

King County Housing Data Project

Student name: T.J. Kyner \ Student pace: Full time \ Instructor name: Abhineet Kulkarni \ Cohort: 040521

Business Problem

The ability to accurately appraise a house is of critical importance for a variety of stakeholders. In addition to buyers and sellers, which each have their own interests in finding the fair market price of a house, other entities such as municipalities benefit from such insight as well. Given that property taxes provide a large portion of tax revenues for municipalities, having an accurate prediction model for house prices can play a key role in efficient financial planning and budgeting. The goal of this project is to provide such a prediction model for the benefit of municipalities in King County, Washington.

Imports & Settings

```
In [1]: # Core tools
import numpy as np
import pandas as pd

# Visualization tools
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# Modeling tools
import scipy.stats as stats
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import RobustScaler
```

Setting the default styling theme for seaborn:

```
In [2]: sns.set_theme()
```

The following code forces tables written in Markdown to be aligned to the left of the cell instead of the center which is the default:

```
In [3]: %%html
<style>
  table {
    display: inline-block
  }
</style>
```

Pulling in the data to a DataFrame:

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

Exploratory Data Analysis

Summary Statistics and Information

To start, I'll print the first five rows of the dataframe to get a quick feel for the information available.

```
In [5]: df.head()
```

```
Out[5]:      id      date   price  bedrooms  bathrooms  sqft_living  sqft_lot  floors  waterfront  view ... grade  sqft_above  sqft_basement  yr_built  yr_renovated
0  7129300520  10/13/2014  221900.0        3       1.0      1180     5650      1.0       NaN     0.0 ...     7     1180         0.0      1955      0.0
1  6414100192   12/9/2014  538000.0        3       2.25      2570     7242      2.0       0.0     0.0 ...     7     2170     400.0      1951  1991.0
2  5631500400   2/25/2015  180000.0        2       1.00      770    10000      1.0       0.0     0.0 ...     6     770         0.0      1933      NaN
3  2487200875   12/9/2014  604000.0        4       3.00      1960     5000      1.0       0.0     0.0 ...     7     1050     910.0      1965      0.0
4  1954400510   2/18/2015  510000.0        3       2.00      1680     8080      1.0       0.0     0.0 ...     8     1680         0.0      1987      0.0
```

5 rows × 21 columns

While most of the 21 columns of data available are fairly self-explanatory, some require a bit more explanation. Along with the original dataset, some metadata on the column names was also provided. Brief descriptions of each column are as follows:

| Column | Description |
|---------------|--|
| id | unique identifier for a house |
| date | house was sold |
| price | is prediction target |
| bedrooms | of Bedrooms/House |
| bathrooms | of bathrooms/bedrooms |
| sqft_living | footage of the home |
| sqft_lot | footage of the lot |
| floors | floors (levels) in house |
| waterfront | House which has a view to a waterfront |
| view | Has been viewed |
| 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 |

Expanded definitions for certain columns, such as `condition` and `grade`, can be found within King County's [Residential Glossary of Terms](#).

While most of the provided descriptions seem logical, I do have a concern regarding the `view` column. It seems more logical for this column to be referring to some sort of grading scale for the view available from the house rather than if it has been "viewed" (by whom? for what purpose?). This idea is further supported by "Views" being defined in King County's [Condo Glossary of Terms](#) as follows:

- For each classification will display blank for no view or "Fair", "Average", "Good" or "Excellent" to reflect the quality of view for that unit

Next, I'll take a look at the datatypes for each column by calling the `.info()` method.

In [6]:

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   id          21597 non-null   int64  
 1   date        21597 non-null   object  
 2   price       21597 non-null   float64 
 3   bedrooms    21597 non-null   int64  
 4   bathrooms   21597 non-null   float64 
 5   sqft_living 21597 non-null   int64  
 6   sqft_lot    21597 non-null   int64  
 7   floors      21597 non-null   float64 
 8   waterfront  19221 non-null   float64 
 9   view        21534 non-null   float64 
 10  condition   21597 non-null   int64  
 11  grade       21597 non-null   int64  
 12  sqft_above  21597 non-null   int64  
 13  sqft_basement 21597 non-null   object  
 14  yr_built    21597 non-null   int64  
 15  yr_renovated 17755 non-null   float64 
 16  zipcode     21597 non-null   int64  
 17  lat         21597 non-null   float64 
 18  long        21597 non-null   float64 
 19  sqft_living15 21597 non-null   int64  
 20  sqft_lot15  21597 non-null   int64  
dtypes: float64(8), int64(11), object(2)
memory usage: 3.5+ MB
```

Observations:

- The `date` column is stored as a string instead of a datetime object. The date down to the specific day is likely too granular to be useful. Using just the month and year may provide better results. This needs to be explored further.
- The `sqft_basement` column is stored as a string instead of an integer like all other columns regarding square footage.
- The `yr_renovated` column is stored as a float and should be converted to an integer since it is referring to a year.

- The `view`, `condition`, `grade`, and `zipcode` columns represent categorical data. These will need to be dealt with accordingly.

In addition to the above observations, it appears that some columns were missing data. Viewing this in an easier to digest manner yields the following:

```
In [7]: df.isna().sum()
```

```
Out[7]: id          0
date         0
price        0
bedrooms     0
bathrooms    0
sqft_living  0
sqft_lot     0
floors       0
waterfront   2376
view         63
condition    0
grade        0
sqft_above   0
sqft_basement 0
yr_built     0
yr_renovated 3842
zipcode      0
lat          0
long          0
sqft_living15 0
sqft_lot15   0
dtype: int64
```

```
In [8]: print('Total rows:', df.shape[0])
```

Total rows: 21597

It appears that the `waterfront` and `yr_renovated` columns have a decent amount of values missing with the `view` column missing a negligible amount.

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

```
Out[9]:
```

| | id | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition | grade | sqft_basement |
|--------------|--------------|--------------|-----------------|------------------|--------------------|-----------------|---------------|-------------------|--------------|------------------|--------------|----------------------|
| count | 2.159700e+04 | 2.159700e+04 | 21597.000000 | 21597.000000 | 21597.000000 | 2.159700e+04 | 21597.000000 | 19221.000000 | 21534.000000 | 21597.000000 | 21597.000000 | 21597.000000 |
| mean | 4.580474e+09 | 5.402966e+05 | 3.373200 | 2.115826 | 2080.321850 | 1.509941e+04 | 1.494096 | 0.007596 | 0.233863 | 3.409825 | 7.657915 | 17.0 |
| std | 2.876736e+09 | 3.673681e+05 | 0.926299 | 0.768984 | 918.106125 | 4.141264e+04 | 0.539683 | 0.086825 | 0.765686 | 0.650546 | 1.173200 | 8.0 |
| min | 1.000102e+06 | 7.800000e+04 | 1.000000 | 0.500000 | 370.000000 | 5.200000e+02 | 1.000000 | 0.000000 | 0.000000 | 1.000000 | 3.000000 | 3.0 |
| 25% | 2.123049e+09 | 3.220000e+05 | 3.000000 | 1.750000 | 1430.000000 | 5.040000e+03 | 1.000000 | 0.000000 | 0.000000 | 3.000000 | 7.000000 | 11.0 |
| 50% | 3.904930e+09 | 4.500000e+05 | 3.000000 | 2.250000 | 1910.000000 | 7.618000e+03 | 1.500000 | 0.000000 | 0.000000 | 3.000000 | 7.000000 | 15.0 |
| 75% | 7.308900e+09 | 6.450000e+05 | 4.000000 | 2.500000 | 2550.000000 | 1.068500e+04 | 2.000000 | 0.000000 | 0.000000 | 4.000000 | 8.000000 | 22.0 |
| max | 9.900000e+09 | 7.700000e+06 | 33.000000 | 8.000000 | 13540.000000 | 1.651359e+06 | 3.500000 | 1.000000 | 4.000000 | 5.000000 | 13.000000 | 94.0 |

Observations:

- There appears to be an outlier with regard to `bedrooms` - the 75th percentile value is 4 bedrooms while the max is 33. This row may possibly need removed.
- The `yr_renovated` column appears to have a large number of `0` values which doesn't make sense given the context. This will be further explored with the `.value_counts()` method.

Value Counts

While I already checked the number of missing values in each column, that does not always paint the full picture. As noted above, the `0` values in the `yr_renovated` column are one such instance where the values may not be missing but are nonetheless inaccurate and simply placeholders. Below, I'll loop over the top five values by frequency for each column as a way to check for any additional placeholder values that may not be readily apparent through summary statistics.

```
In [10]: for col in df.columns:
    print('---', col.upper(), '---')
    print(df[col].value_counts(normalize=True).head(), '\n\n')
```

```
--- ID ---
795000620    0.000139
1825069031   0.000093
2019200220   0.000093
7129304540   0.000093
1781500435   0.000093
Name: id, dtype: float64
```

```
--- DATE ---
6/23/2014    0.006575
6/26/2014    0.006066
6/25/2014    0.006066
7/8/2014     0.005880
4/27/2015    0.005834
Name: date, dtype: float64
```

```
--- PRICE ---  
350000.0  0.007964  
450000.0  0.007964  
550000.0  0.007362  
500000.0  0.007038  
425000.0  0.006945  
Name: price, dtype: float64
```

```
--- BEDROOMS ---  
3  0.454878  
4  0.318655  
2  0.127796  
5  0.074131  
6  0.012594  
Name: bedrooms, dtype: float64
```

```
--- BATHROOMS ---  
2.50  0.248970  
1.00  0.178312  
1.75  0.141131  
2.25  0.094782  
2.00  0.089364  
Name: bathrooms, dtype: float64
```

```
--- SQFT_LIVING ---  
1300  0.006390  
1400  0.006251  
1440  0.006158  
1660  0.005973  
1010  0.005973  
Name: sqft_living, dtype: float64
```

```
--- SQFT_LOT ---  
5000  0.016576  
6000  0.013428  
4000  0.011622  
7200  0.010187  
7500  0.005510  
Name: sqft_lot, dtype: float64
```

```
--- FLOORS ---  
1.0  0.494189  
2.0  0.381303  
1.5  0.088438  
3.0  0.028291  
2.5  0.007455  
Name: floors, dtype: float64
```

```
--- WATERFRONT ---  
0.0  0.992404  
1.0  0.007596  
Name: waterfront, dtype: float64
```

```
--- VIEW ---  
0.0  0.901923  
2.0  0.044441  
3.0  0.023591  
1.0  0.015325  
4.0  0.014721  
Name: view, dtype: float64
```

```
--- CONDITION ---  
3  0.649164  
4  0.262861  
5  0.078761  
2  0.007871  
1  0.001343  
Name: condition, dtype: float64
```

```
--- GRADE ---  
7  0.415521  
8  0.280826  
9  0.121082  
6  0.094365  
10 0.052507  
Name: grade, dtype: float64
```

```
--- SQFT_ABOVE ---  
1300  0.009816  
1010  0.009724  
1200  0.009538  
1220  0.008890  
1140  0.008520
```

```
Name: sqft_above, dtype: float64
```

```
--- SQFT_BASEMENT ---
```

```
0.0      0.593879  
?       0.021021  
600.0    0.010048  
500.0    0.009677  
700.0    0.009631
```

```
Name: sqft_basement, dtype: float64
```

```
--- YR_BUILT ---
```

```
2014     0.025883  
2006     0.020975  
2005     0.020836  
2004     0.020049  
2003     0.019447
```

```
Name: yr_built, dtype: float64
```

```
--- YR_RENOVATED ---
```

```
0.0      0.958096  
2014.0   0.004112  
2003.0   0.001746  
2013.0   0.001746  
2007.0   0.001690
```

```
Name: yr_renovated, dtype: float64
```

```
--- ZIPCODE ---
```

```
98103   0.027874  
98038   0.027272  
98115   0.026994  
98052   0.026578  
98117   0.025605
```

```
Name: zipcode, dtype: float64
```

```
--- LAT ---
```

```
47.6624  0.000787  
47.5491  0.000787  
47.5322  0.000787  
47.6846  0.000787  
47.6711  0.000741
```

```
Name: lat, dtype: float64
```

```
--- LONG ---
```

```
-122.290  0.005325  
-122.300  0.005140  
-122.362  0.004815  
-122.291  0.004630  
-122.372  0.004584
```

```
Name: long, dtype: float64
```

```
--- SQFT_LIVING15 ---
```

```
1540     0.009122  
1440     0.009029  
1560     0.008890  
1500     0.008334  
1460     0.007825
```

```
Name: sqft_living15, dtype: float64
```

```
--- SQFT_LOT15 ---
```

```
5000     0.019771  
4000     0.016484  
6000     0.013335  
7200     0.009724  
4800     0.006714
```

```
Name: sqft_lot15, dtype: float64
```

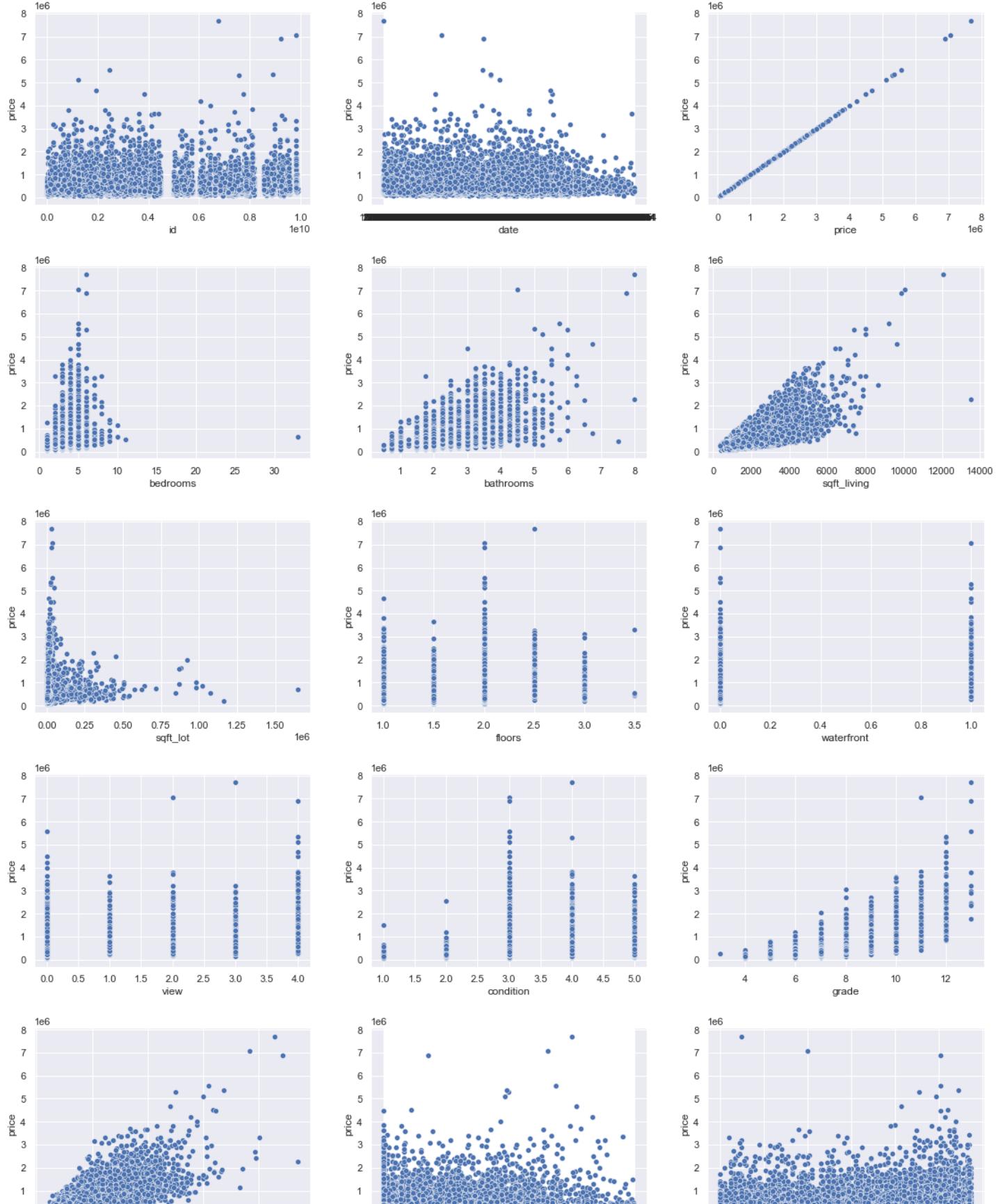
Observations:

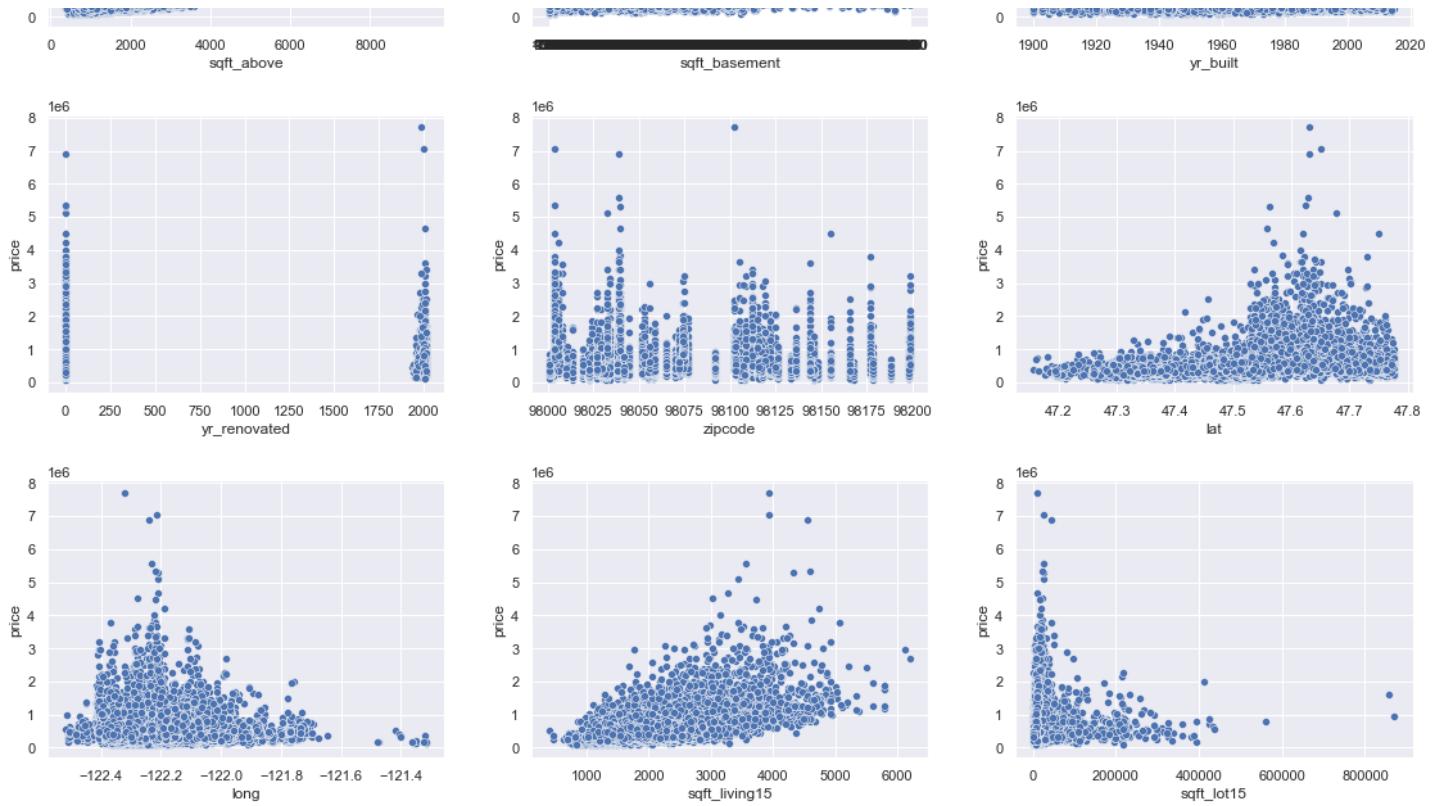
- The `id` column has values that have higher frequencies than 1, indicating that there may either be duplicate rows or the dataset contains entries for each time a house is sold. Regardless, this column does not provide much information and can be dropped.
- Over 99% of houses are not specified as a waterfront property.
- Over 90% of houses do not have a view that warrants a grade.
- Around 2% of houses have `?` as a value for `sqft_basement` which appears to be a placeholder value. These will need to be dealt with but the existence of such a placeholder value bolsters confidence in the nearly 60% of homes with a value of `0.0` for `sqft_basement` meaning that they truly have no basement (versus `0.0` also potentially being a placeholder value).
- Over 95% of houses have a `yr_renovated` value of `0.0`. As mentioned previously, the context indicates that this is meant as a placeholder value. Combined with the fact that this column has the largest amount of missing data, I will likely be dropping it.

Basic Visualizations

Plotting each of the columns against `price` will allow me to quickly see which features are most correlated with the value of a house.

