Housing Analysis

Author: Eric Wehmueller

Overview

This project is the second project for Flatiron School's bootcamp program in Data Science. We are being placed into a hypothetical situation as a Data Scientist and hoping to provide value to our business for the scenario we are given.

Business Problem

I have been hired by a real estate agency that helps homeowners sell homes. For this project, I am to provide expected/estimated home prices to homeowners based on the logistics of their home. This can also give insight on how home renovations might increase the estimated value of their homes, and what type of potential renovations are best.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import scipy.stats as stats
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from statsmodels.formula.api import ols
from sklearn.preprocessing import StandardScaler
from statsmodels.stats.outliers_influence import varianc
```

Data Investigation and Cleaning

To start, we have access to the King County House Sales dataset. Let's take a look at this to get a feel for what our starting point is and what raw data we have to work with.

```
df_original = pd.read_csv("data\kc_house_data.csv")
```

```
df_original.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
```

```
df_original.head(10)
```

	id	date	price	bedrooms	bathroo
0	7129300520	10/13/2014	221900.0	3	1.00
1	6414100192	12/9/2014	538000.0	3	2.25
2	5631500400	2/25/2015	180000.0	2	1.00
3	2487200875	12/9/2014	604000.0	4	3.00
4	1954400510	2/18/2015	510000.0	3	2.00
5	7237550310	5/12/2014	1230000.0	4	4.50
6	1321400060	6/27/2014	257500.0	3	2.25
7	2008000270	1/15/2015	291850.0	3	1.50
8	2414600126	4/15/2015	229500.0	3	1.00
9	3793500160	3/12/2015	323000.0	3	2.50

10 rows × 21 columns

Per the project description, I will be ignoring the following features: date, view, sqft_above, sqft_basement, yr_renovated, zipcode, lat, long, sqft_living15, sqft_lot15. For the time being, I am trying to make my modeling phase in this project as simple as possible.

```
df_col_drops = df_original.drop(columns=['id', 'date', '
display(df_col_drops)
```

	price	bedrooms	bathrooms	sqft_living	sq
0	221900.0	3	1.00	1180	565
1	538000.0	3	2.25	2570	724
2	180000.0	2	1.00	770	100
3	604000.0	4	3.00	1960	500
4	510000.0	3	2.00	1680	808
...	...	...	...	...	...
21592	360000.0	3	2.50	1530	113
21593	400000.0	4	2.50	2310	581
21594	402101.0	2	0.75	1020	135
21595	400000.0	3	2.50	1600	238
21596	325000.0	2	0.75	1020	107

21597 rows × 10 columns

```
df_col_drops.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   price        21597 non-null  float64
 1   bedrooms     21597 non-null  int64  
 2   bathrooms    21597 non-null  float64
 3   sqft_living  21597 non-null  int64  
 4   sqft_lot     21597 non-null  int64  
 5   floors       21597 non-null  float64
 6   waterfront   19221 non-null  float64
 7   condition    21597 non-null  int64  
 8   grade        21597 non-null  int64  
 9   yr_built     21597 non-null  int64  
dtypes: float64(4), int64(6)
memory usage: 1.6 MB
```

Waterfront appears to have ~2000 null values. Let's investigate what values are in this column to see what we can do about the null values.

Which ones are the most important features?

```
df_col_drops.waterfront.value_counts()
```

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

Only 146 have a waterfront view. Since this is a binary-filled column, I believe we can fill in all NaNs with a zero value. This makes sense, as NaNs almost certainly denotes the absence of a waterfront view.

```
df_col_drops.waterfront.fillna(0, inplace=True)
display(df_col_drops.head())
```

	price	bedrooms	bathrooms	sqft_living	sqft_lo
0	221900.0	3	1.00	1180	5650
1	538000.0	3	2.25	2570	7242
2	180000.0	2	1.00	770	10000
3	604000.0	4	3.00	1960	5000
4	510000.0	3	2.00	1680	8080

```
df_col_drops.describe()
```

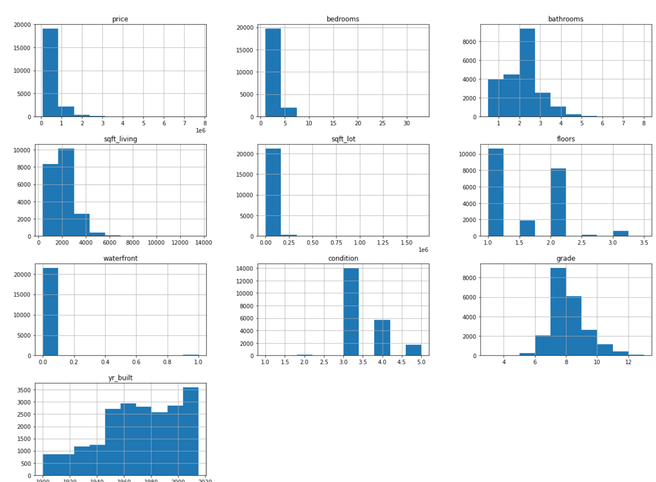
	price	bedrooms	bathrooms	sqft_liv
count	2.159700e+04	21597.000000	21597.000000	21597.00(
mean	5.402966e+05	3.373200	2.115826	2080.321ε
std	3.673681e+05	0.926299	0.768984	918.10612
min	7.800000e+04	1.000000	0.500000	370.0000(
25%	3.220000e+05	3.000000	1.750000	1430.000(
50%	4.500000e+05	3.000000	2.250000	1910.000(
75%	6.450000e+05	4.000000	2.500000	2550.000(
max	7.700000e+06	33.000000	8.000000	13540.00(

```
df_col_drops.columns
```

```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 's
qft_lot', 'floors',
       'waterfront', 'condition', 'grade', 'yr_built'],
      dtype='object')
```

