1 Building A Pricing Model For First Time Home Buyers

2 Overview

In this analysis, I inspect the King County House Sales dataset and iteratively develop a multiple regression model to analyze house prices.

3 Business Problem

There are lots of residents in King County, Washington who are considering buying their first home. These prospective buyers could benefit immensely from being able to accurately forecast the price of their first home based on a set of given parameters.

As an analyst for DLG Real Estate Agency, I am tasked with developing a regression model to help my fellow employees determine which homes are best for their clients.

4 Importing Data, Necessary Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.formula as smf
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.graphics.gofplots import qqplot
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
pd.set_option("display.max_columns", 100)
```

In [4]:
```
df = pd.read_csv('data/kc_house_data.csv')
```

In [5]:
```
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
```

4.1 Column Names and descriptions for Kings County Data Set

- **id** - Unique identifier for a house
- **date** - Date house was sold
- **price** - Price is prediction target
- **bedrooms** - Number of Bedrooms/House
- **bathrooms** - Number of bathrooms/bedrooms
- **sqft_living** - Square footage of the home
- **sqft_lot** - Square footage of the lot
- **floors** - Total floors (levels) in house
- **waterfront** - House which has a view to a waterfront
- **view** - score of view from house
- **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

Due to time constraints on this project, I am focusing solely on the following predictors:

- bedrooms
- bathrooms
- sqft_living
- sqft_lot
- floors
- waterfront
- condition
- grade
- yr_built
- zipcode
- view

Consequently, all other columns are dropped.

In [6]:
```
to_drop = ['date', 'sqft_above', 'sqft_basement', 'yr_renovated', 'lat', 'l
df.drop(to_drop, axis=1, inplace=True)
```

5 Basic Data Cleaning & Initial Model

Looking at the DataFrame information provided above, it appears that some columns have varying amounts of null values. Let's drop those and see if there are enough remaining entries for our analysis (at least 15,000).

In [7]:
```
df.dropna(inplace=True)
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 19164 entries, 1 to 21596
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   id           19164 non-null  int64  
 1   price        19164 non-null  float64
 2   bedrooms     19164 non-null  int64  
 3   bathrooms    19164 non-null  float64
 4   sqft_living  19164 non-null  int64  
 5   sqft_lot     19164 non-null  int64  
 6   floors       19164 non-null  float64
 7   waterfront   19164 non-null  float64
 8   view         19164 non-null  float64
 9   condition    19164 non-null  int64  
 10  grade        19164 non-null  int64  
 11  yr_built     19164 non-null  int64  
 12  zipcode      19164 non-null  int64  
dtypes: float64(5), int64(8)
memory usage: 2.0 MB
```

In [9]:
```
df.head()
```

Out[9]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
1	6414100192	538000.0	3	2.25	2570	7242	2.0	0.
2	5631500400	180000.0	2	1.00	770	10000	1.0	0.
3	2487200875	604000.0	4	3.00	1960	5000	1.0	0.
4	1954400510	510000.0	3	2.00	1680	8080	1.0	0.
5	7237550310	1230000.0	4	4.50	5420	101930	1.0	0.

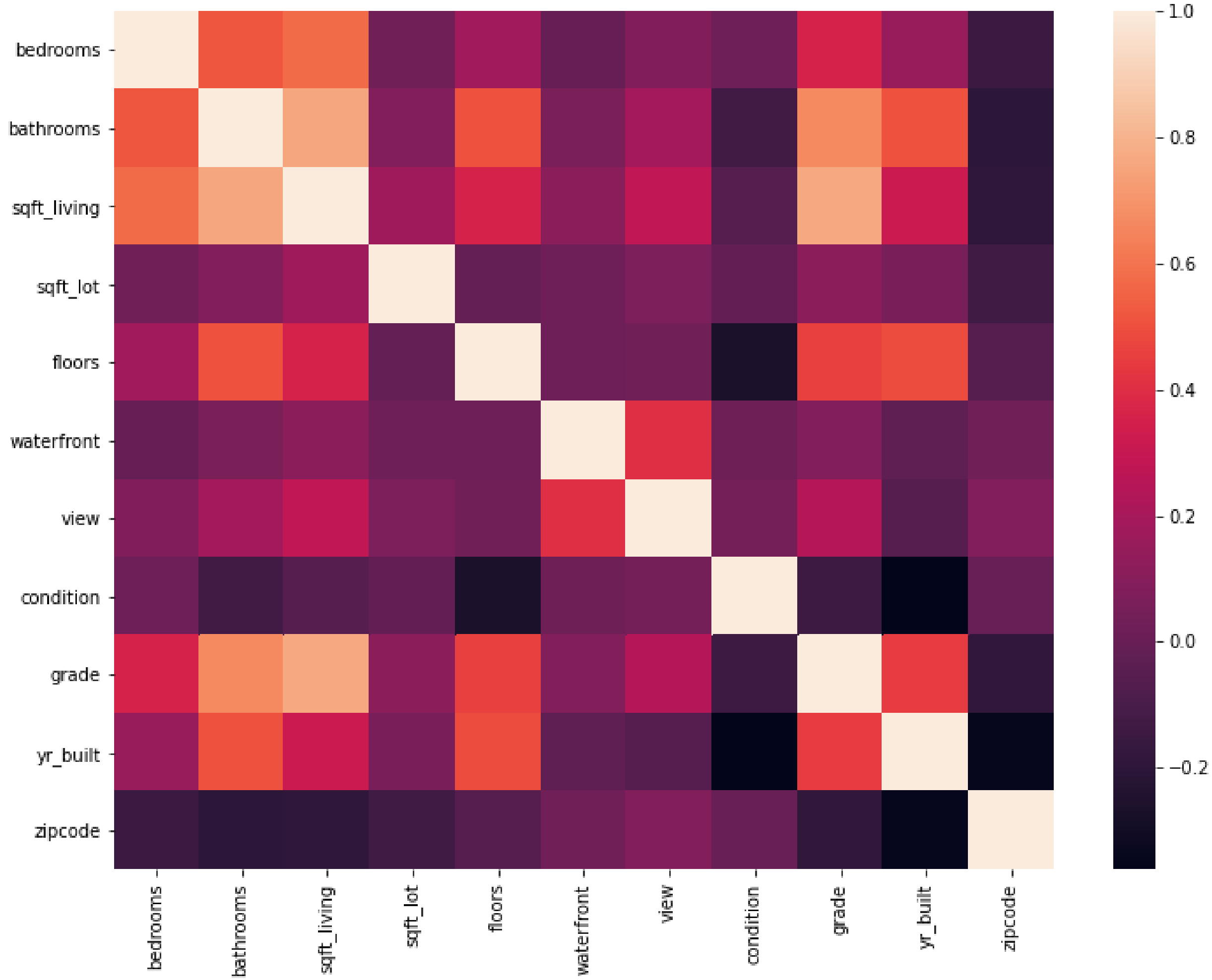
I now create a DataFrame **df_pred** containing only our predictors, dropping the *price* & *id* columns.

In [10]:
```
df_pred = df.drop(['price', 'id'], axis=1)
```

Next, I take a look at the correlation between these features:

In [11]:
```
plt.figure(figsize=(12,9))
sns.heatmap(df_pred.corr())
```

Out[11]: <AxesSubplot:>



In [12]:
```
corr_pair = df_pred.corr().abs().stack().reset_index().sort_values(0, ascen
corr_pair['pairs'] = list(zip(corr_pair.level_0, corr_pair.level_1))
corr_pair.set_index(['pairs'], inplace = True)
corr_pair.drop(columns=['level_1', 'level_0'], inplace = True)
corr_pair.columns = ['cc']
corr_pair.drop_duplicates(inplace=True)
```

In [13]:
```
corr_pair[(corr_pair.cc>.75) & (corr_pair.cc <1)]
```

Out[13]:

pairs	cc
(sqft_living, grade)	0.763701
(bathrooms, sqft_living)	0.755909

As seen in both the heatmap & new DataFrame **corr_pair**, the variables *sqft_living*, *grade*, & *bathrooms* are highly correlated (correlation coefficient having an absolute value of over 0.75, indicated on the heatmap by a light shade).

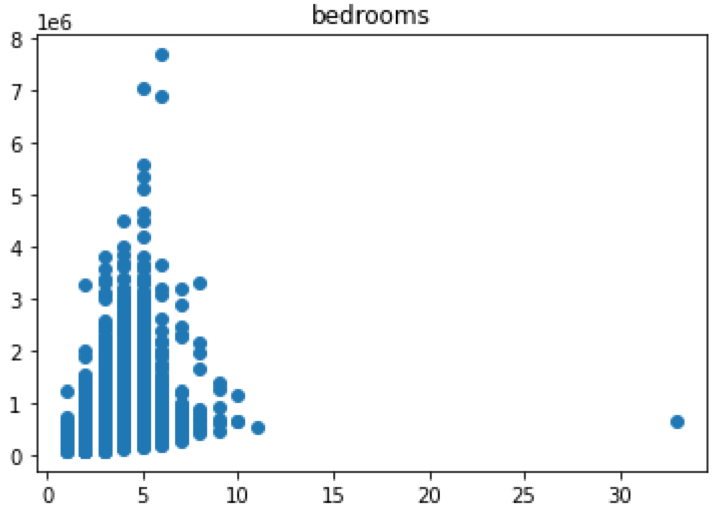
In order to remove collinear features, I drop *grade* & *bathrooms*, leaving only the *sqft_living* predictor.

In [14]:
```
df_pred.drop(columns=['grade', 'bathrooms'], inplace=True)
df.drop(columns=['grade', 'bathrooms'], inplace=True)
```

Next, I inspect how the remaining predictors look when plotted individually against the dependent *price* variable in a scatterplot.

These plots will be referenced again later on for any potential feature manipulation.

In [15]:
```
for col in df_pred.columns:
    plt.scatter(df_pred[col], df['price'])
    plt.title(col)
    plt.show()
```



I now run a baseline regression model using the above set of predictors, unchanged, before evaluating which features to change.

In [16]:
```
outcome = 'price'
x_cols = df_pred.columns
predictors = '+'.join(x_cols)