```
In [11]:  
fig, axes = plt.subplots(nrows=7, ncols=3, figsize=(16, 28))  
# Reshape axes to a 1-D object for easier iteration  
axes = axes.reshape(-1)  
  
for i, col in enumerate(df.columns):  
    sns.scatterplot(x=col, y='price', data=df, ax=axes[i])  
  
# Increasing the padding around each subplot for enhanced clarity  
fig.tight_layout(pad=2.0);
```





Observations:

- Unsurprisingly, the size of the home appears to be strongly correlated with `price`.
- Interestingly, houses with the highest `condition` do not seem to fetch higher average prices than those closer to the middle of the scale.
- Both `waterfront` and `view` appear to have no correlation with `price`.

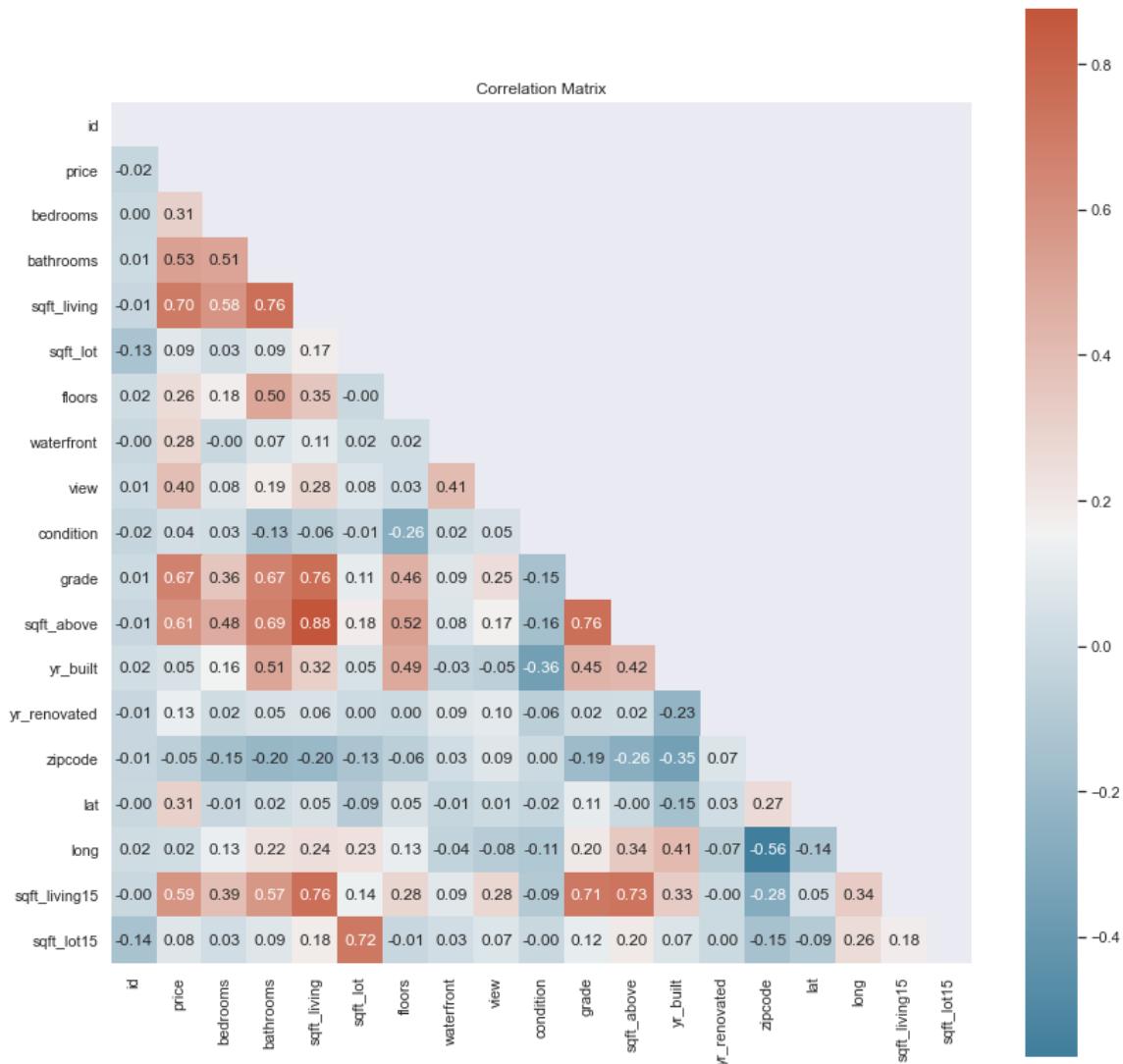
Next, I'll visualize the actual correlation values via a heatmap:

```
In [12]: plt.figure(figsize=(14, 14))
plt.title('Correlation Matrix')

# Creating a mask to block the top right half of the heatmap (redundant information)
mask = np.triu(np.ones_like(df.corr()))

# Custom color map
cmap = sns.diverging_palette(230, 20, as_cmap=True)

sns.heatmap(df.corr(), mask=mask, annot=True, fmt='.2f', square=True, cmap=cmap);
```

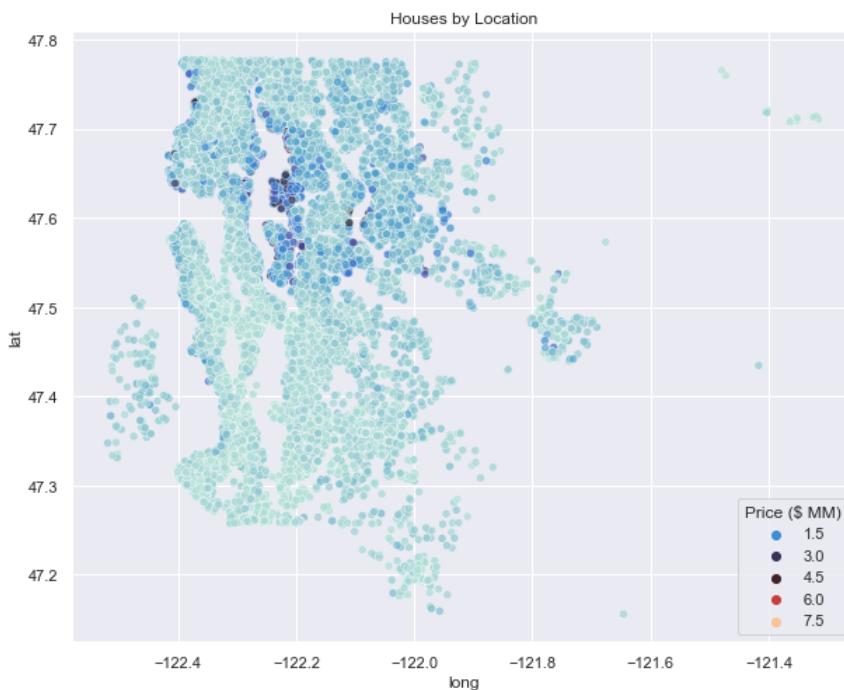


Observations:

- Several columns are highly correlated with one another which will lead to multicollinearity issues in the regression models. This will need to be addressed.

Finally, I'll create a rough map using the latitude and longitude data.

```
In [13]: plt.figure(figsize=(10, 8))
plt.title('Houses by Location')
sns.scatterplot(data=df, x='long', y='lat', hue='price', alpha=0.7, palette='icefire')
plt.legend(title='Price ($ MM)', loc='lower right')
plt.savefig('images/map.png', dpi=150, facecolor='white');
```



Observations:

- Certain areas have a higher density of highly priced homes than others
- This indicates that `lat` and `long` will likely be useful predictors

Data Preprocessing

Based on the observations made in the EDA section, the following list represents the goals for preprocessing the data before moving on to creating the baseline and subsequent prediction models:

1. Drop the `id` column
2. Investigate splitting the `date` column into two columns containing the month and year
3. Convert the `sqft_basement` column to an integer and handle placeholder values
4. Drop the `yr_renovated` column
5. Handle missing values in `waterfront` and `view`
6. Handle multicollinearity between highly correlated columns

1. Drop the `id` column

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

```
Out[14]: Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long',
       'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

2. Investigate splitting the `date` column into two columns containing the month and year

The idea behind this step is to determine whether decreasing the granularity of the `date` column by extracting just the month and year provides better insight for predicting price versus using the full date. To start, I'll convert the `date` column to datetime objects instead of strings.

```
In [15]: df.date = pd.to_datetime(df.date)
df.head(2)
```

```
Out[15]:   date      price  bedrooms  bathrooms  sqft_living  sqft_lot  floors  waterfront  view  condition  grade  sqft_above  sqft_basement  yr_built  yr_renovated  zipcode
0  2014-10-13  221900.0        3       1.0       1180     5650      1.0        NaN      0.0        3        7      1180        0.0      1955        0.0      98178
1  2014-12-09  538000.0        3       2.25       2570     7242      2.0        0.0      0.0        3        7      2170      400.0      1951    1991.0      98125
```

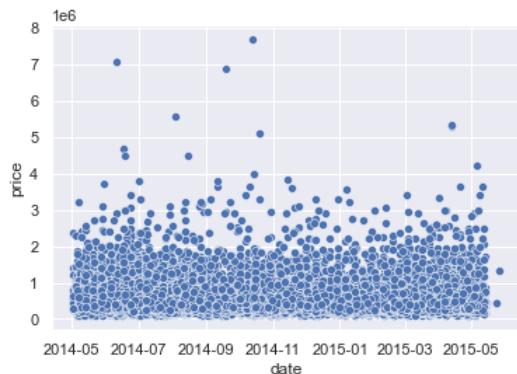
Now that the datatype has been corrected, I want to view the range of dates within the dataset.

```
In [16]: df.date.describe(datetime_is_numeric=True)
```

```
Out[16]: count    21597
mean      2014-10-29 04:20:38.171968512
min       2014-05-02 00:00:00
25%      2014-07-22 00:00:00
50%      2014-10-16 00:00:00
75%      2015-02-17 00:00:00
max       2015-05-27 00:00:00
Name: date, dtype: object
```

The dataset only covers about a year, from May 2014 to May 2015. There likely won't be much of a difference between the years but I'll check it anyways. I'll start with plotting all of the dates versus price to see if there's any trend.

```
In [17]: sns.scatterplot(x='date', y='price', data=df);
```

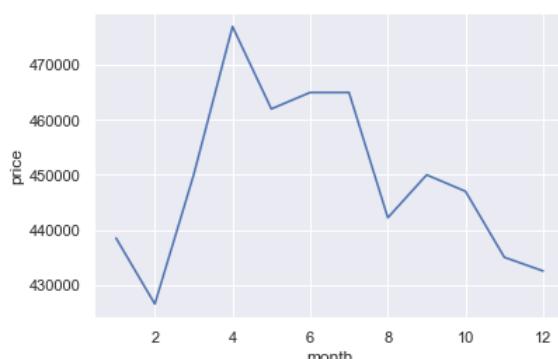


Next, I'll create a column for the month and another for the year. Each of these will be grouped by their respective distinct values and plotted.

```
In [18]: df['month'] = df.date.dt.month
df_month = df.groupby('month').median()
df_month
```

```
Out[18]:   price  bedrooms  bathrooms  sqft_living  sqft_lot  floors  waterfront  view  condition  grade  sqft_above  yr_built  yr_renovated  zipcode  lat  lon
month
1  438500.0      3.0      2.25     1890.0     7800.0      1.0        0.0      0.0      3.0      7.0     1570.0    1974.0        0.0  98065.0  47.55950 -122.23
2  426500.0      3.0      2.00     1830.0     7667.0      1.0        0.0      0.0      3.0      7.0     1489.0    1974.0        0.0  98059.0  47.56120 -122.22
3  450000.0      3.0      2.25     1870.0     7560.0      1.0        0.0      0.0      3.0      7.0     1540.0    1973.0        0.0  98065.0  47.56480 -122.22
4  477000.0      3.0      2.25     1900.0     7500.0      1.5        0.0      0.0      3.0      7.0     1540.0    1976.0        0.0  98072.0  47.56860 -122.22
5  462000.0      3.0      2.25     1930.0     7498.0      1.0        0.0      0.0      3.0      7.0     1540.0    1974.0        0.0  98072.0  47.57870 -122.23
6  465000.0      3.0      2.25     1980.0     7700.0      1.5        0.0      0.0      3.0      8.0     1600.0    1975.0        0.0  98072.0  47.57605 -122.22
7  465000.0      3.0      2.25     1950.0     7695.0      1.5        0.0      0.0      3.0      8.0     1610.0    1977.0        0.0  98059.0  47.57400 -122.21
8  442200.0      3.0      2.25     1940.0     7810.0      1.0        0.0      0.0      3.0      7.0     1590.0    1976.0        0.0  98059.0  47.57720 -122.22
9  450000.0      3.0      2.25     1920.0     7620.0      1.5        0.0      0.0      3.0      7.0     1580.0    1974.0        0.0  98065.0  47.57400 -122.23
10 447000.0      3.0      2.25     1905.0     7413.0      1.5        0.0      0.0      3.0      7.0     1560.0    1974.0        0.0  98065.0  47.57525 -122.24
11 435000.0      3.0      2.00     1870.0     7500.0      1.5        0.0      0.0      3.0      7.0     1560.0    1973.0        0.0  98072.0  47.57380 -122.23
12 432500.0      3.0      2.25     1900.0     7725.0      1.0        0.0      0.0      3.0      7.0     1545.0    1974.0        0.0  98070.0  47.57865 -122.24
```

```
In [19]: sns.lineplot(x='month', y='price', data=df_month);
```



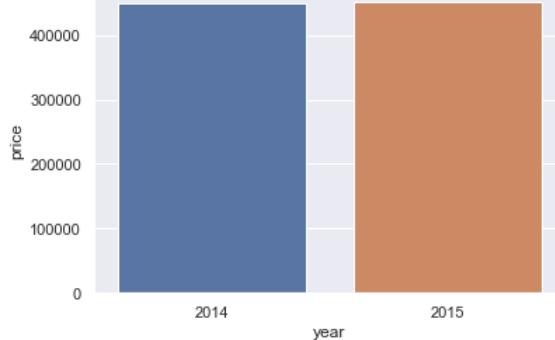
```
In [20]: df['year'] = df.date.dt.year
df_year = df.groupby('year').median()
df_year
```

Out[20]:

| | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition | grade | sqft_above | yr_built | yr_renovated | zipcode | lat | long |
|------|----------|----------|-----------|-------------|----------|--------|------------|------|-----------|-------|------------|----------|--------------|---------|----------|----------|
| year | | | | | | | | | | | | | | | | |
| 2014 | 450000.0 | 3.0 | 2.25 | 1930.0 | 7633.5 | 1.5 | 0.0 | 0.0 | 3.0 | 7.0 | 1580.0 | 1975.0 | 0.0 | 98065.0 | 47.57645 | -122.231 |
| 2015 | 451000.0 | 3.0 | 2.25 | 1880.0 | 7576.0 | 1.0 | 0.0 | 0.0 | 3.0 | 7.0 | 1540.0 | 1974.0 | 0.0 | 98065.0 | 47.56370 | -122.229 |

In [21]:

```
sns.barplot(x='year', y='price', data=df_year.reset_index());
```



As expected, there isn't much of a trend for the individual dates given the relatively short time horizon. Likewise, there's not much of a difference between the values in 2014 and those in 2015. However, there is a clear trend when grouping the data by month with late spring to mid summer seeing the highest median prices. As a result of these findings, the `month` column will be kept in place of the `date` column going forward.

