_above       67.7850      6.086     11.138      0.000      55.856      79.7
14
sqft_basement   147.9630      4.360     33.933      0.000     139.416     156.5
10
yr_built      -2831.0082     64.986    -43.563      0.000   -2958.386   -2703.6
30
zipcode        -505.0256     17.707    -28.522      0.000    -539.732    -470.3
19
lat            5.933e+05   1.06e+04     55.784      0.000    5.72e+05    6.14e+
05
long          -2.166e+05    1.3e+04    -16.706      0.000   -2.42e+05   -1.91e+
05
sqft_living15   -72.5421      5.099    -14.226      0.000     -82.537     -62.5
47
sqft_lot15       -0.3696      0.073     -5.067      0.000      -0.513      -0.2
27
y_l           -1.777e-07   7.34e-09    -24.197      0.000   -1.92e-07   -1.63e-
07
==============================================================================
Omnibus:                    16069.565   Durbin-Watson:                   1.989
```

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1113961.234
Skew:                           3.046   Prob(JB):                         0.00
Kurtosis:                      37.919   Cond. No.                      2.68e+15
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 2.68e+15. This might indicate that there are
strong multicollinearity or other numerical problems.
==============================================================================
```




```
==============================================================================
Train RMSE: 197383.7235964955
Test RMSE: 201544.11038486427
==============================================================================
```

Out[74]: `<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x24ed0fc1c40>`

The first and third interactions did improve the model fit over the baseline, though the qq plots and RMSEs did not change much. In the first, the RMSE may improve a little, but it causes some points on the low end of the qq plot to be skewed radically. The second, while not improving the fit, does actually improve the residuals on that end.

2.4.4 Looking at transformations in improving model fit

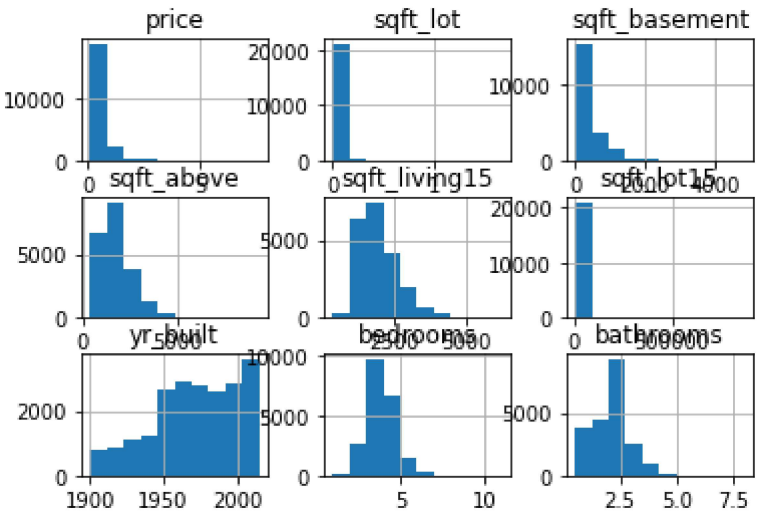
2.4.4.1 Log Transformations

In [75]:
```
def test_log_transforms(dataframe, candidates):
    for candidate in candidates:
        dataframe_copy = dataframe.copy()
        dataframe_copy[candidate] = np.log(dataframe_copy[candidate])
        print(f'Candidate: {candidate}')
        log_model_bool = 0
        if candidate == 'price':
            log_model_bool = 1
        evaluate_model(dataframe_copy,log_model_bool)
```

By making some histograms, we can see which distributions aren't normal and include them in candidates.

In [76]:
```
df6[['price','sqft_lot','sqft_basement','sqft_above','sqft_living15','sqft_lot15'
```

Out[76]:
```
array([[<AxesSubplot:title={'center':'price'}>,
        <AxesSubplot:title={'center':'sqft_lot'}>,
        <AxesSubplot:title={'center':'sqft_basement'}>],
       [<AxesSubplot:title={'center':'sqft_above'}>,
        <AxesSubplot:title={'center':'sqft_living15'}>,
        <AxesSubplot:title={'center':'sqft_lot15'}>],
       [<AxesSubplot:title={'center':'yr_built'}>,
        <AxesSubplot:title={'center':'bedrooms'}>,
        <AxesSubplot:title={'center':'bathrooms'}>]], dtype=object)
```



Seems worth trying them all.

In [77]:
```
candidates = ['price','sqft_lot','sqft_above','sqft_living15','sqft_lot15','yr_bu
test_log_transforms(df6,candidates)
```

```
Candidate: price
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                    price   R-squared:                       0.76
5
Model:                            OLS   Adj. R-squared:                  0.76
4
Method:                 Least Squares   F-statistic:                     460
3.
Date:                Wed, 09 Jun 2021   Prob (F-statistic):               0.0
0
Time:                        11:41:35   Log-Likelihood:                -1152.
1
No. Observations:               21278   AIC:                             233
6.
Df Residuals:                   21262   BIC:                             246
4.
Df Model:                          15
```

Hmmm... so log transforming only the price seems to be the only log transform that really improves the model. To be honest, I'm not so sure that it's the transformation that is improving the fit so much as how log transforming scales the values. From this we may infer that it only makes sense to log transform all or none. We will see how min-max scaling affects the fit, and come back around to this.

2.4.4.2 Min-Max Scaling

In [78]:
```
def test_minmax_scaling(dataframe, candidates):
    scaler = MinMaxScaler(feature_range = (0,10))
    for candidate in candidates:
        dataframe_copy = dataframe.copy()
        dataframe_copy[candidate] = scaler.fit_transform(dataframe_copy[candidate
        print(f'Candidate: {candidate}')
        evaluate_model(dataframe_copy,0)
```

In [79]:
```
candidates = ['price','sqft_lot','sqft_above', 'sqft_living15','sqft_lot15','yr_b
test_log_transforms(df6,candidates)
```

```
Candidate: price
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                   price   R-squared:                       0.76
5
Model:                             OLS   Adj. R-squared:                  0.76
4
Method:                  Least Squares   F-statistic:                     460
3.
Date:                Wed, 09 Jun 2021   Prob (F-statistic):               0.0
0
Time:                        11:41:36   Log-Likelihood:                -1152.
1
No. Observations:               21278   AIC:                             233
6.
Df Residuals:                   21262   BIC:                             246
4.
Df Model:                          15
```

Same as for the log transforms, only the scaling on `price` really makes a huge difference and scaling `sqft_above` has a negative impact. Let's try scaling and log transforming the entire set to see what the fits look like.