f = outcome + '~' + predictors
```

In [17]:
```
model_1 = ols(formula=f, data=df).fit()
model_1.summary()
```

Out[17]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.597
Model:	OLS	**Adj. R-squared:**	0.597
Method:	Least Squares	**F-statistic:**	3149.
Date:	Tue, 26 Jan 2021	**Prob (F-statistic):**	0.00
Time:	00:45:56	**Log-Likelihood:**	-2.6424e+05
No. Observations:	19164	**AIC:**	5.285e+05
Df Residuals:	19154	**BIC:**	5.286e+05
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	6.454e+06	3.51e+06	1.836	0.066	-4.35e+05	1.33e+07
bedrooms	-5.209e+04	2279.762	-22.848	0.000	-5.66e+04	-4.76e+04
sqft_living	306.9798	2.606	117.804	0.000	301.872	312.087
sqft_lot	-0.3727	0.043	-8.681	0.000	-0.457	-0.289
floors	7.485e+04	3811.423	19.638	0.000	6.74e+04	8.23e+04
waterfront	5.634e+05	2.15e+04	26.174	0.000	5.21e+05	6.06e+05
view	5.94e+04	2586.549	22.965	0.000	5.43e+04	6.45e+04
condition	1.905e+04	2861.467	6.656	0.000	1.34e+04	2.47e+04
yr_built	-2494.5937	75.864	-32.883	0.000	-2643.293	-2345.895
zipcode	-16.8060	35.267	-0.477	0.634	-85.932	52.320

Omnibus:	11751.006	**Durbin-Watson:**	1.977
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	433493.239
Skew:	2.378	**Prob(JB):**	0.00
Kurtosis:	25.809	**Cond. No.**	2.05e+08

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.05e+08. This might indicate that there are
strong multicollinearity or other numerical problems.

In [18]:
```
data = df.copy()

y = data['price']
X = data.drop(['price', 'id'], axis = 1)
```

In [19]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

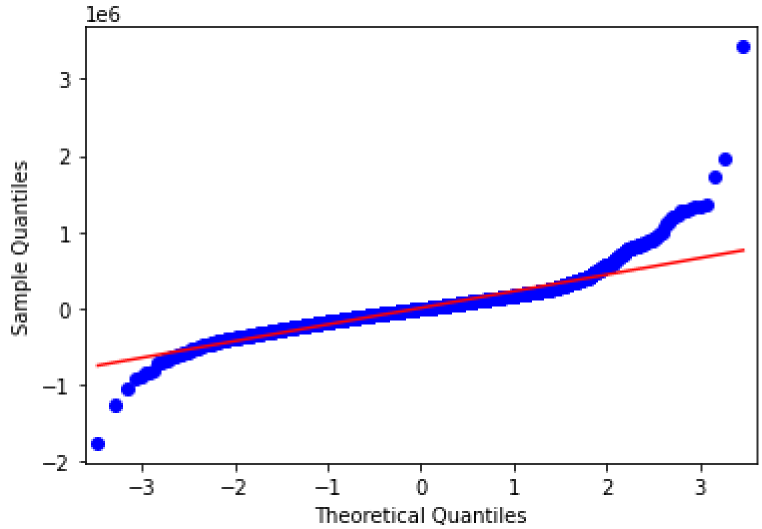
In [20]:
```
linreg = LinearRegression()
linreg.fit(X_train, y_train)

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

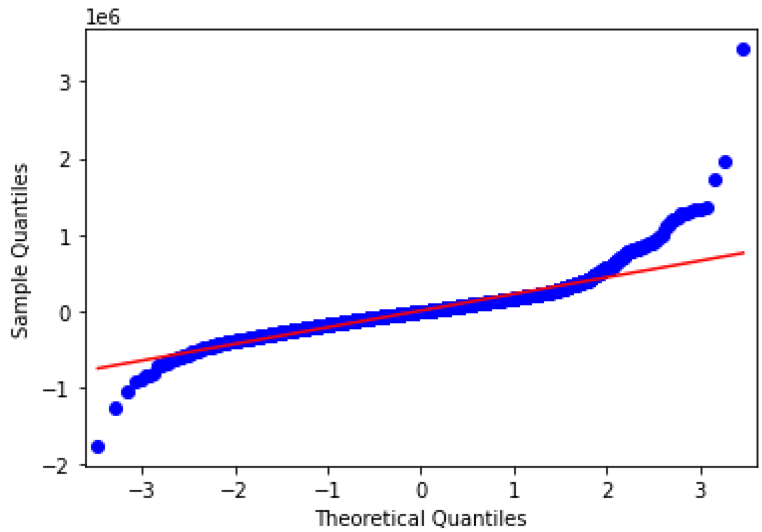
In [21]:
```
mse_train = mean_squared_error(y_train, y_hat_train)
mse_test = mean_squared_error(y_test, y_hat_test)

print('Train MSE:', mse_train)
print('Test MSE:', mse_test)

print('RMSE Train:', np.sqrt(mse_train))
print('RMSE Test:', np.sqrt(mse_test))
```

```
Train MSE: 55834878369.075
Test MSE: 54133700042.9983
RMSE Train: 236294.0506425733
RMSE Test: 232666.49961478834
```

In [22]:
```
residuals = (y_test - y_hat_test)

sm.qqplot(residuals, line = "r")
```

Out[22]:





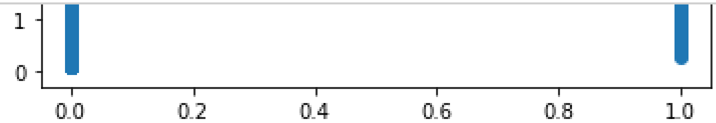
This first model has an R-squared value of 0.597, and a Root Mean Square Error of around 240,000. Additionally, the residuals seem somewhat skewed based on the ends of the Q-Q Plot.

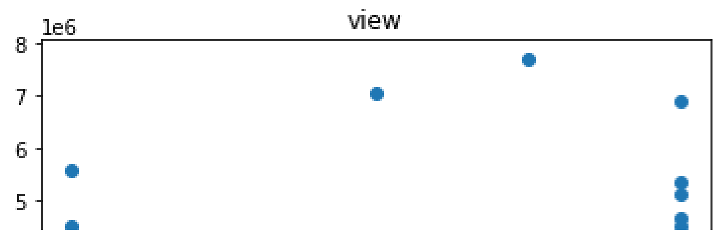
6 Model 2: Dropping Outliers, Using Heuristics & Z-Scores

Now that a baseline model has been constructed to improve on, let's return to the initial business problem: creating a price-prediction model for first time home buyers.

Recall the scatterplots constructed for the individual predictors against price. There are multiple features in the dataset with values that far exceed what would be found in a 'first home'.

In [23]:
```
for col in df_pred.columns:
    plt.scatter(df_pred[col], df['price'])
    plt.title(col)
    plt.show()
```





6.1 Dropping Data Using Heuristics

First, take a look at the *bedrooms* scatterplot. There is an obvious outlier home with 33 bedrooms, which is certainly unfeasible for any first time buyer.

I begin by dropping this entry.

In [24]:
```
df[df['bedrooms'] == 33]
```

Out[24]:

	id	price	bedrooms	sqft_living	sqft_lot	floors	waterfront	view	c
15856	2402100895	640000.0	33	1620	6000	1.0	0.0	0.0	

In [25]:
```
df.drop(labels=15856, axis=0, inplace=True)
```

Now, I inspect the *bedrooms* column.