In [22]:

```
df.drop(columns=['date', 'year'], inplace=True)
df.columns
```

Out[22]:

```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long',
       'sqft_living15', 'sqft_lot15', 'month'],
      dtype='object')
```

3. Convert the `sqft_basement` column to an integer and handle placeholder values

First, I need to handle the placeholder value of `?` discovered while analyzing the value counts. These placeholder values represent about 2% of the data in the column. Since the majority of houses in this dataset do not have a basement at all and I want to keep the continuous nature of the data in this column, I will simply replace each of the placeholder values with `0`.

In [23]:

```
df.sqft_basement = df.sqft_basement.replace('?', 0)
df.sqft_basement.value_counts(normalize=True).head()
```

Out[23]:

```
0.0    0.593879
0     0.021021
600.0  0.010048
500.0  0.009677
700.0  0.009631
Name: sqft_basement, dtype: float64
```

With the placeholders values out of the way, I can now convert the rest of the values to integers to align with the other columns that deal with square footage information. Due to the presence of `0.0`, which cannot be directly converted to an `int`, I must first convert the values to a `float` and then an `int`.

In [24]:

```
df.sqft_basement = df.sqft_basement.apply(lambda x: int(float(x)))
df.sqft_basement.dtype
```

Out[24]:

```
dtype('int64')
```

4. Drop the `yr_renovated` column

This column has the most amount of data missing. Of the data that isn't missing, approximately 95% has a placeholder value of `0.0` which likely indicates the house either hasn't been renovated or the renovation year is unknown. An argument could be made for converting the column to indicate whether a renovation has been made at all regardless of the year. However, given the subjective and arbitrary nature of renovations that cannot be captured quantitatively, the usefulness of that information is debateable at best. Factoring in all of this information, I will simply drop the column from the dataframe.

In [25]:

```
df.drop(columns=['yr_renovated'], inplace=True)
df.columns
```

Out[25]:

```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'zipcode', 'lat', 'long', 'sqft_living15',
       'sqft_lot15', 'month'],
      dtype='object')
```

5. Handle missing values in `waterfront` and `view`

Starting with `waterfront`, I'll take a look at if the median price for houses with a waterfront is materially different than those without.

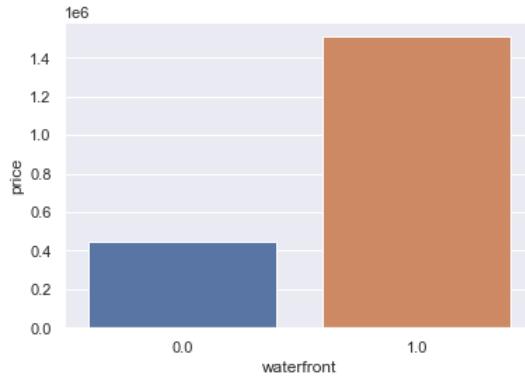
```
In [26]: df.waterfront.value_counts(normalize=True)
```

```
Out[26]: 0.0    0.992404  
1.0    0.007596  
Name: waterfront, dtype: float64
```

```
In [27]: df_waterfront = df.groupby('waterfront').median().reset_index()  
df_waterfront
```

```
Out[27]:   waterfront      price  bedrooms  bathrooms  sqft_living  sqft_lot  floors  view  condition  grade  sqft_above  sqft_basement  yr_built  zipcode      lat      long  
0            0.0  450000.0        3.0       2.25     1910.0    7589.0     1.5     0.0       3.0      7.0      1560.0        0.0  1975.0  98065.0  47.57220 -122.2300  
1            1.0 1510000.0        3.0       2.50     2900.0   17730.5     2.0     4.0       3.0      9.0      2200.0      535.0  1959.5  98075.0  47.54815 -122.2735
```

```
In [28]: sns.barplot(x='waterfront', y='price', data=df_waterfront);
```



Houses with a waterfront have a much higher median value than those without. There are a few different ways to approach handling the missing values in this instance:

- Drop the rows entirely
- Impute the missing values based on the distribution of available values
- Set all missing values equal to the same value

With more than 99% of houses in the dataset not being a waterfront property, I will be setting all of the missing values to `0.0`. Dropping the rows entirely would throw away too much useful information and imputing the values would be better suited for a non-binary variable.

```
In [29]: df.waterfront.isna().sum()
```

```
Out[29]: 2376
```

```
In [30]: df.waterfront.replace(np.nan, 0.0, inplace=True)  
df.waterfront.isna().sum()
```

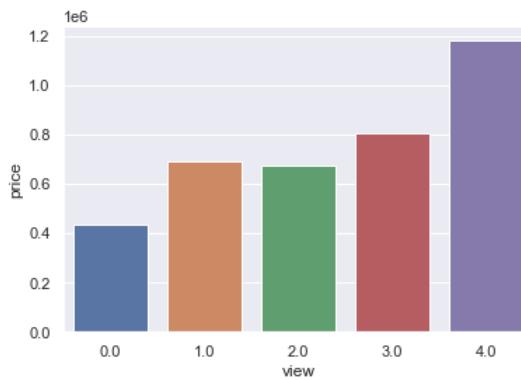
```
Out[30]: 0
```

I'll take a similar approach with handling the missing values in the `view` column.

```
In [31]: df.view.value_counts(normalize=True)
```

```
Out[31]: 0.0    0.901923  
2.0    0.044441  
3.0    0.023591  
1.0    0.015325  
4.0    0.014721  
Name: view, dtype: float64
```

```
In [32]: df_view = df.groupby('view').median().reset_index()  
sns.barplot(x='view', y='price', data=df_view);
```



There's a clear trend where higher view grades are associated with higher median prices. The same options for dealing with the missing values in the `waterfront` column exist for the `view` column as well. In this instance, however, I *will* impute the values since the variable is non-binary and somewhat subjective. Whereas setting a non-waterfront property to a value of having a waterfront is definitively incorrect, there is more leeway with the view categories.

```
In [33]: df.view.isna().sum()
```

```
Out[33]: 63
```

```
In [34]: values = df.view.value_counts().reset_index()['index'].values
probs = df.view.value_counts(normalize=True).values
df.view.replace(np.nan, np.random.choice(a=values, p=probs), inplace=True)
df.view.isna().sum()
```

```
Out[34]: 0
```

Taking a look at all columns for any remaining missing values:

```
In [35]: df.isna().sum()
```

```
Out[35]: price      0
bedrooms    0
bathrooms   0
sqft_living 0
sqft_lot     0
floors      0
waterfront   0
view        0
condition    0
grade        0
sqft_above   0
sqft_basement 0
yr_built     0
zipcode      0
lat          0
long         0
sqft_living15 0
sqft_lot15   0
month        0
dtype: int64
```

6. Handle multicollinearity between highly correlated columns

The first step in handling multicollinearity is determining which columns have the highest absolute correlation with one another.

```
In [36]: df_corr = df.corr().abs().stack().reset_index()
df_corr.head()
```

```
Out[36]:   level_0    level_1      0
0    price      price  1.000000
1    price  bedrooms  0.308787
2    price  bathrooms  0.525906
3    price  sqft_living  0.701917
4    price    sqft_lot  0.089876
```

```
In [37]: df_corr['pairs'] = list(zip(df_corr.level_0, df_corr.level_1))
df_corr.set_index('pairs', inplace=True)
df_corr.head()
```

```
Out[37]:
```

| | level_0 | level_1 | 0 |
|----------------------|---------|-------------|----------|
| pairs | | | |
| (price, price) | price | price | 1.000000 |
| (price, bedrooms) | price | bedrooms | 0.308787 |
| (price, bathrooms) | price | bathrooms | 0.525906 |
| (price, sqft_living) | price | sqft_living | 0.701917 |
| (price, sqft_lot) | price | sqft_lot | 0.089876 |

```
In [38]: df_corr.drop(columns=['level_0', 'level_1'], inplace=True)
df_corr.columns = ['cc']
df_corr.sort_values('cc', ascending=False, inplace=True)
df_corr.drop_duplicates(inplace=True)
df_corr.head()
```

Out[38]:

| | cc |
|------------------------------|----------|
| pairs | |
| (price, price) | 1.000000 |
| (sqft_living, sqft_above) | 0.876448 |
| (grade, sqft_living) | 0.762779 |
| (sqft_living15, sqft_living) | 0.756402 |
| (sqft_above, grade) | 0.756073 |

```
In [39]: df_corr[(df_corr.cc > 0.75) & (df_corr.cc < 1.00)]
```

Out[39]:

| | cc |
|------------------------------|----------|
| pairs | |
| (sqft_living, sqft_above) | 0.876448 |
| (grade, sqft_living) | 0.762779 |
| (sqft_living15, sqft_living) | 0.756402 |
| (sqft_above, grade) | 0.756073 |
| (sqft_living, bathrooms) | 0.755758 |

The `sqft_living` column appears in four of the top five pairs of columns with the highest absolute correlations. As a result, this column will be dropped to help reduce multicollinearity.

```
In [40]: df.drop(columns=['sqft_living'], inplace=True)
df.columns
```

Out[40]:

```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_lot', 'floors', 'waterfront',
       'view', 'condition', 'grade', 'sqft_above', 'sqft_basement', 'yr_built',
       'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15', 'month'],
      dtype='object')
```

Modeling

Baseline Model

```
In [41]: def fit_model(df, target='price'):
    ...
    Description:
    -----
    Takes a dataframe and returns a fitted OLS model with price as the dependent variable.

    Parameters:
    -----
    df : pandas.DataFrame
        This dataframe should include all of the predictors and the target column.

    target: str
        The name of the column being predicted (dependent variable). The default is
        'price'. Useful for transformations where the name is changed for clarity
        (e.g. 'price' --> 'price_log' for a log transformation).

    Example:
    -----
    >>> fit_model(df)
```

```

<statsmodels.regression.linear_model.RegressionResultsWrapper>
...
predictors = df.drop(columns=[target])
predictors = sm.add_constant(predictors)
model = sm.OLS(df[target], predictors).fit()
return model

model = fit_model(df)
model.summary()

```

Out[41]:

| | coef | std err | t | P> t | [0.025 | 0.975] |
|--------------------------|------------------|----------------------------|-------------|----------------------|------------|-----------|
| const | 7.333e+06 | 2.93e+06 | 2.506 | 0.012 | 1.6e+06 | 1.31e+07 |
| bedrooms | -3.562e+04 | 1896.002 | -18.789 | 0.000 | -3.93e+04 | -3.19e+04 |
| bathrooms | 4.606e+04 | 3220.564 | 14.301 | 0.000 | 3.97e+04 | 5.24e+04 |
| sqft_lot | 0.1221 | 0.048 | 2.547 | 0.011 | 0.028 | 0.216 |
| floors | 6703.5287 | 3589.046 | 1.868 | 0.062 | -331.266 | 1.37e+04 |
| waterfront | 6.244e+05 | 1.81e+04 | 34.476 | 0.000 | 5.89e+05 | 6.6e+05 |
| view | 5.343e+04 | 2103.056 | 25.404 | 0.000 | 4.93e+04 | 5.75e+04 |
| condition | 2.545e+04 | 2318.854 | 10.974 | 0.000 | 2.09e+04 | 3e+04 |
| grade | 9.8e+04 | 2157.073 | 45.431 | 0.000 | 9.38e+04 | 1.02e+05 |
| OLS Regression Results | | | | | | |
| Dep. Variable: | price | R-squared: | 0.700 | sqft_above | 178.9092 | 3.659 |
| Model: | OLS | Adj. R-squared: | 0.700 | sqft_basement | 146.7878 | 4.357 |
| Method: | Least Squares | F-statistic: | 2960. | yr_built | -2777.2507 | 68.900 |
| Date: | Mon, 07 Jun 2021 | Prob (F-statistic): | 0.00 | zipcode | -587.4265 | 32.996 |
| Time: | 12:42:26 | Log-Likelihood: | -2.9440e+05 | lat | 6.002e+05 | 1.07e+04 |
| No. Observations: | 21597 | AIC: | 5.888e+05 | long | -2.171e+05 | 1.32e+04 |
| Df Residuals: | 21579 | BIC: | 5.890e+05 | sqft_living15 | 21.1380 | 3.447 |
| Df Model: | 17 | | | sqft_lot15 | -0.3719 | 0.073 |
| Covariance Type: | nonrobust | | | month | -3058.7615 | 440.249 |
| Omnibus: | 18377.672 | Durbin-Watson: | 1.989 | | | |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 1851760.229 | | | |
| Skew: | 3.571 | Prob(JB): | 0.00 | | | |
| Kurtosis: | 47.797 | Cond. No. | 2.15e+08 | | | |

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 2.15e+08. This might indicate that there are strong multicollinearity or other numerical problems.

Observations:

- `bedrooms` has a negative coefficient (surprisingly)
- `floors` is not significant at the alpha = 0.05 level
- `yr_built` has a negative coefficient (surprisingly)
- R-squared value for baseline model is 0.700 - pretty good

I also want to keep track of the RMSE for each model, so I'll create a reusable function to do just that:

In [42]:

```

def print_rmse(df, target='price', decimals=2):
    ...
    Description:
    -----
    Takes a dataframe, splits it into train/test data, fits it to a linear regression model,
    then calculates and prints the RMSE for both the train and test portions rounded to two
    decimal places.

    Parameters:
    -----
    df : pandas.DataFrame
        This dataframe should include all of the predictors and the target column.

    target: str
        The name of the column being predicted (dependent variable). The default is
        'price'. Useful for transformations where the name is changed for clarity
        (e.g. 'price' --> 'price_log' for a log transformation).

```

```

decimals: int
    The number of decimals to round the output to.

Example:
-----
>>> print_rmse(df)

Train RMSE: 100,000.00
Test RMSE: 101,250.00
...

x = df.drop(columns=[target])
y = df[target]

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.75, random_state=85)
linreg = LinearRegression()
linreg.fit(x_train, y_train)

y_pred_train = linreg.predict(x_train)
y_pred_test = linreg.predict(x_test)

rmse_train = mean_squared_error(y_train, y_pred_train, squared=False)
rmse_test = mean_squared_error(y_test, y_pred_test, squared=False)

print('Train RMSE:', round(rmse_train, decimals))
print('Test RMSE:', round(rmse_test, decimals))

```

In [43]: `print_rmse(df)`

```
Train RMSE: 201952.03
Test RMSE: 199371.12
```

These are fairly large values for the RMSE but both the train and test splits are relatively in line with one another which indicates an appropriately fitted model.

Finally, I'll plot the residuals in a Q-Q plot to check for normality.

In [44]: `def get_qqplot(model):`

```

...
Description:
-----
Takes an OLS model and returns a Q-Q plot of the model residuals.

Parameters:
-----
model : a fitted statsmodels.api.OLS() model

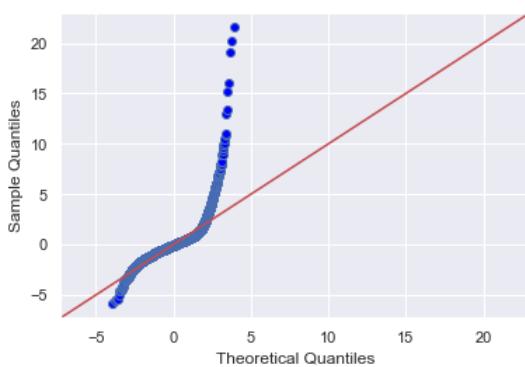
Example:
-----
>>> get_qqplot(model)

<matplotlib.figure.Figure>
...

return sm.graphics.qqplot(model.resid, dist=stats.norm, line='45', fit=True)

```

In [45]: `get_qqplot(model);`



The residuals of the baseline model are not normally distributed which violates a core assumption of linear regression. I will attempt to correct this in subsequent models.

Removing Outliers

In [46]: `df.describe()`

Out[46]:

| | price | bedrooms | bathrooms | sqft_lot | floors | waterfront | view | condition | grade | sqft_above | sqft_basement | |
|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|---------------|----|
| count | 2.159700e+04 | 21597.000000 | 21597.000000 | 2.159700e+04 | 21597.000000 | 21597.000000 | 21597.000000 | 21597.000000 | 21597.000000 | 21597.000000 | 21597.000000 | 21 |
| mean | 5.402966e+05 | 3.373200 | 2.115826 | 1.509941e+04 | 1.494096 | 0.006760 | 0.239015 | 3.409825 | 7.657915 | 1788.596842 | 285.716581 | 1 |
| std | 3.673681e+05 | 0.926299 | 0.768984 | 4.141264e+04 | 0.539683 | 0.081944 | 0.770479 | 0.650546 | 1.173200 | 827.759761 | 439.819830 | |
| min | 7.800000e+04 | 1.000000 | 0.500000 | 5.200000e+02 | 1.000000 | 0.000000 | 0.000000 | 1.000000 | 3.000000 | 370.000000 | 0.000000 | 1 |
| 25% | 3.220000e+05 | 3.000000 | 1.750000 | 5.040000e+03 | 1.000000 | 0.000000 | 0.000000 | 3.000000 | 7.000000 | 1190.000000 | 0.000000 | 1 |
| 50% | 4.500000e+05 | 3.000000 | 2.250000 | 7.618000e+03 | 1.500000 | 0.000000 | 0.000000 | 3.000000 | 7.000000 | 1560.000000 | 0.000000 | 1 |
| 75% | 6.450000e+05 | 4.000000 | 2.500000 | 1.068500e+04 | 2.000000 | 0.000000 | 0.000000 | 4.000000 | 8.000000 | 2210.000000 | 550.000000 | 1 |
| max | 7.700000e+06 | 33.000000 | 8.000000 | 1.651359e+06 | 3.500000 | 1.000000 | 4.000000 | 5.000000 | 13.000000 | 9410.000000 | 4820.000000 | 2 |

The maximum amount of bedrooms being 33 immediately jumps out to me as warranting further investigation.