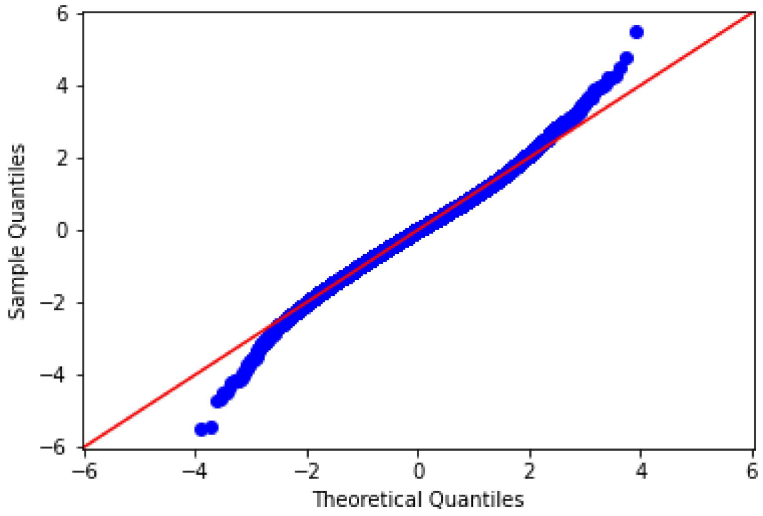
2.4.4.3 Total dataframe transformations

In [80]:
```
df6_log_transformed = df6.copy()
for candidate in candidates:
    df6_log_transformed[candidate] = np.log(df6_log_transformed[candidate])
evaluate_model(df6_log_transformed,1)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.768
Model:                            OLS   Adj. R-squared:                  0.767
Method:                 Least Squares   F-statistic:                     4680.
Date:                Wed, 09 Jun 2021   Prob (F-statistic):               0.00
Time:                        11:41:37   Log-Likelihood:                -1016.5
No. Observations:               21278   AIC:                             2065.
Df Residuals:                   21262   BIC:                             2192.
Df Model:                          15
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept        61.7594      4.286     14.409      0.000      53.358      70.161
bedrooms         -0.0252      0.003     -9.691      0.000      -0.030      -0.020
bathrooms         0.0764      0.004     18.511      0.000       0.068       0.084
sqft_lot          0.0079      0.005      1.553      0.120      -0.002       0.018
floors            0.0324      0.005      6.451      0.000       0.023       0.042
waterfront        0.4086      0.023     17.717      0.000       0.363       0.454
view              0.0646      0.003     23.928      0.000       0.059       0.070
grade             0.1653      0.003     62.171      0.000       0.160       0.170
sqft_above        0.3158      0.009     33.258      0.000       0.297       0.334
sqft_basement     0.0002   5.57e-06     28.152      0.000       0.000       0.000
yr_built         -8.6734      0.172    -50.524      0.000      -9.010      -8.337
zipcode          -0.0007   4.17e-05    -16.034      0.000      -0.001      -0.001
lat               1.3654      0.014     99.779      0.000       1.339       1.392
long             -0.1025      0.017     -6.055      0.000      -0.136      -0.069
sqft_living15     0.2112      0.009     23.793      0.000       0.194       0.229
sqft_lot15       -0.0365      0.006     -6.642      0.000      -0.047      -0.026
==============================================================================
Omnibus:                      356.751   Durbin-Watson:                   1.984
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              714.839
```

```
Skew:                          0.032   Prob(JB):                    5.95e-156
Kurtosis:                      3.896   Cond. No.                     2.42e+08
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 2.42e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
==============================================================================
```




```
==============================================================================
Train RMSE: 187246.74509084504
Test RMSE: 177291.1027187108
==============================================================================
```

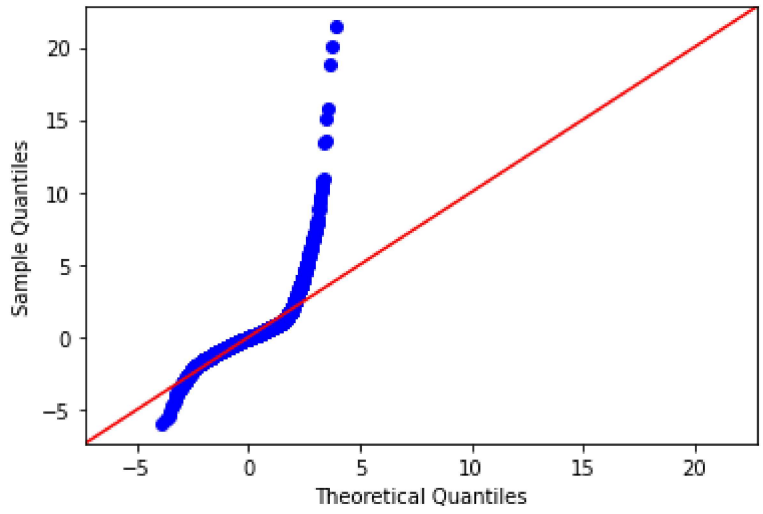
Out[80]: <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x24ec4c5d2e0>

In [81]:
```
df6_scaled = df6.copy()
scaler = MinMaxScaler(feature_range = (0,10))
df6_scaled[candidates] = scaler.fit_transform(df6_scaled[candidates])
evaluate_model(df6_scaled,0)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.699
Model:                            OLS   Adj. R-squared:                  0.699
Method:                 Least Squares   F-statistic:                     3294.
Date:                Wed, 09 Jun 2021   Prob (F-statistic):               0.00
Time:                        11:41:37   Log-Likelihood:                -1854.1
No. Observations:               21278   AIC:                             3740.
Df Residuals:                   21262   BIC:                             3868.
Df Model:                          15
Covariance Type:            nonrobust
==============================================================================
==
                   coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
Intercept         8.4899      3.772      2.251      0.024       1.096      15.8
84
bedrooms         -0.0513      0.003    -19.529      0.000      -0.056      -0.0
46
bathrooms         0.0613      0.004     14.228      0.000       0.053       0.0
70
sqft_lot          0.0246      0.010      2.365      0.018       0.004       0.0
45
floors            0.0029      0.005      0.593      0.553      -0.007       0.0
12
waterfront        0.8186      0.024     34.162      0.000       0.772       0.8
66
view              0.0711      0.003     25.240      0.000       0.066       0.0
77
grade             0.1266      0.003     44.345      0.000       0.121       0.1
32
sqft_above        0.2191      0.004     49.685      0.000       0.210       0.2
28
sqft_basement     0.0002   5.81e-06     35.033      0.000       0.000       0.0
00
yr_built         -0.0446      0.001    -42.307      0.000      -0.047      -0.0
43
zipcode          -0.0008   4.33e-05    -19.006      0.000      -0.001      -0.0
01
lat               0.7799      0.014     55.123      0.000       0.752       0.8
08
long             -0.2830      0.017    -16.328      0.000      -0.317      -0.2
49
sqft_living15     0.0134      0.003      5.079      0.000       0.008       0.0
19
sqft_lot15       -0.0433      0.008     -5.131      0.000      -0.060      -0.0
27
==============================================================================
Omnibus:                    17859.592   Durbin-Watson:                   1.985
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1746739.564
```

```
Skew:                          3.499   Prob(JB):                         0.00
Kurtosis:                     46.832   Cond. No.                     2.04e+08
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 2.04e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
==============================================================================
```




```
==============================================================================
Train RMSE: 0.2630995126426033
Test RMSE: 0.26714663322972015
==============================================================================
```

Out[81]: `<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x24ec49e1c10>`

In [82]:
```
df6_scaled.head()
```

Out[82]:

	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	view	grade	sqft_above	sqft_bas
0	0.188796	3	1.00	0.031075	1	0	0	7	0.896018	
1	0.603516	3	2.25	0.040719	2	0	0	7	1.991150	
2	0.133823	2	1.00	0.057425	1	0	0	6	0.442478	
3	0.690108	4	3.00	0.027138	1	0	0	7	0.752212	
4	0.566780	3	2.00	0.045795	1	0	0	8	1.449115	