```
#iterating over all columns except id to see general dis

df_col_drops.hist(figsize = (20,15));
```



It appears that we have some outliers in this data, so it's a little difficult to get a sense for what some the distrubutions actually are.

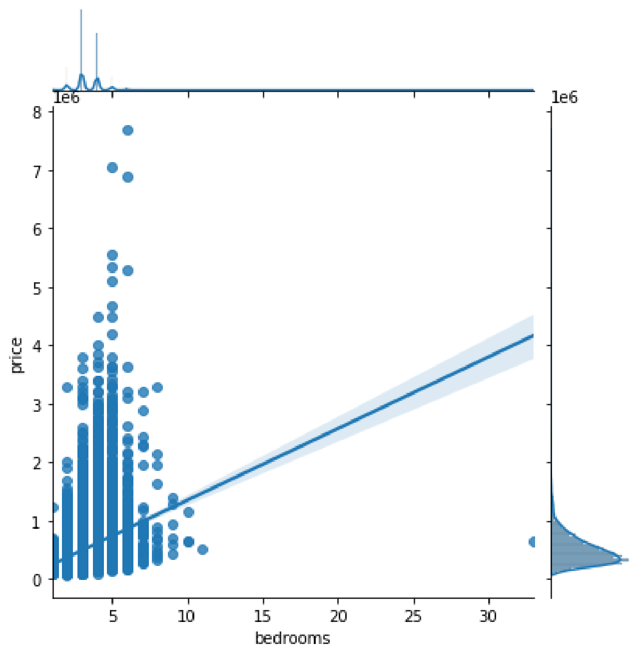
Specifically, I'm seeing a single entry priced at 7.7 million.

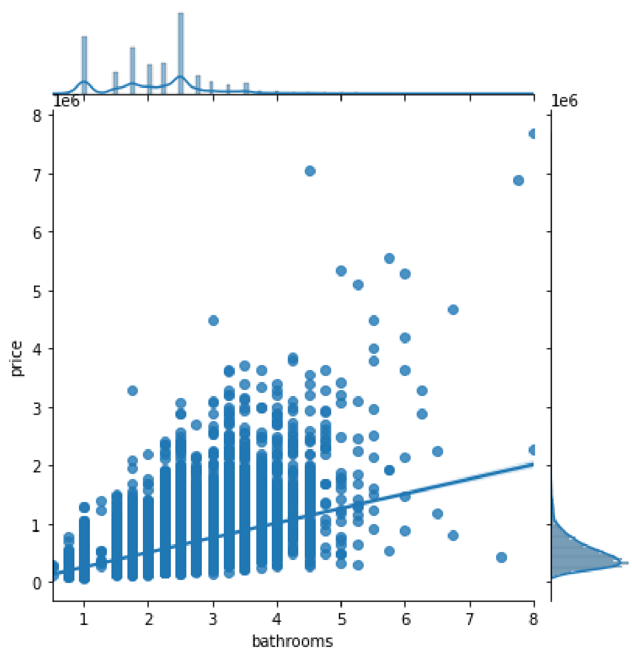
I also can't really tell what the bedroom distribution is with an outlier of 33.

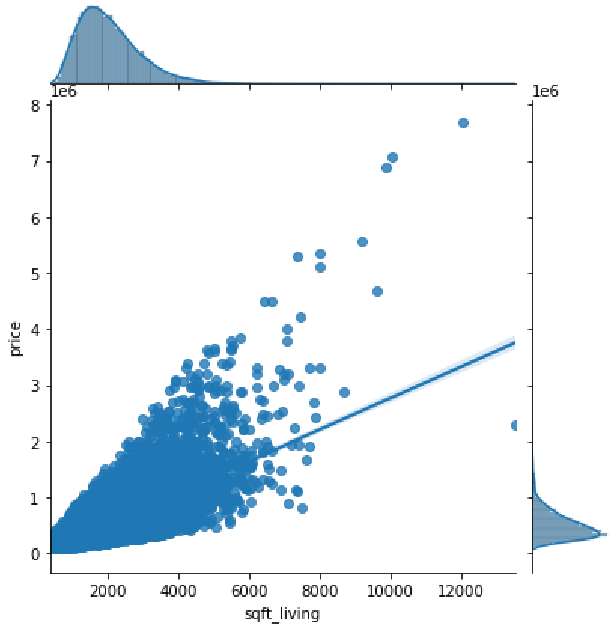
sqft_lot has only a single column in this view and the mean is vastly different from the median. We will need to take a closer look at this as well.

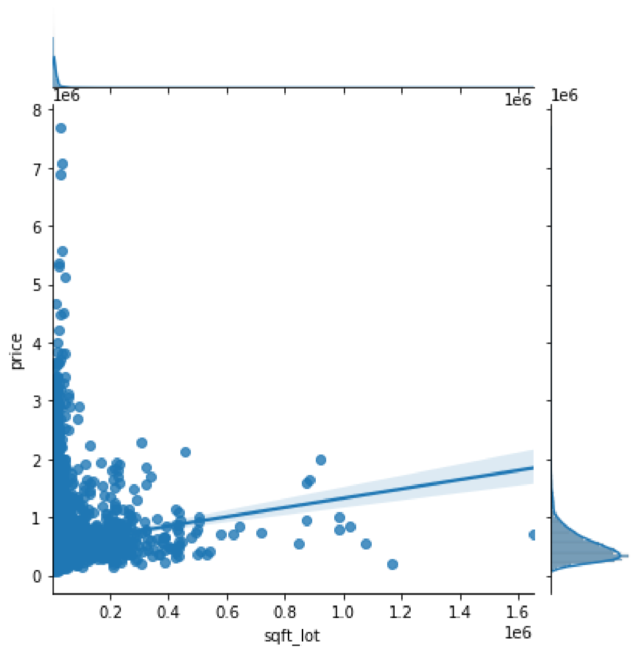
Condition and grade seem to be relatively normal.

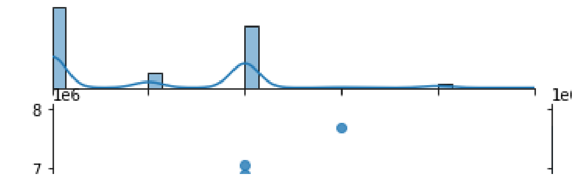
```
#Check for linearity via jointplots
for col_name in df_col_drops.columns[1:]:
    sns.jointplot(x=col_name, y='price', data=df_col_dro
```

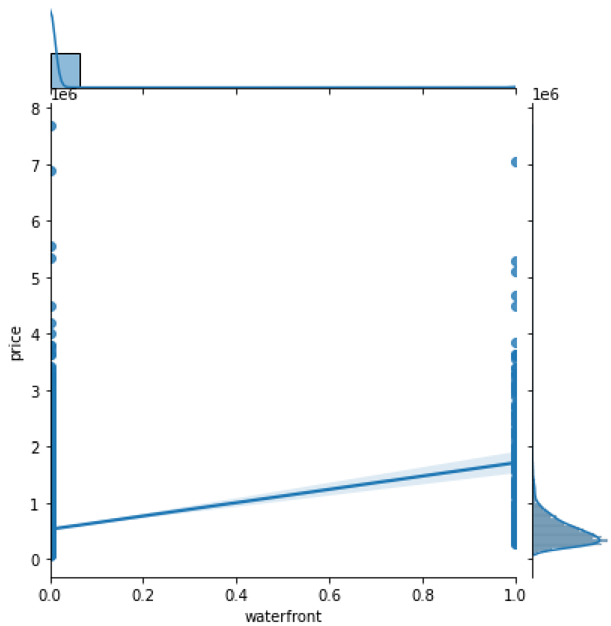


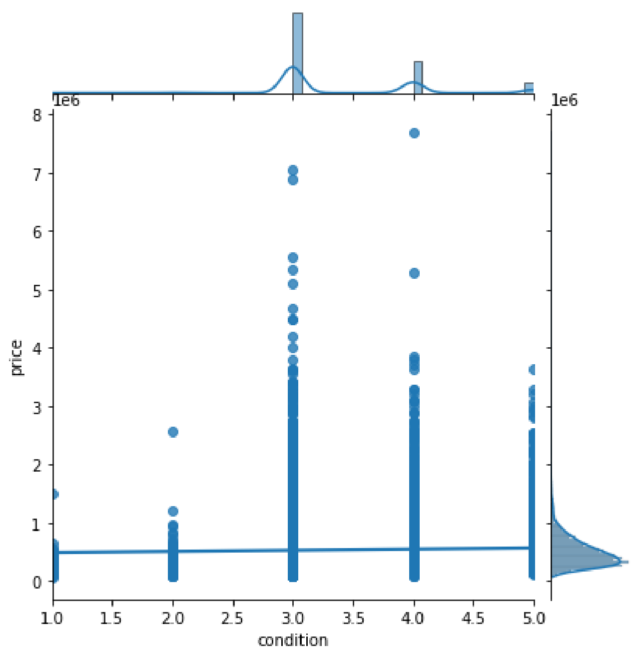


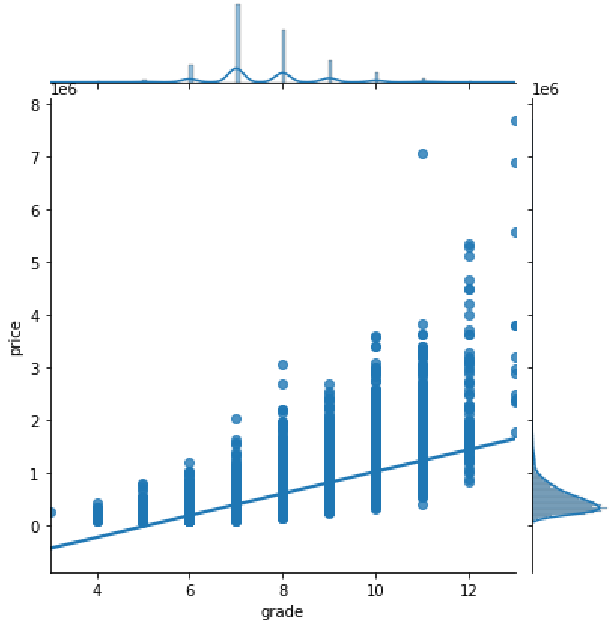

1e6
8
7
6
5
4
3
2
1
0
price
2000
4000
6000
8000
10000
12000
sqft_living
1e6

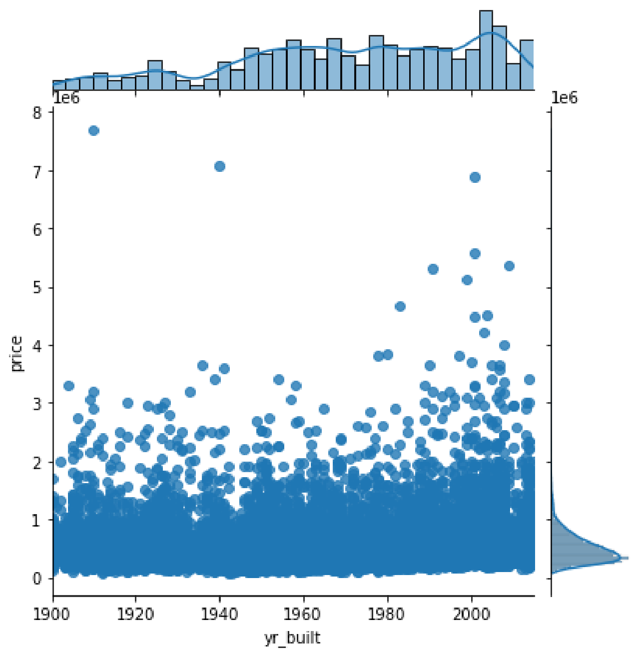

1e6
1e6
1e6
8
7
6
5
4
3
2
1
0
price
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1e6
sqft_lot


1e6
1e6
8
7


1e6
1e6
price
0
1
2
3
4
5
6
7
8
0.0
0.2
0.4
0.6
0.8
1.0
waterfront


1e6
1e6
price
0
1
2
3
4
5
6
7
8
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
condition





It worth noting that these jointplots reveal several of these columns to have linear relations with price.

Strong Linear Relation: sqft_living, grade

Somehwat Linear: bathrooms, sqft_lot, waterfront

Little to No Linear Relation: bedrooms, floors, condition, yr_built

It appears that the features that have the largest impact on the price of a home are the square footage of the home, as well as the Grade- this rating is given by the King County Housing System. I have copied this system below for more context.

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

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

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

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

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

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

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

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

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

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

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

Modeling

Model 1