In [26]:
```
df.sort_values(by=['bedrooms'], axis=0, ascending=False)
```

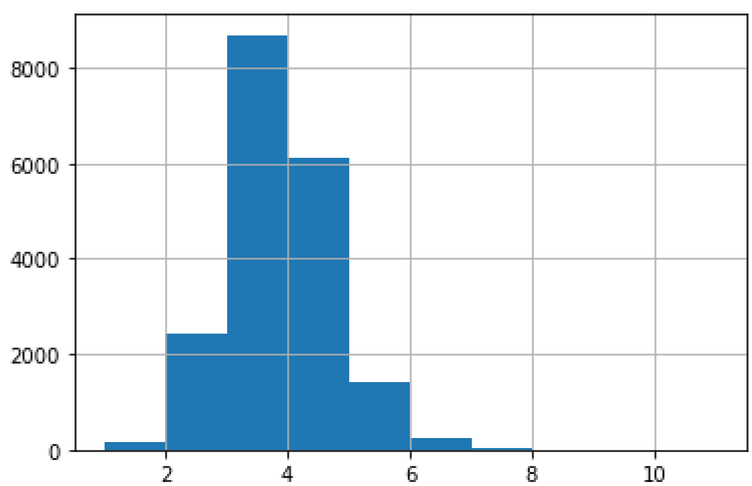
Out[26]:

	id	price	bedrooms	sqft_living	sqft_lot	floors	waterfront	view
8748	1773100755	520000.0	11	3000	4960	2.0	0.0	0.0
13301	627300145	1150000.0	10	4590	10920	1.0	0.0	2.0
19239	8812401450	660000.0	10	2920	3745	2.0	0.0	0.0
15147	5566100170	650000.0	10	3610	11914	2.0	0.0	0.0
4092	1997200215	599999.0	9	3830	6988	2.5	0.0	0.0
...	...	...	...	...	...	...	...	...
648	922049078	157000.0	1	870	26326	1.0	0.0	0.0
7368	7228501903	250000.0	1	780	1033	1.0	0.0	0.0
3380	8807900236	430000.0	1	630	1362	1.0	0.0	0.0
18261	2781600195	285000.0	1	1060	54846	1.0	1.0	4.0
17282	4047200825	400000.0	1	1390	60984	1.0	0.0	0.0

19163 rows × 11 columns

In [27]:
```
df['bedrooms'].hist()
```

Out[27]: <AxesSubplot:>

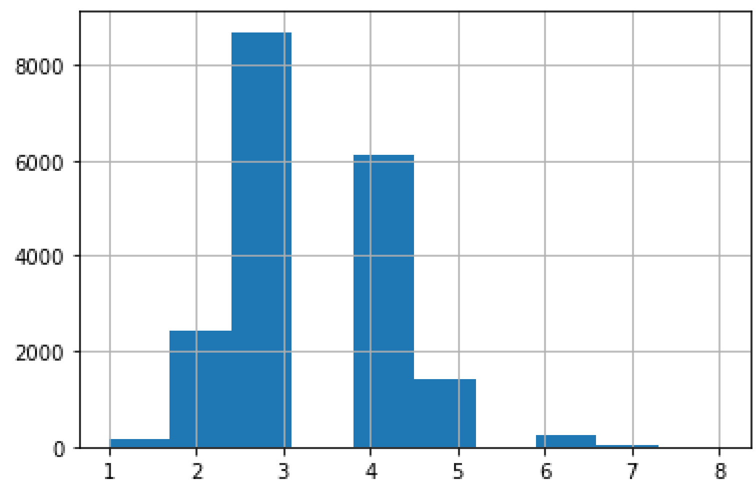


Even if a potential first time buyer is a large family in need of more rooms than usual, it is difficult to envision such a client needing any more than 8 bedrooms.

In [28]:
```
df = df[df['bedrooms'] < 9]
```

In [29]:
```
df['bedrooms'].hist()
```

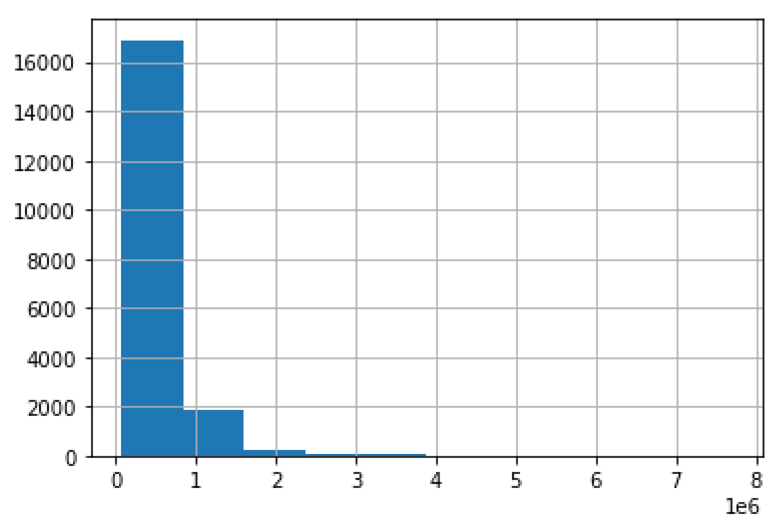
Out[29]: <AxesSubplot:>



6.2 Dropping Data Using Z-Scores of Continuous Variables

In [30]:
```
df['price'].hist()
```

Out[30]: <AxesSubplot:>



In [31]:
```
mean_price = df['price'].mean()
std3_price = 3*df['price'].std()
print(mean_price - std3_price, mean_price + std3_price)
```

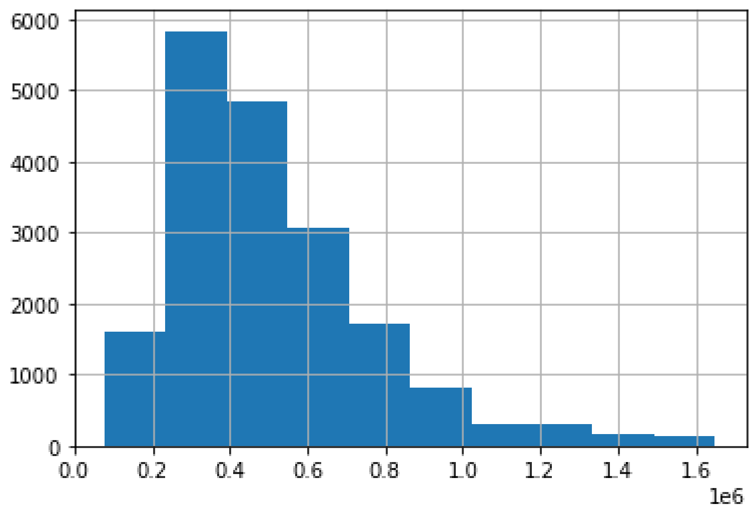
```
-571334.4145640415 1653916.1387847904
```

From the histogram of home prices, we can tell that none of the prices fall more than 3 standard deviations below the mean price (given by the interval above). There are, however, outlying prices more than 3 standard deviations above the mean. I elect to filter these homes out of the dataset.

In [32]:
```
upper_price = mean_price + std3_price
df = df[df['price'] <= upper_price]
```

In [33]:
```
df['price'].hist()
```

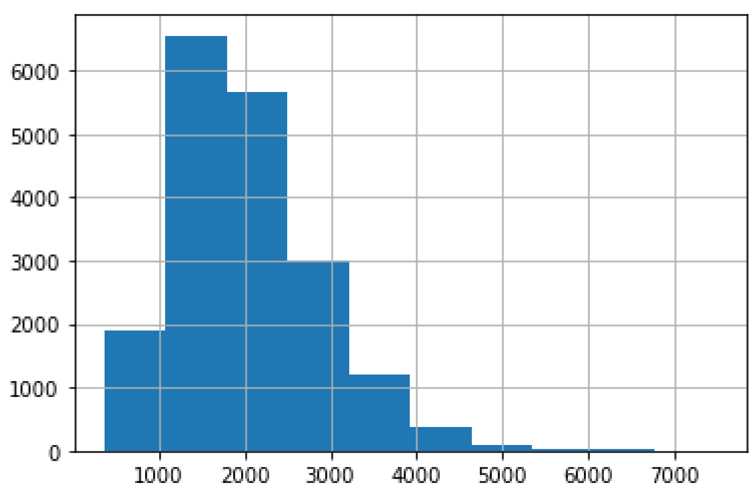
Out[33]: <AxesSubplot:>



Next, I look at the *sqft_living* predictor.

