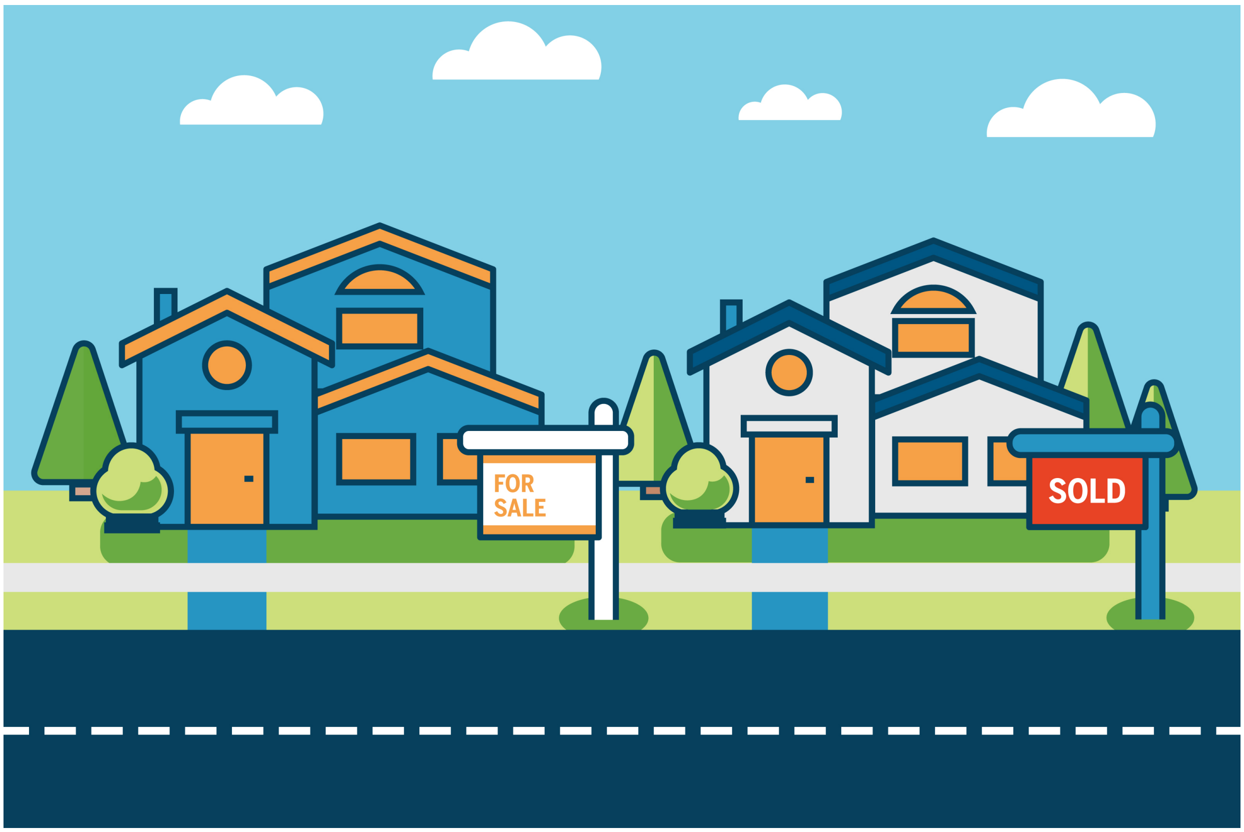
The Price is Right

Helping Homeowners Find The Right Priece to Sell Their Home

In [1]:
```
from IPython.display import Image
Image("/Users/matthewnykaza/Documents/Flatiron/Phase_2_Project/dsc-phase-2-
```

Out[1]:



Overview

For this project I was given, by the Flatiron school, a dataset consisting of the King Country Housing sales over a period of May 2014 to May 2015. I was tasked with finding a relevant business problem and conducting analysis using the given data and conducting statistical models to answer the business question. I decided to try and help basic homeowners find the appropriate price to list their home at, given basic variable (bedrooms, bathrooms, lot size etc.). I began this process by preprocessing the data by taking out unneccesary variables (i.e. views, sqft basement, etc.) and changing some of the variables (changing basement size to whether or not the home has a basement for example). Once this basic data cleaning was complete I set out to create a base model just to see what the correlations, r2 score and other relevant testing metrics looked like before doing any standardization techniques. With that subpar model complete I began to change the data using standardization and normalizing techniques. This started with log-transforming the target variable (selling price) which turned that variable into a much more normal distribution. Following that I used a standard scaler from sklearn which removed the mean from the independant variables and scaled to the variance. This did not necesarily make the data "more normal", but it did bring center the data around a mean of 0 with a standard deviation of 1. This did improve on my model somewhat, but overall did not make my prediction power (as read by R2 score) any stronger. My final model featured log-transformed, and Standard Scaled data, which did not improve the R2 score, but did provide the least error, and gives me confidence that there is something strong to go on here. I recommend that a home seller not use this model quite yet, but with more time I am confident that there can be a great conclusion reached, and a accurate home selling model.

Business Problem

Selling a home can be an exceedingly stressful process. One of the biggest pain points for home sellers is selecting the correct listing price for that home. Pick a price that is too low, and you run the risk of losing out on a lot of potential earnings. To high and you can wind up with a house that remains on the market for a long time, potentially hamstringing efforts to move into that new house. With this dataset, and statistical knowledge I intend to create a model that will assist with people knowing the right price to list their home at.

In [72]:
```
#Start with loading in all the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.preprocessing import PolynomialFeatures
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import Lasso, Ridge, LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, make_scorer
from sklearn.feature_selection import RFE
from scipy import stats
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

King Count Housing Dataset

- In this modeling I used data provided to me by Flatiron School. It consists of a single dataset that include relevant information about housing sales from May 2014 to May 2015 in King County, WA.
- There are many variables in this dataset and they include home ID (a simple identifier unique to each sold home), selling price, number of bedrooms and bathrooms, square feet of the living space, squarefeet of the total lot, number of floors, whether the home is on the waterfront or not, whether the home has been viewed or not, a overall condition rating, a King County

Housing grade, year built, year remodeled, zipcode it is in, the latitude and longitude of the location, and the square feet of the 15 surrounding homes lots and living space.
- The target variable for this modeling will be the selling price ('price' in the data).
- The feature variables that I will be using for this project will be bedrooms, bathrooms, sqft_lot, floors, waterfront, condition, grade, and year built.

In [73]:
```
#Load in the dataset and take a look at its overview
data = pd.read_csv('data/kc_house_data.csv')
```

In [74]:
```
#Take a look at the first 5 rows of data
data.head()
```

Out[74]:

	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

In [75]:
```
#Check out the columns
data.columns
```

Out[75]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
        'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
        'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcod
e',
        'lat', 'long', 'sqft_living15', 'sqft_lot15'],
       dtype='object')
```

In [76]:
```
#Check out the basic stats for the dataset
data.describe()
```

Out[76]:

	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.

A few things noticed here

1. Outliers in a bunch of the categories
   - Especially bedrooms, bathrooms, sqft_lot and sqft_lot15

In [77]:
```
#Look for any NaN data
data.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
```

A few things noticed here

1. We have a few NaNs
2. date and swft_basement are the only two columns with a data type as "object", this makes sense for date, but why sqft_basement?

In [78]:
```
#As show in data.info() sqft_basement is an object, but why
print(data.groupby('sqft_basement')['id'].nunique())
```

```
sqft_basement
0.0       12718
10.0          1
100.0        42
1000.0      146
1008.0        1
          ...
960.0        65
970.0        44
980.0        55
990.0        51
?           454
Name: id, Length: 304, dtype: int64
```

Some 454 are listed as ?, we're just going to make an assumption that this means they don't have a basement because in my opinion it is pretty easy for someone to tell if a place does or not, we will make these values 0.

In [79]:
```
#Change '?' to 0 in sqft_basement and check
data['sqft_basement'] = data['sqft_basement'].replace(['?'], 0)
```

In [80]:
```
#Sanity check
data.loc[lambda df: df['sqft_basement']== '?']
```

Out[80]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	s

0 rows × 21 columns

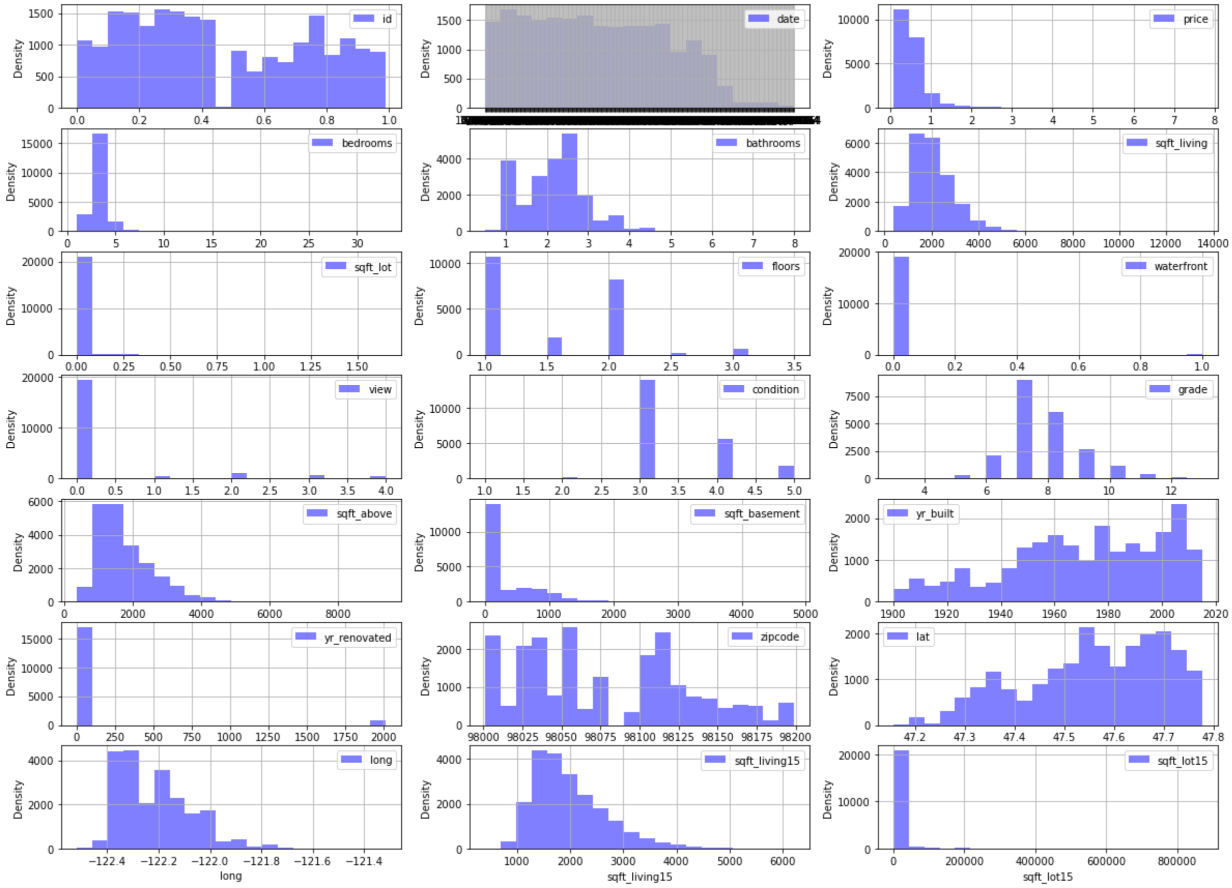
In [81]:
```
#Now change them to numeric in the same way the other sqft variables are
data['sqft_basement'] = data[['sqft_basement']].apply(pd.to_numeric)
```

In [82]:
```
#Sanity check
data['sqft_basement'].dtype
```

Out[82]: dtype('float64')

In [83]:
```
# Idea on distribution of data with a histogram
#Going to createa a function that will make this process easier in the futu
#times throughout the process
#Lets create a histogram to check for normalality
def histogram_view(data_set):
    """This will produce 3 columns of histograms and a corresponding number
    of rows depending on the len(data_set.columns). The result will be a
    side by side view of all of the histograms of each column in a data set
    ncols = 3
    nrows = int(np.ceil(len(data_set.columns) / (1.0*ncols)))
    fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(20, 15))
    # Lazy counter so we can remove unwated axes
    counter = 0
    for i in range(nrows):
        for j in range(ncols):

            ax = axes[i][j]

            # Plot when we have data
            if counter < len(data_set.columns):

                ax.hist(data_set[data_set.columns[counter]], bins=20, color
                ax.set_xlabel('{}'.format(data_set.columns[counter]))
                ax.set_ylabel('Density')
                leg = ax.legend(loc='best')
                leg.draw_frame(True)
                ax.grid(which='both', axis='both', linestyle='-')
            # Remove axis when we no longer have data
            else:
                ax.set_axis_off()

            counter += 1

    plt.show()
```

In [84]:
```
histogram_view(data)
```



We can already get the sense that much of the data is far from normal especially in the cases of id, date, and sqft_basement. For now we are not worrying about the normalilty of the categorical variables which look to be bedrooms, bathrooms, floors, waterfront, view, condition, grade, yr_built, yr_renovated, zipcode, lat and long.