```
outcome = 'price'
x_cols = list(df_col_drops.columns)
x_cols.remove(outcome)
print(x_cols)
```

```
['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floo
rs', 'waterfront', 'condition', 'grade', 'yr_built']
```

```
train, test = train_test_split(df_col_drops)
```

```
for col in x_cols:
    train[col] = (train[col] - train[col].mean())/train[
display(train.head())
print(len(train), len(test))
```

```
<ipython-input-15-f07e438ec62e>:2: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a D
ataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.o
rg/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers
us-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/i
ndexing.html#returning-a-view-versus-a-copy)
  train[col] = (train[col] - train[col].mean())/train[co
l].std()
```

	price	bedrooms	bathrooms	sqft_living	sc
12536	442500.0	-0.402234	-0.803642	0.682700	0.7
13098	381000.0	-0.402234	-0.479047	0.671829	0.0
17379	440000.0	-0.402234	-0.154452	-0.317497	0.4
14127	307300.0	-1.465211	-0.154452	-0.611033	-0.3
14618	267000.0	0.660743	-0.479047	-0.089191	-0.1

```
16197 5400
```

```
predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model = ols(formula=formula, data=train).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.646
Model:	OLS	**Adj. R-squared:**	0.646
Method:	Least Squares	**F-statistic:**	3284.
Date:	Fri, 19 Mar 2021	**Prob (F-statistic):**	0.00
Time:	10:05:27	**Log-Likelihood:**	-2.2218e+05
No. Observations:	16197	**AIC:**	4.444e+05
Df Residuals:	16187	**BIC:**	4.445e+05
Df Model:	9		
Covariance Type:	nonrobust		

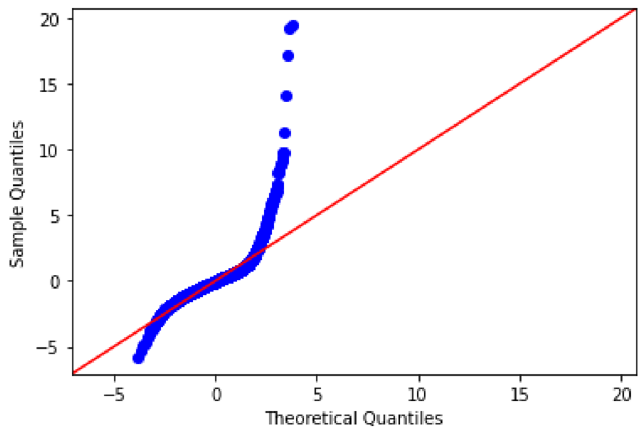
	coef	std err	t	P>|t|	[0.025
Intercept	5.403e+05	1724.545	313.278	0.000	5.37e+05
bedrooms	-3.696e+04	2185.606	-16.911	0.000	-4.12e+04
bathrooms	3.902e+04	3084.654	12.650	0.000	3.3e+04
sqft_living	1.609e+05	3510.825	45.828	0.000	1.54e+05
sqft_lot	-9372.8854	1766.156	-5.307	0.000	-1.28e+04
floors	8600.5384	2161.040	3.980	0.000	4364.661
waterfront	6.307e+04	1742.891	36.184	0.000	5.96e+04
condition	1.199e+04	1876.567	6.387	0.000	8307.926
grade	1.542e+05	2918.802	52.833	0.000	1.48e+05
yr_built	-1.117e+05	2266.720	-49.293	0.000	-1.16e+05

