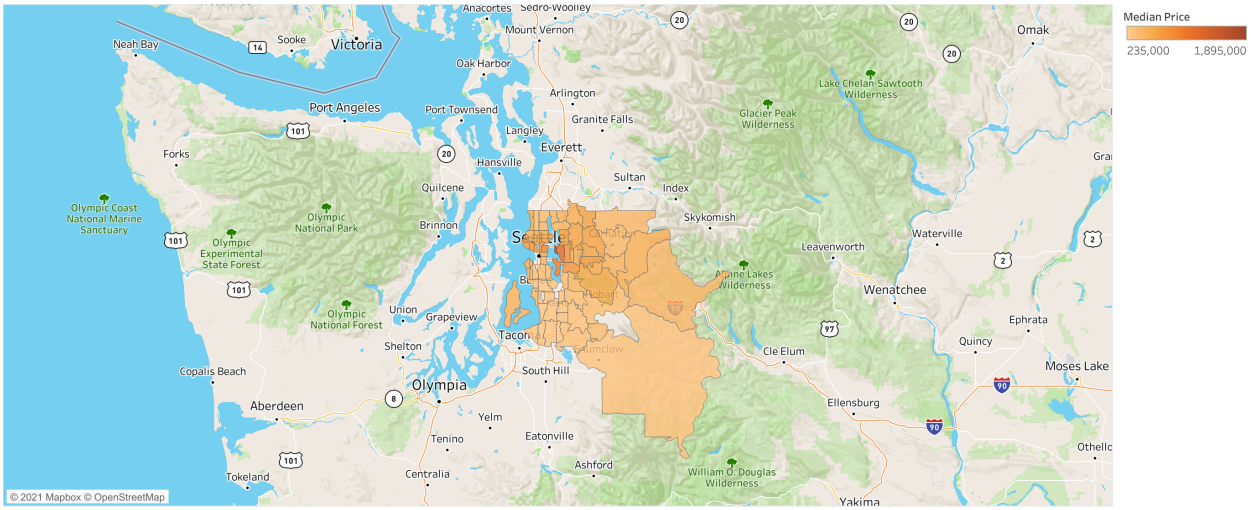
1 King County Housing Characteristics

King County Zip codes by Median Price



Using multiple linear regression analysis models to infer the price of homes based on their existing characteristics.

Business problem:

King County home sales have been increasing as Seattle continues to grow. Top notch labor and a favorable climate make King County a desirable place to live and work. Our real estate team has been tasked with advising clients on the fair value of their home. When our team lists our client's homes, we want to ensure the price is accurate compared to the market.

The model also guides clients on which features to prioritize for increasing home value. This can include renovation, expanding square footage, or other suggestions within the owners control.

1.1 Data

- 21,597 rows by 21 columns
- CSV Formatted

1.2 Roadmap

- Scrub data to handle null values and duplicates
- Add additional features to better infer the price of home based on existing characteristics
- Check for linearity and multicollinearity to make sure that model meets relevant assumptions
- Perform outlier removal methods to better meet the assumptions of the linear regression model
- Use One Hot Encoding to handle categorical variables
- Provide accompanying visualizations to support and interpret the findings of the model

- Circle back to how the multiple linear regression model supports the recommendations for how prices can be predicted based on existing characteristics

2 Scrub Data

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
pd.set_option('display.max_columns', None)

df = pd.read_csv('data/kc_house_data.csv')
df.head()
```

Out[1]:

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

2.1 Descriptions of columns

- **id** - unique identified for a house
- **dateDate** - house was sold
- **pricePrice** - is prediction target
- **bedroomsNumber** - of Bedrooms/House
- **bathroomsNumber** - of bathrooms/bedrooms
- **sqft_livingsquare** - footage of the home
- **sqft_lotsquare** - footage of the lot
- **floorsTotal** - floors (levels) in house
- **waterfront** - House which has a view to a waterfront
- **view** - Quality of view
- **condition** - How good the condition is (Overall)
- **grade** - overall grade given to the housing unit, based on King County grading system
- **sqft_above** - square footage of house apart from basement
- **sqft_basement** - square footage of the basement
- **yr_built** - Built Year
- **yr_renovated** - Year when house was renovated
- **zipcode** - zip
- **lat** - Latitude coordinate
- **long** - Longitude coordinate
- **sqft_living15** - The square footage of interior housing living space for the nearest 15 neighbors
- **sqft_lot15** - The square footage of the land lots of the nearest 15 neighbors

In [2]:
```
# Evaluating if type matches column description

pd.set_option('display.float_format', lambda x: '%.2f' % x)
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
```

- Date should be a datetime object
- Sqft basement should be an integer, not object

In [3]:
```
# Make date into datetime object

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

In [4]:
```
# Observe summary statistics

df.describe()
```

Out[4]:

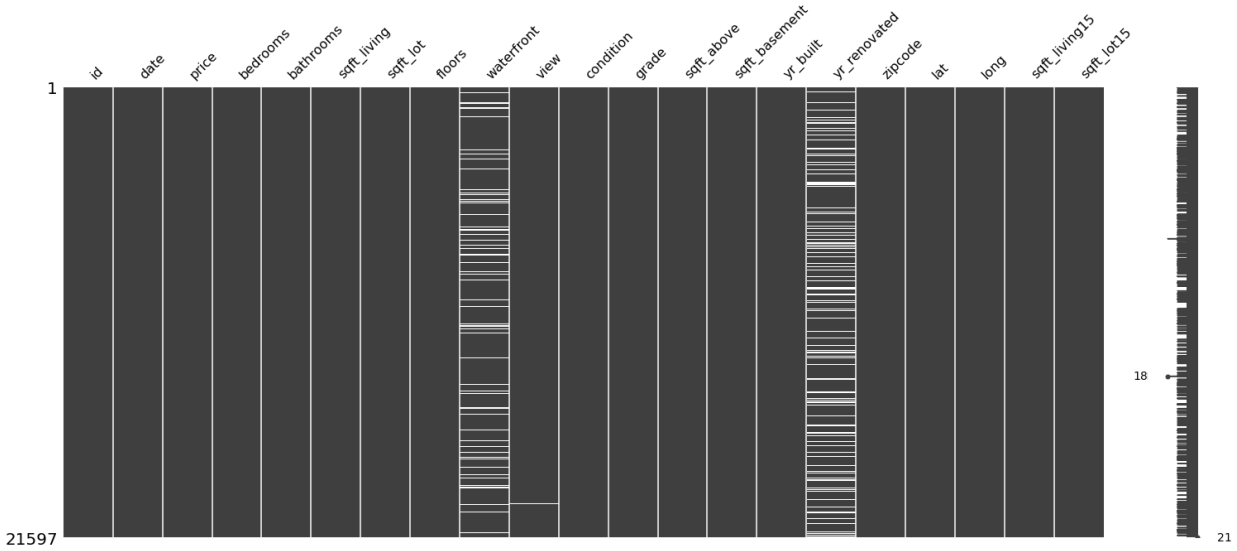
	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfro
count	21597.00	21597.00	21597.00	21597.00	21597.00	21597.00	21597.00	19221.
mean	4580474287.77	540296.57	3.37	2.12	2080.32	15099.41	1.49	0.
std	2876735715.75	367368.14	0.93	0.77	918.11	41412.64	0.54	0.
min	1000102.00	78000.00	1.00	0.50	370.00	520.00	1.00	0.
25%	2123049175.00	322000.00	3.00	1.75	1430.00	5040.00	1.00	0.
50%	3904930410.00	450000.00	3.00	2.25	1910.00	7618.00	1.50	0.
75%	7308900490.00	645000.00	4.00	2.50	2550.00	10685.00	2.00	0.
max	9900000190.00	7700000.00	33.00	8.00	13540.00	1651359.00	3.50	1.

- ID is a random value so should not be evaluated as a continuous variable
- Price has a large standard deviation and most likely contains outlier values
- Waterfront is a binary variable
- Floors, view, condition, and grade are discrete variables
- Zipcode, latitude, and longitude are not continuous variables

2.2 Handling Null Values

In [5]:
```
# Visualize which columns contain null values

import missingno
missingno.matrix(df)
```

Out[5]: <AxesSubplot:>



Waterfront, view, and yr_renovated contain null values

In [6]:
```
# Check how many null values are in each column

null = df.isna().sum()
null[null>1]
```

Out[6]:
```
waterfront      2376
view              63
yr_renovated    3842
dtype: int64
```

In [7]:
```
# Create formulat to impute null values with probability that they appe

def impute_cat(df, col):
    '''
    Impute null value with value based on liklihood
    of occurring in the original column
    '''
    val_prob = dict(df[col].value_counts(1))
    prob = list(val_prob.values())
    val = list(val_prob.keys())
    np.random.choice(val, p=prob)
    df[col].fillna(np.random.choice(val, p=prob), inplace=True)
    return df
```

2.2.1 Fill in missing Values for 'view' column

Interpreting 'view' as quality of the view from the home. For example, a 4 would be a stunning view, maybe of the mountains or a lake. Can be a beautiful view of nature or the urban environmnt. A view of 0 would be described as highly undesirable, like looking directly into a neighbor's property, or an unappealing natural characteristic.

In [8]:
```
print('Value Counts Normalized')
print(df['view'].value_counts(1, dropna=False))
print('------------------------')
print('Value Counts Absolute')
print(df['view'].value_counts(dropna=False))
```

```
Value Counts Normalized
0.00   0.90
2.00   0.04
3.00   0.02
1.00   0.02
4.00   0.01
nan    0.00
Name: view, dtype: float64
------------------------
Value Counts Absolute
0.00    19422
2.00      957
3.00      508
1.00      330
4.00      317
nan        63
Name: view, dtype: int64
```

```
I have gone ahead and made the assumption that a **nan** represents a value that is missing completely at random. I will use impute_cat to insert a value between 0-4 based on the probability of the original variable's distribution
- 90% chance of imputing a 0
- 2% chance of imputing a 1
- 4% of imputing a 2
- 2% of imputing a 3
- 1% of imputing a 4
```

In [9]:
```
1 impute_cat(df, 'view')
```

Out[9]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
0	7129300520	2014-10-13	221900.00	3	1.00	1180	5650	1.00	nan
1	6414100192	2014-12-09	538000.00	3	2.25	2570	7242	2.00	0.00
2	5631500400	2015-02-25	180000.00	2	1.00	770	10000	1.00	0.00
3	2487200875	2014-12-09	604000.00	4	3.00	1960	5000	1.00	0.00
4	1954400510	2015-02-18	510000.00	3	2.00	1680	8080	1.00	0.00
...	...	...	...	...	...	...	...	...	...
21592	263000018	2014-05-21	360000.00	3	2.50	1530	1131	3.00	0.00
21593	6600060120	2015-02-23	400000.00	4	2.50	2310	5813	2.00	0.00
21594	1523300141	2014-06-23	402101.00	2	0.75	1020	1350	2.00	0.00
21595	291310100	2015-01-16	400000.00	3	2.50	1600	2388	2.00	nan
21596	1523300157	2014-10-15	325000.00	2	0.75	1020	1076	2.00	0.00

21597 rows × 21 columns

In [10]:
```
# Confirm that there are no more null values in the view column

df.isna().sum()
```

Out[10]:
```
id                0
date              0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront     2376
view              0
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
```

2.2.2 Fill in missing Values for 'yr_renovated' column

Describes when the home was most recently renovated

In [11]:
```
print('Value Counts Normalized')
print(df['yr_renovated'].value_counts(1, dropna=False))
print('------------------------')
print('Value Counts Absolute')
print(df['yr_renovated'].value_counts(dropna=False))
```

```
Value Counts Normalized
0.00      0.79
nan       0.18
2014.00   0.00
2003.00   0.00
2013.00   0.00
          ...
1944.00   0.00
1948.00   0.00
1976.00   0.00
1934.00   0.00
1953.00   0.00
Name: yr_renovated, Length: 71, dtype: float64
------------------------
Value Counts Absolute
0.00       17011
nan         3842
2014.00       73
2003.00       31
2013.00       31
           ...
1944.00        1
1948.00        1
1976.00        1
1934.00        1
1953.00        1
Name: yr_renovated, Length: 71, dtype: int64
```

Most of the values in yr_renovated are either 0 or nan. When performing imputation going to simply fill in a 0 for missing value as opposed to imputing the year based on probability.

Since this value has ~79% 0 values, I will most likely feature engineer it to be a binary variable. Either the home has been renovated or it has not

In [12]:
```
df[df['yr_renovated']>0].describe()
```

Out[12]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
count	744.00	744.00	744.00	744.00	744.00	744.00	744.00	652.00
mean	4418716401.67	768901.89	3.46	2.31	2327.38	16215.53	1.50	0.04
std	2908265353.00	627125.79	1.07	0.90	1089.00	38235.31	0.49	0.21
min	3600057.00	110000.00	1.00	0.75	520.00	1024.00	1.00	0.00
25%	1922984893.00	412250.00	3.00	1.75	1560.00	5000.00	1.00	0.00
50%	3899100167.50	607502.00	3.00	2.25	2200.00	7375.00	1.50	0.00
75%	7014200237.50	900000.00	4.00	2.75	2872.50	12670.75	2.00	0.00
max	9829200250.00	7700000.00	11.00	8.00	12050.00	478288.00	3.00	1.00

The most recent renovation took place in 2015. The oldest recorded renovation was in 1934

In [13]:
```
# Going to assume that a null value means that the home has never been
# This is equivalent to a 0 which is why I am filling null values with

df['yr_renovated'].fillna(0, inplace=True)
```

In [14]:
```
# Confirm that there are no more null values in the yr_renovated column

df.isna().sum()
```

Out[14]:
```
id                0
date              0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront     2376
view              0
condition         0
grade             0
sqft_above        0
sqft_basement     0
yr_built          0
yr_renovated      0
zipcode           0
lat               0
long              0
sqft_living15     0
sqft_lot15        0
dtype: int64
```

2.2.3 Fill in missing Values for 'waterfront' column

Waterfront is a binary variable. 1 means the home has a view of the water. 0 means the home does not have a view of the water

- Based on national home prices, waterfront properties tend to be more expensive than non-waterfront homes. People appreciate the view of the ocean, or a lake, and enjoy easy access to bodies of water especially during the summer
- I'd like to explore if homes prices at greater than $1,000,000 are more likely to have a waterfront view
- I can then use this finding to subset the data based on a home prices threshold and impute the missing waterfront variables with more accuracy

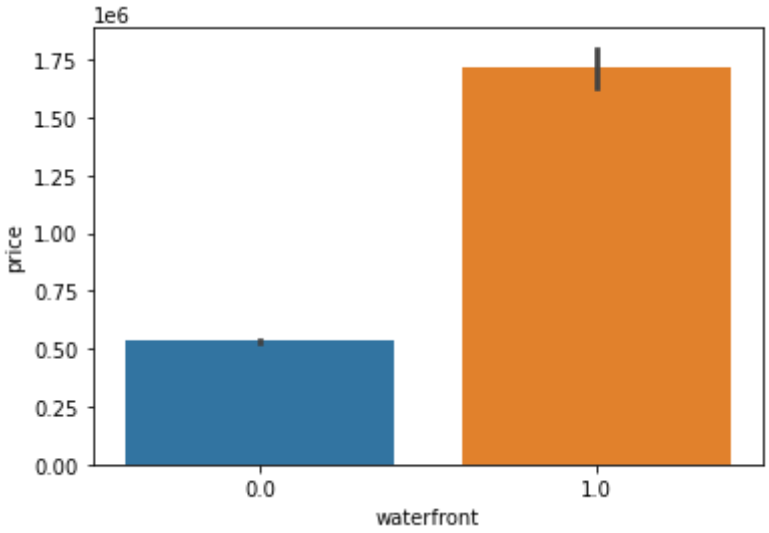
In [15]:
```
# Check if mean of price is greater for homes with waterfront views

df.groupby('waterfront')[['price', 'view']].mean()
```

Out[15]:

waterfront	price	view
0.00	532641.99	0.20
1.00	1717214.73	3.76

As expected, waterfront homes have a greater mean price than non-waterfront homes. Their view score is also much higher because the proximity to water. It is easier to see the ocean, lake, or river which enhances the view score as it is a desirable natural feature

In [16]:
```
import seaborn as sns

sns.barplot(data=df, x='waterfront', y='price', ci=68);
```



Clearly, waterfront homes are more expensive than non-waterfront homes

In [17]:
```
mplus_water =len(df[(df['price']>1000000) & df['waterfront']==1.0])
print(f'Number of houses over $1,000,000 with waterfront view:\t{mplus_
mminus_water = len(df[(df['price']<1000000) & df['waterfront']==1.0])
print(f'Number of houses under $1,000,000 with waterfront view:\t{mminu
```

```
Number of houses over $1,000,000 with waterfront view:  96
Number of houses under $1,000,000 with waterfront view: 49
```

In [18]:
```
# Observe ratio of waterfront homes for homes over $1,000,000 and those

pd.set_option('display.float_format', lambda x: '%.5f' % x)

print('$1M+ with waterfront');
print(df.loc[df['price']>1000000]['waterfront'].value_counts(1));
print('----------------------------------------')
# Prob of having waterfront view for homes under $1,000,000
print('$1M- with waterfront');
print(df.loc[df['price']<1000000]['waterfront'].value_counts(1))
```

```
$1M+ with waterfront
0.00000   0.92666
1.00000   0.07334
Name: waterfront, dtype: float64
----------------------------------------
$1M- with waterfront
0.00000   0.99726
1.00000   0.00274
Name: waterfront, dtype: float64
```

- 7.3% of homes priced over $1 million have waterfront views
- 0.02% of homes priced under $1 million have waterfront views
- As a result, I am going to subset the data by a $1 million threshold limit and then impute the missing waterfront value. The reason I am doing this is because the more expensive homes are far more likely to have waterfront views and I don't want them to have the same probability of being assigned a waterfront view.

In [19]:
```
# Subset the data into two slices based on $1 million threshold

df_1mplus=df.loc[df['price']>1000000]
df_1mminus=df.loc[df['price']<1000000]
```

In [20]:
```
# Use impute_cat on homes over $1,000,000

df_1mplus =impute_cat(df_1mplus, 'waterfront')
```

```
/Users/ethankunin/opt/anaconda3/envs/learn-env/lib/python3.8/site-package
s/pandas/core/series.py:4517: 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-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().fillna(
```

In [21]:
```
# Simply fill the missing waterfront values with 0 for homes under $1,0
# probability of them having a waterfront view is far lower

df_1mminus['waterfront'] =df_1mminus['waterfront'].fillna(0)
```

```
<ipython-input-21-3a74373b56b6>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  df_1mminus['waterfront'] =df_1mminus['waterfront'].fillna(0)
```

In [22]:
```
# Join the data back together

df=pd.concat([df_1mminus, df_1mplus])
```

In [23]:
```
# Confirm there are no more missing values

df.isna().sum()
```

Out[23]:
```
id               0
date             0
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
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

Missing values are handled, next step is to address duplicate values

2.3 Handling Duplicates

In [24]:
```
df[df.duplicated()]
```

Out[24]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	gr

Initially, it appears that we don't have any duplicates in the dataset. However, df.duplicated() only returns duplicate values if **all** columns in the row are matching. It may be produnt to check if there are any duplicates that appear in the 'id' column

In [25]:
```
# Check duplicats by 'id'
display(len(df))
df[df.duplicated(subset=['id'],keep=False)]

```

21565

Out[25]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wate
93	6021501535	2014-07-25	430000.00000	3	1.50000	1580	5000	1.00000	0.
94	6021501535	2014-12-23	700000.00000	3	1.50000	1580	5000	1.00000	0.
324	7520000520	2014-09-05	232000.00000	2	1.00000	1240	12092	1.00000	0.
325	7520000520	2015-03-11	240500.00000	2	1.00000	1240	12092	1.00000	0.
345	3969300030	2014-07-23	165000.00000	4	1.00000	1000	7134	1.00000	0.
...	...	...	...	...	...	...	...	...	
14294	3528000040	2014-10-01	1690000.00000	3	3.25000	5290	224442	2.00000	0.
14295	3528000040	2015-03-26	1800000.00000	3	3.25000	5290	224442	2.00000	0.
15999	5536100020	2015-05-12	1190000.00000	3	2.00000	2160	15788	1.00000	0.
18976	7856400300	2014-07-02	1410000.00000	2	2.50000	3180	9400	2.00000	0.
18977	7856400300	2015-03-22	1510000.00000	2	2.50000	3180	9400	2.00000	0.

353 rows × 21 columns

Here, we see that when we check for duplicates by 'id' we do have duplicate rows. The only difference between the first and last duplicate row is date and price. This leads me to believe that the duplicate is shown because there was a sale. As a result, I am only going to keep the 'last' value because I want to reflect the most recent change of value and accurate price.

In [26]:
```
df=df.drop_duplicates(subset=['id'], keep='last')
print(len(df))
```

21388

Our dataset went from 21,565 observations to 21,388. We removed 177 duplicate values.

In [27]:
```
# Confirm that there are no more duplicate observations in the dataset

df.duplicated(subset=['id'],keep=False).sum()
```

Out[27]: 0

3 Exploratory Data Analysis

- Explore the distribution of each variable and their relationship with price
- Determine if variables are discrete or continuious
- Determine if variables are categoric or numeric
- Check if their skew in the distribution or normal
- Check if their are outlier values and if they appear above the median or below the median

In [28]:
```
def distr_(df, col):
    """
    Produces a boxplot, scatterplot, and histogram/kde
    Produces summary statistics
    """
    fig, ax = plt.subplots(figsize=(8,7), nrows=3, gridspec_kw={'height
    mean=df[col].mean()
    median=df[col].median()
    max_=df[col].max()
    min_=df[col].min()
    std_=df[col].std()
    sns.histplot(df[col],alpha=0.5,stat='density',ax=ax[0]);
    sns.kdeplot(df[col],color='green',ax=ax[0]);
    ax[0].set_xlabel(col)
    ax[0].set_title(f'{col} Distribution')
    ax[0].axvline(mean, label=f'Mean: {mean}', c='red')
    ax[0].axvline(median, label=f'median: {median}', c='red', linestyle
    ax[0].legend()

    sns.boxplot(data=df, x=col, ax=ax[1]);

    sns.scatterplot(data=df, x=df[col], y=df['price']);

    fig.tight_layout();
    print(f'{col.capitalize()} Summary')
    print(f'Median: {median}')
    print(f'Mean: {mean:.4}')
    print(f'Max: {max_}')
    print(f'Min: {min_}')
    print(f'Std: {std_:.4}')
    plt.show()
```

3.1 Handling Error in Basement encoding

When initially running the EDA check, sqft_basement throws an eror so going to further inspect what is causing the error

In [29]:
```
# Check values