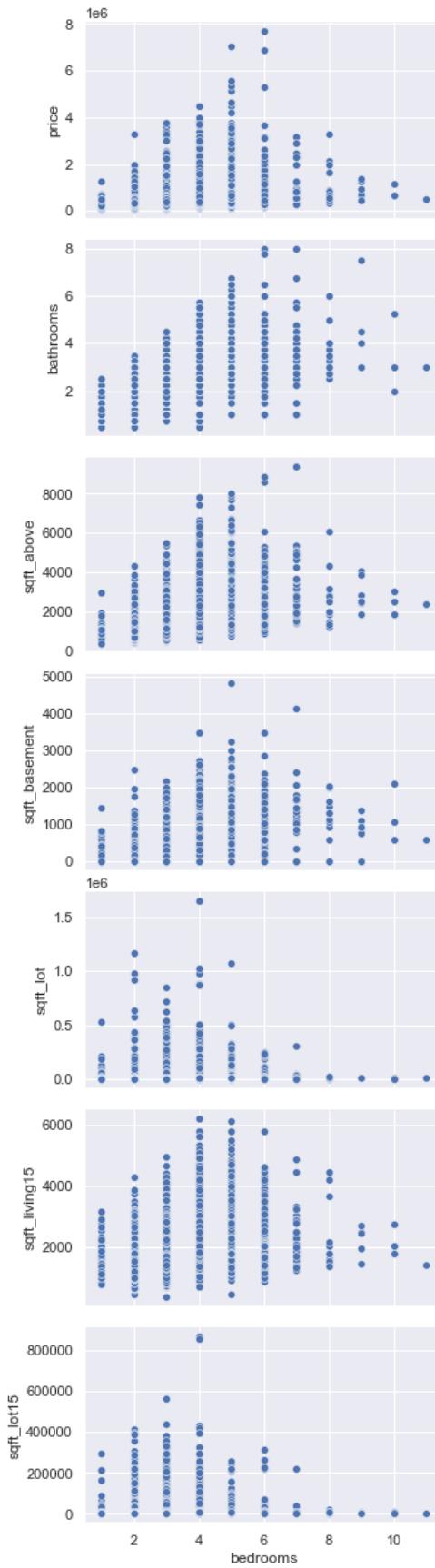
In [47]: `df[df.bedrooms == 33]`

Out[47]:

| | price | bedrooms | bathrooms | sqft_lot | floors | waterfront | view | condition | grade | sqft_above | sqft_basement | yr_built | zipcode | lat | long | sqft_living |
|--------------|----------|----------|-----------|----------|--------|------------|------|-----------|-------|------------|---------------|----------|---------|---------|----------|-------------|
| 15856 | 640000.0 | 33 | 1.75 | 6000 | 1.0 | 0.0 | 0.0 | 5 | 7 | 1040 | 580 | 1947 | 98103 | 47.6878 | -122.331 | 1 |

The price, bathrooms, and square footage of this property do not meet the expectations of a 33 bedroom house. This entry was likely a typo that should have shown only 3 bedrooms. I'll plot bedrooms against some of the other columns as a quick check for anything else that looks off.

In [48]: `sns.pairplot(df[df.bedrooms < 33],
 x_vars=['bedrooms'],
 y_vars=['price', 'bathrooms', 'sqft_above', 'sqft_basement', 'sqft_lot',
 'sqft_living15', 'sqft_lot15'],
 diag_kind=None,
 aspect=2
)`



It appears that houses with more than 8 bedrooms have counterintuitive characteristics such as less bathrooms, less square footage, and a lower price. There could be a number of reasons for these discrepancies including:

- Incorrect data entry
- Nontraditional housing such as dorms / communal living
- Older housing

```
In [49]: print(f'Number of houses with more than 8 bedrooms: {len(df[df.bedrooms > 8])}')
```

Number of houses with more than 8 bedrooms: 11

With only 11 houses having more than 8 bedrooms, I'm comfortable with simply dropping those rows.

```
In [50]: df_no_outliers = df[df.bedrooms <= 8]
df_no_outliers.bedrooms.describe()
```

```
Out[50]: count    21586.00000
mean      3.368989
std       0.894531
min      1.000000
25%      3.000000
50%      3.000000
75%      4.000000
max      8.000000
Name: bedrooms, dtype: float64
```

Next, I also want to investigate the minimum number of bathrooms being only 0.5.

```
In [51]: df_no_outliers[df_no_outliers.bathrooms == 0.5]
```

```
Out[51]:   price  bedrooms  bathrooms  sqft_lot  floors  waterfront  view  condition  grade  sqft_above  sqft_basement  yr_built  zipcode  lat  long  sqft_livin
  2259  273000.0        2        0.5     7750      1.0        0.0      0.0          4      6        590           590      1945    98155  47.7690 -122.316      1
  10413 1090000.0       2        0.5     6900      1.0        0.0      0.0          3      5        580            0      1941    98118  47.5135 -122.262      1
  11662 255000.0        1        0.5     1642      1.0        0.0      0.0          3      6        500           380      1910    98126  47.5732 -122.372      1
  12029 312500.0        4        0.5     5570      2.0        0.0      0.0          3      8       2300            0      1996    98092  47.3285 -122.168      1
```

Of the four results, the first three look like they could possibly be a dorm / communal living type of property given their `sqft_above` values. The fourth result of a property with 4 bedrooms and 2,300 sqft is definitely an outlier and should have at least one full bath. This will be dropped but I'll keep the first three.

```
In [52]: df_no_outliers.drop(index=12029, inplace=True)
```

```
# Checking to make sure it was correctly dropped
df_no_outliers[df_no_outliers.bathrooms == 0.5]
```

```
E:\Programs\anaconda3\envs\learn-env\lib\site-packages\pandas\core\frame.py:4163: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy
return super().drop()
```

```
Out[52]:   price  bedrooms  bathrooms  sqft_lot  floors  waterfront  view  condition  grade  sqft_above  sqft_basement  yr_built  zipcode  lat  long  sqft_livin
  2259  273000.0        2        0.5     7750      1.0        0.0      0.0          4      6        590           590      1945    98155  47.7690 -122.316      1
  10413 1090000.0       2        0.5     6900      1.0        0.0      0.0          3      5        580            0      1941    98118  47.5135 -122.262      1
  11662 255000.0        1        0.5     1642      1.0        0.0      0.0          3      6        500           380      1910    98126  47.5732 -122.372      1
```

The final adjustment for this second model is to remove the `floors` column which is not significant.

```
In [53]: pd.set_option('mode.chained_assignment', None) # Supressing the 'SettingWithCopyWarning'
df_no_outliers.drop(columns=['floors'], inplace=True)
df_no_outliers.columns
```

```
Out[53]: Index(['price', 'bedrooms', 'bathrooms', 'sqft_lot', 'waterfront', 'view',
       'condition', 'grade', 'sqft_above', 'sqft_basement', 'yr_built',
       'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15', 'month'],
      dtype='object')
```

```
In [54]: model_2 = fit_model(df_no_outliers)
model_2.summary()
```

```
Out[54]:
```

| | coef | std err | t | P> t | [0.025 | 0.975] |
|--------------------------|------------------|----------------------------|-------------|----------------------|------------|-----------|
| const | 6.702e+06 | 2.88e+06 | 2.326 | 0.020 | 1.05e+06 | 1.24e+07 |
| bedrooms | -3.933e+04 | 1993.587 | -19.726 | 0.000 | -4.32e+04 | -3.54e+04 |
| bathrooms | 4.897e+04 | 3115.535 | 15.718 | 0.000 | 4.29e+04 | 5.51e+04 |
| sqft_lot | 0.1177 | 0.048 | 2.456 | 0.014 | 0.024 | 0.212 |
| waterfront | 6.23e+05 | 1.81e+04 | 34.422 | 0.000 | 5.88e+05 | 6.59e+05 |
| view | 5.327e+04 | 2101.656 | 25.345 | 0.000 | 4.91e+04 | 5.74e+04 |
| condition | 2.517e+04 | 2312.877 | 10.882 | 0.000 | 2.06e+04 | 2.97e+04 |
| grade | 9.774e+04 | 2150.266 | 45.456 | 0.000 | 9.35e+04 | 1.02e+05 |
| Dep. Variable: | price | R-squared: | 0.700 | sqft_above | 182.1789 | 3.623 |
| Model: | OLS | Adj. R-squared: | 0.700 | sqft_basement | 146.3259 | 4.169 |
| Method: | Least Squares | F-statistic: | 3149. | yr_builtin | -2758.2636 | 67.359 |
| Date: | Mon, 07 Jun 2021 | Prob (F-statistic): | 0.00 | zipcode | -584.1918 | 32.840 |
| Time: | 12:42:28 | Log-Likelihood: | -2.9422e+05 | lat | 6.006e+05 | 1.07e+04 |
| No. Observations: | 21585 | AIC: | 5.885e+05 | long | -2.193e+05 | 1.31e+04 |
| Df Residuals: | 21568 | BIC: | 5.886e+05 | sqft_living15 | 20.3212 | 3.422 |
| Df Model: | 16 | | | sqft_lot15 | -0.3822 | 0.073 |
| Covariance Type: | nonrobust | | | month | -3056.2974 | 440.006 |
| Omnibus: | 18241.957 | Durbin-Watson: | 1.989 | | | |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 1807596.412 | | | |
| Skew: | 3.536 | Prob(JB): | 0.00 | | | |
| Kurtosis: | 47.270 | Cond. No. | 2.12e+08 | | | |

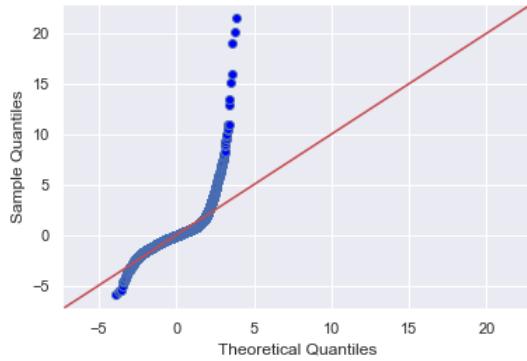
Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 2.12e+08. This might indicate that there are strong multicollinearity or other numerical problems.

```
In [55]: print_rmse(df_no_outliers)
```

```
Train RMSE: 206092.07
Test RMSE: 185791.44
```

```
In [56]: get_qqplot(model_2);
```



Observations:

- No change in the R-squared value - still at 0.700
- Slight decrease in the test RMSE but slight *increase* in the train RMSE
- The Q-Q plot still shows non-normality in the residuals

Categorical Variables

The first two versions of the model were completing neglecting the fact that certain columns were being treated as continuous data when they are in fact categorical. To adjust for this going forward, I'll convert the categorical columns into multiple columns filled with dummy variables.

```
In [57]: df_categoricals = df_no_outliers.copy()
```

```
df_categoricals.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 21585 entries, 0 to 21596
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype  
 --- 
 0   price        21585 non-null   float64 
 1   bedrooms     21585 non-null   int64  
 2   bathrooms    21585 non-null   float64 
 3   sqft_lot     21585 non-null   int64  
 4   waterfront   21585 non-null   float64 
 5   view         21585 non-null   float64 
 6   condition    21585 non-null   int64  
 7   grade        21585 non-null   int64  
 8   sqft_above   21585 non-null   int64  
 9   sqft_basement 21585 non-null   int64  
 10  yr_built    21585 non-null   int64  
 11  zipcode     21585 non-null   int64  
 12  lat          21585 non-null   float64 
 13  long         21585 non-null   float64 
 14  sqft_living15 21585 non-null   int64  
 15  sqft_lot15  21585 non-null   int64  
 16  month        21585 non-null   int64  
dtypes: float64(6), int64(11)
memory usage: 3.0 MB
```

```
In [58]: categoricals = ['waterfront', 'view', 'condition', 'grade', 'zipcode', 'month']

# Generating a temporary dataframe for each of the categorical columns
temp_dfs = [df_categoricals]
for cat in categoricals:
    dummy = pd.get_dummies(df_categoricals[cat], prefix=cat, drop_first=True)
    temp_dfs.append(dummy)

# Combining them all together
df_categoricals = pd.concat(temp_dfs, axis=1)

# Dropping the original columns now that the dummies exist
df_categoricals.drop(columns=categoricals, inplace=True)
```

```
In [59]: df_categoricals.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21585 entries, 0 to 21596
Data columns (total 110 columns):
 #   Column      Dtype  
 --- 
 0   price        float64 
 1   bedrooms     int64  
 2   bathrooms    float64 
 3   sqft_lot     int64  
 4   sqft_above   int64  
 5   sqft_basement  int64  
 6   yr_built    int64  
 7   lat          float64 
 8   long         float64 
 9   sqft_living15  int64  
 10  sqft_lot15  int64  
 11  waterfront_1.0 uint8  
 12  view_1.0    uint8  
 13  view_2.0    uint8  
 14  view_3.0    uint8  
 15  view_4.0    uint8  
 16  condition_2 uint8  
 17  condition_3 uint8  
 18  condition_4 uint8  
 19  condition_5 uint8  
 20  grade_4     uint8  
 21  grade_5     uint8  
 22  grade_6     uint8  
 23  grade_7     uint8  
 24  grade_8     uint8  
 25  grade_9     uint8  
 26  grade_10    uint8  
 27  grade_11    uint8  
 28  grade_12    uint8  
 29  grade_13    uint8  
 30  zipcode_98002 uint8  
 31  zipcode_98003 uint8  
 32  zipcode_98004 uint8  
 33  zipcode_98005 uint8  
 34  zipcode_98006 uint8  
 35  zipcode_98007 uint8  
 36  zipcode_98008 uint8  
 37  zipcode_98010 uint8  
 38  zipcode_98011 uint8  
 39  zipcode_98014 uint8  
 40  zipcode_98019 uint8  
 41  zipcode_98022 uint8  
 42  zipcode_98023 uint8  
 43  zipcode_98024 uint8  
 44  zipcode_98027 uint8  
 45  zipcode_98028 uint8  
 46  zipcode_98029 uint8  
 47  zipcode_98030 uint8  
 48  zipcode_98031 uint8
```

```

49 zipcode_98032 uint8
50 zipcode_98033 uint8
51 zipcode_98034 uint8
52 zipcode_98038 uint8
53 zipcode_98039 uint8
54 zipcode_98040 uint8
55 zipcode_98042 uint8
56 zipcode_98045 uint8
57 zipcode_98052 uint8
58 zipcode_98053 uint8
59 zipcode_98055 uint8
60 zipcode_98056 uint8
61 zipcode_98058 uint8
62 zipcode_98059 uint8
63 zipcode_98065 uint8
64 zipcode_98070 uint8
65 zipcode_98072 uint8
66 zipcode_98074 uint8
67 zipcode_98075 uint8
68 zipcode_98077 uint8
69 zipcode_98092 uint8
70 zipcode_98102 uint8
71 zipcode_98103 uint8
72 zipcode_98105 uint8
73 zipcode_98106 uint8
74 zipcode_98107 uint8
75 zipcode_98108 uint8
76 zipcode_98109 uint8
77 zipcode_98112 uint8
78 zipcode_98115 uint8
79 zipcode_98116 uint8
80 zipcode_98117 uint8
81 zipcode_98118 uint8
82 zipcode_98119 uint8
83 zipcode_98122 uint8
84 zipcode_98125 uint8
85 zipcode_98126 uint8
86 zipcode_98133 uint8
87 zipcode_98136 uint8
88 zipcode_98144 uint8
89 zipcode_98146 uint8
90 zipcode_98148 uint8
91 zipcode_98155 uint8
92 zipcode_98166 uint8
93 zipcode_98168 uint8
94 zipcode_98177 uint8
95 zipcode_98178 uint8
96 zipcode_98188 uint8
97 zipcode_98198 uint8
98 zipcode_98199 uint8
99 month_2 uint8
100 month_3 uint8
101 month_4 uint8
102 month_5 uint8
103 month_6 uint8
104 month_7 uint8
105 month_8 uint8
106 month_9 uint8
107 month_10 uint8
108 month_11 uint8
109 month_12 uint8

```

dtypes: float64(4), int64(7), uint8(99)
memory usage: 4.0 MB

With the dummy variables now in place, it's time to check the impact it had on model performance.

```
In [60]: model_3 = fit_model(df_categoricals)
model_3.summary()
```