Omnibus:	12304.793	**Durbin-Watson:**	2.016
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	895270.137
Skew:	3.055	**Prob(JB):**	0.00
Kurtosis:	38.906	**Cond. No.**	4.74

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

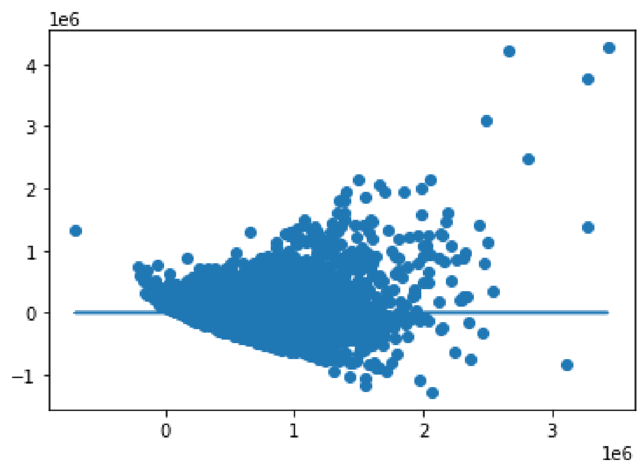
The p-values are less than 0.05 for our selected columns. Let's take a look at our residuals for normality.

```
fig = sm.graphics.qqplot(model.resid, dist=stats.norm, l
```




```
plt.scatter(model.predict(train[x_cols]), model.resid)
plt.plot(model.predict(train[x_cols]), [0 for i in range
```

```
[<matplotlib.lines.Line2D at 0x1f955244820>]
```



This doesn't look great, as our QQ plot looks incorrect and we have a pronounced funnel shape on our check for homoscedasticity. We are going to need to make some changes.

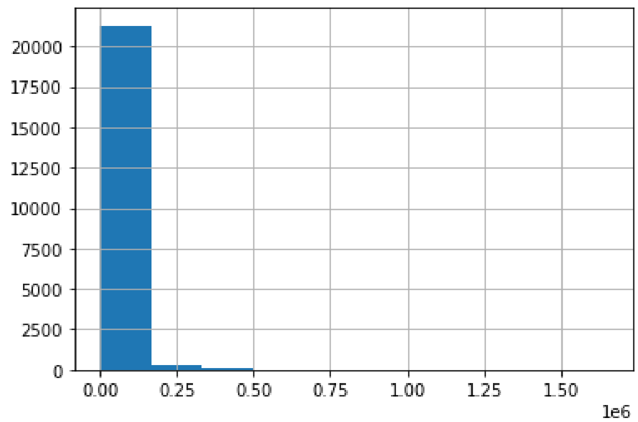
Model 2 Iterations

For this iteration, I'm going to remove some outliers. (log transformation?)

I recall having the most issues determining the normal distributions of sqft_lot and bedrooms, so I'm going to filter on both.

```
df_col_drops.sqft_lot.hist()
```

```
<AxesSubplot:>
```



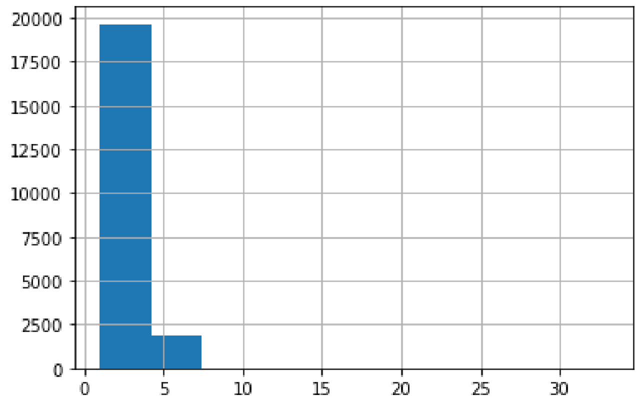

```
for i in range(80,100):
    q = i/100
    print("{} percentile: {}".format(q,df_col_drops.sqft_
```

```
0.8 percentile: 12182.399999999998
0.81 percentile: 12558.0
0.82 percentile: 13055.439999999995
0.83 percentile: 13503.68
0.84 percentile: 14197.0
0.85 percentile: 15000.0
0.86 percentile: 15716.040000000012
0.87 percentile: 16646.640000000003
0.88 percentile: 18000.0
0.89 percentile: 19550.0
0.9 percentile: 21371.600000000006
0.91 percentile: 24149.360000000015
0.92 percentile: 28505.119999999995
0.93 percentile: 34848.0
0.94 percentile: 37643.19999999999
0.95 percentile: 43307.200000000026
0.96 percentile: 50655.28
0.97 percentile: 67381.7199999999
0.98 percentile: 107157.0
0.99 percentile: 213008.0
```

I think filtering out homes with greater than 100k sqaure feet is acceptable here.

```
df_col_drops.bedrooms.hist()
```

```
<AxesSubplot:>
```




```
for i in range(80,100):
    q = i/100
    print("{} percentile: {}".format(q,df_col_drops.bedr
```

```
0.8 percentile: 4.0
0.81 percentile: 4.0
0.82 percentile: 4.0
0.83 percentile: 4.0
0.84 percentile: 4.0
0.85 percentile: 4.0
0.86 percentile: 4.0
0.87 percentile: 4.0
0.88 percentile: 4.0
0.89 percentile: 4.0
0.9 percentile: 4.0
0.91 percentile: 4.0
0.92 percentile: 5.0
0.93 percentile: 5.0
0.94 percentile: 5.0
0.95 percentile: 5.0
0.96 percentile: 5.0
0.97 percentile: 5.0
0.98 percentile: 5.0
0.99 percentile: 6.0
```

```
df_col_drops.bedrooms.value_counts()
```

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

I will also be filtering out all houses with more than 6 bedrooms, removing about 2% of the total entries. (may overlap with sq footage)