In [34]:
```
df['sqft_living'].hist()
```

Out[34]: <AxesSubplot:>



In [35]:
```
df['sqft_living'].describe()
```

Out[35]:
```
count    18803.000000
mean      2033.471999
std        837.690802
min        370.000000
25%       1411.500000
50%       1900.000000
75%       2510.000000
max       7480.000000
Name: sqft_living, dtype: float64
```

In [36]:
```
mean_living = df['sqft_living'].mean()
std3_living = 3*df['sqft_living'].std()
print(mean_living - std3_living, mean_living + std3_living)
```

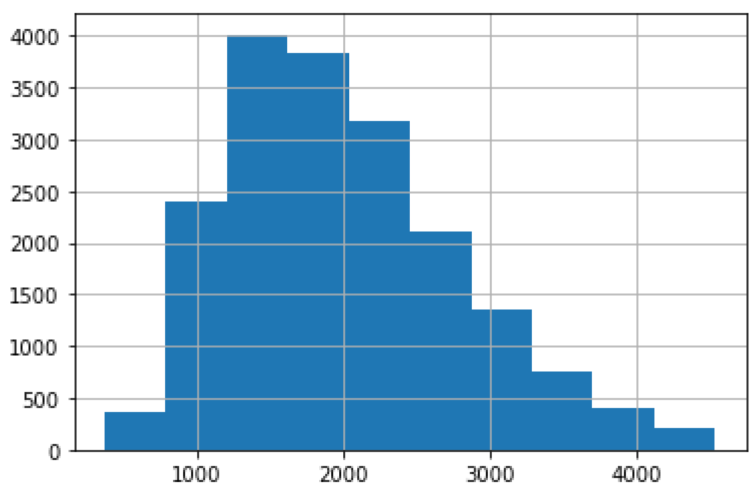
```
-479.600405407758 4546.544403705902
```

The min of *sqft_living* does not fall under 3 standard deviations below the mean. But just by observing the max, it is apparent that at least one entry exceeds 3 standard deviations above the column's mean. I filter out any values exceeding this upper limit.

In [37]:
```
upper_living = mean_living + std3_living
df = df[df['sqft_living'] <= upper_living]
```

In [38]:
```
df['sqft_living'].hist()
```

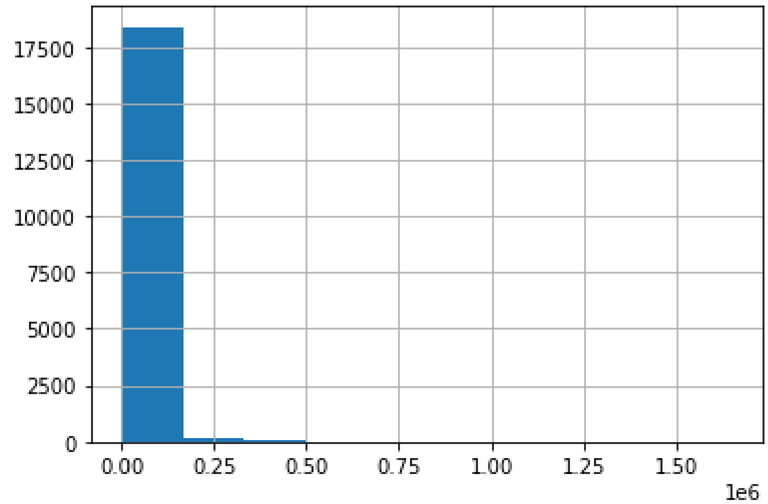
Out[38]: <AxesSubplot:>



Next, I look at the *sqft_lot* columns for any outliers.

In [39]:
```
df['sqft_lot'].hist()
```

Out[39]: <AxesSubplot:>



In [40]:
```
df['sqft_lot'].describe()
```

Out[40]:
```
count    1.863200e+04
mean     1.442871e+04
std      3.869237e+04
min      5.200000e+02
25%      5.000000e+03
50%      7.520000e+03
75%      1.039525e+04
max      1.651359e+06
Name: sqft_lot, dtype: float64
```

In [41]:
```
mean_lot = df['sqft_lot'].mean()
std3_lot = 3*df['sqft_lot'].std()
print(mean_lot - std3_lot, mean_lot + std3_lot)
```

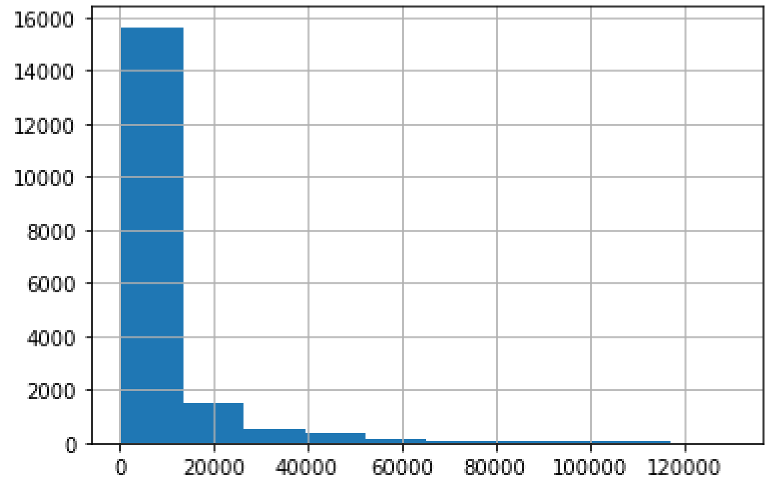
```
-101648.40775254855 130505.82767526215
```

Just like with *sqft_living*, the min *sqft_lot* value isn't 3 standard deviations or more below the mean. On the flip side, though, at least the column's max is more than 3 standard deviations above the mean. Similarly to before, I filter out any values that exceed this upper limit.

In [42]:
```
upper_lot = mean_lot + std3_lot
df = df[df['sqft_lot'] <= upper_lot]
```

In [43]:
```
df['sqft_lot'].hist()
```

Out[43]: <AxesSubplot:>



Finally, to keep up to date with the dataset, let's see if we still have a sufficient amount of entries (at least 15,000).

In [44]:
```
df.count()
```

Out[44]:
```
id             18344
price          18344
bedrooms       18344
sqft_living    18344
sqft_lot       18344
floors         18344
waterfront     18344
view           18344
condition      18344
yr_built       18344
zipcode        18344
dtype: int64
```

Now, I update the **df_pred** DataFrame after making the above changes to the main DataFrame **df.** Once this is done, I'm ready to run the new regression model.

In [45]:
```
df_pred = df.drop(['price', 'id'], axis=1)
```

In [46]:
```
outcome = 'price'
x_cols = df_pred.columns
predictors = '+'.join(x_cols)

f = outcome + '~' + predictors
```

In [47]:
```
model_2 = ols(formula=f, data=df).fit()
model_2.summary()
```

Out[47]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.510
Model:	OLS	**Adj. R-squared:**	0.510
Method:	Least Squares	**F-statistic:**	2120.
Date:	Tue, 26 Jan 2021	**Prob (F-statistic):**	0.00
Time:	00:46:04	**Log-Likelihood:**	-2.4779e+05
No. Observations:	18344	**AIC:**	4.956e+05
Df Residuals:	18334	**BIC:**	4.957e+05
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	8823.0658	2.72e+06	0.003	0.997	-5.33e+06	5.35e+06
bedrooms	-3.629e+04	1916.261	-18.935	0.000	-4e+04	-3.25e+04
sqft_living	237.4873	2.452	96.858	0.000	232.681	242.293
sqft_lot	-0.8521	0.107	-7.989	0.000	-1.061	-0.643
floors	7.65e+04	2991.009	25.575	0.000	7.06e+04	8.24e+04
waterfront	1.448e+05	2.16e+04	6.719	0.000	1.03e+05	1.87e+05
view	5.358e+04	2128.263	25.176	0.000	4.94e+04	5.78e+04
condition	2.018e+04	2208.811	9.136	0.000	1.58e+04	2.45e+04
yr_built	-1999.5005	59.079	-33.845	0.000	-2115.300	-1883.701
zipcode	39.7169	27.320	1.454	0.146	-13.833	93.266

Omnibus:	2808.019	**Durbin-Watson:**	1.968
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	7182.388
Skew:	0.856	**Prob(JB):**	0.00
Kurtosis:	5.543	**Cond. No.**	2.05e+08

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.05e+08. This might indicate that there are
strong multicollinearity or other numerical problems.

In [48]:
```
data = df.copy()

y = data['price']
X = data.drop(['price', 'id'], axis = 1)
```

In [49]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

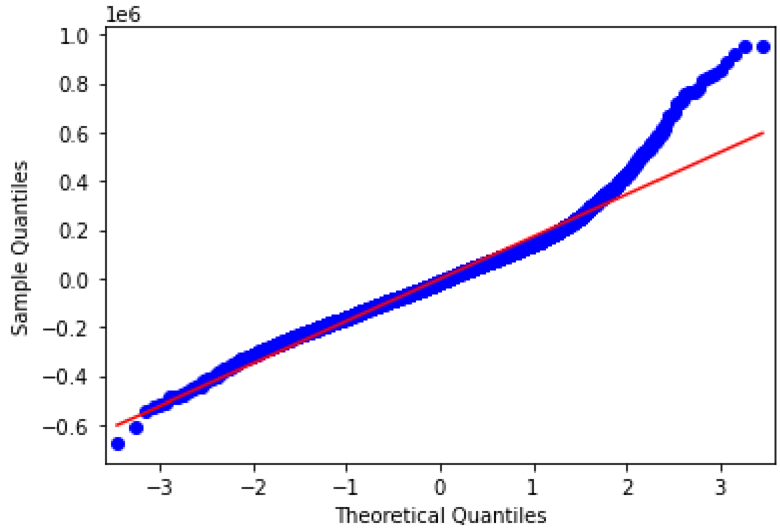
In [50]:
```
linreg = LinearRegression()
linreg.fit(X_train, y_train)

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

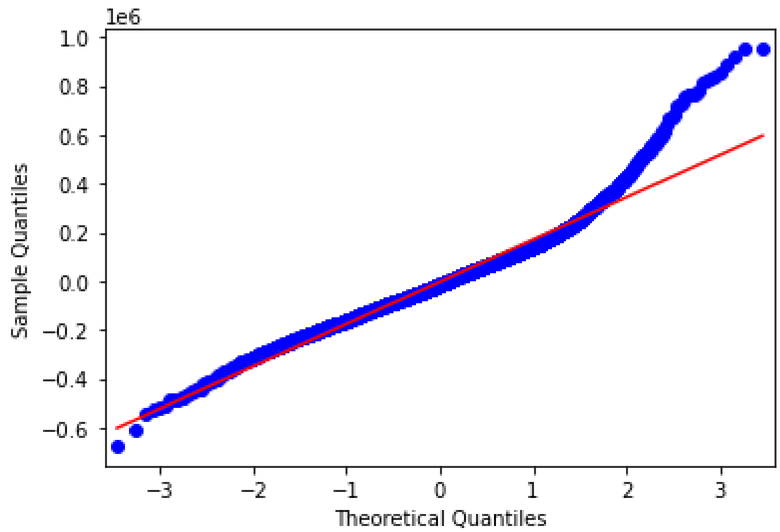
In [51]:
```
mse_train = mean_squared_error(y_train, y_hat_train)
mse_test = mean_squared_error(y_test, y_hat_test)