Data preprocessing

- First I just wanted to make a basic check for any duplicates in the data
- NaN data was addressed and major outliers were identified and corrected accordingly
- Upon starting this project I decided to drop the date, view, sqft_above, yr_renovaged, zipcode, lat, long, sqft_living15 and sqft_lot15 the reasoning for that can be seen below.
- After dropping those named variables I went to check about any NaN data.
-
    - The only category was the Waterfront category, and handling of that issue can be seen below.
- Then I took a look for outliers which

Check for Duplicates

- Will be using the ID column for this

In [85]:
```
duplicates = data[data.id.duplicated(keep=False)]
duplicates
```

Out[85]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	water
93	6021501535	7/25/2014	430000.0	3	1.50	1580	5000	1.0	
94	6021501535	12/23/2014	700000.0	3	1.50	1580	5000	1.0	
313	4139480200	6/18/2014	1380000.0	4	3.25	4290	12103	1.0	
314	4139480200	12/9/2014	1400000.0	4	3.25	4290	12103	1.0	
324	7520000520	9/5/2014	232000.0	2	1.00	1240	12092	1.0	
...	...	...	...	...	...	...	...	...	
20654	8564860270	3/30/2015	502000.0	4	2.50	2680	5539	2.0	
20763	6300000226	6/26/2014	240000.0	4	1.00	1200	2171	1.5	
20764	6300000226	5/4/2015	380000.0	4	1.00	1200	2171	1.5	
21564	7853420110	10/3/2014	594866.0	3	3.00	2780	6000	2.0	
21565	7853420110	5/4/2015	625000.0	3	3.00	2780	6000	2.0	

353 rows × 21 columns

There looks to be some duplicates, but all of them seem to make sense and simply looks like they are the same home being resold numerous times. With the duplicates only representing roughly 2% of the data we are not going to worry about them.

NaN Data

In [86]:
```
#Dealing with NaN's
data.info()
#Looks to be only the waterfront column, so lets dig into that one
```

```
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64
 6   sqft_lot       21597 non-null  int64
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  float64
 9   view           21534 non-null  float64
 10  condition      21597 non-null  int64
 11  grade          21597 non-null  int64
 12  sqft_above     21597 non-null  int64
 13  sqft_basement  21597 non-null  float64
 14  yr_built       21597 non-null  int64
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64
 20  sqft_lot15     21597 non-null  int64
dtypes: float64(9), int64(11), object(1)
memory usage: 3.5+ MB
```

In [87]:
```
#Let's do a quick check on the counts of the waterfront column to give us a
print(data.groupby('waterfront')['id'].nunique())
```

```
waterfront
0.0    18941
1.0      146
Name: id, dtype: int64
```

As determined by the histogram the waterfront variable is a categorical variable. There are roughly 2,000 NaNs in that variable, but with only 146 waterfront homes in total on the dataset I believe that instead of removing the NaNs it is pretty safe to assume that these NaNs are non-waterfront homes and will simply fill them in at 0 indicating they are not on a waterfront.

In future modeling this can be improved by comparing the NaNs and their lat and long values to verify whether or not this is actually the case.

In [88]:
```
#Fill with 0 and sanity check
data['waterfront'] = data['waterfront'].fillna(0)
data.info()
```

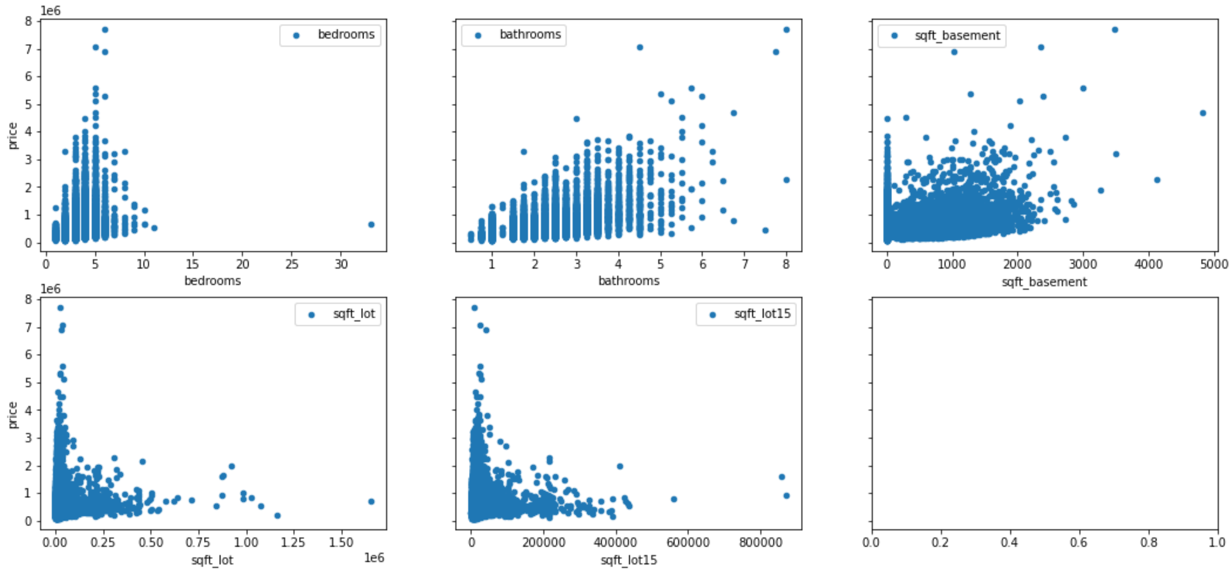
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 1   date           21597 non-null  object 
 2   price          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     21597 non-null  float64
 9   view           21534 non-null  float64
 10  condition      21597 non-null  int64  
 11  grade          21597 non-null  int64  
 12  sqft_above     21597 non-null  int64  
 13  sqft_basement  21597 non-null  float64
```

We have confirmed that all outliers were successfully removed from waterfront, now we do have some in the view and yr_renovated categories, but we will not worry about those will be removed, but more on that later.

Outliers

We're just going to look at the previously highlighted columns of bedrooms, bathrooms, sqft_lot and sqft_lot15

In [89]:
```
outlier_col = ['bedrooms', 'bathrooms', 'sqft_basement', 'sqft_lot', 'sqft_
fig, axes = plt.subplots(2,3,sharey=True, figsize=(18,8))
axe  = axes.ravel()
for idx, column in enumerate(data[outlier_col]):
    data.plot(kind='scatter', x=column, y='price', ax=axe[idx], label=colum
```



1. bedrooms has a clear outlier of 33, going to remove that one
2. bathrooms has a few outliers as well, but those are a little closer together, and I think we'll have a different method of dealing with that variable
3. sqft_basement certainly has some outliers, but they too do not look too out of whack, and we have a different plan for that variable as to be seen later
4. One major outlier here, probably going to remove that one
5. Two major outliers here, same as sqft_lot, going to remove those

A quick look shows that there are some outliers in bedrooms, bathrooms, sqft_living, and sqft_loft. I will look at the bedrooms and bathrooms columns to get an idea of what the outliers look like.

In [90]:
```
#Get a closer look at the bathrooms and bedrooms columns (should be easier
print(data.groupby('bedrooms')['id'].nunique())
print(data.groupby('bathrooms')['id'].nunique())
```

```
bedrooms
1      191
2     2736
3     9731
4     6849
5     1586
6      265
7       38
8       13
9        6
10       3
11       1
33       1
Name: id, dtype: int64
bathrooms
0.50       4
0.75      70
1.00    3794
1.25       9
1.50    1429
1.75    3020
2.00    1913
2.25    2031
2.50    5352
2.75    1182
3.00     747
3.25     586
3.50     729
3.75     155
4.00     134
4.25      79
4.50      99
4.75      23
5.00      21
5.25      13
5.50      10
5.75       4
6.00       6
6.25       2
6.50       2
6.75       2
7.50       1
7.75       1
8.00       2
Name: id, dtype: int64
```

For the data the bathrooms are consistent from .5 to 8 while there is a pretty heavy tail, I think that using some preprocessing techniques will be the real answer here. For the bedrooms the 33 bedroom home is a massive outlier, and I will simply take that one out of the equation.

In [91]:
```
#Take all the bathrooms and round them to the nearest full number, this wil
#that have very little values in them.
data.loc[data['bathrooms'] <= 1.25, 'bathrooms'] = 1
data.loc[(data['bathrooms'] >=1.25) & (data['bathrooms']<= 2.25), 'bathroom
data.loc[(data['bathrooms'] >=2.25) & (data['bathrooms']<= 3.25), 'bathroom
data.loc[(data['bathrooms'] >=3.25) & (data['bathrooms']<= 4.25), 'bathroom
data.loc[(data['bathrooms'] >=4.25) & (data['bathrooms']<= 5.25), 'bathroom
data.loc[(data['bathrooms'] >=5.25) & (data['bathrooms']<= 6.25), 'bathroom
data.loc[(data['bathrooms'] >=6.25) & (data['bathrooms']<= 7.25), 'bathroom
data.loc[(data['bathrooms'] >=7.25) & (data['bathrooms']<= 8.25), 'bathroom
```

In [92]:
```
#Sanity check
print(data.groupby('bathrooms')['id'].nunique())
```