I will also include a log transformation to the price feature, as this may help fix our QQplot from Model 1.

```
orig_tot = len(df_col_drops)
df_outlier_filter = df_col_drops.copy()
df_outlier_filters = df_outlier_filter[df_outlier_filter
print('Percent removed:', (orig_tot -len(df_outlier_filt

df_outlier_filters = df_outlier_filters[df_outlier_filte
print('Percent removed:', (orig_tot -len(df_outlier_filt

#applying a log transformation to the price, which is ri
df_outlier_filter['price'] = np.log(df_outlier_filter['p

train2, test2 = train_test_split(df_outlier_filters)

# Refit model with subset features
predictors = '+'.join(x_cols)
formula = outcome + "~" + predictors
model2 = ols(formula=formula, data=train2).fit()
model2.summary()
```

```
Percent removed: 0.021530768162244755
Percent removed: 0.024355234523313424
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.652
Model:	OLS	**Adj. R-squared:**	0.652
Method:	Least Squares	**F-statistic:**	3284.
Date:	Fri, 19 Mar 2021	**Prob (F-statistic):**	0.00
Time:	10:05:28	**Log-Likelihood:**	-2.1609e+05
No. Observations:	15803	**AIC:**	4.322e+05
Df Residuals:	15793	**BIC:**	4.323e+05
Df Model:	9		
Covariance Type:	nonrobust		

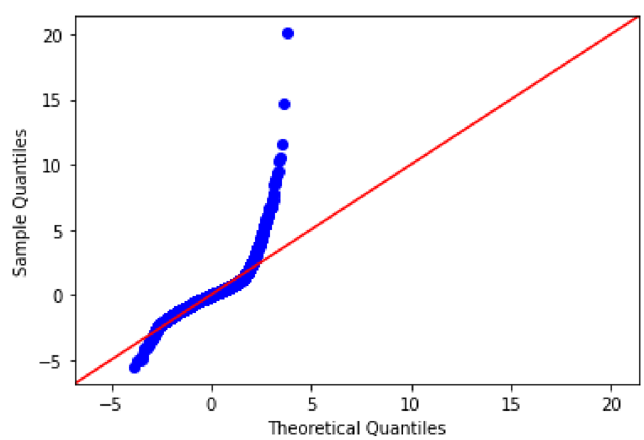
	coef	std err	t	P>|t|	[0.025
Intercept	6.539e+06	1.46e+05	44.654	0.000	6.25e+06
bedrooms	-4.765e+04	2485.774	-19.169	0.000	-5.25e+04
bathrooms	5.407e+04	3930.891	13.756	0.000	4.64e+04
sqft_living	179.6059	3.906	45.984	0.000	171.950
sqft_lot	-1.5529	0.160	-9.714	0.000	-1.866

floors	1.155e+04	3939.660	2.932	0.003	3828.551
waterfront	7.974e+05	2.18e+04	36.615	0.000	7.55e+05
condition	2.005e+04	2784.624	7.201	0.000	1.46e+04
grade	1.304e+05	2446.460	53.311	0.000	1.26e+05
yr_built	-3753.7342	75.323	-49.835	0.000	-3901.375

Omnibus:	10197.238	**Durbin-Watson:**	1.997
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	438980.484
Skew:	2.514	**Prob(JB):**	0.00
Kurtosis:	28.326	**Cond. No.**	1.34e+06

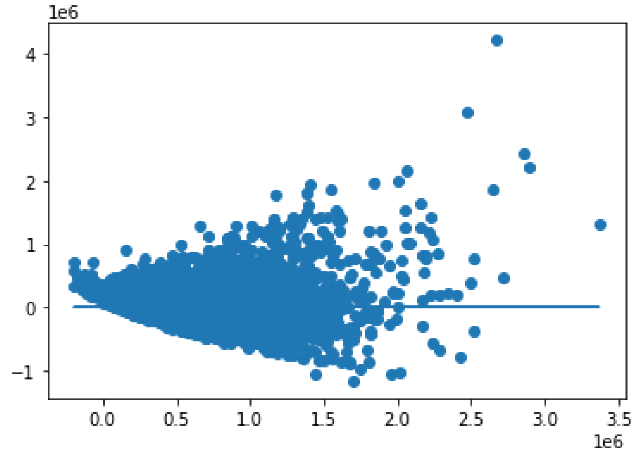
```
Notes:
[1] Standard Errors assume that the covariance matrix of
the errors is correctly specified.
[2] The condition number is large, 1.34e+06. This might
indicate that there are
strong multicollinearity or other numerical problems.
```

```
fig = sm.graphics.qqplot(model2.resid, dist=stats.norm,
```



```
plt.scatter(model2.predict(train2[x_cols]), model2.resid
plt.plot(model2.predict(train2[x_cols]), [0 for i in ran
```

```
[<matplotlib.lines.Line2D at 0x1f95673ef40>]
```



Similar problems as last time, but our OLS has alerted us that there is strong collinearity. Let's investigate what we should remove.

```
X = df_col_drops[x_cols]
vif = [variance_inflation_factor(X.values, i) for i in r
list(zip(x_cols, vif))
```

```
[('bedrooms', 23.09608478897893),
 ('bathrooms', 24.591759875968087),
 ('sqft_living', 25.181513925621946),
 ('sqft_lot', 1.18527557276325),
 ('floors', 13.133195105016583),
 ('waterfront', 1.0252421775002192),
 ('condition', 29.533165474917077),
 ('grade', 124.69739326481557),
 ('yr_built', 124.82668596464562)]
```

Going to drop 'grade' and 'yr_built' from our model for the time being and go from there. You usually want to remove variables with a cif of 10 or greater, incdicating that they are displaying multicollinearity with other variables in the feature set.

```
train2a, test2a = train_test_split(df_outlier_filter)

outcome = 'price'
x_cols = ['bedrooms','bathrooms','sqft_living','sqft_lot
predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model2a = ols(formula=formula, data=train2a).fit()
model2a.summary()
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.512
Model:	OLS	**Adj. R-squared:**	0.512
Method:	Least Squares	**F-statistic:**	2427.
Date:	Fri, 19 Mar 2021	**Prob (F-statistic):**	0.00
Time:	10:05:28	**Log-Likelihood:**	-6739.1
No. Observations:	16197	**AIC:**	1.349e+04
Df Residuals:	16189	**BIC:**	1.356e+04
Df Model:	7		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	11.9256	0.022	552.526	0.000	11.883	11.968
bedrooms	-0.0612	0.004	-14.980	0.000	-0.069	-0.053
bathrooms	0.0371	0.006	5.888	0.000	0.025	0.050
sqft_living	0.0004	5.25e-06	74.830	0.000	0.000	0.000
sqft_lot	-1.69e-07	6.85e-08	-2.468	0.014	-3.03e-07	-3.48e-08
floors	0.0893	0.006	13.983	0.000	0.077	0.102
waterfront	0.5934	0.035	17.050	0.000	0.525	0.662
condition	0.0875	0.005	18.884	0.000	0.078	0.097