print('Train MSE:', mse_train)
print('Test MSE:', mse_test)

print('RMSE Train:', np.sqrt(mse_train))
print('RMSE Test:', np.sqrt(mse_test))
```

```
Train MSE: 31747789192.635834
Test MSE: 31235540437.570656
RMSE Train: 178179.09302899663
RMSE Test: 176735.79274603844
```

In [52]:
```
residuals = (y_test - y_hat_test)

sm.qqplot(residuals, line = "r")
```

Out[52]:





Getting rid of the outliers did lower the model's R-squared value from 0.597 to 0.510. On the flip side, though, the RMSE did imporve considerably (~175,000 here vs ~240,000 before). Additionally, the Q-Q Plot indicates that the residuals have become more Normally distributed, though they still have some right skew.

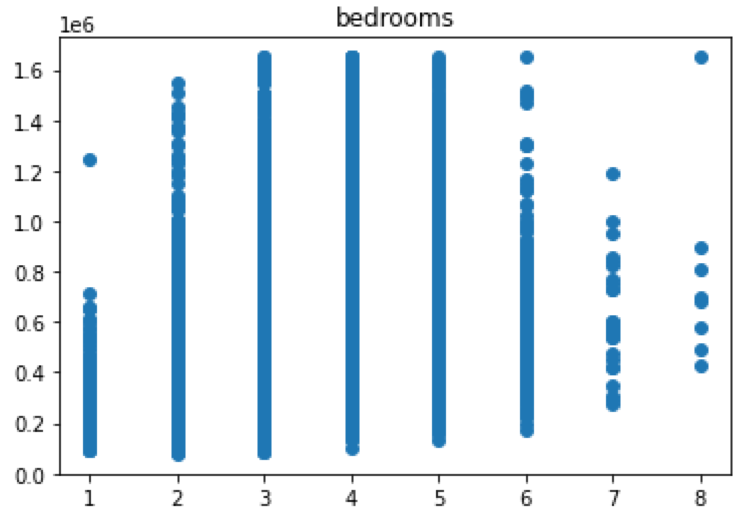
The *zipcode* feature does have a coefficient with a small t-statistic. This does not overly concern me, however, because I will soon get around to encoding this feature, which should eliminate the problem.

Ultimately, I choose to stick with the changes made here, in the hopes that transforming some of the features will sufficiently improve the R-squared value.

7 Model 3: Transforming Continuous Data

Let's look again at the scatterplots of individual predictors vs. home price:

In [53]:
```
for col in df_pred.columns:
    plt.scatter(df_pred[col], df['price'])
    plt.title(col)
    plt.show()
```

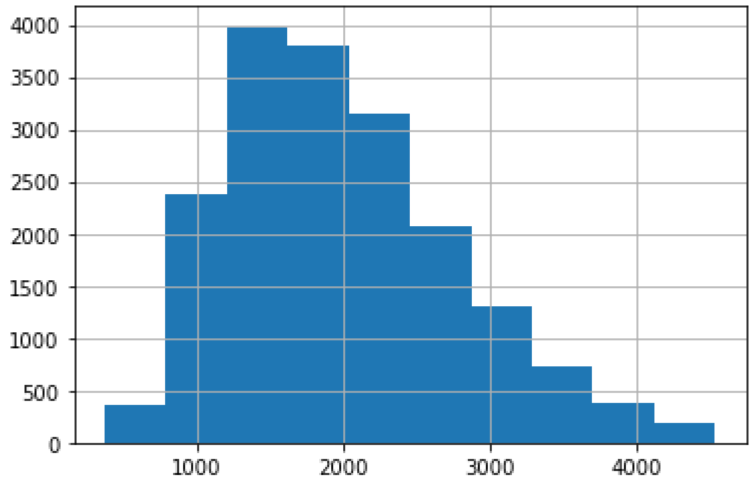


Here, it is apparent that the two continuous variables in the set of features are the *sqft_living* & *sqft_lot* columns. Both columns' scatterplots have a "cloud-like" appearence with no apparent vertically-aligned clusters.

Next, I take another look at the histograms for the two continuous predictors to see how their distribution looks.

In [54]:
```
df['sqft_living'].hist()
```

Out[54]: <AxesSubplot:>

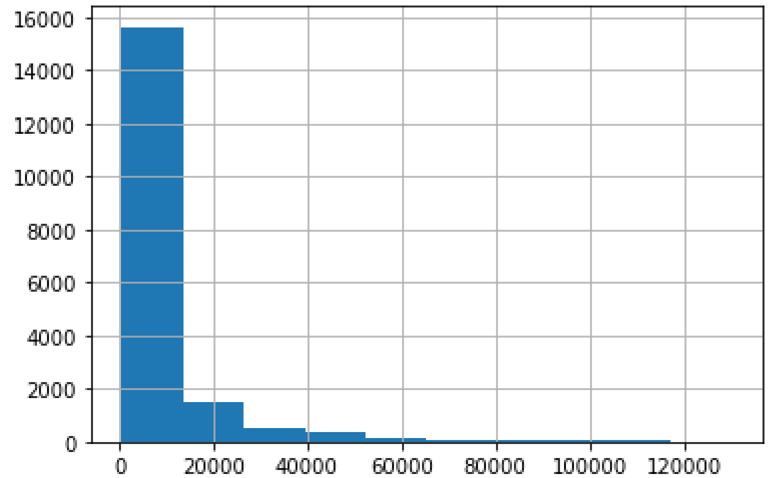


The *sqft_living* values seem to be distributed fairly normally, with a slight right skew. I determine that the distribution is good enough as is and does not require any transformation.

Next, I inspect the *sqft_lot* distribution.

In [55]:
```
df['sqft_lot'].hist()
```

Out[55]: <AxesSubplot:>

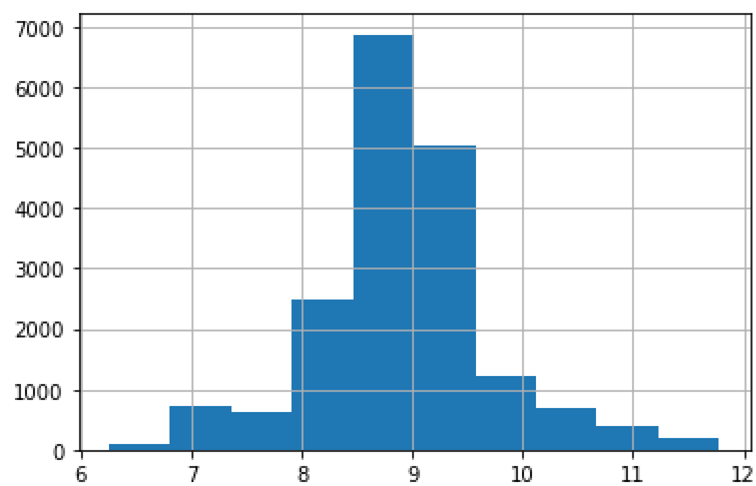


This predictor's distribution does not appear Normal at all. Additionally, from using the .describe() method on the column previously, I now know that all of its values are above 0. This makes the column a prime candidate for log-transformation.

In [56]:
```
df['sqft_lot'] = np.log(df['sqft_lot'])
```

In [57]:
```
df['sqft_lot'].hist()
```

Out[57]: <AxesSubplot:>



The distribution of the column's transformed values is much closer to a Normal one, and should lead to improvements in the next model iteration.

df['sqft_basement'] = (df['sqft_basement'] - mean_val) / basement_range

df['sqft_basement'].hist()

ignore cells above

In [58]:
```
df_pred = df.drop(['price', 'id'], axis=1)
```

In [59]:
```
outcome = 'price'
x_cols = df_pred.columns
predictors = '+'.join(x_cols)