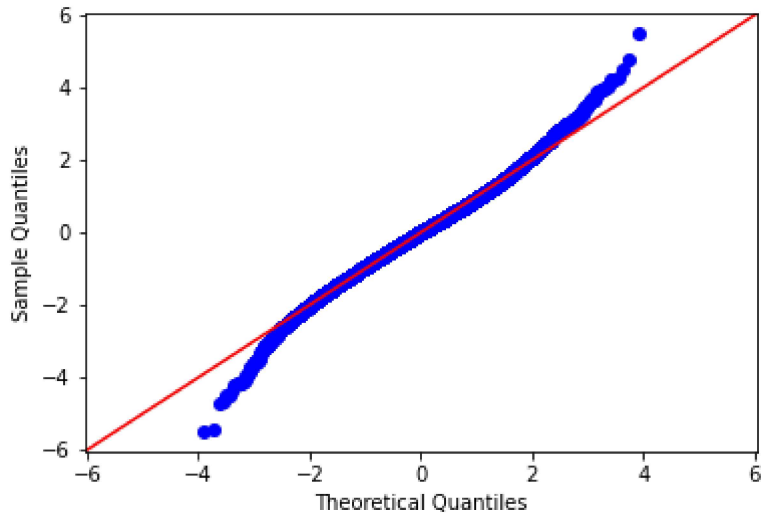
Odd... while log transforming all the candidate columns improves the model fit, min-max scaling all of them does nothing whatsoever. It could be because some remain unscaled, such as sqft_basement, and there are still the long, lat, and other categories we aren't considering? Changing the feature range does not seem to make a difference, nor does just scaling all the columns, not just some. Just for fun, let's try doing both, maybe in the order that we log transform and then scale?

In [83]:
```
df6_both = df6.copy()
for candidate in candidates:
    df6_both[candidate] = np.log(df6_both[candidate])
scaler = MinMaxScaler(feature_range = (0,10))
df6_both[candidates] = scaler.fit_transform(df6_both[candidates])
evaluate_model(df6_both,0)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.768
Model:                            OLS   Adj. R-squared:                  0.767
Method:                 Least Squares   F-statistic:                     4680.
Date:                Wed, 09 Jun 2021   Prob (F-statistic):               0.00
Time:                        11:41:38   Log-Likelihood:                -17575.
No. Observations:               21278   AIC:                         3.518e+04
Df Residuals:                   21262   BIC:                         3.531e+04
Df Model:                          15
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Intercept       -26.2199      8.066     -3.251      0.001     -42.029     -10.411
bedrooms         -0.0550      0.006     -9.691      0.000      -0.066      -0.044
bathrooms         0.1664      0.009     18.511      0.000       0.149       0.184
sqft_lot          0.0138      0.009      1.553      0.120      -0.004       0.031
floors            0.0706      0.011      6.451      0.000       0.049       0.092
waterfront        0.8897      0.050     17.717      0.000       0.791       0.988
view              0.1407      0.006     23.928      0.000       0.129       0.152
grade             0.3599      0.006     62.171      0.000       0.349       0.371
sqft_above        0.2225      0.007     33.258      0.000       0.209       0.236
sqft_basement     0.0003   1.21e-05     28.152      0.000       0.000       0.000
yr_built         -0.1110      0.002    -50.524      0.000      -0.115      -0.107
zipcode          -0.0015   9.07e-05    -16.034      0.000      -0.002      -0.001
```

```
lat              2.9732      0.030     99.779      0.000       2.915
3.032
long            -0.2231      0.037     -6.055      0.000      -0.295      -
0.151
sqft_living15    0.1262      0.005     23.793      0.000       0.116
0.137
sqft_lot15      -0.0573      0.009     -6.642      0.000      -0.074      -
0.040
==============================================================================
=
Omnibus:                      356.751   Durbin-Watson:                   1.98
4
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              714.83
9
Skew:                           0.032   Prob(JB):                     5.95e-15
6
Kurtosis:                       3.896   Cond. No.                      2.09e+0
8
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 2.09e+08. This might indicate that there a
re
strong multicollinearity or other numerical problems.
==============================================================================
=
```




```
==============================================================================
Train RMSE: 0.5540609036146587
Test RMSE: 0.5489174951622746
==============================================================================
```

Out[83]: `<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x24ec5933760>`

Well.. scaling doesn't improve the fit, and without some work the errors are hard to interpret, so we will just leave scaling out of our final model.

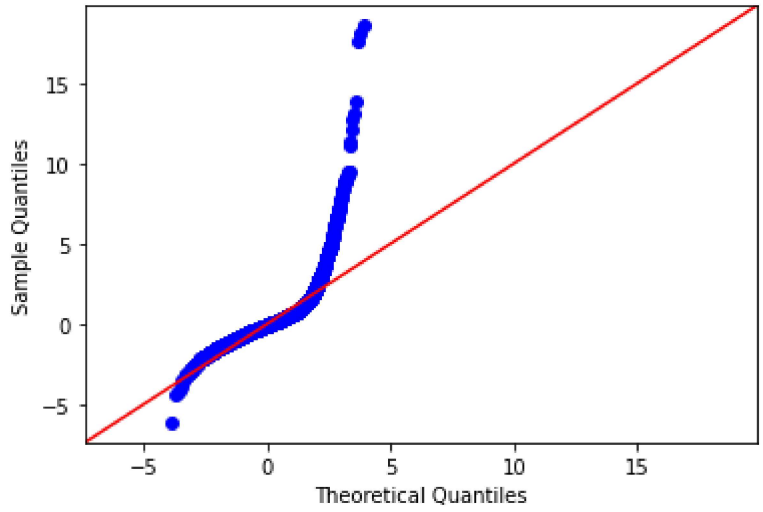
2.4.5 Last but not least... dropping columns that don't appear to have a linear relationship with price

Finally, there is always the requirement that there exists a linear relationship with price at all. Let's see if the model improves simply by dropping columns that didn't appear to have a linear relationship with price based on the correlation matrix and scatter plots. From our pairplot observations above, we found that `price` looks like it may have linear relationships with respect to `bedrooms`, `bathrooms`, `sqft_lot`, `grade`, `sqft_above`, `sqft_basement`, `sqft_living15`, `sqft_lot15`, and `lat`. Our correlation heatmap shows positive correlation > .3 with `bedrooms`, `bathrooms`, `sqft_above`, `sqft_basement`, `sqft_living15`, and `lat`. Thus, let's try fitting a model to a dataframe with only the ones in the previous sentence.