Omnibus:	5.730	**Durbin-Watson:**	1.980
Prob(Omnibus):	0.057	**Jarque-Bera (JB):**	6.076
Skew:	0.012	**Prob(JB):**	0.0479
Kurtosis:	3.092	**Cond. No.**	5.51e+05

```
Notes:
```

```
[1] Standard Errors assume that the covariance matrix of
the errors is correctly specified.
[2] The condition number is large, 5.51e+05. This might
indicate that there are
strong multicollinearity or other numerical problems.
```

```
X = df_col_drops[x_cols]
vif = [variance_inflation_factor(X.values, i) for i in r
list(zip(x_cols, vif))
```

```
[('bedrooms', 20.155807501016387),
 ('bathrooms', 24.05414728245284),
 ('sqft_living', 16.680775340246925),
 ('sqft_lot', 1.17958229220644),
 ('floors', 10.093490615100084),
 ('waterfront', 1.0251110953470544),
 ('condition', 10.865278180945477)]
```

```
train2b, test2b = train_test_split(df_outlier_filter)

x_cols = ['sqft_living','sqft_lot', 'floors','waterfront
predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model2b = ols(formula=formula, data=train2b).fit()
model2b.summary()
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.505
Model:	OLS	**Adj. R-squared:**	0.505
Method:	Least Squares	**F-statistic:**	3307.
Date:	Fri, 19 Mar 2021	**Prob (F-statistic):**	0.00
Time:	10:05:28	**Log-Likelihood:**	-6837.0
No. Observations:	16197	**AIC:**	1.369e+04
Df Residuals:	16191	**BIC:**	1.373e+04
Df Model:	5		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	11.8351	0.020	588.019	0.000	11.796	11.875
sqft_living	0.0004	3.45e-06	108.001	0.000	0.000	0.000
sqft_lot	-1.919e-07	7.12e-08	-2.696	0.007	-3.31e-07	-5.24e-08
floors	0.1051	0.006	17.704	0.000	0.093	0.117
waterfront	0.6124	0.035	17.475	0.000	0.544	0.681
condition	0.0816	0.005	17.591	0.000	0.073	0.091

Omnibus:	1.639	**Durbin-Watson:**	2.028
Prob(Omnibus):	0.441	**Jarque-Bera (JB):**	1.639
Skew:	0.009	**Prob(JB):**	0.441
Kurtosis:	2.954	**Cond. No.**	5.34e+05

```
Notes:
[1] Standard Errors assume that the covariance matrix of
the errors is correctly specified.
[2] The condition number is large, 5.34e+05. This might
```

```
indicate that there are

strong multicollinearity or other numerical problems.
```

```
X = df_col_drops[x_cols]
vif = [variance_inflation_factor(X.values, i) for i in r
list(zip(x_cols, vif))
```

```
[('sqft_living', 7.1722391606368525),
 ('sqft_lot', 1.1732206092964013),
 ('floors', 7.77186134275548),
 ('waterfront', 1.017459668782625),
 ('condition', 6.7043028937497136)]
```

```
train2c, test2c = train_test_split(df_outlier_filter)

x_cols = ['sqft_living','sqft_lot','waterfront','conditi
predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model2c = ols(formula=formula, data=train2c).fit()
model2c.summary()
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.502
Model:	OLS	**Adj. R-squared:**	0.502
Method:	Least Squares	**F-statistic:**	4081.
Date:	Fri, 19 Mar 2021	**Prob (F-statistic):**	0.00
Time:	10:05:29	**Log-Likelihood:**	-7009.4
No. Observations:	16197	**AIC:**	1.403e+04
Df Residuals:	16192	**BIC:**	1.407e+04
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	12.0085	0.017	689.970	0.000	11.974	12.043
sqft_living	0.0004	3.26e-06	122.372	0.000	0.000	0.000
sqft_lot	-2.595e-07	7.31e-08	-3.548	0.000	-4.03e-07	-1.16e-07
waterfront	0.6200	0.034	18.106	0.000	0.553	0.687
condition	0.0610	0.005	13.489	0.000	0.052	0.070

Omnibus:	46.777	**Durbin-Watson:**	1.997
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	36.592
Skew:	0.002	**Prob(JB):**	1.13e-08
Kurtosis:	2.767	**Cond. No.**	5.07e+05

```
Notes:
[1] Standard Errors assume that the covariance matrix of
the errors is correctly specified.
[2] The condition number is large, 5.07e+05. This might
```

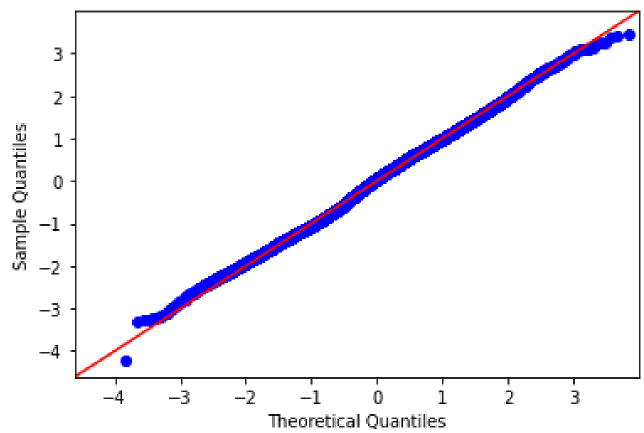
```
indicate that there are

strong multicollinearity or other numerical problems.
```

```
X = df_col_drops[x_cols]
vif = [variance_inflation_factor(X.values, i) for i in r
list(zip(x_cols, vif))
```

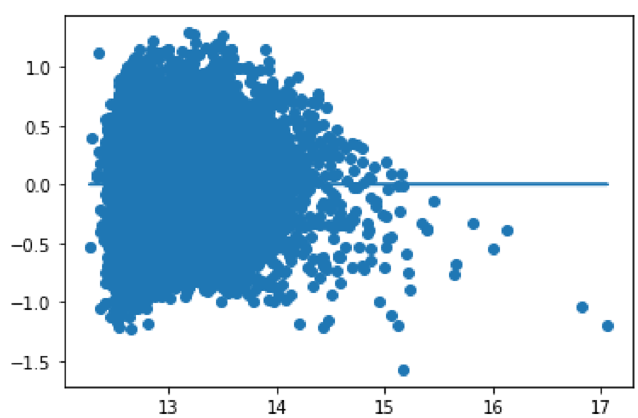
```
[('sqft_living', 5.2175232204298165),
 ('sqft_lot', 1.1680613101385844),
 ('waterfront', 1.0162448940537252),
 ('condition', 4.999908921746516)]
```

```
fig = sm.graphics.qqplot(model2c.resid, dist=stats.norm,
```




```
plt.scatter(model2c.predict(train2c[x_cols]), model2c.re
plt.plot(model2c.predict(train2c[x_cols]), [0 for i in r
```

```
[<matplotlib.lines.Line2D at 0x1f95685af70>]
```



This is a modeling choice. There are pros and cons to this approach versus the first model. Removing multiple components has substantially diminished the model's performance, as indicated by the r-squared value. However, multicollinearity between the features has been reduced.