df['sqft_basement'].value_counts(0)
```

Out[29]:
```
0.0       12701
?           452
600.0       215
500.0       205
700.0       205
           ...
2130.0        1
861.0         1
3480.0        1
4130.0        1
516.0         1
Name: sqft_basement, Length: 304, dtype: int64
```

452 values have a question mark. I am going to assume that a question mark means the data is missing. Since the question mark only represents ~2% of the data, changing it to 0 will not alter the original distribution of the data. Additionally, since there are a high number of 0 values ~59% of the data, I am going to turn sqft_basement into a binary variable. 0 for no basement, 1 for basement is present. Otherwise, the standard deviation will be very large and it will be difficult to interpret

In [30]:
```
# Replace the question mark with a zero

df['sqft_basement'].replace(to_replace='?', value='0.0',inplace=True)
```

In [31]:
```
df['sqft_basement'].value_counts(1)
```

Out[31]:
```
0.0      0.61497
600.0    0.01005
700.0    0.00958
500.0    0.00958
800.0    0.00940
           ...
2130.0   0.00005
861.0    0.00005
3480.0   0.00005
4130.0   0.00005
516.0    0.00005
Name: sqft_basement, Length: 303, dtype: float64
```

Question mark is gone. 61% of homes do not have a basement

In [32]:
```
# Use .map to make sqft_basement into a binary variable

df['basementyes'] = (df['sqft_basement']>'0.0').map({True:1,
                                                     False: 0})
```

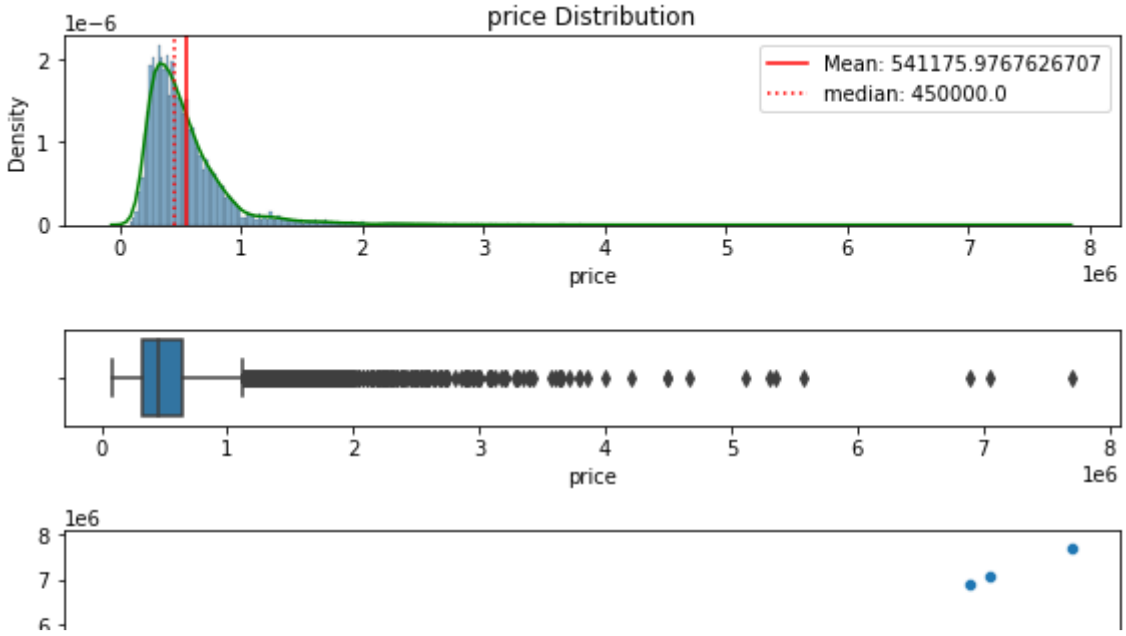
In [33]:
```
# Confirm that original sqft_basement and new column have equal distrib

df['basementyes'].value_counts(1)
```

Out[33]:
```
0    0.61497
1    0.38503
Name: basementyes, dtype: float64
```

In [34]:
```
# Drop sqft_basement because we are replacing it with a binary reprsent
# or it does not)

df=df.drop('sqft_basement', axis=1)
```

3.2 Return to checking distributions, outliers, and relationship with price

In [35]:
```
# Checking all variables except for id, zipcode, latitude, and longitud

eda_check = ['price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15', 'basementyes']
for col in eda_check:
    print(distr_(df, col))
    print('----------------------------------------------------------------
```



3.2.1 Individual EDA Analysis

1. Price
   A. **Distribution**: Binomial, Right skewed
   B. **Outliers**: Outliers upper IQR threshold
   C. **Relationship with price**: NA

2. Bedrooms
    A. **Distribution**: Bimodal
    B. **Outliers**: Outliers upper IQR threshold. Extreme outlier at 33 that should most likely be removed
    C. **Relationship with price**: Linear until 5/6 bedrooms
    D. **Discrete or Continuous**: Discrete-Possibly ordinal
3. Bathrooms
    A. **Distribution**: Bimodal
    B. **Outliers**: Outliers upper IQR threshold.
    C. **Relationship with price**: Linear
    D. **Discrete or Continuous**: Discrete-Ordinal
4. Sqft_living
    A. **Distribution**: Bimodal, skewed
    B. **Outliers**: Outliers upper IQR threshold.
    C. **Relationship with price**: Linear
    D. **Discrete or Continuous**: Continuous
5. Sqft_lot
    A. **Distribution**: Binomial, Right skewed
    B. **Outliers**: Outliers upper IQR threshold.
    C. **Relationship with price**: Non-Linear
    D. **Discrete or Continuous**: Continuous
6. Floors
    A. **Distribution**: Bimodal, Right skewed
    B. **Outliers**: None
    C. **Relationship with price**: Non-Linear-Non Ordinal
    D. **Discrete or Continuous**: Discrete
7. Waterfront
    A. **Distribution**: Bernoulli
    B. **Outliers**: None
    C. **Relationship with price**: Unclear
    D. **Discrete or Continuous**: Discrete - Binary
8. View
    A. **Distribution**: Bernoulli
    B. **Outliers**: None
    C. **Relationship with price**: Unclear
    D. **Discrete or Continuous**: Discrete
9. Condition
    A. **Distribution**: Bernoulli
    B. **Outliers**: None
    C. **Relationship with price**: Unclear. Seems to increase until 3 and then move down
    D. **Discrete or Continuous**: Discrete
10. Grade
    A. **Distribution**: Bernoulli
    B. **Outliers**: None
    C. **Relationship with price**: Linear
    D. **Discrete or Continuous**: Discrete
11. Sqft_above
    A. **Distribution**: Right skewed
    B. **Outliers**: Outliers upper IQR threshold.

   C. **Relationship with price**: Linear
   D. **Discrete or Continuous**: Continuous
12. Yr_Built
   A. **Distribution**: Left skewed
   B. **Outliers**: Outliers upper IQR threshold.
   C. **Relationship with price**: None
   D. **Discrete or Continuous**: Continuous
13. Yr_Renovated
   A. **Distribution**: Bernoulli
   B. **Outliers**: None
   C. **Relationship with price**: None
   D. **Discrete or Continuous**: Continuous
14. Sqft_living15
   A. **Distribution**: Right skewed
   B. **Outliers**: Outliers upper IQR threshold.
   C. **Relationship with price**: Linear
   D. **Discrete or Continuous**: Continuous
15. Sqft_lot15
   A. **Distribution**: Right skewed
   B. **Outliers**: Outliers upper IQR threshold.
   C. **Relationship with price**: Linear
   D. **Discrete or Continuous**: Continuous
16. Basementyes
   A. **Distribution**: Binary
   B. **Outliers**: None
   C. **Relationship with price**: Unclear
   D. **Discrete or Continuous**: Discrete-Binary

3.2.2 Overall EDA Analysis

- Most of the continuous variables are right skeweed
- Supported by the distribution and mean being greater than the median
- High outlier values on the upper IQR threshold
- Yr_renovated has a lot of 0 values so may be improved by turning into a binary variable
- Bedrooms has a mistaken entry (33 bedrooms)

3.2.2.1 Handle Bedroom error

In [36]:
```
# Find the observation where bedroom is 33

df[df['bedrooms']>20]
```

Out[36]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	water
15856	2402100895	2014-06-25	640000.00000	33	1.75000	1620	6000	1.00000	0.0

Appears to be a mistake because there is only 1.75 bathrooms and sqft living is only 1620. Going to drop column

In [37]:
```
df.drop(index=15856, inplace=True)
```

In [38]:
```
# Confirm it has been removed

df[df['bedrooms']>20]
```

Out[38]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	gr

3.2.3 Turn yr_renovated into binary values

- High amount of zero values, binary encoding will be a better indicator of relationship with price
- Otherwise, the standard deviation will be very high and it will be difficult to effectively interpret

In [39]:
```
df['yr_renovated'].value_counts(1)
```

Out[39]:
```
0.00000       0.96549
2014.00000    0.00341
2003.00000    0.00145
2013.00000    0.00145
2007.00000    0.00140
                ...
1934.00000    0.00005
1971.00000    0.00005
1954.00000    0.00005
1950.00000    0.00005
1944.00000    0.00005
Name: yr_renovated, Length: 70, dtype: float64
```

Aprox. 97% of values suggest the home has not been renovated. Safe to encode as has been renovated or has not been renovated

In [40]:
```
df['renovated_yes'] = (df['yr_renovated']>0).map({True:1,
                                                  False: 0})

```

In [41]:
```
# Confirm distribution has not changed

df['renovated_yes'].value_counts(1)
```

Out[41]:
```
0   0.96549
1   0.03451
Name: renovated_yes, dtype: float64
```

In [42]:
```
# Dropping yr_renovated because we are replacing it with a binary varia

df.drop('yr_renovated', axis=1, inplace=True)
```

4 Feature Engineering

- Explore adding additional predictor values to the model so that it can more accurately predict price
- Feature engineering allows for us to broaden our predictor values then what we are limited to with the original dataset
- Be cautious of multicollinearity because features will be engineering by transforming existing columns

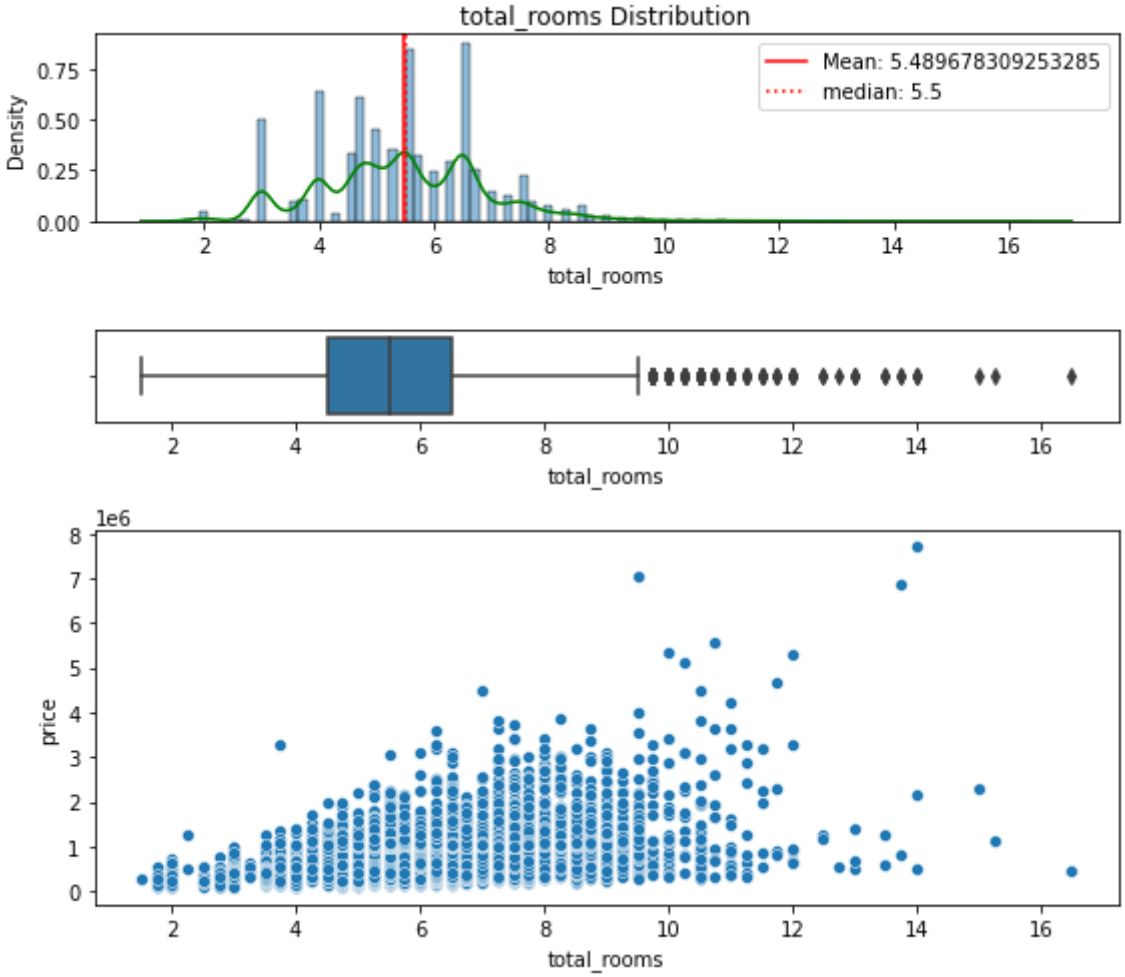
4.1 Total Rooms

- Add bedrooms and bathrooms to create new column called total_rooms
- This will provide us a summary of the number of rooms in the home

In [43]:
```
# Combine bedrooms and bathrooms

df['total_rooms'] = df['bedrooms']+df['bathrooms']
```

In [44]:
```
1 distr_(df, 'total_rooms')
```

```
Total_rooms Summary
Median: 5.5
Mean: 5.49
Max: 16.5
Min: 1.5
Std: 1.463
```



```
- Distribution is not normal
- Looks like there are a significant number of outliers to the right of the upper IQR threshold
- Initially looks like there is a positive linear relationship with price
```

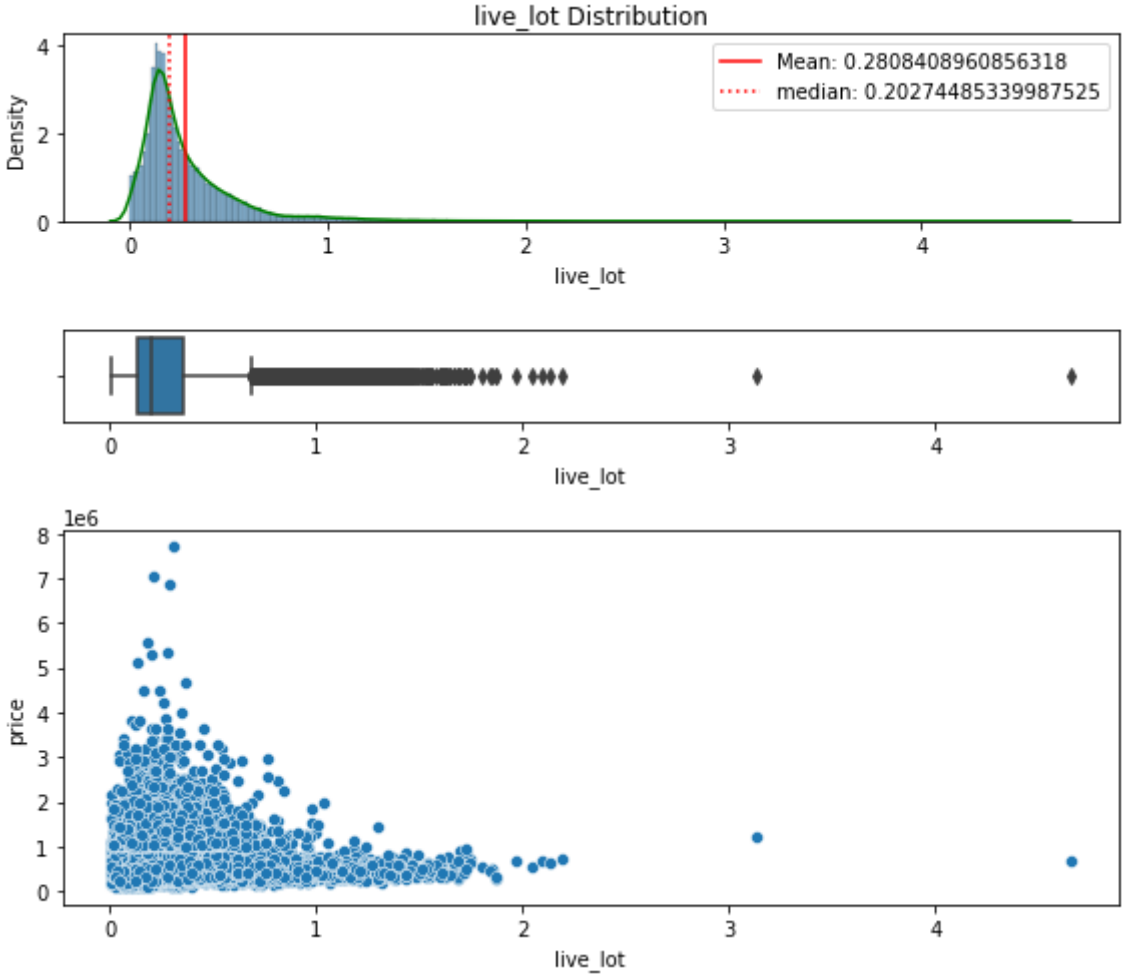
4.2 Backyard Size as a proportion of the total lot

- Use sqft_above/sqft_lot as a proxy for backyard size
- Essentially, we are capturing how big the home is compared to the lot
- A larger value means a relatively smaller backyard and vice versa

In [45]:
```
# Divide sqft_living/sqft_lot
# Using sqft_above as opposed to sqft_living because living includes ba

df['live_lot'] = df['sqft_above']/df['sqft_lot']
```

In [46]:
```
1 distr_(df, 'live_lot')
```

```
Live_lot Summary
Median: 0.20274485339987525
Mean: 0.2808
Max: 4.653846153846154
Min: 0.0006095498431482305
Std: 0.2426
```



- Data is right skewed
- Significant number of outliers to the right of the upper IQR threshold
- Values may be greater than 1 because the living space can be greater than the lot space in the event that a house is built on a small lot and has many floors. In other words, it has a lot of sqft footage on the inside because it goes up vertically

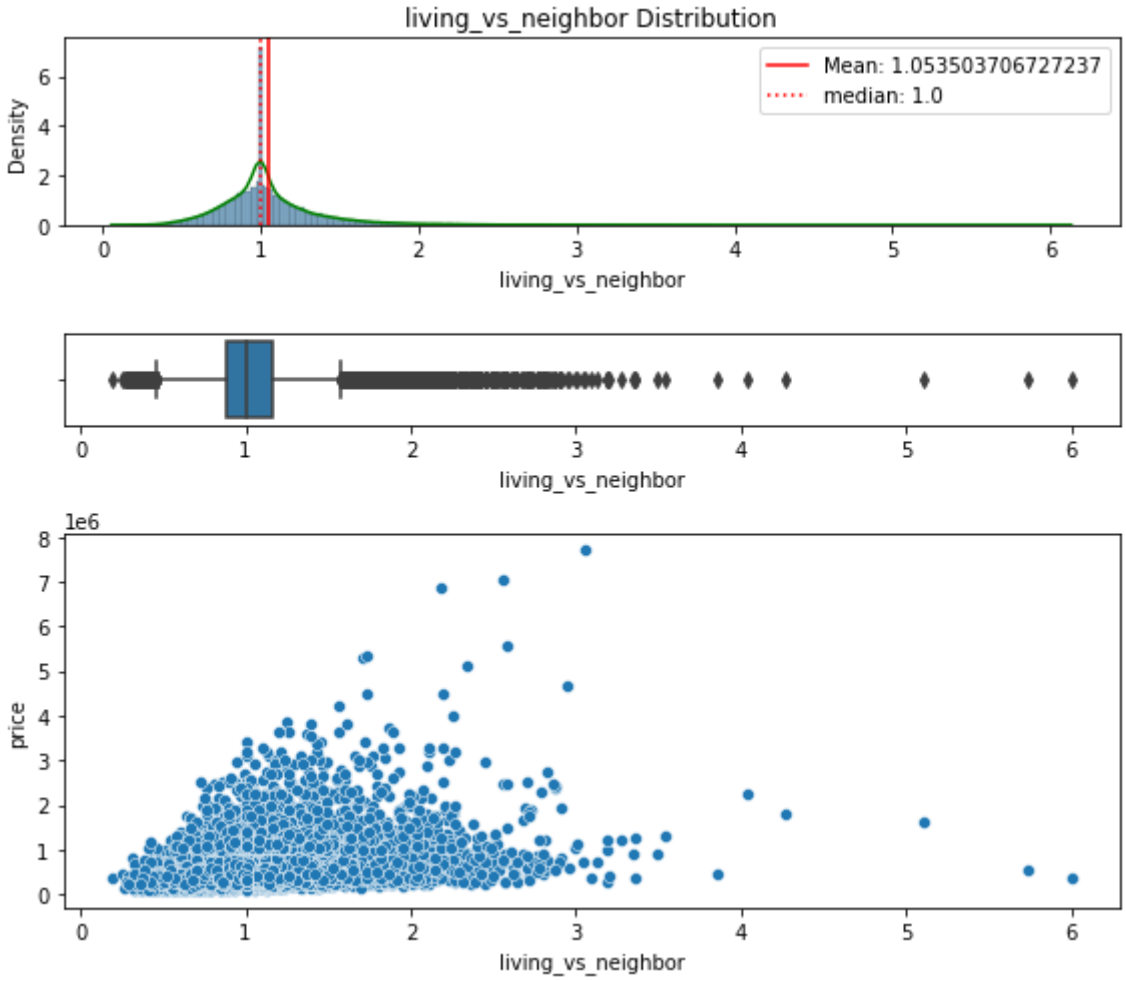
4.3 Comparison of Square Foot living and Lot vs Neighbors

- Sqft_living15 represents the average living space of the 15 nearest neighbors
- Would like to compare how the living space of the observed home compares to the neighbors

In [47]:
```
# SQF_living compared to neighbors
df['living_vs_neighbor'] = df['sqft_living']/df['sqft_living15']
```

In [48]:
```
distr_(df, 'living_vs_neighbor')
```