In [84]:
```
df6_only_visibly_linear = df6[['price','bedrooms','bathrooms','sqft_above','sqft_
evaluate_model(df6_only_visibly_linear,0)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.585
Model:                            OLS   Adj. R-squared:                  0.585
Method:                 Least Squares   F-statistic:                     4997.
Date:                Wed, 09 Jun 2021   Prob (F-statistic):               0.00
Time:                        11:41:38   Log-Likelihood:            -2.9347e+05
No. Observations:               21278   AIC:                         5.869e+05
Df Residuals:                   21271   BIC:                         5.870e+05
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept  -3.296e+07   5.6e+05    -58.869      0.000   -3.41e+07   -3.19e+07
bedrooms   -5.519e+04   2276.219   -24.248      0.000   -5.97e+04   -5.07e+04
bathrooms   1.105e+04   3274.476     3.375      0.001    4634.038    1.75e+04
sqft_above   263.0890      3.868    68.013      0.000     255.507     270.671
sqft_basement  281.2696      4.576    61.464      0.000     272.300     290.239
sqft_living15   59.6987      3.722    16.039      0.000      52.403      66.994
lat         6.937e+05   1.18e+04    58.936      0.000    6.71e+05    7.17e+05
==============================================================================
Omnibus:                    17300.058   Durbin-Watson:                   1.990
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1072328.932
Skew:                           3.480   Prob(JB):                         0.00
Kurtosis:                      37.074   Cond. No.                     9.92e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 9.92e+05. This might indicate that there are strong multicollinearity or other numerical problems.
==============================================================================
```




```
==============================================================================
Train RMSE: 235689.7266791706
Test RMSE: 238416.42634657904
==============================================================================
```

Out[84]: `<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x24ece76c9a0>`

Well... that's the worst yet, so I don't think we'll pursue this line of thinking further.

2.4.6 Building a final model

Woohoo! It's time to put all the steps together. To recap, we saw that the biggest improvements in our model fit came from: Log transforming our continuous variables, adding interactions to reduce the effects of multicollinearity, and casting categorical variables into dummy columns. Based on google, I should take create interaction terms first and then do log transforms last.

In [85]:
```
cat_candidates = ['floors', 'waterfront', 'view', 'grade', 'zipcode']
df7 = df6.copy()
for cat_candidate in cat_candidates:
    dummies = pd.get_dummies(df7[cat_candidate], prefix = cat_candidate, drop_fir
    col_to_drop = df7[cat_candidate].value_counts().index[0]
    dummies.drop(columns = f'{cat_candidate}_{col_to_drop}',inplace = True)
    df7 = pd.concat([df7,dummies], axis = 1)
    df7.drop(columns = cat_candidate, inplace = True)
evaluate_model(df7,0)
```

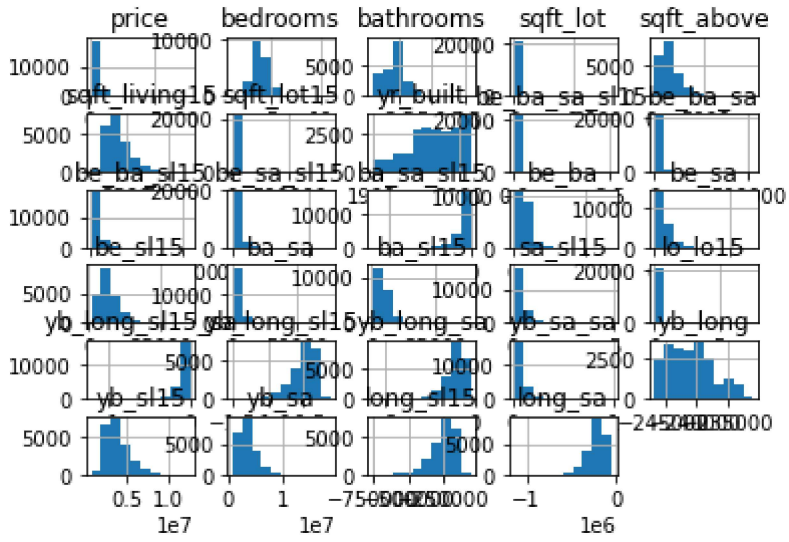
```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                   price   R-squared:                       0.83
4
Model:                             OLS   Adj. R-squared:                  0.83
3
Method:                  Least Squares   F-statistic:                     110
8.
Date:                 Wed, 09 Jun 2021   Prob (F-statistic):               0.0
0
Time:                         11:41:38   Log-Likelihood:            -2.8372e+0
5
No. Observations:               21278   AIC:                          5.676e+0
5
Df Residuals:                   21181   BIC:                          5.684e+0
5
Df Model:                          96
Covariance Type:            nonrobust
```

In [86]:
```
df8 = df7.copy()
df8['be_ba_sa_sl15'] = df8.bedrooms * df8.bathrooms * df8.sqft_above * df8.sqft_l
df8['be_ba_sa'] = df8.bedrooms * df8.bathrooms * df8.sqft_above
df8['be_ba_sl15'] = df8.bedrooms * df8.bathrooms * df8.sqft_living15
df8['be_sa_sl15'] = df8.bedrooms * df8.sqft_above * df8.sqft_living15
df8['ba_sa_sl15'] = df8.bathrooms * df8.sqft_above * df8.sqft_living15
df8['be_ba'] = df8.bedrooms * df8.bathrooms
df8['be_sa'] = df8.bedrooms * df8.sqft_above
df8['be_sl15'] = df8.bedrooms * df8.sqft_living15
df8['ba_sa'] = df8.bathrooms * df8.sqft_above
df8['ba_sl15'] = df8.bathrooms * df8.sqft_living15
df8['sa_sl15'] = df8.sqft_above * df8.sqft_living15
df8['lo_lo15'] = df8.sqft_lot * df8.sqft_lot15
df8['yb_long_sl15_sa'] =  df8.yr_built * df8.long * df8.sqft_living15 * df8.sqft_
df8['yb_long_sl15'] = df8.yr_built * df8.long * df8.sqft_living15
df8['yb_long_sa'] = df8.yr_built * df8.long * df8.sqft_above
df8['yb_sa_sa'] = df8.yr_built * df8.sqft_living15 * df8.sqft_above
df8['ba_sa_sl15'] = df8.long * df8.sqft_living15 * df8.sqft_above
df8['yb_long'] = df8.yr_built * df8.long
df8['yb_sl15'] = df8.yr_built * df8.sqft_living15
df8['yb_sa'] = df8.yr_built * df8.sqft_above
df8['long_sl15'] = df8.long * df8.sqft_living15
df8['long_sa'] = df8.long * df8.sqft_above
df8['lo_lo15'] = df8.sqft_lot * df8.sqft_lot15
evaluate_model(df8,0)
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  price   R-squared:                       0.85
7
Model:                            OLS   Adj. R-squared:                  0.85
6
Method:                 Least Squares   F-statistic:                     116
5.
Date:                Wed, 09 Jun 2021   Prob (F-statistic):               0.0
0
Time:                        11:41:39   Log-Likelihood:            -2.8212e+0
5
No. Observations:               21278   AIC:                         5.645e+0
5
Df Residuals:                   21168   BIC:                         5.653e+0
5
Df Model:                         109
Covariance Type:            nonrobust
```

In [87]:
```
for_log_transform = ['price','bedrooms','bathrooms','sqft_lot','sqft_above','sqft
                     'be_ba_sa_sl15', 'be_ba_sa', 'be_ba_sl15','be_sa_sl15', 'ba_s
                     'ba_sl15', 'sa_sl15', 'lo_lo15', 'yb_long_sl15_sa', 'yb_long_
                     'yb_sl15', 'yb_sa', 'long_sl15', 'long_sa']
```

In [88]:
```
df8[for_log_transform].hist();
```



In [89]:
```
df9 = df8.copy()
for column in for_log_transform:
    df9[column] = np.log(abs(df9[column]))
df9['long'] = abs(df9.long)
evaluate_model(df9,1)
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  price   R-squared:                       0.88
0
Model:                            OLS   Adj. R-squared:                  0.88
0
Method:                 Least Squares   F-statistic:                     160
5.
Date:                Wed, 09 Jun 2021   Prob (F-statistic):               0.0
0
Time:                        11:41:42   Log-Likelihood:                6040.
1
No. Observations:               21278   AIC:                         -1.188e+0
4
Df Residuals:                   21180   BIC:                         -1.110e+0
4
Df Model:                          97
Covariance Type:            nonrobust
```

Well... Compared to any of the other models this has the highest F Statistic and best R^2, and the best RMSEs. Now, to interpret what all the coefficients and errors actually mean so we can communicate them.

