1 Final Project Submission

Please fill out:

- Student name: **Brian Bentson**
- Student pace: self paced / part time / full time: **Full Time**
- Scheduled project review date/time: 4/29/21 @ 4pm CST
- Instructor name: **James Irving**
- Blog post URL:
- Video of 5-min Non-Technical Presentation:

2 Table of Contents

Click to jump to matching Markdown Header.

3 Introduction

3.1 Business Statement

Most people's largest asset is their home which acts as the foundation for their net worth. Therefore, it is imperative that the value of this asset improves over time through either property value inflation or smart renovations. Since property values are largely based on location and current market conditions, which are outside of your control, renovations are the only controllable factor when trying to improve a homes value. In this analysis, I will explore which factors in a house are most correlated to higher value by looking at historical sales of homes in King County, Washington. I will then make recommendations to prospective home rennovators to help them make smart decisions to improve their homes value.

3.2 Analysis Methodology

I will be analyzing historic home sales from King County, Washington in order to see which factors affect home price and how a model can be built to predict good estimates for home listing prices. This model will give insights into what a current home owner could do in order to improve their home value. I will focus only on features which a home owner has control over.

4 Obtain

4.1 Import Packages

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.style as style
import seaborn as sns
plt.style.use('fivethirtyeight')

import statsmodels.api as sm
from statsmodels.formula.api import ols
import scipy.stats as stats
import statsmodels.stats.api as sms
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from statsmodels.stats.outliers_influence import variance_inflatio
```

In [2]:

```
pd.set_option("display.max_columns", 30)
pd.options.display.float_format = '{:,}'.format
```

4.2 Global Functions

In [4]:

```
#function to look at plots and stats of column with or without out
def get_plots_updated(df, x_col, y_col='price', outlier='none'):

    """This function takes in a dataframe and a column, removes ou
       standard deviations or iqr and produces a histogram, scatte
       boxplot of the values with descriptive statistics"""

    #plots for std
    if outlier == 'std':
        #create variables
        col_mean = df[x_col].mean()
        col_std = df[x_col].std()
        upper_thresh_std = col_mean + 3*col_std
        lower_thresh_std = col_mean - 3*col_std

        #create new df
        idx_std_outliers = (df[x_col] > lower_thresh_std) & (df[x_
        clean_df = df.loc[idx_std_outliers]


    elif outlier == 'iqr':
        #create variables
        q25 = df[x_col].quantile(0.25)
        q75 = df[x_col].quantile(0.75)
        iqr = q75-q25
        upper_thresh_iqr = q75 + 1.5*iqr
        lower_thresh_iqr = q25 - 1.5*iqr

        #create new df
        idx_iqr_outliers = (df[x_col] > lower_thresh_iqr) & (df[x_
        clean_df = df.loc[idx_iqr_outliers]


    elif outlier == 'none':
        df_clean = df
        #plots


    fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(15,10));
    histogram = df_clean[x_col].hist(ax=ax[0,0]);
    ax[0,0].set_title(f'Distribution of {x_col}');

    scatter = df_clean.plot(kind='scatter', x=x_col, y=y_col,ax=ax
    ax[0,1].set_title(f'{y_col} vs {x_col}');

    boxplot = df_clean.boxplot(column=x_col, ax=ax[1,0]);
```

```
    ax[1,0].set_title(f'Boxplot of {x_col}');

    sm.graphics.qqplot(df_clean[x_col], dist=stats.norm, line='45'
    ax[1,1].set_title(f'QQ plot of {x_col}');

    #stats
    desc_stats = df_clean[x_col].describe()
    plt.tight_layout()


    print(desc_stats)
    plt.show()

    return
```

In [5]:

```
#function to look at plots and stats of column with or without out
def get_plots(df, x_col, y_col='price', outlier='none'):

    """This function takes in a dataframe and a column, removes ou
       standard deviations or iqr and produces a histogram, scatte
       boxplot of the values with descriptive statistics"""

    #plots for std
    if outlier == 'std':
        #create variables
        col_mean = df[x_col].mean()
        col_std = df[x_col].std()
        upper_thresh_std = col_mean + 3*col_std
        lower_thresh_std = col_mean - 3*col_std

        #create new df
        idx_std_outliers = (df[x_col] > lower_thresh_std) & (df[x_
        std_df = df.loc[idx_std_outliers]

        #plots
        fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(15,10));
        histogram = std_df[x_col].hist(ax=ax[0,0]);
        ax[0,0].set_title(f'Distribution of {x_col}');

        scatter = std_df.plot(kind='scatter', x=x_col, y=y_col,ax=
        ax[0,1].set_title(f'{y_col} vs {x_col}');

        boxplot = std_df.boxplot(column=x_col, ax=ax[1,0]);
        ax[1,0].set_title(f'Boxplot of {x_col}');

        sm.graphics.qqplot(std_df[x_col], dist=stats.norm, line='4
        ax[1,1].set_title(f'QQ plot of {x_col}');

        #stats
        rows_removed = len(df) - len(std_df)
```

```
        print(f'The number of rows removed is {rows_removed}')
        desc_stats = std_df[x_col].describe()
        plt.tight_layout()

    elif outlier == 'iqr':
        #create variables
        q25 = df[x_col].quantile(0.25)
        q75 = df[x_col].quantile(0.75)
        iqr = q75-q25
        upper_thresh_iqr = q75 + 1.5*iqr
        lower_thresh_iqr = q25 - 1.5*iqr

        #create new df
        idx_iqr_outliers = (df[x_col] > lower_thresh_iqr) & (df[x_
        iqr_df = df.loc[idx_iqr_outliers]

        #plots
        fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(15,10));
        histogram = iqr_df[x_col].hist(ax=ax[0,0]);
        ax[0,0].set_title(f'Distribution of {x_col}');

        scatter = iqr_df.plot(kind='scatter', x=x_col, y=y_col,ax=
        ax[0,1].set_title(f'{y_col} vs {x_col}');

        boxplot = iqr_df.boxplot(column=x_col, ax=ax[1,0]);
        ax[1,0].set_title(f'Boxplot of {x_col}');

        sm.graphics.qqplot(iqr_df[x_col], dist=stats.norm, line='4
        ax[1,1].set_title(f'QQ plot of {x_col}');

        #stats
        rows_removed = len(df) - len(iqr_df)
        print(f'The number of rows removed is {rows_removed}')
        desc_stats = df[x_col].describe()
        plt.tight_layout()

    elif outlier == 'none':
        #plots
        fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(15,10));
        histogram = df[x_col].hist(ax=ax[0,0]);
        ax[0,0].set_title(f'Distribution of {x_col}');

        scatter = df.plot(kind='scatter', x=x_col, y=y_col,ax=ax[0

        ax[0,1].set_title(f'{y_col} vs {x_col}');

        boxplot = df.boxplot(column=x_col, ax=ax[1,0]);
        ax[1,0].set_title(f'Boxplot of {x_col}');

        sm.graphics.qqplot(df[x_col], dist=stats.norm, line='45',
        ax[1,1].set_title(f'QQ plot of {x_col}');
```

```
        #stats
        desc_stats = df[x_col].describe()
        plt.tight_layout()


    print(desc_stats)
    plt.show()

    return
```

In [6]:

```
#function to preprocess and create a new model
def fit_new_model(df, x_cols=None, y_col=None, norm=False, diagnos
    '''This function takes in a dataframe, a list of independent a
       variables and whether or not you want to normalize the colu
       output is a multiple linear regression model with checks fo
       multicollinearity, normality and homoscedasticity.'''

    #step 1: normalize columns
    if norm == True:
        for col in x_cols:
            df[col] = (df[col] - df[col].mean())/df[col].std()
        #display the normalized df
        display(df.round(2).head())
        print('\n')
    else:
        #display the df
        display(df.round(2).head())
        print('\n')

    #step 2: create model

    #set up model parameters
    x_cols = x_cols
    outcome = y_col
    predictors = '+'.join(x_cols)
    formula = outcome + '~' + predictors
    #fit the model
    model = ols(formula=formula, data=df).fit()
    print(model.summary())
    print('\n')

    if diagnose == True:
        #step 3: check multicollinearity
        print('VIF Multicollinearity Test Results')
        print('=====================================================
        #run VIF test
        X = df[x_cols]
        vif = [variance_inflation_factor(X.values, i) for i in ran
        display(list(zip(x_cols, vif)))
```

```
        print('\n')

        #step 4: check normality
        print('Normality Test Results')
        print('=====================================================
        #plot qqplot
        fig, ax = plt.subplots(figsize=(15,10))
        sm.graphics.qqplot(model.resid, dist=stats.norm, line='45'
        ax.set_title('QQPlot for Model Residuals')
        plt.show()
        print('\n')

        #step 5: check homoscedasticity
        print('Homoscedasticity Test Results')
        print('=====================================================
        #scatter plot
        fig, ax = plt.subplots(figsize=(15,10))
        plt.scatter(model.predict(df[x_cols]), model.resid)
        plt.plot(model.predict(df[x_cols]), [0 for i in range(len(
        ax.set_title('Model Residuals vs Model Predictions')
        plt.show()
    else:
        pass
    return model
```

In [100]:
```
#function to delete outliers using either iqr or std
def outliers(df, x_col, outlier='std'):
    '''This function takes in a dataframe, a column in the datafra
       whether or not to remove outliers via standard deviations o
       interquartile range.'''

    if outlier == 'std':
        #create outlier variables
        col_mean = df[x_col].mean()
        col_std = df[x_col].std()
        upper_thresh_std = col_mean + 3*col_std
        lower_thresh_std = col_mean - 3*col_std
        #update dataframe
        df_new = df.loc[(df[x_col] > lower_thresh_std) & (df[x_col
        print(f'There were {len(df) - len(df_new)} outliers remove
    elif outlier == 'iqr':
        #create outlier variables
        q25 = df[x_col].quantile(0.25)
        q75 = df[x_col].quantile(0.75)
        iqr = q75-q25
        upper_thresh_iqr = q75 + 1.5*iqr
        lower_thresh_iqr = q25 - 1.5*iqr
        #create new dataframe with outliers removed
        df_new = df.loc[(df[x_col] > lower_thresh_iqr) & (df[x_col
        print(f'There were {len(df) - len(df_new)} outliers remove

    return df_new
```

4.3 Import Data into Pandas

I will be importing a csv dataset which provides me with the information necessary to begin the analysis.

In [8]:

```
#import the dataset from local csv
df_original = pd.read_csv('Data/kc_house_data.csv')
df_original
```

Out[8]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	v
0	7129300520	10/13/2014	221,900.0	3	1.0	1180	5650	1.0	
1	6414100192	12/9/2014	538,000.0	3	2.25	2570	7242	2.0	
2	5631500400	2/25/2015	180,000.0	2	1.0	770	10000	1.0	
3	2487200875	12/9/2014	604,000.0	4	3.0	1960	5000	1.0	
4	1954400510	2/18/2015	510,000.0	3	2.0	1680	8080	1.0	
...	...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360,000.0	3	2.5	1530	1131	3.0	
21593	6600060120	2/23/2015	400,000.0	4	2.5	2310	5813	2.0	
21594	1523300141	6/23/2014	402,101.0	2	0.75	1020	1350	2.0	
21595	291310100	1/16/2015	400,000.0	3	2.5	1600	2388	2.0	
21596	1523300157	10/15/2014	325,000.0	2	0.75	1020	1076	2.0	

21597 rows × 21 columns

4.4 Data Schema

Taken from [https://rstudio-pubs-static.s3.amazonaws.com/155304_cc51f448116744069664b35e7762999f.html (https://rstudio-pubs-static.s3.amazonaws.com/155304_cc51f448116744069664b35e7762999f.html)](https://rstudio-pubs-static.s3.amazonaws.com/155304_cc51f448116744069664b35e7762999f.html)

id - Unique ID for each home sold

date - Date of the home sale

price - Price of each home sold

bedrooms - Number of bedrooms

bathrooms - Number of bathrooms, where .5 accounts for a room with a toilet but no shower

sqft_living - Square footage of the apartments interior living space

sqft_lot - Square footage of the land space floors - Number of floors

waterfront - A dummy variable for whether the apartment was overlooking the waterfront or not

view - An index from 0 to 4 of how good the view of the property was

condition - An index from 1 to 5 on the condition of the home

grade - An index from 1 to 13, where 1-3 falls short of building construction and design, 7 has an average level of construction and design, and 11-13 have a high quality level of construction and design.

sqft_above - The square footage of the interior housing space that is above ground level

sqft_basement - The square footage of the interior housing space that is below ground level

yr_built - The year the house was initially built

yr_renovated - The year of the house’s last renovation

zipcode - What zipcode area the house is in

lat - Lattitude

long - Longitude

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

4.5 Investigate Data

I will preliminarily investigate the data to identify any glaring issues to fix later.

In [9]:
```
#column names
df_original.columns
```

Out[9]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grad
e',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zi
pcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

In [10]:
```
#view df info to inspect data types
df_original.info()
```

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

OBSERVATIONS

- `waterfront` values should be updated to a binary categorical data type
- `yr_renovated` values should be updated to binary categorical data type
- `sqft_basement` values should be updated to a binary categorical data type

In [12]:
```
#check for null values
df_original.isna().sum()/len(df_original)*100
```

Out[12]:
```
id                              0.0
date                            0.0
price                           0.0
bedrooms                        0.0
bathrooms                       0.0
sqft_living                     0.0
sqft_lot                        0.0
floors                          0.0
waterfront        11.00152798999861
view             0.29170718155299347
condition                       0.0
grade                           0.0
sqft_above                      0.0
sqft_basement                   0.0
yr_built                        0.0
yr_renovated      17.78950780200954
zipcode                         0.0
lat                             0.0
long                            0.0
sqft_living15                   0.0
sqft_lot15                      0.0
dtype: float64
```

OBSERVATIONS

- `waterfront` has 11% null values which is a large number to simply drop. Will evaluate options.
- `view` should be explored further to see what it means
- `yr_renovated` has 18% null values which is a large number to simply drop. Will evaluate options.
- All other columns have 0 nulls

In [13]:
```
#check numeric data
df_original.describe()
```

Out[13]:

	id	price	bedrooms	bathrooms	
count	21,597.0	21,597.0	21,597.0	21,597.0	
mean	4,580,474,287.770987	540,296.5735055795	3.3731999814789093	2.1158262721674306	2,08
std	2,876,735,715.74778	367,368.1401013945	0.9262988945421479	0.7689842966527209	91
min	1,000,102.0	78,000.0	1.0	0.5	
25%	2,123,049,175.0	322,000.0	3.0	1.75	
50%	3,904,930,410.0	450,000.0	3.0	2.25	
75%	7,308,900,490.0	645,000.0	4.0	2.5	
max	9,900,000,190.0	7,700,000.0	33.0	8.0	

5 Scrub

I will make a new dataframe which is a copy of the `df_original` dataframe to begin making changes.

In [14]:

```
#create a copy of the original dataframe
df_scrub = df_original.copy()
df_scrub
```

Out[14]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	v
0	7129300520	10/13/2014	221,900.0	3	1.0	1180	5650	1.0	
1	6414100192	12/9/2014	538,000.0	3	2.25	2570	7242	2.0	
2	5631500400	2/25/2015	180,000.0	2	1.0	770	10000	1.0	
3	2487200875	12/9/2014	604,000.0	4	3.0	1960	5000	1.0	
4	1954400510	2/18/2015	510,000.0	3	2.0	1680	8080	1.0	
...	...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360,000.0	3	2.5	1530	1131	3.0	
21593	6600060120	2/23/2015	400,000.0	4	2.5	2310	5813	2.0	
21594	1523300141	6/23/2014	402,101.0	2	0.75	1020	1350	2.0	
21595	291310100	1/16/2015	400,000.0	3	2.5	1600	2388	2.0	
21596	1523300157	10/15/2014	325,000.0	2	0.75	1020	1076	2.0	

21597 rows × 21 columns

5.1 Feature Engineering

5.1.1 `basement` Column

In the dataset we have 3 related columns:

- `sqft_above`
- `sqft_basement`
- `sqft_living`

These columns are related in that `sqft_living` equals `sqft_above` plus `sqft_basement` . I do not think the square footage of the basement is as important as just knowing that a house has one. Therefore, I will create a new column which shows whether or not a house has a basement.

In [15]:
```
#investigate values in sqft_basement
display(df_scrub['sqft_basement'].value_counts(),len(df_scrub), df
```

```
0.0       12826
?           454
600.0       217
500.0       209
700.0       208
          ...  
2600.0        1
374.0         1
2350.0        1
225.0         1
1880.0        1
Name: sqft_basement, Length: 304, dtype: int64
```

```
21597
```

```
dtype('O')
```

ACTIONS

- '?' impedes the ability to create a new column. Will drop convert this to a 0 to indicate that the house does not have a basement.

In [16]:
```
#convert rows with a '?' to a 0
df_scrub.loc[df_scrub['sqft_basement'] == '?', ['sqft_basement']]
# display(df_scrub['sqft_basement'].value_counts(),len(df_scrub))
(df_scrub == 0).sum()
```

Out[16]:
```
id                   0
date                 0
price                0
bedrooms             0
bathrooms            0
sqft_living          0
sqft_lot             0
floors               0
waterfront       19075
view             19422
condition            0
grade                0
sqft_above           0
sqft_basement        0
yr_built             0
yr_renovated     17011
zipcode              0
lat                  0
long                 0
sqft_living15        0
sqft_lot15           0
dtype: int64
```

In [17]:
```
#prove that these columns are related
df_scrub['sqft_basement'] = df_scrub['sqft_basement'].astype(float
sqft = df_scrub[['sqft_living', 'sqft_above', 'sqft_basement']]
(sqft['sqft_above'] + sqft['sqft_basement'] == sqft['sqft_living']
```

Out[17]:
```
True     21427
False      170
dtype: int64
```

In [18]:
```
#investigate the Falses
df_scrub.loc[(sqft['sqft_above'] + sqft['sqft_basement'] == sqft['
```

Out[18]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
112	2525310310	9/16/2014	272,500.0	3	1.75	1540	12600	1.0
115	3626039325	11/21/2014	740,500.0	3	3.5	4380	6350	2.0
309	3204800200	1/8/2015	665,000.0	4	2.75	3320	10574	2.0
384	713500030	7/28/2014	1,350,000.0	5	3.5	4800	14984	2.0
508	5113400431	5/8/2014	615,000.0	2	1.0	1540	6872	1.0
...	...	...	...	...	...	...	...	...
21000	291310180	6/13/2014	379,500.0	3	2.25	1410	1287	2.0
21109	3438500250	6/23/2014	515,000.0	5	3.25	2910	5027	2.0
21210	3278600680	6/27/2014	235,000.0	1	1.5	1170	1456	2.0
21356	6169901185	5/20/2014	490,000.0	5	3.5	4460	2975	3.0
21442	3226049565	7/11/2014	504,600.0	5	3.0	2360	5000	1.0

170 rows × 21 columns

OBSERVATIONS

- It seems that 170 homes have a difference between the `sqft_living` and `sqft_above` that were originially classified as a '?'.

ACTIONS

- I will now assume that the difference in these 170 homes is due to having sq_ft in the basement. I will change from a 0 to the difference in `sqft_living` and `sqft_above`

In [19]:
```
#replace sqft_basement with the desscrepency between sqft_living a
df_scrub['sqft_basement'] = df_scrub['sqft_living']- df_scrub['sqf
display(df_scrub['sqft_basement'].value_counts(), len(df_scrub))
```

```
0       13110
600       221
700       218
500       214
800       206
        ...
792         1
2590        1
935         1
2390        1
248         1
Name: sqft_basement, Length: 306, dtype: int64

21597
```

In [20]:
```
#check previous id where sqft_basement was a '?'
df_scrub.loc[df_scrub['id'] == 2525310310]
```

Out[20]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wate
112	2525310310	9/16/2014	272,500.0	3	1.75	1540	12600	1.0	

In [21]:
```
#check the rows have been dropped
df_scrub.loc[df_scrub['sqft_basement'] == '?']['sqft_basement'].co
```

Out[21]: 0

In [22]:
```
#check to ensure all descrepencies are gone
df_scrub.loc[(df_scrub['sqft_above'] + df_scrub['sqft_basement'] =
```

Out[22]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditio

In [23]:
```
#what is the median basement sqft
df_scrub.loc[df_scrub['sqft_basement'] > 0,'sqft_basement'].mediar
```

Out[23]: 700.0

ACTIONS

- Will now create new column named `basement` which represents whether or not a house has a basement.

In [24]:
```
#create new column for basement and verify
df_scrub['basement'] = np.where(df_scrub['sqft_basement'] > 0, 1,0
df_scrub[['sqft_basement','basement']].value_counts()
```

Out[24]:
```
sqft_basement  basement
0              0           13110
600            1             221
700            1             218
500            1             214
800            1             206
                           ...
2360           1               1
475            1               1
2350           1               1
1930           1               1
4820           1               1
Length: 306, dtype: int64
```

In [25]:
```
#how much more sqft does a house have when they have a basement?
df_scrub.groupby(by='basement')['sqft_living'].median()
```

Out[25]:
```
basement
0    1740
1    2100
Name: sqft_living, dtype: int64
```

5.1.2 `renovated` Column

I want to reconfigure the `yr_renovated` column so that it is compatible with the model. I will convert null rows and create a new column which indicates whether or not a house has been renovated.

In [26]:
```
#check values in yr_renovated column
df_scrub['yr_renovated'].value_counts(dropna=False).head(20)
```

Out[26]:
```
0.0        17011
nan         3842
2,014.0       73
2,003.0       31
2,013.0       31
2,007.0       30
2,005.0       29
2,000.0       29
1,990.0       22
2,004.0       22
2,009.0       21
1,989.0       20
2,006.0       20
2,002.0       17
1,991.0       16
1,998.0       16
1,984.0       16
1,999.0       15
2,001.0       15
2,008.0       15
Name: yr_renovated, dtype: int64
```

ACTIONS

- I will set the null values to 0 which will be converted to a No

In [27]:
```
#convert null to 0 and verify
df_scrub.loc[df_scrub['yr_renovated'].isna(),['yr_renovated']] = 0
df_scrub['yr_renovated'].isna().sum()
```

Out[27]: 0

ACTIONS

- Create new `renovated` column which gives a 0 if false and 1 if true

In [28]:

```
#create new column based on yr_renovated
df_scrub['renovated'] = np.where(df_scrub['yr_renovated'] == 0, 0,
df_scrub[['yr_renovated','renovated']]
```

Out[28]:

	yr_renovated	renovated
0	0.0	0
1	1,991.0	1
2	0.0	0
3	0.0	0
4	0.0	0
...	...	...
21592	0.0	0
21593	0.0	0
21594	0.0	0
21595	0.0	0
21596	0.0	0

21597 rows × 2 columns

5.1.3 home_age Column

I want to create a column named `home_age` which represents the homes age which I believe will be more informative in a model. I will take the sale `date` and subtract the `yr_built` from it to get the home age.

In [29]:
```
#explore data values
df_scrub['yr_built'].value_counts()
```

Out[29]:
```
2014    559
2006    453
2005    450
2004    433
2003    420
       ...
1933     30
1901     29
1902     27
1935     24
1934     21
Name: yr_built, Length: 116, dtype: int64
```

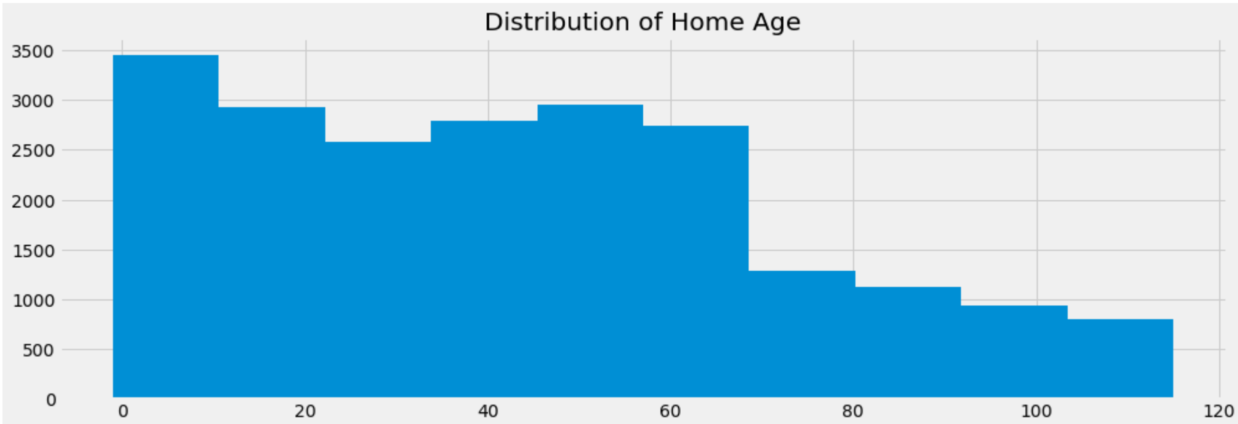
In [30]:
```
#create yr_sold column first
df_scrub['date'] = pd.to_datetime(df_scrub['date'])
df_scrub['yr_sold'] = df_scrub['date'].dt.year
df_scrub[['date','yr_built','yr_sold']]
```

Out[30]:

	date	yr_built	yr_sold
0	2014-10-13	1955	2014
1	2014-12-09	1951	2014
2	2015-02-25	1933	2015
3	2014-12-09	1965	2014
4	2015-02-18	1987	2015
...	...	...	...
21592	2014-05-21	2009	2014
21593	2015-02-23	2014	2015
21594	2014-06-23	2009	2014
21595	2015-01-16	2004	2015
21596	2014-10-15	2008	2014

21597 rows × 3 columns

In [31]:
```
#create new column
df_scrub['home_age'] = df_scrub['yr_sold'] - df_scrub['yr_built']
#plot distribution
fig, ax = plt.subplots(figsize=(15,5))
df_scrub['home_age'].hist(ax=ax);
ax.set_title('Distribution of Home Age');
```



5.2 Change Data Types

In [32]:
```
#check data types
df_scrub.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 25 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21597 non-null  int64
 1   date           21597 non-null  datetime64[ns]
 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  int64
 14  yr_built       21597 non-null  int64
 15  yr_renovated   21597 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
 21  basement       21597 non-null  int64
 22  renovated      21597 non-null  int64
 23  yr_sold        21597 non-null  int64
 24  home_age       21597 non-null  int64
dtypes: datetime64[ns](1), float64(8), int64(16)
memory usage: 4.1 MB
```

OBSERVATIONS

- All data types seem good for the model

5.3 Null Values

In [33]:
```
#check for null values
df_scrub.isna().sum()
```

Out[33]:
```
id                  0
date                0
price               0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront       2376
view               63
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_built            0
yr_renovated        0
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
basement            0
renovated           0
yr_sold             0
home_age            0
dtype: int64
```

5.3.1 waterfront Column

In [36]:
```
#view values in waterfront column
df_scrub['waterfront'].value_counts(dropna=False)
```

Out[36]:
```
0.0    19075
nan     2376
1.0      146
Name: waterfront, dtype: int64
```

OBSERVATIONS

- `waterfront` has 11% null values.

ACTIONS

- I will explore how I can fill the nulls in the `waterfront` values

In [37]:
```
#correlation of waterfront
df_scrub.corr()['waterfront']
```

Out[37]:
```
id               -0.004176270319423232
price             0.2762953839352147
bedrooms        -0.0023863997212393968
bathrooms         0.06728248839282633
sqft_living       0.11022963065969003
sqft_lot         0.023142981248142104
floors            0.02188315298160894
waterfront                        1.0
view              0.40665414022705837
condition        0.017642055701219666
grade              0.0873830334183805
sqft_above        0.07546267923971185
sqft_basement     0.08788403196747058
yr_built         -0.026078832177521185
yr_renovated      0.07924510606164326
zipcode          0.031057036096177812
lat              -0.012771945478265526
long              -0.03986418861080471
sqft_living15     0.08886026188331088
sqft_lot15       0.032001712349143514
basement         0.042454767375031836
renovated         0.07960111613498165
yr_sold          -0.005162787518761674
home_age         0.025995139447540193
Name: waterfront, dtype: float64
```

OBSERVATIONS

- `waterfront` correlates most closely with `view` at a coefficient of 0.40

ACTIONS

- I will determine how i can utilize the `view` column to fill out the nulls in the `waterfall` column

In [38]:
```
#number of waterfront properties in each view category
df_scrub.groupby('view')['waterfront'].sum()
```

Out[38]:
```
view
0.0      0.0
1.0      1.0
2.0      7.0
3.0     14.0
4.0    123.0
Name: waterfront, dtype: float64
```

OBSERVATIONS

- It seems that most of the waterfront homes also have a view ranking of 3 or 4

In [39]:
```
#there are 19 null values in waterfront with a view of 4
df_scrub.loc[(df_scrub['waterfront'].isna()) & (df_scrub['view'] =
```

Out[39]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	water
582	2998800125	2014-07-01	730,000.0	2	2.25	2130	4920	1.5	
1732	913000340	2015-01-02	252,000.0	1	1.0	680	1638	1.0	
2464	9275700016	2014-07-06	1,280,000.0	4	2.5	3160	4620	1.5	
2563	7856400240	2015-02-11	1,650,000.0	4	3.0	3900	9750	1.0	
3825	8550001515	2014-10-01	429,592.0	2	2.75	1992	10946	1.5	

In [40]:
```
#there are 54 null values in waterfront with a view of 3
df_scrub.loc[(df_scrub['waterfront'].isna()) & (df_scrub['view'] =
```

Out[40]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfr
60	1516000055	2014-12-10	650,000.0	3	2.25	2150	21235	1.0	
216	46100204	2015-02-21	1,510,000.0	5	3.0	3300	33474	1.0	
527	3225079035	2014-06-18	1,600,000.0	6	5.0	6050	230652	2.0	
673	1959701890	2014-07-29	865,000.0	4	1.75	1800	4180	2.0	
707	4022907770	2014-10-14	550,000.0	4	1.75	2480	14782	1.0	

ACTIONS

- Fill in the null waterfront value when the view is 3 or 4

In [41]:
```
#fill in null where view is 4
df_scrub.loc[(df_scrub['waterfront'].isna()) & (df_scrub['view'] =
```

In [42]:
```
#fill in null where view is 3
df_scrub.loc[(df_scrub['waterfront'].isna()) & (df_scrub['view'] =
```

In [43]:
```
#check the changes
df_scrub.loc[(df_scrub['waterfront'].isna()) & (df_scrub['view'] =
```

Out[43]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditio

In [44]:
```
#check the changes
df_scrub.loc[(df_scrub['waterfront'].isna()) & (df_scrub['view'] =
```

Out[44]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditio

In [45]:
```
#view values in waterfront column
df_scrub['waterfront'].value_counts(dropna=False)/len(df_scrub)
```

Out[45]:
```
0.0     0.8832245219243413
nan    0.10538500717692272
1.0   0.011390470898735936
Name: waterfront, dtype: float64
```

OBSERVATIONS

- The number of nulls in the `waterfront` column is still 10.5%

ACTIONS

- I will convert the rest of the nulls to zeros as they do not seem to have any other indicators of being a waterfront property

In [46]:
```
#convert waterfront null values to 0
df_scrub.loc[df_scrub['waterfront'].isna(),['waterfront']] = 0
```

In [47]:
```
#check waterfront values
df_scrub['waterfront'].value_counts(dropna=False)
```

Out[47]:
```
0.0    21351
1.0      246
Name: waterfront, dtype: int64
```

In [48]:
```
#print number of rows in dataframe
print(f'The dataframe now has {len(df_scrub)} many rows.')
```

```
The dataframe now has 21597 many rows.
```

5.3.2 view Column

In [49]:
```
#view the values of the view column
df_scrub['view'].value_counts(dropna=False)
```

Out[49]:
```
0.0    19422
2.0      957
3.0      508
1.0      330
4.0      317
nan       63
Name: view, dtype: int64
```

ACTIONS

- I will drop the 39 null values

In [50]:
```
#drop rows
df_scrub.dropna(subset=['view'], inplace=True)
df_scrub['view'].isna().sum()
```

Out[50]:
```
0
```

In [51]:
```
#check the view column
df_scrub['view'].value_counts(dropna=False)
```

Out[51]:
```
0.0    19422
2.0      957
3.0      508
1.0      330
4.0      317
Name: view, dtype: int64
```

In [52]:
```
#recheck null values in dataframe
df_scrub.isna().sum()
```

Out[52]:
```
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
basement         0
renovated        0
yr_sold          0
home_age         0
dtype: int64
```

5.4 Duplicates

5.4.1 Duplicates for `id`

In [53]:
```
#check for duplicates
duplicate_id = df_scrub.loc[df_scrub.duplicated(subset='id', keep=
duplicate_id.head()
```

Out[53]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfr
2495	1000102	2015-04-22	300,000.0	6	3.0	2400	9373	2.0	
2494	1000102	2014-09-16	280,000.0	6	3.0	2400	9373	2.0	
16800	7200179	2014-10-16	150,000.0	2	1.0	840	12750	1.0	
16801	7200179	2015-04-24	175,000.0	2	1.0	840	12750	1.0	
11422	109200390	2014-10-20	250,000.0	3	1.75	1480	3900	1.0	

In [54]:
```
#check duplicates
df_scrub.loc[df_scrub['id'] == 4139480200]
```

Out[54]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfr
313	4139480200	2014-06-18	1,380,000.0	4	3.25	4290	12103	1.0	
314	4139480200	2014-12-09	1,400,000.0	4	3.25	4290	12103	1.0	

OBSERVATOINS

- Duplicates in the `id` column seem to represent multiple sales of the same house.

ACTIONS

- I will consider these duplicates as separate homes and keep them in the dataset. The column `id` will be removed later.

5.5 Column Drop

In [55]:
```
#look at columns
df_scrub.columns
```

Out[55]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grad
e',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zi
pcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15', 'basement', 'ren
ovated',
       'yr_sold', 'home_age'],
      dtype='object')
```

5.5.1 The `sqft_basement` Column

The `sqft_basement` column can be eliminated now that I have a column which represents whether or not a house has a basement.

In [56]:
```
#drop the sqft_basement column
df_scrub.drop(columns='sqft_basement', inplace=True)
df_scrub.columns
```

Out[56]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grad
e',
       'sqft_above', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'l
ong',
       'sqft_living15', 'sqft_lot15', 'basement', 'renovated', 'yr_so
ld',
       'home_age'],
      dtype='object')
```

5.5.2 The `sqft_living15` and `sqft_lot15` Columns

The `sqft_living15` and `sqft_lot15` columns do not seem to be relevant for predicting home listing prices. I will remove these.

In [57]:
```
#drop columns
df_scrub.drop(columns=['sqft_living15','sqft_lot15'], inplace=True
df_scrub.columns
```

Out[57]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grad
e',
       'sqft_above', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'l
ong',
       'basement', 'renovated', 'yr_sold', 'home_age'],
      dtype='object')
```

5.5.3 yr_renovated Column

In [58]:
```
#drop the yr_renovated column
df_scrub.drop(columns='yr_renovated', inplace=True)
```

5.5.4 id Column

In [59]:
```
#drop the id column
df_scrub.drop(columns='id', inplace=True)
```

5.6 State of Dataframe

In [60]:
```
#state of the dataframe
df_scrub
```

Out[60]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	c
0	2014-10-13	221,900.0	3	1.0	1180	5650	1.0	0.0	0.0	
1	2014-12-09	538,000.0	3	2.25	2570	7242	2.0	0.0	0.0	
2	2015-02-25	180,000.0	2	1.0	770	10000	1.0	0.0	0.0	
3	2014-12-09	604,000.0	4	3.0	1960	5000	1.0	0.0	0.0	
4	2015-02-18	510,000.0	3	2.0	1680	8080	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	
21592	2014-05-21	360,000.0	3	2.5	1530	1131	3.0	0.0	0.0	
21593	2015-02-23	400,000.0	4	2.5	2310	5813	2.0	0.0	0.0	
21594	2014-06-23	402,101.0	2	0.75	1020	1350	2.0	0.0	0.0	
21595	2015-01-16	400,000.0	3	2.5	1600	2388	2.0	0.0	0.0	
21596	2014-10-15	325,000.0	2	0.75	1020	1076	2.0	0.0	0.0	

21534 rows × 20 columns

6 Explore

I will now explore the dataset after initial scrubbing. I will investigate linearity and multicollinearity and correct any issues before modeling.

In [61]:
```
#create a copy of the scrub dataframe
df_explore = df_scrub.copy()
df_explore.head()
```

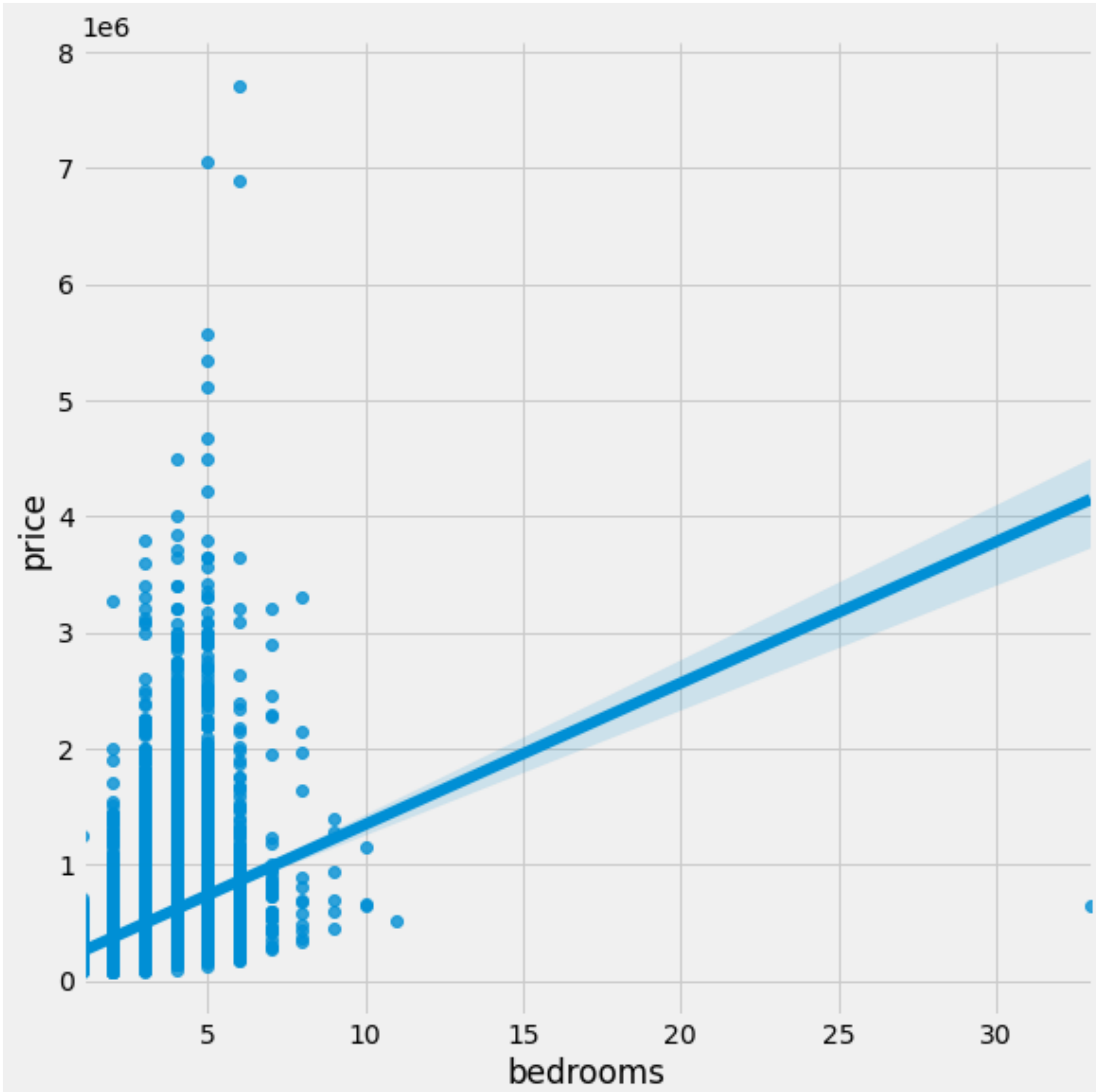
Out[61]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condi
0	2014-10-13	221,900.0	3	1.0	1180	5650	1.0	0.0	0.0	
1	2014-12-09	538,000.0	3	2.25	2570	7242	2.0	0.0	0.0	
2	2015-02-25	180,000.0	2	1.0	770	10000	1.0	0.0	0.0	
3	2014-12-09	604,000.0	4	3.0	1960	5000	1.0	0.0	0.0	
4	2015-02-18	510,000.0	3	2.0	1680	8080	1.0	0.0	0.0	

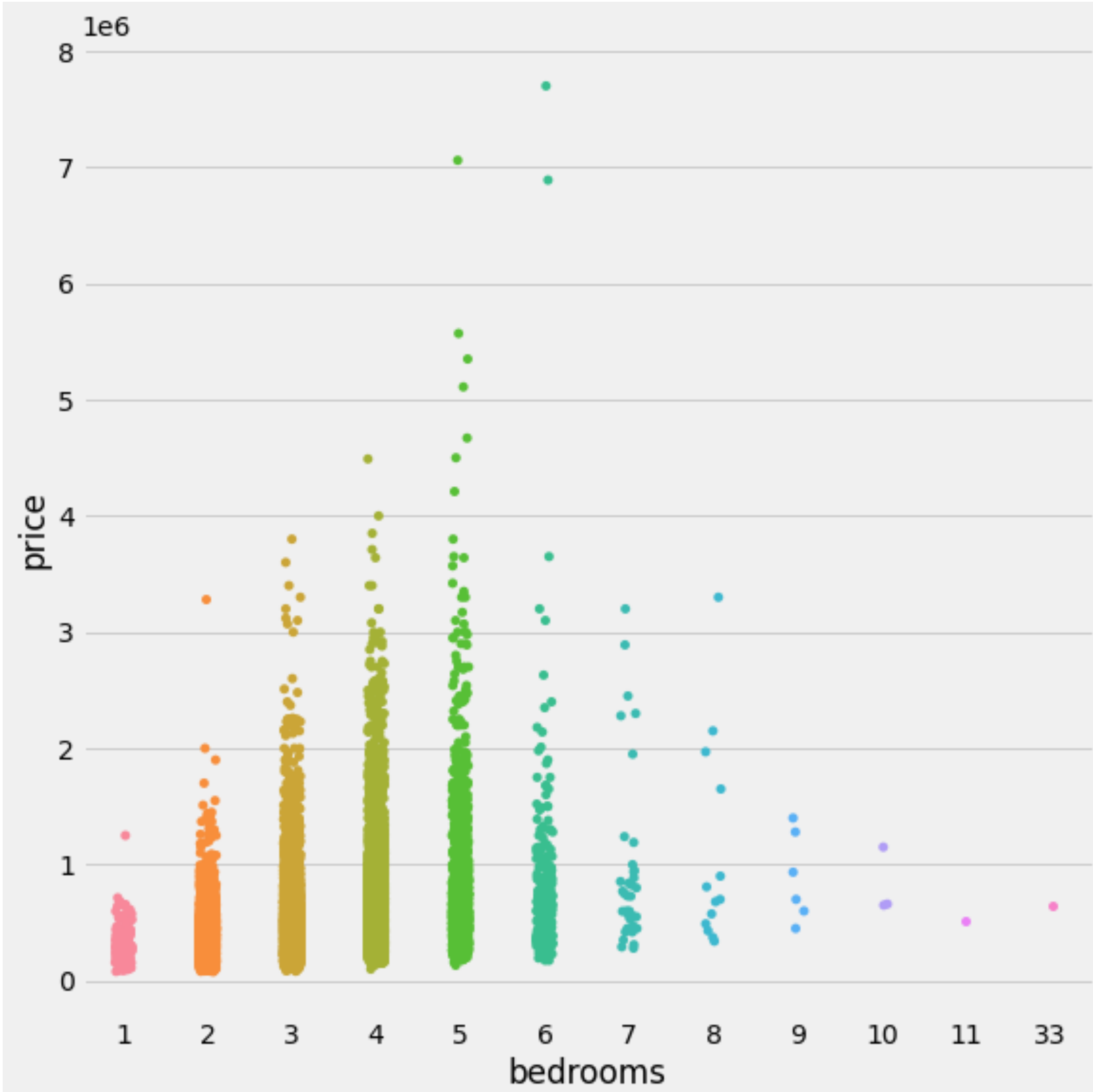
6.1 Linearity

6.1.1 bedrooms

In [62]:
```
#check linearity between bedrooms and price
sns.lmplot(data=df_explore, x='bedrooms', y='price', height=8);
```



In [63]:
```
#view price vs bedrooms another way
sns.catplot(data=df_explore, x='bedrooms', y='price', height=8);
```

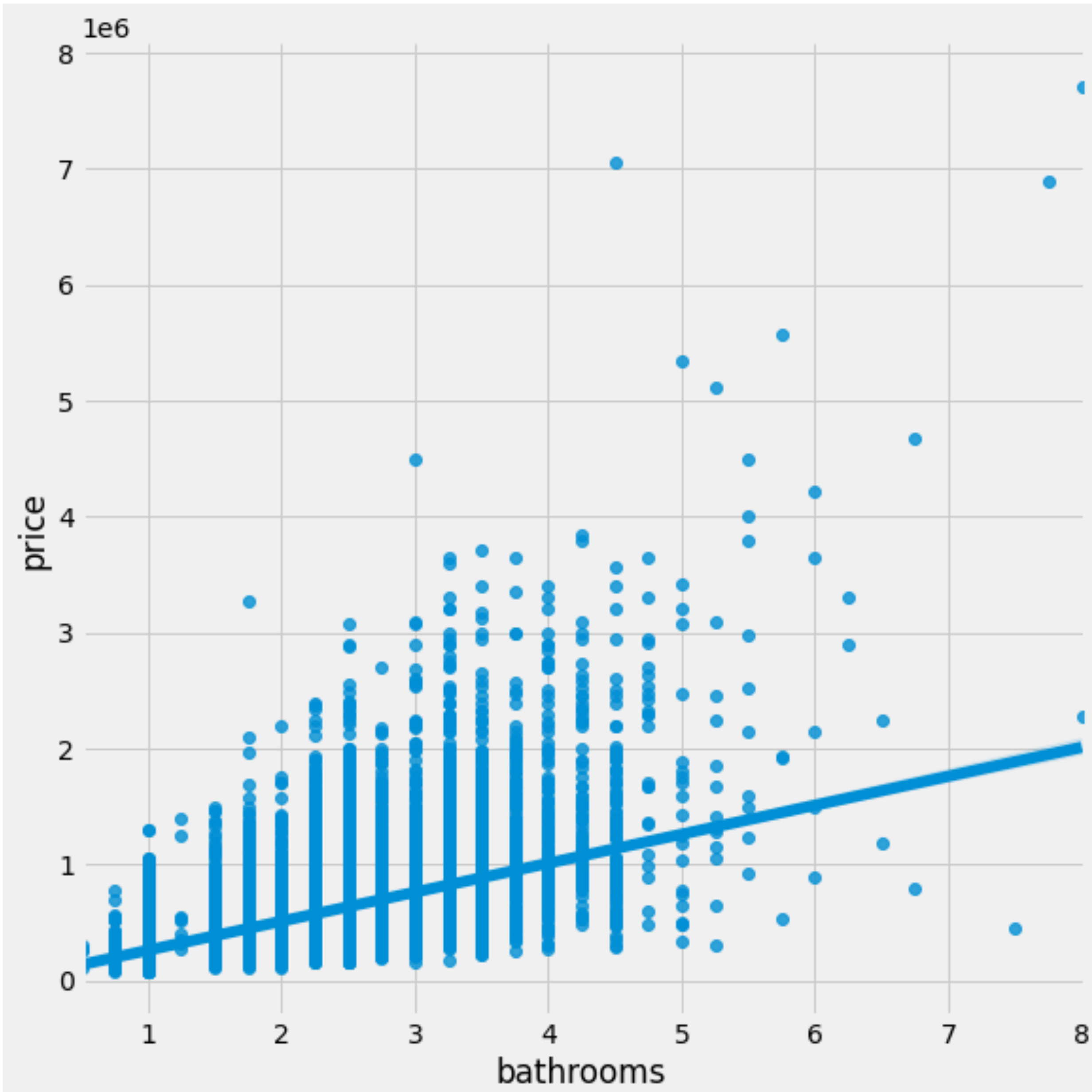


OBSERVATIONS

- There seems to be a positive linear relationship between the number of bedrooms and the price of the home for homes with a 1-5 bedrooms. Homes with 6+ bedrooms seem to be valued at a lower price.
- I notice some outliers that I will need to remove.

6.1.2 bathrooms

In [64]:
```
#check linearity between bathrooms and price
sns.lmplot(data=df_explore, x='bathrooms', y='price', height=8);
```



OBSERVATIONS

- There seems to be a positive linear relationship between `bathrooms` and `price`.