```
bathrooms
1.0    3877
2.0    8393
3.0    7867
4.0    1097
5.0     156
6.0      22
7.0       4
8.0       4
Name: id, dtype: int64
```

In [93]:
```
#Just remove that 33 bedroom outlier, that might even just be mistyped or a
#Effect the model.
data = data[data.bedrooms != 33]
data.head()
```

Out[93]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
0	7129300520	10/13/2014	221900.0	3	1.0	1180	5650	1.0	0.0
1	6414100192	12/9/2014	538000.0	3	2.0	2570	7242	2.0	0.0
2	5631500400	2/25/2015	180000.0	2	1.0	770	10000	1.0	0.0
3	2487200875	12/9/2014	604000.0	4	3.0	1960	5000	1.0	0.0
4	1954400510	2/18/2015	510000.0	3	2.0	1680	8080	1.0	0.0

5 rows × 21 columns

Bedrooms and bathrooms outliers have been taken care of, now we'll move on to our sqft_lot and sqft_lot15

In [94]:
```
data.sort_values(by=['sqft_lot'], ascending=False)
```

Out[94]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	water
1717	1020069017	3/27/2015	700000.0	4	1.0	1300	1651359	1.0	
17305	3326079016	5/4/2015	190000.0	2	1.0	710	1164794	1.0	
7640	2623069031	5/21/2014	542500.0	5	3.0	3010	1074218	1.5	
7762	2323089009	1/19/2015	855000.0	4	4.0	4030	1024068	2.0	
3945	722069232	9/5/2014	998000.0	4	3.0	3770	982998	2.0	
...	...	...	...	...	...	...	...	...	
20588	7899800857	12/15/2014	256950.0	2	2.0	1070	635	2.0	
3449	2559950110	4/22/2015	1230000.0	2	3.0	2470	609	3.0	
7582	6371000026	1/22/2015	367500.0	2	2.0	1030	600	2.0	
5821	1773101159	1/7/2015	250000.0	3	2.0	1050	572	2.0	
15729	9828702895	10/22/2014	700000.0	4	2.0	2420	520	1.5	

21596 rows × 21 columns

Looks like that one major 5000 sqft outlier, let's remove it as to mitigate any muddiness of the data later on

In [95]:
```
#remove and check for removal
data = data[data.sqft_lot != 1651359]
data.loc[lambda df: df['sqft_lot']== 1651359]
```

Out[95]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	s

0 rows × 21 columns

In [96]:
```
#Do the same as above for sqft_lot15
data.sort_values(by=['sqft_lot15'], ascending=False)
```

Out[96]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wate
9705	225079036	1/7/2015	937500.0	4	4.0	5545	871200	2.0	
20436	1125079111	4/15/2015	1600000.0	4	6.0	6530	871200	2.0	
13451	3420069060	11/7/2014	790000.0	3	3.0	2640	432036	1.5	
8655	3226079059	10/19/2014	549950.0	3	2.0	2930	266587	2.0	
3797	1550000463	8/26/2014	637000.0	4	4.0	3080	118918	2.0	
...	...	...	...	...	...	...	...	...	
20891	8562780540	12/22/2014	325000.0	2	2.0	1150	711	2.0	
20999	8562780530	3/28/2015	338500.0	2	2.0	1150	711	2.0	
513	2827100070	11/5/2014	290000.0	4	1.0	1330	8184	1.5	
20733	2827100075	7/27/2014	286308.0	2	2.0	1220	1036	3.0	

This time we have two outliers, but they two are by a considerable margin so we will remove them from the data

In [97]:
```
data = data[data.sqft_lot15 != 871200]
data = data[data.sqft_lot15 != 858132]
data.loc[lambda df: df['sqft_lot15']> 858131]
```

Out[97]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	s

0 rows × 21 columns

Check Assumptions

1. Linearity - Needs a linear relationship between x and y to get linear regression
2. Normality - Residuals are normally distributed
    - Check after modeling
3. No multicollinearity = independant variables not highly correlated
4. Homoscedasticity - variance of error terms are similar across the values of the independant variables
    - Check after modeling

In [98]:
```
#Let's create scatterplots of all our data to get an idea linear relationsh
#We can also use this to check our outliers
def scatter_view(data_set):
    ncols = 3
    nrows = int(np.ceil(len(data_set.columns) / (1.0*ncols)))
    fig, axes = plt.subplots(nrows=nrows, ncols=ncols, sharey=True, figsize
    axe  = axes.ravel()
    for idx, column in enumerate(data_set.columns):
        data_set.plot(kind='scatter', x=column, y='price', ax=axe[idx], lab
```

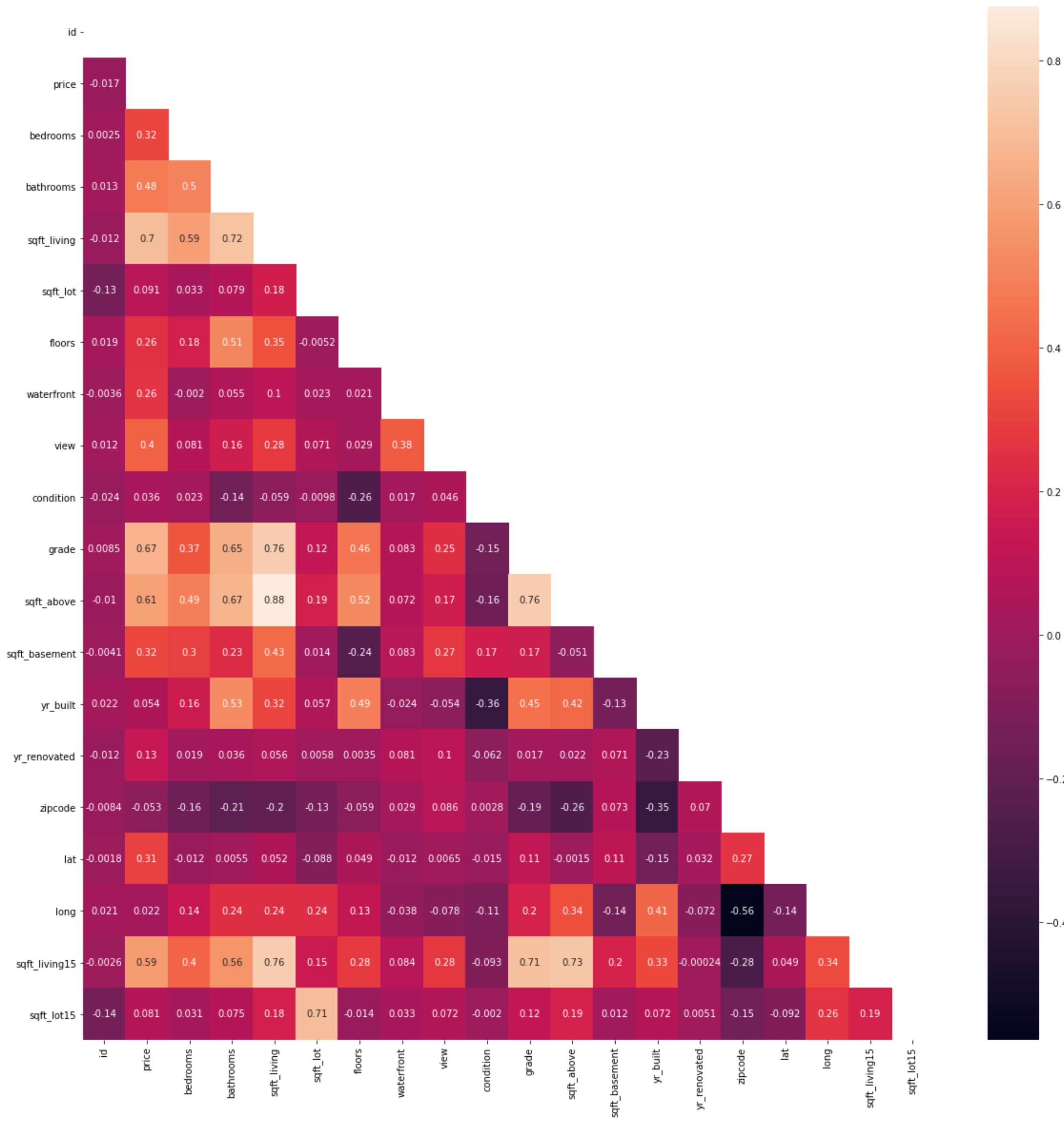
In [99]:
```
scatter_view(data)
```

Some takeaways

- Our outliers look mostly mitigated, especially in the columns we are most concerned about
- id, date, yr_built, yr_renovated, zipcode, lat, long and sqft_lot15 don't seem to have much of a correlation

In [100]:
```python
#Heatmap to double check
plt.figure(figsize=(20,20))
matrix = np.triu(data.corr())
sns.heatmap(data.corr(), annot=True, mask=matrix)
```

Out[100]: <AxesSubplot:>



With this heatmap we are simply looking for any bit of correlation along the price

- 1 is perfect correltaion
- .5 is medium

- .3 and less is small to nill
- 0 is no correlation

While this would not be exactly normal we are going to set an alpha level of .2 for our correlation, anything less will be tossed out, anything more will be kept Lost features - id, date (is an object and won't show here), sqft_lot, condition, yr_built, yr_renovated, long, lat (while it does correlate, it doesn't really do us much good if we don't have long), sqft_lot15

We will additionally remove view, as our business problem is trying to help people find a price to list their homes at there is no way for them to know how many views they will get.

In [101]:
```
unneeded columns to make data easier to digest, with reasoning
op(['id','date', 'view', 'sqft_lot', 'condition', 'yr_built', 'yr_renovated'
ad()
```

Out[101]:

	price	bedrooms	bathrooms	sqft_living	floors	waterfront	grade	sqft_above	sqft_basemen
0	221900.0	3	1.0	1180	1.0	0.0	7	1180	0.(
1	538000.0	3	2.0	2570	2.0	0.0	7	2170	400.(
2	180000.0	2	1.0	770	1.0	0.0	6	770	0.(
3	604000.0	4	3.0	1960	1.0	0.0	7	1050	910.(
4	510000.0	3	2.0	1680	1.0	0.0	8	1680	0.(

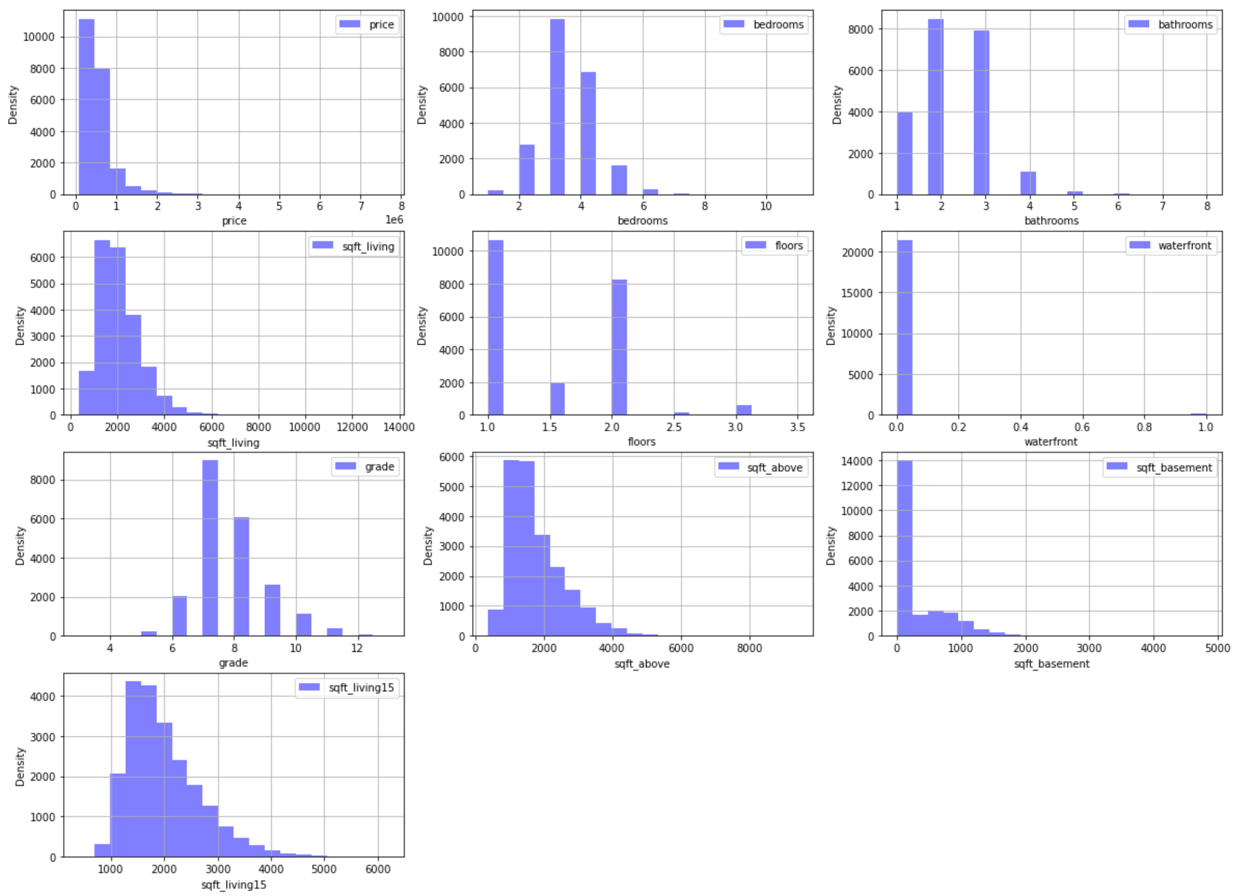
Reasoning for removing columns

- id
  - Now that we have checked duplicates we no longer really need the id as there is no correlation
- date
  - Will be difficult to put into a linear model, as well as be hard to have any use going into the future
- view
  - Does not fit our business problem
- sqft_lot and sqft_lot15
  - Looking at correlations it does not appear that lot has much of an impact on selling price
    - Mind you I am sure this is all location based, and if we had more time to refine the model we could reinclude this.
- yr_built
  - Scatter and heatmap do not show much of a correlation here
- yr_renovated
  - There is a lot of NaN values here, and it will be difficult to select a proper metric for that NaN data that will not end up skewing our model
- zipcode
  - There are greater than 70 zipcodes in the dataset, and for the current project it will be too time consuming to put them into different bins and model on that. Again, could be good

for future modeling
- Likely helpful in further investigations

- lat, long
  - Each home has it's own value, and otherwise similar reasoning as the zipcode column
    - A point to look at for further investigation

In [102]:
```
#Lets take an overview of our remaining data
histogram_view(data)
```



Most of the data at least looks somewhat normal, at least we should be able to standardize and such to get better results

Dealing with the massive tail of sqft_basement

In [103]:
```
#sqft_basement has many values that are 0, going to create a new column tha
#for no basement, and 1 for has basement (which will be determined as 0 sqf
#a basement)
conditions = [(data['sqft_basement']==0), (data['sqft_basement'] > 0)]
values = [0,1]
data['basement'] = np.select(conditions, values)
data.head()
```

Out[103]:

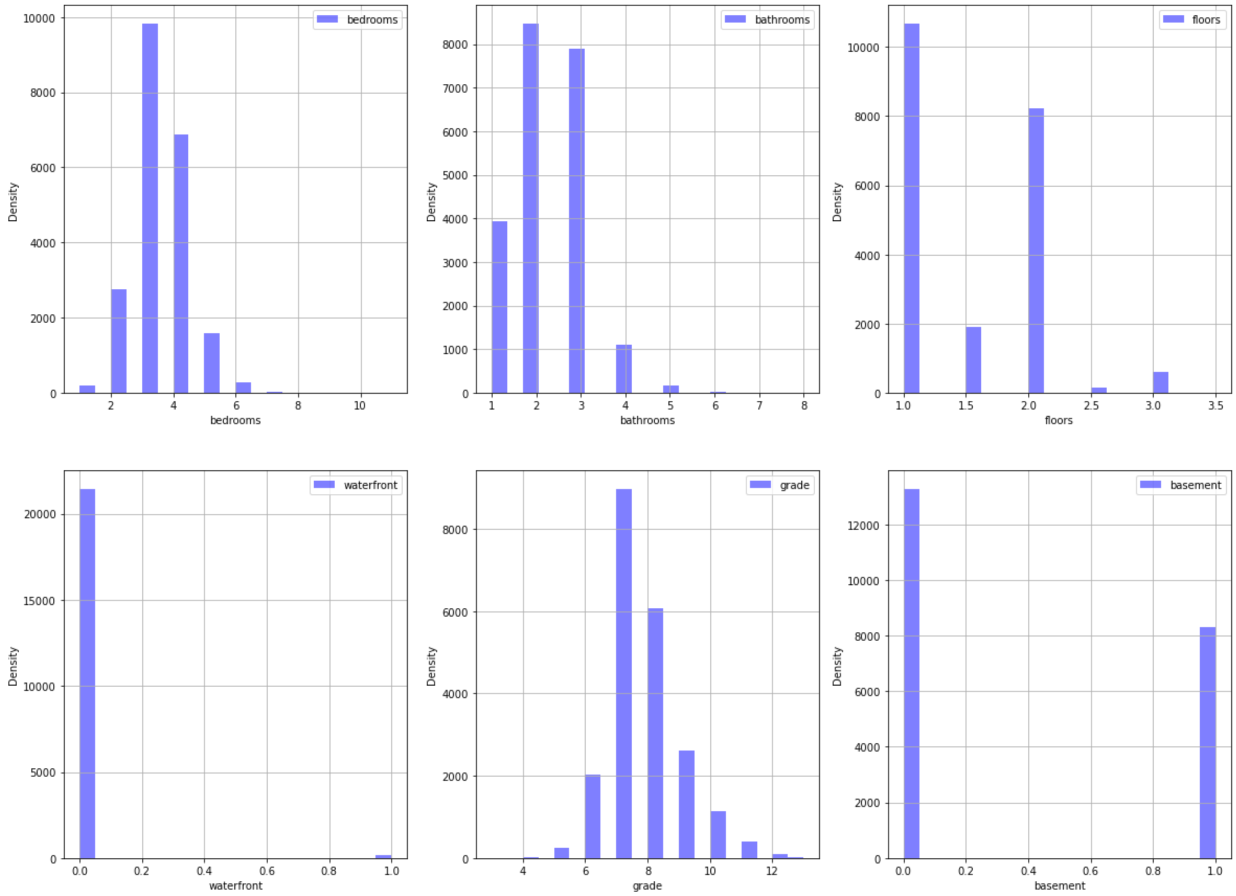
	price	bedrooms	bathrooms	sqft_living	floors	waterfront	grade	sqft_above	sqft_basemen
0	221900.0	3	1.0	1180	1.0	0.0	7	1180	0.(
1	538000.0	3	2.0	2570	2.0	0.0	7	2170	400.(
2	180000.0	2	1.0	770	1.0	0.0	6	770	0.(
3	604000.0	4	3.0	1960	1.0	0.0	7	1050	910.(
4	510000.0	3	2.0	1680	1.0	0.0	8	1680	0.(

In [104]:
```
#Drop obsolete sqft_basement column
data.drop(['sqft_basement'], axis=1, inplace=True)
#Sanity check
data.columns
```

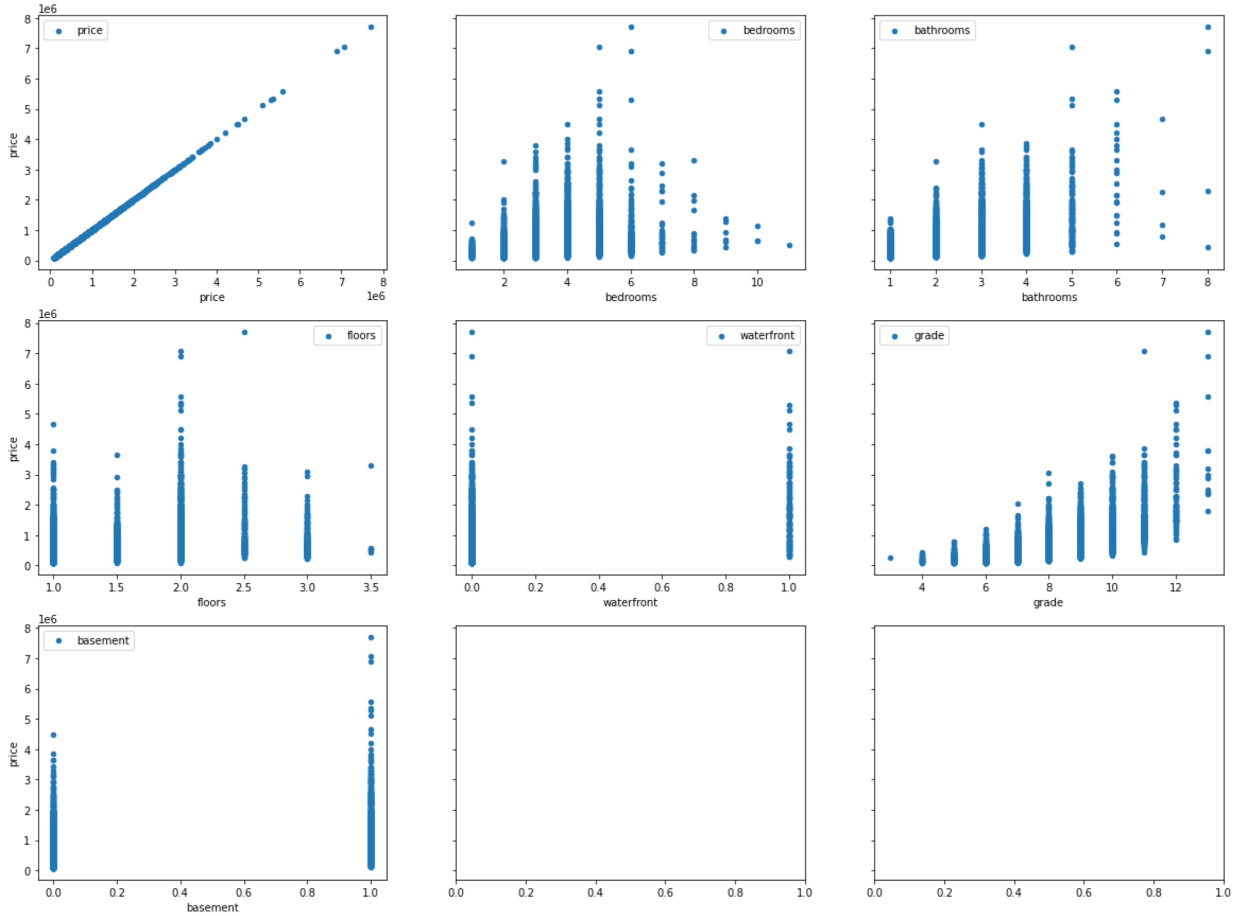
Out[104]:
```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'floors', 'waterf
ront',
       'grade', 'sqft_above', 'sqft_living15', 'basement'],
      dtype='object')
```

Dealing with categorical data

In [105]:
```
#Check out histograms and Scatter plots
categoricals = ['bedrooms', 'bathrooms', 'floors', 'waterfront', 'grade', '
histogram_view(data[categoricals])
```



In [106]:
```
#Add price back in to cat_data to
categoricals_w_price =['price', 'bedrooms', 'bathrooms', 'floors', 'waterfr
scatter_view(data[categoricals_w_price])
```



Looks like most of our conditionals have some sort of solid relationship with our price.

##ONE HOT ENCODING

In [107]:
```
#Check data types to get ready to O-H-E
#We had an issue with these not being in category when running this before
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21593 entries, 0 to 21596
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          21593 non-null  float64
 1   bedrooms       21593 non-null  int64  
 2   bathrooms      21593 non-null  float64
 3   sqft_living    21593 non-null  int64  
 4   floors         21593 non-null  float64
 5   waterfront     21593 non-null  float64
 6   grade          21593 non-null  int64  
 7   sqft_above     21593 non-null  int64  
 8   sqft_living15  21593 non-null  int64  
 9   basement       21593 non-null  int64  
dtypes: float64(4), int64(6)
memory usage: 1.8 MB
```

In [108]:
```
cat_data = data[categoricals].astype('category')
```

In [109]:
```
cat_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21593 entries, 0 to 21596
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   bedrooms    21593 non-null  category
 1   bathrooms   21593 non-null  category
 2   floors      21593 non-null  category
 3   waterfront  21593 non-null  category
 4   grade       21593 non-null  category
 5   basement    21593 non-null  category
dtypes: category(6)
memory usage: 296.8 KB
```

In [110]:
```
#Looks like bedrooms, bathrooms, floors, waterfront, condition, and basemen
#Let's turn them into dummies for the future modeling