2.5 Interpret!

Okay, this is going to be a challenge to interpret. The first thing that we notice is log transforming wrecks the p-values of a lot of the things that obviously make a difference and were significant according the the p-value just before the transformation. It isn't logical that improving the overall R^2 and adjusted R^2, F statistic and RMSEs is undesirable just because all of a sudden many of our predictors have p-values near .2. For our overall model, the P(F) is still 0.0 indicating model significance. Our AIC and BIC values are also large and very negative, which is a good sign. The Durbin-Watson should be between 1-2, roughly, so a value just over 2 seems fine. The Cond. No is very high, but we expect that with so many terms that are obviously collinear. We can't just remove all the interactions that, a step ago, were significant and improved the model and so seem to be meaningful, plus domain knowledge and our correlation matrix suggests that these interactions exist. The most useful resource I have found is this one (https://stats.stackexchange.com/questions/65898/why-could-centering-independent-variables-change-the-main-effects-with-moderatio/65917#65917). Specifically, it gives this example for being able to interpret overall effects for the main predictors in models using interactions. For a model with interactions such as \begin{equation} y = b0 + b1*x1 + b2*x2 + b3*x3 + b4*x4 + b5*x5 + b12*x1*x2 + b13*x1*x3 + b23*x2*x3 + b45*x4*x5 + b123*x1*x2*x3 + e , \end{equation} to calculate the overall effects for x1-x5, B1-B5, you must calculate: \begin{equation} B1 = b1 + b12*mean(x2) + b13*mean(x3) + b123*mean(x2*x3), \end{equation} \begin{equation} B2 = b2 + b12*mean(x1) + b23*mean(x3) + b123*mean(x1*x3), \end{equation} \begin{equation} B3 = b3 + b13*mean(x1) + b23*mean(x2) + b123*mean(x1*x2), \end{equation} \begin{equation} B4 = b4 + b45*mean(x5), \end{equation} \begin{equation} B5 = b5 + b45*mean(x4). \end{equation}

So, let's try that. However, it is more complicated by the fact that we have included log transformation in the model. What's more is that we conducted the log transformations after including all the interaction terms and they themselves were log transformed. I have found another resource that discusses log transformations (https://journals.sagepub.com/doi/full/10.1177/1094428121991907). From the paper, "If nonnormality is not a problem for OLS regression analysis, what do transformations do then? Transformations model nonlinear effects..." suggesting that using a log transformation is only valuable if the underlying priniciple is non-linear instead of linear as we have hypothesized. While the log transformation does slightly improve our model and so we may surmise that there are non-linear effects, it is not so much better than our linear model that it is worth the headache in interpretation. While we may leave the log transformations in to actually construct a tool that we use to predict what a house should sell for, in communicating overall effects of our main predictors we will trying looking at the version of the model just prior to adding in the linear transformation. While we have the general rule that the variance of our residuals should be normally distributed, the source also suggests that non-normality in this distribution is not truly a problem,

> "The second assumption, normality of the error term, is only required for the proof that the t statistics used for calculating the p values of the regression coefficients and the F statistic that is used for the overall model test follow their reference distributions in small samples (Wooldridge, 2013, Theorem 4.2). But this assumption is mostly irrelevant for applied research because the large sample behavior of OLS regression, which does not depend on the normality of the error term, starts to kick in at sample sizes well under 100 (Wooldridge, 2013, Section 5.2)."

All things considered, we will attempt the above calculations with the second to last version of our model, calculated from `df8` . Let's reprint the df8 summary for easy reference. Let's also remember that, for each coefficient, that is the unit increase in price with a unit increase in that coefficient WITH ALL OTHER VARIABLES HELD CONSTANT.

In [90]:
```
final_model_df8 = evaluate_model(df8,0)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.85
7
Model:                            OLS   Adj. R-squared:                  0.85
6
Method:                 Least Squares   F-statistic:                     116
5.
Date:                Wed, 09 Jun 2021   Prob (F-statistic):               0.0
0
Time:                        11:41:42   Log-Likelihood:            -2.8212e+0
5
No. Observations:               21278   AIC:                         5.645e+0
5
Df Residuals:                   21168   BIC:                         5.653e+0
5
Df Model:                         109
Covariance Type:            nonrobust
```

In [91]:
```
final_model_df8.params.bathrooms
```

Out[91]:
```
-140214.1798080384
```

In [92]:
```
overall_bed = final_model_df8.params.bedrooms + (final_model_df8.params.be_ba_sa_
    (final_model_df8.params.be_ba_sl15 * np.mean(df8.be_ba_sl15)) + (final_model_
    (final_model_df8.params.be_sa * np.mean(df8.be_sa)) + (final_model_df8.params
overall_bath = final_model_df8.params.bathrooms + (final_model_df8.params.be_ba_s
    (final_model_df8.params.be_ba_sl15 * np.mean(df8.be_ba_sl15)) + (final_model_
    (final_model_df8.params.ba_sa * np.mean(df8.ba_sa)) + (final_model_df8.params
overall_sqft_lot = final_model_df8.params.sqft_lot + (final_model_df8.params.lo_l
# overall_sqft_above =
# overall_yr =
# overall_long =
# overall_sl15 =
# overall_lo15 =
```

In [93]:
```
print(overall_bed)
print(overall_bath)
print(overall_sqft_lot)
```

```
-117177.24649400126
1617281250.8791442
-389.02980512119257
```

Well.. this doesn't seem to be working at all. You definitely aren't increasing in price per bathroom

by over a billion dollars. Sigh... I guess it is because it isn't that logical to keep everything exactly the same and suddenly have an extra bathroom, or bedroom, say. Let's try making some predictions using our model built on df9 to see if at least they seem reasonable. From an ordered list (or list of lists) containing the basic listing info, we'll create a listing dataframe, i.e. a dataframe with all the main predictors that you could get from a house listing online. Then, from that we will construct a dataframe with all of the predictor columns as above. From there, we should be able to use our model's predict method to predict a price for our fake listings.