6.1.3 `sqft_living`

In [65]:
```
#check linearity between sqft_living and price
sns.lmplot(data=df_explore, x='sqft_living', y='price', height=8);
```

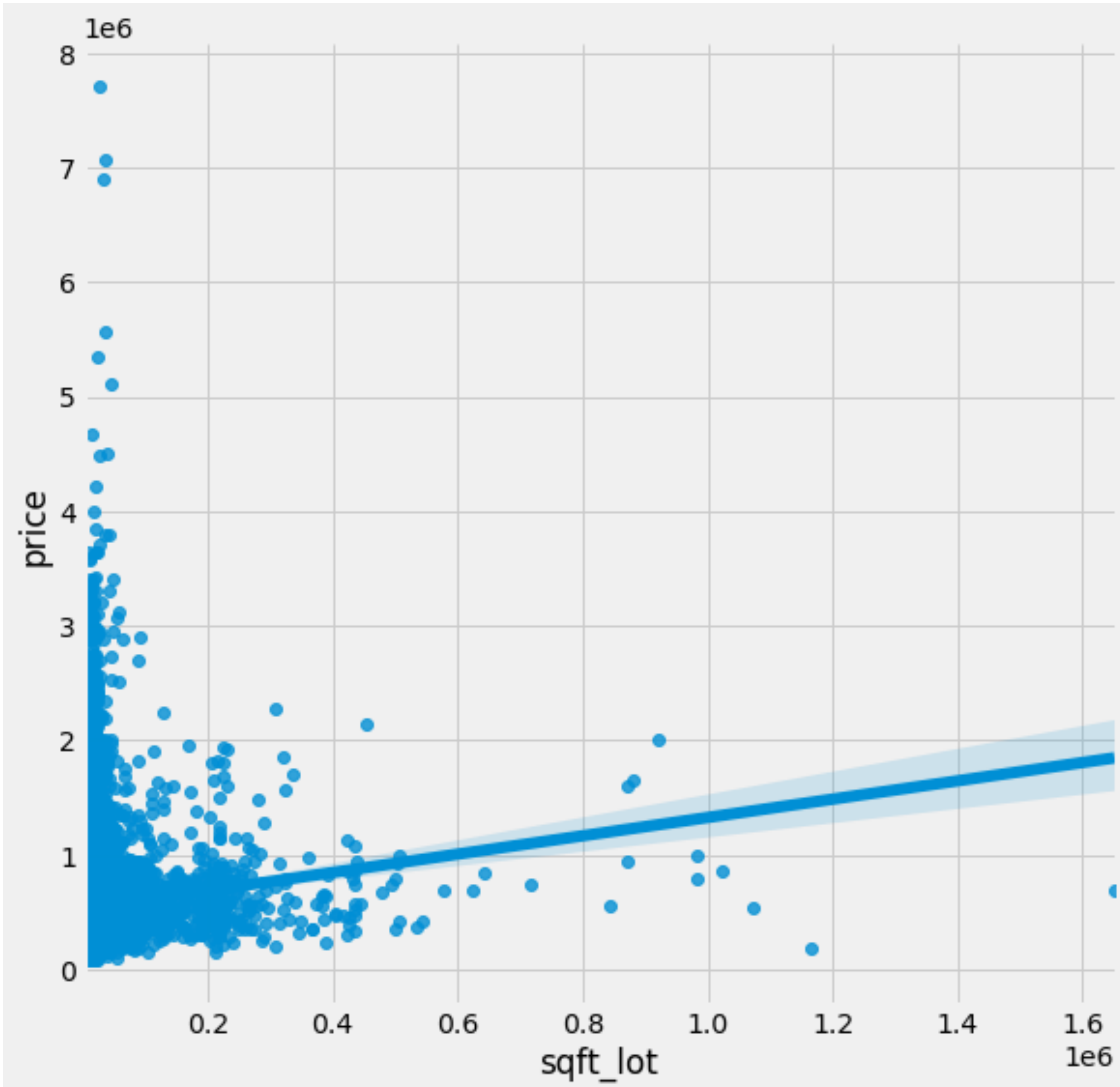
OBSERVATIONS

- There seems to be a strong positive linear relationship between `sqft_living` and `price`.

6.1.4 sqft_lot

In [66]:
```
#check linearity between sqft_lot and price
sns.lmplot(data=df_explore, x='sqft_lot', y='price', height=8)
```

Out[66]: <seaborn.axisgrid.FacetGrid at 0x7fdfd967cd60>

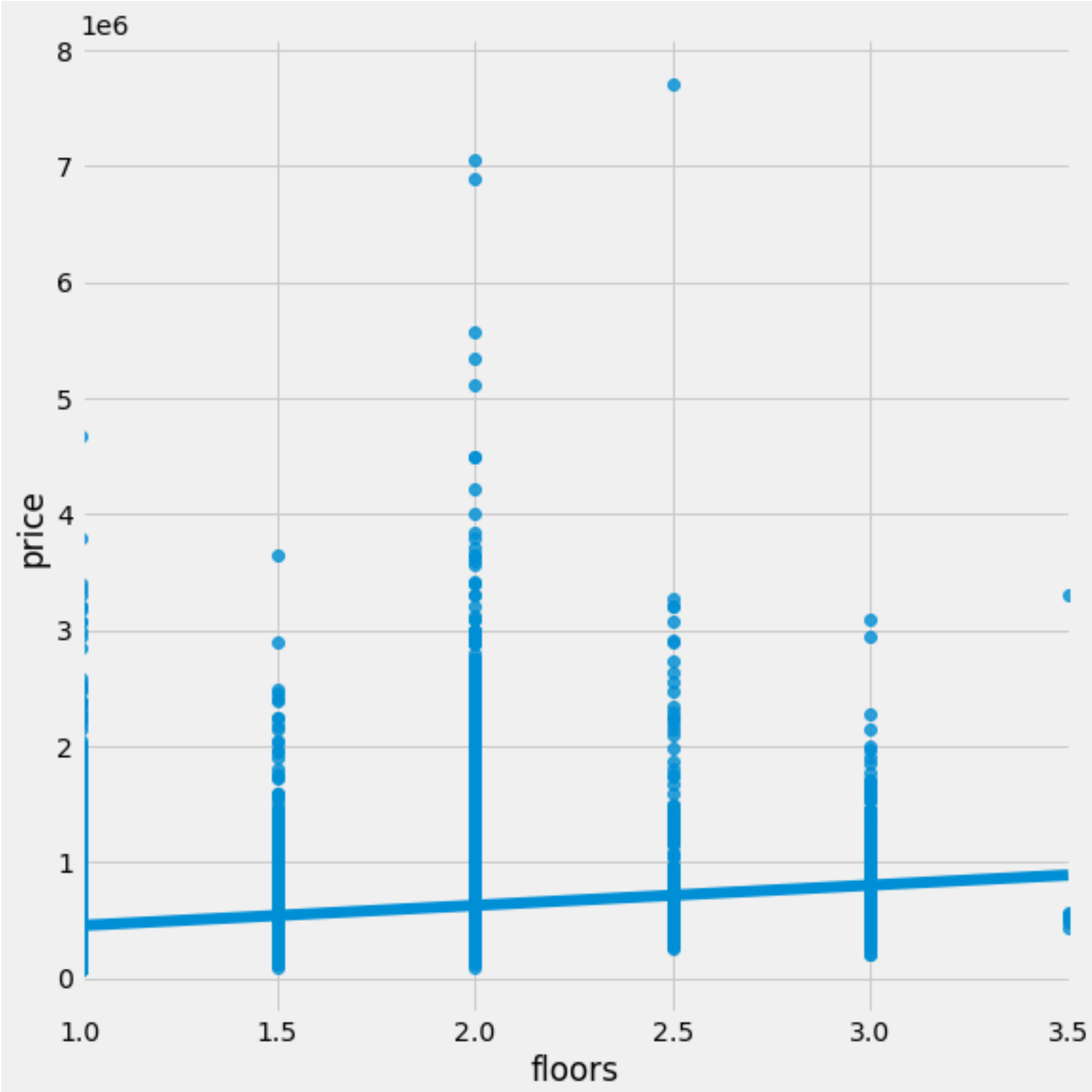


OBSERVATIONS

- There seems to be a linear relationship between `sqft_lot` and `price`. However, there seems to be 2 types of high value homes, 1) very small lot homes with high prices and 2) large lot homes with high prices

6.1.5 `floors`

In [67]:
```
#check linearity between floors and price
sns.lmplot(data=df_explore, x='floors', y='price', height=8);
```

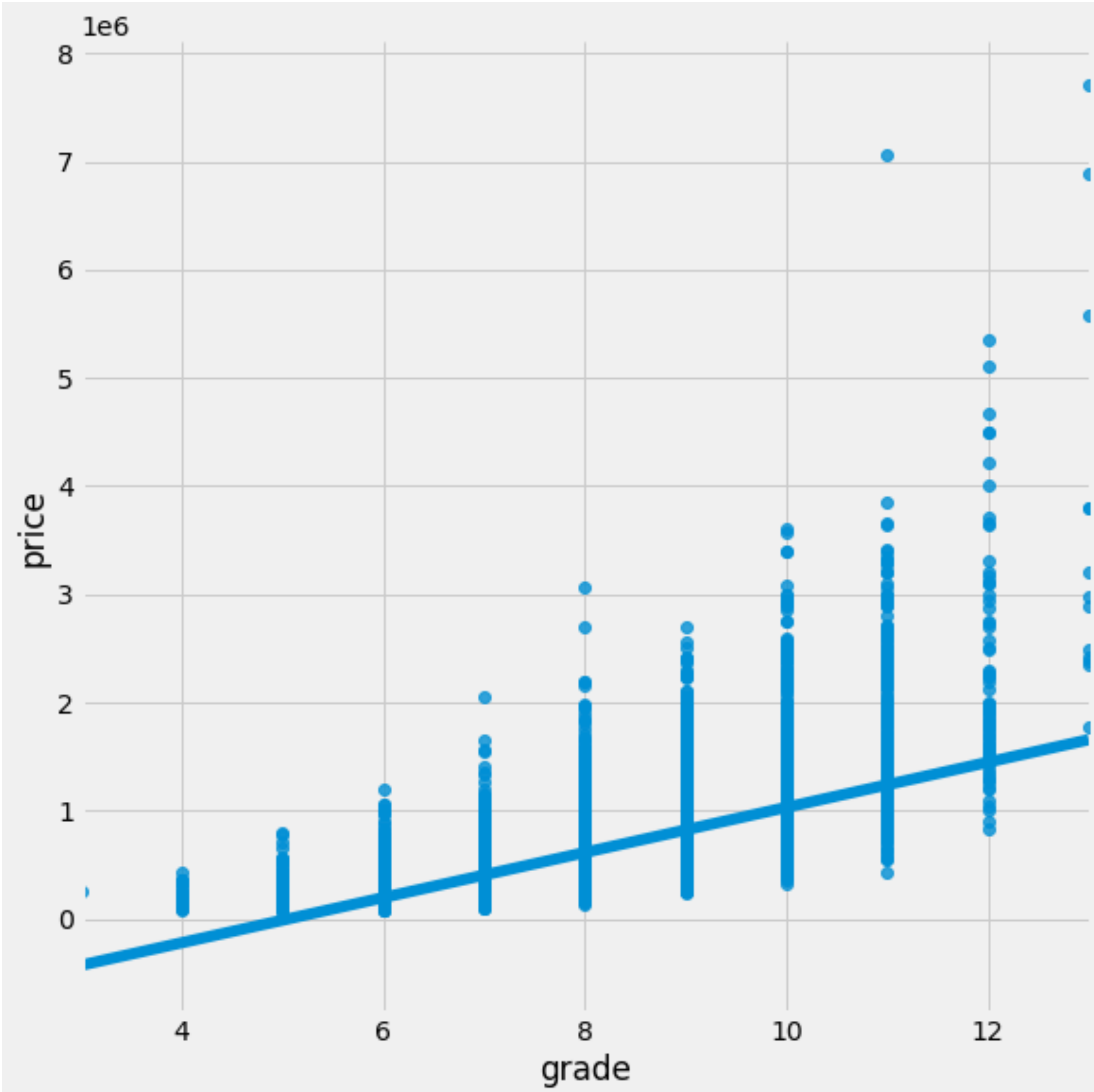


OBSERVATIONS

- There seems to be a linear relationship between `floors` and `price`.

6.1.6 `grade`

In [68]:
```
#check linearity between grade and price
sns.lmplot(data=df_explore, x='grade', y='price', height=8);
```

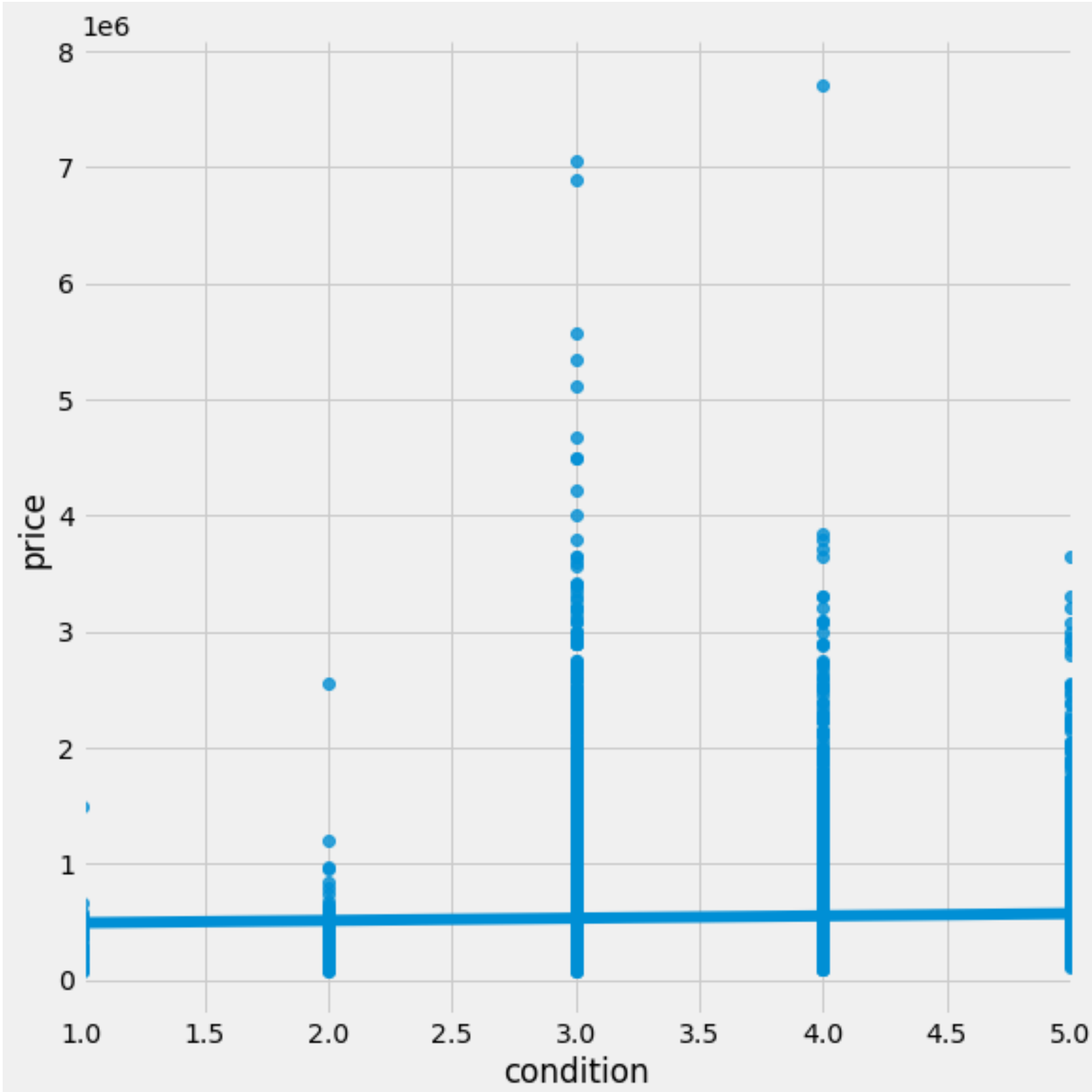


OBSERVATIONS

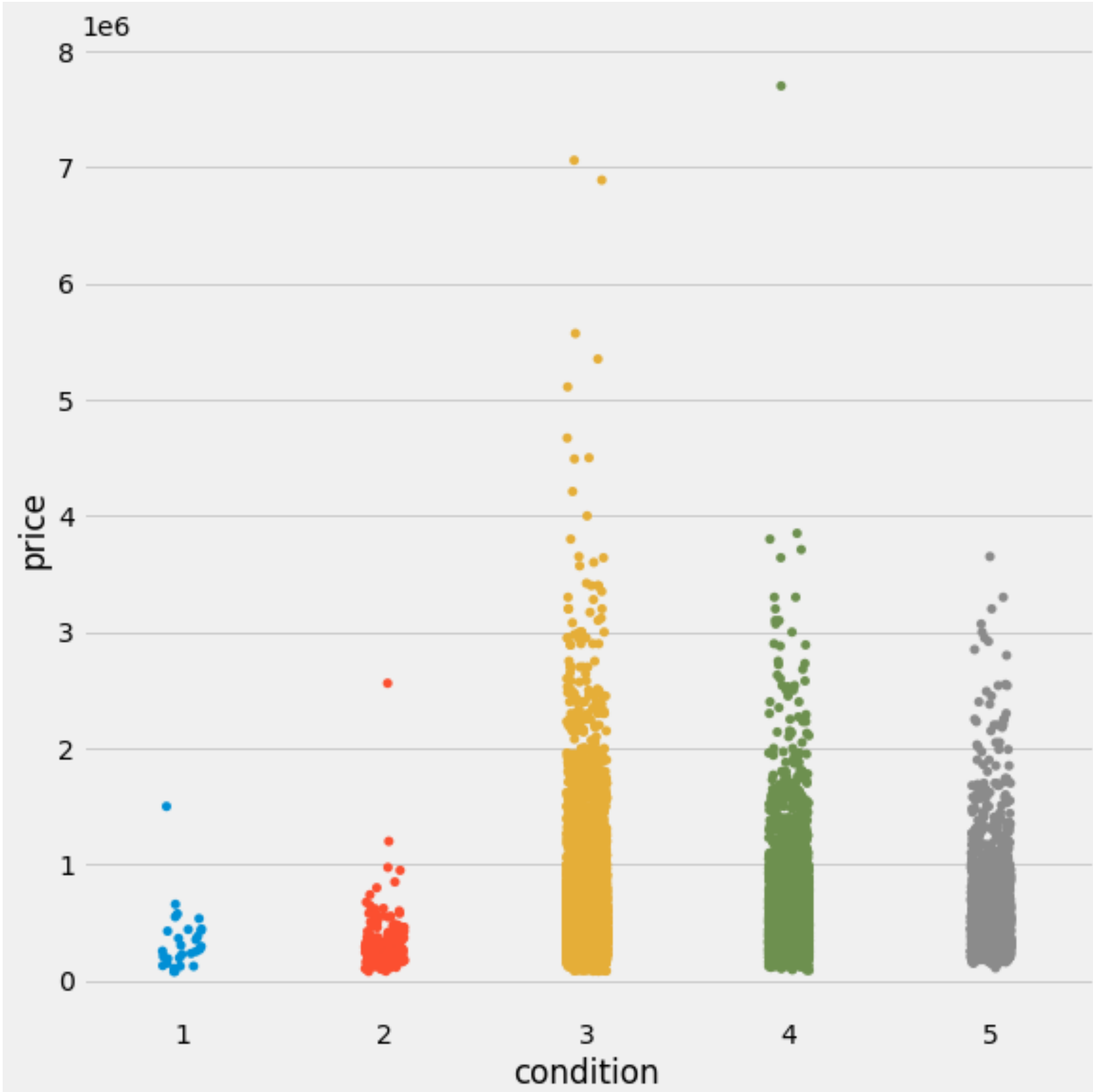
- There seems to be a linear relationship between `grade` and `price` .

6.1.7 `condition`

In [69]:
```
#check linearity between condition and price
sns.lmplot(data=df_explore, x='condition', y='price', height=8);
```



In [70]:
```python
#check relationship another way
sns.catplot(data=df_explore, x='condition', y='price', height=8);
```

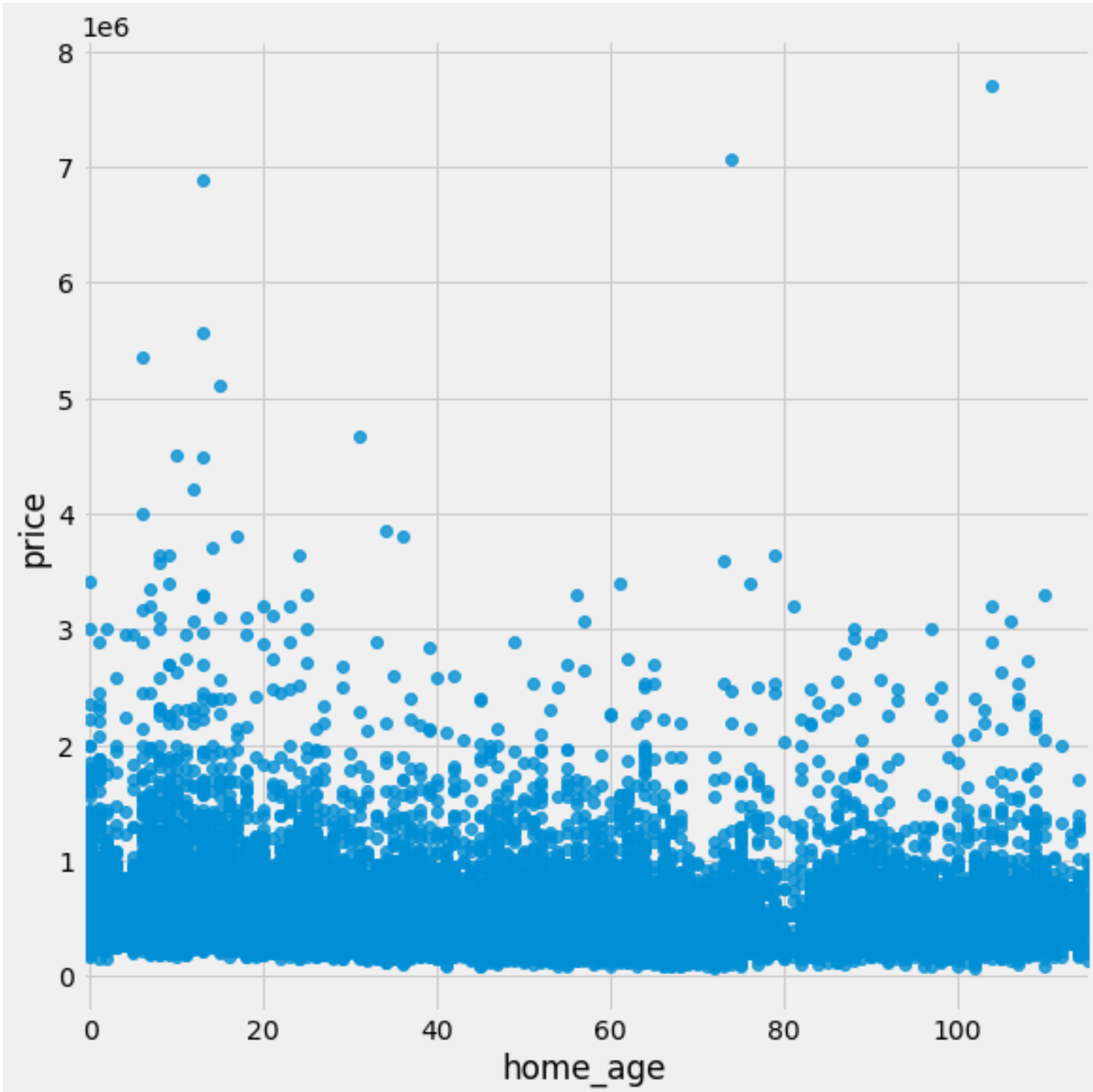


OBSERVATIONS

- There seems to be a linear relationship between `condition` and `price` , however, there seems to be a sweet spot around 3 i.e. not any additional value to condition 4 and 5. I will test this later

6.1.8 home_age

In [71]:
```
#check linearity between condition and price
sns.lmplot(data=df_explore, x='home_age', y='price', height=8);
```



OBSERVATIONS

- There seems to be a linear relationship between `home_age` and `price`, however, it seems like it is a neutral relationship.