f = outcome + '~' + predictors
```

In [60]:
```
model_3 = ols(formula=f, data=df).fit()
model_3.summary()
```

Out[60]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.517
Model:	OLS	**Adj. R-squared:**	0.517
Method:	Least Squares	**F-statistic:**	2181.
Date:	Tue, 26 Jan 2021	**Prob (F-statistic):**	0.00
Time:	00:46:09	**Log-Likelihood:**	-2.4765e+05
No. Observations:	18344	**AIC:**	4.953e+05
Df Residuals:	18334	**BIC:**	4.954e+05
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.023e+07	2.77e+06	3.689	0.000	4.79e+06	1.57e+07
bedrooms	-3.558e+04	1894.968	-18.775	0.000	-3.93e+04	-3.19e+04
sqft_living	249.7833	2.540	98.357	0.000	244.806	254.761
sqft_lot	-3.752e+04	2057.473	-18.235	0.000	-4.16e+04	-3.35e+04
floors	5.565e+04	3229.210	17.234	0.000	4.93e+04	6.2e+04
waterfront	1.672e+05	2.14e+04	7.799	0.000	1.25e+05	2.09e+05
view	5.346e+04	2112.888	25.302	0.000	4.93e+04	5.76e+04
condition	1.968e+04	2193.067	8.973	0.000	1.54e+04	2.4e+04
yr_built	-2053.5894	58.737	-34.962	0.000	-2168.720	-1938.459
zipcode	-60.0071	27.781	-2.160	0.031	-114.461	-5.553

Omnibus:	2850.958	**Durbin-Watson:**	1.970
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	7557.925
Skew:	0.856	**Prob(JB):**	0.00
Kurtosis:	5.638	**Cond. No.**	2.09e+08

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

In [61]:
```
data = df.copy()

y = data['price']
X = data.drop(['price', 'id'], axis = 1)
```

In [62]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

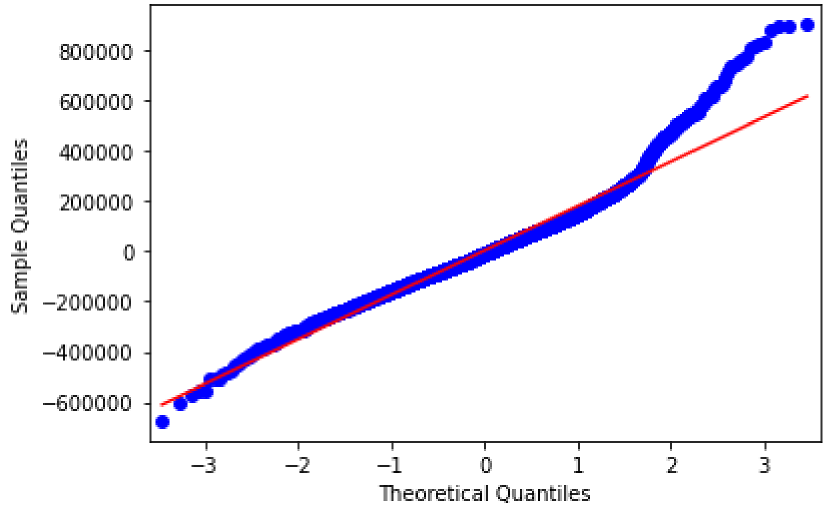
In [63]:
```
linreg = LinearRegression()
linreg.fit(X_train, y_train)

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

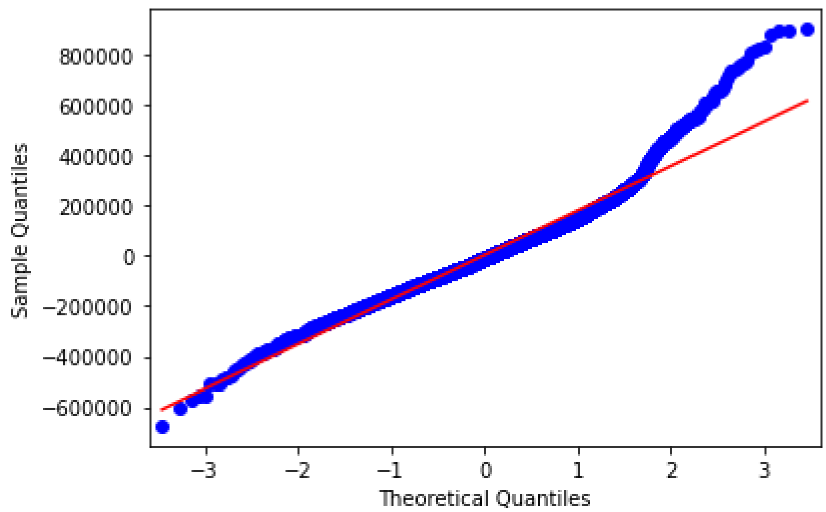
In [64]:
```
mse_train = mean_squared_error(y_train, y_hat_train)
mse_test = mean_squared_error(y_test, y_hat_test)

print('Train MSE:', mse_train)
print('Test MSE:', mse_test)

print('RMSE Train:', np.sqrt(mse_train))
print('RMSE Test:', np.sqrt(mse_test))
```

```
Train MSE: 30829252622.54272
Test MSE: 32645058539.010094
RMSE Train: 175582.60911190129
RMSE Test: 180679.43584982242
```

In [65]:
```
residuals = (y_test - y_hat_test)

sm.qqplot(residuals, line = "r")
```

Out[65]:





This latest model has multiple improvements, but they are relatively subtle.

The R-squared value increased slightly from 0.510 to 0.517. Additionally, none of the current features' coefficients have low t-scores.

8 Model 4: Dealing With Categorical Data

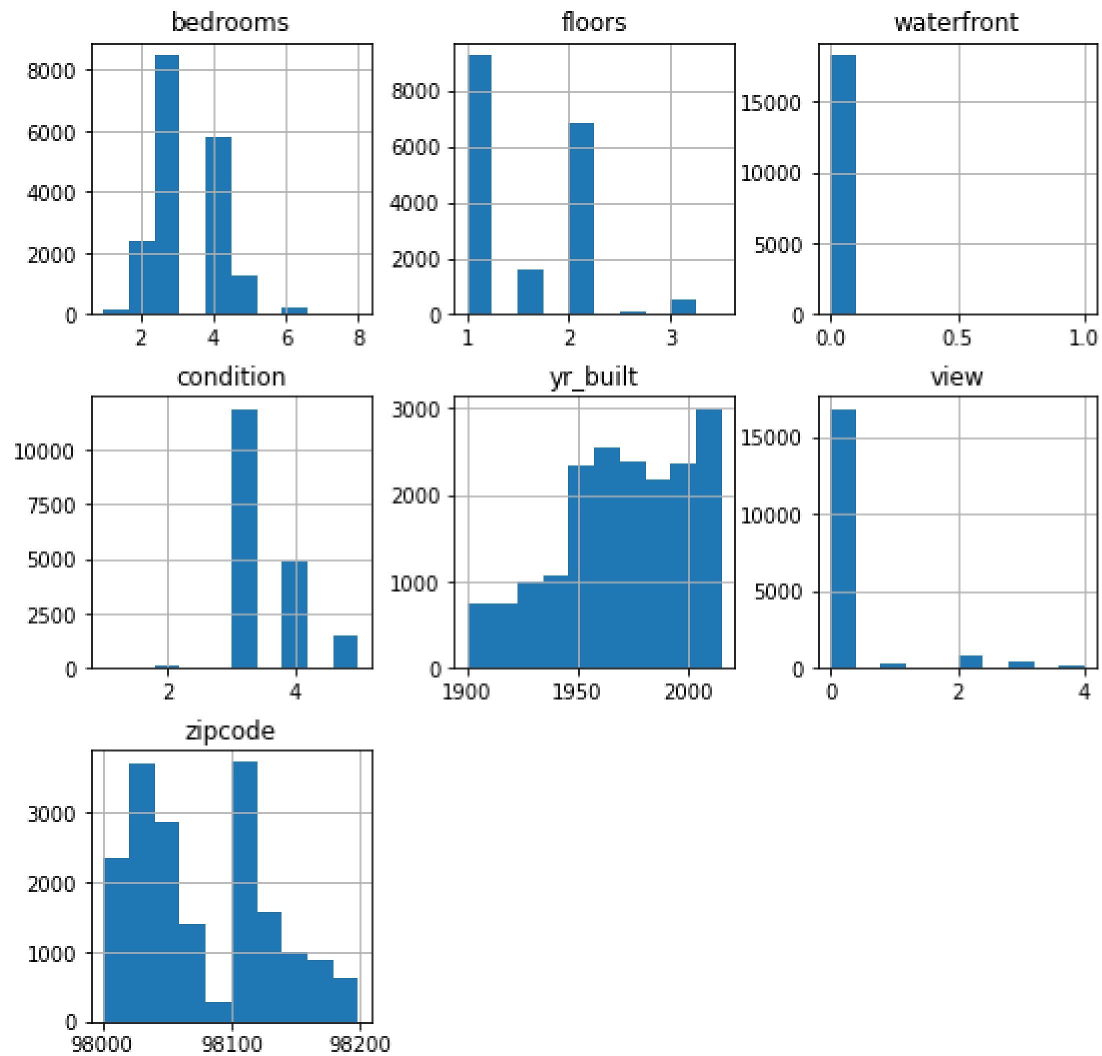
In [66]:
```
cat = ['bedrooms', 'floors', 'waterfront', 'condition', 'yr_built', 'view',
```