In [94]:
```
final_model_df9 = evaluate_model(df9,1)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.88
0
Model:                            OLS   Adj. R-squared:                  0.88
0
Method:                 Least Squares   F-statistic:                     160
5.
Date:                Wed, 09 Jun 2021   Prob (F-statistic):               0.0
0
Time:                        11:41:43   Log-Likelihood:                6040.
1
No. Observations:               21278   AIC:                         -1.188e+0
4
Df Residuals:                   21180   BIC:                         -1.110e+0
4
Df Model:                          97
Covariance Type:            nonrobust
```

To get the easy stuff out of the way for this model before making predictions, let's quickly discuss the categorical columns. Okay, so for `floors`, floor_1 was dropped, for `waterfront`, `waterfront_0` was dropped, for `grade`, `grade_7` was dropped, `view_0` was dropped for `view`, and there was one dropped for `zipcode` but I don't really have enough information about all the zipcodes to make much of a statement about them. Based on our coefficients, we can see that, relative to the variable that was dropped:

1. There is less of a unit increase in price for more than one floor when compared to just having a floor at all (silly to say, but it's just important that the house exists).
2. The price of a home increases radically for a home that has waterfront property.
3. Relative to the most common grade of 7, houses graded less sell for less and houses graded for more sell for more in a way that is relative to their grade. Grade 3 is an exception, but there is only one house with grade 3, so it is not that reliable.
4. Houses with a view experience positive increases in unit increase in price per increase in view, scaling as the rating of the view increases.
5. Homes in some zipcodes sell for more or less relative to the zipcode with the most houses in it. We would have to do further investigation on the location of these zipcodes to make sense of this information.

Next, we can quickly comment on the RMSE. We see that, on average, a prediction using our model is off in either direction by up to 130,000. Looking at our qq plot, we can theorize that the RMSE would be much lower, but the distribution of our residuals is very heavy in both tails,

meaning that we have a lot of values in the extremes. What if we quickly look at the middle 70 percent of homes, price-wise.

In [95]:
```
lower_quantile, upper_quantile = df9.price.quantile([.1, .90])
df10 = df9.loc[(df9.price > lower_quantile) & (df9.price < upper_quantile)]
```

In [96]:
```
df10.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16999 entries, 1 to 21277
Columns: 118 entries, price to long_sa
dtypes: float64(32), uint8(86)
memory usage: 5.7 MB
```

In [97]:
```
evaluate_model(df10,1)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.80
0
Model:                            OLS   Adj. R-squared:                  0.79
9
Method:                 Least Squares   F-statistic:                     702.
7
Date:                Wed, 09 Jun 2021   Prob (F-statistic):               0.0
0
Time:                        11:41:43   Log-Likelihood:                 8052.
9
No. Observations:               16999   AIC:                         -1.591e+0
4
Df Residuals:                   16902   BIC:                         -1.516e+0
4
Df Model:                          96
Covariance Type:            nonrobust
```

So, as theorized, the RMSE for the homes without some of the extreme price values is much lower. We still have a decent model here as well, with an adjusted R^2 of .799. It would be nice in the future to be produce errors relative to the predicted selling price of the home.

Okay, so back to seeing what types of our predictions our model will make for novel data, and then varying them slightly around a point to see what the local slopes look like. First, we need to build a function to take in the general information (non-logged) you may find in a standard home listing, then add all the columns necessary to put it into our model predictor.

In [98]:
```
def construct_predict_input(listing_info,model_df,log_model_bool):
    """Takes in a list of lists, where each list is a collection of regular
    values that could be easily found in a home for sale listing, then expands
    this information into a format that can be passed into a fit() model predict(

    # Take the list of lists and convert it into a pandas dataframe
    basic_columns = ['bedrooms','bathrooms','sqft_lot','sqft_above','sqft_basemer
                     'lat','long','sqft_living15','sqft_lot15']
    cat_columns = ['floors','waterfront','view','grade','zipcode']
    listing_df = pd.DataFrame(data = listing_info,columns = basic_columns + cat_c

    # initiate dataframe with same dimensions as df model was built from
    expanded_listing_df = pd.DataFrame(data = 0, index = range(len(listing_info))
    expanded_listing_df.drop(columns = 'price', inplace = True)

    # replace values in expanded frame with those from listing frame
    # for cat columns, check if it wasn't the one dropped when dummy-encoding
    for column in basic_columns:
        expanded_listing_df[column] = listing_df[column]
    for cat_column in cat_columns:
        dummy_col = f'{cat_column}_{listing_df[cat_column].iloc[0]}'
        if dummy_col in expanded_listing_df.columns:
            expanded_listing_df[dummy_col] = 1

    # add interaction columns from listing values
    expanded_listing_df['be_ba_sa_sl15'] = expanded_listing_df.bedrooms * expande
    expanded_listing_df['be_ba_sa'] = expanded_listing_df.bedrooms * expanded_lis
    expanded_listing_df['be_ba_sl15'] = expanded_listing_df.bedrooms * expanded_l
    expanded_listing_df['be_sa_sl15'] = expanded_listing_df.bedrooms * expanded_l
    expanded_listing_df['ba_sa_sl15'] = expanded_listing_df.bathrooms * expanded_
    expanded_listing_df['be_ba'] = expanded_listing_df.bedrooms * expanded_listir
    expanded_listing_df['be_sa'] = expanded_listing_df.bedrooms * expanded_listir
    expanded_listing_df['be_sl15'] = expanded_listing_df.bedrooms * expanded_list
    expanded_listing_df['ba_sa'] = expanded_listing_df.bathrooms * expanded_listi
    expanded_listing_df['ba_sl15'] = expanded_listing_df.bathrooms * expanded_lis
    expanded_listing_df['sa_sl15'] = expanded_listing_df.sqft_above * expanded_li
    expanded_listing_df['lo_lo15'] = expanded_listing_df.sqft_lot * expanded_list
    expanded_listing_df['yb_long_sl15_sa'] =  expanded_listing_df.yr_built * expa
    expanded_listing_df['yb_long_sl15'] = expanded_listing_df.yr_built * expanded
    expanded_listing_df['yb_long_sa'] = expanded_listing_df.yr_built * expanded_l
    expanded_listing_df['yb_sa_sa'] = expanded_listing_df.yr_built * expanded_lis
    expanded_listing_df['ba_sa_sl15'] = expanded_listing_df.long * expanded_listi
    expanded_listing_df['yb_long'] = expanded_listing_df.yr_built * expanded_list
    expanded_listing_df['yb_sl15'] = expanded_listing_df.yr_built * expanded_list
    expanded_listing_df['yb_sa'] = expanded_listing_df.yr_built * expanded_listir
    expanded_listing_df['long_sl15'] = expanded_listing_df.long * expanded_listir
    expanded_listing_df['long_sa'] = expanded_listing_df.long * expanded_listing_
    expanded_listing_df['lo_lo15'] = expanded_listing_df.sqft_lot * expanded_list

    # if putting this into a model where everything was log transformed, need to
    # so the model will make predictions that are real
    if log_model_bool:
        for_log_transform = ['bedrooms','bathrooms','sqft_lot','sqft_above','sqft
                     'be_ba_sa_sl15', 'be_ba_sa', 'be_ba_sl15','be_sa_sl15', 'ba_s
                     'ba_sl15', 'sa_sl15', 'lo_lo15', 'yb_long_sl15_sa', 'yb_long_
                     'yb_sl15', 'yb_sa', 'long_sl15', 'long_sa']
```

```
        expanded_listing_df[for_log_transform] = expanded_listing_df[for_log_tran

    return expanded_listing_df
```

Okay, now let's make a single fake record based on the median data from our non-log-transformed dataset. For the categorical variables, we are using the ones that were dropped when encoding as the standard.