6.2 Multicollinearity

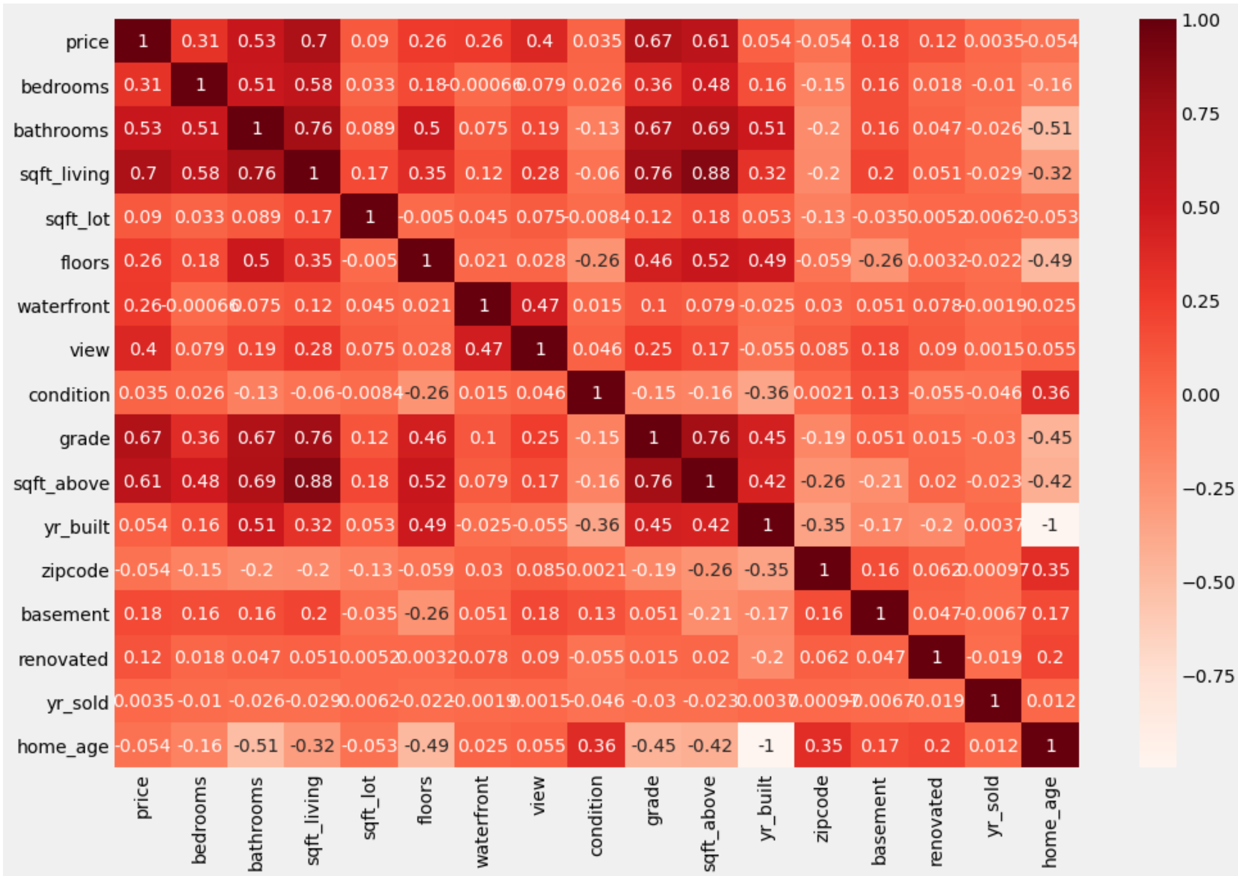
I want to check to see if the independent variables are truly independent from each other by checking for multicollinearity.

6.2.1 Two Variable Multicollinearity Check

In [72]:
```
#remove lat and long columns
df_explore.drop(columns=['lat','long'], inplace=True)
```

In [73]:

```
#create and plot correlations
corr = df_explore.corr()
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(corr, cmap='Reds', annot=True, ax=ax);
```



ACTIONS

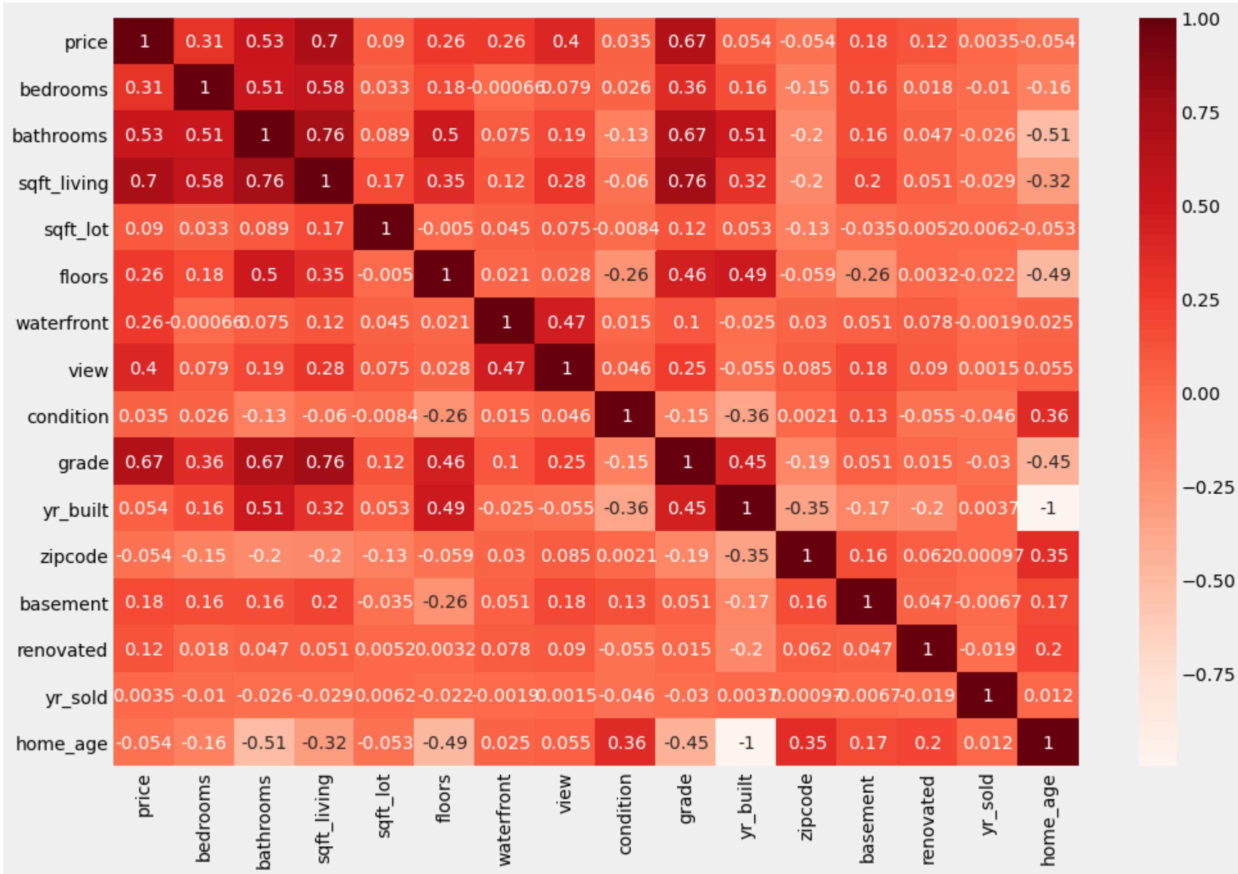
- Remove `sqft_above` as it correlates very closesly with `sqft_living`

In [74]:

```
#remove sqft_above
df_explore.drop(columns='sqft_above', inplace=True)
```

In [75]:

```
corr = df_explore.corr()
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(corr, cmap='Reds', annot=True, ax=ax);
```



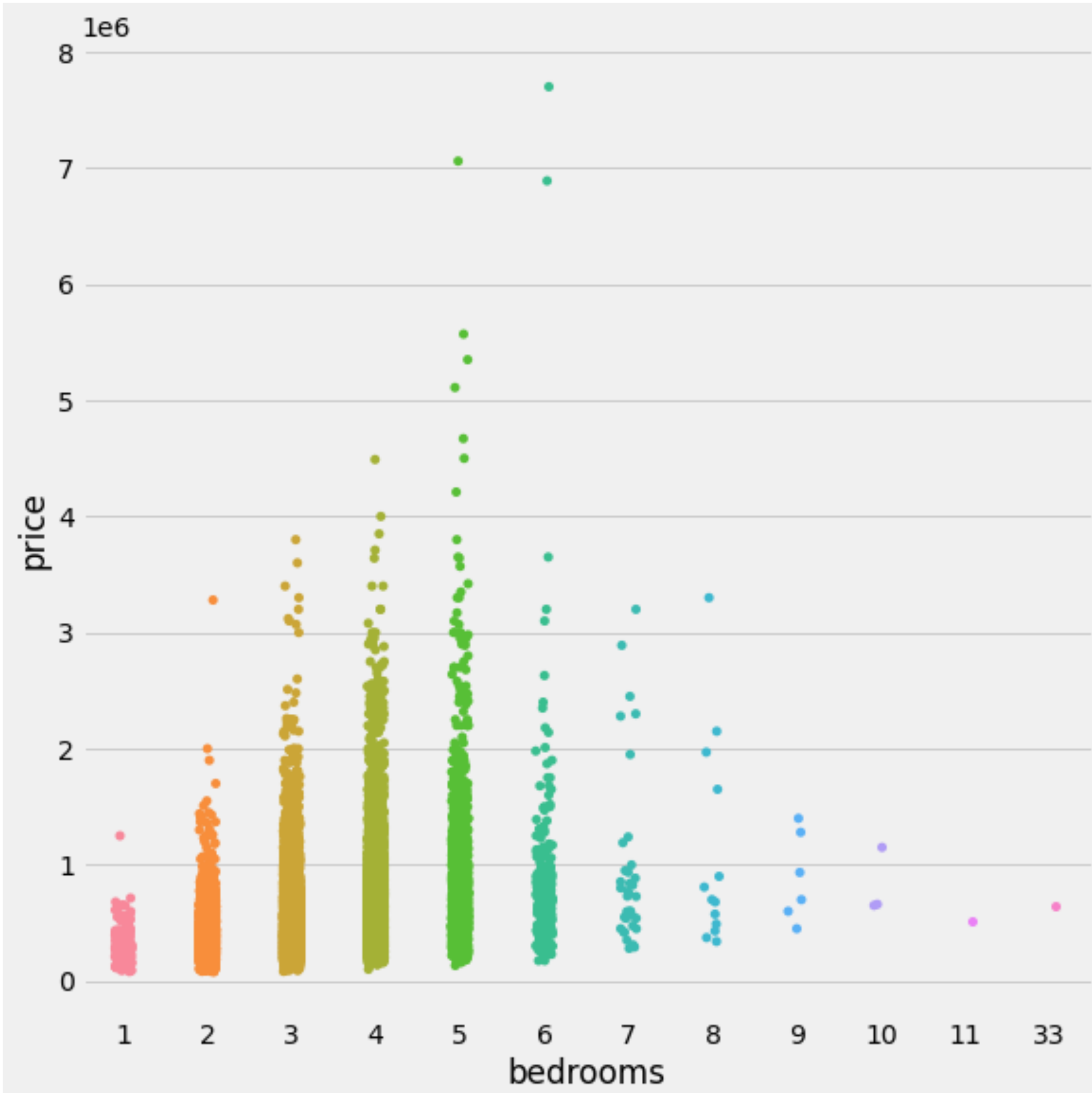
OBSERVATIONS

- There are no more variables which correlate above .75, therefore, variables are now considered independent.

6.3 Outlier Removal

6.3.1 `bedrooms`

In [76]:
```
#check linearity between bedrooms and price
sns.catplot(data=df_explore, x='bedrooms', y='price', height=8);
```



OBSERVATIONS

- I believe a single model will struggle with accurately predicting homes with less than 6 homes with homes with 6 or more bedrooms.

ACTIONS

- I will keep only homes with 5 or fewer bedrooms

In [77]:

```
#remove outliers
df_explore = df_explore.loc[df_explore['bedrooms'] <= 5]
len(df_explore)
```

Out[77]: 21202

7 Model

In [78]:
```
#create a copy of the explore dataframe
df_model_base = df_explore.copy()
df_model_base
```

Out[78]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	c
0	2014-10-13	221,900.0	3	1.0	1180	5650	1.0	0.0	0.0	
1	2014-12-09	538,000.0	3	2.25	2570	7242	2.0	0.0	0.0	
2	2015-02-25	180,000.0	2	1.0	770	10000	1.0	0.0	0.0	
3	2014-12-09	604,000.0	4	3.0	1960	5000	1.0	0.0	0.0	
4	2015-02-18	510,000.0	3	2.0	1680	8080	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	
21592	2014-05-21	360,000.0	3	2.5	1530	1131	3.0	0.0	0.0	
21593	2015-02-23	400,000.0	4	2.5	2310	5813	2.0	0.0	0.0	
21594	2014-06-23	402,101.0	2	0.75	1020	1350	2.0	0.0	0.0	
21595	2015-01-16	400,000.0	3	2.5	1600	2388	2.0	0.0	0.0	
21596	2014-10-15	325,000.0	2	0.75	1020	1076	2.0	0.0	0.0	

21202 rows × 17 columns

7.1 Model Preprocessing

7.1.1 Column Drop

7.1.1.1 `yr_built` Column

I will be removing `yr_built` as it is related to the new column I created named `home_age`

In [79]:
```
#drop the yr_built column
df_model_base.drop(columns='yr_built', inplace=True)
```

7.1.1.2 `date` Column

The `date` column represents the sale date which I do not think is relevant to the model's output since it is a datetime object.

In [80]:
```
#drop the date column
df_model_base.drop(columns='date', inplace=True)
```

7.1.1.3 `yr_sold` Column

The `yr_sold` column I created in order to create the `home_age` column. It is no longer needed.

In [81]:
```
#drop the yr_sold column
df_model_base.drop(columns='yr_sold', inplace=True)
```

7.2 Model 1

- The data is now ready for the first model run. So far, I have taken the following steps:
    1. Removed irrelevant columns
    2. Removed some outliers in the raw data
    3. Removed columns due to 2-variable multicollinearity

7.2.1 Model Creation

I will now create the initial model by copying the df_model_original dataframe.

In [82]:
```
#create a new model dataframe
df_model_1 = df_model_base.copy()
df_model_1
```

Out[82]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditio
0	221,900.0	3	1.0	1180	5650	1.0	0.0	0.0	
1	538,000.0	3	2.25	2570	7242	2.0	0.0	0.0	
2	180,000.0	2	1.0	770	10000	1.0	0.0	0.0	
3	604,000.0	4	3.0	1960	5000	1.0	0.0	0.0	
4	510,000.0	3	2.0	1680	8080	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	.
21592	360,000.0	3	2.5	1530	1131	3.0	0.0	0.0	
21593	400,000.0	4	2.5	2310	5813	2.0	0.0	0.0	
21594	402,101.0	2	0.75	1020	1350	2.0	0.0	0.0	
21595	400,000.0	3	2.5	1600	2388	2.0	0.0	0.0	
21596	325,000.0	2	0.75	1020	1076	2.0	0.0	0.0	

21202 rows × 14 columns

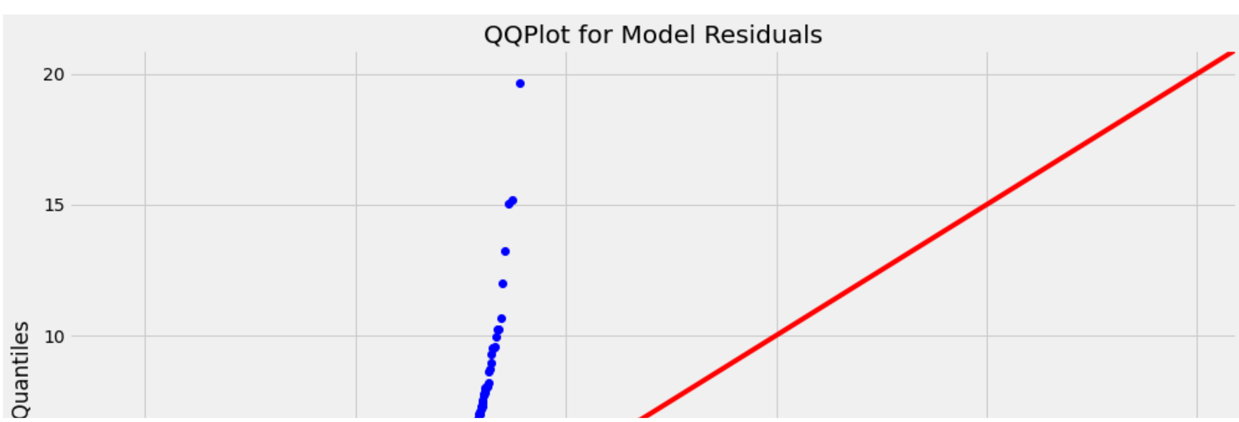
In [83]:
```
#define indpendent and dependent variables
x_cols = df_model_1.drop(columns='price').columns
y_col = 'price'
#run funciton to create model and check assumptions
model_1 = fit_new_model(df_model_1, x_cols=x_cols, y_col=y_col, nc
```

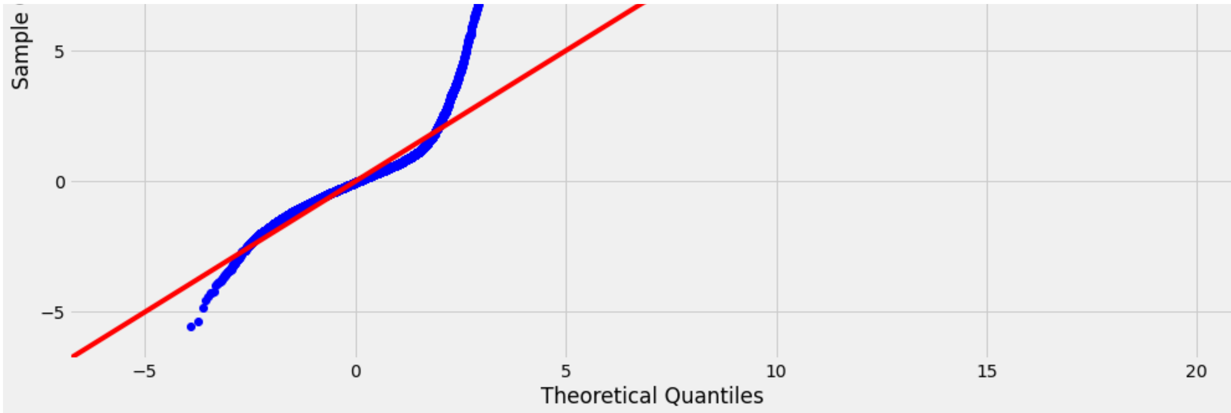
	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	g
0	221,900.0	-0.39	-1.47	-0.98	-0.23	-0.91	-0.11	-0.3	-0.63	-
1	538,000.0	-0.39	0.2	0.57	-0.19	0.94	-0.11	-0.3	-0.63	-
2	180,000.0	-1.6	-1.47	-1.44	-0.12	-0.91	-0.11	-0.3	-0.63	-
3	604,000.0	0.81	1.2	-0.11	-0.24	-0.91	-0.11	-0.3	2.45	-
4	510,000.0	-0.39	-0.13	-0.42	-0.17	-0.91	-0.11	-0.3	-0.63	

```
                            OLS Regression Results
======================================================================
=========
Dep. Variable:                  price   R-squared:
0.646
```

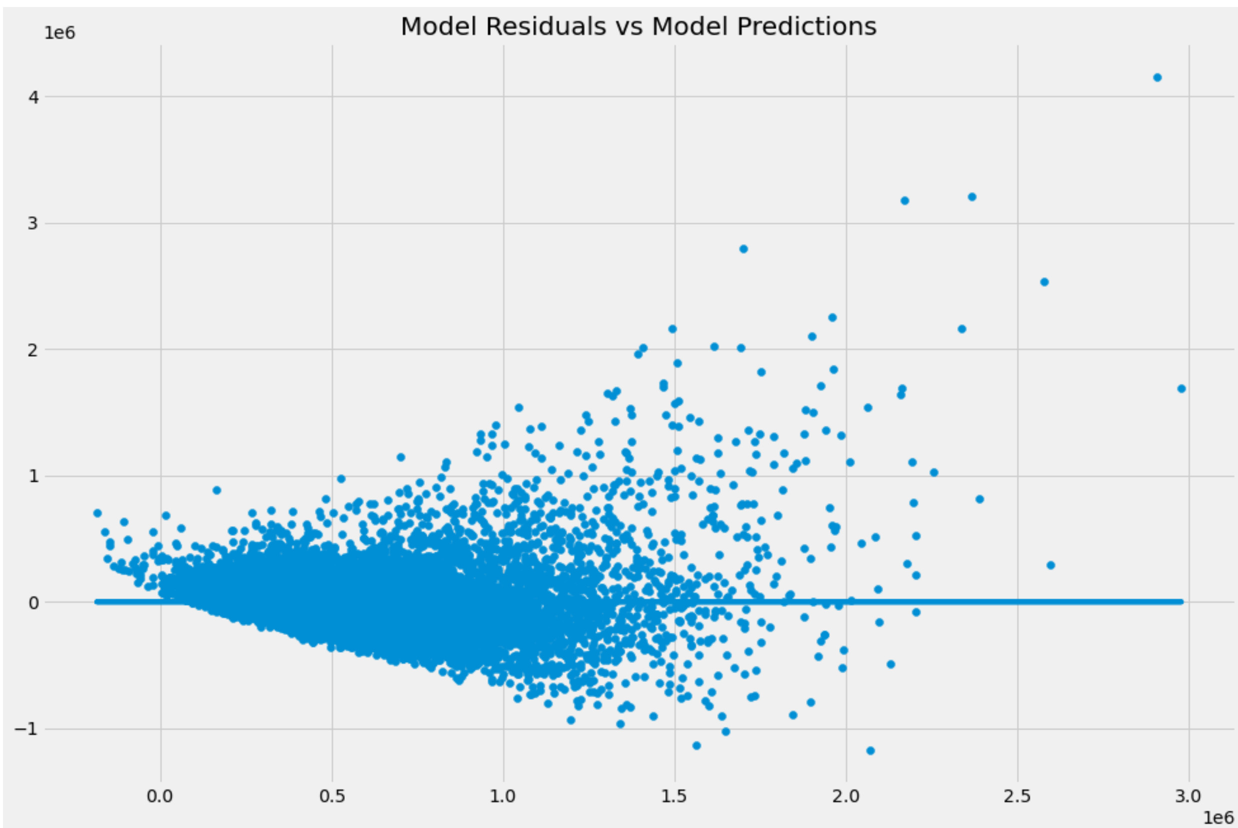
```
Model:                            OLS   Adj. R-squared:
0.646
Method:                 Least Squares   F-statistic:
2972.
Date:                Thu, 29 Apr 2021   Prob (F-statistic):
0.00
Time:                        20:04:56   Log-Likelihood:            -2
.9003e+05
No. Observations:               21202   AIC:
5.801e+05
Df Residuals:                   21188   BIC:
5.802e+05
Df Model:                          13
Covariance Type:            nonrobust
======================================================================
==========
                  coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
----------
Intercept    5.354e+05   1450.805    369.027      0.000    5.33e+05
5.38e+05
bedrooms    -3.647e+04   1871.197    -19.490      0.000   -4.01e+04
-3.28e+04
bathrooms    3.246e+04   2668.998     12.160      0.000    2.72e+04
3.77e+04
sqft_living  1.496e+05   2985.019     50.109      0.000    1.44e+05
1.55e+05
sqft_lot    -1.041e+04   1496.929     -6.952      0.000   -1.33e+04
-7471.874
floors       1.474e+04   1970.821      7.480      0.000    1.09e+04
1.86e+04
waterfront   3.911e+04   1650.641     23.695      0.000    3.59e+04
4.23e+04
view          3.44e+04   1752.703     19.627      0.000     3.1e+04
3.78e+04
condition    1.238e+04   1610.528      7.687      0.000    9223.970
1.55e+04
grade        1.453e+05   2496.258     58.191      0.000     1.4e+05
1.5e+05
zipcode     -2686.6331   1642.800     -1.635      0.102   -5906.645
533.379
basement     3766.4128   1692.949      2.225      0.026     448.105
7084.721
renovated    4644.8809   1529.273      3.037      0.002    1647.389
7642.373
home_age     1.028e+05   2159.510     47.601      0.000    9.86e+04
1.07e+05
======================================================================
=========
```

```
Omnibus:                    14041.588   Durbin-Watson:
1.979
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          6
22863.894
Skew:                           2.610   Prob(JB):
0.00
Kurtosis:                      29.035   Cond. No.
4.88
======================================================================
=========

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


VIF Multicollinearity Test Results
======================================================================
=================================

[('bedrooms', 1.6634151863007192),
 ('bathrooms', 3.384216274463494),
 ('sqft_living', 4.233073367969656),
 ('sqft_lot', 1.0645449383303005),
 ('floors', 1.8452529576458807),
 ('waterfront', 1.2943949642942094),
 ('view', 1.4594126334420228),
 ('condition', 1.2322485777082728),
 ('grade', 2.9603332351927296),
 ('zipcode', 1.282126447422387),
 ('basement', 1.3615986624041567),
 ('renovated', 1.111045269347492),
 ('home_age', 2.2155003722600553)]


Normality Test Results
======================================================================
=================================
```