Out[60]:

| | coef | std err | t | P> t | [0.025 | 0.975] |
|-----------------------|------------|----------|---------|-------|-----------|-----------|
| const | -3.303e+07 | 5.73e+06 | -5.762 | 0.000 | -4.43e+07 | -2.18e+07 |
| bedrooms | -1.334e+04 | 1540.689 | -8.662 | 0.000 | -1.64e+04 | -1.03e+04 |
| bathrooms | 2.627e+04 | 2395.323 | 10.969 | 0.000 | 2.16e+04 | 3.1e+04 |
| sqft_lot | 0.2443 | 0.036 | 6.821 | 0.000 | 0.174 | 0.315 |
| sqft_above | 158.4609 | 2.848 | 55.639 | 0.000 | 152.879 | 164.043 |
| sqft_basement | 120.3377 | 3.165 | 38.025 | 0.000 | 114.135 | 126.541 |
| yr_built | -682.5467 | 55.682 | -12.258 | 0.000 | -791.688 | -573.406 |
| lat | 2.268e+05 | 5.91e+04 | 3.840 | 0.000 | 1.11e+05 | 3.43e+05 |
| long | -1.929e+05 | 4.25e+04 | -4.540 | 0.000 | -2.76e+05 | -1.1e+05 |
| sqft_living15 | 16.9104 | 2.703 | 6.255 | 0.000 | 11.611 | 22.209 |
| sqft_lot15 | -0.1084 | 0.056 | -1.924 | 0.054 | -0.219 | 0.002 |
| waterfront_1.0 | 5.966e+05 | 1.54e+04 | 38.656 | 0.000 | 5.66e+05 | 6.27e+05 |
| view_1.0 | 8.701e+04 | 8523.128 | 10.208 | 0.000 | 7.03e+04 | 1.04e+05 |

| | view_2.v | 0.0508e+04 | 0.040./c2 | 15.300 | 0.000 | 5.0/e+04 | 1.05e+05 |
|----------------------|------------|------------|-----------|--------|-----------|-----------|----------|
| view_3.0 | 1.539e+05 | 7113.4111 | 21.638 | 0.000 | 1.4e+05 | 1.68e+05 | |
| view_4.0 | 2.98e+05 | 1.07e+04 | 27.766 | 0.000 | 2.77e+05 | 3.19e+05 | |
| condition_2 | 9.139e+04 | 3.02e+04 | 3.023 | 0.003 | 3.21e+04 | 1.51e+05 | |
| condition_3 | 1.034e+05 | 2.81e+04 | 3.675 | 0.000 | 4.82e+04 | 1.59e+05 | |
| condition_4 | 1.272e+05 | 2.82e+04 | 4.519 | 0.000 | 7.2e+04 | 1.82e+05 | |
| condition_5 | 1.706e+05 | 2.83e+04 | 6.025 | 0.000 | 1.15e+05 | 2.26e+05 | |
| grade_4 | -1.03e+05 | 1.53e+05 | -0.674 | 0.500 | -4.02e+05 | 1.96e+05 | |
| grade_5 | -1.37e+05 | 1.5e+05 | -0.912 | 0.362 | -4.32e+05 | 1.58e+05 | |
| grade_6 | -1.357e+05 | 1.5e+05 | -0.904 | 0.366 | -4.3e+05 | 1.58e+05 | |
| grade_7 | -1.293e+05 | 1.5e+05 | -0.862 | 0.389 | -4.23e+05 | 1.65e+05 | |
| grade_8 | -1.084e+05 | 1.5e+05 | -0.722 | 0.470 | -4.03e+05 | 1.86e+05 | |
| grade_9 | -3.412e+04 | 1.5e+05 | -0.227 | 0.820 | -3.28e+05 | 2.6e+05 | |
| grade_10 | 8.772e+04 | 1.5e+05 | 0.584 | 0.559 | -2.07e+05 | 3.82e+05 | |
| grade_11 | 2.852e+05 | 1.5e+05 | 1.896 | 0.058 | -9704.489 | 5.8e+05 | |
| grade_12 | 6.947e+05 | 1.51e+05 | 4.593 | 0.000 | 3.98e+05 | 9.91e+05 | |
| grade_13 | 1.809e+06 | 1.56e+05 | 11.568 | 0.000 | 1.5e+06 | 2.12e+06 | |
| zipcode_98002 | 1.576e+04 | 1.35e+04 | 1.169 | 0.242 | -1.07e+04 | 4.22e+04 | |
| zipcode_98003 | -1.208e+04 | 1.21e+04 | -1.002 | 0.316 | -3.57e+04 | 1.15e+04 | |
| zipcode_98004 | 7.104e+05 | 2.19e+04 | 32.419 | 0.000 | 6.67e+05 | 7.53e+05 | |
| zipcode_98005 | 2.61e+05 | 2.34e+04 | 11.144 | 0.000 | 2.15e+05 | 3.07e+05 | |
| zipcode_98006 | 2.163e+05 | 1.91e+04 | 11.300 | 0.000 | 1.79e+05 | 2.54e+05 | |
| zipcode_98007 | 2.084e+05 | 2.42e+04 | 8.627 | 0.000 | 1.61e+05 | 2.56e+05 | |
| zipcode_98008 | 2.244e+05 | 2.3e+04 | 9.777 | 0.000 | 1.79e+05 | 2.69e+05 | |
| zipcode_98010 | 1.098e+05 | 2.05e+04 | 5.344 | 0.000 | 6.95e+04 | 1.5e+05 | |
| zipcode_98011 | 4.857e+04 | 2.98e+04 | 1.628 | 0.104 | -9921.803 | 1.07e+05 | |
| zipcode_98014 | 9.376e+04 | 3.28e+04 | 2.860 | 0.004 | 2.95e+04 | 1.58e+05 | |
| zipcode_98019 | 5.801e+04 | 3.23e+04 | 1.795 | 0.073 | -5345.957 | 1.21e+05 | |
| zipcode_98022 | 6.898e+04 | 1.79e+04 | 3.856 | 0.000 | 3.39e+04 | 1.04e+05 | |
| zipcode_98023 | -4.302e+04 | 1.11e+04 | -3.880 | 0.000 | -6.48e+04 | -2.13e+04 | |
| zipcode_98024 | 1.674e+05 | 2.89e+04 | 5.802 | 0.000 | 1.11e+05 | 2.24e+05 | |
| zipcode_98027 | 1.601e+05 | 1.96e+04 | 8.156 | 0.000 | 1.22e+05 | 1.99e+05 | |
| zipcode_98028 | 3.485e+04 | 2.9e+04 | 1.203 | 0.229 | -2.19e+04 | 9.16e+04 | |
| zipcode_98029 | 2.123e+05 | 2.24e+04 | 9.468 | 0.000 | 1.68e+05 | 2.56e+05 | |
| zipcode_98030 | 7812.5881 | 1.32e+04 | 0.590 | 0.555 | -1.82e+04 | 3.38e+04 | |
| zipcode_98031 | 1.223e+04 | 1.38e+04 | 0.886 | 0.376 | -1.48e+04 | 3.93e+04 | |
| zipcode_98032 | -1.127e+04 | 1.6e+04 | -0.704 | 0.482 | -4.27e+04 | 2.01e+04 | |
| zipcode_98033 | 2.941e+05 | 2.49e+04 | 11.829 | 0.000 | 2.45e+05 | 3.43e+05 | |
| zipcode_98034 | 1.22e+05 | 2.67e+04 | 4.576 | 0.000 | 6.98e+04 | 1.74e+05 | |
| zipcode_98038 | 6.739e+04 | 1.49e+04 | 4.526 | 0.000 | 3.82e+04 | 9.66e+04 | |
| zipcode_98039 | 1.166e+06 | 2.97e+04 | 39.327 | 0.000 | 1.11e+06 | 1.22e+06 | |
| zipcode_98040 | 4.602e+05 | 1.94e+04 | 23.747 | 0.000 | 4.22e+05 | 4.98e+05 | |
| zipcode_98042 | 2.329e+04 | 1.27e+04 | 1.835 | 0.066 | -1585.209 | 4.82e+04 | |
| zipcode_98045 | 1.611e+05 | 2.75e+04 | 5.857 | 0.000 | 1.07e+05 | 2.15e+05 | |
| zipcode_98052 | 1.866e+05 | 2.54e+04 | 7.351 | 0.000 | 1.37e+05 | 2.36e+05 | |
| zipcode_98053 | 1.746e+05 | 2.72e+04 | 6.416 | 0.000 | 1.21e+05 | 2.28e+05 | |
| zipcode_98055 | 1.865e+04 | 1.54e+04 | 1.214 | 0.225 | -1.15e+04 | 4.87e+04 | |
| zipcode_98056 | 6.048e+04 | 1.67e+04 | 3.624 | 0.000 | 2.78e+04 | 9.32e+04 | |
| zipcode_98058 | 2.822e+04 | 1.45e+04 | 1.943 | 0.052 | -248.800 | 5.67e+04 | |
| zipcode_98059 | 6.587e+04 | 1.64e+04 | 4.020 | 0.000 | 3.37e+04 | 9.8e+04 | |
| zipcode_98065 | 1.172e+05 | 2.53e+04 | 4.624 | 0.000 | 6.75e+04 | 1.67e+05 | |
| zipcode_98070 | -5.319e+04 | 1.94e+04 | -2.748 | 0.006 | -9.11e+04 | -1.52e+04 | |

| | | | | | | | | | | |
|--------------------------|------------------|----------------------------|-------------|-----------------|------------|----------|--------|-------|-----------|----------|
| Dep. Variable: | price | R-squared: | 0.835 | month_2 | 8.843e+04 | 2.97e+04 | 2.979 | 0.003 | 3.02e+04 | 1.47e+05 |
| Model: | OLS | Adj. R-squared: | 0.834 | month_3 | 2.928e+04 | 5916.500 | 4.948 | 0.000 | 1.77e+04 | 4.09e+04 |
| Method: | Least Squares | F-statistic: | 997.4 | month_4 | 3.594e+04 | 5756.083 | 6.244 | 0.000 | 2.47e+04 | 4.72e+04 |
| Date: | Mon, 07 Jun 2021 | Prob (F-statistic): | 0.00 | month_5 | 6183.2874 | 5688.130 | 1.087 | 0.277 | -4965.871 | 1.73e+04 |
| Time: | 12:42:28 | Log-Likelihood: | -2.8777e+05 | month_6 | 1425.7513 | 5781.644 | 0.247 | 0.805 | -9906.701 | 1.28e+04 |
| No. Observations: | 21585 | AIC: | 5.758e+05 | month_7 | -1875.5697 | 5766.517 | -0.325 | 0.745 | -1.32e+04 | 9427.234 |
| Df Residuals: | 21475 | BIC: | 5.766e+05 | month_8 | -2052.5228 | 5890.795 | -0.348 | 0.728 | -1.36e+04 | 9493.875 |
| Df Model: | 109 | | | month_9 | -7385.5655 | 5974.838 | -1.236 | 0.216 | -1.91e+04 | 4325.562 |
| Covariance Type: | nonrobust | | | month_10 | -6496.9318 | 5917.817 | -1.098 | 0.272 | -1.81e+04 | 5102.429 |
| Omnibus: | 16952.310 | Durbin-Watson: | 1.996 | month_11 | -4395.3215 | 6244.931 | -0.704 | 0.482 | -1.66e+04 | 7845.208 |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 2191278.257 | month_12 | -2103.6762 | 6190.477 | -0.340 | 0.734 | -1.42e+04 | 1e+04 |
| Skew: | 3.036 | Prob(JB): | 0.00 | | | | | | | |
| Kurtosis: | 51.985 | Cond. No. | 2.85e+08 | | | | | | | |

Notes:

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

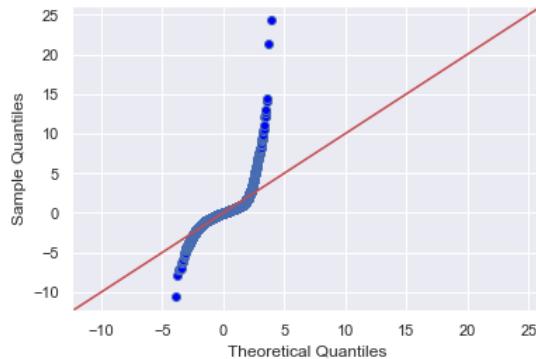
[2] The condition number is large, 2.85e+08. This might indicate that there are

strong multicollinearity or other numerical problems.

```
In [61]: print_rmse(df_categoricals)
```

Train RMSE: 150726.83
Test RMSE: 147645.95

```
In [62]: get_qqplot(model_3);
```



Observations:

- Large improvement in the R-squared value
- Many of the dummy variable columns are showing non-significant p-values
- Also good improvement in the RMSE values, model remains decently fit
- The Q-Q plot is still showing non-normality in the residuals

Transformations

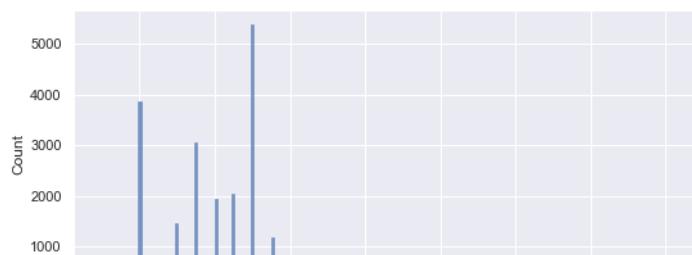
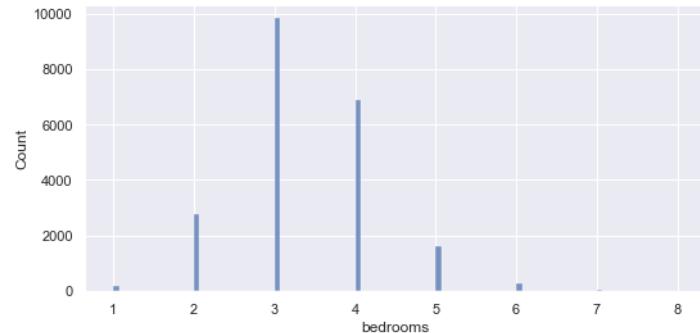
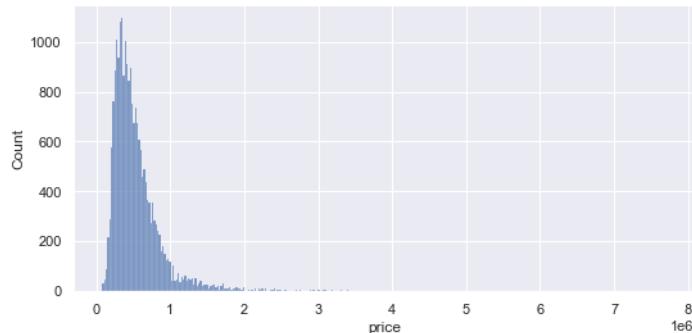
Log Transformation

```
In [63]: df_log = df_categoricals.copy()  
continuous = df_log.columns[:11]  
continuous
```

```
Out[63]: Index(['price', 'bedrooms', 'bathrooms', 'sqft_lot', 'sqft_above',  
       'sqft_basement', 'yr_builtin', 'lat', 'long', 'sqft_living15',  
       'sqft_lot15'],  
       dtype='object')
```

Checking the distributions of each of the continuous variables:

```
In [64]: fig, axes = plt.subplots(nrows=6, ncols=2, figsize=(16, 24))  
axes = axes.reshape(-1)  
  
for i, col in enumerate(continuous):  
    sns.histplot(data=df_log[col], ax=axes[i])  
  
fig.tight_layout(pad=2.0)
```





All of the continuous variables exhibit skewness to some degree. Applying a log transformation to each of those columns may help and increase the model's R-squared value. However, `sqft_basement` contains many zero values and `long` contains all negative values which are invalid for log transformations. These will remain as they currently are.

```
In [65]: continuous = continuous.drop(['sqft_basement', 'long'])

for col in continuous:
    df_log[col] = df_log[col].map(lambda x: np.log(x))
    df_log.rename(columns={col: col+'_log'}, inplace=True)
```

```
In [66]: df_log.head()
```

```
Out[66]:
```

| | price_log | bedrooms_log | bathrooms_log | sqft_lot_log | sqft_above_log | sqft_basement | yr_builtin_log | lat_log | long | sqft_living15_log | ... | month_3 | month_4 | mo |
|---|-----------|--------------|---------------|--------------|----------------|---------------|----------------|----------|----------|-------------------|-----|---------|---------|----|
| 0 | 12.309982 | 1.098612 | 0.000000 | 8.639411 | 7.073270 | 0 | 7.578145 | 3.860965 | -122.257 | 7.200425 | ... | 0 | 0 | |
| 1 | 13.195614 | 1.098612 | 0.810930 | 8.887653 | 7.682482 | 400 | 7.576097 | 3.865372 | -122.319 | 7.432484 | ... | 0 | 0 | |
| 2 | 12.100712 | 0.693147 | 0.000000 | 9.210340 | 6.646391 | 0 | 7.566828 | 3.865726 | -122.233 | 7.908387 | ... | 0 | 0 | |
| 3 | 13.311329 | 1.386294 | 1.098612 | 8.517193 | 6.956545 | 910 | 7.583248 | 3.861168 | -122.393 | 7.215240 | ... | 0 | 0 | |
| 4 | 13.142166 | 1.098612 | 0.693147 | 8.997147 | 7.426549 | 0 | 7.594381 | 3.863186 | -122.045 | 7.495542 | ... | 0 | 0 | |