In [99]:
```
listing_info = [[df8.bedrooms.median(), df8.bathrooms.median(), df8.sqft_lot.medi
                 df8.yr_built.median(), df8.lat.median(), abs(df8.long.median()),
                 1,0,0,7,98103]]
for item in listing_info:
    print(item)
one_listing_test = construct_predict_input(listing_info,df9,1)
```

```
[3.0, 2.25, 7627.5, 1560.0, 0.0, 1975.0, 47.5714, 122.229, 1840.0, 7628.0, 1,
0, 0, 7, 98103]
```

Double check that there are no NaNs, otherwise the model predictor will return NaN

In [100]:
```
one_listing_test.loc[:, one_listing_test.isna().any()]
```

Out[100]:

0

In [101]:
```
one_listing_test.head()
```

Out[101]:

	bedrooms	bathrooms	sqft_lot	sqft_above	sqft_basement	yr_built	lat	long	sqft_li
0	1.098612	0.81093	8.939515	7.352441	0.0	7.588324	47.5714	122.229	7.5

1 rows × 117 columns

In [102]:
```
price_pred = final_model_df9.predict(one_listing_test)
print(np.exp(price_pred))
```

```
0    559298.774353
dtype: float64
```

In [103]:
```
np.mean(df[(df.bedrooms == 3) & (df.grade == 7) & (df.zipcode == 98103)].price)
```

Out[103]:
```
588088.8870967742
```

Okay, so our model can make predictions, at least. While the median price is not expected to be exactly equal to a house constructed from all median values, it is encouraging that it is really close. Now, to interpret our model, let's see if we make some changes, what happens to the price.

In [104]:
```
listing_info = [[df8.bedrooms.median() + 1, df8.bathrooms.median(), df8.sqft_lot.
                 df8.yr_built.median(), df8.lat.median(), abs(df8.long.median()),
                 1,0,0,7,98103]]
one_listing_test = construct_predict_input(listing_info,df9,1)
price_pred = final_model_df9.predict(one_listing_test)
print(np.exp(price_pred))
```

```
0    554796.006579
dtype: float64
```

Well, as you can see there is a negative impact by increasing the number of bedrooms. Seeing as everything else would be the same, that is likely because the additional bedroom affords less liveable space to the other rooms, making the house more cramped. So, while it feels strange, it does sort of make sense.

In [105]:
```
listing_info = [[df8.bedrooms.median(), df8.bathrooms.median() + .5, df8.sqft_lot
                 df8.yr_built.median(), df8.lat.median(), abs(df8.long.median()),
                 1,0,0,7,98103]]
one_listing_test = construct_predict_input(listing_info,df9,1)
price_pred = final_model_df9.predict(one_listing_test)
print(np.exp(price_pred))
```

```
0    570099.523592
dtype: float64
```

Bathrooms increase the price.

In [106]:
```
listing_info = [[df8.bedrooms.median(), df8.bathrooms.median(), df8.sqft_lot.medi
                 df8.yr_built.median(), df8.lat.median(), abs(df8.long.median()),
                 1,0,0,8,98103]]
one_listing_test = construct_predict_input(listing_info,df9,1)
price_pred = final_model_df9.predict(one_listing_test)
print(np.exp(price_pred))
```

```
0    607850.044377
dtype: float64
```

Increasing the grade increases the price.

In [107]:
```
listing_info = [[df8.bedrooms.median(), df8.bathrooms.median(), df8.sqft_lot.medi
                 df8.yr_built.median(), df8.lat.median(), abs(df8.long.median()),
                 1,0,0,7,98103]]
one_listing_test = construct_predict_input(listing_info,df9,1)
price_pred = final_model_df9.predict(one_listing_test)
print(np.exp(price_pred))
```

```
0    621612.810314
dtype: float64
```

Increasing the sqft_above by 500 sqft increases the price greatly. So now that we know this is a pretty effective way of interpreting our effects and we feel like the predictions we are getting are reasonable, let's try plotting predicted price while varying each variable individually over a small range around the median.

In [108]:
```
median_listing = [df8.bedrooms.median(), df8.bathrooms.median(), df8.sqft_lot.mec
                  df8.yr_built.median(), df8.lat.median(), abs(df8.long.median()),
                  1,0,0,7,98103]
varying_beds = []
bed_count = np.linspace(start = 2,stop = 8,num = 7)
for beds in bed_count:
    median_listing[0] = np.log(beds)
    new_row = median_listing.copy()
    varying_beds.extend([new_row])
```

In [109]:
```
beds_df = construct_predict_input(varying_beds,df9,1)
beds_df['price_pred'] = final_model_df9.predict(beds_df)
np.exp(beds_df.price_pred)
```

Out[109]:
```
0    582804.063536
1    575310.654873
2    571563.122183
3    569171.198459
4    567457.564854
5    566143.163958
6    565088.370875
Name: price_pred, dtype: float64
```

In [110]:
```
np.exp(beds_df.price_pred[5]) - np.exp(beds_df.price_pred[4])
```

Out[110]:
```
-1314.4008959506173
```

Okay, so here we are finally getting an interpretable result. Based on how our model predicts price, if you could somehow keep everything exactly the same and add an extra bedroom, you would expect the price to decrease. However, how much it decreases depends on how many bedrooms there already are. Let's do the same variation analysis for all our main predictors.

In [111]:
```
def vary_predictors(index_to_vary, vary_amt, parent_df):
    median_listing = [parent_df.bedrooms.median(), parent_df.bathrooms.median(),
                parent_df.yr_built.median(), parent_df.lat.median(), abs(parent_d
                1,0,0,7,98103]
    test_vals = np.linspace(start = median_listing[index_to_vary] - (vary_amt * 2
                            stop = median_listing[index_to_vary] + (vary_amt * 2)
                           num = 5)
    varied_listings = []
    print(test_vals)
    for val in test_vals:
        median_listing[index_to_vary] = val
        new_row = median_listing.copy()
        varied_listings.extend([new_row])
    df_in = construct_predict_input(varied_listings,parent_df,1)
    df_in['price_pred'] = np.exp(final_model_df9.predict(df_in))
    return df_in['price_pred']
```