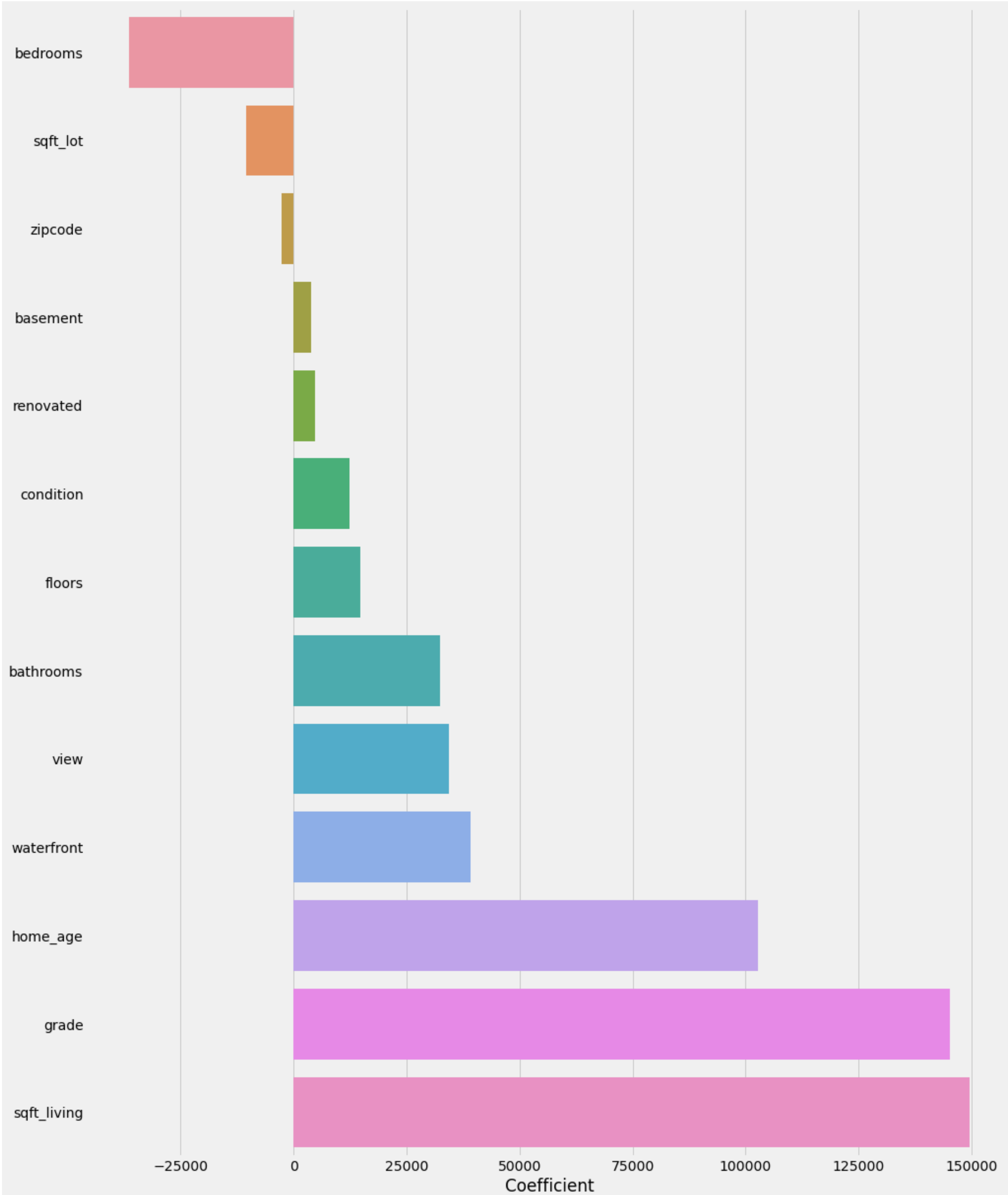



```
Homoscedasticity Test Results
================================================================================================================
```



In [84]:
```
#create dataframe of feature coefficients
coefficients_m1_df = pd.DataFrame(model_1.params, columns=['Coeffi
coefficients_m1_df.drop('Intercept', inplace=True)
coefficients_m1_df = coefficients_m1_df.sort_values(by='Coefficier
```

In [85]:
```
#bar plot showing coefficients
fig, ax = plt.subplots(figsize=(15,20))
sns.barplot(data = coefficients_m1_df, y=coefficients_m1_df.index,
```



7.2.2 Model Interpretation

OBSERVATOINS

- Adjusted R-Squared of 0.645
- All features with significant p-values except for `zipcode` and `basement`
- The most positively correlated features to price are `sqft_living` , `grade` and `home_age`
- The most negatively correlated features to price are `bedrooms` , `zipcode` and `sqft_lot`
- No multicollinearity found
- Residuals not normal on the high end of the distribution
- I am seeing heteroscedasticity along the bottom edge plus as the price gets higher

ACTIONS

- I will look at one hot encoding `zipcode`

7.2.3 Model Tuning

7.2.3.1 OHE Columns

I will evaluate `zipcode` and `grade` for OHE in order to better model this feature.

In [86]:

```
#check column names
df_model_base.columns
```

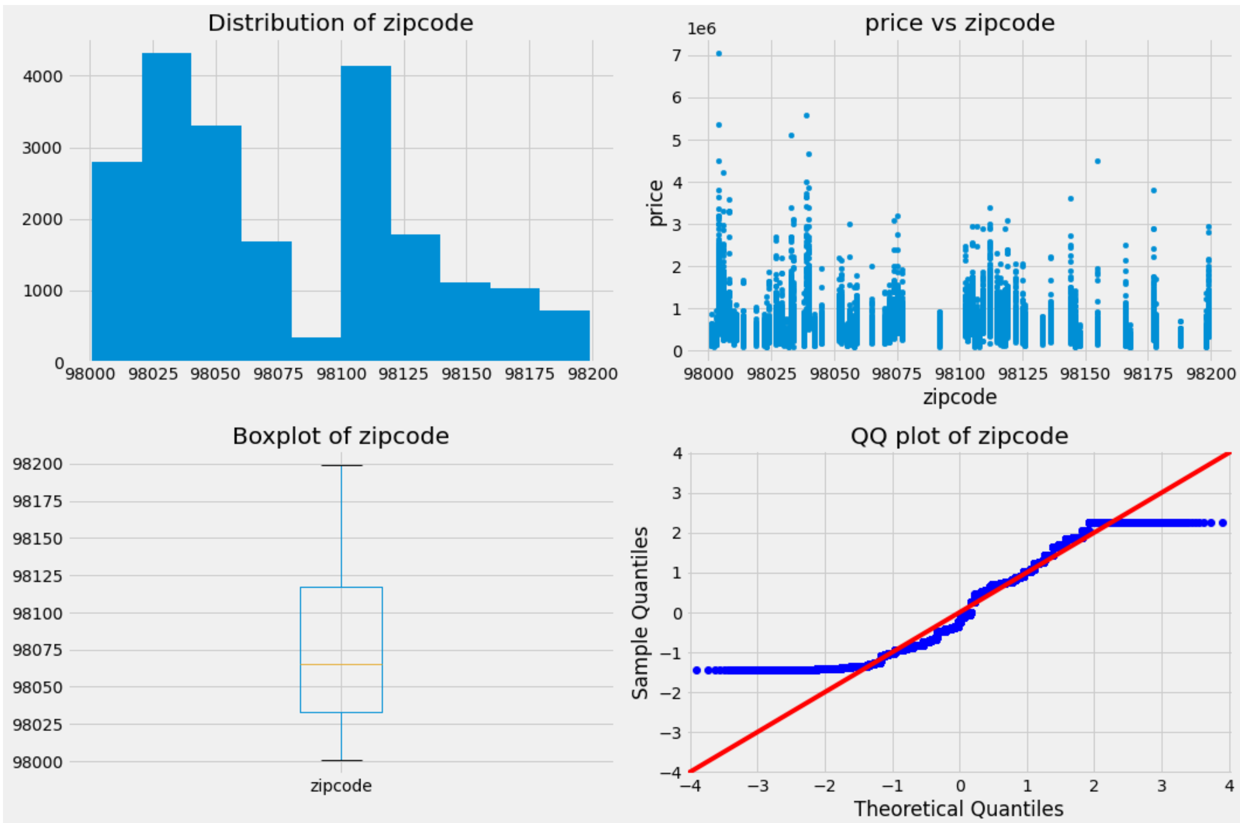
Out[86]:

```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', '
floors',
       'waterfront', 'view', 'condition', 'grade', 'zipcode', 'baseme
nt',
       'renovated', 'home_age'],
      dtype='object')
```

In [87]:

```
#investigate zipcode
get_plots(df_model_base,'zipcode')
```

```
count            21,202.0
mean     98,077.89897179512
std       53.50334101280348
min              98,001.0
25%              98,033.0
50%              98,065.0
75%              98,117.0
max              98,199.0
Name: zipcode, dtype: float64
```



OBSERVATIONS

- `zipcode` seems to be categorical and needs to be hot-one encoded to improve the model since it has a high coefficient.

In [88]:
```
#fit the data
cat_zipcode = ['zipcode']
encoder = OneHotEncoder(drop='first', sparse=False)
encoder.fit(df_model_base[cat_zipcode])
```

Out[88]:
```
OneHotEncoder(drop='first', sparse=False)
```

In [89]:
```
#transform the data
ohe_vars = encoder.transform(df_model_base[cat_zipcode])
ohe_vars
```

Out[89]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [90]:
```
#get the features
encoder.get_feature_names(cat_zipcode)
```

Out[90]:
```
array(['zipcode_98002', 'zipcode_98003', 'zipcode_98004', 'zipcode_98
005',
       'zipcode_98006', 'zipcode_98007', 'zipcode_98008', 'zipcode_98
010',
       'zipcode_98011', 'zipcode_98014', 'zipcode_98019', 'zipcode_98
022',
       'zipcode_98023', 'zipcode_98024', 'zipcode_98027', 'zipcode_98
028',
       'zipcode_98029', 'zipcode_98030', 'zipcode_98031', 'zipcode_98
032',
       'zipcode_98033', 'zipcode_98034', 'zipcode_98038', 'zipcode_98
039',
       'zipcode_98040', 'zipcode_98042', 'zipcode_98045', 'zipcode_98
052',
       'zipcode_98053', 'zipcode_98055', 'zipcode_98056', 'zipcode_98
058',
       'zipcode_98059', 'zipcode_98065', 'zipcode_98070', 'zipcode_98
072',
       'zipcode_98074', 'zipcode_98075', 'zipcode_98077', 'zipcode_98
092',
       'zipcode_98102', 'zipcode_98103', 'zipcode_98105', 'zipcode_98
106',
       'zipcode_98107', 'zipcode_98108', 'zipcode_98109', 'zipcode_98
112',
       'zipcode_98115', 'zipcode_98116', 'zipcode_98117', 'zipcode_98
118',
       'zipcode_98119', 'zipcode_98122', 'zipcode_98125', 'zipcode_98
126',
       'zipcode_98133', 'zipcode_98136', 'zipcode_98144', 'zipcode_98
146',
       'zipcode_98148', 'zipcode_98155', 'zipcode_98166', 'zipcode_98
168',
       'zipcode_98177', 'zipcode_98178', 'zipcode_98188', 'zipcode_98
198',
       'zipcode_98199'], dtype=object)
```

In [91]:

```
#convert to dataframe
df_cat_zipcode = pd.DataFrame(ohe_vars, columns=encoder.get_featur
df_cat_zipcode
```

Out[91]:

	zipcode_98002	zipcode_98003	zipcode_98004	zipcode_98005	zipcode_98006	zipcode_9
0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	
21592	0.0	0.0	0.0	0.0	0.0	
21593	0.0	0.0	0.0	0.0	0.0	
21594	0.0	0.0	0.0	0.0	0.0	
21595	0.0	0.0	0.0	0.0	0.0	
21596	0.0	0.0	0.0	0.0	0.0	

21202 rows × 69 columns

In [92]:
```
#concat original dataframe to zipcode dataframe and prepare for mo
df_model_base = pd.concat([df_model_base.drop(['zipcode'], axis=1)
df_model_base
```

Out[92]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditio
0	221,900.0	3	1.0	1180	5650	1.0	0.0	0.0	
1	538,000.0	3	2.25	2570	7242	2.0	0.0	0.0	
2	180,000.0	2	1.0	770	10000	1.0	0.0	0.0	
3	604,000.0	4	3.0	1960	5000	1.0	0.0	0.0	
4	510,000.0	3	2.0	1680	8080	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	.
21592	360,000.0	3	2.5	1530	1131	3.0	0.0	0.0	
21593	400,000.0	4	2.5	2310	5813	2.0	0.0	0.0	
21594	402,101.0	2	0.75	1020	1350	2.0	0.0	0.0	
21595	400,000.0	3	2.5	1600	2388	2.0	0.0	0.0	
21596	325,000.0	2	0.75	1020	1076	2.0	0.0	0.0	

21202 rows × 82 columns

7.3 Model 2

Going to refit the model with the new OHE `zipcode` columns

In [93]:
```
#copy the base model with the OHE column
df_model_2 = df_model_base.copy()
df_model_2
```

Out[93]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditio
0	221,900.0	3	1.0	1180	5650	1.0	0.0	0.0	
1	538,000.0	3	2.25	2570	7242	2.0	0.0	0.0	
2	180,000.0	2	1.0	770	10000	1.0	0.0	0.0	
3	604,000.0	4	3.0	1960	5000	1.0	0.0	0.0	
4	510,000.0	3	2.0	1680	8080	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	.
21592	360,000.0	3	2.5	1530	1131	3.0	0.0	0.0	
21593	400,000.0	4	2.5	2310	5813	2.0	0.0	0.0	
21594	402,101.0	2	0.75	1020	1350	2.0	0.0	0.0	
21595	400,000.0	3	2.5	1600	2388	2.0	0.0	0.0	
21596	325,000.0	2	0.75	1020	1076	2.0	0.0	0.0	

21202 rows × 82 columns

7.3.1 Model Creation

In [94]:
```
#define indpendent and dependent variables
x_cols = df_model_2.drop(columns='price').columns
y_col = 'price'
#run funciton to create model and check assumptions
model_2 = fit_new_model(df_model_2, x_cols=x_cols, y_col=y_col, nc
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	gı
0	221,900.0	-0.39	-1.47	-0.98	-0.23	-0.91	-0.11	-0.3	-0.63	-
1	538,000.0	-0.39	0.2	0.57	-0.19	0.94	-0.11	-0.3	-0.63	-
2	180,000.0	-1.6	-1.47	-1.44	-0.12	-0.91	-0.11	-0.3	-0.63	-
3	604,000.0	0.81	1.2	-0.11	-0.24	-0.91	-0.11	-0.3	2.45	-
4	510,000.0	-0.39	-0.13	-0.42	-0.17	-0.91	-0.11	-0.3	-0.63	

5 rows × 82 columns

```
                            OLS Regression Results
======================================================================
=========
Dep. Variable:                  price   R-squared:
0.805
Model:                            OLS   Adj. R-squared:
0.804
Method:                 Least Squares   F-statistic:
1078.
Date:                Thu, 29 Apr 2021   Prob (F-statistic):
0.00
Time:                        20:05:31   Log-Likelihood:            -2
.8369e+05
No. Observations:               21202   AIC:
5.676e+05
Df Residuals:                   21120   BIC:
5.682e+05
Df Model:                          81
Covariance Type:            nonrobust
======================================================================
============
                    coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
------------
Intercept      5.354e+05   1077.688    496.792      0.000    5.33e+05
5.37e+05
bedrooms      -2.313e+04   1414.376    -16.353      0.000   -2.59e+04
-2.04e+04
bathrooms      1.989e+04   1998.509      9.951      0.000     1.6e+04
2.38e+04
sqft_living    1.713e+05   2328.074     73.591      0.000    1.67e+05
1.76e+05
sqft_lot       7738.1094   1187.196      6.518      0.000    5411.114
1.01e+04
floors        -1.945e+04   1629.076    -11.937      0.000   -2.26e+04
-1.63e+04
waterfront     4.507e+04   1241.271     36.312      0.000    4.26e+04
4.75e+04
view           4.121e+04   1336.570     30.835      0.000    3.86e+04
4.38e+04
condition      1.382e+04   1226.372     11.266      0.000    1.14e+04
1.62e+04
grade           7.31e+04   2017.855     36.226      0.000    6.91e+04
7.71e+04
basement      -2.747e+04   1359.008    -20.210      0.000   -3.01e+04
-2.48e+04
renovated      5436.5675   1144.221      4.751      0.000    3193.806
7679.329
```

```
home_age       2.305e+04   1860.918     12.385      0.000    1.94e+04
2.67e+04
zipcode_98002  2722.2823   1331.915      2.044      0.041     111.628
5332.937
zipcode_98003 -1571.1554   1426.969     -1.101      0.271   -4368.123
1225.812
zipcode_98004  9.318e+04   1482.031     62.870      0.000    9.03e+04
9.61e+04
zipcode_98005  2.672e+04   1305.470     20.465      0.000    2.42e+04
2.93e+04
zipcode_98006   4.01e+04   1658.516     24.178      0.000    3.68e+04
4.33e+04
zipcode_98007  1.895e+04   1261.495     15.021      0.000    1.65e+04
2.14e+04
zipcode_98008  2.918e+04   1432.499     20.370      0.000    2.64e+04
3.2e+04
zipcode_98010  4900.5970   1220.045      4.017      0.000    2509.215
7291.979
zipcode_98011  1.198e+04   1330.517      9.007      0.000    9376.039
1.46e+04
zipcode_98014  7457.4279   1260.858      5.915      0.000    4986.050
9928.806
zipcode_98019  8508.1747   1330.083      6.397      0.000    5901.111
1.11e+04
zipcode_98022 -2826.9948   1389.871     -2.034      0.042   -5551.249
-102.741
zipcode_98023 -4522.7437   1645.266     -2.749      0.006   -7747.591
-1297.896
zipcode_98024  9338.2674   1197.100      7.801      0.000    6991.860
1.17e+04
zipcode_98027  2.375e+04   1567.311     15.156      0.000    2.07e+04
2.68e+04
zipcode_98028  1.375e+04   1430.103      9.617      0.000     1.1e+04
1.66e+04
zipcode_98029  2.558e+04   1484.042     17.234      0.000    2.27e+04
2.85e+04
zipcode_98030   948.3135   1398.817      0.678      0.498   -1793.475
3690.102
zipcode_98031  2156.6356   1419.324      1.519      0.129    -625.347
4938.619
zipcode_98032   288.2005   1246.571      0.231      0.817   -2155.174
2731.575
zipcode_98033  5.128e+04   1586.482     32.320      0.000    4.82e+04
5.44e+04
zipcode_98034  3.251e+04   1686.355     19.276      0.000    2.92e+04
3.58e+04
zipcode_98038  5457.3662   1732.117      3.151      0.002    2062.284
8852.448
zipcode_98039  6.026e+04   1156.798     52.094      0.000     5.8e+04
6.25e+04
```

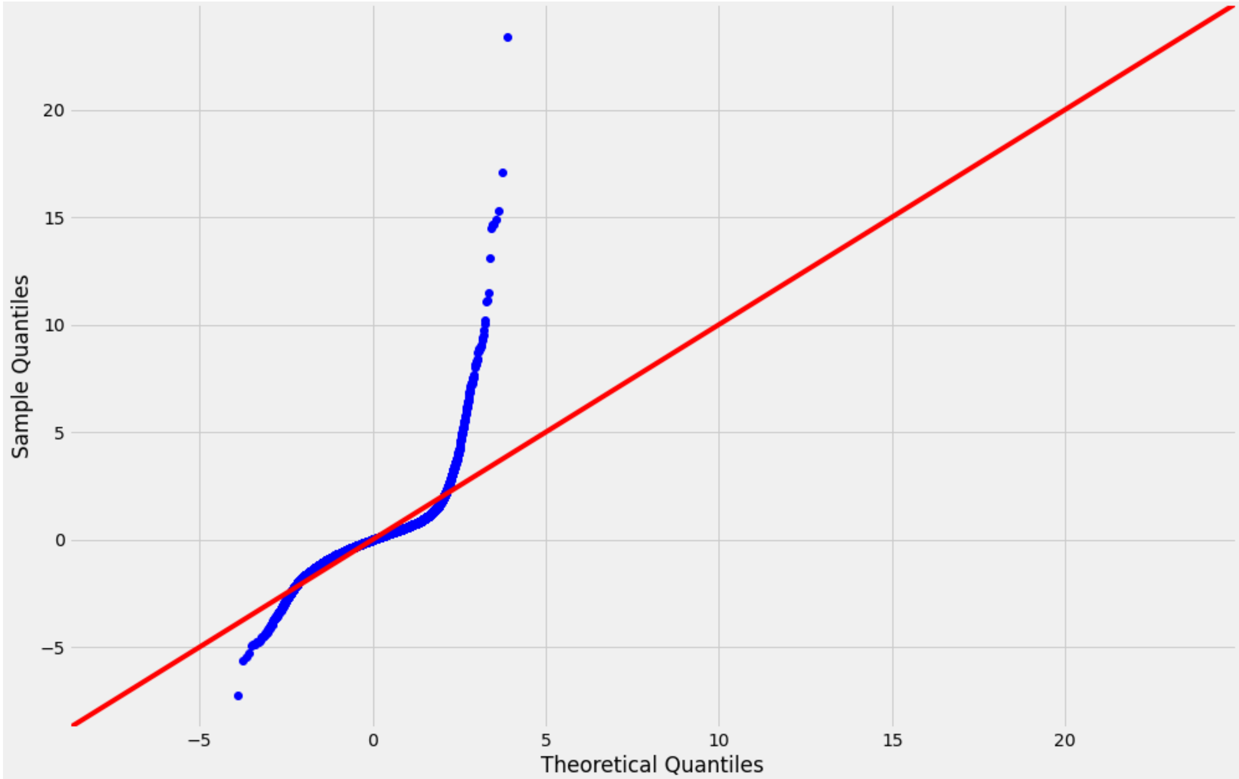
zipcode_98040	5.738e+04	1444.063	39.736	0.000	5.46e+04	6.02e+04
zipcode_98042	1301.4871	1689.224	0.770	0.441	-2009.520	4612.494
zipcode_98045	9094.5574	1367.145	6.652	0.000	6414.849	1.18e+04
zipcode_98052	3.773e+04	1716.027	21.985	0.000	3.44e+04	4.11e+04
zipcode_98053	2.69e+04	1569.454	17.139	0.000	2.38e+04	3e+04
zipcode_98055	5092.3968	1413.187	3.603	0.000	2322.443	7862.350
zipcode_98056	1.327e+04	1554.522	8.535	0.000	1.02e+04	1.63e+04
zipcode_98058	4627.8194	1602.854	2.887	0.004	1486.104	7769.535
zipcode_98059	1.336e+04	1618.155	8.257	0.000	1.02e+04	1.65e+04
zipcode_98065	1.09e+04	1464.593	7.444	0.000	8031.450	1.38e+04
zipcode_98070	1597.3900	1262.141	1.266	0.206	-876.502	4071.282
zipcode_98072	1.764e+04	1423.847	12.388	0.000	1.48e+04	2.04e+04
zipcode_98074	2.513e+04	1606.022	15.645	0.000	2.2e+04	2.83e+04
zipcode_98075	2.345e+04	1527.916	15.348	0.000	2.05e+04	2.64e+04
zipcode_98077	1.225e+04	1349.465	9.078	0.000	9605.845	1.49e+04
zipcode_98092	-4669.1122	1503.234	-3.106	0.002	-7615.566	-1722.658
zipcode_98102	3.223e+04	1243.120	25.924	0.000	2.98e+04	3.47e+04
zipcode_98103	5.473e+04	1792.283	30.537	0.000	5.12e+04	5.82e+04
zipcode_98105	4.687e+04	1393.187	33.639	0.000	4.41e+04	4.96e+04
zipcode_98106	1.859e+04	1492.394	12.454	0.000	1.57e+04	2.15e+04
zipcode_98107	3.795e+04	1443.689	26.290	0.000	3.51e+04	4.08e+04
zipcode_98108	1.153e+04	1327.437	8.684	0.000	8925.377	1.41e+04
zipcode_98109	3.382e+04	1250.255	27.050	0.000	3.14e+04	3.63e+04
zipcode_98112	6.713e+04	1460.292	45.970	0.000	6.43e+04	7e+04
zipcode_98115	5.412e+04	1758.378	30.779	0.000	5.07e+04	5.76e+04