5 rows × 110 columns

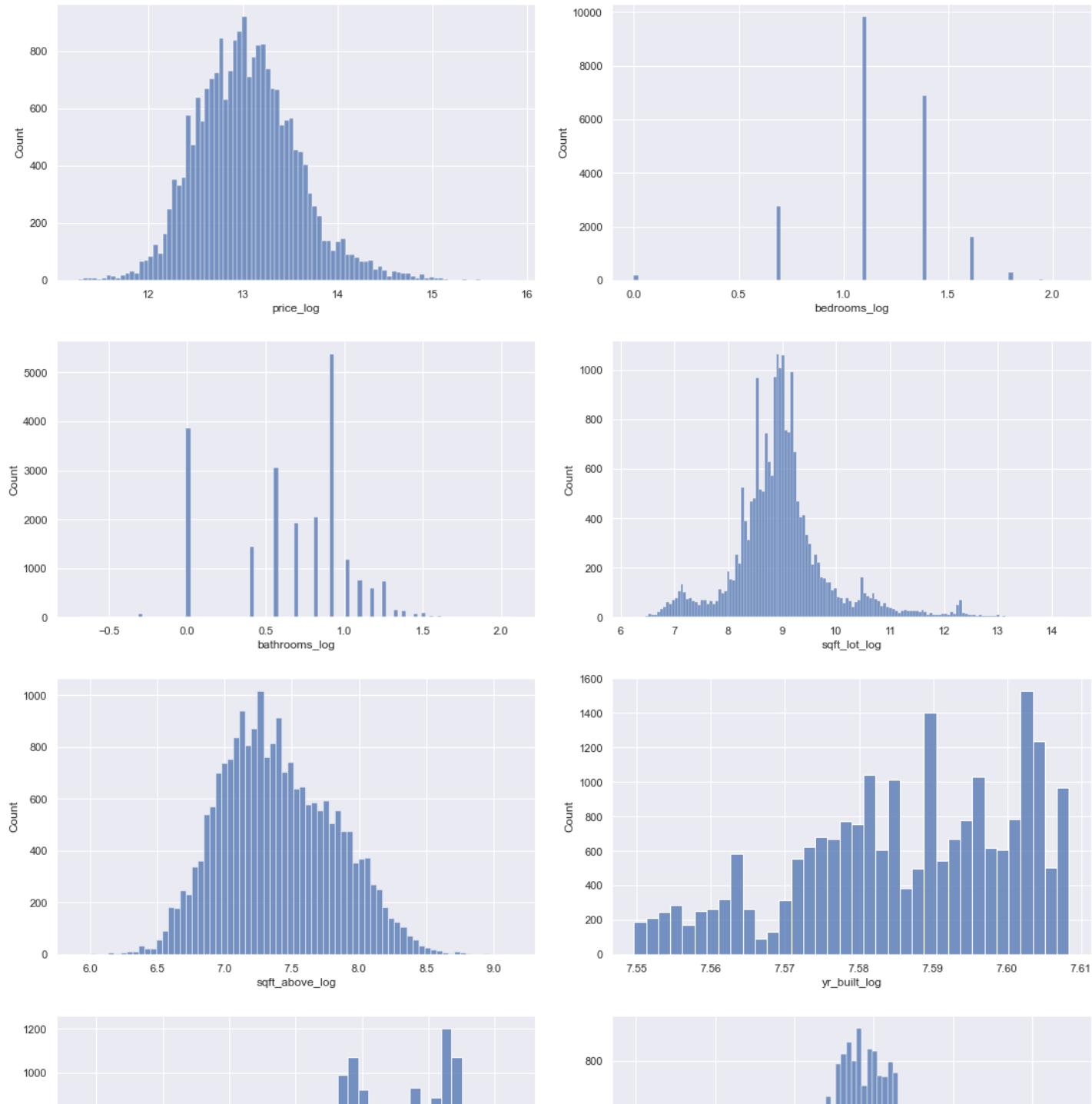


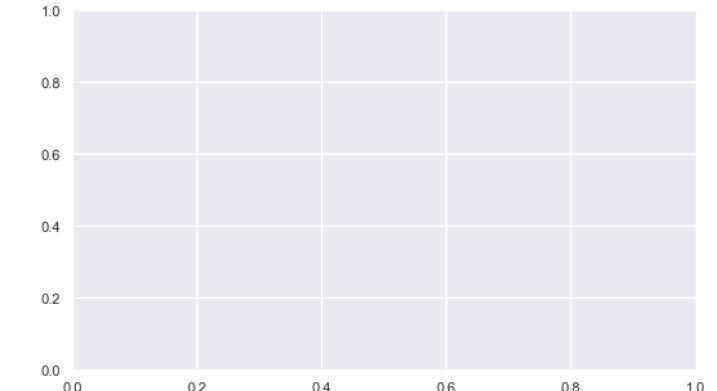
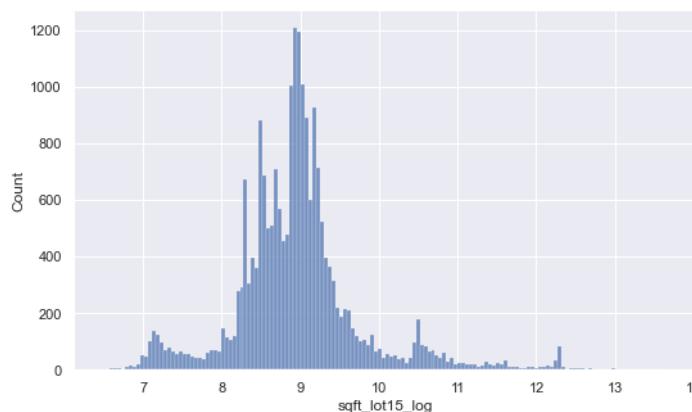
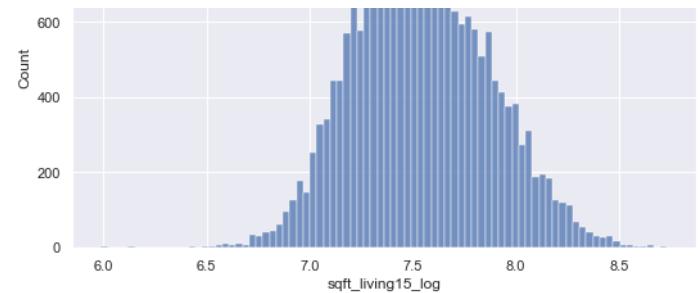
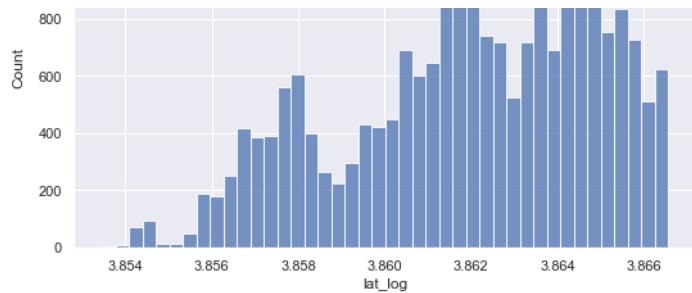
Checking the distributions again:

```
In [67]: fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(16, 24))
axes = axes.reshape(-1)

for i, col in enumerate(continuous):
    sns.histplot(data=df_log[col+'_log'], ax=axes[i])

fig.tight_layout(pad=2.0)
```





It's not perfect, but it's definitely better than before. Time to see how it impacted the model.

```
In [68]: model_4 = fit_model(df_log, target='price_log')
model_4.summary()
```

Out[68]:

| | coef | std err | t | P> t | [0.025 | 0.975] |
|--------------------------|-----------|----------|--------|-------|----------|----------|
| const | -132.9987 | 14.087 | -9.441 | 0.000 | -160.610 | -105.388 |
| bedrooms_log | -0.0267 | 0.006 | -4.460 | 0.000 | -0.038 | -0.015 |
| bathrooms_log | 0.0872 | 0.006 | 15.649 | 0.000 | 0.076 | 0.098 |
| sqft_lot_log | 0.0761 | 0.004 | 21.231 | 0.000 | 0.069 | 0.083 |
| sqft_above_log | 0.3683 | 0.006 | 57.266 | 0.000 | 0.356 | 0.381 |
| sqft_basement | 0.0001 | 3.76e-06 | 32.678 | 0.000 | 0.000 | 0.000 |
| yr_builtin_log | -0.9847 | 0.137 | -7.176 | 0.000 | -1.254 | -0.716 |
| lat_log | 25.1321 | 3.359 | 7.481 | 0.000 | 18.547 | 31.717 |
| long | -0.4213 | 0.051 | -8.229 | 0.000 | -0.522 | -0.321 |
| sqft_living15_log | 0.1638 | 0.007 | 24.891 | 0.000 | 0.151 | 0.177 |
| sqft_lot15_log | -0.0179 | 0.004 | -4.508 | 0.000 | -0.026 | -0.010 |
| waterfront_1.0 | 0.4465 | 0.018 | 24.185 | 0.000 | 0.410 | 0.483 |
| view_1.0 | 0.1078 | 0.010 | 10.573 | 0.000 | 0.088 | 0.128 |
| view_2.0 | 0.0942 | 0.006 | 15.591 | 0.000 | 0.082 | 0.106 |
| view_3.0 | 0.1658 | 0.008 | 19.521 | 0.000 | 0.149 | 0.182 |
| view_4.0 | 0.2718 | 0.013 | 21.169 | 0.000 | 0.247 | 0.297 |
| condition_2 | 0.1567 | 0.036 | 4.330 | 0.000 | 0.086 | 0.228 |
| condition_3 | 0.2987 | 0.034 | 8.864 | 0.000 | 0.233 | 0.365 |
| condition_4 | 0.3368 | 0.034 | 9.987 | 0.000 | 0.271 | 0.403 |
| condition_5 | 0.4046 | 0.034 | 11.922 | 0.000 | 0.338 | 0.471 |
| grade_4 | -0.5176 | 0.183 | -2.831 | 0.005 | -0.876 | -0.159 |
| grade_5 | -0.5265 | 0.180 | -2.927 | 0.003 | -0.879 | -0.174 |
| grade_6 | -0.4466 | 0.180 | -2.486 | 0.013 | -0.799 | -0.095 |
| grade_7 | -0.3673 | 0.180 | -2.045 | 0.041 | -0.719 | -0.015 |
| grade_8 | -0.2875 | 0.180 | -1.600 | 0.110 | -0.640 | 0.065 |
| grade_9 | -0.1669 | 0.180 | -0.928 | 0.353 | -0.519 | 0.185 |
| grade_10 | -0.0756 | 0.180 | -0.420 | 0.674 | -0.428 | 0.277 |
| grade_11 | 0.0349 | 0.180 | 0.194 | 0.847 | -0.318 | 0.388 |
| grade_12 | 0.1586 | 0.181 | 0.876 | 0.281 | 0.106 | 0.512 |

| | 0.1500 | 0.101 | 0.070 | 0.301 | -0.190 | 0.513 |
|----------------------|---------|-------|--------|-------|--------|--------|
| grade_12 | | | | | | |
| grade_13 | 0.3924 | 0.187 | 2.099 | 0.036 | 0.026 | 0.759 |
| zipcode_98002 | 0.0219 | 0.016 | 1.352 | 0.176 | -0.010 | 0.054 |
| zipcode_98003 | 0.0011 | 0.014 | 0.073 | 0.942 | -0.027 | 0.029 |
| zipcode_98004 | 0.9895 | 0.026 | 37.681 | 0.000 | 0.938 | 1.041 |
| zipcode_98005 | 0.6086 | 0.028 | 21.698 | 0.000 | 0.554 | 0.664 |
| zipcode_98006 | 0.5609 | 0.023 | 24.436 | 0.000 | 0.516 | 0.606 |
| zipcode_98007 | 0.5466 | 0.029 | 18.877 | 0.000 | 0.490 | 0.603 |
| zipcode_98008 | 0.5633 | 0.028 | 20.464 | 0.000 | 0.509 | 0.617 |
| zipcode_98010 | 0.3364 | 0.025 | 13.680 | 0.000 | 0.288 | 0.385 |
| zipcode_98011 | 0.2426 | 0.036 | 6.796 | 0.000 | 0.173 | 0.313 |
| zipcode_98014 | 0.2802 | 0.039 | 7.146 | 0.000 | 0.203 | 0.357 |
| zipcode_98019 | 0.2197 | 0.039 | 5.677 | 0.000 | 0.144 | 0.296 |
| zipcode_98022 | 0.2091 | 0.021 | 9.763 | 0.000 | 0.167 | 0.251 |
| zipcode_98023 | -0.0553 | 0.013 | -4.169 | 0.000 | -0.081 | -0.029 |
| zipcode_98024 | 0.4379 | 0.034 | 12.693 | 0.000 | 0.370 | 0.506 |
| zipcode_98027 | 0.4923 | 0.024 | 20.895 | 0.000 | 0.446 | 0.538 |
| zipcode_98028 | 0.1903 | 0.035 | 5.488 | 0.000 | 0.122 | 0.258 |
| zipcode_98029 | 0.5874 | 0.027 | 21.756 | 0.000 | 0.534 | 0.640 |
| zipcode_98030 | 0.0502 | 0.016 | 3.159 | 0.002 | 0.019 | 0.081 |
| zipcode_98031 | 0.0601 | 0.017 | 3.632 | 0.000 | 0.028 | 0.093 |
| zipcode_98032 | -0.0477 | 0.019 | -2.485 | 0.013 | -0.085 | -0.010 |
| zipcode_98033 | 0.6312 | 0.030 | 21.193 | 0.000 | 0.573 | 0.690 |
| zipcode_98034 | 0.3603 | 0.032 | 11.288 | 0.000 | 0.298 | 0.423 |
| zipcode_98038 | 0.2347 | 0.018 | 13.108 | 0.000 | 0.200 | 0.270 |
| zipcode_98039 | 1.1460 | 0.035 | 32.294 | 0.000 | 1.076 | 1.216 |
| zipcode_98040 | 0.7557 | 0.023 | 32.541 | 0.000 | 0.710 | 0.801 |
| zipcode_98042 | 0.0968 | 0.015 | 6.357 | 0.000 | 0.067 | 0.127 |
| zipcode_98045 | 0.4488 | 0.033 | 13.608 | 0.000 | 0.384 | 0.513 |
| zipcode_98052 | 0.5093 | 0.030 | 16.748 | 0.000 | 0.450 | 0.569 |
| zipcode_98053 | 0.4938 | 0.033 | 15.154 | 0.000 | 0.430 | 0.558 |
| zipcode_98055 | 0.1028 | 0.018 | 5.575 | 0.000 | 0.067 | 0.139 |
| zipcode_98056 | 0.2591 | 0.020 | 12.939 | 0.000 | 0.220 | 0.298 |
| zipcode_98058 | 0.1470 | 0.017 | 8.439 | 0.000 | 0.113 | 0.181 |
| zipcode_98059 | 0.3019 | 0.020 | 15.364 | 0.000 | 0.263 | 0.340 |
| zipcode_98065 | 0.4641 | 0.031 | 15.211 | 0.000 | 0.404 | 0.524 |
| zipcode_98070 | 0.1657 | 0.023 | 7.128 | 0.000 | 0.120 | 0.211 |
| zipcode_98072 | 0.2947 | 0.036 | 8.296 | 0.000 | 0.225 | 0.364 |
| zipcode_98074 | 0.4813 | 0.029 | 16.703 | 0.000 | 0.425 | 0.538 |
| zipcode_98075 | 0.5175 | 0.028 | 18.658 | 0.000 | 0.463 | 0.572 |
| zipcode_98077 | 0.2790 | 0.037 | 7.548 | 0.000 | 0.207 | 0.351 |
| zipcode_98092 | 0.0620 | 0.014 | 4.293 | 0.000 | 0.034 | 0.090 |
| zipcode_98102 | 0.8076 | 0.031 | 26.277 | 0.000 | 0.747 | 0.868 |
| zipcode_98103 | 0.6433 | 0.029 | 22.412 | 0.000 | 0.587 | 0.700 |
| zipcode_98105 | 0.7783 | 0.030 | 26.310 | 0.000 | 0.720 | 0.836 |
| zipcode_98106 | 0.2378 | 0.021 | 11.126 | 0.000 | 0.196 | 0.280 |
| zipcode_98107 | 0.6620 | 0.030 | 22.366 | 0.000 | 0.604 | 0.720 |
| zipcode_98108 | 0.2615 | 0.024 | 11.102 | 0.000 | 0.215 | 0.308 |
| zipcode_98109 | 0.8316 | 0.031 | 27.222 | 0.000 | 0.772 | 0.891 |
| zipcode_98112 | 0.8976 | 0.027 | 33.102 | 0.000 | 0.844 | 0.951 |
| zipcode_98115 | 0.6443 | 0.029 | 22.039 | 0.000 | 0.587 | 0.702 |
| zipcode_98116 | 0.6120 | 0.024 | 25.718 | 0.000 | 0.565 | 0.659 |

| | | | | | | | | | | |
|--------------------------|------------------|----------------------------|------------|-----------------|---------|-------|--------|-------|--------|-------|
| Dep. Variable: | price_log | R-squared: | 0.885 | month_2 | 0.0215 | 0.008 | 2.808 | 0.005 | 0.007 | 0.037 |
| Model: | OLS | Adj. R-squared: | 0.884 | month_3 | 0.0486 | 0.007 | 6.861 | 0.000 | 0.035 | 0.062 |
| Method: | Least Squares | F-statistic: | 1516. | month_4 | 0.0691 | 0.007 | 10.027 | 0.000 | 0.056 | 0.083 |
| Date: | Mon, 07 Jun 2021 | Prob (F-statistic): | 0.00 | month_5 | 0.0120 | 0.007 | 1.760 | 0.078 | -0.001 | 0.025 |
| Time: | 12:42:48 | Log-Likelihood: | 6563.1 | month_6 | 0.0020 | 0.007 | 0.293 | 0.770 | -0.012 | 0.016 |
| No. Observations: | 21585 | AIC: | -1.291e+04 | month_7 | -0.0040 | 0.007 | -0.586 | 0.558 | -0.018 | 0.009 |
| Df Residuals: | 21475 | BIC: | -1.203e+04 | month_8 | -0.0034 | 0.007 | -0.486 | 0.627 | -0.017 | 0.010 |
| Df Model: | 109 | | | month_9 | -0.0094 | 0.007 | -1.315 | 0.189 | -0.023 | 0.005 |
| Covariance Type: | nonrobust | | | month_10 | -0.0099 | 0.007 | -1.401 | 0.161 | -0.024 | 0.004 |
| Omnibus: | 1609.175 | Durbin-Watson: | 2.002 | month_11 | -0.0095 | 0.007 | -1.277 | 0.202 | -0.024 | 0.005 |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 6740.814 | month_12 | -0.0006 | 0.007 | -0.076 | 0.939 | -0.015 | 0.014 |
| Skew: | -0.263 | Prob(JB): | 0.00 | | | | | | | |
| Kurtosis: | 5.687 | Cond. No. | 6.25e+06 | | | | | | | |

Notes:

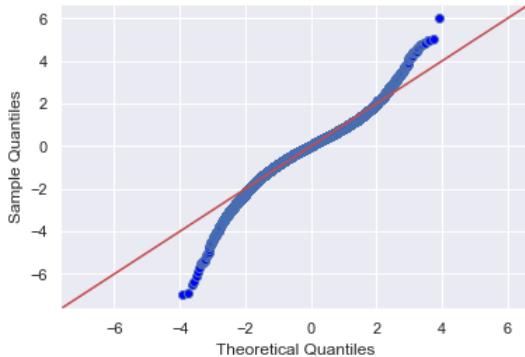
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 6.25e+06. This might indicate that there are strong multicollinearity or other numerical problems.

```
In [69]: print_rmse(df_log, target='price_log', decimals=4)
```

```
Train RMSE: 0.1781
Test RMSE: 0.1809
```

```
In [70]: get_qqplot(model_4);
```