```
Living_vs_neighbor Summary
Median: 1.0
Mean: 1.054
Max: 6.0
Min: 0.1872791519434629
Std: 0.3203
```



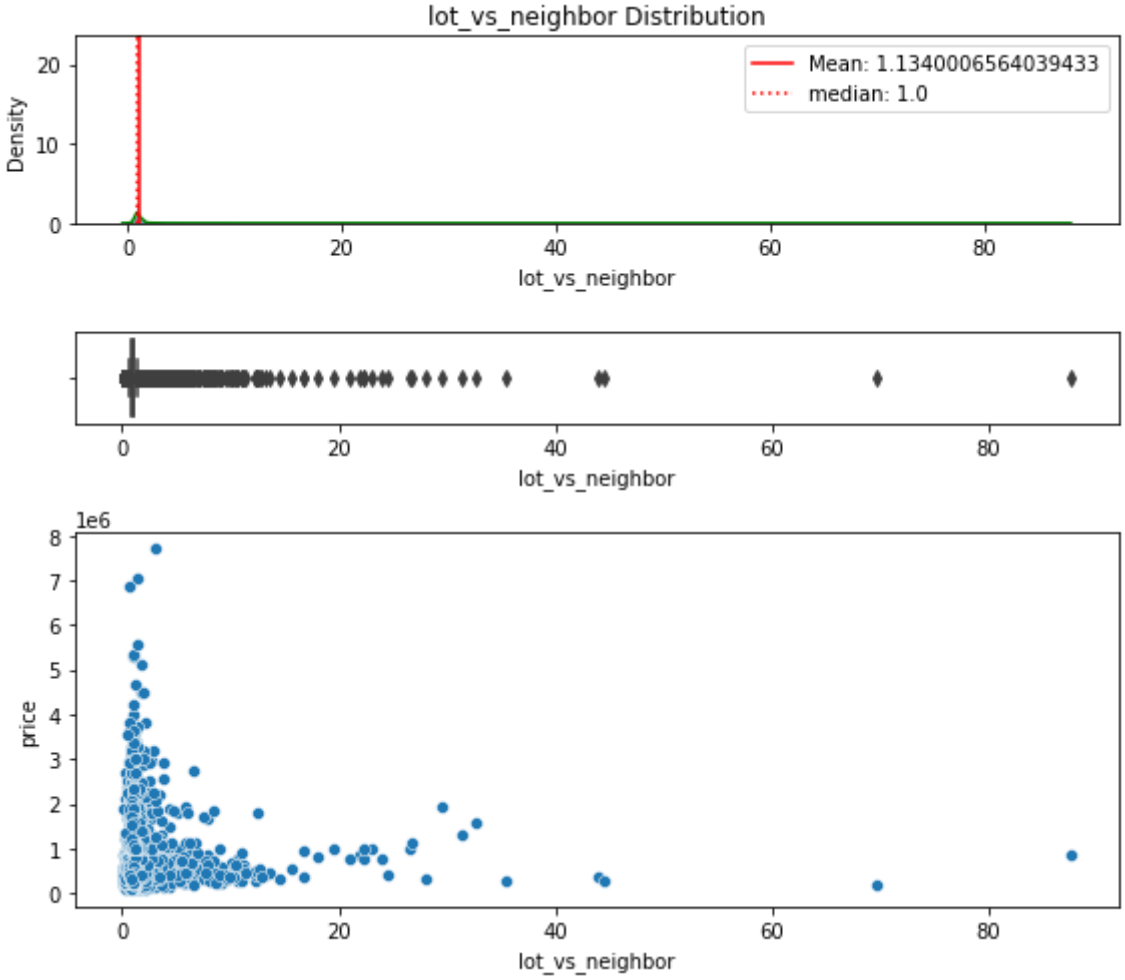

```
- Data looks pretty normally distributed because mean and median are very close
- Outliers to the right of the upper threshold
- Unclear if there is a linear relationship with price
- A lot of the much larger homes in comparison to the neighbors actually have a negative relationship with price
```

- Sqft_lot15 represents the average living space of the 15 nearest neighbors
- Would like to compare how the lot size of the observed home compares to the neighbors

In [49]:
```
df['lot_vs_neighbor'] = df['sqft_lot']/df['sqft_lot15']
```

In [50]:
```
1 distr_(df, 'lot_vs_neighbor')
```

```
Lot_vs_neighbor Summary
Median: 1.0
Mean: 1.134
Max: 87.52717948717948
Min: 0.054971997700810314
Std: 1.286
```



- Hard to tell distribution because there are clearly significant outlier values
- There are values greater than 80 which seem unreasonable unless there is a neighborhood with apartments and one extremely large home
- Linearity seems unlinkely because as lot_vs_neighbor increases, the change in price is not constant

5 Check Assumptions of Linearity and Multicollinearity

- For the model to provide acccurate inferences, it must meet assumptions of linearity and multicollinearity

5.1 Check Assumption of Linearity

- There must be a linear relationship between the predictor variable and target variable
  - In our case, the predictor variable refers to the home features and the target refers to price
- By linear relationship, we mean that as the x-value increases, the y-value must change by a constant amount
- If we do not meet assumption of linearity, our model will not accuratly infer home prices
- Must check each predictor that we are we going to include in the model

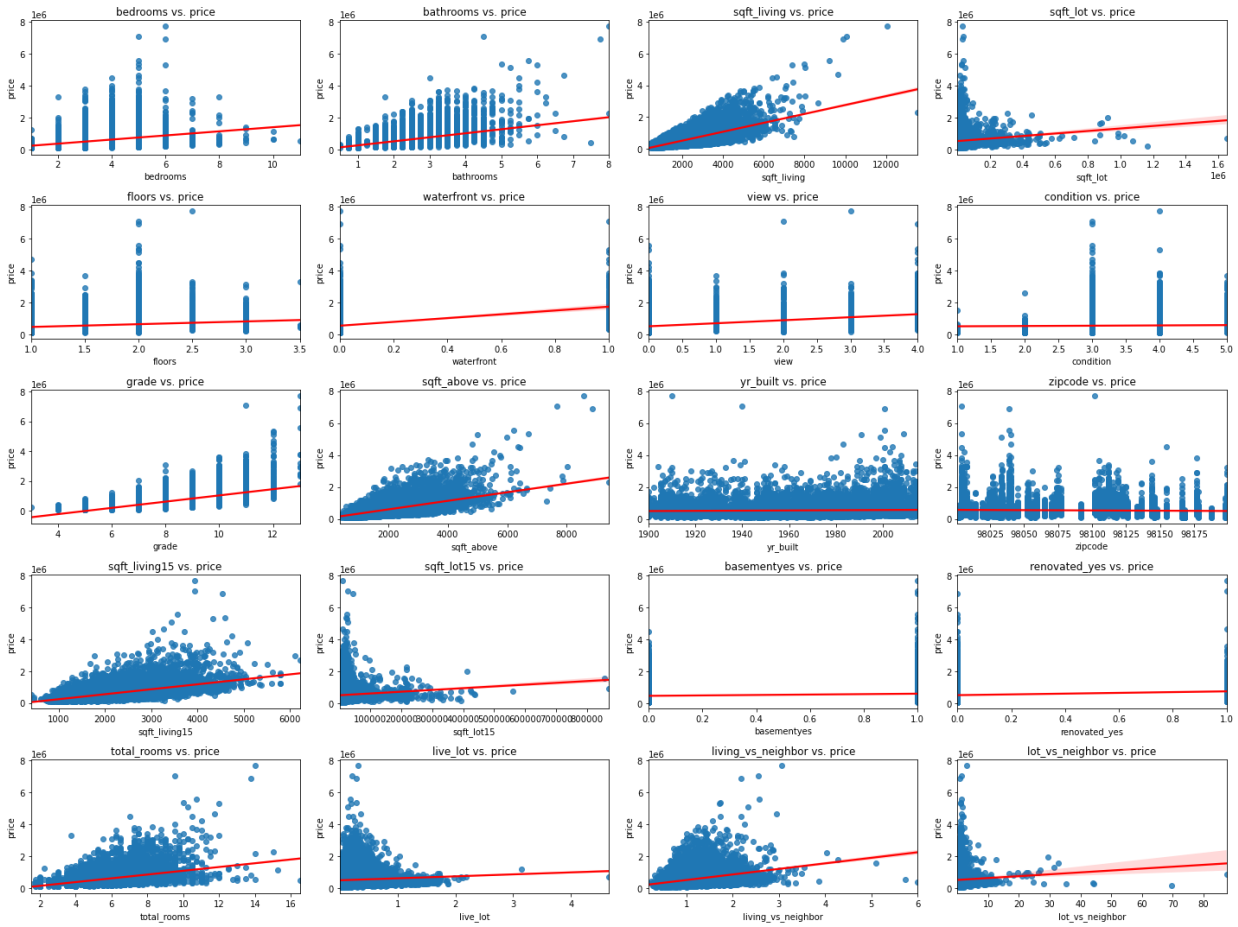
In [51]:
```
def lin_check(df, cols, ncols=4, figsize=(20,15)):
    """
    Produces regplot for each feature against price
    """
    if ncols%4==0:
        fig, axes = plt.subplots(nrows=(len(cols)//ncols), ncols=ncols,
        for ax, col in zip(axes.flatten(), cols):
            sns.regplot(data=df, x=col, y='price', ax=ax, line_kws={'co
            ax.set_title(f'{col} vs. price')
        fig.tight_layout()
    else:
        fig, axes = plt.subplots(nrows=(len(cols)//ncols)+1, ncols=ncol
        for ax, col in zip(axes.flatten(), cols):
            sns.regplot(data=df, x=col, y='price', ax=ax, line_kws={'co
            ax.set_title(f'{col} vs. price')
        fig.tight_layout()
```

In [52]:
```
df.columns
```

Out[52]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'yr_built', 'zipcode', 'lat', 'long', 'sqft_living1
5',
       'sqft_lot15', 'basementyes', 'renovated_yes', 'total_rooms', 'live
_lot',
       'living_vs_neighbor', 'lot_vs_neighbor'],
      dtype='object')
```

In [53]:
```
# Choosing to remove latitude and longitude from predictor variables be
# is a sufficient proxy for location. Additionally, for matters of inte
# will be easier for residents to understand zip code recommendations a

cols_to_check = ['bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade'
       'sqft_above', 'yr_built', 'zipcode','sqft_living15',
       'sqft_lot15', 'basementyes', 'renovated_yes', 'total_rooms', 'li
       'living_vs_neighbor', 'lot_vs_neighbor']
```

In [54]:
```
# Checking for linearity between predictor values and price

lin_check(df, cols_to_check)
```



The following cells do not have a linear relationship with price:

- Sqft_lot
- Floors (categorical)
- View (categorical)
- Condition (categorical)
- Yr_built
- Zip code (categorical)
- Basementyes (categorical)
- Renovatedyes (categorical)
- Sqft_lot15
- Live_lot

Sqft_lot, Yr_built, sqft_lot15, and live_lot are numeric variables that very clearly do not have a linear relationship with price. Of the categorical variables that do not have a linear relationship with price, will proceed by creater bar plots to evaluate their relationship with price. If there is not a clear linear

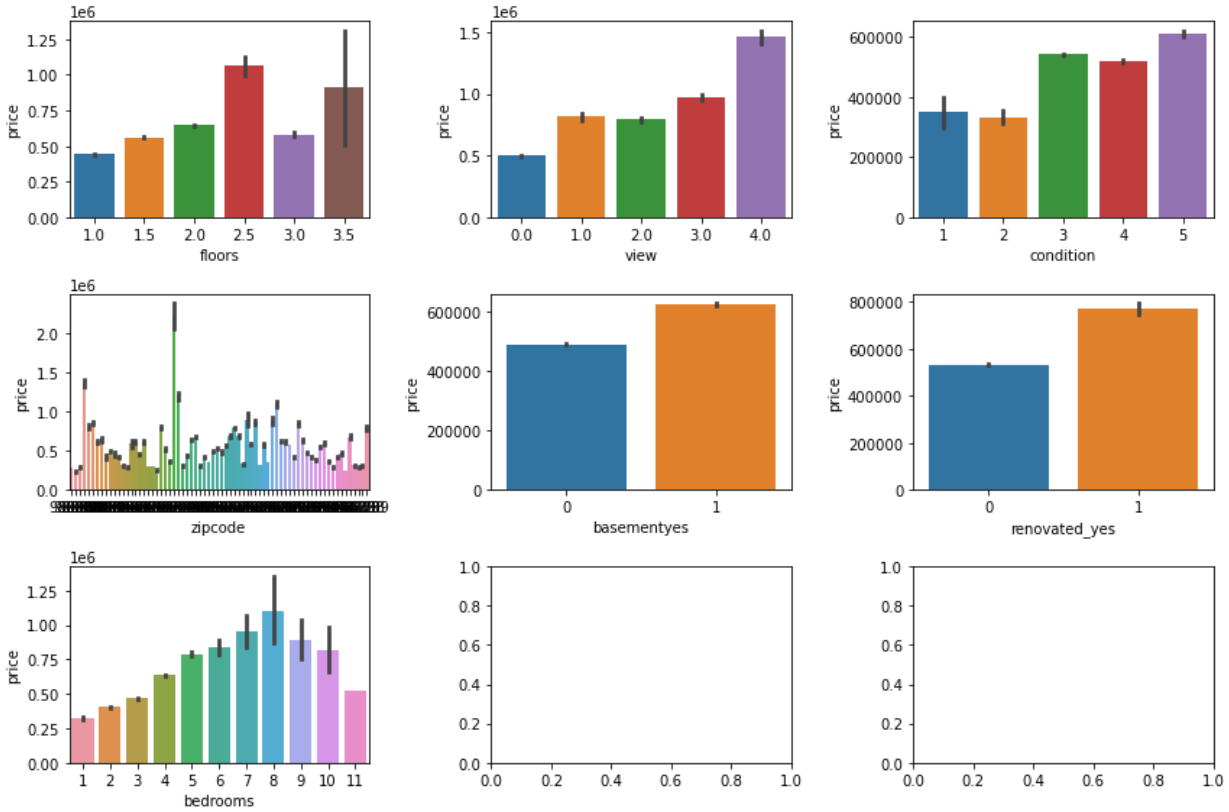
relationship with price, I will One Hot Encode them

In [55]:
```
1 df.columns
```

Out[55]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'yr_built', 'zipcode', 'lat', 'long', 'sqft_living1
5',
       'sqft_lot15', 'basementyes', 'renovated_yes', 'total_rooms', 'live
_lot',
       'living_vs_neighbor', 'lot_vs_neighbor'],
      dtype='object')
```

In [56]:
```
1 # These were continuous variables that do not have a linear relationshi
2
3 cols_to_drop = ['sqft_lot', 'sqft_lot15', 'live_lot', 'yr_built']
4 df_lin = df.drop(cols_to_drop, axis=1)
```

In [57]:
```
1 # Checking if categorical variables appear to have a linear relationshi
2
3 cat_bars = ['floors', 'view', 'condition', 'zipcode', 'basementyes', 'r
4 fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(12,8))
5 for ax, col in zip(axes.flatten(), cat_bars):
6     sns.barplot(data=df, x=col, y='price', ax=ax, ci=68)
7     fig.tight_layout()
8
```



- Floors, and zipcode do not have linear relationships with price
- Condition is close, but roughly has a linear relationship with price

- Will turn floors, zip code, and bedrooms into OHE variables to see if they make an impact on the model

Conclusion:

- Will be dropping: Sqft_lot, Yr_built, sqft_lot15, and live_lot are numeric
- Will be One Hot Encoding: Floors, zipcode, and bedrooms

5.2 Check Assumption of Multicollinearity

For a multiple linear regression model to be accurate, it must meet the assumption that the predictor variables do not have multicollinearity. This means that not only should the predictors have a linear relationship with target, but they should not have a linear relationship with each other. In essense, if two variables move very close together, then they are redundant for inferential capabilities and may make it confusing to interpret which variable is contributing to the change in predictor values

RoadMap for checking assumptions of multicollinearity

1. Run initial check of correlation with price
2. Observe heatmap triangle to see which predictors have strong correlation with each other and price
3. Build table to show which variables have a correlation of greater than 0.75 with each other
    - 0.75 is the norm for determing if predictor values have muliticollinearity

In [58]:
```
# Check correlation with price

def initial_corr_check(df, col='price'):
    return df.corr()['price'].round(2).sort_values(ascending=False)
```

In [59]:
```
initial_corr_check(df_lin.drop(columns='id'))
```

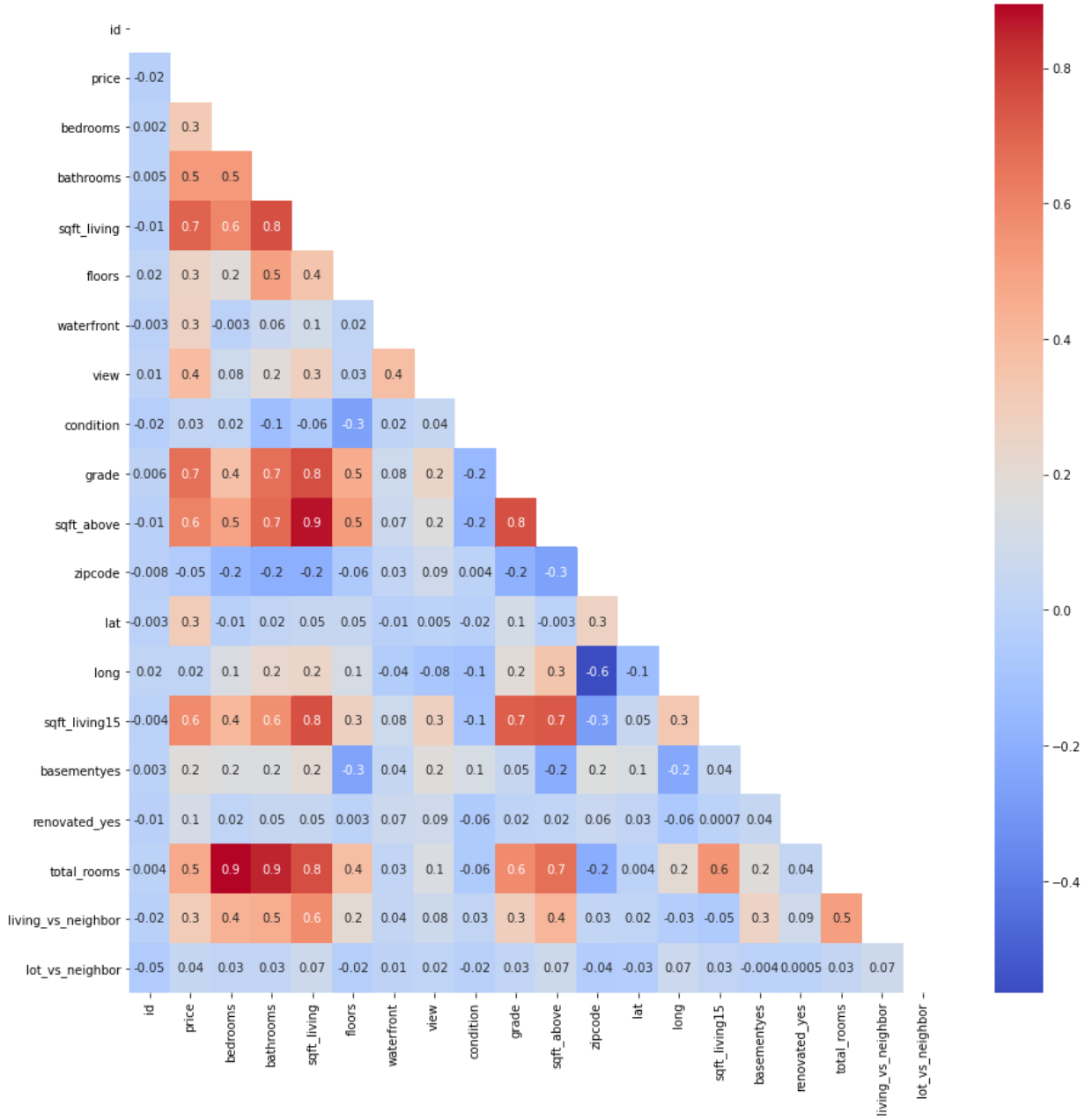
Out[59]:
```
price                 1.00000
sqft_living           0.70000
grade                 0.67000
sqft_above            0.60000
sqft_living15         0.58000
bathrooms             0.53000
total_rooms           0.47000
view                  0.39000
bedrooms              0.32000
lat                   0.31000
living_vs_neighbor    0.30000
waterfront            0.27000
floors                0.25000
basementyes           0.18000
renovated_yes         0.12000
lot_vs_neighbor       0.04000
condition             0.03000
long                  0.02000
zipcode              -0.05000
Name: price, dtype: float64
```

Initial correlation check shows that sqft_living, grade, and sqft_above have the most positive linear relationship with price

In [60]:
```
# Reference: https://heartbeat.fritz.ai/seaborn-heatmaps-13-ways-to-cus

def corr_triangle(df):
    """
    Correlation heatcamp, including price
    """
    corr2 = df.corr()
    fig, ax = plt.subplots(figsize=(15,15))
    matrix = np.triu(corr2)
    return sns.heatmap(corr2,cmap="coolwarm", annot=True, fmt='.1g', ma
```

In [61]:
```
1 corr_triangle(df_lin)
```

Out[61]: <AxesSubplot:>



There are a number of features that have strong multicollinearity. This is partially due to feature engineering and transforming existing columns to produce new columns

- Total rooms correlates strongly with bedrooms and bathrooms because that is how it was developed
- Sqft_living and sqft_living 15 are strongly correlated
- Sqft_living and sqft_above and grade are strongly correlated
  - Because sqft_living and sqft_above will be the same for homes that do not have basements

In [62]:
```
# Reference:https://github.com/learn-co-curriculum/dsc-multicollinearit

def corr_finder(df):
    """
    Shows pairs of features that have a correlation of greater than 0.7
    each other
    """
    df_corr = df.corr().abs().stack().reset_index().sort_values(0, asce
    df_corr['pairs'] = list(zip(df_corr.level_0, df_corr.level_1))
    df_corr.set_index(['pairs'], inplace = True)
    df_corr.drop(columns=['level_1', 'level_0'], inplace = True)