dummies = pd.get_dummies(cat_data[categoricals], prefix=categoricals, drop_

data_preprocessed = data.drop(categoricals, axis=1)

data_preprocessed = pd.concat([data_preprocessed, dummies], axis=1)

data_preprocessed.head()
```

Out[110]:

	price	sqft_living	sqft_above	sqft_living15	bedrooms_2	bedrooms_3	bedrooms_4	bedrooms
0	221900.0	1180	1180	1340	0	1	0	
1	538000.0	2570	2170	1690	0	1	0	
2	180000.0	770	770	2720	1	0	0	
3	604000.0	1960	1050	1360	0	0	1	
4	510000.0	1680	1680	1800	0	1	0	

5 rows × 38 columns

In [111]:
```
#One final look at our preprocessed data
data_preprocessed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21593 entries, 0 to 21596
Data columns (total 38 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   price           21593 non-null  float64
 1   sqft_living     21593 non-null  int64  
 2   sqft_above      21593 non-null  int64  
 3   sqft_living15   21593 non-null  int64  
 4   bedrooms_2      21593 non-null  uint8  
 5   bedrooms_3      21593 non-null  uint8  
 6   bedrooms_4      21593 non-null  uint8  
 7   bedrooms_5      21593 non-null  uint8  
 8   bedrooms_6      21593 non-null  uint8  
 9   bedrooms_7      21593 non-null  uint8  
 10  bedrooms_8      21593 non-null  uint8  
 11  bedrooms_9      21593 non-null  uint8  
 12  bedrooms_10     21593 non-null  uint8  
 13  bedrooms_11     21593 non-null  uint8  
 14  bathrooms_2.0   21593 non-null  uint8  
 15  bathrooms_3.0   21593 non-null  uint8  
 16  bathrooms_4.0   21593 non-null  uint8  
 17  bathrooms_5.0   21593 non-null  uint8  
 18  bathrooms_6.0   21593 non-null  uint8  
 19  bathrooms_7.0   21593 non-null  uint8  
 20  bathrooms_8.0   21593 non-null  uint8  
 21  floors_1.5      21593 non-null  uint8  
 22  floors_2.0      21593 non-null  uint8  
 23  floors_2.5      21593 non-null  uint8  
 24  floors_3.0      21593 non-null  uint8  
 25  floors_3.5      21593 non-null  uint8  
 26  waterfront_1.0  21593 non-null  uint8  
 27  grade_4         21593 non-null  uint8  
 28  grade_5         21593 non-null  uint8  
 29  grade_6         21593 non-null  uint8  
 30  grade_7         21593 non-null  uint8  
 31  grade_8         21593 non-null  uint8  
 32  grade_9         21593 non-null  uint8  
 33  grade_10        21593 non-null  uint8  
 34  grade_11        21593 non-null  uint8  
 35  grade_12        21593 non-null  uint8  
 36  grade_13        21593 non-null  uint8  
 37  basement_1      21593 non-null  uint8  
dtypes: float64(1), int64(3), uint8(34)
memory usage: 1.5 MB
```

Begin modeling

We're going to start modeling with just one simple model that has had no scaling or transformations done on it. We're also going to make a function that returns all

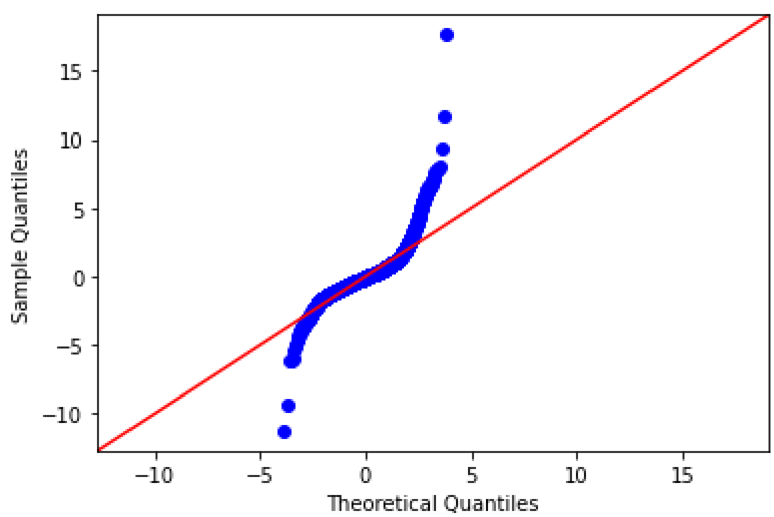
In [112]:
```
def linreg_process(X_data, y_data):
    """ This is a function that returns all relevant data from a modeling p
    All you need is to know what you x and y variables in a dataset are and
    them into test and train sets, create an OLS model and print the summar
    tests and print out those values"""
    linreg = LinearRegression()
    X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, tes
    predictors = sm.add_constant(X_train)
    model = sm.OLS(y_train, predictors).fit()
    print(model.summary())
    linreg.fit(X_train, y_train)
    y_pred_train = linreg.predict(X_train)
    y_pred_test = linreg.predict(X_test)
    print('--------------- Scoring Stats ---------------')
    print(f"R2 Train Score: {r2_score(y_train, y_pred_train)}")
    print(f"R2 Test Score: {r2_score(y_test, y_pred_test)}")
    print(f"Intercept: {linreg.intercept_}")
    print(f"Coefficient: {linreg.coef_}")
    print(f"MSE Train: {mean_squared_error(y_train, y_pred_train)}")
    print(f"MSE Test: {mean_squared_error(y_test, y_pred_test)}")
    residuals = model.resid
    print('--------------- Q-Q Plot ---------------')
    fig = sm.graphics.qqplot(residuals, dist=stats.norm, line ='45',fit=Tru
    fig.show()
```

In [113]:
```
X = data_preprocessed.drop('price', axis=1)
y = data_preprocessed['price']
linreg_process(X, y)
```