Observations:

- Highest R-squared value yet!
- The RMSE is now in log units but is still well fit
- Large improvement in the normality of the residuals but still some issues at the tails
- There's a mismatch in the scale between different coefficients since only some variables have been log transformed while others haven't

Scaling

In an attempt to deal with the mismatch of scale between certain variables as noted above, I'll apply the `RobustScaler` from `sklearn.preprocessing`. This scaler removes the median and scales the data according to the IQR. Furthermore, this scaler in particular is less sensitive to outliers than other popular scalers such as the `MinMaxScaler` which is ideal for this particular dataset.

```
In [71]: df_scaled = df_log.copy()
columns = [col + '_scaled' for col in df_scaled.columns]

# Apply the scaler
scaler = RobustScaler()
df_scaled = scaler.fit_transform(df_scaled)

# Converting back to a dataframe and previewing
df_scaled = pd.DataFrame(df_scaled, columns=columns)
df_scaled.head()
```

```
Out[71]:   price_log_scaled  bedrooms_log_scaled  bathrooms_log_scaled  sqft_lot_log_scaled  sqft_above_log_scaled  sqft_basement_scaled  yr_built_log_scaled  lat_log_scaled  long_
0      -1.017737          0.000000         -2.273583        -0.397967        -0.450975          0.000000       -0.436758      -0.292956     -0.1
1       0.257106          0.000000         0.000000        -0.067692        0.533151          0.727273       -0.524646      0.719685     -0.2
2      -1.318976         -1.409421         -2.273583        0.361630        -1.140559          0.000000       -0.922383      0.801062     -0.0
3       0.423675          1.000000         0.806567        -0.560572        -0.639532          1.654545       -0.217823      -0.246523     -0.8
4       0.180169          0.000000         -0.330225        0.077985          0.119715          0.000000       0.259937      0.217301     0.5
```

5 rows × 110 columns

```
In [72]: model_5 = fit_model(df_scaled, target='price_log_scaled')
model_5.summary()
```

| | coef | std err | t | P> t | [0.025 | 0.975] |
|---------------------------------|---------|---------|--------|-------|--------|--------|
| const | -0.1761 | 0.261 | -0.676 | 0.499 | -0.687 | 0.335 |
| bedrooms_log_scaled | -0.0111 | 0.002 | -4.460 | 0.000 | -0.016 | -0.006 |
| bathrooms_log_scaled | 0.0448 | 0.003 | 15.649 | 0.000 | 0.039 | 0.050 |
| sqft_lot_log_scaled | 0.0824 | 0.004 | 21.231 | 0.000 | 0.075 | 0.090 |
| sqft_above_log_scaled | 0.3282 | 0.006 | 57.266 | 0.000 | 0.317 | 0.339 |
| sqft_basement_scaled | 0.0973 | 0.003 | 32.678 | 0.000 | 0.091 | 0.103 |
| yr_built_log_scaled | -0.0330 | 0.005 | -7.176 | 0.000 | -0.042 | -0.024 |
| lat_log_scaled | 0.1574 | 0.021 | 7.481 | 0.000 | 0.116 | 0.199 |
| long_scaled | -0.1231 | 0.015 | -8.229 | 0.000 | -0.152 | -0.094 |
| sqft_living15_log_scaled | 0.1084 | 0.004 | 24.891 | 0.000 | 0.100 | 0.117 |
| sqft_lot15_log_scaled | -0.0176 | 0.004 | -4.508 | 0.000 | -0.025 | -0.010 |
| waterfront_1.0_scaled | 0.6428 | 0.027 | 24.185 | 0.000 | 0.591 | 0.695 |
| view_1.0_scaled | 0.1552 | 0.015 | 10.573 | 0.000 | 0.126 | 0.184 |
| view_2.0_scaled | 0.1356 | 0.009 | 15.591 | 0.000 | 0.119 | 0.153 |
| view 3.0 scaled | 0.2386 | 0.012 | 19.521 | 0.000 | 0.215 | 0.263 |

| | | | | | | |
|-----------------------------|---------|-------|--------|-------|--------|--------|
| view_4.0_scaled | 0.3913 | 0.018 | 21.169 | 0.000 | 0.355 | 0.428 |
| condition_2_scaled | 0.2256 | 0.052 | 4.330 | 0.000 | 0.123 | 0.328 |
| condition_3_scaled | 0.4300 | 0.049 | 8.864 | 0.000 | 0.335 | 0.525 |
| condition_4_scaled | 0.4848 | 0.049 | 9.987 | 0.000 | 0.390 | 0.580 |
| condition_5_scaled | 0.5824 | 0.049 | 11.922 | 0.000 | 0.487 | 0.678 |
| grade_4_scaled | -0.7450 | 0.263 | -2.831 | 0.005 | -1.261 | -0.229 |
| grade_5_scaled | -0.7579 | 0.259 | -2.927 | 0.003 | -1.266 | -0.250 |
| grade_6_scaled | -0.6428 | 0.259 | -2.486 | 0.013 | -1.150 | -0.136 |
| grade_7_scaled | -0.5287 | 0.259 | -2.045 | 0.041 | -1.036 | -0.022 |
| grade_8_scaled | -0.4138 | 0.259 | -1.600 | 0.110 | -0.921 | 0.093 |
| grade_9_scaled | -0.2403 | 0.259 | -0.928 | 0.353 | -0.748 | 0.267 |
| grade_10_scaled | -0.1089 | 0.259 | -0.420 | 0.674 | -0.616 | 0.399 |
| grade_11_scaled | 0.0502 | 0.259 | 0.194 | 0.847 | -0.458 | 0.558 |
| grade_12_scaled | 0.2283 | 0.261 | 0.876 | 0.381 | -0.282 | 0.739 |
| grade_13_scaled | 0.5649 | 0.269 | 2.099 | 0.036 | 0.037 | 1.092 |
| zipcode_98002_scaled | 0.0315 | 0.023 | 1.352 | 0.176 | -0.014 | 0.077 |
| zipcode_98003_scaled | 0.0015 | 0.021 | 0.073 | 0.942 | -0.039 | 0.042 |
| zipcode_98004_scaled | 1.4243 | 0.038 | 37.681 | 0.000 | 1.350 | 1.498 |
| zipcode_98005_scaled | 0.8760 | 0.040 | 21.698 | 0.000 | 0.797 | 0.955 |
| zipcode_98006_scaled | 0.8075 | 0.033 | 24.436 | 0.000 | 0.743 | 0.872 |
| zipcode_98007_scaled | 0.7868 | 0.042 | 18.877 | 0.000 | 0.705 | 0.868 |
| zipcode_98008_scaled | 0.8109 | 0.040 | 20.464 | 0.000 | 0.733 | 0.889 |
| zipcode_98010_scaled | 0.4842 | 0.035 | 13.680 | 0.000 | 0.415 | 0.554 |
| zipcode_98011_scaled | 0.3493 | 0.051 | 6.796 | 0.000 | 0.249 | 0.450 |
| zipcode_98014_scaled | 0.4034 | 0.056 | 7.146 | 0.000 | 0.293 | 0.514 |
| zipcode_98019_scaled | 0.3162 | 0.056 | 5.677 | 0.000 | 0.207 | 0.425 |
| zipcode_98022_scaled | 0.3010 | 0.031 | 9.763 | 0.000 | 0.241 | 0.361 |
| zipcode_98023_scaled | -0.0797 | 0.019 | -4.169 | 0.000 | -0.117 | -0.042 |
| zipcode_98024_scaled | 0.6303 | 0.050 | 12.693 | 0.000 | 0.533 | 0.728 |
| zipcode_98027_scaled | 0.7086 | 0.034 | 20.895 | 0.000 | 0.642 | 0.775 |
| zipcode_98028_scaled | 0.2739 | 0.050 | 5.488 | 0.000 | 0.176 | 0.372 |
| zipcode_98029_scaled | 0.8455 | 0.039 | 21.756 | 0.000 | 0.769 | 0.922 |
| zipcode_98030_scaled | 0.0722 | 0.023 | 3.159 | 0.002 | 0.027 | 0.117 |
| zipcode_98031_scaled | 0.0865 | 0.024 | 3.632 | 0.000 | 0.040 | 0.133 |
| zipcode_98032_scaled | -0.0686 | 0.028 | -2.485 | 0.013 | -0.123 | -0.014 |
| zipcode_98033_scaled | 0.9085 | 0.043 | 21.193 | 0.000 | 0.824 | 0.993 |
| zipcode_98034_scaled | 0.5186 | 0.046 | 11.288 | 0.000 | 0.429 | 0.609 |
| zipcode_98038_scaled | 0.3378 | 0.026 | 13.108 | 0.000 | 0.287 | 0.388 |
| zipcode_98039_scaled | 1.6497 | 0.051 | 32.294 | 0.000 | 1.550 | 1.750 |
| zipcode_98040_scaled | 1.0878 | 0.033 | 32.541 | 0.000 | 1.022 | 1.153 |
| zipcode_98042_scaled | 0.1393 | 0.022 | 6.357 | 0.000 | 0.096 | 0.182 |
| zipcode_98045_scaled | 0.6461 | 0.047 | 13.608 | 0.000 | 0.553 | 0.739 |
| zipcode_98052_scaled | 0.7331 | 0.044 | 16.748 | 0.000 | 0.647 | 0.819 |
| zipcode_98053_scaled | 0.7108 | 0.047 | 15.154 | 0.000 | 0.619 | 0.803 |
| zipcode_98055_scaled | 0.1479 | 0.027 | 5.575 | 0.000 | 0.096 | 0.200 |
| zipcode_98056_scaled | 0.3730 | 0.029 | 12.939 | 0.000 | 0.317 | 0.430 |
| zipcode_98058_scaled | 0.2116 | 0.025 | 8.439 | 0.000 | 0.162 | 0.261 |
| zipcode_98059_scaled | 0.4346 | 0.028 | 15.364 | 0.000 | 0.379 | 0.490 |
| zipcode_98065_scaled | 0.6681 | 0.044 | 15.211 | 0.000 | 0.582 | 0.754 |
| zipcode_98070_scaled | 0.2385 | 0.033 | 7.128 | 0.000 | 0.173 | 0.304 |
| zipcode_98072_scaled | 0.4241 | 0.051 | 8.296 | 0.000 | 0.324 | 0.524 |

| | | | | | | | |
|--------------------------|----------------------|----------------------------|----------|--------|-------|--------|-------|
| | zipcode_98074_scaled | 0.6928 | 0.041 | 16.703 | 0.000 | 0.611 | 0.774 |
| | zipcode_98075_scaled | 0.7449 | 0.040 | 18.658 | 0.000 | 0.667 | 0.823 |
| | zipcode_98077_scaled | 0.4016 | 0.053 | 7.548 | 0.000 | 0.297 | 0.506 |
| | zipcode_98092_scaled | 0.0892 | 0.021 | 4.293 | 0.000 | 0.049 | 0.130 |
| | zipcode_98102_scaled | 1.1626 | 0.044 | 26.277 | 0.000 | 1.076 | 1.249 |
| | zipcode_98103_scaled | 0.9261 | 0.041 | 22.412 | 0.000 | 0.845 | 1.007 |
| | zipcode_98105_scaled | 1.1203 | 0.043 | 26.310 | 0.000 | 1.037 | 1.204 |
| | zipcode_98106_scaled | 0.3423 | 0.031 | 11.126 | 0.000 | 0.282 | 0.403 |
| | zipcode_98107_scaled | 0.9529 | 0.043 | 22.366 | 0.000 | 0.869 | 1.036 |
| | zipcode_98108_scaled | 0.3765 | 0.034 | 11.102 | 0.000 | 0.310 | 0.443 |
| | zipcode_98109_scaled | 1.1971 | 0.044 | 27.222 | 0.000 | 1.111 | 1.283 |
| | zipcode_98112_scaled | 1.2921 | 0.039 | 33.102 | 0.000 | 1.216 | 1.369 |
| | zipcode_98115_scaled | 0.9274 | 0.042 | 22.039 | 0.000 | 0.845 | 1.010 |
| | zipcode_98116_scaled | 0.8810 | 0.034 | 25.718 | 0.000 | 0.814 | 0.948 |
| | zipcode_98117_scaled | 0.8860 | 0.043 | 20.804 | 0.000 | 0.803 | 0.969 |
| | zipcode_98118_scaled | 0.5400 | 0.030 | 18.005 | 0.000 | 0.481 | 0.599 |
| | zipcode_98119_scaled | 1.1626 | 0.042 | 28.009 | 0.000 | 1.081 | 1.244 |
| | zipcode_98122_scaled | 0.9858 | 0.037 | 26.577 | 0.000 | 0.913 | 1.059 |
| | zipcode_98125_scaled | 0.5247 | 0.045 | 11.547 | 0.000 | 0.436 | 0.614 |
| | zipcode_98126_scaled | 0.6289 | 0.032 | 19.953 | 0.000 | 0.567 | 0.691 |
| | zipcode_98133_scaled | 0.3304 | 0.047 | 7.044 | 0.000 | 0.238 | 0.422 |
| | zipcode_98136_scaled | 0.7939 | 0.032 | 24.598 | 0.000 | 0.731 | 0.857 |
| | zipcode_98144_scaled | 0.8108 | 0.035 | 23.491 | 0.000 | 0.743 | 0.878 |
| | zipcode_98146_scaled | 0.2365 | 0.029 | 8.197 | 0.000 | 0.180 | 0.293 |
| | zipcode_98148_scaled | 0.1113 | 0.039 | 2.839 | 0.005 | 0.034 | 0.188 |
| | zipcode_98155_scaled | 0.2809 | 0.049 | 5.756 | 0.000 | 0.185 | 0.377 |
| | zipcode_98166_scaled | 0.3051 | 0.026 | 11.550 | 0.000 | 0.253 | 0.357 |
| | zipcode_98168_scaled | -0.0223 | 0.028 | -0.800 | 0.424 | -0.077 | 0.032 |
| | zipcode_98177_scaled | 0.4802 | 0.049 | 9.797 | 0.000 | 0.384 | 0.576 |
| | zipcode_98178_scaled | 0.1258 | 0.029 | 4.364 | 0.000 | 0.069 | 0.182 |
| | zipcode_98188_scaled | 0.0433 | 0.030 | 1.467 | 0.142 | -0.015 | 0.101 |
| | zipcode_98198_scaled | 0.0172 | 0.022 | 0.769 | 0.442 | -0.027 | 0.061 |
| | zipcode_98199_scaled | 0.9533 | 0.040 | 23.552 | 0.000 | 0.874 | 1.033 |
| | month_2_scaled | 0.0310 | 0.011 | 2.808 | 0.005 | 0.009 | 0.053 |
| | month_3_scaled | 0.0699 | 0.010 | 6.861 | 0.000 | 0.050 | 0.090 |
| Dep. Variable: | price_log_scaled | R-squared: | 0.885 | | | | |
| Model: | OLS | Adj. R-squared: | 0.884 | | | | |
| Method: | Least Squares | F-statistic: | 1516. | | | | |
| Date: | Mon, 07 Jun 2021 | Prob (F-statistic): | 0.00 | | | | |
| Time: | 12:42:49 | Log-Likelihood: | -1299.8 | | | | |
| No. Observations: | 21585 | AIC: | 2820. | | | | |
| Df Residuals: | 21475 | BIC: | 3697. | | | | |
| Df Model: | 109 | | | | | | |
| Covariance Type: | nonrobust | | | | | | |
| Omnibus: | 1609.175 | Durbin-Watson: | 2.002 | | | | |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 6740.814 | | | | |
| Skew: | -0.263 | Prob(JB): | 0.00 | | | | |
| Kurtosis: | 5.687 | Cond. No. | 895. | | | | |

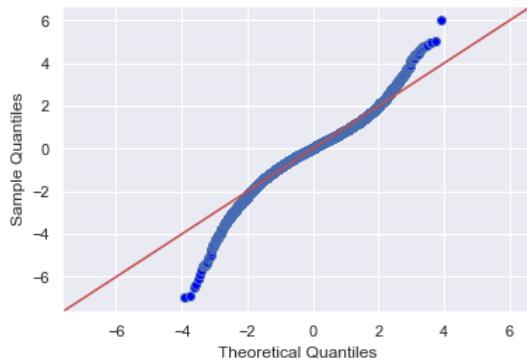
Notes:

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

```
In [73]: print_rmse(df_scaled, target='price_log_scaled', decimals=4)
```

Train RMSE: 0.2563
Test RMSE: 0.2604

```
In [74]: get_qqplot(model_5);
```



Observations:

- No change in the R-squared value from just the log transformed model
- Interpretability is much more challenging for this model
- While worth exploring, this model doesn't provide any additional benefit and will not be used as the final version