Model 3

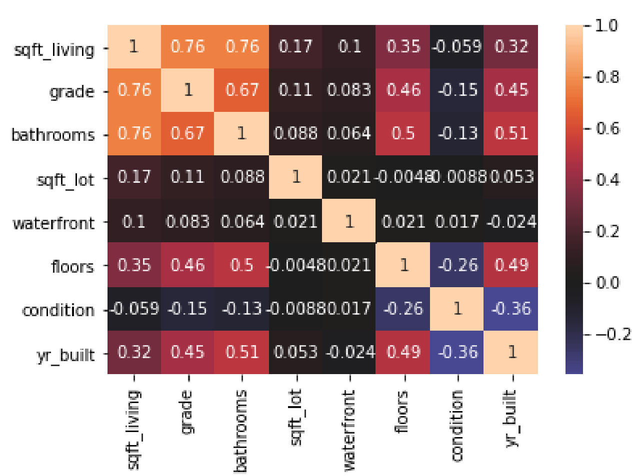
Going back to the drawing board, let's look at a multicolinearity heatmap to determine the columns to remove from our model.

```
first_features = ['sqft_living', 'grade', 'bathrooms', '
corr = df_col_drops[first_features].corr()
corr
```

	sqft_living	grade	bathrooms	sqft_lot
sqft_living	1.000000	0.762779	0.755758	0.173453
grade	0.762779	1.000000	0.665838	0.114731
bathrooms	0.755758	0.665838	1.000000	0.088373
sqft_lot	0.173453	0.114731	0.088373	1.000000
waterfront	0.104637	0.082818	0.063629	0.021459
floors	0.353953	0.458794	0.502582	-0.004814
condition	-0.059445	-0.146896	-0.126479	-0.008830
yr_built	0.318152	0.447865	0.507173	0.052946

```
sns.heatmap(corr, center=0, annot=True)
```

<AxesSubplot:>



sqft_living and grade = 0.76

sqft_living and bathrooms = 0.76

grade and bathrooms = 0.67

Let's remove grade and bathrooms for this model. We will also use our previous outlier filter, as this seems to be a step in the right direction.

```
train3, test3 = train_test_split(df_outlier_filter)

x_cols = ['sqft_living', 'sqft_lot', 'waterfront', 'floo
predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model3 = ols(formula=formula, data=train3).fit()
model3.summary()
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.537
Model:	OLS	**Adj. R-squared:**	0.537
Method:	Least Squares	**F-statistic:**	3126.
Date:	Fri, 19 Mar 2021	**Prob (F-statistic):**	0.00
Time:	10:05:30	**Log-Likelihood:**	-6355.8
No. Observations:	16197	**AIC:**	1.273e+04
Df Residuals:	16190	**BIC:**	1.278e+04
Df Model:	6		
Covariance Type:	nonrobust		

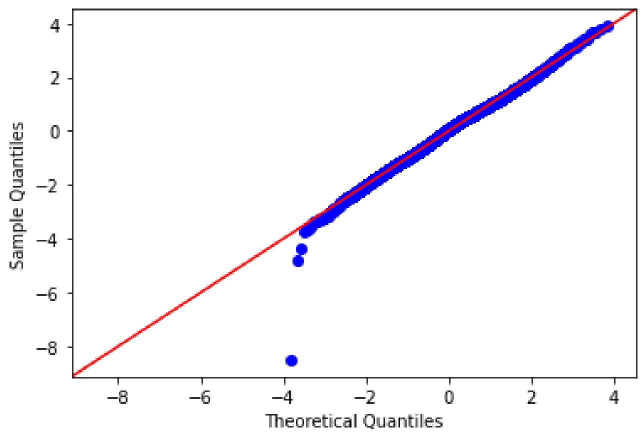
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	19.1667	0.231	83.101	0.000	18.715	19.61ɕ
sqft_living	0.0004	3.42e-06	115.938	0.000	0.000	0.000
sqft_lot	-1.651e-07	7.15e-08	-2.310	0.021	-3.05e-07	-2.5e-08
waterfront	0.5972	0.035	16.833	0.000	0.528	0.667
floors	0.1724	0.006	27.550	0.000	0.160	0.185
condition	0.0380	0.005	8.137	0.000	0.029	0.047
yr_built	-0.0037	0.000	-31.884	0.000	-0.004	-0.003

Omnibus:	115.168	**Durbin-Watson:**	2.011
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	141.348
Skew:	-0.133	**Prob(JB):**	2.03e-31
Kurtosis:	3.372	**Cond. No.**	3.52e+06

```
Notes:
[1] Standard Errors assume that the covariance matrix of
```

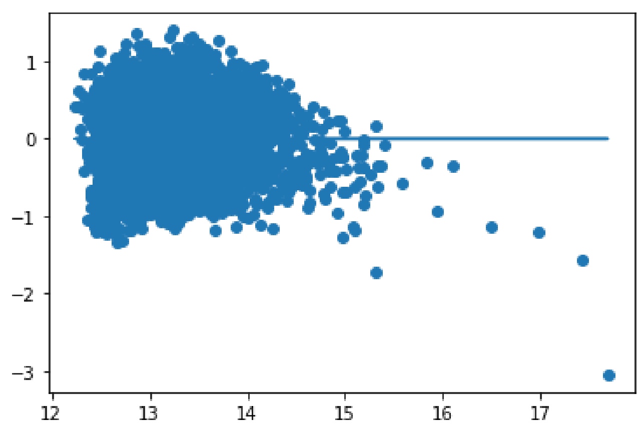
```
the errors is correctly specified.
[2] The condition number is large, 3.52e+06. This might
indicate that there are
strong multicollinearity or other numerical problems.
```

```
fig = sm.graphics.qqplot(model3.resid, dist=stats.norm,
```




```
plt.scatter(model3.predict(train3[x_cols]), model3.resid
plt.plot(model3.predict(train3[x_cols]), [0 for i in ran
```

```
[<matplotlib.lines.Line2D at 0x1f94f8e4790>]
```



Model 4

Our QQ plots are less than ideal in previous models. Let's see if we can fix that by using a transform on the appropriate features.

```
for col_name in df_outlier_filter.columns[1:]:
    print(col_name)
    print(df_outlier_filter[col_name].skew())
```

```
bedrooms
2.023641235344595
bathrooms
0.5197092816403838
sqft_living
1.473215455425834
sqft_lot
13.072603567136046
floors
0.6144969756263127
waterfront
12.039584643829357
condition
1.0360374245132955
grade
0.7882366363846076
yr_built
-0.4694499764949978
```

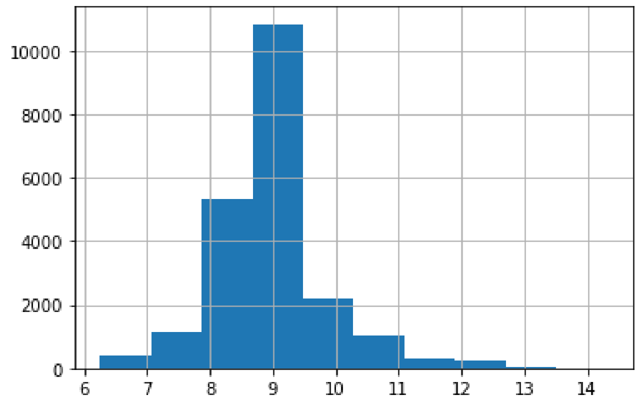
'sqft_lot' seems to be the main issue with the highest skew coefficient. I'm not sure if I should apply this to waterfront. We may need to use another method here, or look elsewhere for model improvements.