```
                            OLS Regression Results
==============================================================================
=====
Dep. Variable:                  price   R-squared:
0.647
Model:                            OLS   Adj. R-squared:
0.646
Method:                 Least Squares   F-statistic:
823.5
Date:                Wed, 03 Feb 2021   Prob (F-statistic):
0.00
Time:                        20:27:22   Log-Likelihood:            -2.220
3e+05
No. Observations:               16194   AIC:                         4.44
1e+05
Df Residuals:                   16157   BIC:                         4.44
4e+05
Df Model:                          36
Covariance Type:            nonrobust
==============================================================================
=========
                     coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
---------
const           5.027e+05   2.06e+04     24.406      0.000    4.62e+05
5.43e+05
sqft_living      164.1925      7.616     21.558      0.000     149.264
179.121
sqft_above       -38.4451      8.421     -4.565      0.000     -54.952
-21.938
sqft_living15     45.8615      4.259     10.768      0.000      37.513
54.210
bedrooms_2      7681.1604   1.86e+04      0.414      0.679   -2.87e+04
4.41e+04
bedrooms_3      -4.74e+04   1.85e+04     -2.557      0.011   -8.37e+04
-1.11e+04
bedrooms_4     -6.733e+04   1.89e+04     -3.554      0.000   -1.04e+05
-3.02e+04
bedrooms_5     -5.366e+04      2e+04     -2.677      0.007    -9.3e+04
-1.44e+04
bedrooms_6      -6.44e+04   2.43e+04     -2.648      0.008   -1.12e+05
-1.67e+04
bedrooms_7     -1.495e+05   4.41e+04     -3.391      0.001   -2.36e+05
-6.31e+04
bedrooms_8      1.411e+05   7.26e+04      1.944      0.052   -1201.447
2.83e+05
bedrooms_9     -4.852e+05   1.15e+05     -4.211      0.000   -7.11e+05
-2.59e+05
bedrooms_10    -8.833e+04   1.56e+05     -0.566      0.571   -3.94e+05
2.17e+05
bedrooms_11    -5.731e+04   2.19e+05     -0.262      0.794   -4.87e+05
3.72e+05
```

```
bathrooms_2.0  -3.078e+04   5947.522     -5.175      0.000   -4.24e+04
-1.91e+04
bathrooms_3.0  -7.117e+04   7542.674     -9.436      0.000    -8.6e+04
-5.64e+04
bathrooms_4.0  -8932.8997   1.19e+04     -0.752      0.452   -3.22e+04
1.43e+04
bathrooms_5.0   1.898e+05   2.43e+04      7.811      0.000    1.42e+05
2.37e+05
bathrooms_6.0   7.008e+05   6.03e+04     11.612      0.000    5.82e+05
8.19e+05
bathrooms_7.0  -9.916e+04   1.29e+05     -0.767      0.443   -3.52e+05
1.54e+05
bathrooms_8.0   1.544e+06   1.25e+05     12.391      0.000     1.3e+06
1.79e+06
floors_1.5      1.083e+05   6362.073     17.023      0.000    9.58e+04
1.21e+05
floors_2.0     -7263.3548   5444.174     -1.334      0.182   -1.79e+04
3407.830
floors_2.5      1.898e+05   2.03e+04      9.348      0.000     1.5e+05
2.3e+05
floors_3.0      7.993e+04   1.13e+04      7.054      0.000    5.77e+04
1.02e+05
floors_3.5      2.173e+05   8.34e+04      2.606      0.009    5.38e+04
3.81e+05
waterfront_1.0  7.259e+05   2.19e+04     33.177      0.000    6.83e+05
7.69e+05
grade_4        -4.297e+05   4.48e+04     -9.587      0.000   -5.18e+05
-3.42e+05
grade_5        -4.571e+05   1.81e+04    -25.298      0.000   -4.93e+05
-4.22e+05
grade_6        -4.152e+05   1.21e+04    -34.417      0.000   -4.39e+05
-3.92e+05
grade_7        -3.564e+05   1.08e+04    -32.899      0.000   -3.78e+05
-3.35e+05
grade_8        -2.763e+05   1.06e+04    -25.967      0.000   -2.97e+05
-2.55e+05
grade_9        -1.354e+05    1.1e+04    -12.274      0.000   -1.57e+05
-1.14e+05
grade_10        3.657e+04   1.23e+04      2.964      0.003    1.24e+04
6.08e+04
grade_11         2.83e+05   1.66e+04     17.025      0.000     2.5e+05
3.16e+05
grade_12        6.223e+05   2.78e+04     22.401      0.000    5.68e+05
6.77e+05
grade_13        1.631e+06   7.48e+04     21.800      0.000    1.48e+06
1.78e+06
basement_1      4.404e+04   6345.616      6.940      0.000    3.16e+04
5.65e+04
======================================================================
=====
Omnibus:                     7897.630   Durbin-Watson:
1.992
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           20886
2.528
Skew:                           1.789   Prob(JB):
0.00
Kurtosis:                      20.226   Cond. No.                     6.5
```

```
9e+15
==============================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The smallest eigenvalue is 4.9e-21. This might indicate that there ar
e
strong multicollinearity problems or that the design matrix is singular.
--------------- Scoring Stats ---------------
R2 Train Score: 0.6472383143454867
R2 Test Score: 0.6483069411223183
Intercept: 552948.0719841787
Coefficient: [ 1.64192500e+02 -3.84451339e+01  4.58614630e+01  7.68116040
e+03
 -4.74003750e+04 -6.73257548e+04 -5.36625632e+04 -6.44042498e+04
 -1.49450619e+05  1.41085739e+05 -4.85187934e+05 -8.83330269e+04
 -5.73074216e+04 -3.07789575e+04 -7.11689597e+04 -8.93289970e+03
  1.89752547e+05  7.00755831e+05 -9.91557343e+04  1.54420159e+06
  1.08303477e+05 -7.26335475e+03  1.89827523e+05  7.99276845e+04
  2.17292837e+05  7.25886994e+05 -4.80007439e+05 -5.07366034e+05
 -4.65455409e+05 -4.06682187e+05 -3.26603941e+05 -1.85626860e+05
 -1.36942190e+04  2.32692519e+05  5.72009269e+05  1.58073430e+06
  4.40413935e+04]
MSE Train: 47490846003.166756
MSE Test: 47737041399.685776
--------------- Q-Q Plot ---------------


<ipython-input-112-dbfb471e5a3b>:24: UserWarning: Matplotlib is currently
using module://ipykernel.pylab.backend_inline, which is a non-GUI backen
d, so cannot show the figure.
  fig.show()
```



Some takeaways from the model

- R2 Score:

  - The training got about a .647 meaning that the model explains roughly 65% of the variability of the response data around the mean
  - This is not a great score, but certaily something we can work with
  - When comparing the train and test R2 scores it appears that the test R2 is slightly higher at .648 this seems to indicate some very mild overfitting, but this could also just be randomness in the samples (cross validation should show if this is the case or not)
- P-values:
  - Most of our data is below of the alpha level 0.05, there are quite a few outliers (mostly within some of our one-hot-encoded variables that are certainly skewing the data somewhat
- Mean Squared Error (MSE):
  - This is an extremely high score, this indicates that we have a lot of errors in our modeling and this thing is absolutely no where close to perfect
  - Scaling and transformations should help
- JB Test
  - The JB test indicates that our data is anything but normal (backed up by the QQ plot)
- F stat
  - Indicates that our data is not homoscedastic
- Overall, our data needs a lot of work to get anywhere close to a respectable model

Some takeaways from the Q-Q Plot

- We do not have a normal distribution
- looks like a whole lot of our data is heavy on the left, but performing some scaling should help a lot with fixing this.

Quick Cross-Validation

In [114]:
```
#This will give us the mean Mean Squared Error to check our model
def cross_val(X_data, y_data):
    linreg = LinearRegression()
    X_train_cv, X_test_cv, y_train_cv, y_test_cv = train_test_split(X_data,
    predictors = sm.add_constant(X_train_cv)
    linreg.fit(X_train_cv, y_train_cv)
    mse = make_scorer(mean_squared_error)
    cv_5_results = cross_val_score(linreg, X_data, y_data, cv=5, scoring=ms
    print(cv_5_results.mean())
```

In [115]:
```
cross_val(X, y)
```

```
48807152240.95346
```

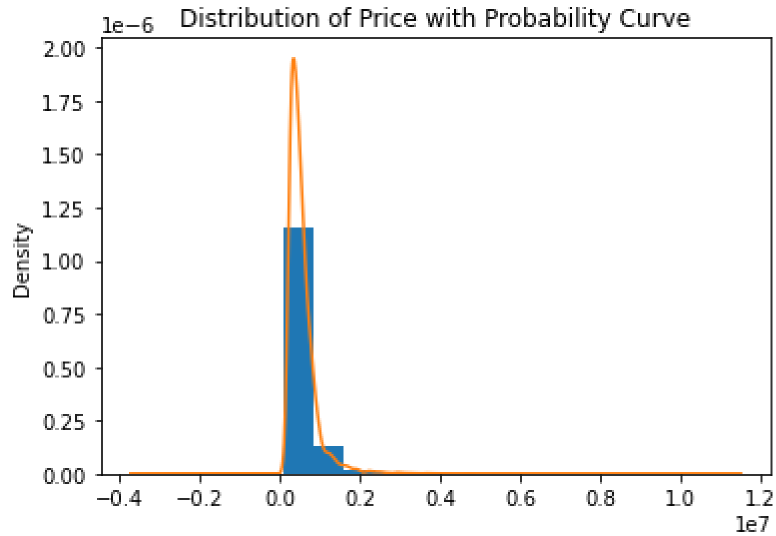
Not all that far off from our data, we have a lot of work to get this running better

Scaling

Let's start the scaling process by taking a look at our most important variable, the target variable price!

In [116]:
```
#Plot histogram with probability density to check distribution of y axis
plt.hist(data_preprocessed['price'], density = True)
data_preprocessed['price'].plot.kde()
plt.title("Distribution of Price with Probability Curve")
```

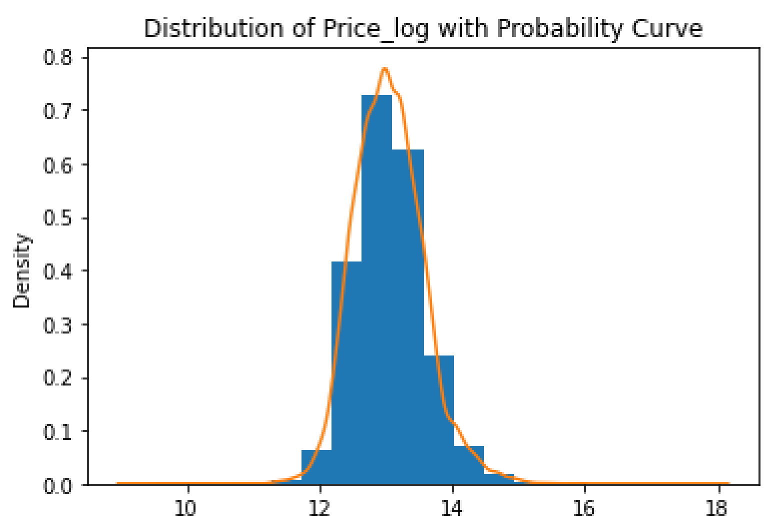
Out[116]: Text(0.5, 1.0, 'Distribution of Price with Probability Curve')



This shows that our data is heavily skewed, and one of the easiest ways to "normalize" this is to perform a log-transformation.

In [117]:
```
#Now lets start our scaling with normalizing this with a log-transform
data_preprocessed['price_log'] = np.log(data['price'])
plt.hist(data_preprocessed['price_log'], density = True)
data_preprocessed['price_log'].plot.kde()
plt.title("Distribution of Price_log with Probability Curve")
```

Out[117]: Text(0.5, 1.0, 'Distribution of Price_log with Probability Curve')



Woo! look at that normal data. now let's get to the independant variables

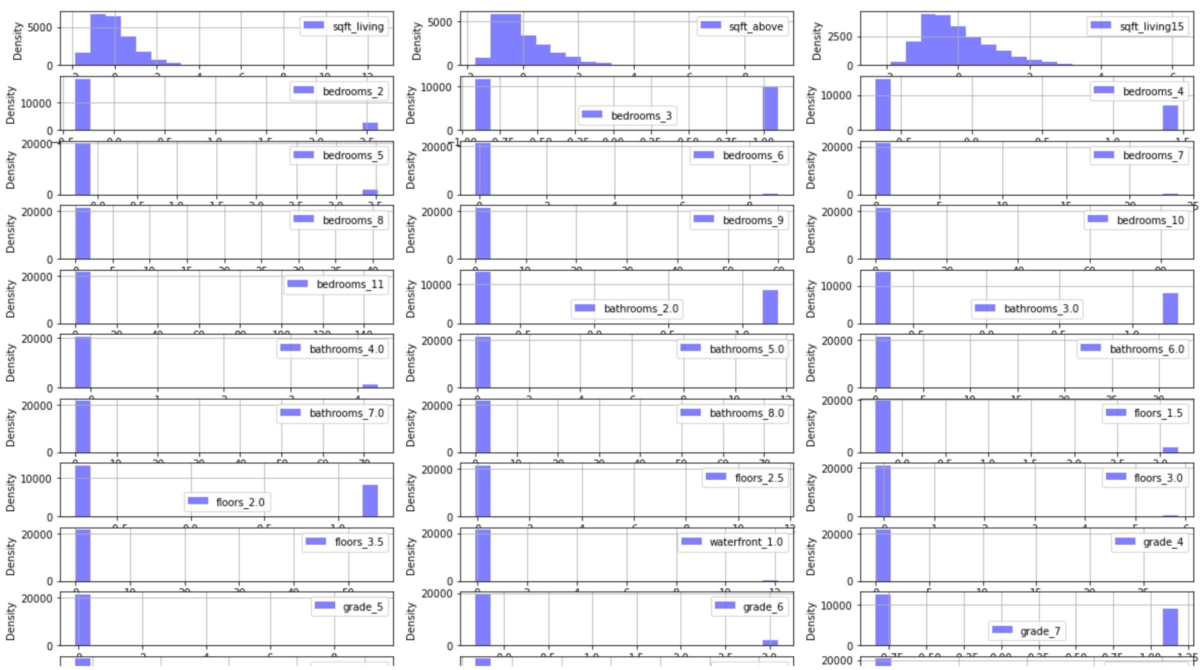
Standard Scaler

We're going to start with using a standard scaler on our data. This can be sensitive to outliers, but as we have already mitigated those I am not too concerned about that, what it will do is bring all of our data to a mean of 0 with a standard deviation of 1. This centers the data and brings it to a Gaussian standard normal deviation, which helps with any modeling.

In [118]:
```
#Standard scaler