Dropping Non-Significant Variables

```
In [75]: nonsig = model_4.pvalues.where(model_4.pvalues > 0.05)  
nonsig.dropna(inplace=True)  
nonsig.index
```

```
Out[75]: Index(['grade_8', 'grade_9', 'grade_10', 'grade_11', 'grade_12',  
       'zipcode_98002', 'zipcode_98003', 'zipcode_98168', 'zipcode_98188',  
       'zipcode_98198', 'month_5', 'month_6', 'month_7', 'month_8', 'month_9',  
       'month_10', 'month_11', 'month_12'],  
      dtype='object')
```

```
In [76]: df_drop_nonsig = df_log.copy()  
df_drop_nonsig.drop(columns=nonsig.index, inplace=True)  
df_drop_nonsig.columns
```

```
Out[76]: Index(['price_log', 'bedrooms_log', 'bathrooms_log', 'sqft_lot_log',  
       'sqft_above_log', 'sqft_basement', 'yr_built_log', 'lat_log', 'long',  
       'sqft_living15_log', 'sqft_lot15_log', 'waterfront_1.0', 'view_1.0',  
       'view_2.0', 'view_3.0', 'view_4.0', 'condition_2', 'condition_3',  
       'condition_4', 'condition_5', 'grade_4', 'grade_5', 'grade_6',  
       'grade_7', 'grade_13', 'zipcode_98004', 'zipcode_98005',  
       'zipcode_98006', 'zipcode_98007', 'zipcode_98008', 'zipcode_98010',  
       'zipcode_98011', 'zipcode_98014', 'zipcode_98019', 'zipcode_98022',  
       'zipcode_98023', 'zipcode_98024', 'zipcode_98027', 'zipcode_98028',  
       'zipcode_98029', 'zipcode_98030', 'zipcode_98031', 'zipcode_98032',  
       'zipcode_98033', 'zipcode_98034', 'zipcode_98038', 'zipcode_98039',  
       'zipcode_98040', 'zipcode_98042', 'zipcode_98045', 'zipcode_98052',  
       'zipcode_98053', 'zipcode_98055', 'zipcode_98056', 'zipcode_98058',  
       'zipcode_98059', 'zipcode_98065', 'zipcode_98070', 'zipcode_98072',  
       'zipcode_98074', 'zipcode_98075', 'zipcode_98077', 'zipcode_98092',  
       'zipcode_98102', 'zipcode_98103', 'zipcode_98105', 'zipcode_98106',  
       'zipcode_98107', 'zipcode_98108', 'zipcode_98109', 'zipcode_98112',  
       'zipcode_98115', 'zipcode_98116', 'zipcode_98117', 'zipcode_98118',  
       'zipcode_98119', 'zipcode_98122', 'zipcode_98125', 'zipcode_98126',  
       'zipcode_98133', 'zipcode_98136', 'zipcode_98144', 'zipcode_98146',  
       'zipcode_98148', 'zipcode_98155', 'zipcode_98166', 'zipcode_98177',  
       'zipcode_98178', 'zipcode_98199', 'month_2', 'month_3', 'month_4'],  
      dtype='object')
```

```
In [77]: model_6 = fit_model(df_drop_nonsig, target='price_log')  
model_6.summary()
```

Out[77]:

| | coef | std err | t | P> t | [0.025 | 0.975] |
|-----------------------|-----------|----------|---------|-------|----------|----------|
| const | -143.2925 | 10.431 | -13.738 | 0.000 | -163.737 | -122.848 |
| bedrooms_log | -0.0612 | 0.006 | -9.918 | 0.000 | -0.073 | -0.049 |
| bathrooms_log | 0.0810 | 0.006 | 13.976 | 0.000 | 0.070 | 0.092 |
| sqft_lot_log | 0.0803 | 0.004 | 21.532 | 0.000 | 0.073 | 0.088 |
| sqft_above_log | 0.4667 | 0.006 | 74.692 | 0.000 | 0.454 | 0.479 |
| sqft_basement | 0.0001 | 3.88e-06 | 37.101 | 0.000 | 0.000 | 0.000 |
| yr_built_log | -0.0874 | 0.141 | -0.620 | 0.535 | -0.364 | 0.189 |

| | | | | | | |
|--------------------------|---------|-------|---------|-------|--------|--------|
| lat_log | 26.7335 | 2.405 | 11.117 | 0.000 | 22.020 | 31.447 |
| long | -0.3881 | 0.051 | -7.657 | 0.000 | -0.487 | -0.289 |
| sqft_living15_log | 0.2023 | 0.007 | 29.868 | 0.000 | 0.189 | 0.216 |
| sqft_lot15_log | -0.0109 | 0.004 | -2.622 | 0.009 | -0.019 | -0.003 |
| waterfront_1.0 | 0.4543 | 0.019 | 23.628 | 0.000 | 0.417 | 0.492 |
| view_1.0 | 0.1090 | 0.011 | 10.270 | 0.000 | 0.088 | 0.130 |
| view_2.0 | 0.1071 | 0.006 | 17.056 | 0.000 | 0.095 | 0.119 |
| view_3.0 | 0.1910 | 0.009 | 21.663 | 0.000 | 0.174 | 0.208 |
| view_4.0 | 0.3151 | 0.013 | 23.650 | 0.000 | 0.289 | 0.341 |
| condition_2 | 0.1797 | 0.038 | 4.764 | 0.000 | 0.106 | 0.254 |
| condition_3 | 0.3242 | 0.035 | 9.234 | 0.000 | 0.255 | 0.393 |
| condition_4 | 0.3569 | 0.035 | 10.158 | 0.000 | 0.288 | 0.426 |
| condition_5 | 0.4273 | 0.035 | 12.088 | 0.000 | 0.358 | 0.497 |
| grade_4 | -0.1563 | 0.037 | -4.251 | 0.000 | -0.228 | -0.084 |
| grade_5 | -0.1820 | 0.013 | -13.562 | 0.000 | -0.208 | -0.156 |
| grade_6 | -0.1210 | 0.006 | -19.195 | 0.000 | -0.133 | -0.109 |
| grade_7 | -0.0782 | 0.004 | -21.245 | 0.000 | -0.085 | -0.071 |
| grade_13 | 0.4342 | 0.052 | 8.273 | 0.000 | 0.331 | 0.537 |
| zipcode_98004 | 1.0180 | 0.018 | 55.902 | 0.000 | 0.982 | 1.054 |
| zipcode_98005 | 0.6163 | 0.021 | 29.027 | 0.000 | 0.575 | 0.658 |
| zipcode_98006 | 0.5848 | 0.016 | 35.798 | 0.000 | 0.553 | 0.617 |
| zipcode_98007 | 0.5502 | 0.023 | 24.386 | 0.000 | 0.506 | 0.594 |
| zipcode_98008 | 0.5492 | 0.020 | 26.967 | 0.000 | 0.509 | 0.589 |
| zipcode_98010 | 0.3121 | 0.024 | 13.036 | 0.000 | 0.265 | 0.359 |
| zipcode_98011 | 0.2125 | 0.025 | 8.442 | 0.000 | 0.163 | 0.262 |
| zipcode_98014 | 0.2324 | 0.034 | 6.914 | 0.000 | 0.166 | 0.298 |
| zipcode_98019 | 0.1582 | 0.031 | 5.174 | 0.000 | 0.098 | 0.218 |
| zipcode_98022 | 0.1776 | 0.020 | 8.742 | 0.000 | 0.138 | 0.217 |
| zipcode_98023 | -0.0551 | 0.011 | -4.990 | 0.000 | -0.077 | -0.033 |
| zipcode_98024 | 0.4015 | 0.031 | 12.849 | 0.000 | 0.340 | 0.463 |
| zipcode_98027 | 0.4800 | 0.019 | 25.483 | 0.000 | 0.443 | 0.517 |
| zipcode_98028 | 0.1620 | 0.023 | 6.927 | 0.000 | 0.116 | 0.208 |
| zipcode_98029 | 0.5777 | 0.022 | 26.229 | 0.000 | 0.534 | 0.621 |
| zipcode_98030 | 0.0309 | 0.014 | 2.278 | 0.023 | 0.004 | 0.057 |
| zipcode_98031 | 0.0448 | 0.014 | 3.310 | 0.001 | 0.018 | 0.071 |
| zipcode_98032 | -0.0483 | 0.017 | -2.777 | 0.005 | -0.082 | -0.014 |
| zipcode_98033 | 0.6336 | 0.020 | 31.419 | 0.000 | 0.594 | 0.673 |
| zipcode_98034 | 0.3494 | 0.021 | 16.553 | 0.000 | 0.308 | 0.391 |
| zipcode_98038 | 0.1983 | 0.016 | 12.523 | 0.000 | 0.167 | 0.229 |
| zipcode_98039 | 1.2049 | 0.030 | 39.615 | 0.000 | 1.145 | 1.264 |
| zipcode_98040 | 0.7909 | 0.016 | 48.014 | 0.000 | 0.759 | 0.823 |
| zipcode_98042 | 0.0740 | 0.013 | 5.816 | 0.000 | 0.049 | 0.099 |
| zipcode_98045 | 0.4110 | 0.031 | 13.270 | 0.000 | 0.350 | 0.472 |
| zipcode_98052 | 0.4962 | 0.021 | 23.266 | 0.000 | 0.454 | 0.538 |
| zipcode_98053 | 0.4587 | 0.025 | 18.589 | 0.000 | 0.410 | 0.507 |
| zipcode_98055 | 0.0896 | 0.014 | 6.354 | 0.000 | 0.062 | 0.117 |
| zipcode_98056 | 0.2469 | 0.014 | 17.372 | 0.000 | 0.219 | 0.275 |
| zipcode_98058 | 0.1303 | 0.013 | 9.831 | 0.000 | 0.104 | 0.156 |
| zipcode_98059 | 0.2814 | 0.015 | 19.302 | 0.000 | 0.253 | 0.310 |
| zipcode_98065 | 0.4150 | 0.027 | 15.328 | 0.000 | 0.362 | 0.468 |
| zipcode_98070 | 0.1315 | 0.021 | 6.336 | 0.000 | 0.091 | 0.172 |

| | | | | | | | | | | |
|--------------------------|----------------------|----------------------------|------------|----------------------|--------|-------|--------|-------|-------|-------|
| | zipcode_98072 | 0.2723 | 0.025 | 10.777 | 0.000 | 0.223 | 0.322 | | | |
| | zipcode_98074 | 0.4885 | 0.022 | 22.349 | 0.000 | 0.446 | 0.531 | | | |
| | zipcode_98075 | 0.5295 | 0.022 | 24.133 | 0.000 | 0.486 | 0.572 | | | |
| | zipcode_98077 | 0.2794 | 0.028 | 10.056 | 0.000 | 0.225 | 0.334 | | | |
| | zipcode_98092 | 0.0416 | 0.013 | 3.240 | 0.001 | 0.016 | 0.067 | | | |
| | zipcode_98102 | 0.8615 | 0.024 | 36.656 | 0.000 | 0.815 | 0.908 | | | |
| | zipcode_98103 | 0.6624 | 0.018 | 36.873 | 0.000 | 0.627 | 0.698 | | | |
| | zipcode_98105 | 0.8059 | 0.020 | 39.738 | 0.000 | 0.766 | 0.846 | | | |
| | zipcode_98106 | 0.2593 | 0.014 | 18.225 | 0.000 | 0.231 | 0.287 | | | |
| | zipcode_98107 | 0.6824 | 0.020 | 34.605 | 0.000 | 0.644 | 0.721 | | | |
| | zipcode_98108 | 0.2724 | 0.017 | 15.897 | 0.000 | 0.239 | 0.306 | | | |
| | zipcode_98109 | 0.8720 | 0.023 | 37.782 | 0.000 | 0.827 | 0.917 | | | |
| | zipcode_98112 | 0.9465 | 0.018 | 51.394 | 0.000 | 0.910 | 0.983 | | | |
| | zipcode_98115 | 0.6581 | 0.018 | 35.735 | 0.000 | 0.622 | 0.694 | | | |
| | zipcode_98116 | 0.6373 | 0.016 | 40.262 | 0.000 | 0.606 | 0.668 | | | |
| | zipcode_98117 | 0.6384 | 0.019 | 34.448 | 0.000 | 0.602 | 0.675 | | | |
| | zipcode_98118 | 0.3878 | 0.013 | 29.390 | 0.000 | 0.362 | 0.414 | | | |
| | zipcode_98119 | 0.8458 | 0.020 | 41.816 | 0.000 | 0.806 | 0.885 | | | |
| | zipcode_98122 | 0.7153 | 0.017 | 41.207 | 0.000 | 0.681 | 0.749 | | | |
| | zipcode_98125 | 0.3582 | 0.020 | 17.626 | 0.000 | 0.318 | 0.398 | | | |
| | zipcode_98126 | 0.4593 | 0.014 | 31.777 | 0.000 | 0.431 | 0.488 | | | |
| | zipcode_98133 | 0.2297 | 0.021 | 11.104 | 0.000 | 0.189 | 0.270 | | | |
| | zipcode_98136 | 0.5716 | 0.016 | 36.609 | 0.000 | 0.541 | 0.602 | | | |
| | zipcode_98144 | 0.5829 | 0.016 | 36.766 | 0.000 | 0.552 | 0.614 | | | |
| | zipcode_98146 | 0.1644 | 0.014 | 11.798 | 0.000 | 0.137 | 0.192 | | | |
| Dep. Variable: | price_log | R-squared: | 0.875 | zipcode_98148 | 0.0745 | 0.025 | 2.923 | 0.003 | 0.025 | 0.125 |
| Model: | OLS | Adj. R-squared: | 0.874 | zipcode_98155 | 0.1849 | 0.022 | 8.483 | 0.000 | 0.142 | 0.228 |
| Method: | Least Squares | F-statistic: | 1653. | zipcode_98166 | 0.2089 | 0.014 | 15.297 | 0.000 | 0.182 | 0.236 |
| Date: | Mon, 07 Jun 2021 | Prob (F-statistic): | 0.00 | zipcode_98177 | 0.3262 | 0.023 | 14.432 | 0.000 | 0.282 | 0.371 |
| Time: | 12:42:49 | Log-Likelihood: | 5659.7 | zipcode_98178 | 0.0811 | 0.014 | 5.618 | 0.000 | 0.053 | 0.109 |
| No. Observations: | 21585 | AIC: | -1.114e+04 | zipcode_98199 | 0.6965 | 0.018 | 37.680 | 0.000 | 0.660 | 0.733 |
| Df Residuals: | 21493 | BIC: | -1.040e+04 | month_2 | 0.0242 | 0.005 | 4.407 | 0.000 | 0.013 | 0.035 |
| Df Model: | 91 | | | month_3 | 0.0512 | 0.005 | 11.215 | 0.000 | 0.042 | 0.060 |
| Covariance Type: | nonrobust | | | month_4 | 0.0696 | 0.004 | 16.463 | 0.000 | 0.061 | 0.078 |
| Omnibus: | 1274.609 | Durbin-Watson: | 2.003 | | | | | | | |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 5004.151 | | | | | | | |
| Skew: | -0.158 | Prob(JB): | 0.00 | | | | | | | |
| Kurtosis: | 5.337 | Cond. No. | 4.42e+06 | | | | | | | |

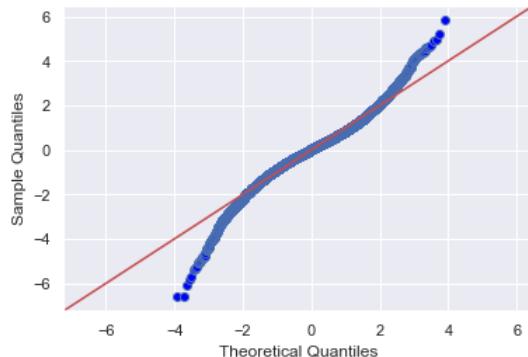
Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 4.42e+06. This might indicate that there are strong multicollinearity or other numerical problems.

```
In [78]: print_rmse(df_drop_nonsig, target='price_log', decimals=4)
```

```
Train RMSE: 0.1859
Test RMSE: 0.1877
```

```
In [79]: get_qqplot(model_6);
```



Observations:

- Small decrease in R-squared
- All but one variable (`yr_built`) are now significant at the alpha = 0.05 level
- RMSE increased slightly

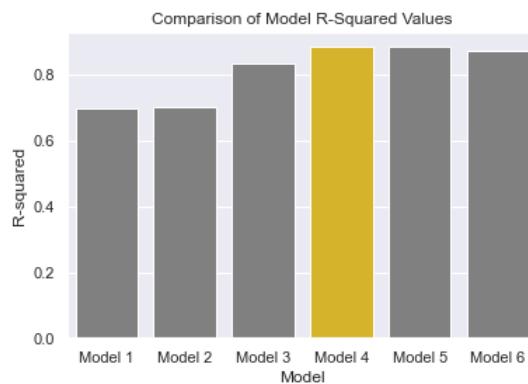
Model Comparisons

```
In [80]: rsquareds = [model.rsquared, model_2.rsquared, model_3.rsquared,
                 model_4.rsquared, model_5.rsquared, model_6.rsquared]

df_comparison = pd.DataFrame({'Model': [f'Model {i}' for i in range(1, 7)],
                               'R-squared': rsquareds})
df_comparison
```

```
Out[80]:   Model  R-squared
0  Model 1    0.699846
1  Model 2    0.700252
2  Model 3    0.835055
3  Model 4    0.885019
4  Model 5    0.885019
5  Model 6    0.874979
```

```
In [81]: sns.barplot(data=df_comparison,
                  x='Model',
                  y='R-squared',
                  palette=['#f1c40f' if m == 'Model 4' else 'gray' for m in df_comparison.Model]
                 )
plt.title('Comparison of Model R-Squared Values')
plt.savefig('images/r2_comp.png', dpi=150, facecolor='white');
```



Conclusion

Results

The fourth model, which removes outliers, includes dummy variables for categorical data, and log transforms continuous data, was the best performing model. This model explains approximately 88.5% of the variations in price for houses in the dataset. Some of the most impactful variables include:

- Being located in zip code 98039 (Medina, WA)
- Having a waterfront property
- Having higher rated `condition` and `grade`

- Being further north (higher latitude)

While not perfect, this model has the potential to be a useful tool for municipalities seeking a better estimate of future tax revenues. Instead of relying on the results of infrequent and costly appraisals for an estimate of taxable value, this model can provide a decently accurate estimate in a short amount of time.

Next Steps

There are many additional ways in which this model can be improved upon over time.

- Further iteration on the model to test for non-additive interactions and various other transformations
- A direct incorporation of an adjustment to the predicted house values to derive the estimated taxable value
- Enhanced location data that includes items such as proximity to amenities and walkability
- Inclusion of macroeconomic variables such as mortgage rates, new constructions, bank lending conditions, etc.