    # # cc for correlation coefficient
    df_corr.columns = ['cc']
    df_corr.drop_duplicates(inplace=True)

    return df_corr[(df_corr.cc>.75) & (df_corr.cc<1)]
```

In [63]:
```
corr_finder(df_lin)
```

Out[63]:

	cc
pairs	
(total_rooms, bedrooms)	0.89510
(sqft_living, sqft_above)	0.87655
(bathrooms, total_rooms)	0.85186
(total_rooms, sqft_living)	0.76392
(sqft_living, grade)	0.76243
(sqft_living15, sqft_living)	0.75630
(grade, sqft_above)	0.75610
(bathrooms, sqft_living)	0.75581

Methodology to handle collinearity:

- For each pair, drop the feature that has the lower correlation with price
  - Maintaining the feature that has a stronger relationship with price

- Bedrooms & Total_Rooms: Drop Total Rooms because it has multicollinearity with many other features an reduces the nuance of the difference between suggesting adding either bedrooms or bathrooms
- Sqft_living & Sqft_above: Drop Sqft_above because lower correlation with price. Additionally sqft_living included basement which we have now represented with a binary variable. Nuance of the basement is not lost
- Total_Rooms and Bathrooms: Already eliminated Total_Rooms
- Total_Rooms and Sqft_living: Already eliminated both features
- Sqft_living15 and Sqft_living: Already eliminated sqft_living
- Sqft_above and grade: Electing to keep because they represent very different features
- Sqft_living and bathrooms: Already dropped sqft_living

In [64]:
```
cols_to_drop = ['total_rooms', 'sqft_above', 'bathrooms', 'sqft_living1
df_linco = df_lin.drop(cols_to_drop, axis=1)
```

In [65]:
```
# Confirm no multicollinearity issues except for sqft_living and grade
# Elected to keep because they represent very different predictions

corr_finder(df_linco)
```

Out[65]:

	cc
pairs	
(grade, sqft_living)	0.76243

Our model is now closer to meeting all the necessary assumptions to provide accurate inferences.

Next steps will be to handle outliers and then categorical variables

6 Model 1: Baseline Model

- Now that assumptions of linearity and no multicollinearity have been met, going to run a baseline OLS model
- Mainly observing R squared, Adjusted R squared, QQ Plot, and Homoskedacicity model
- Will check if certain variables are statistically insignificant based on their P-Value but will not be dropping variables because we have not yet handled outliers

What is R^2, Adjusted R^2, QQ Plot, and Homoskedacicity?

- R^2: Indicated how much variance in the dependent variable is explained by the independent variable. This is our 'goodness of fit' test. The higher the value (between 0-1), the better our model does at explaining the variance of the dependent model

- Adjusted R^2: Similar to R squared however it accounts for the number of independent variables. In other words, it has a downward biance as the number of independent variables increases.

- Homoskedacicity: For our model to be accurate, it must meet the assumption of homoskedacicity. This means that our residuals (actual value-predicted value) cannot deviate accross different independent variables. The variance shuld not have a recognizable pattern

- QQ Plot: Helps us measure homoskedacicity. Want to see the dots follow the 45 degree line. Shows us where the variance becomes non-uniform

Ref: https://statisticsbyjim.com/regression/interpret-r-squared-regression/ (https://statisticsbyjim.com/regression/interpret-r-squared-regression/)

In [66]:
```
import statsmodels.api as sm
import statsmodels.stats.api as sms
import statsmodels.formula.api as smf

from scipy import stats
from sklearn.preprocessing import StandardScaler
from statsmodels.formula.api import ols

from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
import sklearn.metrics as metrics
```

In [67]:
```
def model_summary(df, X_targets, y, qq=True):
    '''
    Produces OLS Linear Regression summary. True/False toggles if the Q
    plot is displayed below the summary
    '''
    outcome = y
    x_cols = X_targets
    predictors = '+'.join(x_cols)
    formula = outcome + '~' + predictors
    model = ols(formula=formula, data=df).fit()
    resid1 = model.resid
    display(model.summary())
    if qq==True:

        sm.graphics.qqplot(resid1, dist=stats.norm, line='45', fit=True

    return model
```

In [68]:
```
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
import sklearn.metrics as metrics


def sked_show(df, X_cols, lr=None, val='price'):
    '''
    Produces scatter plot showing measue of homoskedacicity
    '''
    if lr is None:
        lr = LinearRegression()
        lr.fit(df[X_cols], df[val])

        y_hat = lr.predict(df[X_cols])
    else:
        y_hat = lr.predict(df)


    resid = (df[val] - y_hat)
    fig, ax= plt.subplots(figsize=(5,5))
    ax.scatter(x=y_hat,y=resid, alpha=0.1)
    ax.axhline(0, color='red')
    ax.set_xlabel('Price')
    ax.set_ylabel('Residual')
    return fig,ax
```

In [69]:
```
# Begin by using all columns as predictor values. Establishing a baseli

x_targs = ['bedrooms', 'floors', 'waterfront',
       'view', 'condition', 'grade', 'zipcode',
       'basementyes', 'renovated_yes', 'living_vs_neighbor',
       'lot_vs_neighbor', 'sqft_living']
```

In [70]:
```
model_base = model_summary(df_linco, x_targs, 'price')
sked_show(df, x_targs, model_base)
```

OLS Regression Results

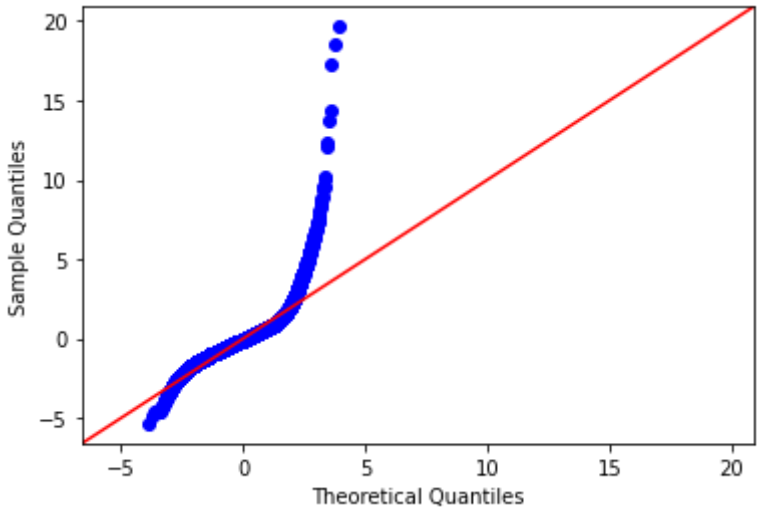
Dep. Variable:	price	**R-squared:**	0.618
Model:	OLS	**Adj. R-squared:**	0.618
Method:	Least Squares	**F-statistic:**	2883.
Date:	Wed, 05 May 2021	**Prob (F-statistic):**	0.00
Time:	14:10:57	**Log-Likelihood:**	-2.9411e+05
No. Observations:	21387	**AIC:**	5.882e+05
Df Residuals:	21374	**BIC:**	5.884e+05
Df Model:	12		
Covariance Type:	nonrobust		

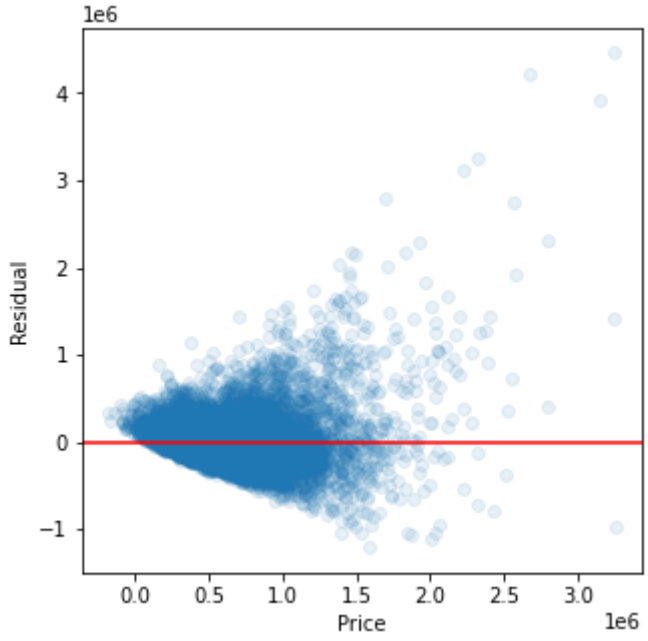
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-5.254e+07	3.07e+06	-17.129	0.000	-5.86e+07	-4.65e+07
bedrooms	-3.239e+04	2204.113	-14.694	0.000	-3.67e+04	-2.81e+04
floors	-3302.0105	3555.724	-0.929	0.353	-1.03e+04	3667.475
waterfront	6.124e+05	2.05e+04	29.829	0.000	5.72e+05	6.53e+05
view	4.962e+04	2361.401	21.012	0.000	4.5e+04	5.42e+04
condition	6.098e+04	2514.696	24.248	0.000	5.6e+04	6.59e+04
grade	9.624e+04	2284.261	42.133	0.000	9.18e+04	1.01e+05
zipcode	528.9805	31.267	16.918	0.000	467.695	590.266
basementyes	2.791e+04	3651.532	7.643	0.000	2.08e+04	3.51e+04
renovated_yes	1.524e+05	8613.485	17.697	0.000	1.36e+05	1.69e+05
living_vs_neighbor	-1.008e+05	6466.963	-15.591	0.000	-1.13e+05	-8.81e+04
lot_vs_neighbor	716.1896	1216.573	0.589	0.556	-1668.385	3100.764
sqft_living	213.4715	3.621	58.950	0.000	206.374	220.569

Omnibus:	15558.225	**Durbin-Watson:**	1.580
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	926735.693
Skew:	2.918	**Prob(JB):**	0.00
Kurtosis:	34.716	**Cond. No.**	1.94e+08

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.94e+08. This might indicate that there are strong multicollinearity or other numerical problems.

Out[70]: (<Figure size 360x360 with 1 Axes>,
<AxesSubplot:xlabel='Price', ylabel='Residual'>)





Conclusions

- R^2: 0.618
- Adjusted R^2: 0.618
- QQ Plot: Deviates upwards at the 2nd/3rd quantile. Suggests we have high outlier values that the model is failing to explain
- Homoskedacicity: Becomes cone shaped aournd $1.25 million. Suggests outlier values
- Non-Statistically Significant Preditors: Floors, lot_vs_neighbor
- Not dropping non-statistically significant values yet because they may have a significant relationship with price when we remove outlier values

7 Outlier Removal: IQR + Z-Score

- Due to high outliers shown in the QQ Plot, next step is to remove outliers
- Will try using Z-Score and IQR method
- Will evaluate data loss to determine which methodology I will pursue for modeling purposes

7.1 IQR Method

- The IQR method uses quantiles to determine if a value is considered an outlier
- The process:

    1. Determine IQR which is calcuated as Quantile 3 - Quantile 1
    2. Upper Thresh: Quantile 3 * 1.5 *IQR
    3. Lower Thresh: Quantile 1 * 1.5 * -IQR
    4. Any values outside of the upper and lower thresholds are considered outlier values
- This method is more strict for evaluating outliers than Z-Score

7.1.1 IQR Method Accross All Columns

- In this approach, we are going to classify outlier observations as those with at least one outlier accross all features
- For example, if an observation is has bedrooms which are an outlier, it will be removed from the dataset
- This method is more strict for determining outliers
- CAUTION: we may have significant data loss

In [71]:
```
def find_outliers_IQR(data):
    """Detects outliers using the 1.5*IQR thresholds.
    Returns a boolean Series where True=outlier"""
    res = data.describe()
    q1 = res['25%']
    q3 = res['75%']
    thresh = 1.5*(q3-q1)
    idx_outliers =(data < (q1-thresh)) | (data > (q3+thresh))
    return idx_outliers
```

In [72]:
```
df_linco.columns
```

Out[72]:
```
Index(['id', 'date', 'price', 'bedrooms', 'sqft_living', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'zipcode', 'lat', 'lon
g',
       'basementyes', 'renovated_yes', 'living_vs_neighbor',
       'lot_vs_neighbor'],
      dtype='object')
```

In [73]:
```
# Cols we are going to check with the IQR method
# Do not perform IQR check on binary variables or categorical variables

iqr_check = ['price', 'bedrooms',
             'sqft_living',
            'living_vs_neighbor', 'lot_vs_neighbor']

```

In [74]:
```
iqr_outliers = pd.DataFrame()
for col in iqr_check:
    iqr_outliers[col]=find_outliers_IQR(df_linco[col])
iqr_outliers['total'] = iqr_outliers.any(axis=1)
df_iqr = df_linco[~iqr_outliers['total']].copy()
```

In [75]:
```
# Confirm that our data looks correct in terms of columns

df_iqr.head()
```

Out[75]:

	id	date	price	bedrooms	sqft_living	floors	waterfront	view	condition
0	7129300520	2014-10-13	221900.00000	3	1180	1.00000	0.00000	0.00000	3
1	6414100192	2014-12-09	538000.00000	3	2570	2.00000	0.00000	0.00000	3
3	2487200875	2014-12-09	604000.00000	4	1960	1.00000	0.00000	0.00000	5
4	1954400510	2015-02-18	510000.00000	3	1680	1.00000	0.00000	0.00000	3
6	1321400060	2014-06-27	257500.00000	3	1715	2.00000	0.00000	0.00000	3

In [76]:
```
df_iqr.describe()
```

Out[76]:

	id	price	bedrooms	sqft_living	floors	waterfront	
count	15475.00000	15475.00000	15475.00000	15475.00000	15475.00000	15475.00000	1547
mean	4734745636.78559	467032.92775	3.30546	1924.12284	1.46378	0.00097	
std	2878922190.79975	200357.29212	0.76801	696.80695	0.53187	0.03112	
min	1200019.00000	81000.00000	2.00000	560.00000	1.00000	0.00000	
25%	2297400055.00000	310000.00000	3.00000	1400.00000	1.00000	0.00000	
50%	4046710050.00000	429000.00000	3.00000	1820.00000	1.00000	0.00000	
75%	7504400730.00000	586750.00000	4.00000	2360.00000	2.00000	0.00000	
max	9900000190.00000	1120000.00000	5.00000	4230.00000	3.50000	1.00000	

With the new DataFrame, the range of values has been constrained to the following:

- Price

    - Min: $81,000
    - Max: $1,120,000
- Bedrooms
    - Min: 2
    - Max: 5
- Floors
    - Min: 1
    - Max: 3.5
- Sqft_Living
    - Min: 560
    - Max: 4230

This constrains our dataset to only be able to provide inferential capabilities for the above types of homes. The min and max price seem accurate because the QQ Plot was trailing off aroun $1.25 million

In [77]:
```
print(f'Num observations before dropping with IQR: {len(df):,}')
print(f'Num observations after dropping with IQR: {len(df_iqr):,}')
print(f'Num observations removed: {len(df)-len(df_iqr):,}')
print(f'Num observations removed as percent of original DF: {round(100*
```

```
Num observations before dropping with IQR: 21,387
Num observations after dropping with IQR: 15,475
Num observations removed: 5,912
Num observations removed as percent of original DF: 27.64%
```

With this type of outlier removal we have significant data loss. Aprox. 28% of our data will not be included for modeling purposes. Going to see how the model performs with these constraints

7.1.1.1 Model 2: IQR All Outliers Removed

- Check to see if assumption of homoskedacicity has improved
- Check if significant p-values has changed

In [78]:
```
# Ensure our DataFrame only includes non outlier values

df_iqr.describe()
```

Out[78]:

	id	price	bedrooms	sqft_living	floors	waterfront	
count	15475.00000	15475.00000	15475.00000	15475.00000	15475.00000	15475.00000	1547
mean	4734745636.78559	467032.92775	3.30546	1924.12284	1.46378	0.00097	
std	2878922190.79975	200357.29212	0.76801	696.80695	0.53187	0.03112	
min	1200019.00000	81000.00000	2.00000	560.00000	1.00000	0.00000	
25%	2297400055.00000	310000.00000	3.00000	1400.00000	1.00000	0.00000	
50%	4046710050.00000	429000.00000	3.00000	1820.00000	1.00000	0.00000	
75%	7504400730.00000	586750.00000	4.00000	2360.00000	2.00000	0.00000	
max	9900000190.00000	1120000.00000	5.00000	4230.00000	3.50000	1.00000	

In [79]:
```
x_targs = df_iqr.columns
x_targs = list(x_targs)
x_targs = [x for x in x_targs if x not in ('id', 'date', 'price')]
```

In [80]:
```
model_iqra = model_summary(df_iqr, x_targs, 'price')
sked_show(df_iqr, x_targs, model_iqra)
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.676
Model:	OLS	**Adj. R-squared:**	0.676
Method:	Least Squares	**F-statistic:**	2303.
Date:	Wed, 05 May 2021	**Prob (F-statistic):**	0.00
Time:	14:10:57	**Log-Likelihood:**	-2.0216e+05
No. Observations:	15475	**AIC:**	4.043e+05
Df Residuals:	15460	**BIC:**	4.045e+05
Df Model:	14		
Covariance Type:	nonrobust		

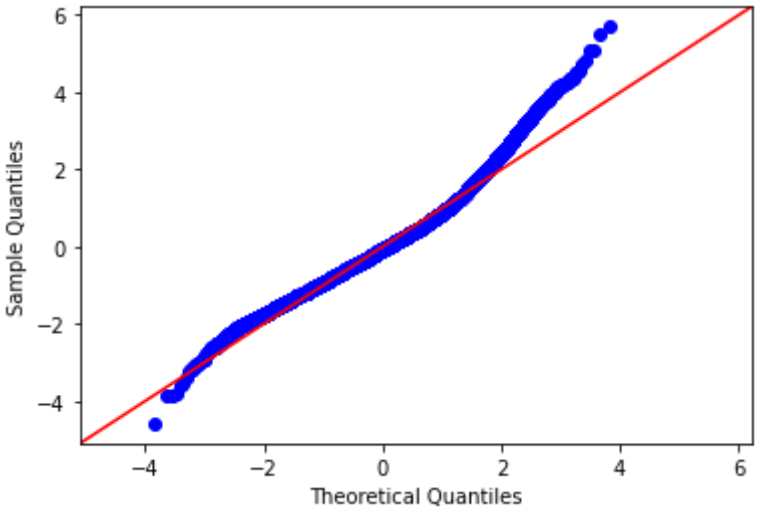
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-2.777e+07	1.84e+06	-15.108	0.000	-3.14e+07	-2.42e+07
bedrooms	-8472.8199	1525.406	-5.554	0.000	-1.15e+04	-5482.845
sqft_living	134.7622	2.678	50.328	0.000	129.514	140.011
floors	1.37e+04	2184.062	6.273	0.000	9419.236	1.8e+04
waterfront	1.016e+05	3.01e+04	3.373	0.001	4.26e+04	1.61e+05
view	3.655e+04	1686.343	21.674	0.000	3.32e+04	3.99e+04
condition	4.756e+04	1535.022	30.983	0.000	4.46e+04	5.06e+04
grade	5.888e+04	1520.542	38.722	0.000	5.59e+04	6.19e+04
zipcode	-119.1025	21.897	-5.439	0.000	-162.022	-76.183
lat	5.857e+05	6849.178	85.512	0.000	5.72e+05	5.99e+05
long	-9.245e+04	8475.166	-10.909	0.000	-1.09e+05	-7.58e+04
basementyes	1.76e+04	2287.099	7.694	0.000	1.31e+04	2.21e+04
renovated_yes	9.912e+04	6055.472	16.368	0.000	8.72e+04	1.11e+05
living_vs_neighbor	-1.105e+05	5856.715	-18.869	0.000	-1.22e+05	-9.9e+04
lot_vs_neighbor	3044.9759	8182.017	0.372	0.710	-1.3e+04	1.91e+04

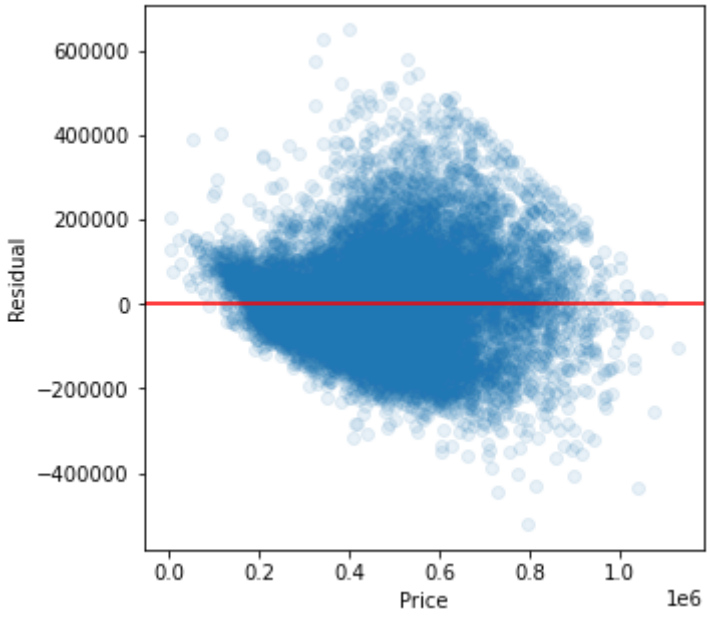
Omnibus:	1495.077	**Durbin-Watson:**	1.867
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2626.118
Skew:	0.679	**Prob(JB):**	0.00
Kurtosis:	4.494	**Cond. No.**	1.97e+08

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.97e+08. This might indicate that there are
strong multicollinearity or other numerical problems.

Out[80]: (<Figure size 360x360 with 1 Axes>,
<AxesSubplot:xlabel='Price', ylabel='Residual'>)





Conclusions

- R^2: 0.676
- Adjusted R^2: 0.676
- QQ Plot: Meets assumption but treads up slightly after the 2nd quantile. Scale on sample quantiles is -4 to 6 which is passable. To perfectly meet assumption the scale would be -4 to 4

- Homoskedacicity: No cone shape, passes assumptions. Possibly slight cone shape as data moves towards $800,000. Significant improvement from baseline model
- Non-Statistically Significant Preditors: lot_vs_neighbor
- Not going to use this model because there is too much data loss

7.1.2 IQR Price Outliers Removed

- Rather than considering an observation an outlier solely based on one feature, we are only considering an observation to be an outlier if price is an outlier
- This method reduces data loss because we are less strict on classifying outliers

In [81]:
```
# Finding the upper bound of the price threshold for outliers
# Since prices cannot be negative, we are not concerned with the lower

res=df_linco['price'].describe()
thresh = res['75%'] -res['25%']
u_bound=res['75%']+1.5*thresh
u_bound
```

Out[81]:
```
1125564.75
```

Maximum price for this model will be $1,125,564

In [82]:
```
# Subsetting the data to only include values below the upper threshold

df_iqrp = df_linco[df_linco['price'] <=u_bound]
```

In [83]:
```
# Ensure that observations have been dropped

print(len(df_iqrp))
print(df_iqrp['price'].max())
```

```
20235
1120000.0
```

Confirmed that we have less data than the original DataFrame

In [84]:
```
print(f'Num observations before dropping with IQR: {len(df):,}')
print(f'Num observations after dropping with IQR: {len(df_iqrp):,}')
print(f'Num observations removed: {len(df)-len(df_iqrp):,}')
print(f'Num observations removed as percent of original DF: {round(100*
```

```
Num observations before dropping with IQR: 21,387
Num observations after dropping with IQR: 20,235
Num observations removed: 1,152
Num observations removed as percent of original DF: 5.39%
```

With this type of outlier removal we have much lower data loss. Aprox. 5% of our data will not be included for modeling purposes. Going to see how the model performs with these constraints.

Compared to removing values based on a single feature determinig an observation as an outlier, this method preserves far more data for our model. This means that our model can provide inferences for a wider range of homes which is valuable for our purposes.