In [112]:
```
basic_columns = ['bedrooms','bathrooms','sqft_lot','sqft_above','sqft_basement',
                 'lat','long','sqft_living15','sqft_lot15']
vary_amounts = [1,.5,100,100,100,5,.01,.01,100,100]
print('====================================')
for idx, col in enumerate(basic_columns):
    print(f'Varying Column: {col}')
    prices_out = vary_predictors(idx,vary_amounts[idx],df8)
    print(prices_out)
print('====================================')
```

```
====================================
Varying Column: bedrooms
[1. 2. 3. 4. 5.]
0    576832.961401
1    565707.188789
2    559298.774352
3    554796.006575
4    551328.367757
Name: price_pred, dtype: float64
Varying Column: bathrooms
[1.25 1.75 2.25 2.75 3.25]
0    528825.393079
1    546060.356099
2    559298.774352
3    570099.523588
4    579249.838077
Name: price_pred, dtype: float64
Varying Column: sqft_lot
[7427.5 7527.5 7627.5 7727.5 7827.5]
0    558312.061746
1    558808.477394
2    559298.774352
3    559783.106789
4    560261.623055
Name: price_pred, dtype: float64
Varying Column: sqft_above
[1360. 1460. 1560. 1660. 1760.]
0    530889.724709
1    545396.141264
2    559298.774352
3    572659.182127
4    585529.369701
Name: price_pred, dtype: float64
Varying Column: sqft_basement
[-200. -100.    0.  100.  200.]
0    545148.806874
1    552178.467094
2    559298.774352
3    566510.897532
4    573816.020593
Name: price_pred, dtype: float64
Varying Column: yr_built
[1965. 1970. 1975. 1980. 1985.]
0    562947.823496
1    561118.017263
2    559298.774352
3    557490.007275
```

```
4    555691.629473
Name: price_pred, dtype: float64
Varying Column: lat
[47.5514 47.5614 47.5714 47.5814 47.5914]
0    553785.031205
1    556535.074553
2    559298.774352
3    562076.198419
4    564867.414906
Name: price_pred, dtype: float64
Varying Column: long
[122.209 122.219 122.229 122.239 122.249]
0    555446.667932
1    557376.743541
2    559298.774352
3    561212.633487
4    563118.194310
Name: price_pred, dtype: float64
Varying Column: sqft_living15
[1640. 1740. 1840. 1940. 2040.]
0    549779.247965
1    554655.449014
2    559298.774352
3    563732.137660
4    567975.154374
Name: price_pred, dtype: float64
Varying Column: sqft_lot15
[7428. 7528. 7628. 7728. 7828.]
0    559484.847568
1    559391.185269
2    559298.774352
3    559207.582019
4    559117.576741
Name: price_pred, dtype: float64
====================================
```

Sweet! Now, for a certain house, we can see how the price would change if we varied one of the variables. We can use this to look at how much a home might sell after renovation.. we can predict a price with pre-renovation values, then predict again with the new number of bathrooms, bedrooms, grade, etc.

In [113]:
```
cat_columns = ['floors','waterfront','view','grade'] # without zip because that t
vary_amounts = [1,1,1,1]
print('====================================')
for idx, col in enumerate(cat_columns):
    print(f'Varying Column: {col}')
    prices_out = vary_predictors(idx+len(basic_columns),vary_amounts[idx],df8)
    print(prices_out)
print('====================================')
```

```
====================================
Varying Column: floors
[-1.  0.  1.  2.  3.]
0    559298.774352
1    559298.774352
2    559298.774352
3    559298.774352
4    559298.774349
Name: price_pred, dtype: float64
Varying Column: waterfront
[-2. -1.  0.  1.  2.]
0    559298.774352
1    559298.774352
2    559298.774352
3    559298.774352
4    559298.774349
Name: price_pred, dtype: float64
Varying Column: view
[-2. -1.  0.  1.  2.]
0    559298.774352
1    559298.774352
2    559298.774352
3    559298.774352
4    559298.774349
Name: price_pred, dtype: float64
Varying Column: grade
[5. 6. 7. 8. 9.]
0    559298.774352
1    559298.774352
2    559298.774352
3    559298.774352
4    559298.774349
Name: price_pred, dtype: float64
====================================
```

In [114]:
```
listing_info = [[df8.bedrooms.median(), df8.bathrooms.median(), df8.sqft_lot.medi
                 df8.yr_built.median(), df8.lat.median(), abs(df8.long.median()),
                 1,0,0,7,98103]]
one_listing_test = construct_predict_input(listing_info,df8,1)
price_pred = final_model_df9.predict(one_listing_test)
print(np.exp(price_pred))
```

```
0    559298.774353
dtype: float64
```

In [115]:
```
listing_info = [[df8.bedrooms.median(), df8.bathrooms.median(), df8.sqft_lot.medi
                 df8.yr_built.median(), df8.lat.median(), abs(df8.long.median()),
                 1,0,0,8,98103]]
one_listing_test = construct_predict_input(listing_info,df8,1)
price_pred = final_model_df9.predict(one_listing_test)
print(np.exp(price_pred))
```

```
0    607850.044377
dtype: float64
```

In [116]:
```
listing_info = [[df8.bedrooms.median(), df8.bathrooms.median(), df8.sqft_lot.medi
                 df8.yr_built.median(), df8.lat.median(), abs(df8.long.median()),
                 1,0,0,9,98103]]
one_listing_test = construct_predict_input(listing_info,df8,1)
price_pred = final_model_df9.predict(one_listing_test)
print(np.exp(price_pred))
```

```
0    683911.842357
dtype: float64
```

I guess I'm not sure why vary_predictors() didn't work, when I can do the steps manually and see price prediction changes. Oh well, we can just report the categoricals qualitatively bsaed on their ocefficients.

3 Conclusion

In the end, we produced a Multiple Linear Regression model with an adjusted R^2 = .88 and a Prob(F-stat) < .001. Thus, we can be fairly confident in the ability of our model to produce solid home pricing estimates. With confidence in our model, we built an estimator that takes in the information you may easily find in a home for sale listing, converts it to have all the attributes needed to be input into our model predictor, and returns an approximate price for any given home. Note that this estimator is very specific to King's County, WA. Based on our model, we can make some general statements. Keeping all other predictors the same, for a unit increase (add 1) in:

1. bedrooms, the price of the home will decrease
2. bathrooms, the price of the home will increase
3. sqft_lot, the price will increase slightly
4. sqft_above, the price will increase strongly
5. sqft_basement, the price will increase
6. yr_built, for each year the home is newer the price decreases slightly
7. latitude, for each tenth of a degree northward the price will increase
8. longitude, for each tenth of a degree westward the price will increase
9. sqft_living15, for each extra 100 sqft on average the 15 homes nearby have the price of your home will increase
10. sqft_lot15, for each sqft of land your 15 nearest neighbors have on average, your home price decreases, but only very slightly

Therefore, we can assert that the most important factors in determining our housing price are: that is has bedrooms, but only as many as necessary so that they don't take up additional living space; That it has as many bathrooms as is reasonable, such that people do not have to share; that the home is large and has a liveable basement, that the home is older and thus probably built closer to the city center; that your home is located as far west and as far north as possible, again because it places you closest to waterfront property and closest to the city center / industrial centers. The lot size does not matter so much, likely because expensive homes nearer the city have less lot space while lower priced homes further away have more lot space.

Finally, for the categorical columns, we can say qualitatively:

1. There is less of a unit increase in price for more than one floor when compared to just having a floor at all (silly to say, but it's just important that the house exists).
2. The price of a home increases radically for a home that has waterfront property.
3. Relative to the most common grade of 7, houses graded less sell for less and houses graded for more sell for more in a way that is relative to their grade. Grade 3 is an exception, but there is only one house with grade 3, so it is not that reliable.
4. Houses with a view experience positive increases in unit increase in price per increase in view, scaling as the rating of the view increases.
5. Homes in some zipcodes sell for more or less relative to the zipcode with the most houses in it. We would have to do further investigation on the location of these zipcodes to make sense of this information.