In [67]:
```
df[cat].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18344 entries, 1 to 21596
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   bedrooms    18344 non-null  int64  
 1   floors      18344 non-null  float64
 2   waterfront  18344 non-null  float64
 3   condition   18344 non-null  int64  
 4   yr_built    18344 non-null  int64  
 5   view        18344 non-null  float64
 6   zipcode     18344 non-null  int64  
dtypes: float64(3), int64(4)
memory usage: 1.1 MB
```

In [68]:
```python
df[cat].hist(figsize=(9,9))
```

Out[68]:
```
array([[<AxesSubplot:title={'center':'bedrooms'}>,
        <AxesSubplot:title={'center':'floors'}>,
        <AxesSubplot:title={'center':'waterfront'}>],
       [<AxesSubplot:title={'center':'condition'}>,
        <AxesSubplot:title={'center':'yr_built'}>,
        <AxesSubplot:title={'center':'view'}>],
       [<AxesSubplot:title={'center':'zipcode'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



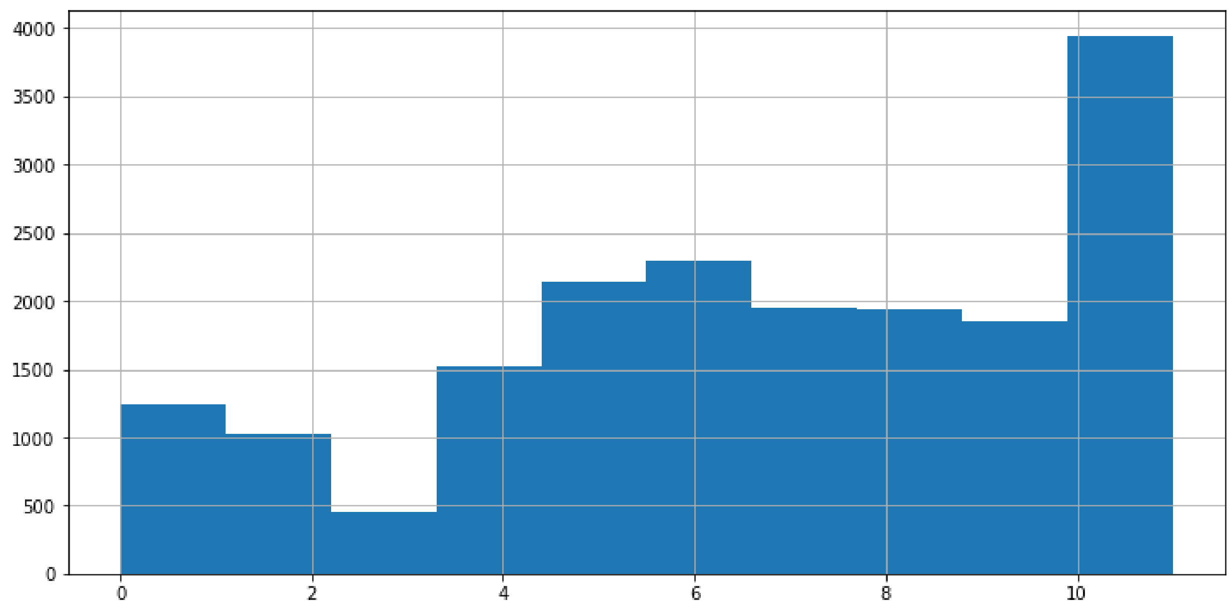
8.1 Binning The 'Year Built' Column Into Decades

In [69]:
```
decade_bins = [1900, 1910, 1920, 1930, 1940, 1950, 1960, 1970, 1980, 1990,
```

In [70]:
```
df['dec_built'] = pd.cut(df['yr_built'], bins=decade_bins,
                         right=False)
df['dec_built'] = df['dec_built'].cat.codes
```

In [71]:
```
df['dec_built'].hist(figsize=(12,6))
```

Out[71]: <AxesSubplot:>



8.2 One Hot Encoding The 'Condition', 'Floors' & 'Zipcode' Columns

In [72]:
```
cond_dummies = pd.get_dummies(df['condition'], prefix = 'cond', drop_first
floor_dummies = pd.get_dummies(df['floors'], prefix = 'floor', drop_first =
zip_dummies = pd.get_dummies(df['zipcode'], prefix = 'zip', drop_first = Tr
```

In [73]:
```
df_d = pd.concat([df, cond_dummies, floor_dummies, zip_dummies], axis=1)
```

In [74]:
```
df_d.columns = df_d.columns.str.replace('.','_')
df_d.drop(['condition', 'floors', 'zipcode', 'yr_built'], axis=1, inplace=T
```

In [75]:
```
df_d.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18344 entries, 1 to 21596
Data columns (total 86 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   id           18344 non-null  int64  
 1   price        18344 non-null  float64
 2   bedrooms     18344 non-null  int64  
 3   sqft_living  18344 non-null  int64  
 4   sqft_lot     18344 non-null  float64
 5   waterfront   18344 non-null  float64
 6   view         18344 non-null  float64
 7   dec_built    18344 non-null  int8   
 8   cond_2       18344 non-null  uint8  
 9   cond_3       18344 non-null  uint8  
 10  cond_4       18344 non-null  uint8  
 11  cond_5       18344 non-null  uint8  
 12  floor_1_5    18344 non-null  uint8  
 13  floor_2_0    18344 non-null  uint8  
 14  floor_2_5    18344 non-null  uint8  
 15  floor_3_0    18344 non-null  uint8  
 16  floor_3_5    18344 non-null  uint8  
 17  zip_98002    18344 non-null  uint8  
 18  zip_98003    18344 non-null  uint8  
 19  zip_98004    18344 non-null  uint8  
 20  zip_98005    18344 non-null  uint8  
 21  zip_98006    18344 non-null  uint8  
 22  zip_98007    18344 non-null  uint8  
 23  zip_98008    18344 non-null  uint8  
 24  zip_98010    18344 non-null  uint8  
 25  zip_98011    18344 non-null  uint8  
 26  zip_98014    18344 non-null  uint8  
 27  zip_98019    18344 non-null  uint8  
 28  zip_98022    18344 non-null  uint8  
 29  zip_98023    18344 non-null  uint8  
 30  zip_98024    18344 non-null  uint8  
 31  zip_98027    18344 non-null  uint8  
 32  zip_98028    18344 non-null  uint8  
 33  zip_98029    18344 non-null  uint8  
 34  zip_98030    18344 non-null  uint8  
 35  zip_98031    18344 non-null  uint8  
 36  zip_98032    18344 non-null  uint8  
 37  zip_98033    18344 non-null  uint8  
 38  zip_98034    18344 non-null  uint8  
 39  zip_98038    18344 non-null  uint8  
 40  zip_98039    18344 non-null  uint8  
 41  zip_98040    18344 non-null  uint8  
 42  zip_98042    18344 non-null  uint8  
 43  zip_98045    18344 non-null  uint8  
 44  zip_98052    18344 non-null  uint8  
 45  zip_98053    18344 non-null  uint8  
 46  zip_98055    18344 non-null  uint8  
 47  zip_98056    18344 non-null  uint8  
 48  zip_98058    18344 non-null  uint8  
 49  zip_98059    18344 non-null  uint8  
```