In [85]:
```
max_ip = df_iqrp['price'].max()
min_ip = df_iqrp['price'].min()

print(f'Min Price: ${min_ip:,}')
print(f'Max Price: ${max_ip:,}')
```

```
Min Price: $78,000.0
Max Price: $1,120,000.0
```

Our model provides inferential capabilities for homes priced between $78,000 and $1,120,000

7.1.2.1 Model 3: IQR Price Outliers Removed

- Considering an observation an outlier if price is above threshold
- This method reduces data loss compared to previous model because we are less strict on classifying outliers

In [86]:
```
model_iqrp = model_summary(df_iqrp, x_targs, 'price')
sked_show(df_iqrp, x_targs, model_iqrp)
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.667
Model:	OLS	**Adj. R-squared:**	0.667
Method:	Least Squares	**F-statistic:**	2896.
Date:	Wed, 05 May 2021	**Prob (F-statistic):**	0.00
Time:	14:10:57	**Log-Likelihood:**	-2.6522e+05
No. Observations:	20235	**AIC:**	5.305e+05
Df Residuals:	20220	**BIC:**	5.306e+05
Df Model:	14		
Covariance Type:	nonrobust		

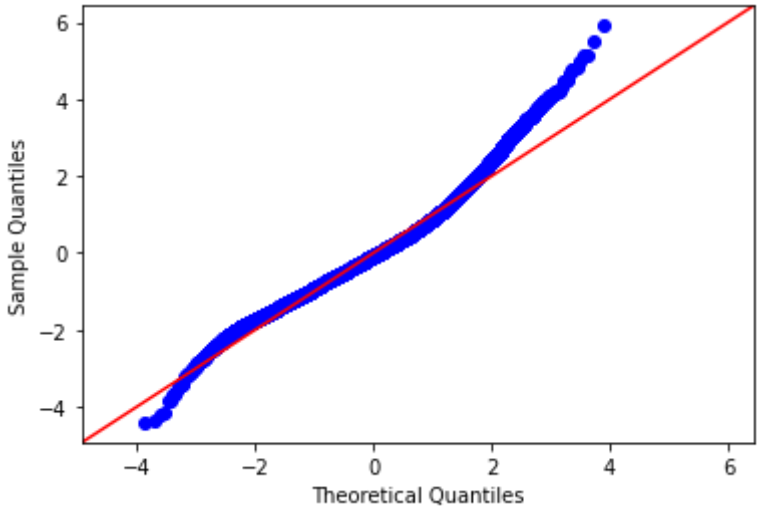
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-2.723e+07	1.67e+06	-16.304	0.000	-3.05e+07	-2.4e+07
bedrooms	-7627.4038	1221.868	-6.242	0.000	-1e+04	-5232.444
sqft_living	121.3012	2.212	54.847	0.000	116.966	125.636
floors	1.412e+04	1936.055	7.293	0.000	1.03e+04	1.79e+04
waterfront	1.211e+05	1.76e+04	6.880	0.000	8.66e+04	1.56e+05
view	3.667e+04	1428.352	25.676	0.000	3.39e+04	3.95e+04
condition	4.31e+04	1374.848	31.352	0.000	4.04e+04	4.58e+04
grade	6.184e+04	1288.038	48.009	0.000	5.93e+04	6.44e+04
zipcode	-134.2122	20.068	-6.688	0.000	-173.548	-94.877
lat	5.895e+05	6321.978	93.251	0.000	5.77e+05	6.02e+05
long	-9.848e+04	7632.839	-12.902	0.000	-1.13e+05	-8.35e+04
basementyes	1.008e+04	2030.602	4.962	0.000	6095.958	1.41e+04
renovated_yes	8.853e+04	4918.509	18.000	0.000	7.89e+04	9.82e+04
living_vs_neighbor	-6.305e+04	3695.609	-17.062	0.000	-7.03e+04	-5.58e+04
lot_vs_neighbor	4833.7807	678.485	7.124	0.000	3503.894	6163.667

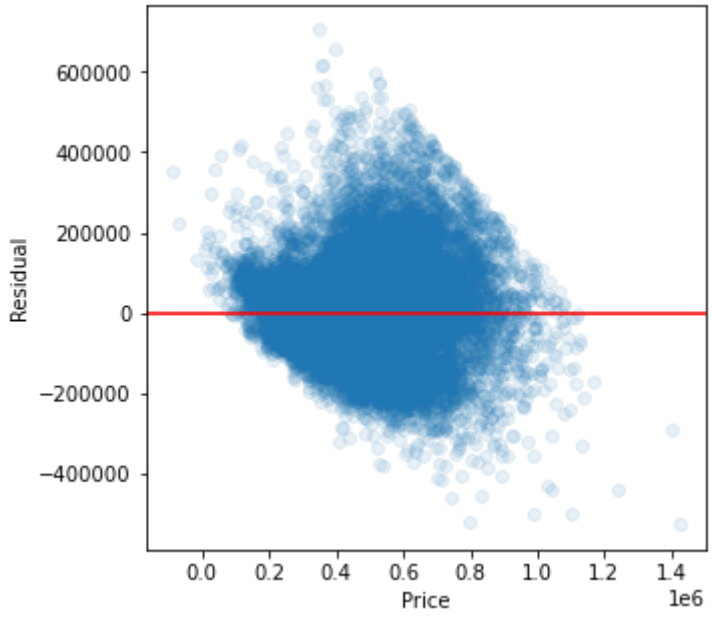
Omnibus:	1847.605	**Durbin-Watson:**	1.837
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3189.742
Skew:	0.655	**Prob(JB):**	0.00
Kurtosis:	4.438	**Cond. No.**	1.96e+08

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.96e+08. This might indicate that there are
strong multicollinearity or other numerical problems.

Out[86]: (<Figure size 360x360 with 1 Axes>,
<AxesSubplot:xlabel='Price', ylabel='Residual'>)





Conclusions

- R^2: 0.667
- Adjusted R^2: 0.667

- QQ Plot: Meets assumption but treads up slightly after the 2nd quantile. Scale on sample quantiles between -4 and 6
- Homoskedacicity: No cone shape, passes assumptions.
- Non-Statistically Significant Preditors: None
- This model is highly preferable to removing all columns because all assumptions can be met and our data loss is much less significant. Can provide a wider range of inferences

7.2 Z-Score Method

- Z-score outlier removal is another method for classifying data as outliers
- To begin we standardize our values as z scores. This means that we are going to transform the data so that it has a mean of 0 and a standard deviation of 1. This allows us to compare different features without having to worry about differences in magnitude or units
- We will not standardize Boolean values
- From there, determine if a value is an outlier based on having a z-score greater than or less than 3
  - This rule of thumb comes from the empirical rule which states that 99.7% of observations will lay between 0 and 3 standard deviations from the mean
  - Essentially, we are classifying a vale as an outlier if it is in the 0.03% quintile
- This method is less strict at classifying outliers than IQR method

7.2.1 Z-Score Method Accross All Columns

- In this approach, we are going to classify outlier observations as those with at least one outlier accross all features
- For example, if an observation is has bedrooms which are an outlier, it will be removed from the dataset
- This method is more strict for determining outliers
- CAUTION: we may have significant data loss

In [87]:
```
# Create scaler object

scaler = StandardScaler()
scaler
```

Out[87]:
```
StandardScaler()
```

In [88]:
```
# Create new DF to prepare for fit and transform

df_z = df_linco.copy()
```

In [89]:
```
df_z.columns
```

Out[89]:
```
Index(['id', 'date', 'price', 'bedrooms', 'sqft_living', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'zipcode', 'lat', 'lon
g',
       'basementyes', 'renovated_yes', 'living_vs_neighbor',
       'lot_vs_neighbor'],
      dtype='object')
```

In [90]:
```
# Not scaling binary variables such as waterfront, renovated, and basem
# Binary Variables are already encoded and will not have outlier values
# 0 or 1
# Note: Scaling does not actually change values relative to each other,

cols_to_scale = ['price', 'bedrooms', 'floors',
       'view', 'condition', 'grade', 'sqft_living',
       'living_vs_neighbor',
       'lot_vs_neighbor']
```

In [91]:
```
# Fit and transform original values into scaled values

df_z[cols_to_scale] = scaler.fit_transform(df_z[cols_to_scale])
df_z.describe()
```

Out[91]:

	id	price	bedrooms	sqft_living	floors	waterfront	
count	21387.00000	21387.00000	21387.00000	21387.00000	21387.00000	21387.00000	21387.0
mean	4581721940.59443	0.00000	0.00000	-0.00000	-0.00000	0.00678	0.0
std	2876772841.46664	1.00002	1.00002	1.00002	1.00002	0.08206	1.0
min	1000102.00000	-1.26066	-2.62718	-1.86373	-0.91774	0.00000	-0.3
25%	2124049194.50000	-0.58940	-0.41185	-0.70992	-0.91774	0.00000	-0.3
50%	3904930240.00000	-0.24815	-0.41185	-0.17655	0.00814	0.00000	-0.3
75%	7309100170.00000	0.28260	0.69582	0.50921	0.93402	0.00000	-0.3
max	9900000190.00000	19.48493	8.44950	12.47185	3.71165	1.00000	4.9

As we can see, the columns that we have scaled have a mean of 0 and standard deviation of 1

In [92]:
```
# Create new DataFrame where we are only going to include values that w
# less than 3 and greater than -3

outliers_z = pd.DataFrame()

for col in cols_to_scale:
    outliers_z[col] = df_z[col].abs()>3

outliers_z['total'] = outliers_z.any(axis=1)
df_za = df_z[~outliers_z['total']].copy()
```

In [93]:
```
print(len(df_za))
df_za.describe()
```

```
19735
```

Out[93]:

	id	price	bedrooms	sqft_living	floors	waterfront	
count	19735.00000	19735.00000	19735.00000	19735.00000	19735.00000	19735.00000	19735.0
mean	4608056409.49688	-0.14098	-0.05404	-0.11819	-0.02592	0.00030	-0.1
std	2876186313.51690	0.64731	0.94099	0.83404	0.99458	0.01743	0.5
min	1000102.00000	-1.24841	-2.62718	-1.86373	-0.91774	0.00000	-0.3
25%	2140950145.00000	-0.61559	-0.41185	-0.74257	-0.91774	0.00000	-0.3
50%	3918400013.00000	-0.27945	-0.41185	-0.25275	-0.91774	0.00000	-0.3
75%	7334550735.00000	0.17373	0.69582	0.37858	0.93402	0.00000	-0.3
max	9900000190.00000	2.99080	2.91116	2.98011	2.78577	1.00000	2.3

- We have created a new DataFrame that only includes observations where all scaled values fall between -3 and 3 Z-scores
- Our mean and SD are no longer (0,1) because we have removed outiers, however this makes sense given that we are trying to remove outliers
- In total, we now have 19,735 observations

In [94]:
```
# Formula that returns z-score back to original value

def z_to_value(z, mu=df['price'].mean(), sd=df['price'].std()):
    """
    Converts z-score to original value
    """
    x = sd*z+mu
    return round(x,2)
```

In [95]:
```
print(f'Max price: ${z_to_value(2.99):,}')
print(f'Min price: ${z_to_value(-1.24841):,}')
```

```
Max price: $1,639,733.25
Min price: $82,490.54
```

The model will be able to infer prices of homes between $82,490 and $1,639,733. This is a wider range of price constraints which makes sense because Z-score is less strict in terms of classifying outliers

In [96]:
```
print(f'Num observations before dropping with IQR: {len(df):,}')
print(f'Num observations after dropping with IQR: {len(df_za):,}')
print(f'Num observations removed: {len(df)-len(df_za):,}')
print(f'Num observations removed as percent of original DF: {round(100*
```

```
Num observations before dropping with IQR: 21,387
Num observations after dropping with IQR: 19,735
Num observations removed: 1,652
Num observations removed as percent of original DF: 7.72%
```

With this type of outlier removal our data loss is aprox 8%. Going to see how the model performs with these constraints. Check model to see if assumptions are met

This method will remove more data then when we only consider an observation an outlier based on price constraints

7.2.2 Model 3: Z-Score All Outliers Removed

In [97]:
```
model_za = model_summary(df_za, x_targs, 'price')
sked_show(df_za, x_targs, model_za)
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.660
Model:	OLS	**Adj. R-squared:**	0.660
Method:	Least Squares	**F-statistic:**	2734.
Date:	Wed, 05 May 2021	**Prob (F-statistic):**	0.00
Time:	14:10:58	**Log-Likelihood:**	-8773.8
No. Observations:	19735	**AIC:**	1.758e+04
Df Residuals:	19720	**BIC:**	1.770e+04
Df Model:	14		
Covariance Type:	nonrobust		

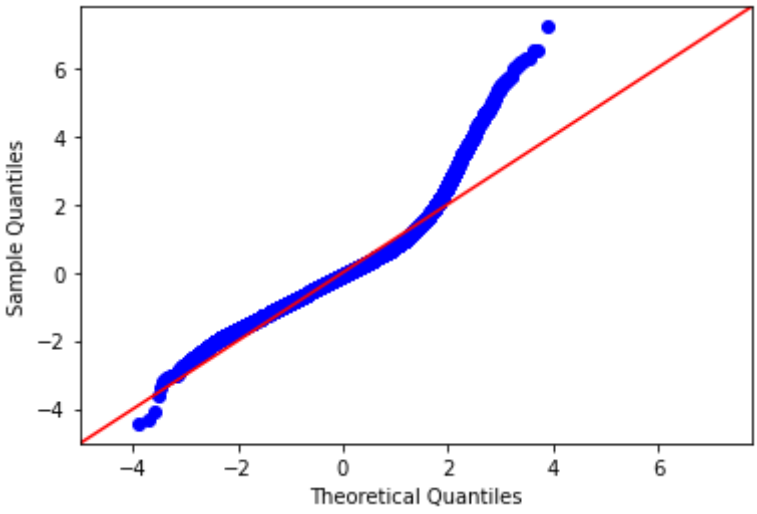
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-81.9507	5.385	-15.219	0.000	-92.505	-71.396
bedrooms	-0.0249	0.004	-6.775	0.000	-0.032	-0.018
sqft_living	0.3705	0.007	55.250	0.000	0.357	0.384
floors	0.0162	0.003	4.762	0.000	0.010	0.023
waterfront	0.6130	0.155	3.962	0.000	0.310	0.916
view	0.1000	0.005	19.771	0.000	0.090	0.110
condition	0.0888	0.003	30.751	0.000	0.083	0.094
grade	0.2319	0.005	46.912	0.000	0.222	0.242
zipcode	-0.0006	6.47e-05	-9.042	0.000	-0.001	-0.000
lat	1.6675	0.020	81.557	0.000	1.627	1.708
long	-0.4903	0.025	-19.848	0.000	-0.539	-0.442
basementyes	0.0290	0.007	4.421	0.000	0.016	0.042
renovated_yes	0.2978	0.016	18.476	0.000	0.266	0.329
living_vs_neighbor	-0.0819	0.004	-18.459	0.000	-0.091	-0.073
lot_vs_neighbor	0.0457	0.009	4.909	0.000	0.027	0.064

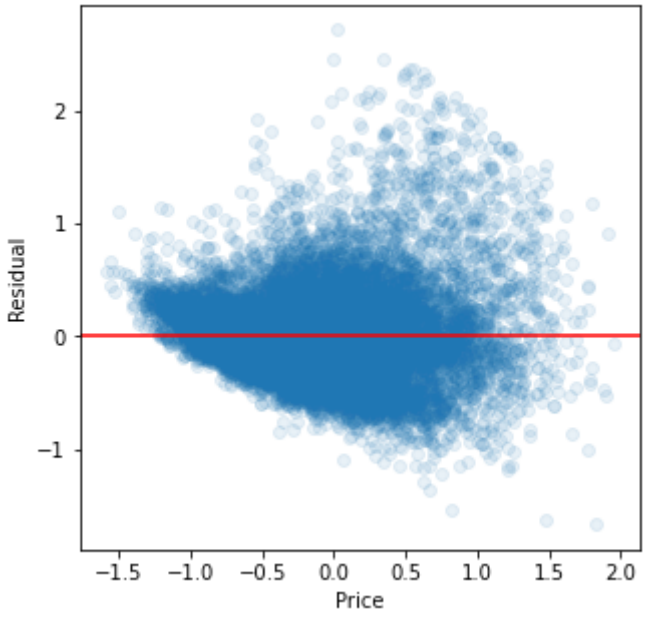
Omnibus:	5210.929	**Durbin-Watson:**	1.486
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	21272.001
Skew:	1.256	**Prob(JB):**	0.00
Kurtosis:	7.423	**Cond. No.**	1.96e+08

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.96e+08. This might indicate that there are
strong multicollinearity or other numerical problems.

Out[97]:
```
(<Figure size 360x360 with 1 Axes>,
 <AxesSubplot:xlabel='Price', ylabel='Residual'>)
```





Conclusions

- R^2: 0.660
- Adjusted R^2: 0.660
- QQ Plot: Does not do a good job at meeting assumptions as residuals begin to trail off at 2nd quantile. Same scale but much more deviation towards 2nd quantile
- Homoskedacicity: Cone shaped, especially as model reaches Z-score of 0.75
- Non-Statistically Significant Preditors: None
- Model does not meet all 4 assumptions, however data loss is better than IQR all drop

7.2.3 Z-Score Price Outliers Removed

- Rather than considering an observation an outlier solely based on one feature, we are only considering an observation to be an outlier if price is an outlier
- This method reduces data loss because we are less strict on classifying outliers
- Will be using same scaled data from previous model but only looking at price column

In [98]:
```
# Check to see if we have values greater than 3 and less than 3 in the
# Only need to worry about values greater than 3

df_z.describe()
```

Out[98]:

	id	price	bedrooms	sqft_living	floors	waterfront	
count	21387.00000	21387.00000	21387.00000	21387.00000	21387.00000	21387.00000	21387.0
mean	4581721940.59443	0.00000	0.00000	-0.00000	-0.00000	0.00678	0.0
std	2876772841.46664	1.00002	1.00002	1.00002	1.00002	0.08206	1.0
min	1000102.00000	-1.26066	-2.62718	-1.86373	-0.91774	0.00000	-0.3
25%	2124049194.50000	-0.58940	-0.41185	-0.70992	-0.91774	0.00000	-0.3
50%	3904930240.00000	-0.24815	-0.41185	-0.17655	0.00814	0.00000	-0.3
75%	7309100170.00000	0.28260	0.69582	0.50921	0.93402	0.00000	-0.3
max	9900000190.00000	19.48493	8.44950	12.47185	3.71165	1.00000	4.9

In [99]:
```
# Create new DataFrame only containing observations where the price has

df_zp = df_z[df_z['price'] < 3]
```

In [100]:
```
# Confirm that data has been removed
# Confirm that max price is less than 3

print(len(df_za))
df_zp['price'].max()
```

```
19735
```

Out[100]:
```
2.990795957515217
```

In [101]:
```
print(f'Max price: ${z_to_value(2.99):,}')
print(f'Min price: ${z_to_value(-1.26066):,}')
```

```
Max price: $1,639,733.25
Min price: $77,989.74
```

The model will be able to infer prices of homes between $77,989 and $1,639,733

In [102]:
```
print(f'Num observations before dropping with IQR: {len(df):,}')
print(f'Num observations after dropping with IQR: {len(df_zp):,}')
print(f'Num observations removed: {len(df)-len(df_zp):,}')
print(f'Num observations removed as percent of original DF: {round(100*
```

```
Num observations before dropping with IQR: 21,387
Num observations after dropping with IQR: 20,983
Num observations removed: 404
Num observations removed as percent of original DF: 1.89%
```

With this type of outlier removal our data loss is aprox 2%. Going to see how the model performs with these constraints. Check model to see if assumptions are met

This method removes the least amount of data which makes sense for two reasons. First, it is only classifying outliers as those that have a price outlier value. Second, Z-Score outlier removal is more strict than IQR outlier removal

7.2.3.1 Model 4: Z-Score Price Outliers Removed

- Considering an observation an outlier if price is above threshold
- This method reduces data loss compared to previous model because we are less strict on classifying outliers