```
zipcode_98116  3.538e+04   1512.564     23.390      0.000    3.24e+04
3.83e+04
zipcode_98117  4.981e+04   1732.588     28.749      0.000    4.64e+04
5.32e+04
zipcode_98118   2.66e+04   1667.402     15.950      0.000    2.33e+04
2.99e+04
zipcode_98119  4.253e+04   1351.677     31.464      0.000    3.99e+04
4.52e+04
zipcode_98122  3.919e+04   1478.135     26.510      0.000    3.63e+04
4.21e+04
zipcode_98125  2.784e+04   1563.339     17.811      0.000    2.48e+04
3.09e+04
zipcode_98126  2.456e+04   1526.419     16.090      0.000    2.16e+04
2.76e+04
zipcode_98133    2.5e+04   1647.510     15.174      0.000    2.18e+04
2.82e+04
zipcode_98136  2.813e+04   1429.891     19.675      0.000    2.53e+04
3.09e+04
zipcode_98144  3.491e+04   1520.378     22.959      0.000    3.19e+04
3.79e+04
zipcode_98146  1.336e+04   1439.574      9.279      0.000    1.05e+04
1.62e+04
zipcode_98148  3654.4997   1160.554      3.149      0.002    1379.725
5929.274
zipcode_98155  2.163e+04   1598.922     13.528      0.000    1.85e+04
2.48e+04
zipcode_98166  7549.7382   1403.584      5.379      0.000    4798.607
1.03e+04
zipcode_98168  8793.2262   1427.293      6.161      0.000    5995.622
1.16e+04
zipcode_98177  2.352e+04   1411.006     16.671      0.000    2.08e+04
2.63e+04
zipcode_98178  4892.2972   1416.547      3.454      0.001    2115.758
7668.837
zipcode_98188  3074.1888   1258.706      2.442      0.015     607.029
5541.348
zipcode_98198   282.1254   1431.293      0.197      0.844   -2523.318
3087.569
zipcode_98199  4.602e+04   1498.884     30.701      0.000    4.31e+04
4.9e+04
======================================================================
=========
Omnibus:                    17701.447   Durbin-Watson:
1.992
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            19
37452.310
Skew:                           3.428   Prob(JB):
0.00
Kurtosis:                      49.326   Cond. No.
15.1
```

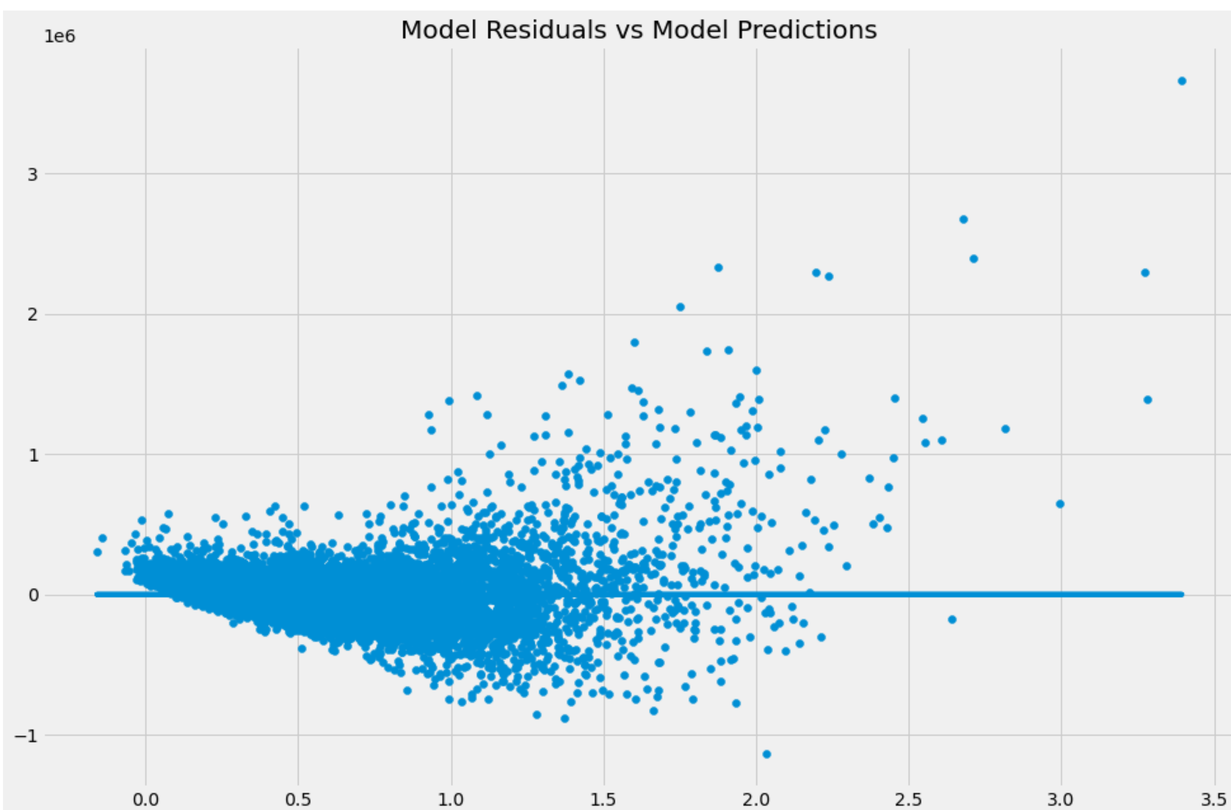
```
==============================================================================
=========

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


VIF Multicollinearity Test Results
==============================================================================
=================================

[('bedrooms', 1.7223561676900025),
 ('bathrooms', 3.438786877914085),
 ('sqft_living', 4.666450946830134),
 ('sqft_lot', 1.2134954436912395),
 ('floors', 2.2849447077338643),
 ('waterfront', 1.3265590973456276),
 ('view', 1.538072485291057),
 ('condition', 1.2949032825650884),
 ('grade', 3.505685940045277),
 ('basement', 1.5901460258875515),
 ('renovated', 1.1272318867279),
 ('home_age', 2.981587441406763),
 ('zipcode_98002', 1.5273765581844436),
 ('zipcode_98003', 1.7531618985631878),
 ('zipcode_98004', 1.891070112982308),
 ('zipcode_98005', 1.4673277441232027),
 ('zipcode_98006', 2.36827667463131),
 ('zipcode_98007', 1.3701388886041022),
 ('zipcode_98008', 1.7667767462191037),
 ('zipcode_98010', 1.2815780107864305),
 ('zipcode_98011', 1.5241715620262517),
 ('zipcode_98014', 1.3687545560666137),
 ('zipcode_98019', 1.523177140070671),
 ('zipcode_98022', 1.6631918125439553),
 ('zipcode_98023', 2.330588346695761),
 ('zipcode_98024', 1.2338261905325254),
 ('zipcode_98027', 2.1149655828655076),
 ('zipcode_98028', 1.7608709998222198),
 ('zipcode_98029', 1.896205874365982),
 ('zipcode_98030', 1.684670819098971),
 ('zipcode_98031', 1.7344276196973263),
 ('zipcode_98032', 1.3379113461985932),
 ('zipcode_98033', 2.1670234675110063),
 ('zipcode_98034', 2.448449594175458),
 ('zipcode_98038', 2.5831389994101834),
 ('zipcode_98039', 1.1521486810430575),
 ('zipcode_98040', 1.795417356508061),
 ('zipcode_98042', 2.4567865768251647),
```

```
 ('zipcode_98045', 1.6092455495064428),
 ('zipcode_98052', 2.535370810516652),
 ('zipcode_98053', 2.120754321243446),
 ('zipcode_98055', 1.7194603109021782),
 ('zipcode_98056', 2.080590870359287),
 ('zipcode_98058', 2.21197879983812),
 ('zipcode_98059', 2.254413001610954),
 ('zipcode_98065', 1.8468319131922675),
 ('zipcode_98070', 1.3715409228644393),
 ('zipcode_98072', 1.7455002770602046),
 ('zipcode_98074', 2.220731888349811),
 ('zipcode_98075', 2.009980689495965),
 ('zipcode_98077', 1.5678929957027985),
 ('zipcode_98092', 1.945568944458815),
 ('zipcode_98102', 1.3305138002776458),
 ('zipcode_98103', 2.7657070697447446),
 ('zipcode_98105', 1.6711364635150159),
 ('zipcode_98106', 1.917609326102108),
 ('zipcode_98107', 1.7944871059263747),
 ('zipcode_98108', 1.5171238599326309),
 ('zipcode_98109', 1.345831501333294),
 ('zipcode_98112', 1.8359985828365886),
 ('zipcode_98115', 2.662058922969858),
 ('zipcode_98116', 1.9697937318637597),
 ('zipcode_98117', 2.5845427891898085),
 ('zipcode_98118', 2.3937225644597753),
 ('zipcode_98119', 1.5730384266254676),
 ('zipcode_98122', 1.8811419349562721),
 ('zipcode_98125', 2.104261762051141),
 ('zipcode_98126', 2.006046332905773),
 ('zipcode_98133', 2.3369481410882056),
 ('zipcode_98136', 1.7603507341550704),
 ('zipcode_98144', 1.9901987979535123),
 ('zipcode_98146', 1.7842717581324588),
 ('zipcode_98148', 1.159641786660884),
 ('zipcode_98155', 2.201140741937576),
 ('zipcode_98166', 1.6961714433992445),
 ('zipcode_98168', 1.7539600500768904),
 ('zipcode_98177', 1.7141593445611454),
 ('zipcode_98178', 1.7276468680206347),
 ('zipcode_98188', 1.3640856469171478),
 ('zipcode_98198', 1.7638036081764346),
 ('zipcode_98199', 1.934325275960213)]


Normality Test Results
================================================================================
================================
```

QQPlot for Model Residuals




```
Homoscedasticity Test Results
================================================================================================
```

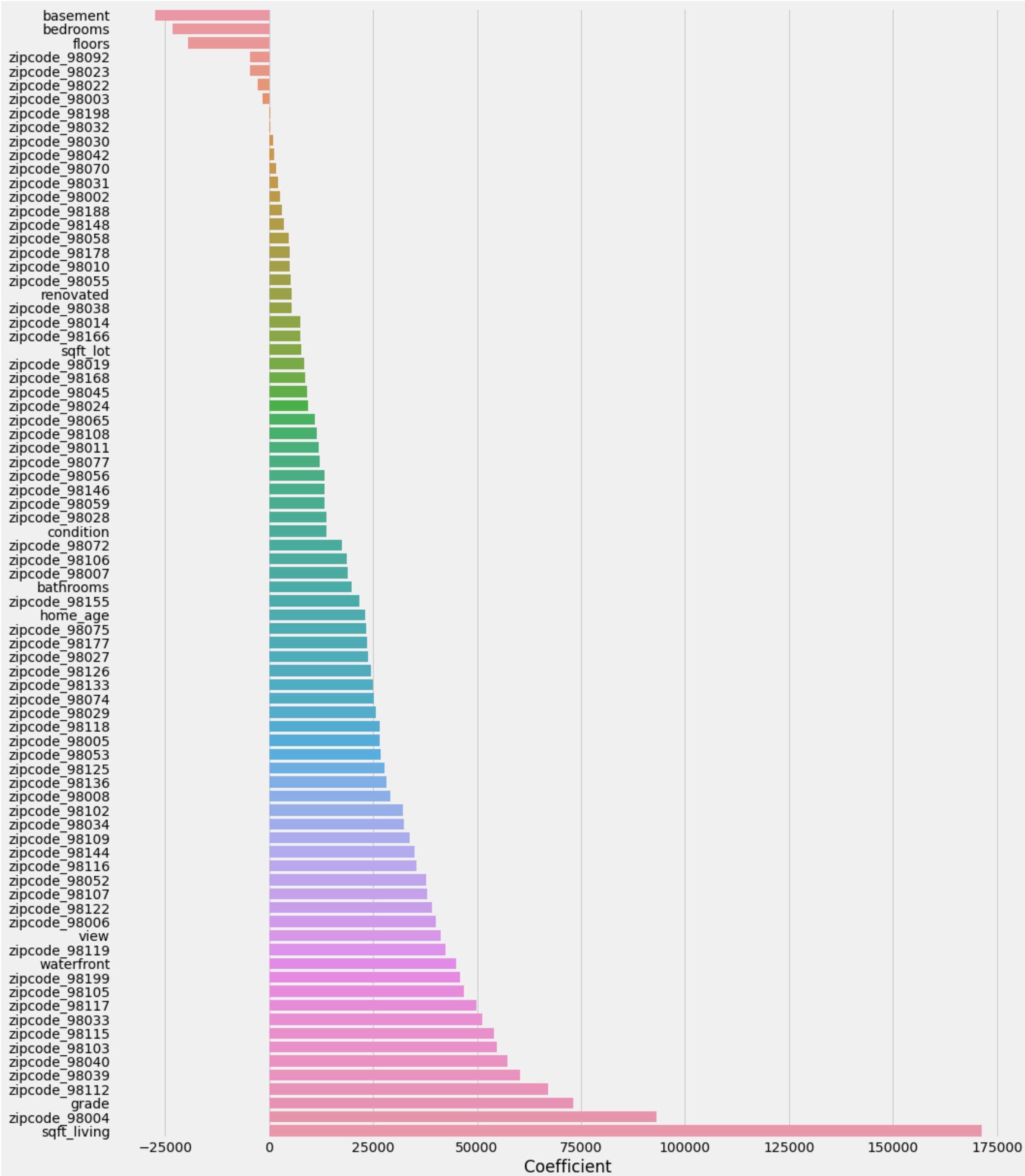


1e6

In [95]:

```
#create dataframe of feature coefficients
coefficients_m2_df = pd.DataFrame(model_2.params, columns=['Coeffi
coefficients_m2_df.drop('Intercept', inplace=True)
coefficients_m2_df = coefficients_m2_df.sort_values(by='Coefficier
```

In [96]:
```
#bar plot showing coefficients
fig, ax = plt.subplots(figsize=(15,20))
sns.barplot(data = coefficients_m2_df, y=coefficients_m2_df.index,
```



7.3.2 Model Interpretation

OBSERVATOINS

- Adjusted R-Squared of 0.804
- All features with significant p-values except for some zipcodes
- The most positively correlated features to price are `sqft_living`, `waterfront`, `grade` and `view`
- The most negatively correlated features to price are `bedrooms`, `basement` and `floors`
- QQ plot shows non-normality amongst the residuals
- Homoscedasticity plot shows a larger spread of residuals in the upper range of `price`

ACTIONS

- I will proceed with removing outliers on `price` due to it not being modeled accurately on the high end

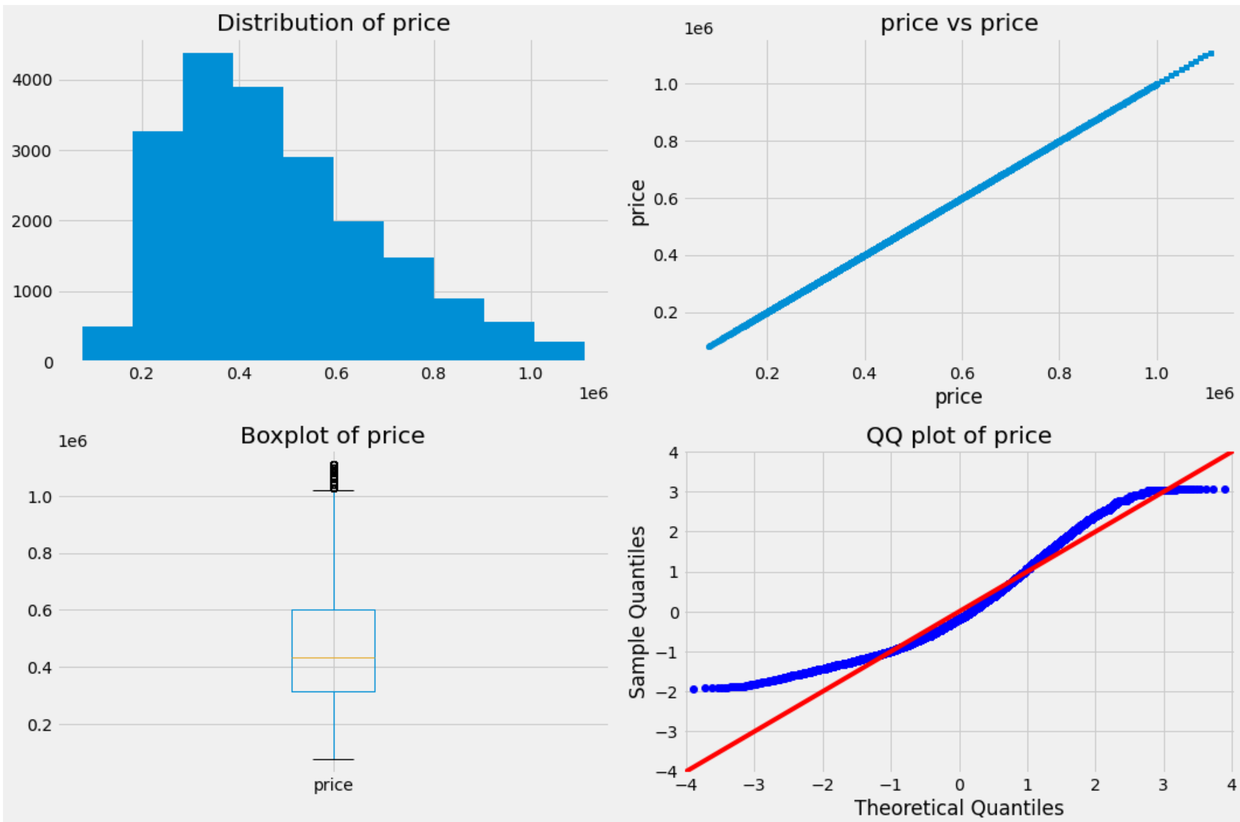
7.3.3 Model Tuning

7.3.3.1 `price` Outlier Removal

I will investigate `price` for outliers.

In [97]:
```
get_plots(df_model_base,'price',outlier='iqr')
```

```
The number of rows removed is 1104
count              21,202.0
mean        535,386.45241015
std       354,857.12923559395
min                78,000.0
25%               320,000.0
50%               450,000.0
75%               639,897.0
max             7,060,000.0
Name: price, dtype: float64
```



OBSERVATIONS

- I will use iqr to remove outliers because there are a lot of outliers on the high side of `price` .

In [98]:

```
#create a copy of model_2 to set up model_3
df_model_3 = df_model_base.copy()
df_model_3
```

Out[98]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditio
0	221,900.0	3	1.0	1180	5650	1.0	0.0	0.0	
1	538,000.0	3	2.25	2570	7242	2.0	0.0	0.0	
2	180,000.0	2	1.0	770	10000	1.0	0.0	0.0	
3	604,000.0	4	3.0	1960	5000	1.0	0.0	0.0	
4	510,000.0	3	2.0	1680	8080	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	.
21592	360,000.0	3	2.5	1530	1131	3.0	0.0	0.0	
21593	400,000.0	4	2.5	2310	5813	2.0	0.0	0.0	
21594	402,101.0	2	0.75	1020	1350	2.0	0.0	0.0	
21595	400,000.0	3	2.5	1600	2388	2.0	0.0	0.0	
21596	325,000.0	2	0.75	1020	1076	2.0	0.0	0.0	

21202 rows × 82 columns

In [101]:

```
#remove outliers based off iqr
df_model_base = outliers(df_model_base, 'price', 'iqr')
df_model_3 = df_model_base.copy()
df_model_3
```

There were 1104 outliers removed.

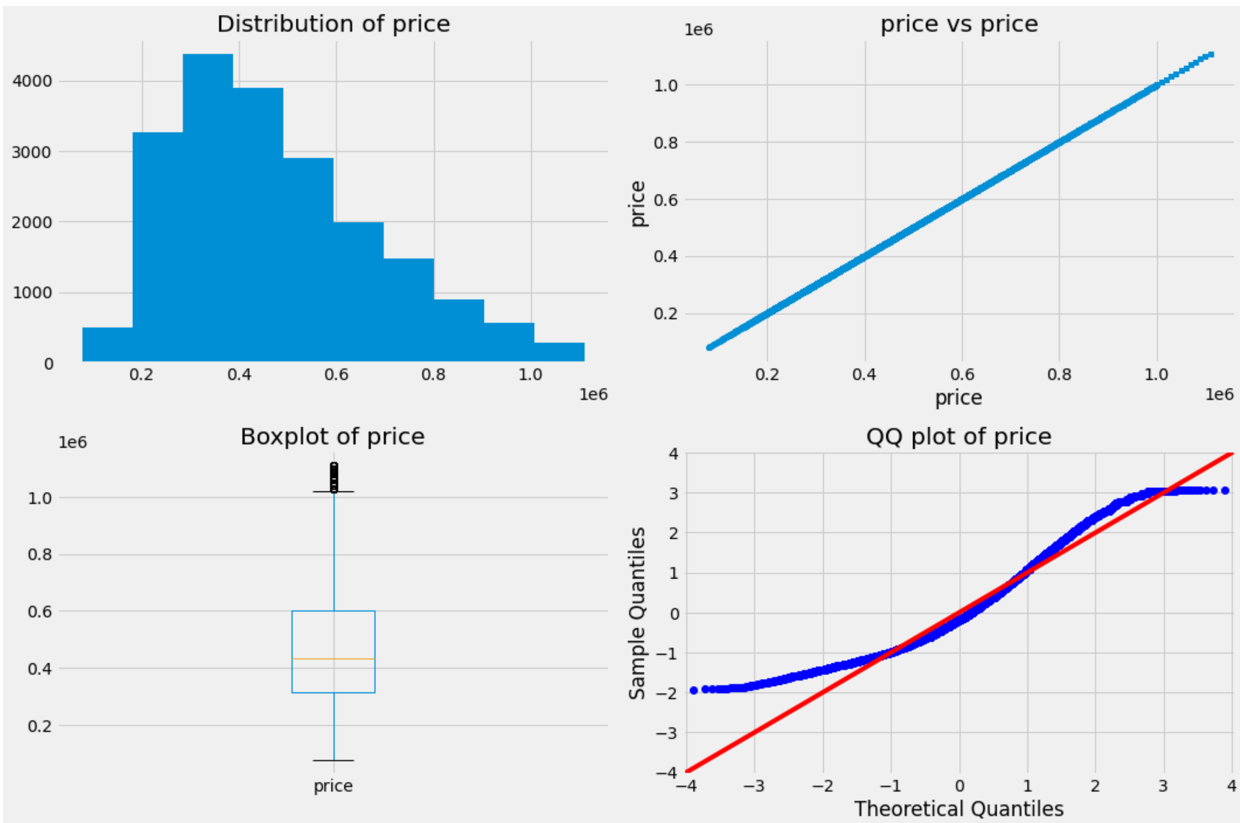
Out[101]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditio
0	221,900.0	3	1.0	1180	5650	1.0	0.0	0.0	
1	538,000.0	3	2.25	2570	7242	2.0	0.0	0.0	
2	180,000.0	2	1.0	770	10000	1.0	0.0	0.0	
3	604,000.0	4	3.0	1960	5000	1.0	0.0	0.0	
4	510,000.0	3	2.0	1680	8080	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	
21592	360,000.0	3	2.5	1530	1131	3.0	0.0	0.0	
21593	400,000.0	4	2.5	2310	5813	2.0	0.0	0.0	
21594	402,101.0	2	0.75	1020	1350	2.0	0.0	0.0	
21595	400,000.0	3	2.5	1600	2388	2.0	0.0	0.0	
21596	325,000.0	2	0.75	1020	1076	2.0	0.0	0.0	

20098 rows × 82 columns

In [102]:
```
#recheck the price column
get_plots(df_model_3,'price',outlier='none')
```

```
count                20,098.0
mean     474,738.45974723855
std      206,719.25387615876
min                  78,000.0
25%                 315,000.0
50%                 435,000.0
75%                 600,000.0
max               1,110,000.0
Name: price, dtype: float64
```



7.4 Model 3

In [103]:
```
#view model_3 dataframe
df_model_3
```

Out[103]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditio
0	221,900.0	3	1.0	1180	5650	1.0	0.0	0.0	
1	538,000.0	3	2.25	2570	7242	2.0	0.0	0.0	
2	180,000.0	2	1.0	770	10000	1.0	0.0	0.0	
3	604,000.0	4	3.0	1960	5000	1.0	0.0	0.0	
4	510,000.0	3	2.0	1680	8080	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	.
21592	360,000.0	3	2.5	1530	1131	3.0	0.0	0.0	
21593	400,000.0	4	2.5	2310	5813	2.0	0.0	0.0	
21594	402,101.0	2	0.75	1020	1350	2.0	0.0	0.0	
21595	400,000.0	3	2.5	1600	2388	2.0	0.0	0.0	
21596	325,000.0	2	0.75	1020	1076	2.0	0.0	0.0	

20098 rows × 82 columns

7.4.1 Model Creation

In [104]:
```
#define indpendent and dependent variables
x_cols = df_model_3.drop(columns='price').columns
y_col = 'price'
#run funciton to create model and check assumptions
model_3 = fit_new_model(df_model_3, x_cols=x_cols, y_col=y_col, nc
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	g
0	221,900.0	-0.35	-1.49	-1.02	-0.22	-0.88	-0.08	-0.27	-0.63	
1	538,000.0	-0.35	0.3	0.8	-0.18	0.98	-0.08	-0.27	-0.63	
2	180,000.0	-1.58	-1.49	-1.56	-0.11	-0.88	-0.08	-0.27	-0.63	
3	604,000.0	0.87	1.38	-0.0	-0.24	-0.88	-0.08	-0.27	2.47	
4	510,000.0	-0.35	-0.06	-0.37	-0.16	-0.88	-0.08	-0.27	-0.63	