scaler = StandardScaler()
dataX2 = data_preprocessed.drop(['price_log', 'price'], axis=1)
y2 = data_preprocessed['price_log']
X_scaled = scaler.fit_transform(dataX2)
scaled_df = pd.DataFrame(X_scaled, columns = dataX2.columns) #For later usa
```

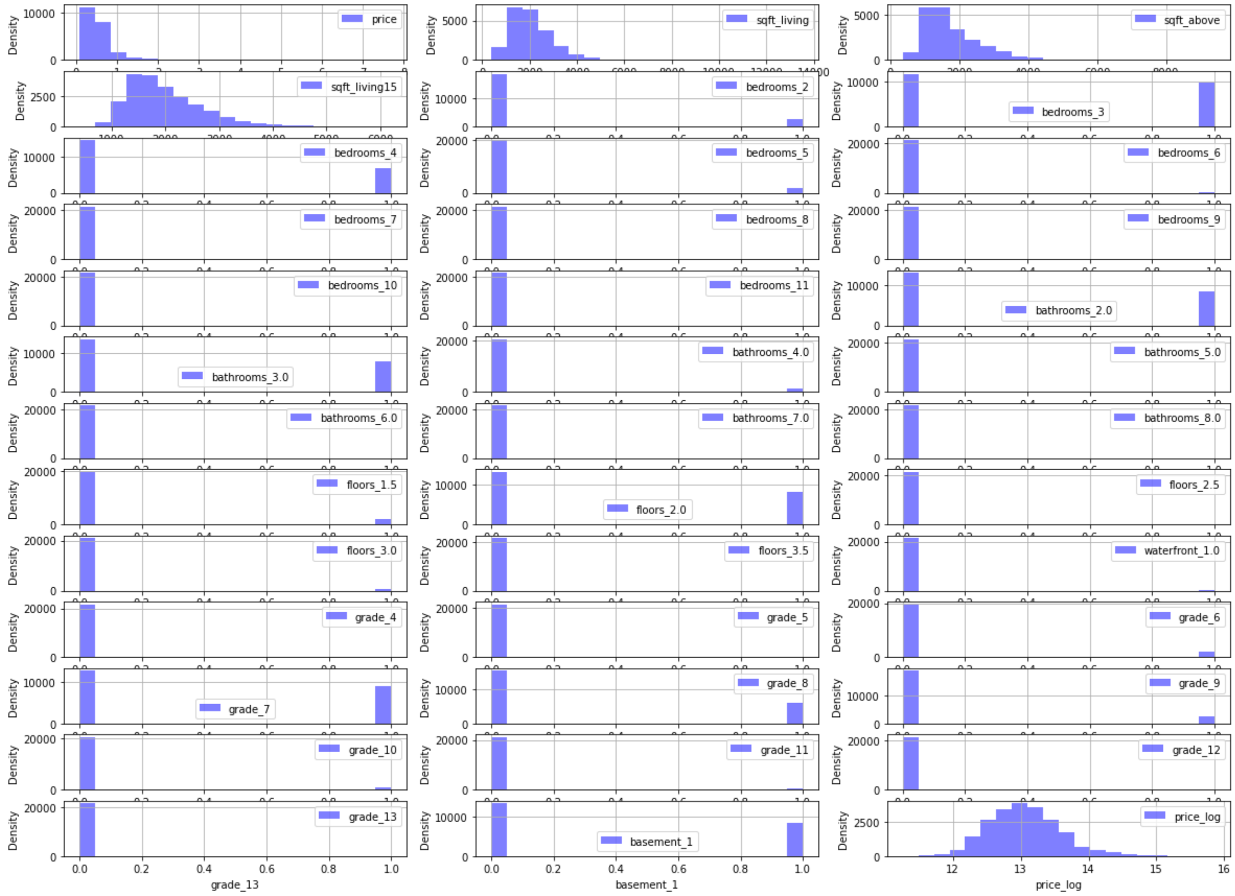
In [119]:
```
cv_scaled_df = pd.concat([scaled_df, data_preprocessed['price_log']], axis=
```

In [120]:
```
histogram_view(scaled_df)
```



Compared with the original

In [121]:
```
histogram_view(data_preprocessed)
```



All in all the data is now centered around 0 and looks to be slightly more normal, hopefully we can see the results for real on our model.

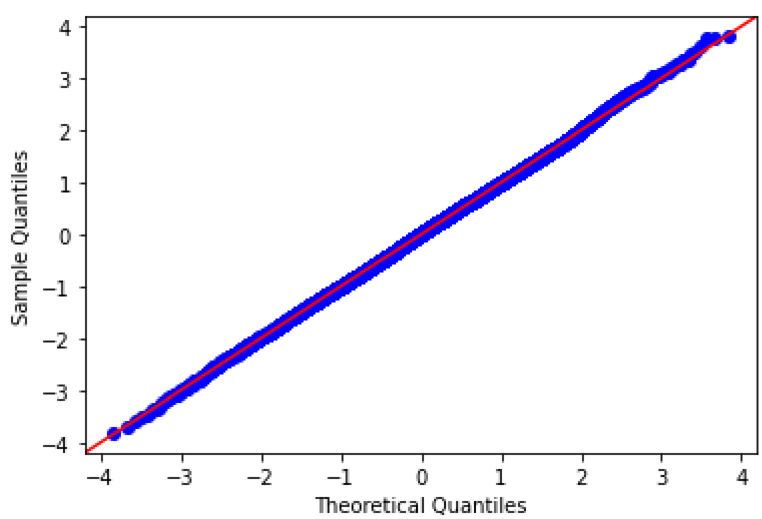
Model #2 log-transformed y and scaled X

In [122]:
```
linreg_process(X_scaled, y2)
```

```
                            OLS Regression Results
==========================================================================
=====
Dep. Variable:              price_log   R-squared:
0.605
Model:                            OLS   Adj. R-squared:
0.605
Method:                 Least Squares   F-statistic:
688.7
Date:                Wed, 03 Feb 2021   Prob (F-statistic):
0.00
Time:                        20:27:31   Log-Likelihood:                 -5
032.3
No. Observations:               16194   AIC:                          1.01
4e+04
Df Residuals:                   16157   BIC:                          1.04
2e+04
Df Model:                          36
Covariance Type:            nonrobust
==========================================================================
=====
                 coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------
-----
const         13.0458      0.003   5020.401      0.000      13.041      1
3.051
x1             0.1605      0.011     15.161      0.000       0.140
0.181
x2            -0.0060      0.011     -0.573      0.567      -0.027
0.015
x3             0.0787      0.004     17.790      0.000       0.070
0.087
x4             0.0112      0.009      1.198      0.231      -0.007
0.030
x5            -0.0401      0.014     -2.868      0.004      -0.068      -
0.013
x6            -0.0428      0.013     -3.201      0.001      -0.069      -
0.017
x7            -0.0203      0.008     -2.548      0.011      -0.036      -
0.005
x8            -0.0086      0.004     -2.092      0.036      -0.017      -
0.001
x9            -0.0030      0.003     -1.073      0.283      -0.008
0.002
x10            0.0022      0.003      0.811      0.417      -0.003
0.007
x11            0.0025      0.003      0.868      0.385      -0.003
0.008
x12            0.0011      0.003      0.389      0.697      -0.004
0.007
x13            0.0004      0.002      0.166      0.868      -0.004
0.005
x14           -0.0016      0.004     -0.353      0.724      -0.010
0.007
```

```
x15          -0.0238      0.006     -4.332      0.000      -0.035      -
0.013
x16           0.0025      0.004      0.643      0.520      -0.005
0.010
x17           0.0044      0.003      1.397      0.162      -0.002
0.010
x18           0.0024      0.003      0.852      0.394      -0.003
0.008
x19          -0.0083      0.003     -3.126      0.002      -0.014      -
0.003
x20          -0.0060      0.003     -2.343      0.019      -0.011      -
0.001
x21           0.0654      0.003     23.880      0.000       0.060
0.071
x22           0.0018      0.004      0.461      0.645      -0.006
0.010
x23           0.0162      0.003      6.133      0.000       0.011
0.021
x24           0.0281      0.003      9.855      0.000       0.022
0.034
x25           0.0040      0.002      1.743      0.081      -0.000
0.008
x26           0.0456      0.003     16.795      0.000       0.040
0.051
x27          -0.0196      0.003     -7.552      0.000      -0.025      -
0.015
x28          -0.0570      0.003    -21.639      0.000      -0.062      -
0.052
x29          -0.0994      0.003    -34.106      0.000      -0.105      -
0.094
x30          -0.0682      0.002    -30.110      0.000      -0.073      -
0.064
x31           0.0230      0.002     11.322      0.000       0.019
0.027
x32           0.0849      0.003     31.943      0.000       0.080
0.090
x33           0.0901      0.003     30.225      0.000       0.084
0.096
x34           0.0667      0.003     21.001      0.000       0.060
0.073
x35           0.0373      0.003     12.865      0.000       0.032
0.043
x36           0.0191      0.003      6.242      0.000       0.013
0.025
x37           0.0643      0.005     13.739      0.000       0.055
0.073
==========================================================================
=====
Omnibus:                        0.714   Durbin-Watson:
1.998
Prob(Omnibus):                  0.700   Jarque-Bera (JB):
0.687
Skew:                           0.011   Prob(JB):
0.709
Kurtosis:                       3.024   Cond. No.                      2.4
9e+15
==========================================================================
```

```
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The smallest eigenvalue is 1.26e-26. This might indicate that there a
re
strong multicollinearity problems or that the design matrix is singular.
--------------- Scoring Stats ---------------
R2 Train Score: 0.6054593246510145
R2 Test Score: -6.224978467286221e+17
Intercept: -1420043.7427191257
Coefficient: [ 1.60479524e-01 -6.04218143e-03  7.86501653e-02  1.12362217
e-02
 -4.01157409e-02 -4.28253718e-02 -2.02757531e-02 -8.59895068e-03
 -2.99946278e-03  2.18514193e-03  2.51684804e-03  1.08169522e-03
  3.76149136e-04 -1.55096664e-03 -2.38491002e-02  2.54393797e-03
  4.36799491e-03  2.42034242e-03 -8.33044592e-03 -6.02156888e-03
  6.53613765e-02  1.84311712e-03  1.62306068e-02  2.80546470e-02
  3.96455921e-03  4.56223927e-02  1.08360848e+09  3.22791809e+09
  8.96274914e+09  1.51113834e+10  1.37809411e+10  1.00038410e+10
  6.83997384e+09  4.11934268e+09  1.96453758e+09  7.52147660e+08
  6.42681412e-02]
MSE Train: 0.10900365141648688
MSE Test: 1.741502357089267e+17
--------------- Q-Q Plot ---------------