In [103]:
```
model_zp = model_summary(df_zp, x_targs, 'price')
sked_show(df_zp, x_targs, model_zp)
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.686
Model:	OLS	**Adj. R-squared:**	0.686
Method:	Least Squares	**F-statistic:**	3269.
Date:	Wed, 05 May 2021	**Prob (F-statistic):**	0.00
Time:	14:10:58	**Log-Likelihood:**	-10259.
No. Observations:	20983	**AIC:**	2.055e+04
Df Residuals:	20968	**BIC:**	2.067e+04
Df Model:	14		
Covariance Type:	nonrobust		

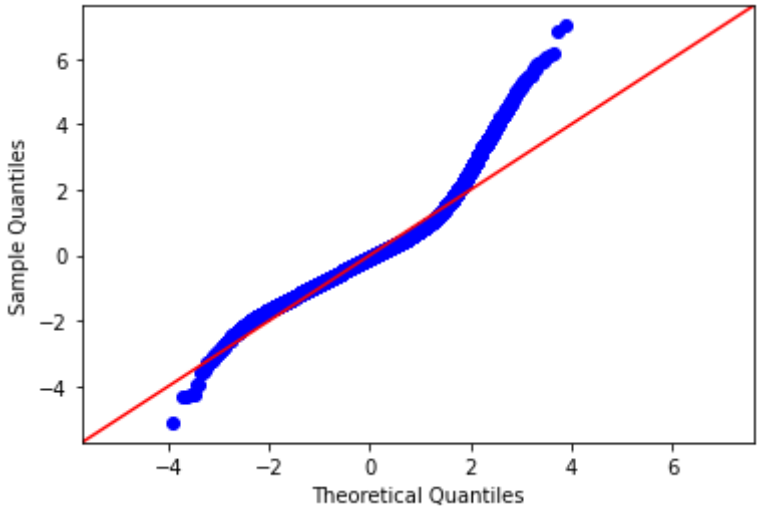
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-79.8958	5.424	-14.730	0.000	-90.528	-69.264
bedrooms	-0.0267	0.004	-7.542	0.000	-0.034	-0.020
sqft_living	0.3624	0.006	57.164	0.000	0.350	0.375
floors	0.0146	0.003	4.305	0.000	0.008	0.021
waterfront	0.5487	0.047	11.691	0.000	0.457	0.641
view	0.0991	0.003	30.002	0.000	0.093	0.106
condition	0.0860	0.003	29.774	0.000	0.080	0.092
grade	0.2411	0.005	50.040	0.000	0.232	0.251
zipcode	-0.0006	6.52e-05	-9.711	0.000	-0.001	-0.001
lat	1.7002	0.021	81.776	0.000	1.659	1.741
long	-0.4999	0.025	-20.061	0.000	-0.549	-0.451
basementyes	0.0215	0.007	3.271	0.001	0.009	0.034
renovated_yes	0.2953	0.016	19.038	0.000	0.265	0.326
living_vs_neighbor	-0.0637	0.004	-17.037	0.000	-0.071	-0.056
lot_vs_neighbor	0.0156	0.003	5.641	0.000	0.010	0.021

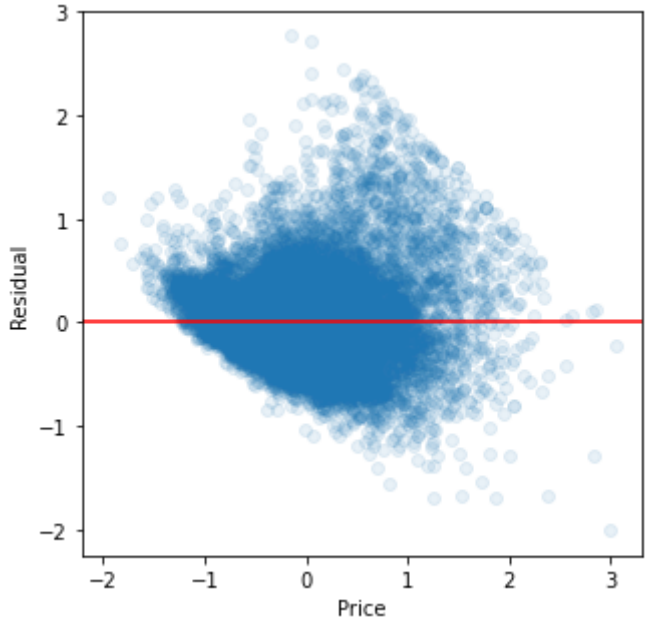
Omnibus:	4868.625	**Durbin-Watson:**	1.497
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	17767.809
Skew:	1.134	**Prob(JB):**	0.00
Kurtosis:	6.896	**Cond. No.**	1.95e+08

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.95e+08. This might indicate that there are
strong multicollinearity or other numerical problems.

Out[103]:
```
(<Figure size 360x360 with 1 Axes>,
 <AxesSubplot:xlabel='Price', ylabel='Residual'>)
```





Conclusions

- R^2: 0.686
- Adjusted R^2: 0.686
- QQ Plot: Does not do a good job at meeting assumptions as residuals begin to trail off at 2nd quantile
- Homoskedacicity: Cone shaped, especially as model reaches Z-score of 0.75
- Non-Statistically Significant Preditors: Bathrooms and Floors
- Model does not meet all 4 assumptions, however data loss is **very** low

Unfortunately, this model is not sufficient because it does not meet the assumption of homoskedacicity

7.3 Table to Compare 4 Outlier Removal Methods

- Want a simply way to evaluate all 4 models and conclude which outlier removal strategy is most effective

In [104]:
```
# Create DataFrame that compares all 4 outlier removal types

d = {
    'Outlier Type': ['IQR-All', 'IQR-Price', 'Z-All', 'Z-Price'],
    'Data Loss %':[27.6,5.4,7.7,1.9],
     'R^2':[0.676, 0.677, 0.660, 0.686],
    'Homoskedacicty':['Pass', 'Pass', 'Fail', 'Fail'],
    'QQ Plot':['Pass', 'Pass', 'Fail', 'Fail'],
    'Min Price':[81000, 78000.0, 82490, 77989],
    'Max Price':[1120000, 1120000, 1639733, 1639733]
    }
table_o = pd.DataFrame(d)
table_o.set_index('Outlier Type')
```

Out[104]:

Outlier Type	Data Loss %	R^2	Homoskedacicty	QQ Plot	Min Price	Max Price
IQR-All	27.60000	0.67600	Pass	Pass	81000.00000	1120000
IQR-Price	5.40000	0.67700	Pass	Pass	78000.00000	1120000
Z-All	7.70000	0.66000	Fail	Fail	82490.00000	1639733
Z-Price	1.90000	0.68600	Fail	Fail	77989.00000	1639733

As we can see, each outlier removal type creates varying degrees of data loss. Due to the constraints of linear regression, our model must pass the assumption of homoskedacicity. For that reason, I am going to rule out Z-Score outlier removal methodologies. When left with IQR-Price and IQR-All, choosing price makes far more sense because the dat loss is much lower and the R^2 is higher. Max/Min price are very close together

In conclusion, will move forward modeling with **IQR-Price**

8 Handling Categorical Variables with One Hot Encoding

- Dummy variables (one hot encoded variables) must be used to handle categorical variables because otherwise we will run into problems of multicollinearity. In other words, one of the dummy variables will be dropped because it can be explained by all of the others
- Our next step is to One Hot Encode the ordinal variables in our model
- These variables, when evaluated from an ordinal perpsetive, did not have a linear relationship with price
- However, we will evaluate their P-Values to determine if they have statistical significance as categorical variables

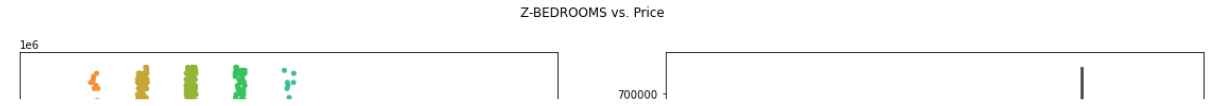
- Majority of OHE variables must be statistically significant
- If not, can potentially feature engineer them with nuance

8.1 Check relationship of Non-Linear Categorical Variables with Price

In [105]:
```
def ordinal_check(df, col, val='price'):
    """
    Produces stripplot and barplot to see if there is a linear relation
    the feature and price
    """
    fig, axes = plt.subplots(ncols=2, figsize=(20,6))
    sns.stripplot(data=df, x=col, y=val, ax=axes[0])
    sns.barplot(data=df, x=col, y=val, ax=axes[1], ci=68)

    fig.suptitle(f'Z-{col.upper()} vs. Price')
    plt.show()
    print('----------------------------------------------------------------
    print(df[col].value_counts(1))
```

In [106]:
```
# Based on our findings in our linearity check we are going to inspect
# Don't need to check binary variables because they are implicity code

cat_bars = ['floors', 'view', 'condition', 'zipcode', 'bedrooms']
for col in cat_bars:
    ordinal_check(df_iqrp, col)
```


```
------------------------------------------------------------
98038   0.02881
98103   0.02866
98052   0.02763
98115   0.02718
98042   0.02698
          ...
98102   0.00435
98109   0.00420
98024   0.00356
98148   0.00277
98039   0.00020
Name: zipcode, Length: 70, dtype: float64
```



Based on the results we are going to One Hot Encode the following variables:

- Floors: Increases up to 2.5 and then decreases
- Zipcode: Completely random
- Bedrooms: Increases up to 6 and then decreases
- Condition: Increases at 3 and then decreases at 4

This is because they do not appear ordinal. In other words, as the value of the independent variable increases, the price does not change at a constant rate. Check if they are statistically significant

8.2 One Hot Encode Categorical Non-Ordinal Columns

In [107]:
```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(sparse=False, drop='first')
encoder
```

Out[107]:
```
OneHotEncoder(drop='first', sparse=False)
```

In [108]:
```
# Seperate the columns we are going to OHE

cat_cols=['floors', 'zipcode', 'bedrooms', 'condition']

```

In [109]:
```
# Fit and transform categorical columns
# Turn matrix into DataFrame

encoder.fit(df_linco[cat_cols])

ohe_vars = encoder.transform(df_iqrp[cat_cols])
encoder.get_feature_names(cat_cols)
cat_vars = pd.DataFrame(ohe_vars,columns=encoder.get_feature_names(cat_
```

In [110]:
```
# Confirm variables are OHE

cat_vars
```

Out[110]:

	floors_1.5	floors_2.0	floors_2.5	floors_3.0	floors_3.5	zipcode_98002	zipcode_98003	zipcod
0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
1	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
2	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
3	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
4	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
...	...	...	...	...	...	...	...	
20230	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
20231	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
20232	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
20233	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
20234	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	

20235 rows × 88 columns

In [111]:
```
# OLS Formula does not accept '.' so must replace these with '_'

name_dict = {}
for col in cat_vars.columns:
    name_dict[col]=col.replace('.','_')
name_dict
```

Out[111]:
```
{'floors_1.5': 'floors_1_5',
 'floors_2.0': 'floors_2_0',
 'floors_2.5': 'floors_2_5',
 'floors_3.0': 'floors_3_0',
 'floors_3.5': 'floors_3_5',
 'zipcode_98002': 'zipcode_98002',
 'zipcode_98003': 'zipcode_98003',
 'zipcode_98004': 'zipcode_98004',
 'zipcode_98005': 'zipcode_98005',
 'zipcode_98006': 'zipcode_98006',
 'zipcode_98007': 'zipcode_98007',
 'zipcode_98008': 'zipcode_98008',
 'zipcode_98010': 'zipcode_98010',
 'zipcode_98011': 'zipcode_98011',
 'zipcode_98014': 'zipcode_98014',
 'zipcode_98019': 'zipcode_98019',
 'zipcode_98022': 'zipcode_98022',
 'zipcode_98023': 'zipcode_98023',
 'zipcode_98024': 'zipcode_98024',
 'zipcode_98027': 'zipcode_98027',
 'zipcode_98028': 'zipcode_98028',
 'zipcode_98029': 'zipcode_98029',
 'zipcode_98030': 'zipcode_98030',
 'zipcode_98031': 'zipcode_98031',
 'zipcode_98032': 'zipcode_98032',
 'zipcode_98033': 'zipcode_98033',
 'zipcode_98034': 'zipcode_98034',
 'zipcode_98038': 'zipcode_98038',
 'zipcode_98039': 'zipcode_98039',
 'zipcode_98040': 'zipcode_98040',
 'zipcode_98042': 'zipcode_98042',
 'zipcode_98045': 'zipcode_98045',
 'zipcode_98052': 'zipcode_98052',
 'zipcode_98053': 'zipcode_98053',
 'zipcode_98055': 'zipcode_98055',
 'zipcode_98056': 'zipcode_98056',
 'zipcode_98058': 'zipcode_98058',
 'zipcode_98059': 'zipcode_98059',
 'zipcode_98065': 'zipcode_98065',
 'zipcode_98070': 'zipcode_98070',
 'zipcode_98072': 'zipcode_98072',
 'zipcode_98074': 'zipcode_98074',
 'zipcode_98075': 'zipcode_98075',
 'zipcode_98077': 'zipcode_98077',
 'zipcode_98092': 'zipcode_98092',
 'zipcode_98102': 'zipcode_98102',
 'zipcode_98103': 'zipcode_98103',
 'zipcode_98105': 'zipcode_98105',
 'zipcode_98106': 'zipcode_98106',

 'zipcode_98107': 'zipcode_98107'
```

```
 'zipcode_98107': 'zipcode_98107',
'zipcode_98108': 'zipcode_98108',
'zipcode_98109': 'zipcode_98109',
'zipcode_98112': 'zipcode_98112',
'zipcode_98115': 'zipcode_98115',
'zipcode_98116': 'zipcode_98116',
'zipcode_98117': 'zipcode_98117',
'zipcode_98118': 'zipcode_98118',
'zipcode_98119': 'zipcode_98119',
'zipcode_98122': 'zipcode_98122',
'zipcode_98125': 'zipcode_98125',
'zipcode_98126': 'zipcode_98126',
'zipcode_98133': 'zipcode_98133',
'zipcode_98136': 'zipcode_98136',
'zipcode_98144': 'zipcode_98144',
'zipcode_98146': 'zipcode_98146',
'zipcode_98148': 'zipcode_98148',
'zipcode_98155': 'zipcode_98155',
'zipcode_98166': 'zipcode_98166',
'zipcode_98168': 'zipcode_98168',
'zipcode_98177': 'zipcode_98177',
'zipcode_98178': 'zipcode_98178',
'zipcode_98188': 'zipcode_98188',
'zipcode_98198': 'zipcode_98198',
'zipcode_98199': 'zipcode_98199',
'bedrooms_2': 'bedrooms_2',
'bedrooms_3': 'bedrooms_3',
'bedrooms_4': 'bedrooms_4',
'bedrooms_5': 'bedrooms_5',
'bedrooms_6': 'bedrooms_6',
'bedrooms_7': 'bedrooms_7',
'bedrooms_8': 'bedrooms_8',
'bedrooms_9': 'bedrooms_9',
'bedrooms_10': 'bedrooms_10',
'bedrooms_11': 'bedrooms_11',
'condition_2': 'condition_2',
'condition_3': 'condition_3',
'condition_4': 'condition_4',
'condition_5': 'condition_5'}
```

In [112]:
```
# Rename cat_vars DF with new names so it can be processed by OLS formu

cat_vars.rename(columns=name_dict, inplace=True)
cat_vars
```

Out[112]:

	floors_1_5	floors_2_0	floors_2_5	floors_3_0	floors_3_5	zipcode_98002	zipcode_98003	zipc
0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
1	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
2	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
3	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
4	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
...	...	...	...	...	...	...	...	
20230	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
20231	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
20232	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
20233	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
20234	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	

20235 rows × 88 columns

In [113]:
```
# Join OHE DataFrame back with original DateFrame
# Ensure each DataFrame has the same number of rows
# Begin by resetting index

df_iqrp=df_iqrp.reset_index()
```

In [114]:
```
# Check number of rows and ensure that it matches with the OHE DataFram
# Should be 20235

df_iqrp
```

Out[114]:

	index	id	date	price	bedrooms	sqft_living	floors	waterfront	viev
0	0	7129300520	2014-10-13	221900.00000	3	1180	1.00000	0.00000	0.0000
1	1	6414100192	2014-12-09	538000.00000	3	2570	2.00000	0.00000	0.0000
2	2	5631500400	2015-02-25	180000.00000	2	770	1.00000	0.00000	0.0000
3	3	2487200875	2014-12-09	604000.00000	4	1960	1.00000	0.00000	0.0000
4	4	1954400510	2015-02-18	510000.00000	3	1680	1.00000	0.00000	0.0000
...	...	...	...	...	...	...	...	...	.
20230	21453	1245002281	2014-05-12	1050000.00000	4	3280	2.00000	0.00000	0.0000
20231	21461	7010700308	2014-11-12	1010000.00000	4	3610	2.00000	0.00000	0.0000
20232	21532	8835770330	2014-08-19	1060000.00000	2	2370	2.00000	0.00000	0.0000
20233	21577	8672200110	2015-03-17	1090000.00000	5	4170	2.00000	0.00000	2.0000
20234	21590	7936000429	2015-03-26	1010000.00000	4	3510	2.00000	0.00000	0.0000

20235 rows × 18 columns

In [115]:
```
# Check to ensure that concat was done on the proper axis

df_ohe = pd.concat([df_iqrp, cat_vars], axis=1, ignore_index=False)
```

In [116]:
```
# Ensure we do not have any missing values

df_ohe.isna().sum()
```

Out[116]:
```
index           0
id              0
date            0
price           0
bedrooms        0
               ..
bedrooms_11     0
condition_2     0
condition_3     0
condition_4     0
condition_5     0
Length: 106, dtype: int64
```

We now have a new DataFrame called df_ohe that includes non-linear categorical variable as one hot encoded variables. This way, our model will be better meet assumptions of no multicollinearity, and all independent variables having a linear relationship with price. Important to drop original non OHE variables from DataFrame so they are not double counted

OHE variables are interpreted as such: With respect to the intercept, for this specific category, the target would variable would altered by the value of the coefficient. For example, if floors_1_5 had a coefficient of 15, the price would move up by $15.

In [117]:
```
# Drop non-OHE variables so they are not double counted

cols_to_drop = ['floors', 'zipcode', 'bedrooms', 'condition']
df_ohe.drop(cols_to_drop, axis=1, inplace=True)
```

In [118]:
```
# Ensure we do not have extra columns

df_ohe.describe()
```

Out[118]:

	index	id	price	sqft_living	waterfront	view	
count	20235.00000	20235.00000	20235.00000	20235.00000	20235.00000	20235.00000	2023
mean	10778.21596	4605299135.81389	477281.03736	1976.69795	0.00247	0.17163	
std	6231.88071	2877559932.98793	206564.79018	773.94673	0.04965	0.63834	
min	0.00000	1000102.00000	78000.00000	370.00000	0.00000	0.00000	
25%	5381.50000	2136600137.50000	316000.00000	1400.00000	0.00000	0.00000	
50%	10758.00000	3905081500.00000	439000.00000	1860.00000	0.00000	0.00000	
75%	16173.00000	7338220140.00000	600000.00000	2440.00000	0.00000	0.00000	
max	21596.00000	9900000190.00000	1120000.00000	7480.00000	1.00000	4.00000	1

In [119]:
```
# Confirm non-OHE variables have been dropped

df_ohe.head()
```

Out[119]:

	index	id	date	price	sqft_living	waterfront	view	grade	lat	
0	0	7129300520	2014-10-13	221900.00000	1180	0.00000	0.00000	7	47.51120	-122.2
1	1	6414100192	2014-12-09	538000.00000	2570	0.00000	0.00000	7	47.72100	-122.3
2	2	5631500400	2015-02-25	180000.00000	770	0.00000	0.00000	6	47.73790	-122.2
3	3	2487200875	2014-12-09	604000.00000	1960	0.00000	0.00000	7	47.52080	-122.3
4	4	1954400510	2015-02-18	510000.00000	1680	0.00000	0.00000	8	47.61680	-122.0

In [120]:
```
# Create list of columns excluding id, date, and price because that is

x_targs = df_ohe.columns
x_targs = list(x_targs)
x_targs = [x for x in x_targs if x not in ('id', 'date', 'price', 'inde
x_targs
```

Out[120]:
```
['sqft_living',
 'waterfront',
 'view',
 'grade',
 'lat',
 'long',
 'basementyes',
 'renovated_yes',
 'living_vs_neighbor',
 'lot_vs_neighbor',
 'floors_1_5',
 'floors_2_0',
 'floors_2_5',
 'floors_3_0',
 'floors_3_5',
 'zipcode_98002',
 'zipcode_98003',
 '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_98168',
'zipcode_98177',
'zipcode_98178',
'zipcode_98188',
'zipcode_98198',
'zipcode_98199',
'bedrooms_2',
'bedrooms_3',
'bedrooms_4',
'bedrooms_5',
'bedrooms_6',
'bedrooms_7',
'bedrooms_8',
'bedrooms_9',
'bedrooms_10',
'bedrooms_11',
'condition_2',
'condition_3',
'condition_4',
'condition_5']
```

8.2.1 Model 5: OHE Iteration 1

In [121]:
```
model_ohe = model_summary(df_ohe, x_targs, 'price')
sked_show(df_ohe, x_targs, model_ohe)
```

sqft_living	135.6679	1.717	79.017	0.000	132.303	139.033
waterfront	1.644e+05	1.31e+04	12.581	0.000	1.39e+05	1.9e+05
view	3.255e+04	1057.566	30.775	0.000	3.05e+04	3.46e+04
grade	4.433e+04	987.127	44.906	0.000	4.24e+04	4.63e+04
lat	1.483e+05	3.49e+04	4.246	0.000	7.98e+04	2.17e+05
long	-5.204e+04	2.48e+04	-2.096	0.036	-1.01e+05	-3369.335
basementyes	-2.049e+04	1541.175	-13.296	0.000	-2.35e+04	-1.75e+04
renovated_yes	4.836e+04	3577.607	13.516	0.000	4.13e+04	5.54e+04
living_vs_neighbor	-4.941e+04	2794.428	-17.681	0.000	-5.49e+04	-4.39e+04
lot_vs_neighbor	5681.0665	492.684	11.531	0.000	4715.365	6646.768
floors_1_5	1.535e+04	2386.316	6.431	0.000	1.07e+04	2e+04
floors_2_0	-7407.2962	1778.178	-4.166	0.000	-1.09e+04	-3921.922

Conclusions

- R^2: 0.828
- Adjusted R^2: 0.827
- QQ Plot: Does a good job of meeting assumption but residuals begin to slgihtly trail off at 2nd quantile
- Homoskedacicity: Meets assumptions
- Non-Statistically Significant Preditors:
  - Maintaining bedroom because a majority of values are statistically significant
  - Floors are majority statistically significant. Will alter half floors into full floors
  - Majority of zipcodes are statistically significant

Next step will be to turn floors into integer values

8.2.2 Model 5: Iteration 2 - Handling Floor Values

- In this iteration we are going to turn half floors into their integer form (ie 1.5 becomes 1). Typically, the 0.5 will signify a loft or singular room which some may not classify as an additional floor.

In [122]:
```
df_iqrp['floors'].value_counts(1)
```

Out[122]:
```
1.00000   0.50971
2.00000   0.36768
1.50000   0.08891
3.00000   0.02797
2.50000   0.00544
3.50000   0.00030
Name: floors, dtype: float64
```

- ~51% of homes are 1 story
- ~37% are 2 stories
- ~3% are 3 stories

Going to add the half story to their respective integer value (1.5 will become 1, etc...)

In [123]:
```
# Function to adjust floors to their nearest ingeger value

df_iqrp['floors'] = df_iqrp['floors'].map(lambda x: int(x))
```

In [124]:
```
# Ensure transformation worked

df_iqrp['floors'].value_counts(1)
```

Out[124]:
```
1   0.59862
2   0.37312
3   0.02827
Name: floors, dtype: float64
```

As we can see, the 0.5 values were added to the respective integer floors.

Now, we can try OHE with floors and the number of categories will be cut down to 2 (floors_2 and floors_3)

In [125]:
```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(sparse=False, drop='first')
encoder

# Seperate the columns we are going to OHE

cat_cols=[]

# Fit and transform categorical columns
# Turn matrix into DataFrame

def onehotencoder(df, cat_cols):
    encoder.fit(df[cat_cols])

    ohe_vars = encoder.transform(df[cat_cols])
    encoder.get_feature_names(cat_cols)
    cat_vars = pd.DataFrame(ohe_vars,columns=encoder.get_feature_names(

    # OLS Formula does not accept '.' so must replace these with '_'

    name_dict = {}
    for col in cat_vars.columns:
        name_dict[col]=col.replace('.','_')

    # Rename cat_vars DF with new names so it can be processed by OLS f

    cat_vars.rename(columns=name_dict, inplace=True)

    # Join OHE DataFrame back with original DateFrame
    # Ensure each DataFrame has the same number of rows
    # Begin by resetting index

    df=df.reset_index()

    df_ohe = pd.concat([df, cat_vars], axis=1, ignore_index=False)