```
#only run once
df_outlier_filter['sqft_lot'] = np.log(df_outlier_filter
df_outlier_filter['sqft_lot'].skew()
```

```
0.9625003856495555
```

```
df_outlier_filter['sqft_lot'].hist()
```

```
<AxesSubplot:>
```




```
df_outlier_filter['bedrooms'] = np.log(df_outlier_filter
df_outlier_filter['bedrooms'].skew()
```

```
-0.6805637280656164
```

```
x_cols = list(df_outlier_filter.columns)
x_cols.remove(outcome)

train4, test4 = train_test_split(df_outlier_filter)

predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model4 = ols(formula=formula, data=train4).fit()
model4.summary()
```

OLS Regression Results

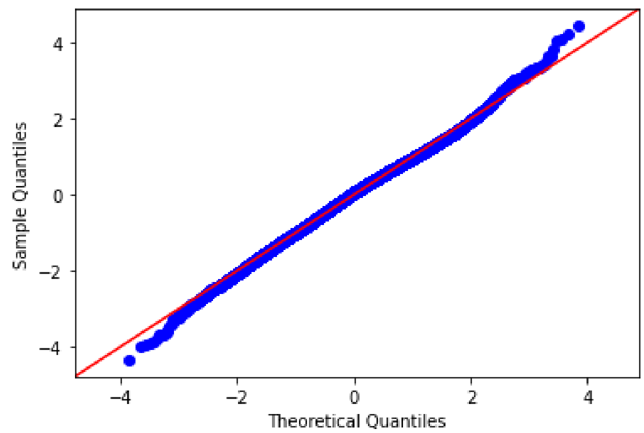
Dep. Variable:	price	**R-squared:**	0.646
Model:	OLS	**Adj. R-squared:**	0.645
Method:	Least Squares	**F-statistic:**	3275.
Date:	Fri, 19 Mar 2021	**Prob (F-statistic):**	0.00
Time:	10:05:31	**Log-Likelihood:**	-4151.9
No. Observations:	16197	**AIC:**	8324.
Df Residuals:	16187	**BIC:**	8401.
Df Model:	9		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	22.0160	0.214	102.848	0.000	21.596	22.436
bedrooms	-0.1065	0.011	-9.635	0.000	-0.128	-0.085
bathrooms	0.0808	0.006	14.091	0.000	0.070	0.092
sqft_living	0.0002	5.7e-06	36.097	0.000	0.000	0.000
sqft_lot	-0.0470	0.003	-14.600	0.000	-0.053	-0.041
floors	0.0520	0.006	8.578	0.000	0.040	0.064
waterfront	0.4809	0.031	15.731	0.000	0.421	0.541
condition	0.0458	0.004	11.110	0.000	0.038	0.054
grade	0.2273	0.004	63.858	0.000	0.220	0.234
yr_built	-0.0056	0.000	-50.456	0.000	-0.006	-0.005

Omnibus:	66.219	**Durbin-Watson:**	2.038
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	77.043
Skew:	-0.100	**Prob(JB):**	1.86e-17
Kurtosis:	3.273	**Cond. No.**	2.57e+05

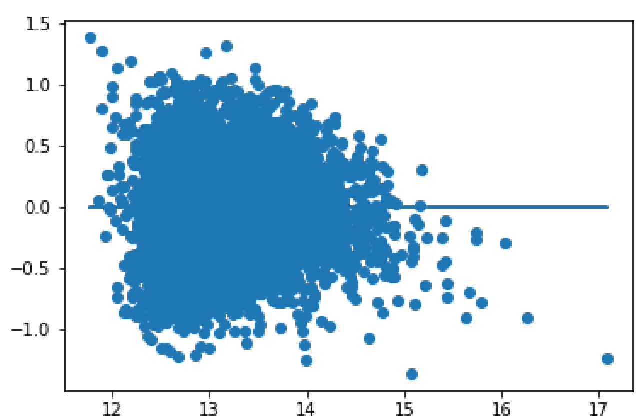
```
Notes:
[1] Standard Errors assume that the covariance matrix of
the errors is correctly specified.
[2] The condition number is large, 2.57e+05. This might
indicate that there are
strong multicollinearity or other numerical problems.
```

```
fig = sm.graphics.qqplot(model4.resid, dist=stats.norm,
```




```
plt.scatter(model4.predict(train4[x_cols]), model4.resid
plt.plot(model4.predict(train4[x_cols]), [0 for i in ran
```

```
[<matplotlib.lines.Line2D at 0x1f95378cdc0>]
```



This is a nice improvement. This is our best model thus far. It passes the normality check from looking at the QQ plot and it is homoscedastic.

Interpreting this model:

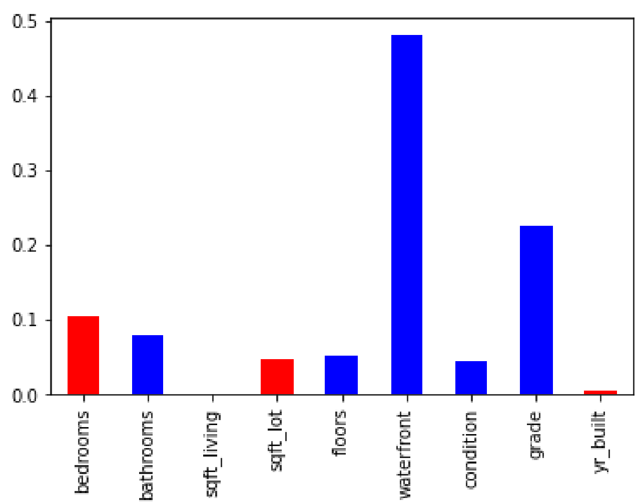
R-squared: 64.6% variation in the price can be explained by 'sqft_living', 'grade', 'bathrooms', 'sqft_lot', 'waterfront', 'floors', 'condition', 'yr_built', and 'bedrooms'

Durbin-waton: A value preferred between 1-2 implies that the regression results are reliable from the side of homoscedasticity.

The highest coefficients belong to Grade and Waterfront: namely, what grade the home has been given by the King County Housing System. Additionally, having a waterfront view as a part of your home largely impacts the price.

When needed, we can now use this model to give us prediction values for an estimated price, given the values for the features of a home we are trying to sell. Obviously, someone would be unable to renovate their home to suddenly have a waterfront view, but doing something like adding a bathroom (the 3rd highest coefficient) seems to also have a significant impact of the expected price of a home for this model as well.

```
model4.params[1:].abs().plot.bar(color=['red','blue','bl
```



This is a visualization of our coefficients. To compare, I have taken the absolute value of each in the series, but made sure to indicate negative coefficients in red columns.

Conclusion

I believe the best model is Model 4, where the outliers have been filtered out and none of the features are removed . Although this suffers from multicollinearity, it has an r-squared value of ~0.647, which is the most accurate model in our analysis.

I believe this is acceptable within the context of this scenario. It affects the coefficients and p-values, but it does not influence the predicitons, precision of the predictions, and the statistics determining goodness of fit. Our primary goal is to have a model to make predictions for us.

To further improve this, I would use more of the columns included in the original dataset to try to increase my r-squared value and hopefully fix the QQplot issues I was having for all of my models.