<ipython-input-112-dbfb471e5a3b>:24: UserWarning: Matplotlib is currently
using module://ipykernel.pylab.backend_inline, which is a non-GUI backen
d, so cannot show the figure.
  fig.show()
```



- R2 Score:
  - Pretty much the same as our first model, nothing much to report here
- P-values:
  - Same issue as before, some of our one_hot_encoded variables are certainly skewing the data some here
- Mean Squared Error (MSE):
  - This is where we saw the best effect on our scaling, as our data is much more normal now, there is not a lot of error in our model, it's just not that good at predicting
- JB Test
  - Massive change here, we have a respectible score showing that our data does not have a lot of skew or kurtosis, it is much closer to normal than before the scaling
- F stat
  - Indicates that our data is not homoscedastic, we have errors all over the regression line, but it is better than the previous attempt
- Overall, I think we moved in the right direction, but there still isn't much predictive power with this model

Some takeaways from the Q-Q Plot

- With our scaling it looks like we have achieved the goal of a normal distribution of our residuals!

In [123]:
```
#Quick cross val for reference
cross_val(X_scaled, y2)
```

```
5.073375499028728e+16
```

This is actually considerably higher than in the tests we ran, we have more vairance from the mean than initially expected.

Feature Selection

This should help the model by showing us which features from the independant data have the highest

In [124]:
```
#We are going to remove the worst 7 variables from our preprocessed data, l
#impactful independant variables

lr = LinearRegression()
selector = RFE(lr, n_features_to_select = 30)
selector = selector.fit(X, y.ravel()) # convert y to 1d np array to prevent
selector.support_
selected_columns = X.columns[selector.support_ ]
```

In [125]:
```
selected_columns
```

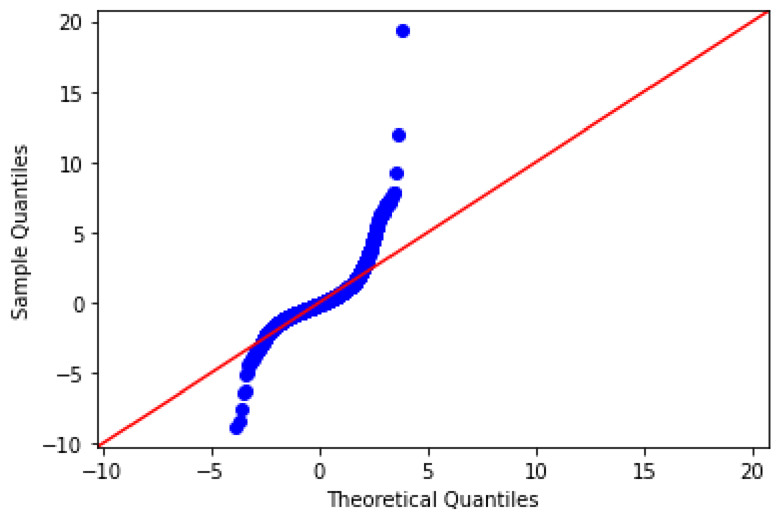
Out[125]:
```
Index(['bedrooms_2', 'bedrooms_4', 'bedrooms_5', 'bedrooms_6', 'bedrooms_
7',
       'bedrooms_8', 'bedrooms_9', 'bedrooms_10', 'bedrooms_11',
       'bathrooms_4.0', 'bathrooms_5.0', 'bathrooms_6.0', 'bathrooms_7.
0',
       'bathrooms_8.0', 'floors_1.5', 'floors_2.5', 'floors_3.0', 'floors
_3.5',
       'waterfront_1.0', 'grade_4', 'grade_5', 'grade_6', 'grade_7', 'gra
de_8',
       'grade_9', 'grade_10', 'grade_11', 'grade_12', 'grade_13',
       'basement_1'],
      dtype='object')
```

In [126]:
```
linreg_process(X[selected_columns], y)
```

```
                            OLS Regression Results                            
==============================================================================
=====
Dep. Variable:                  price   R-squared:                       
0.613
Model:                            OLS   Adj. R-squared:                  
0.612
Method:                 Least Squares   F-statistic:                     
882.8
Date:                Wed, 03 Feb 2021   Prob (F-statistic):              
0.00
Time:                        20:27:32   Log-Likelihood:                -2.227
8e+05
No. Observations:               16194   AIC:                             4.45
6e+05
Df Residuals:                   16164   BIC:                             4.45
9e+05
Df Model:                          29                                         
Covariance Type:            nonrobust                                         
==============================================================================
=========
                     coef    std err          t      P>|t|      [0.025      
0.975]
------------------------------------------------------------------------------
---------
const           8.158e+05   1.03e+04     79.439      0.000    7.96e+05    
8.36e+05
bedrooms_2      1.915e+04   5900.524      3.245      0.001    7583.276    
3.07e+04
bedrooms_4      2.919e+04   4329.057      6.743      0.000    2.07e+04    
3.77e+04
bedrooms_5      8.581e+04   7531.666     11.393      0.000     7.1e+04    
1.01e+05
bedrooms_6      9.728e+04   1.61e+04      6.060      0.000    6.58e+04    
1.29e+05
bedrooms_7      5.224e+04   4.15e+04      1.259      0.208   -2.91e+04    
1.34e+05
bedrooms_8      3.692e+05   7.32e+04      5.044      0.000    2.26e+05    
5.13e+05
bedrooms_9     -4.595e+05   1.19e+05     -3.859      0.000   -6.93e+05   
-2.26e+05
bedrooms_10     1.134e+05   1.62e+05      0.700      0.484   -2.04e+05    
4.31e+05
bedrooms_11      9.41e+04   2.29e+05      0.412      0.680   -3.54e+05    
5.42e+05
bathrooms_4.0    1.07e+05   9100.807     11.757      0.000    8.92e+04    
1.25e+05
bathrooms_5.0   3.785e+05   2.34e+04     16.156      0.000    3.33e+05    
4.24e+05
bathrooms_6.0   9.847e+05   6.19e+04     15.908      0.000    8.63e+05    
1.11e+06
bathrooms_7.0   3.369e+05   1.34e+05      2.518      0.012    7.46e+04    
5.99e+05
bathrooms_8.0   2.294e+06   1.28e+05     17.903      0.000    2.04e+06    
2.55e+06
```

```
floors_1.5      1.133e+05   6335.797     17.884      0.000    1.01e+05
1.26e+05
floors_2.5      1.936e+05   2.07e+04      9.344      0.000    1.53e+05
2.34e+05
floors_3.0      1.431e+04    1.1e+04      1.299      0.194   -7291.886
3.59e+04
floors_3.5      1.798e+05   8.72e+04      2.063      0.039    8970.070
3.51e+05
waterfront_1.0  7.977e+05   2.28e+04     34.968      0.000    7.53e+05
8.42e+05
grade_4        -6.026e+05   4.63e+04    -13.026      0.000   -6.93e+05
-5.12e+05
grade_5        -6.238e+05   1.81e+04    -34.403      0.000   -6.59e+05
-5.88e+05
grade_6        -5.722e+05   1.15e+04    -49.946      0.000   -5.95e+05
-5.5e+05
grade_7        -4.873e+05   1.03e+04    -47.183      0.000   -5.08e+05
-4.67e+05
grade_8         -3.56e+05   1.04e+04    -34.107      0.000   -3.76e+05
-3.36e+05
grade_9        -1.344e+05    1.1e+04    -12.256      0.000   -1.56e+05
-1.13e+05
grade_10        1.187e+05   1.21e+04      9.788      0.000    9.49e+04
1.42e+05
grade_11          4.7e+05   1.61e+04     29.176      0.000    4.38e+05
5.02e+05
grade_12        9.266e+05   2.76e+04     33.628      0.000    8.73e+05
9.81e+05
grade_13        2.077e+06   7.73e+04     26.881      0.000    1.93e+06
2.23e+06
basement_1      9.744e+04   3801.342     25.634      0.000       9e+04
1.05e+05
===========================================================================
=====
Omnibus:                     8884.197   Durbin-Watson:
1.986
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            27461
4.996
Skew:                           2.067   Prob(JB):
0.00
Kurtosis:                      22.746   Cond. No.                      9.3
5e+15
===========================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The smallest eigenvalue is  3e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
--------------- Scoring Stats ---------------
R2 Train Score: 0.6098926440472492
R2 Test Score: -1.1552672976079153e+22
Intercept: -2.909670123625102e+18
Coefficient: [ 1.91489567e+04  2.49117721e+04  8.14156176e+04  1.77261151
e+03
  1.90251959e+04  5.26717233e+05 -3.61645471e+05  1.80704109e+05
```

```
  1.29117522e+05  1.23424043e+05  3.69725164e+05  9.34006371e+05
  3.38667524e+05  2.31691087e+06  1.14969826e+05  1.93744193e+05
  2.27014659e+04  5.38048172e+04  7.52325402e+05  2.90967012e+18
  2.90967012e+18  2.90967012e+18  2.90967012e+18  2.90967012e+18
  2.90967012e+18  2.90967012e+18  2.90967012e+18  2.90967012e+18
  2.90967012e+18  9.78885051e+04]
MSE Train: 52518539058.11955
MSE Test: 1.568101542566779e+33
--------------- Q-Q Plot ---------------

<ipython-input-112-dbfb471e5a3b>:24: UserWarning: Matplotlib is currently
using module://ipykernel.pylab.backend_inline, which is a non-GUI backen
d, so cannot show the figure.
  fig.show()
```



Some takeaways from the model

- R2 Stat got worse, we're still not really getting anywhere with the model
- P-values are now all at 0, which is good
- Mean squared error is the worst that it has ever been
- JB test showing that our data could not be further from normal if we tried
- F stat indicates that our data is not homoscedastic, and it's worse somehow
- Overall, we didn't get anywhere, but I have some hope trying this on our scaled data

In [127]:
```
cv_scaled_df.dropna(inplace=True)
```

In [128]:
```
X_scaled_cv = cv_scaled_df.drop('price_log', axis=1)
y_log_cv = cv_scaled_df['price_log']
```

In [129]:
```
#Start up selector that will give us the 5 best features
lr = LinearRegression()
selector = RFE(lr, n_features_to_select = 5)
selector = selector.fit(X_scaled_cv, y_log_cv.ravel()) # convert y to 1d np
selector.support_
selected_columns = scaled_df.columns[selector.support_ ]
```

In [130]:
```
X_scaled_cv.shape
```

Out[130]:
```
(21589, 37)
```

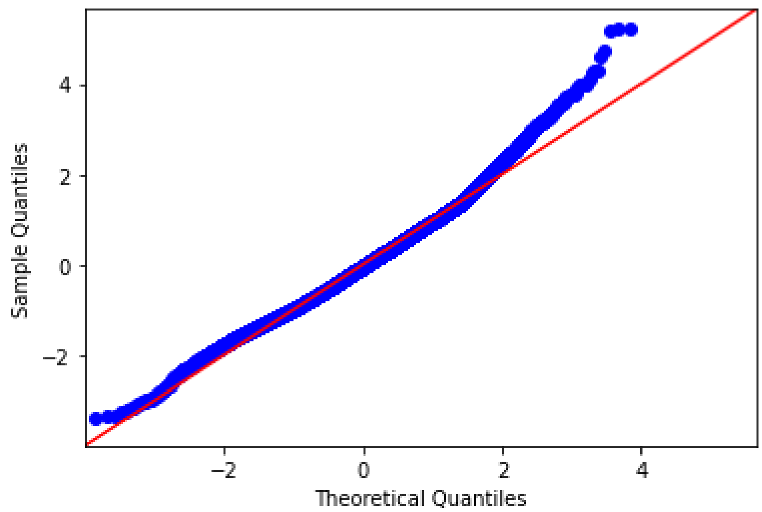
In [131]:
```
y_log_cv.shape
```

Out[131]:
```
(21589,)
```

In [132]:
```
linreg_process(X_scaled_cv[selected_columns], y_log_cv)
```