```
 50  zip_98065    18344 non-null  uint8
 51  zip_98070    18344 non-null  uint8
 52  zip_98072    18344 non-null  uint8
 53  zip_98074    18344 non-null  uint8
 54  zip_98075    18344 non-null  uint8
 55  zip_98077    18344 non-null  uint8
 56  zip_98092    18344 non-null  uint8
 57  zip_98102    18344 non-null  uint8
 58  zip_98103    18344 non-null  uint8
 59  zip_98105    18344 non-null  uint8
 60  zip_98106    18344 non-null  uint8
 61  zip_98107    18344 non-null  uint8
 62  zip_98108    18344 non-null  uint8
 63  zip_98109    18344 non-null  uint8
 64  zip_98112    18344 non-null  uint8
 65  zip_98115    18344 non-null  uint8
 66  zip_98116    18344 non-null  uint8
 67  zip_98117    18344 non-null  uint8
 68  zip_98118    18344 non-null  uint8
 69  zip_98119    18344 non-null  uint8
 70  zip_98122    18344 non-null  uint8
 71  zip_98125    18344 non-null  uint8
 72  zip_98126    18344 non-null  uint8
 73  zip_98133    18344 non-null  uint8
 74  zip_98136    18344 non-null  uint8
 75  zip_98144    18344 non-null  uint8
 76  zip_98146    18344 non-null  uint8
 77  zip_98148    18344 non-null  uint8
 78  zip_98155    18344 non-null  uint8
 79  zip_98166    18344 non-null  uint8
 80  zip_98168    18344 non-null  uint8
 81  zip_98177    18344 non-null  uint8
 82  zip_98178    18344 non-null  uint8
 83  zip_98188    18344 non-null  uint8
 84  zip_98198    18344 non-null  uint8
 85  zip_98199    18344 non-null  uint8
dtypes: float64(4), int64(3), int8(1), uint8(78)
memory usage: 2.5 MB
```

In [76]:
```
df_pred = df_d.drop(['price', 'id'], axis=1)
```

In [77]:
```
outcome = 'price'
x_cols = df_pred.columns
predictors = '+'.join(x_cols)

f = outcome + '~' + predictors
```

In [78]:
```
model_4 = ols(formula=f, data=df_d).fit()
model_4.summary()
```

Out[78]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.808
Model:	OLS	**Adj. R-squared:**	0.807
Method:	Least Squares	**F-statistic:**	913.3
Date:	Tue, 26 Jan 2021	**Prob (F-statistic):**	0.00
Time:	00:46:13	**Log-Likelihood:**	-2.3921e+05
No. Observations:	18344	**AIC:**	4.786e+05
Df Residuals:	18259	**BIC:**	4.792e+05
Df Model:	84		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]

The sizable jump in R-squared value is very encouraging, but before I proceed with the rest of modeling, I observe the t-scores (and their associated p-values) for the predictor coefficients.

A few of the p-values for dummy variable coefficients exceed what most would consider an acceptable cutoff of p=0.05. Consequently, I drop these dummy columns before re-running the model.

In [79]:
```
high_t_score = ['cond_2','floor_3_5','zip_98002','zip_98003','zip_98022','z
df_d.drop(high_t_score, axis=1, inplace=True)
```

In [80]:
```
df_pred = df_d.drop(['price', 'id'], axis=1)
```

In [81]:
```
outcome = 'price'
x_cols = df_pred.columns
predictors = '+'.join(x_cols)

f = outcome + '~' + predictors
```

In [82]:
```
model_4 = ols(formula=f, data=df_d).fit()
model_4.summary()
```

Out[82]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.808
Model:	OLS	**Adj. R-squared:**	0.807
Method:	Least Squares	**F-statistic:**	1022.
Date:	Tue, 26 Jan 2021	**Prob (F-statistic):**	0.00
Time:	00:46:15	**Log-Likelihood:**	-2.3921e+05
No. Observations:	18344	**AIC:**	4.786e+05
Df Residuals:	18268	**BIC:**	4.792e+05
Df Model:	75		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]

A couple more zipcode dummy columns have coefficients with p-values over 0.05. I drop these last columns before finishing my model.

In [83]:
```
df_d.drop(['zip_98042','zip_98070'], axis=1, inplace=True)
```

In [84]:
```
df_pred = df_d.drop(['price', 'id'], axis=1)
```

In [85]:
```
outcome = 'price'
x_cols = df_pred.columns
predictors = '+'.join(x_cols)

f = outcome + '~' + predictors
```

In [86]:
```
model_4 = ols(formula=f, data=df_d).fit()
model_4.summary()
```

Out[86]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.808
Model:	OLS	**Adj. R-squared:**	0.807
Method:	Least Squares	**F-statistic:**	1050.
Date:	Tue, 26 Jan 2021	**Prob (F-statistic):**	0.00
Time:	00:46:17	**Log-Likelihood:**	-2.3922e+05
No. Observations:	18344	**AIC:**	4.786e+05
Df Residuals:	18270	**BIC:**	4.792e+05
Df Model:	73		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]

In [87]:
```
data = df_d.copy()

y = data['price']
X = data.drop(['price', 'id'], axis = 1)
```

In [88]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

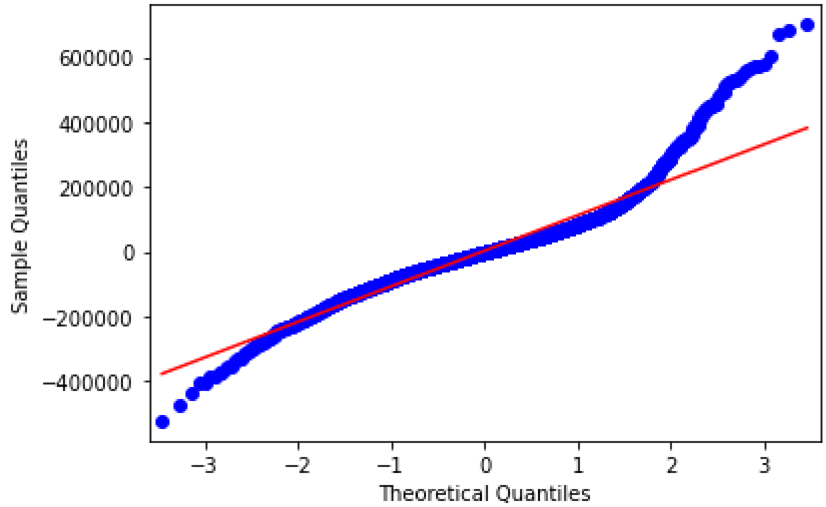
In [89]:
```
linreg = LinearRegression()
linreg.fit(X_train, y_train)

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

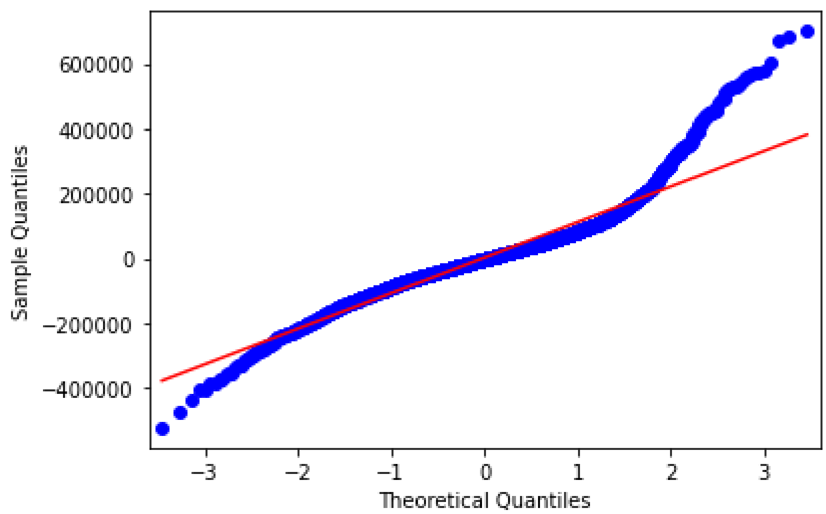
In [90]:
```
mse_train = mean_squared_error(y_train, y_hat_train)
mse_test = mean_squared_error(y_test, y_hat_test)

print('Train MSE:', mse_train)
print('Test MSE:', mse_test)

print('RMSE Train:', np.sqrt(mse_train))
print('RMSE Test:', np.sqrt(mse_test))
```

```
Train MSE: 12298396429.302235
Test MSE: 13033700783.34583
RMSE Train: 110898.13537342382
RMSE Test: 114165.23456528188
```

In [91]:
```
residuals = (y_test - y_hat_test)

sm.qqplot(residuals, line = "r")
```

Out[91]:





9 Final Model Evaluation

My final regression model, built to predict the price of a house in King County, can be evaluated using the following metrics.

- **R-squared = 0.808** : This value indicates that my model explains 80.8% of the variation in home prices from their mean value.

- **Root Mean Square Error (Using Test Data) = 114522** : On average, my model's predicted price is +/- $114,522 from the home's actual value.
- **Q-Q Plot (see above)** : From the plot, it seems that the residuals produced by my model have a relatively Normal distribution within 2 standard deviations of the mean. However, there is some skew, especially towards the right. In other words, my model is generally best at predicting the prices of homes that cost up to about $1,009,014.

10 Conclusions

My analysis leads to the following advice for any prospective first time home buyer in King County, WA:

- Clients looking to save on their home purchase could start by looking in these zip codes: 98198, 98188, 98031, 98038, 98178, 98168 & 98058.
- Conversely, clients looking to make a bigger investment could start by looking in these zip codes: 98039, 98004, 98119, 98112, 98109, 98102 & 98040.
- Clients looking to save money should consider the home's condition grade, as it seems to have a sizable impact on price. Even going from an average grade to a high grade can increase a home's price significantly.
- As expected, there seems to be significant correlation between a home's square footage & its price.

11 Future Work

Given more time, I would look at the features I had to initially omit. This would involve using the *lat* & *long* columns to further inspect the impact of specific locations (beyond zip codes) on price. Additionally, I would look at the *sqft_living15* & *sqft_lot15* columns to get insight on what neighborhoods are most expensive to live in and the overall impact of comparative size of neighbors' homes.

```
In [ ]:
```