    # Drop original column names
    cols_to_drop = cat_cols
    df_ohe.drop(cols_to_drop, axis=1, inplace=True)
    return df_ohe
```

In [126]:
```
# Use function to re-OHE with new floors values


cat_cols = ['floors', 'zipcode', 'bedrooms', 'condition']
df_ohefloors = onehotencoder(df_iqrp, cat_cols)
```

In [127]:
```
# Should see floors_2 and floors_3

df_ohefloors.head()
```

Out[127]:

ated_yes	living_vs_neighbor	lot_vs_neighbor	floors_2	floors_3	zipcode_98002	zipcode_98003	zipco
0	0.88060	1.00000	0.00000	0.00000	0.00000	0.00000	
1	1.52071	0.94803	1.00000	0.00000	0.00000	0.00000	
0	0.28309	1.24039	0.00000	0.00000	0.00000	0.00000	
0	1.44118	1.00000	0.00000	0.00000	0.00000	0.00000	
0	0.93333	1.07690	0.00000	0.00000	0.00000	0.00000	

In [128]:
```
# Confirm that we do not have any null values in our DataFrame

df_ohefloors.isna().sum().any()
```

Out[128]: False

In [129]:
```
# Create list of columns excluding id, date, and price because that is
# Remove additional indices created by resetting the index
# Removing lat and long because zipcode acts as the proxy for location

x_targs = df_ohefloors.columns
x_targs = list(x_targs)
x_targs = [x for x in x_targs if x not in ('id', 'date', 'price', 'inde
x_targs
```

Out[129]:
```
['sqft_living',
 'waterfront',
 'view',
 'grade',
 'basementyes',
 'renovated_yes',
 'living_vs_neighbor',
 'lot_vs_neighbor',
 'floors_2',
 'floors_3',
 'zipcode_98002',
 'zipcode_98003',
 '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_98168',
 'zipcode_98177',
 'zipcode_98178',
 'zipcode_98188',
 'zipcode_98198',
 'zipcode_98199',
 'bedrooms_2',
 'bedrooms_3',
 'bedrooms_4',
 'bedrooms_5',
 'bedrooms_6',
 'bedrooms_7',
 'bedrooms_8',
 'bedrooms_9',
 'bedrooms_10',
 'bedrooms_11',
 'condition_2',
 'condition_3',
 'condition_4',
 'condition_5']
```

Targets look correct with new OHE variables and no excess indices.

Next step is to run Model 5 Iteration 2 and check if floors have a significant p-value

In [130]:
```
model_ohefloors = model_summary(df_ohefloors, x_targs, 'price')
sked_show(df_ohefloors, x_targs, model_ohefloors)
```

floors_3	-6.452e+04	4129.581	-15.623	0.000	-7.26e+04	-5.64e+04
zipcode_98002	1.167e+04	7660.497	1.523	0.128	-3346.308	2.67e+04
zipcode_98003	-4413.3594	6900.304	-0.640	0.522	-1.79e+04	9111.800
zipcode_98004	5.169e+05	8431.130	61.312	0.000	5e+05	5.33e+05
zipcode_98005	3.309e+05	8446.732	39.169	0.000	3.14e+05	3.47e+05
zipcode_98006	2.714e+05	6362.838	42.655	0.000	2.59e+05	2.84e+05
zipcode_98007	2.625e+05	8732.050	30.061	0.000	2.45e+05	2.8e+05
zipcode_98008	2.44e+05	6999.033	34.861	0.000	2.3e+05	2.58e+05
zipcode_98010	9.473e+04	9781.390	9.685	0.000	7.56e+04	1.14e+05
zipcode_98011	1.474e+05	7690.627	19.160	0.000	1.32e+05	1.62e+05
zipcode_98014	1.249e+05	9099.142	13.730	0.000	1.07e+05	1.43e+05
zipcode_98019	1.06e+05	7742.104	13.693	0.000	9.08e+04	1.21e+05

As we can see, floors are statistically significant now. Floors_2 and Floors_3 are both statstically significant with price

Conclusions

- R^2: 0.827
- Adjusted R^2: 0.827
- QQ Plot: Does a good job of meeting assumption but residuals begin to slightly trail off at 2nd quantile
- Homoskedacicity: Meets assumptions
- Non-Statistically Significant Preditors:
  - Maintaining bedroom because a majority of values are statistically significant
  - Majority of zipcodes are statistically significant

- **Model 5 Iteration 2 will act as our final model**
  - It does not have any statistically insignificant predictor values except for a handful of zipcodes which are OHE variables
  - R^2 of 0.83 which means that the predictor variables explain ~83% of the variation in the target variable
  - Meets all assumptions:
    - Homoskedacicity
    - Predictor variables have a linear relationship with the target variable
    - No multicollinearity between predictor variables

9 Intepretation

- Share what the results of our multiple linear regression model mean for our stakeholders
- Support these points with visualizations from the dataset

9.1 Standardize data for intepretation

For our data to be on the same unit, we must use a Standard Scaler so we can interpret coefficients on the same magnitude and unit. Important to note that **standardizing** data will change the range and distribution of the data. Given that variables will be on the same scale, can compare a unit 1 increase evenly across predictors

In [131]:
```
df_zf = df_ohefloors.copy()
```

In [132]:
```
# Predictor values, need to evaluate which are appropriate to scale wit
# standard scaler

x_scale = df_zf.columns
x_scale
```

Out[132]:
```
Index(['level_0', 'index', 'id', 'date', 'price', 'sqft_living', 'waterfr
ont',
       'view', 'grade', 'lat', 'long', 'basementyes', 'renovated_yes',
       'living_vs_neighbor', 'lot_vs_neighbor', 'floors_2', 'floors_3',
       'zipcode_98002', 'zipcode_98003', 'zipcode_98004', 'zipcode_9800
5',
       'zipcode_98006', 'zipcode_98007', 'zipcode_98008', 'zipcode_9801
0',
       'zipcode_98011', 'zipcode_98014', 'zipcode_98019', 'zipcode_9802
2',
       'zipcode_98023', 'zipcode_98024', 'zipcode_98027', 'zipcode_9802
8',
       'zipcode_98029', 'zipcode_98030', 'zipcode_98031', 'zipcode_9803
2',
       'zipcode_98033', 'zipcode_98034', 'zipcode_98038', 'zipcode_9803
9',
       'zipcode_98040', 'zipcode_98042', 'zipcode_98045', 'zipcode_9805
2',
       'zipcode_98053', 'zipcode_98055', 'zipcode_98056', 'zipcode_9805
8',
       'zipcode_98059', 'zipcode_98065', 'zipcode_98070', 'zipcode_9807
2',
       'zipcode_98074', 'zipcode_98075', 'zipcode_98077', 'zipcode_9809
2',
       'zipcode_98102', 'zipcode_98103', 'zipcode_98105', 'zipcode_9810
6',
       'zipcode_98107', 'zipcode_98108', 'zipcode_98109', 'zipcode_9811
2',
       'zipcode_98115', 'zipcode_98116', 'zipcode_98117', 'zipcode_9811
8',
       'zipcode_98119', 'zipcode_98122', 'zipcode_98125', 'zipcode_9812
6',
       'zipcode_98133', 'zipcode_98136', 'zipcode_98144', 'zipcode_9814
6',
       'zipcode_98148', 'zipcode_98155', 'zipcode_98166', 'zipcode_9816
8',
       'zipcode_98177', 'zipcode_98178', 'zipcode_98188', 'zipcode_9819
8',
       'zipcode_98199', 'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedroo
ms_5',
       'bedrooms_6', 'bedrooms_7', 'bedrooms_8', 'bedrooms_9', 'bedrooms_
10',
       'bedrooms_11', 'condition_2', 'condition_3', 'condition_4',
       'condition_5'],
      dtype='object')
```

We don't scale our target variable (price) for interpretation because then conclusions will changes in prices based on comparisons of standard deviation

In [133]:
```
# Not going to scale OHE variables because they are already implicity s
# Not going to scale binary variables because they are already implicit

x_scale = [x for x in x_targs if x in ('sqft_living', 'view', 'grade',
                                       'lot_vs_neighbor')]
```

In [134]:
```
x_scale
```

Out[134]:
```
['sqft_living', 'view', 'grade', 'living_vs_neighbor', 'lot_vs_neighbor']
```

We are ready to scale all of our numeric data

In [135]:
```
df_zf=df_zf.drop(['level_0', 'index', 'date', 'lat', 'long'], axis=1)
```

Dropping lat, and long because they are difficult to interpret for our purposes and we have zipcode to act as a proxy for location

In [136]:
```
# Fit and transform original values into scaled values

df_zf[x_scale] = scaler.fit_transform(df_zf[x_scale])
df_zf.describe()
```

Out[136]:

	id	price	sqft_living	waterfront	view	grade	base
count	20235.00000	20235.00000	20235.00000	20235.00000	20235.00000	20235.00000	202
mean	4605299135.81389	477281.03736	0.00000	0.00247	-0.00000	-0.00000	
std	2877559932.98793	206564.79018	1.00002	0.04965	1.00002	1.00002	
min	1000102.00000	78000.00000	-2.07603	0.00000	-0.26888	-4.38492	
25%	2136600137.50000	316000.00000	-0.74516	0.00000	-0.26888	-0.51621	
50%	3905081500.00000	439000.00000	-0.15079	0.00000	-0.26888	-0.51621	
75%	7338220140.00000	600000.00000	0.59864	0.00000	-0.26888	0.45097	
max	9900000190.00000	1120000.00000	7.11087	1.00000	5.99752	4.31968	

As we can observe, the column that we have used the standard scaler to fit and transform now have a mean of 0 and a standard deviation of 1. They have been converted into Z-scores which allows us to compare predictors against price and not have to consider the impact of magnitude and units. For example, a 1 bedroom increase and a 1 sqft_living increase now represent equal changes to the existing home

9.2 View which predictors make the most significant impact on home price

- When evaluating coefficients, we are looking at two things:

1. As we increase this predictor variable, does it make a positive or negative impact on price (+/-)
2. How large is the coefficient. As we increase the predictor variable by a single unit, how much does that impact price. In the case of Z-scores, a 1 unit increase means a 1 standard deviation increase. For binary variables, it means either this feature is present or it is not
3. Coefficient values are interpreted as 'with respect for the intercept (baseline)'

In [137]:
```
1 x_targs
```

Out[137]:
```
['sqft_living',
 'waterfront',
 'view',
 'grade',
 'basementyes',
 'renovated_yes',
 'living_vs_neighbor',
 'lot_vs_neighbor',
 'floors_2',
 'floors_3',
 'zipcode_98002',
 'zipcode_98003',
 '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_98168',
'zipcode_98177',
'zipcode_98178',
'zipcode_98188',
'zipcode_98198',
'zipcode_98199',
'bedrooms_2',
'bedrooms_3',
'bedrooms_4',
'bedrooms_5',
'bedrooms_6',
'bedrooms_7',
'bedrooms_8',
'bedrooms_9',
'bedrooms_10',
'bedrooms_11',
'condition_2',
'condition_3',
'condition_4',
'condition_5']
```

In [138]:
```
# Run model to get parameter (coefficient) values from statsmodels

model_final = model_summary(df_zf, x_targs, 'price')
sked_show(df_zf, x_targs, model_final)
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.827
Model:	OLS	**Adj. R-squared:**	0.827
Method:	Least Squares	**F-statistic:**	1039.
Date:	Wed, 05 May 2021	**Prob (F-statistic):**	0.00
Time:	14:11:05	**Log-Likelihood:**	-2.5858e+05
No. Observations:	20235	**AIC:**	5.173e+05
Df Residuals:	20141	**BIC:**	5.181e+05
Df Model:	93		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	2.431e+05	1.82e+04	13.339	0.000	2.07e+05	2.79e+05

Confirmed that assumptions and values look the same

Scaling data does not make a difference

In [139]:
```
# Use statsmodels .params to extract coefficient values into a DataFram
# Include coefficient and absolute coefficient for analysis and visuali

df_coeff=pd.DataFrame({'coeff': model_final.params, 'abs_coeff': abs(mo
```

In [140]:
```
1 df_coeff
```

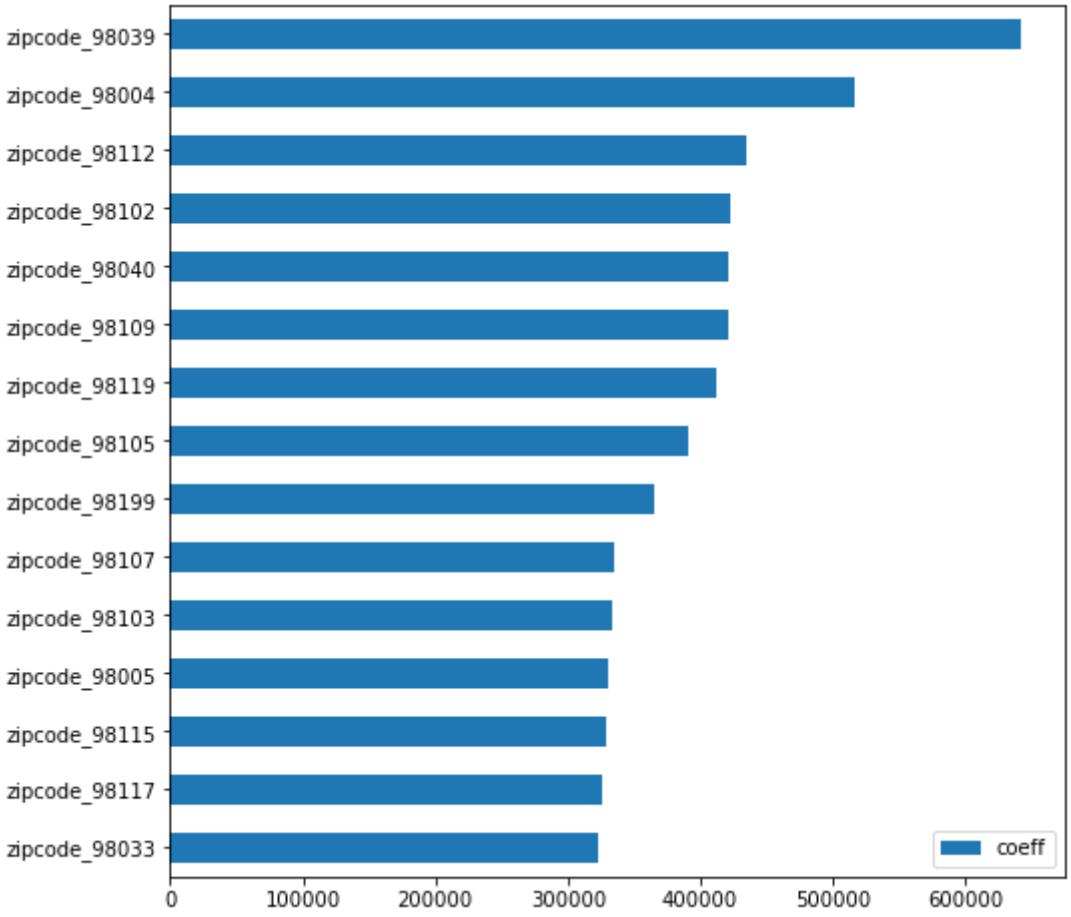
Out[140]:

	coeff	abs_coeff
Intercept	243052.98038	243052.98038
sqft_living	105564.12643	105564.12643
waterfront	164587.53385	164587.53385
view	20736.15964	20736.15964
grade	45704.32075	45704.32075
...	...	...
bedrooms_11	11786.85452	11786.85452
condition_2	48276.34358	48276.34358
condition_3	66106.77111	66106.77111
condition_4	87080.87703	87080.87703
condition_5	122763.55463	122763.55463

94 rows × 2 columns

In [141]:
```
# Create horizontal bar plot to view the 15 most impactful predictor va
# absolute perspective

df_coeff.drop('Intercept', axis=0).sort_values(by='abs_coeff').tail(15)
```

Out[141]: <AxesSubplot:>




```
As we can see, the 15 most impactful predictor variables are all zip
codes. With respect to the intercept, some of the zipcodes can explain
~$600,000 in price.

98039, 98004, and 98112 are the top 3 most expensive zip codes. These
zips may be closest to downtown Seattle. Check that later

Almost all zip codes are statistically significant as we can see they
have a major impact on price
```

9.2.1 Digging Deepr Into Zip codes

- Evaluating how the top zipcodes compare to the average zip code in Kings County

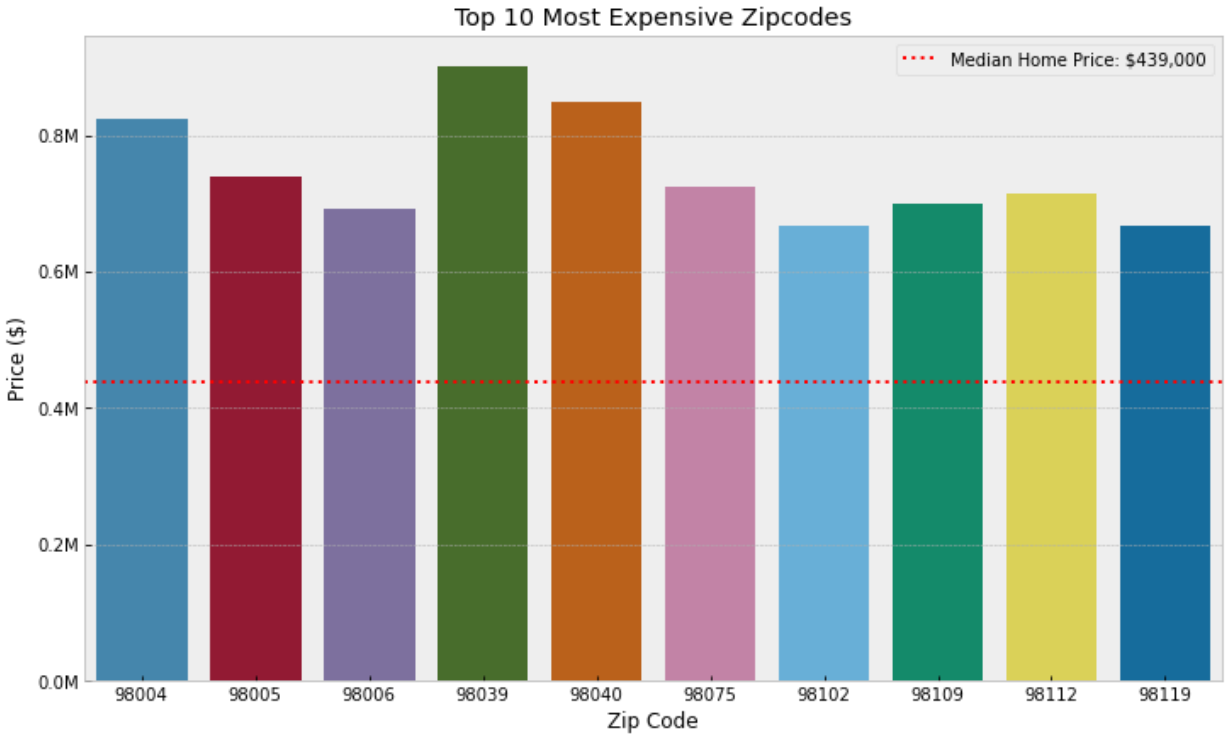
In [142]:
```
# Gather data on the top 10 zipcodes in Kings County

top_10_zips = df_iqrp.groupby('zipcode')['price'].median().sort_values(
top_10_zips
```

Out[142]:

	zipcode	price
0	98039	901250.00000
1	98040	850000.00000
2	98004	825000.00000
3	98005	740000.00000
4	98075	725393.00000
5	98112	714250.00000
6	98109	700000.00000
7	98006	691100.00000
8	98119	667500.00000
9	98102	667475.00000

Using median in the groupby statement to handle outlier values. The dataset does not contain predicted outliers but still useful to use median to reduce 'noise'

In [143]:
```
# Comparing top 10 zip codes by median to Kings Country overall median

# Function to display values in millions
from matplotlib.ticker import FuncFormatter

def millions(x, pos):
    return '%1.1fM' % (x * 1e-6)

# Kings County median home price
median = int(df_iqrp['price'].median())

# Construct visualization
with plt.style.context('bmh'):
    fig, ax = plt.subplots(figsize=(12,7))
    sns.barplot(data=top_10_zips, x='zipcode', y='price',ax=ax)
    ax.set_xlabel('Zip Code')
    ax.set_ylabel('Price ($)')
    ax.set_title('Top 10 Most Expensive Zipcodes')
    ax.axhline(median, color='r', ls=':', label=f'Median Home Price: ${
    ax.legend()

    formatter = FuncFormatter(millions)
    ax.yaxis.set_major_formatter(formatter)
```



The median home price in King County is $439,000. The top 10 most expensive zipcodes exceed that value by a minimum of ~$250,000 up to ~ $450,000. The most expensive zip codes tend to be situated around urban Seattle.