5 rows × 82 columns

```
                            OLS Regression Results
==============================================================================
=========
Dep. Variable:                  price   R-squared:
0.827
Model:                            OLS   Adj. R-squared:
0.827
Method:                 Least Squares   F-statistic:
1185.
Date:                Thu, 29 Apr 2021   Prob (F-statistic):
0.00
Time:                        20:10:24   Log-Likelihood:                -2
.5685e+05
No. Observations:               20098   AIC:
5.139e+05
Df Residuals:                   20016   BIC:
5.145e+05
Df Model:                          81
Covariance Type:            nonrobust
==============================================================================
============
                    coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------------
Intercept      4.747e+05    607.019    782.081      0.000    4.74e+05
4.76e+05
bedrooms      -3473.9094    800.631     -4.339      0.000   -5043.212
-1904.607
bathrooms      1.027e+04   1075.216      9.549      0.000    8159.745
1.24e+04
sqft_living    8.813e+04   1215.076     72.529      0.000    8.57e+04
9.05e+04
sqft_lot       1.128e+04    667.011     16.916      0.000    9975.773
1.26e+04
floors        -5523.2436    931.728     -5.928      0.000   -7349.507
-3696.981
waterfront     4792.2301    670.733      7.145      0.000    3477.539
6106.922
view           2.121e+04    698.317     30.377      0.000    1.98e+04
2.26e+04
condition      1.295e+04    689.700     18.769      0.000    1.16e+04
1.43e+04
grade           5.25e+04   1040.823     50.445      0.000    5.05e+04
5.45e+04
basement      -1.294e+04    768.573    -16.843      0.000   -1.45e+04
-1.14e+04
renovated      4859.9082    639.935      7.594      0.000    3605.583
6114.234
home_age         1.7e+04   1061.729     16.008      0.000    1.49e+04
```

```
1.91e+04
zipcode_98002   467.0533    750.313      0.622      0.534   -1003.622
1937.729
zipcode_98003  -588.3876    803.612     -0.732      0.464   -2163.534
986.759
zipcode_98004  4.473e+04    725.487     61.653      0.000    4.33e+04
4.62e+04
zipcode_98005   2.83e+04    724.989     39.040      0.000    2.69e+04
2.97e+04
zipcode_98006  3.778e+04    881.069     42.879      0.000    3.61e+04
3.95e+04
zipcode_98007  1.975e+04    706.903     27.934      0.000    1.84e+04
2.11e+04
zipcode_98008  2.692e+04    796.008     33.815      0.000    2.54e+04
2.85e+04
zipcode_98010  6296.0353    687.432      9.159      0.000    4948.612
7643.458
zipcode_98011  1.464e+04    749.469     19.528      0.000    1.32e+04
1.61e+04
zipcode_98014  8345.1790    707.511     11.795      0.000    6958.399
9731.959
zipcode_98019   1.01e+04    749.286     13.480      0.000    8631.658
1.16e+04
zipcode_98022  -488.1655    784.032     -0.623      0.534   -2024.933
1048.602
zipcode_98023 -3122.2699    925.277     -3.374      0.001   -4935.889
-1308.651
zipcode_98024  9113.2338    668.031     13.642      0.000    7803.838
1.04e+04
zipcode_98027  2.702e+04    872.955     30.948      0.000    2.53e+04
2.87e+04
zipcode_98028  1.589e+04    804.130     19.764      0.000    1.43e+04
1.75e+04
zipcode_98029  2.863e+04    831.986     34.410      0.000     2.7e+04
3.03e+04
zipcode_98030   957.4305    787.668      1.216      0.224    -586.464
2501.325
zipcode_98031  1713.0311    799.226      2.143      0.032     146.482
3279.580
zipcode_98032  -808.2470    702.176     -1.151      0.250   -2184.569
568.075
zipcode_98033  4.307e+04    855.947     50.313      0.000    4.14e+04
4.47e+04
zipcode_98034   2.97e+04    939.500     31.616      0.000    2.79e+04
3.15e+04
zipcode_98038  7641.1315    973.665      7.848      0.000    5732.669
9549.594
zipcode_98039  1.119e+04    612.266     18.283      0.000    9993.915
1.24e+04
zipcode_98040  3.751e+04    731.426     51.281      0.000    3.61e+04
```

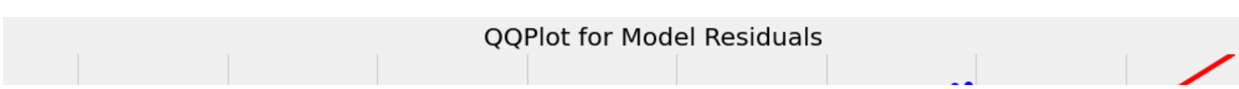
```
3.89e+04
zipcode_98042  2107.9970    950.895      2.217       0.027     244.165
3971.829
zipcode_98045  1.104e+04    768.653     14.361       0.000    9532.144
1.25e+04
zipcode_98052  4.291e+04    960.587     44.669       0.000      4.1e+04
4.48e+04
zipcode_98053  3.366e+04    870.616     38.660       0.000      3.2e+04
3.54e+04
zipcode_98055  4696.7560    795.843      5.902       0.000    3136.838
6256.674
zipcode_98056   1.42e+04    873.622     16.250       0.000    1.25e+04
1.59e+04
zipcode_98058  5896.5118    901.307      6.542       0.000    4129.876
7663.148
zipcode_98059  1.658e+04    902.178     18.377       0.000    1.48e+04
1.83e+04
zipcode_98065  1.698e+04    823.189     20.623       0.000    1.54e+04
1.86e+04
zipcode_98070  8657.0686    714.779     12.112       0.000    7256.042
1.01e+04
zipcode_98072  2.095e+04    796.298     26.312       0.000    1.94e+04
2.25e+04
zipcode_98074  3.227e+04    896.053     36.015       0.000    3.05e+04
3.4e+04
zipcode_98075  3.158e+04    845.621     37.348       0.000    2.99e+04
3.32e+04
zipcode_98077  1.814e+04    754.577     24.044       0.000    1.67e+04
1.96e+04
zipcode_98092 -1648.5178    846.717     -1.947       0.052   -3308.153
11.118
zipcode_98102  2.598e+04    688.167     37.746       0.000    2.46e+04
2.73e+04
zipcode_98103  5.091e+04   1008.220     50.497       0.000    4.89e+04
5.29e+04
zipcode_98105   3.36e+04    752.098     44.673       0.000    3.21e+04
3.51e+04
zipcode_98106  1.441e+04    841.267     17.130       0.000    1.28e+04
1.61e+04
zipcode_98107  3.433e+04    812.915     42.235       0.000    3.27e+04
3.59e+04
zipcode_98108  1.025e+04    748.115     13.701       0.000    8783.879
1.17e+04
zipcode_98109   2.59e+04    685.421     37.793       0.000    2.46e+04
2.72e+04
zipcode_98112  3.772e+04    751.381     50.198       0.000    3.62e+04
3.92e+04
zipcode_98115  5.053e+04    979.186     51.603       0.000    4.86e+04
5.24e+04
zipcode_98116  3.475e+04    842.892     41.224       0.000    3.31e+04
```

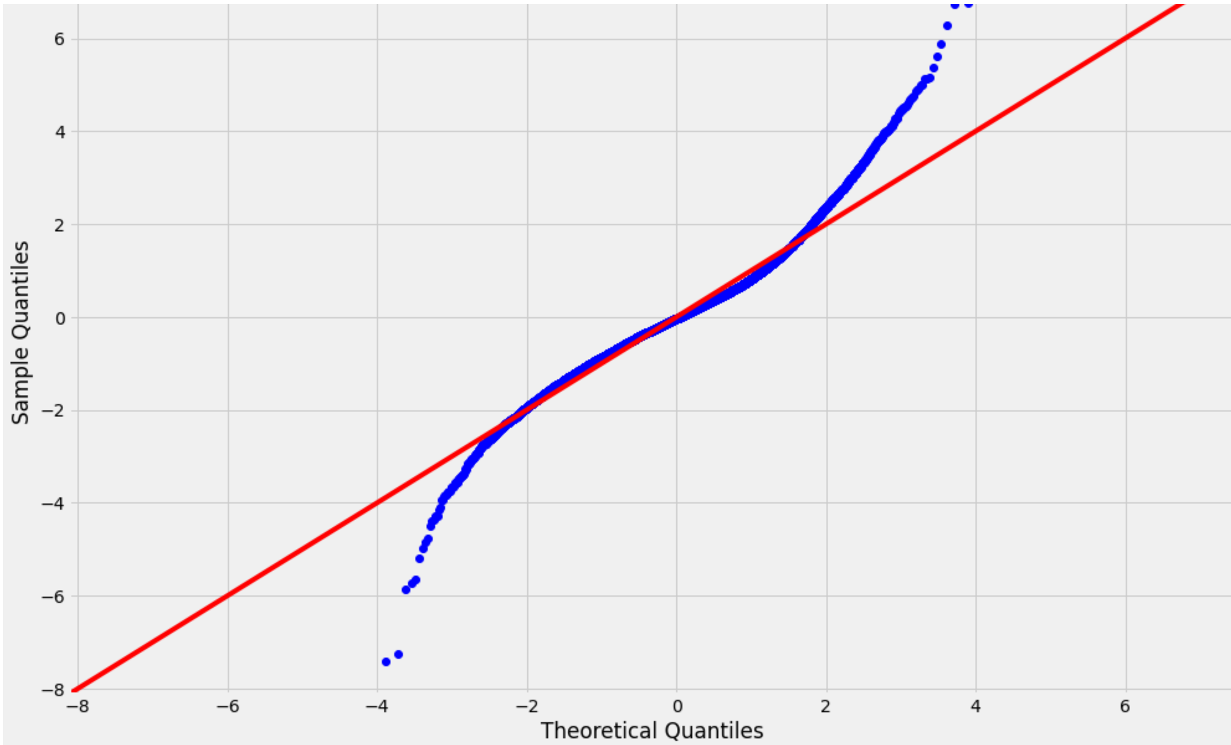
```
3.64e+04
zipcode_98117  4.805e+04    973.565     49.356      0.000    4.61e+04
5e+04
zipcode_98118  2.438e+04    937.527     26.007      0.000    2.25e+04
2.62e+04
zipcode_98119  3.333e+04    740.350     45.026      0.000    3.19e+04
3.48e+04
zipcode_98122  3.359e+04    820.541     40.936      0.000     3.2e+04
3.52e+04
zipcode_98125  2.615e+04    876.070     29.845      0.000    2.44e+04
2.79e+04
zipcode_98126  2.346e+04    860.936     27.248      0.000    2.18e+04
2.51e+04
zipcode_98133  2.219e+04    928.283     23.901      0.000    2.04e+04
2.4e+04
zipcode_98136  2.744e+04    802.051     34.212      0.000    2.59e+04
2.9e+04
zipcode_98144  2.862e+04    842.017     33.995      0.000     2.7e+04
3.03e+04
zipcode_98146  1.224e+04    807.893     15.154      0.000    1.07e+04
1.38e+04
zipcode_98148  2885.9220    653.730      4.415      0.000    1604.557
4167.287
zipcode_98155  1.967e+04    897.237     21.922      0.000    1.79e+04
2.14e+04
zipcode_98166  1.087e+04    785.585     13.837      0.000    9330.031
1.24e+04
zipcode_98168  5188.1019    804.348      6.450      0.000    3611.514
6764.690
zipcode_98177  2.219e+04    778.462     28.509      0.000    2.07e+04
2.37e+04
zipcode_98178  5449.6392    797.868      6.830      0.000    3885.753
7013.526
zipcode_98188  2360.9105    708.941      3.330      0.001     971.327
3750.494
zipcode_98198  2423.5448    805.519      3.009      0.003     844.660
4002.429
zipcode_98199  4.028e+04    814.598     49.453      0.000    3.87e+04
4.19e+04
==================================================================
=========
Omnibus:                     1847.047   Durbin-Watson:
1.981
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
7112.378
Skew:                           0.410   Prob(JB):
0.00
Kurtosis:                       5.797   Cond. No.
14.6
==================================================================
```

```
=========

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


VIF Multicollinearity Test Results
======================================================================
==================================

[('bedrooms', 1.739554438609784),
 ('bathrooms', 3.137363296715625),
 ('sqft_living', 4.006641105187843),
 ('sqft_lot', 1.2073673543921162),
 ('floors', 2.355870732932782),
 ('waterfront', 1.2208787035039756),
 ('view', 1.3233627599024573),
 ('condition', 1.2909040649801051),
 ('grade', 2.9398661058201836),
 ('basement', 1.6030387685422716),
 ('renovated', 1.1113361046871744),
 ('home_age', 3.059148776503065),
 ('zipcode_98002', 1.5277724069993754),
 ('zipcode_98003', 1.7525351741384556),
 ('zipcode_98004', 1.4283435414688896),
 ('zipcode_98005', 1.4263817431618115),
 ('zipcode_98006', 2.106655087324166),
 ('zipcode_98007', 1.356104488378294),
 ('zipcode_98008', 1.7195249523617249),
 ('zipcode_98010', 1.2824274611960729),
 ('zipcode_98011', 1.524336428737748),
 ('zipcode_98014', 1.3584382177033811),
 ('zipcode_98019', 1.5235907454583748),
 ('zipcode_98022', 1.6681742352432292),
 ('zipcode_98023', 2.3233631017247056),
 ('zipcode_98024', 1.2110634979147963),
 ('zipcode_98027', 2.068033966750772),
 ('zipcode_98028', 1.7547934027270997),
 ('zipcode_98029', 1.8784765086196622),
 ('zipcode_98030', 1.6836813580270469),
 ('zipcode_98031', 1.7334551088232675),
 ('zipcode_98032', 1.338028011144665),
 ('zipcode_98033', 1.9882316135862104),
 ('zipcode_98034', 2.395337685108652),
 ('zipcode_98038', 2.572718186900457),
 ('zipcode_98039', 1.0173115687245977),
 ('zipcode_98040', 1.451825940890629),
 ('zipcode_98042', 2.4537958621962037),
 ('zipcode_98045', 1.6033715226234582),
```

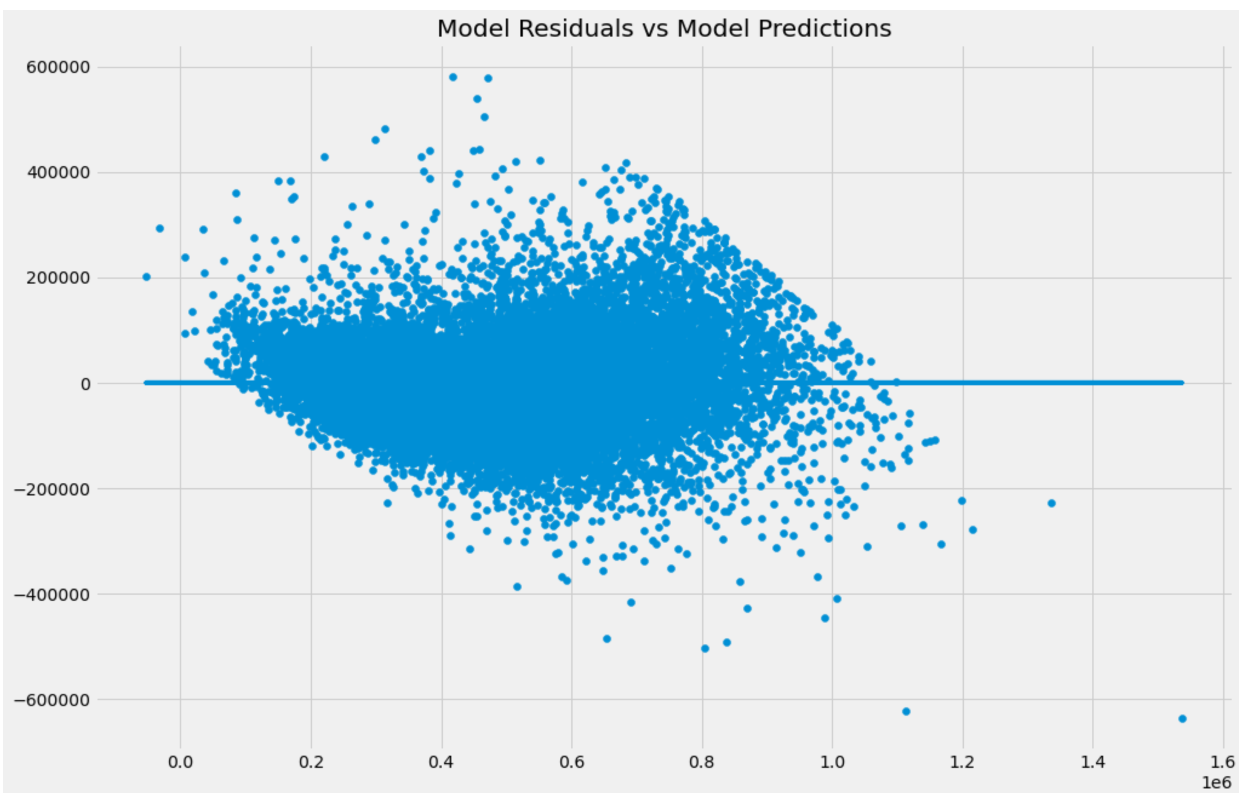
```
 ('zipcode_98052', 2.504070946953088),
 ('zipcode_98053', 2.0569632866156304),
 ('zipcode_98055', 1.7188114679243927),
 ('zipcode_98056', 2.071194350275581),
 ('zipcode_98058', 2.2045458468344026),
 ('zipcode_98059', 2.208809123281588),
 ('zipcode_98065', 1.8389601938515623),
 ('zipcode_98070', 1.3864923033670384),
 ('zipcode_98072', 1.7207765464838705),
 ('zipcode_98074', 2.178919480332135),
 ('zipcode_98075', 1.940549000021154),
 ('zipcode_98077', 1.5451850005736973),
 ('zipcode_98092', 1.9455848821188375),
 ('zipcode_98102', 1.2851705567698435),
 ('zipcode_98103', 2.7585729318207295),
 ('zipcode_98105', 1.535050461549122),
 ('zipcode_98106', 1.920618222530517),
 ('zipcode_98107', 1.7933464988344479),
 ('zipcode_98108', 1.5188321931481918),
 ('zipcode_98109', 1.2749352857021923),
 ('zipcode_98112', 1.5321240872656159),
 ('zipcode_98115', 2.601980014705791),
 ('zipcode_98116', 1.9280476249327134),
 ('zipcode_98117', 2.5721937070473735),
 ('zipcode_98118', 2.385289827236226),
 ('zipcode_98119', 1.4874697175406641),
 ('zipcode_98122', 1.8271484116051262),
 ('zipcode_98125', 2.082815358127845),
 ('zipcode_98126', 2.0114794650430055),
 ('zipcode_98133', 2.338483327019047),
 ('zipcode_98136', 1.7457302959491185),
 ('zipcode_98144', 1.9240457071999935),
 ('zipcode_98146', 1.771257091879799),
 ('zipcode_98148', 1.1597663388926513),
 ('zipcode_98155', 2.184678832728718),
 ('zipcode_98166', 1.6747897851657252),
 ('zipcode_98168', 1.7557451373133284),
 ('zipcode_98177', 1.6445561996524847),
 ('zipcode_98178', 1.7275680421627948),
 ('zipcode_98188', 1.3639357602004007),
 ('zipcode_98198', 1.76086279050894),
 ('zipcode_98199', 1.8007784541995495)]
```

```
Normality Test Results
==============================================================================================
```



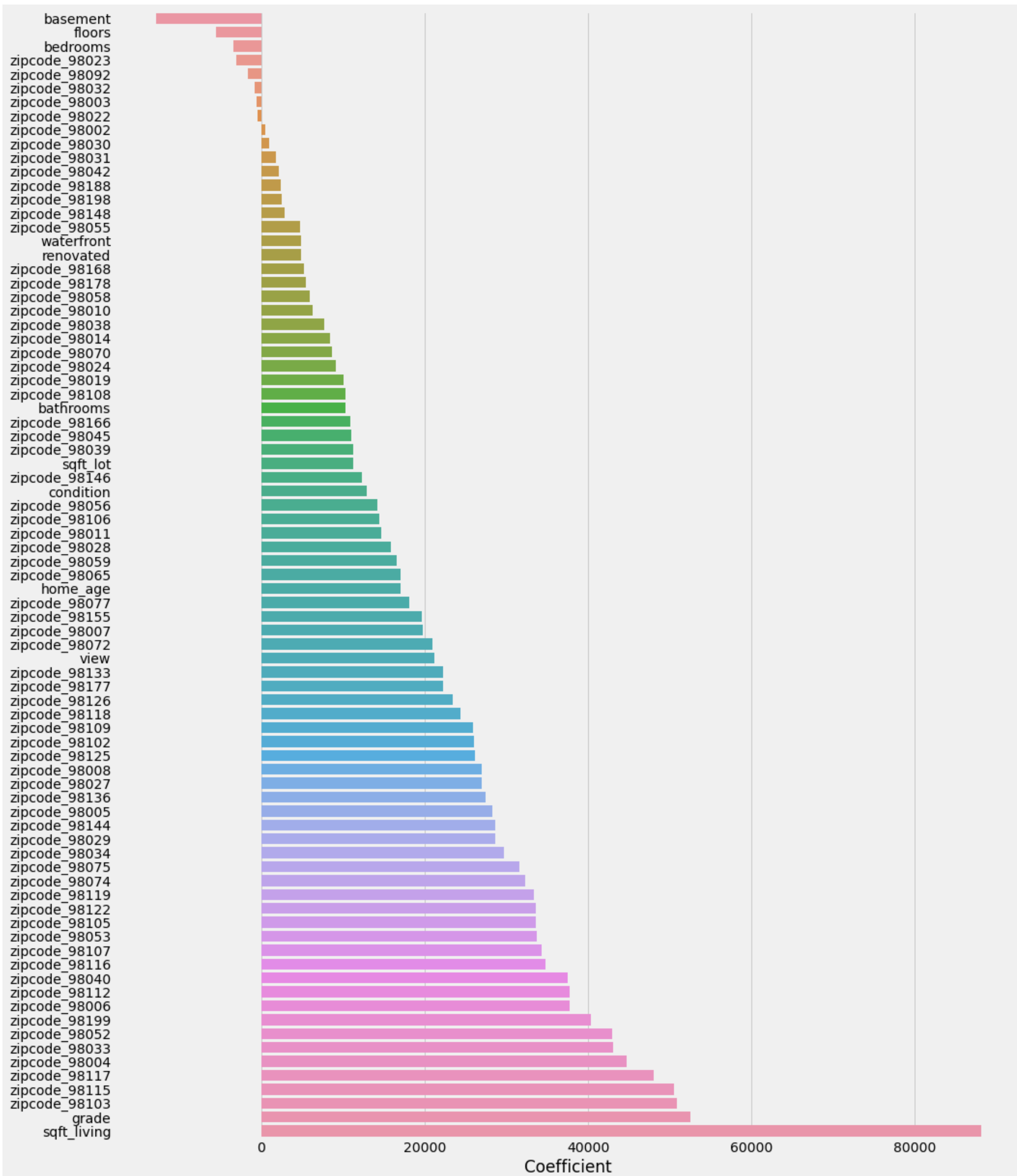


Homoscedasticity Test Results
==



In [105]:
```
#create dataframe of feature coefficients
coefficients_m3_df = pd.DataFrame(model_3.params, columns=['Coeffi
coefficients_m3_df.drop('Intercept', inplace=True)
coefficients_m3_df = coefficients_m3_df.sort_values(by='Coefficier
```

In [106]:
```
#bar plot showing coefficients
fig, ax = plt.subplots(figsize=(15,20))
sns.barplot(data = coefficients_m3_df, y=coefficients_m3_df.index,
```



7.4.2 Model Interpretation

OBSERVATIONS

- Adjusted R-Squared is now 0.827
- All features with a significant p-value except for some zipcodes
- Majority of zipcodes with significant p-values so I will keep them in
- Coefficients of features are smaller in absolute than they were in model 2. I believe this is because of removing outliers in pricing
- The distribution of the residuals is more normal
- The variance in the residuals is more even throughout the prediction of price

ACTIONS

- Going to look through outliers of all columns and remove extreme values

7.4.3 Model Tuning

In [107]:
```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model_3,'sqft_lot', fig=fig)
plt.show()
```

> **OBSERVATIONS**
>
> - The residuals of `sqft_lot` show heteroscedasticity toward the lower side. This seems mostly skewed by the high priced homes which have very small lot sizes that may represent homes closer to the city.