```
                            OLS Regression Results
==============================================================================
=====
Dep. Variable:              price_log   R-squared:
0.004
Model:                            OLS   Adj. R-squared:
0.004
Method:                 Least Squares   F-statistic:
13.63
Date:                Wed, 03 Feb 2021   Prob (F-statistic):           2.6
7e-13
Time:                        20:27:32   Log-Likelihood:                -1
2598.
No. Observations:               16191   AIC:                         2.52
1e+04
Df Residuals:                   16185   BIC:                         2.52
5e+04
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
=====
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const         13.0424      0.004   3149.224      0.000      13.034      1
3.050
grade_7        0.0197      0.007      2.854      0.004       0.006
0.033
grade_8        0.0349      0.007      5.297      0.000       0.022
0.048
grade_9        0.0324      0.006      5.792      0.000       0.021
0.043
grade_10       0.0288      0.005      5.929      0.000       0.019
0.038
grade_11       0.0165      0.005      3.587      0.000       0.007
0.026
==============================================================================
=====
Omnibus:                      660.939   Durbin-Watson:
2.007
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               84
3.012
Skew:                           0.439   Prob(JB):                    8.76
e-184
Kurtosis:                       3.691   Cond. No.
3.11
==============================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
--------------- Scoring Stats ---------------
R2 Train Score: 0.004192454703602966
```

```
R2 Test Score: 0.0018149917275585015
Intercept: 13.04235344043171
Coefficient: [0.01965861 0.034929   0.03242326 0.02881512 0.01649134]
MSE Train: 0.2775519047375082
MSE Test: 0.27188522372178453
--------------- Q-Q Plot ---------------


<ipython-input-112-dbfb471e5a3b>:24: UserWarning: Matplotlib is currently
using module://ipykernel.pylab.backend_inline, which is a non-GUI backen
d, so cannot show the figure.
  fig.show()
```



Some takeaways from the model

- Obviously something terrible happened here...

Log-Transform X Variables w/ Scaler

This time we are going to log transform everything, then use the StandardScaler(), the biggest issue our data was having was it's lack of normality, so this is a last stand for getting that appropriate.

In [133]:
```
data_preprocessed.head()
```

Out[133]:

	price	sqft_living	sqft_above	sqft_living15	bedrooms_2	bedrooms_3	bedrooms_4	bedrooms
0	221900.0	1180	1180	1340	0	1	0	
1	538000.0	2570	2170	1690	0	1	0	
2	180000.0	770	770	2720	1	0	0	
3	604000.0	1960	1050	1360	0	0	1	
4	510000.0	1680	1680	1800	0	1	0	

5 rows × 39 columns

In [134]:
```
#Start with removing
data_preprocessed.drop(['price_log'], axis=1, inplace=True)
```

In [135]:
```
continuous = ['price', 'sqft_living', 'sqft_above', 'sqft_living15']
data_cont = data_preprocessed[continuous]
log_names = [f'{column}_log' for column in data_cont.columns]
preped_log = np.log(data_preprocessed[continuous])
preped_log.columns = log_names
```

In [136]:
```
data_preprocessed.head()
```

Out[136]:

	price	sqft_living	sqft_above	sqft_living15	bedrooms_2	bedrooms_3	bedrooms_4	bedrooms
0	221900.0	1180	1180	1340	0	1	0	
1	538000.0	2570	2170	1690	0	1	0	
2	180000.0	770	770	2720	1	0	0	
3	604000.0	1960	1050	1360	0	0	1	
4	510000.0	1680	1680	1800	0	1	0	

5 rows × 38 columns

In [137]:
```
data_preprocessed.drop(continuous, axis=1, inplace=True)
```

In [138]:
```
full_preped_df = pd.concat([preped_log, data_preprocessed],axis=1)
full_preped_df.head()
```

Out[138]:

	price_log	sqft_living_log	sqft_above_log	sqft_living15_log	bedrooms_2	bedrooms_3	bedrooms_
0	12.309982	7.073270	7.073270	7.200425	0	1	
1	13.195614	7.851661	7.682482	7.432484	0	1	
2	12.100712	6.646391	6.646391	7.908387	1	0	
3	13.311329	7.580700	6.956545	7.215240	0	0	
4	13.142166	7.426549	7.426549	7.495542	0	1	

5 rows × 38 columns

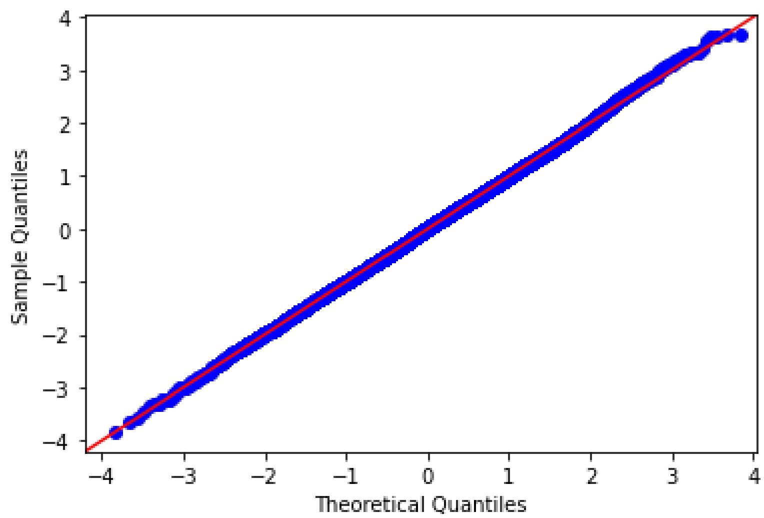
In [139]:
```
scaler = StandardScaler()
prep_X = full_preped_df.drop(['price_log'], axis=1)
y_log = full_preped_df['price_log']
X_scaled = scaler.fit_transform(prep_X)
```

In [140]:
```
linreg_process(prep_X, y_log)
```

```
                            OLS Regression Results
==========================================================================
=====
Dep. Variable:              price_log   R-squared:
0.605
Model:                            OLS   Adj. R-squared:
0.604
Method:                 Least Squares   F-statistic:
687.0
Date:                Wed, 03 Feb 2021   Prob (F-statistic):
0.00
Time:                        20:27:33   Log-Likelihood:                 -5
044.3
No. Observations:               16194   AIC:                          1.01
6e+04
Df Residuals:                   16157   BIC:                          1.04
5e+04
Df Model:                          36
Covariance Type:            nonrobust
==========================================================================
============
                        coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------
------------
const                 8.0974      0.105     77.057      0.000       7.891
8.303
sqft_living_log       0.3651      0.026     13.836      0.000       0.313
0.417
sqft_above_log       -0.0261      0.026     -1.000      0.317      -0.077
0.025
sqft_living15_log     0.2476      0.013     18.982      0.000       0.222
0.273
bedrooms_2            0.0003      0.028      0.011      0.991      -0.055
0.056
bedrooms_3           -0.1396      0.028     -4.910      0.000      -0.195
-0.084
bedrooms_4           -0.1524      0.029     -5.214      0.000      -0.210
-0.095
bedrooms_5           -0.1229      0.031     -3.976      0.000      -0.183
-0.062
bedrooms_6           -0.1149      0.037     -3.076      0.002      -0.188
-0.042
bedrooms_7           -0.0902      0.067     -1.346      0.178      -0.222
0.041
bedrooms_8            0.0935      0.110      0.849      0.396      -0.122
0.309
bedrooms_9           -0.0392      0.175     -0.224      0.823      -0.382
0.304
bedrooms_10           0.0582      0.236      0.246      0.806      -0.405
0.522
bedrooms_11           0.0315      0.332      0.095      0.925      -0.620
0.683
bathrooms_2.0        -0.0357      0.009     -3.844      0.000      -0.054
-0.017
```

```
bathrooms_3.0       -0.0825      0.012     -6.991      0.000     -0.106
-0.059
bathrooms_4.0        0.0172      0.018      0.953      0.340     -0.018
0.052
bathrooms_5.0        0.1102      0.036      3.027      0.002      0.039
0.182
bathrooms_6.0        0.2278      0.091      2.510      0.012      0.050
0.406
bathrooms_7.0       -0.3155      0.194     -1.623      0.105     -0.696
0.065
bathrooms_8.0        0.1618      0.186      0.868      0.385     -0.203
0.527
floors_1.5           0.2206      0.010     22.556      0.000      0.201
0.240
floors_2.0           0.0026      0.008      0.312      0.755     -0.014
0.019
floors_2.5           0.1948      0.031      6.328      0.000      0.134
0.255
floors_3.0           0.1799      0.017     10.420      0.000      0.146
0.214
floors_3.5           0.2593      0.126      2.051      0.040      0.011
0.507
waterfront_1.0       0.5727      0.033     17.299      0.000      0.508
0.638
grade_4              0.1543      0.068      2.277      0.023      0.021
0.287
grade_5              0.1213      0.027      4.533      0.000      0.069
0.174
grade_6              0.2985      0.018     16.890      0.000      0.264
0.333
grade_7              0.4730      0.017     28.356      0.000      0.440
0.506
grade_8              0.6567      0.018     36.660      0.000      0.622
0.692
grade_9              0.8886      0.020     44.152      0.000      0.849
0.928
grade_10             1.0727      0.023     47.069      0.000      1.028
1.117
grade_11             1.2422      0.029     43.324      0.000      1.186
1.298
grade_12             1.4237      0.044     32.339      0.000      1.337
1.510
grade_13             1.7665      0.114     15.558      0.000      1.544
1.989
basement_1           0.1223      0.011     11.433      0.000      0.101
0.143
==========================================================================
=====
Omnibus:                        1.364   Durbin-Watson:
1.997
Prob(Omnibus):                  0.505   Jarque-Bera (JB):
1.336
Skew:                           0.016   Prob(JB):
0.513
Kurtosis:                       3.030   Cond. No.                      5.6
0e+15
==========================================================================
```

```
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The smallest eigenvalue is 8.83e-26. This might indicate that there a
re
strong multicollinearity problems or that the design matrix is singular.
--------------- Scoring Stats ---------------
R2 Train Score: 0.6048743096944382
R2 Test Score: 0.6152590685646006
Intercept: 8.907141974483777
Coefficient: [ 3.65056551e-01 -2.60842552e-02  2.47647475e-01  3.12192417
e-04
 -1.39612358e-01 -1.52369775e-01 -1.22905173e-01 -1.14869696e-01
 -9.01963856e-02  9.34857491e-02 -3.92274948e-02  5.82292576e-02
  3.14879045e-02 -3.56516479e-02 -8.24980751e-02  1.71638668e-02
  1.10221467e-01  2.27816468e-01 -3.15515520e-01  1.61807567e-01
  2.20598839e-01  2.56324544e-03  1.94807808e-01  1.79867925e-01
  2.59317701e-01  5.72700532e-01 -6.55427380e-01 -6.88415470e-01
 -5.11249119e-01 -3.36695510e-01 -1.53054441e-01  7.88312043e-02
  2.62915003e-01  4.32412031e-01  6.13962772e-01  9.56720910e-01
  1.22299784e-01]
MSE Train: 0.10916527927993505
MSE Test: 0.107635270143443
--------------- Q-Q Plot ---------------


<ipython-input-112-dbfb471e5a3b>:24: UserWarning: Matplotlib is currently
using module://ipykernel.pylab.backend_inline, which is a non-GUI backen
d, so cannot show the figure.
  fig.show()
```



In [141]:
```
cross_val(prep_X, y_log)
```

```
0.10949017844190456
```

Some takeaways

- R2 Score:
  - Ultimately despite our best efforts at scaling the data, the original model managed to perform the best for the percentage of variance explained by the model, but this effort at right around 61% isn't awful.
  - Test R2 was about 1.5% better than the train R2 indicating some mild over-fitting, but nothing out of the ordinary
- P-values:
  - We had 13 variables above the alpha level, meaning that there was a lot of reason to believe that many of these were not having an effect on our target variable
- Mean Squared Error (MSE):
  - This is where we saw the greatest effect on our model, we did not have a whole lot of errors and overall our regression line was pretty close to the actual values
  - It wasn't 0 (indicating an overfit perfect line), but there wasn't any indication of overfitting when comparing the train and test data
- JB Test:
  - We also performed very well here and showed that we were able to get very normal data out of this set, especially since it started so far off
  - This was also absolutely confirmed in our Q-Q plot
- Overall:
  - This wasn't a terrible model, and essentially can account for roughly 60% of an accurate home price (in layman's terms)
  - We didn't have any significant overfitting, and were able to get very normal data, there is more work that needs to be done to achieve a more accurate model, but we can get into that in the conclusion

Conclusions

I think this is a great starting point to assist with someone in setting a selling point for their home. with this current model I was able to predict roughly 60% accurate home prices, which leaves a lot to be desired for the homeseller, but there is more work that could certainly be done to improve this further. There is a lot that goes into home selling and unfortunately, due to time constraints, we were not able to model with one of the most important sets of variables...location. I think that a lot of our errors in the modeling came as a result of not having location data and you could see that in some of the outliers. A small home in the Downtown Seattle will be far more expensive than a large home in the furthest reaches of King County. In some ways the issue with this data set was that it was too large, and some future work would be breaking it down by zipcode or geographic area (using latitude and longitude).

Further Work

- As I stated in the conclusions section, I think the most important aspect would be breaking down the zipcode and/or longitude and latitude to make smaller models based on more relevant data
- Another thing I would like to do is to break down and seperate some of the larger homes and larger lots

- Many of these are likely on large plots of cheaper land, and really shouldn't be compared with homes in more desireable locations