9.3 Evaluating the predictors not based on location

- Evaluate how the predictor values outside of zip code impact home price

In [144]:
```
# Create list of features excluding zip codes

indices = df_coeff.index
non_zips = []
for ind in indices:
    if ind.startswith('zip'):
        pass
    else:
        non_zips.append(ind)
```

In [145]:
```
# These are the predictors that are not zip codes

non_zips
```

Out[145]:
```
['Intercept',
 'sqft_living',
 'waterfront',
 'view',
 'grade',
 'basementyes',
 'renovated_yes',
 'living_vs_neighbor',
 'lot_vs_neighbor',
 'floors_2',
 'floors_3',
 'bedrooms_2',
 'bedrooms_3',
 'bedrooms_4',
 'bedrooms_5',
 'bedrooms_6',
 'bedrooms_7',
 'bedrooms_8',
 'bedrooms_9',
 'bedrooms_10',
 'bedrooms_11',
 'condition_2',
 'condition_3',
 'condition_4',
 'condition_5']
```

In [146]:
```
# Select the non-zip rows

df_coeff2 = df_coeff.loc[non_zips]
```

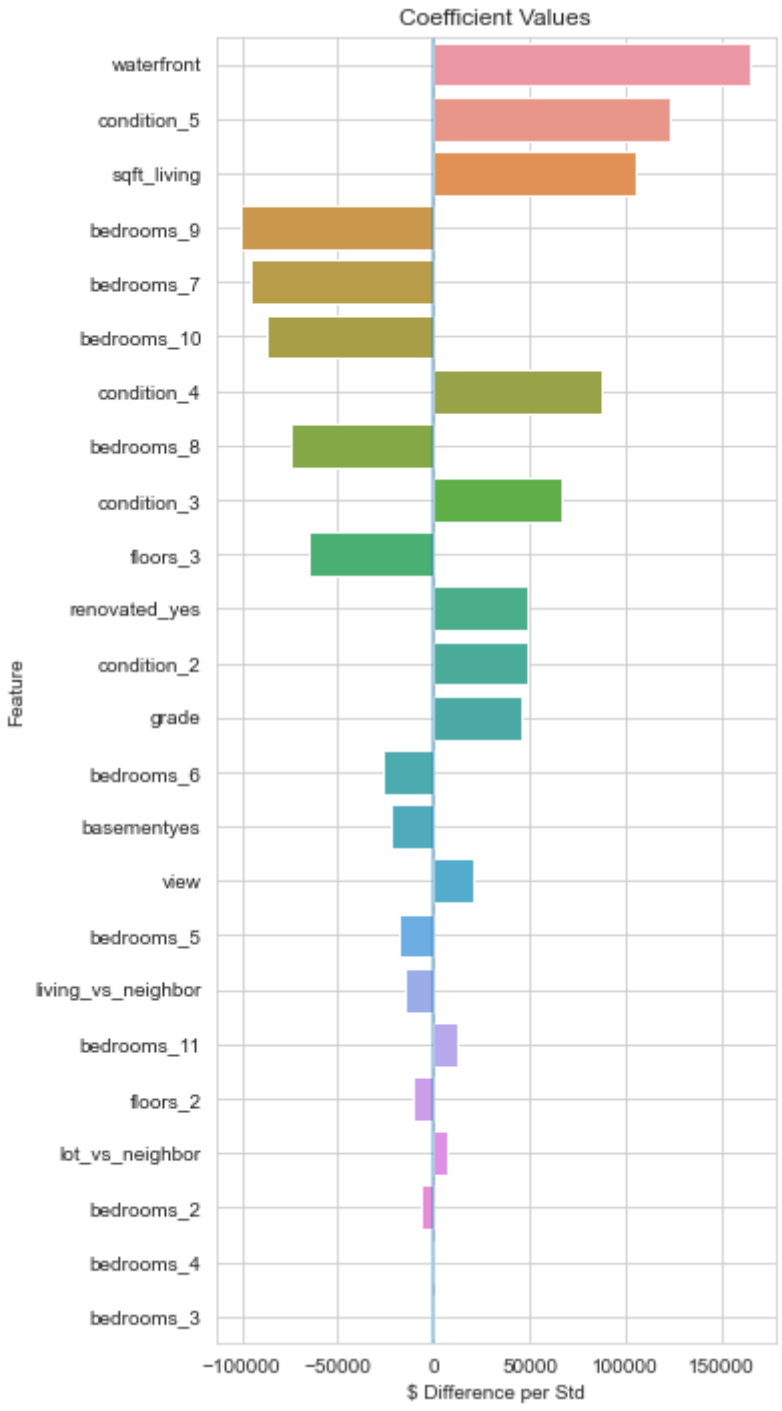
In [147]:
```
# Remove intercept because it is not a coefficient value

df_coeff2.drop('Intercept', axis=0, inplace=True)
```

In [148]:
```
# Reset index so that we can sort by index for the seaborn plot

df_coeff2=df_coeff2.reset_index()
```

In [149]:

```
# Sort in descending order so the visualization will be easier to conce

df_coeff2=df_coeff2.sort_values(by='abs_coeff', ascending=False)
```

In [150]:
```
sns.set_style('whitegrid')
fig, ax = plt.subplots(figsize=(5,12))


sns.barplot(data=df_coeff2, y='index', x='coeff', orient='h')
ax.set_title('Coefficient Values')
ax.set_xlabel('$ Difference per Std')
ax.set_ylabel('Feature')


ax.axvline(0, alpha=0.5)
ax.grid(True)
```



Based on the graph above, waterfront (binary variable) is the most important indicator of price. If a

home has a waterfront view, it is estimated to cost ~$150,000 more. Next comes condition, all condition coefficients have positive values. The greater the condition, the greater the estimated home value will be. Condition represents the overall state of the house. In my estimation, for a home to have a high condition value, it must be constructed with quality materials, have structural integrity, be generally well kept, and look visually appealing from the outside. Next is square foot living, the larger the home, the higher the price. A 1 Standard Deviation increase in sqft_living results in a $100,000 price increase. This means that that compared to the mean, the home that is 1 standard deviation above the mean sqft_living will cost $100,000 more.

Bedrooms 9,7, and 10 all have negative impacts on price. As we observed earlier, bedrooms did not have a linear relationship with price which is why we needed to OHE them.

All floors with respect to 1 floor have a negative relationship with price. This means that if you add a second or third floor your home price will be lower

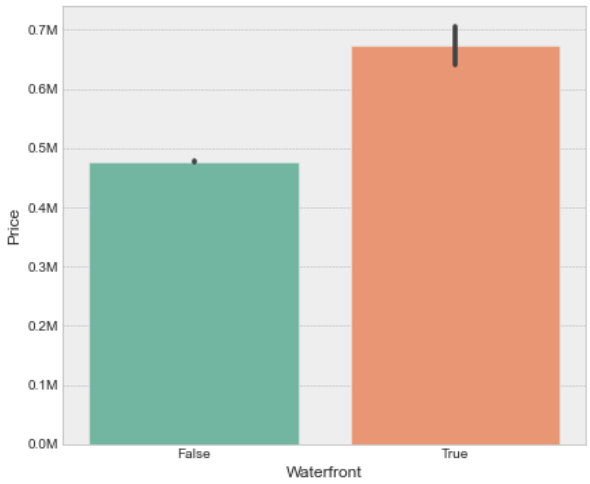
Grade has a positive impact on price, which is how the King City council determines the quality of the home. Measures if a home needs maintance or is in good condition.

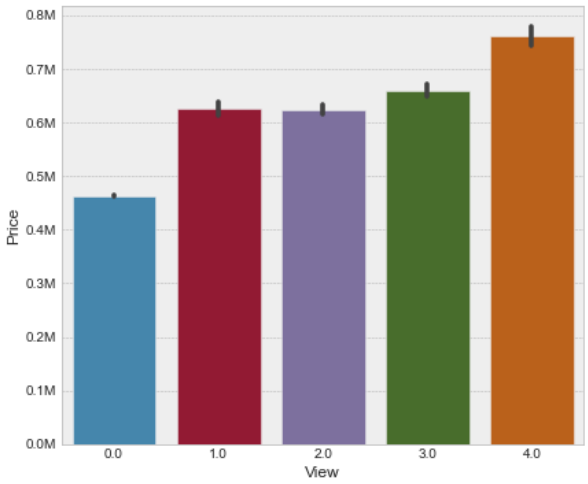
Homes with basements have a lower price on average than those that do not.

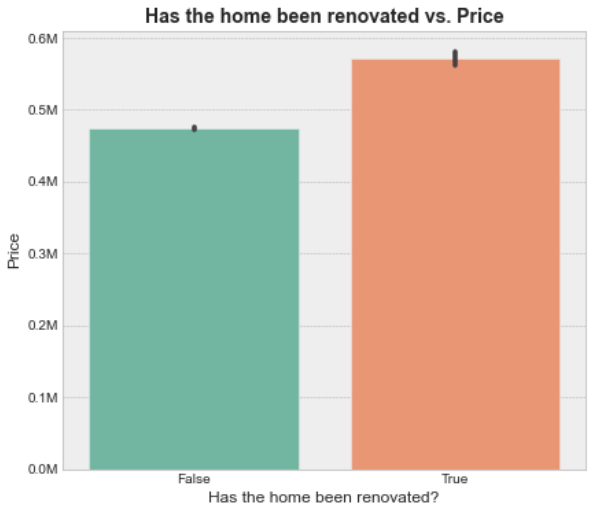
View has a positive relationship with price.

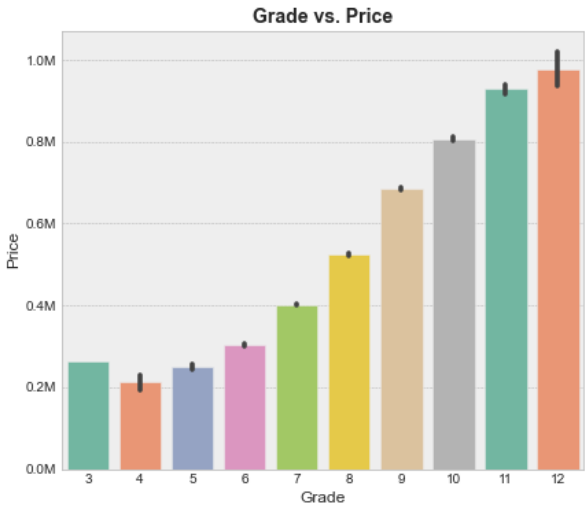
In [158]:

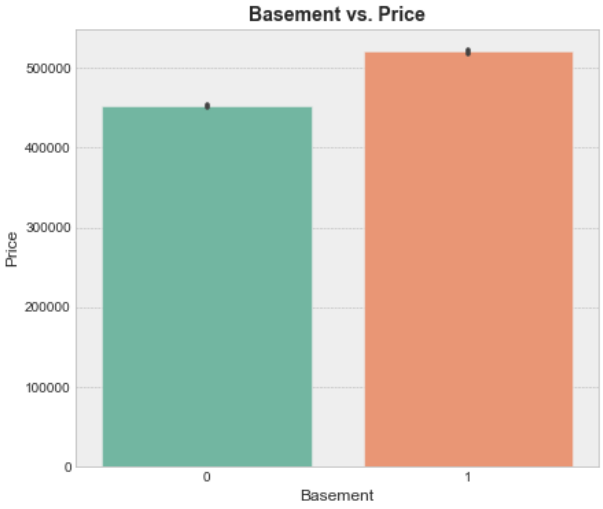
```
with plt.style.context('bmh'):

    fig, axs=plt.subplots(nrows=3, ncols=2, figsize=(15,20))

    sns.barplot(data=df_ohefloors, x='waterfront', y='price', ax=axs[0,
    sns.barplot(data=df_ohefloors, x='view', y='price', ax=axs[0,1], ci
    sns.barplot(data=df_ohefloors, x='renovated_yes', y='price', ax=axs
    sns.barplot(data=df_ohefloors, x='grade', y='price',ax=axs[1,1], pa
    sns.barplot(data=df_ohefloors, x='basementyes', y='price',ax=axs[2,
    sns.barplot(data=df_iqrp, x='condition', y='price',ax=axs[2,1], pal


    tf_labs = ['False', 'True']

    axs[0,0].set_title('Waterfront vs. Price', fontsize=14, fontweight=
    axs[0,0].set_xlabel('Waterfront')
    axs[0,0].set_ylabel('Price')
    axs[0,0].set_xticklabels(tf_labs)

    axs[0,1].set_title('View vs. Price',fontsize=14, fontweight='bold')
    axs[0,1].set_xlabel('View')
    axs[0,1].set_ylabel('Price')

    axs[1,0].set_title('Has the home been renovated vs. Price', fontsiz
    axs[1,0].set_xlabel('Has the home been renovated?')
    axs[1,0].set_ylabel('Price')
    axs[1,0].set_xticklabels(tf_labs)

    axs[1,1].set_title('Grade vs. Price', fontsize=14, fontweight='bold
    axs[1,1].set_xlabel('Grade')
    axs[1,1].set_ylabel('Price')

    axs[2,0].set_title('Basement vs. Price', fontsize=14, fontweight='b
    axs[2,0].set_xlabel('Basement')
    axs[2,0].set_ylabel('Price')

    axs[2,1].set_title('Condition vs. Price', fontsize=14, fontweight='
    axs[2,1].set_xlabel('Condition')
    axs[2,1].set_ylabel('Price')


    formatter = FuncFormatter(millions)
    axs[0,0].yaxis.set_major_formatter(formatter)
    axs[0,1].yaxis.set_major_formatter(formatter)
    axs[1,0].yaxis.set_major_formatter(formatter)
    axs[1,1].yaxis.set_major_formatter(formatter)
```

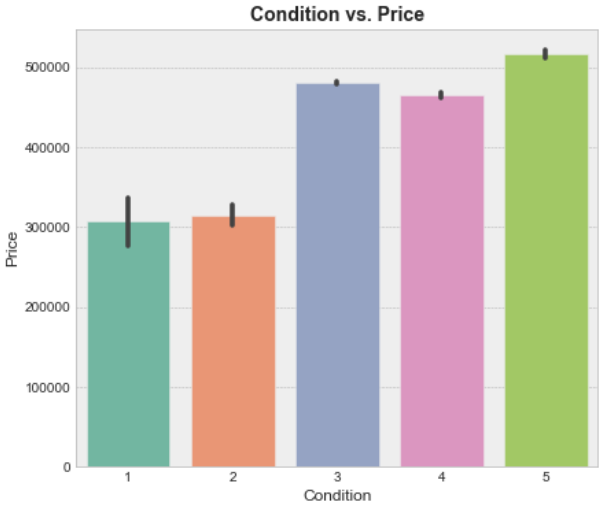

0.7M
0.6M
0.5M
0.4M
0.3M
0.2M
0.1M
0.0M
Price
False
True
Waterfront


0.8M
0.7M
0.6M
0.5M
0.4M
0.3M
0.2M
0.1M
0.0M
Price
0.0
1.0
2.0
3.0
4.0
View


Has the home been renovated vs. Price
0.6M
0.5M
0.4M
0.3M
0.2M
0.1M
0.0M
Price
False
True
Has the home been renovated?


Grade vs. Price
1.0M
0.8M
0.6M
0.4M
0.2M
0.0M
Price
3
4
5
6
7
8
9
10
11
12
Grade


Basement vs. Price
500000
400000
300000
200000
100000
0
Price
0
1
Basement

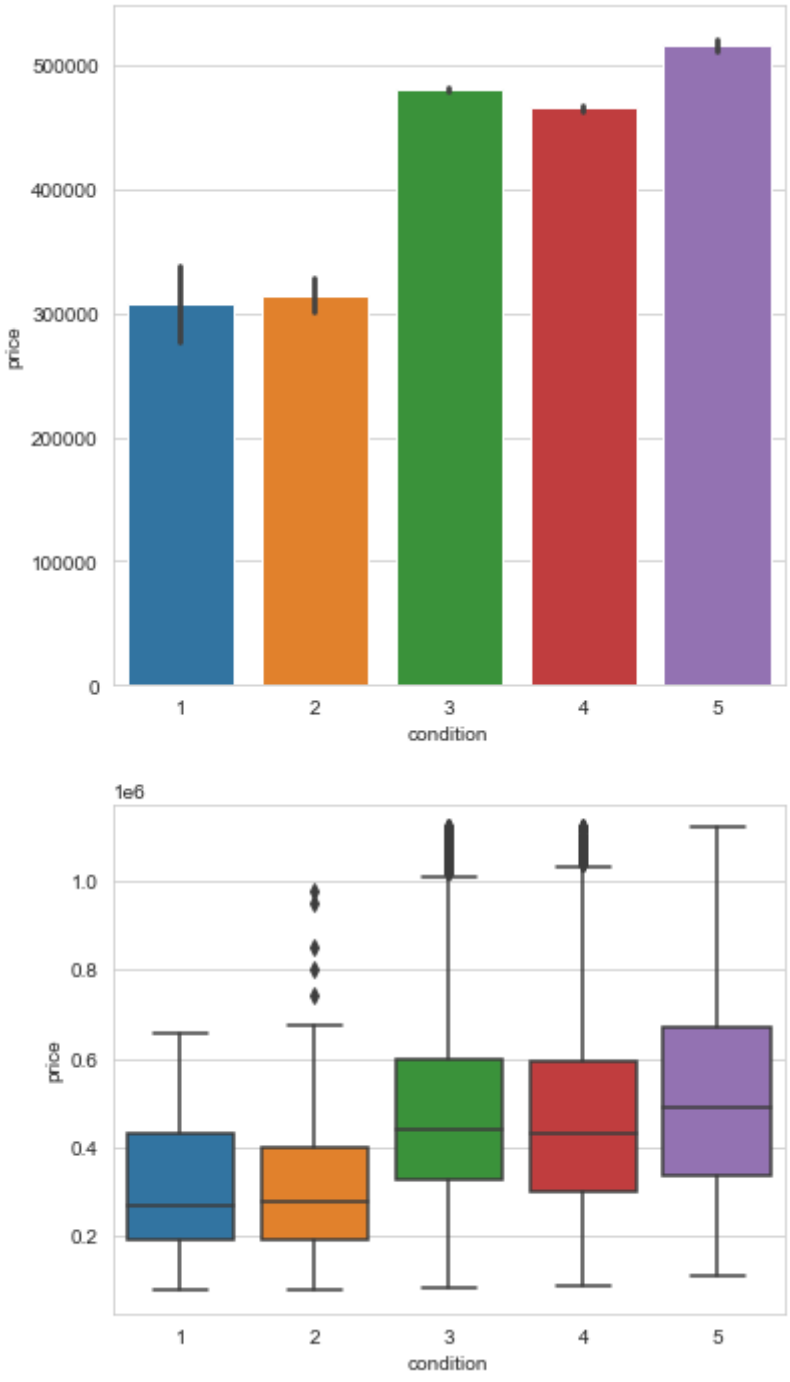

Condition vs. Price
500000
400000
300000
200000
100000
0
Price
1
2
3
4
5
Condition

- Waterfront homes are more expensive than non-waterfront homes
- View increases in a semi linear fashion
- Homes that have been renovated are more expensive on average
- Condition has a linear relationship with price
- Homes with basements are marginally more expensive than those without
- Condition does not have a linear relationship with price becasue it increases up to 3 then decreases and increases again

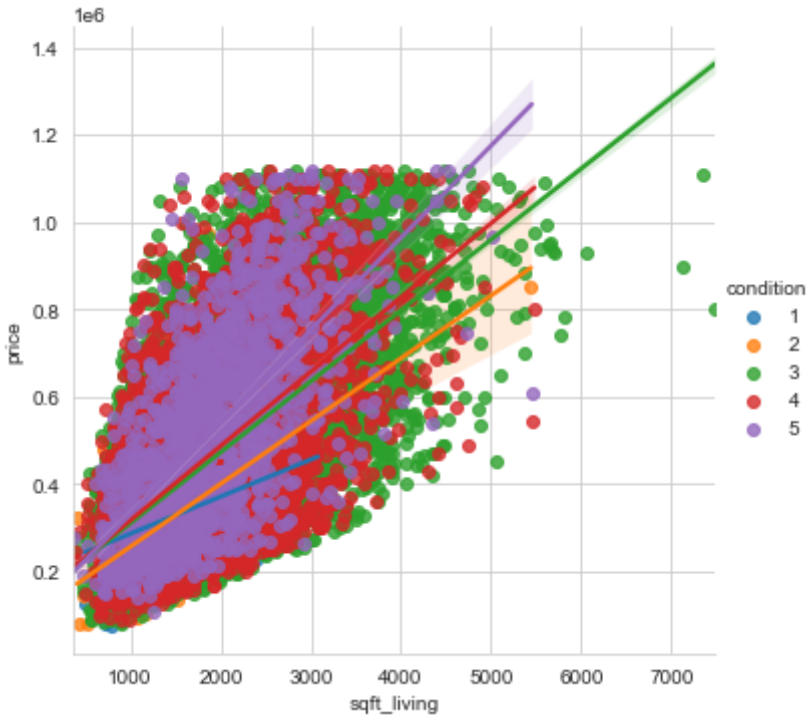
10 Models for Presentation

In [152]:
```
1 fig, axs=plt.subplots(nrows=2, ncols=1, figsize=(6,12), gridspec_kw={'h
2
3 sns.barplot(data=df_iqrp, x='condition', y='price', ax=axs[0], ci=68)
4 sns.boxplot(data=df_iqrp, x='condition', y='price', ax=axs[1])
```

Out[152]: <AxesSubplot:xlabel='condition', ylabel='price'>

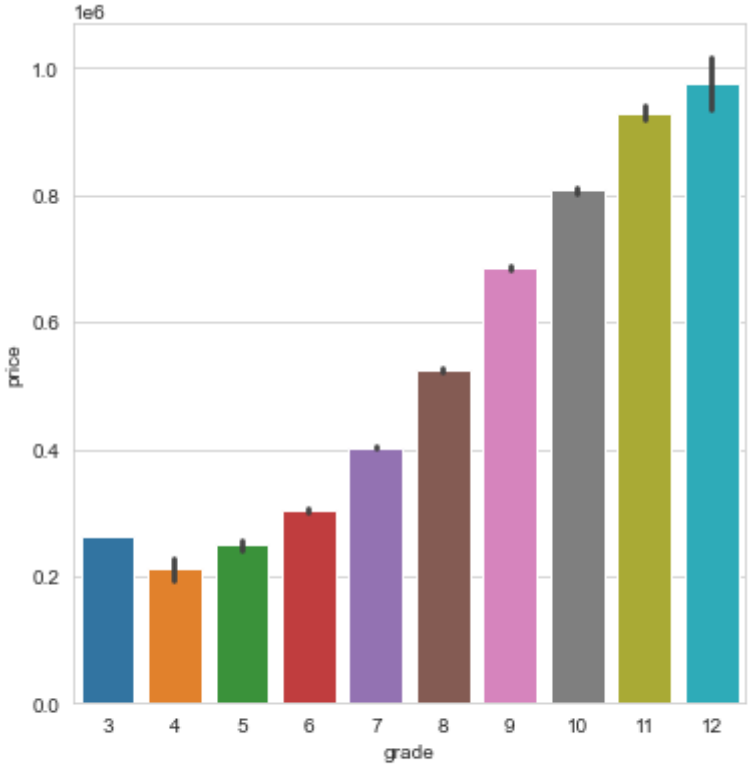


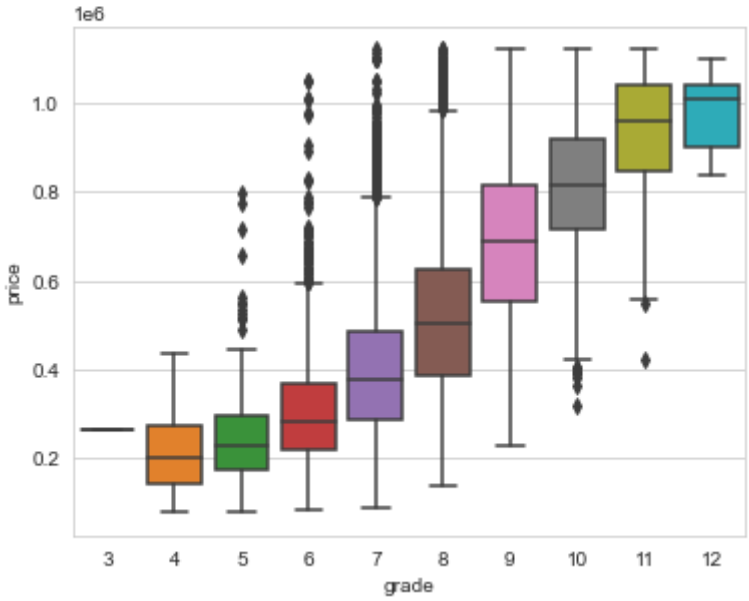
In [153]:
```
1 g = sns.lmplot(data=df_iqrp, x="sqft_living", y="price", hue="condition
```



In [154]:
```
fig, axs=plt.subplots(nrows=2, ncols=1, figsize=(6,12), gridspec_kw={'h

sns.barplot(data=df_iqrp, x='grade', y='price', ax=axs[0], ci=68)
sns.boxplot(data=df_iqrp, x='grade', y='price', ax=axs[1])
```

Out[154]: <AxesSubplot:xlabel='grade', ylabel='price'>





In [ ]:
```
1
```