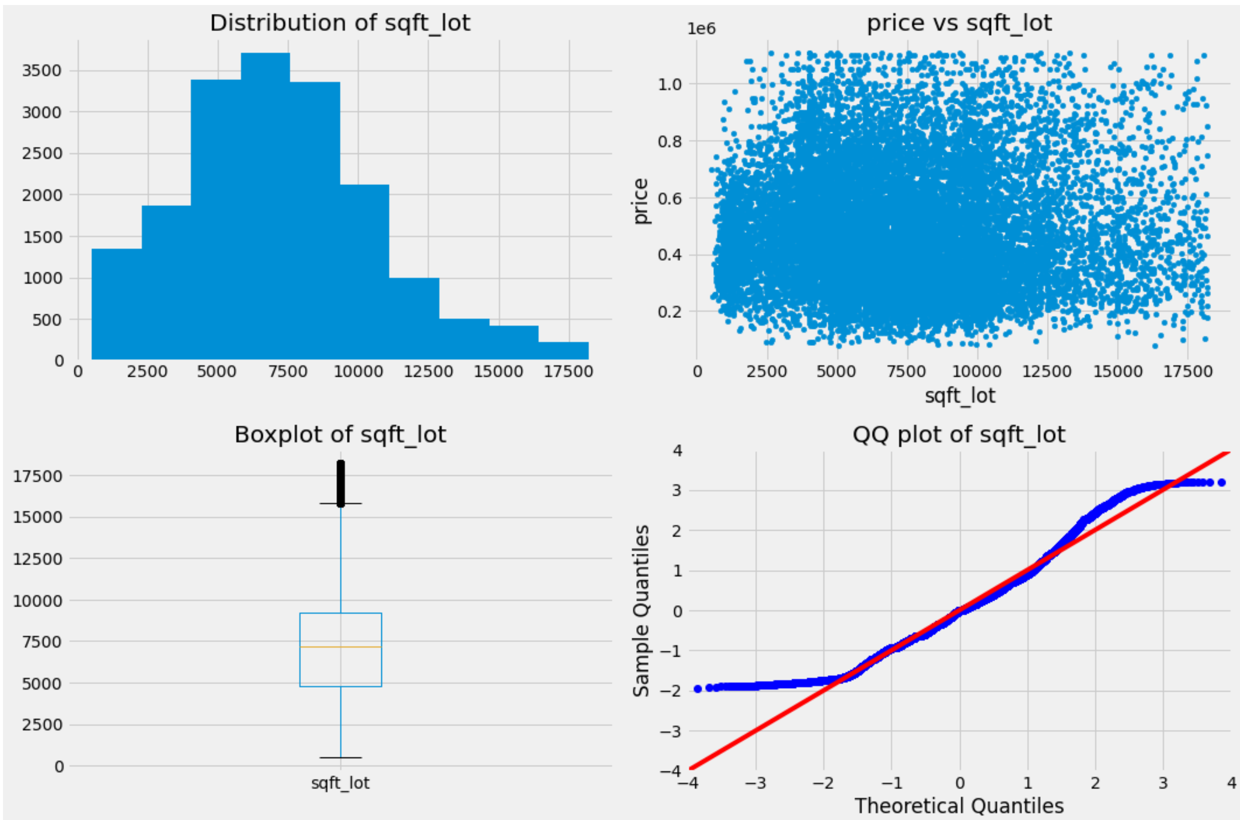
> **ACTIONS**
>
> - Will remove the outliers of sqft_lot first and maybe and see if there is any improvement in the overall model.

7.4.3.1 `sqft_lot` Outlier Removal

I will investigate `sqft_lot` for outliers.

In [108]:
```
get_plots(df_model_base,'sqft_lot',outlier='iqr')
```

```
The number of rows removed is 2189
count               20,098.0
mean     14,555.342024081998
std      40,083.658164637716
min                    520.0
25%                  5,000.0
50%                  7,500.0
75%                 10,289.0
max              1,651,359.0
Name: sqft_lot, dtype: float64
```



OBSERVATIONS

- I will use iqr to remove outliers of `sqft_lot` .

In [109]:

```
#create a copy of model_2 to set up model_3
df_model_4 = df_model_base.copy()
df_model_4
```

Out[109]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditio
0	221,900.0	3	1.0	1180	5650	1.0	0.0	0.0	
1	538,000.0	3	2.25	2570	7242	2.0	0.0	0.0	
2	180,000.0	2	1.0	770	10000	1.0	0.0	0.0	
3	604,000.0	4	3.0	1960	5000	1.0	0.0	0.0	
4	510,000.0	3	2.0	1680	8080	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	.
21592	360,000.0	3	2.5	1530	1131	3.0	0.0	0.0	
21593	400,000.0	4	2.5	2310	5813	2.0	0.0	0.0	
21594	402,101.0	2	0.75	1020	1350	2.0	0.0	0.0	
21595	400,000.0	3	2.5	1600	2388	2.0	0.0	0.0	
21596	325,000.0	2	0.75	1020	1076	2.0	0.0	0.0	

20098 rows × 82 columns

In [110]:

```
#remove outliers based off iqr
df_model_base = outliers(df_model_base, 'sqft_lot', 'iqr')
df_model_4 = df_model_base.copy()
df_model_4
```

```
There were 2189 outliers removed.
```

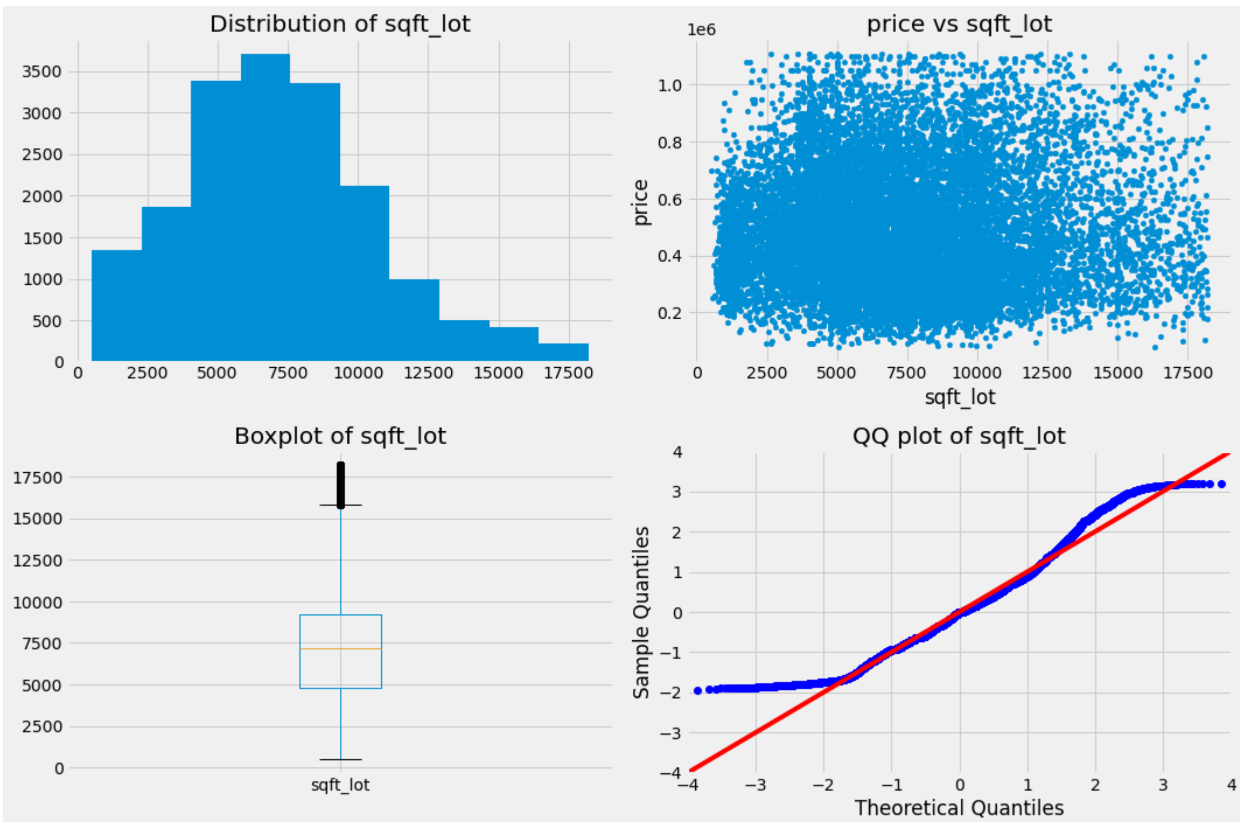
Out[110]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditio
0	221,900.0	3	1.0	1180	5650	1.0	0.0	0.0	[illegible]
1	538,000.0	3	2.25	2570	7242	2.0	0.0	0.0	[illegible]
2	180,000.0	2	1.0	770	10000	1.0	0.0	0.0	[illegible]
3	604,000.0	4	3.0	1960	5000	1.0	0.0	0.0	[illegible]
4	510,000.0	3	2.0	1680	8080	1.0	0.0	0.0	[illegible]
...	...	...	...	...	...	...	...	...	[illegible]
21592	360,000.0	3	2.5	1530	1131	3.0	0.0	0.0	[illegible]
21593	400,000.0	4	2.5	2310	5813	2.0	0.0	0.0	[illegible]
21594	402,101.0	2	0.75	1020	1350	2.0	0.0	0.0	[illegible]
21595	400,000.0	3	2.5	1600	2388	2.0	0.0	0.0	[illegible]
21596	325,000.0	2	0.75	1020	1076	2.0	0.0	0.0	[illegible]

17909 rows × 82 columns

In [111]:
```
#recheck the sqft_lot column
get_plots(df_model_4,'sqft_lot',outlier='none')
```

```
count              17,909.0
mean      7,189.531464626724
std      3,443.9304409405627
min                   520.0
25%                 4,800.0
50%                 7,171.0
75%                 9,200.0
max                18,205.0
Name: sqft_lot, dtype: float64
```



7.5 Model 4

In [112]:

```
#view model_3 dataframe
df_model_4
```

Out[112]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditio
0	221,900.0	3	1.0	1180	5650	1.0	0.0	0.0	
1	538,000.0	3	2.25	2570	7242	2.0	0.0	0.0	
2	180,000.0	2	1.0	770	10000	1.0	0.0	0.0	
3	604,000.0	4	3.0	1960	5000	1.0	0.0	0.0	
4	510,000.0	3	2.0	1680	8080	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	
21592	360,000.0	3	2.5	1530	1131	3.0	0.0	0.0	
21593	400,000.0	4	2.5	2310	5813	2.0	0.0	0.0	
21594	402,101.0	2	0.75	1020	1350	2.0	0.0	0.0	
21595	400,000.0	3	2.5	1600	2388	2.0	0.0	0.0	
21596	325,000.0	2	0.75	1020	1076	2.0	0.0	0.0	

17909 rows × 82 columns

7.5.1 Model Creation

In [113]:

```
#define indpendent and dependent variables
x_cols = df_model_4.drop(columns='price').columns
y_col = 'price'
#run funciton to create model and check assumptions
model_4 = fit_new_model(df_model_4, x_cols=x_cols, y_col=y_col, nc
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	g
0	221,900.0	-0.33	-1.47	-1.0	-0.45	-0.87	-0.06	-0.26	-0.63	
1	538,000.0	-0.33	0.34	0.94	0.02	0.97	-0.06	-0.26	-0.63	
2	180,000.0	-1.56	-1.47	-1.57	0.82	-0.87	-0.06	-0.26	-0.63	
3	604,000.0	0.89	1.42	0.09	-0.64	-0.87	-0.06	-0.26	2.46	
4	510,000.0	-0.33	-0.02	-0.3	0.26	-0.87	-0.06	-0.26	-0.63	

5 rows × 82 columns

```
                            OLS Regression Results
======================================================================
=========
Dep. Variable:                  price   R-squared:
0.836
Model:                            OLS   Adj. R-squared:
0.835
Method:                 Least Squares   F-statistic:
1120.
Date:                Thu, 29 Apr 2021   Prob (F-statistic):
0.00
Time:                        20:11:52   Log-Likelihood:            -2
.2813e+05
No. Observations:               17909   AIC:
4.564e+05
Df Residuals:                   17827   BIC:
4.571e+05
Df Model:                          81
Covariance Type:            nonrobust
======================================================================
============
                    coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
------------
Intercept      4.649e+05    617.095    753.397      0.000    4.64e+05
4.66e+05
bedrooms      -3257.0485    829.527     -3.926      0.000   -4883.002
-1631.095
bathrooms      9947.3764   1095.740      9.078      0.000    7799.619
1.21e+04
sqft_living    8.538e+04   1233.691     69.210      0.000      8.3e+04
8.78e+04
sqft_lot       5500.5717    894.159      6.152      0.000    3747.933
7253.211
floors        -3959.1623   1010.370     -3.919      0.000   -5939.585
-1978.740
waterfront     5018.3550    668.794      7.504      0.000    3707.454
6329.256
view            2.11e+04    701.421     30.080      0.000    1.97e+04
2.25e+04
condition      1.319e+04    704.643     18.716      0.000    1.18e+04
1.46e+04
grade          4.832e+04   1050.718     45.990      0.000    4.63e+04
5.04e+04
basement      -1.197e+04    791.238    -15.129      0.000   -1.35e+04
-1.04e+04
renovated      4710.3618    651.060      7.235      0.000    3434.221
5986.503
home_age       1.728e+04   1120.748     15.419      0.000    1.51e+04
```

```
1.95e+04
zipcode_98002  1063.9664    789.644      1.347      0.178    -483.813
2611.746
zipcode_98003  -332.2358    844.203     -0.394      0.694   -1986.956
1322.484
zipcode_98004  4.577e+04    749.993     61.032      0.000    4.43e+04
4.72e+04
zipcode_98005  2.588e+04    723.040     35.799      0.000    2.45e+04
2.73e+04
zipcode_98006  3.839e+04    920.255     41.712      0.000    3.66e+04
4.02e+04
zipcode_98007  2.068e+04    736.301     28.083      0.000    1.92e+04
2.21e+04
zipcode_98008  2.796e+04    838.820     33.330      0.000    2.63e+04
2.96e+04
zipcode_98010  4921.8042    663.799      7.415      0.000    3620.693
6222.916
zipcode_98011   1.49e+04    778.494     19.138      0.000    1.34e+04
1.64e+04
zipcode_98014  5700.6462    665.457      8.567      0.000    4396.286
7005.006
zipcode_98019  9040.2825    750.552     12.045      0.000    7569.127
1.05e+04
zipcode_98022   294.8333    753.030      0.392      0.695   -1181.179
1770.846
zipcode_98023 -2611.4871    979.115     -2.667      0.008   -4530.648
-692.326
zipcode_98024  5098.5061    644.062      7.916      0.000    3836.083
6360.930
zipcode_98027  2.803e+04    841.341     33.322      0.000    2.64e+04
2.97e+04
zipcode_98028  1.582e+04    836.182     18.918      0.000    1.42e+04
1.75e+04
zipcode_98029  3.028e+04    883.086     34.290      0.000    2.86e+04
3.2e+04
zipcode_98030  1089.4458    825.879      1.319      0.187    -529.357
2708.249
zipcode_98031  1882.8535    834.587      2.256      0.024     246.982
3518.725
zipcode_98032  -326.9733    726.384     -0.450      0.653   -1750.756
1096.809
zipcode_98033    4.4e+04    903.819     48.686      0.000    4.22e+04
4.58e+04
zipcode_98034  3.098e+04   1001.692     30.929      0.000     2.9e+04
3.29e+04
zipcode_98038  7317.3214    999.375      7.322      0.000    5358.450
9276.193
zipcode_98039  1.191e+04    623.585     19.106      0.000    1.07e+04
1.31e+04
zipcode_98040  3.854e+04    758.765     50.798      0.000    3.71e+04
```

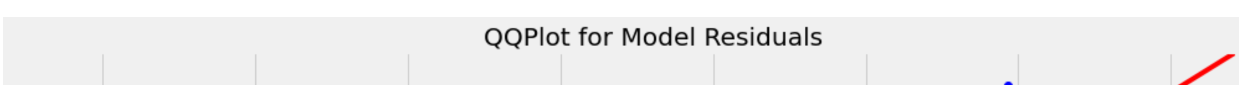
```
4e+04
zipcode_98042  2258.5598    963.771      2.343      0.019    369.476
4147.644
zipcode_98045  8701.4127    748.534     11.625      0.000   7234.214
1.02e+04
zipcode_98052  4.359e+04   1009.231     43.195      0.000   4.16e+04
4.56e+04
zipcode_98053  3.051e+04    835.549     36.515      0.000   2.89e+04
3.21e+04
zipcode_98055  5061.3919    835.571      6.057      0.000   3423.591
6699.193
zipcode_98056  1.467e+04    923.991     15.876      0.000   1.29e+04
1.65e+04
zipcode_98058  5480.0522    925.650      5.920      0.000   3665.688
7294.417
zipcode_98059  1.603e+04    931.420     17.213      0.000   1.42e+04
1.79e+04
zipcode_98065  1.782e+04    864.372     20.612      0.000   1.61e+04
1.95e+04
zipcode_98070  2578.5263    654.318      3.941      0.000   1295.999
3861.053
zipcode_98072    1.4e+04    746.603     18.754      0.000   1.25e+04
1.55e+04
zipcode_98074  3.184e+04    929.252     34.261      0.000      3e+04
3.37e+04
zipcode_98075   2.88e+04    848.948     33.921      0.000   2.71e+04
3.05e+04
zipcode_98077  9212.6592    662.567     13.904      0.000   7913.963
1.05e+04
zipcode_98092 -1937.4091    838.654     -2.310      0.021  -3581.253
-293.566
zipcode_98102  2.816e+04    722.468     38.976      0.000   2.67e+04
2.96e+04
zipcode_98103  5.547e+04   1116.266     49.695      0.000   5.33e+04
5.77e+04
zipcode_98105  3.638e+04    803.529     45.273      0.000   3.48e+04
3.8e+04
zipcode_98106  1.617e+04    903.303     17.905      0.000   1.44e+04
1.79e+04
zipcode_98107  3.749e+04    878.100     42.692      0.000   3.58e+04
3.92e+04
zipcode_98108  1.155e+04    794.567     14.537      0.000   9993.541
1.31e+04
zipcode_98109  2.806e+04    720.582     38.943      0.000   2.66e+04
2.95e+04
zipcode_98112  4.075e+04    803.507     50.709      0.000   3.92e+04
4.23e+04
zipcode_98115  5.455e+04   1074.751     50.752      0.000   5.24e+04
5.67e+04
zipcode_98116  3.758e+04    912.592     41.174      0.000   3.58e+04
```

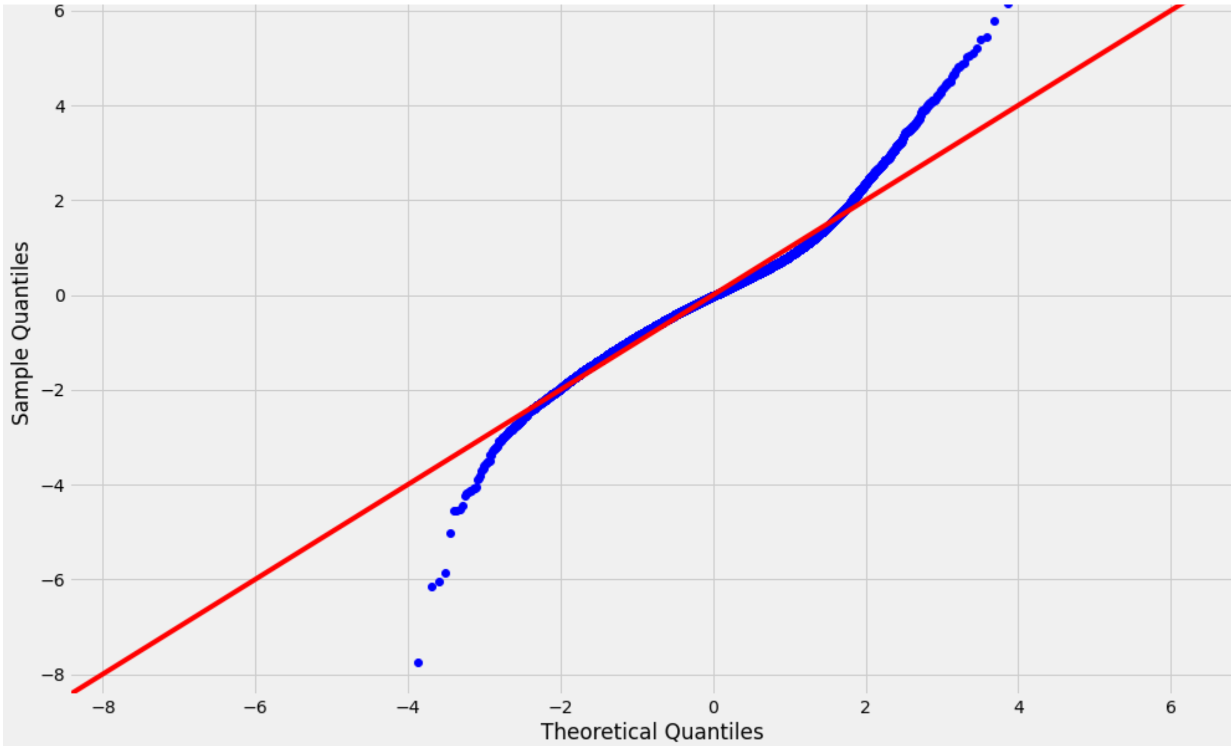
```
3.94e+04
zipcode_98117  5.221e+04   1072.930     48.661      0.000    5.01e+04
5.43e+04
zipcode_98118  2.688e+04   1023.138     26.277      0.000    2.49e+04
2.89e+04
zipcode_98119  3.611e+04    789.966     45.707      0.000    3.46e+04
3.77e+04
zipcode_98122  3.672e+04    889.322     41.286      0.000     3.5e+04
3.85e+04
zipcode_98125  2.785e+04    938.035     29.687      0.000     2.6e+04
2.97e+04
zipcode_98126  2.584e+04    933.564     27.680      0.000     2.4e+04
2.77e+04
zipcode_98133  2.425e+04   1001.368     24.217      0.000    2.23e+04
2.62e+04
zipcode_98136  2.977e+04    857.815     34.709      0.000    2.81e+04
3.15e+04
zipcode_98144  3.135e+04    916.164     34.219      0.000    2.96e+04
3.31e+04
zipcode_98146   1.32e+04    852.623     15.482      0.000    1.15e+04
1.49e+04
zipcode_98148  3262.8027    673.349      4.846      0.000    1942.973
4582.633
zipcode_98155  2.067e+04    947.680     21.808      0.000    1.88e+04
2.25e+04
zipcode_98166  9906.9342    802.891     12.339      0.000    8333.190
1.15e+04
zipcode_98168  5775.4612    829.351      6.964      0.000    4149.853
7401.069
zipcode_98177  2.184e+04    806.579     27.079      0.000    2.03e+04
2.34e+04
zipcode_98178  6046.4468    847.171      7.137      0.000    4385.909
7706.985
zipcode_98188  2465.4287    735.161      3.354      0.001    1024.442
3906.416
zipcode_98198  2740.8658    839.201      3.266      0.001    1095.950
4385.781
zipcode_98199  4.337e+04    878.372     49.370      0.000    4.16e+04
4.51e+04
==================================================================
=========
Omnibus:                     1439.001   Durbin-Watson:
1.992
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
5085.914
Skew:                           0.369   Prob(JB):
0.00
Kurtosis:                       5.504   Cond. No.
15.3
==================================================================
```

```
=========

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


VIF Multicollinearity Test Results
======================================================================
=================================

[('bedrooms', 1.8068953378552643),
 ('bathrooms', 3.1527318640987088),
 ('sqft_living', 3.9965470512155634),
 ('sqft_lot', 2.0994310916524705),
 ('floors', 2.6806034223047535),
 ('waterfront', 1.1745090113267127),
 ('view', 1.2919023235931795),
 ('condition', 1.303797006636501),
 ('grade', 2.8989717538829654),
 ('basement', 1.6439400717400996),
 ('renovated', 1.11304838801856),
 ('home_age', 3.298284452364991),
 ('zipcode_98002', 1.6373254833965631),
 ('zipcode_98003', 1.8713961166776192),
 ('zipcode_98004', 1.4770186480460288),
 ('zipcode_98005', 1.372764833688333),
 ('zipcode_98006', 2.223762564455269),
 ('zipcode_98007', 1.4235814624625447),
 ('zipcode_98008', 1.847605496867797),
 ('zipcode_98010', 1.1570330319531175),
 ('zipcode_98011', 1.5914127295860976),
 ('zipcode_98014', 1.162818426518677),
 ('zipcode_98019', 1.4792237474479761),
 ('zipcode_98022', 1.489006914609613),
 ('zipcode_98023', 2.5173260775654094),
 ('zipcode_98024', 1.089248384048354),
 ('zipcode_98027', 1.8587302083219783),
 ('zipcode_98028', 1.836002872568174),
 ('zipcode_98029', 2.047756868779405),
 ('zipcode_98030', 1.7910376591835975),
 ('zipcode_98031', 1.8290057092907592),
 ('zipcode_98032', 1.3854921105054403),
 ('zipcode_98033', 2.1450394617742097),
 ('zipcode_98034', 2.634757935964246),
 ('zipcode_98038', 2.622580334870421),
 ('zipcode_98039', 1.0210891544605647),
 ('zipcode_98040', 1.51177461328527),
 ('zipcode_98042', 2.4390417142457363),
 ('zipcode_98045', 1.4712769416207014),
```

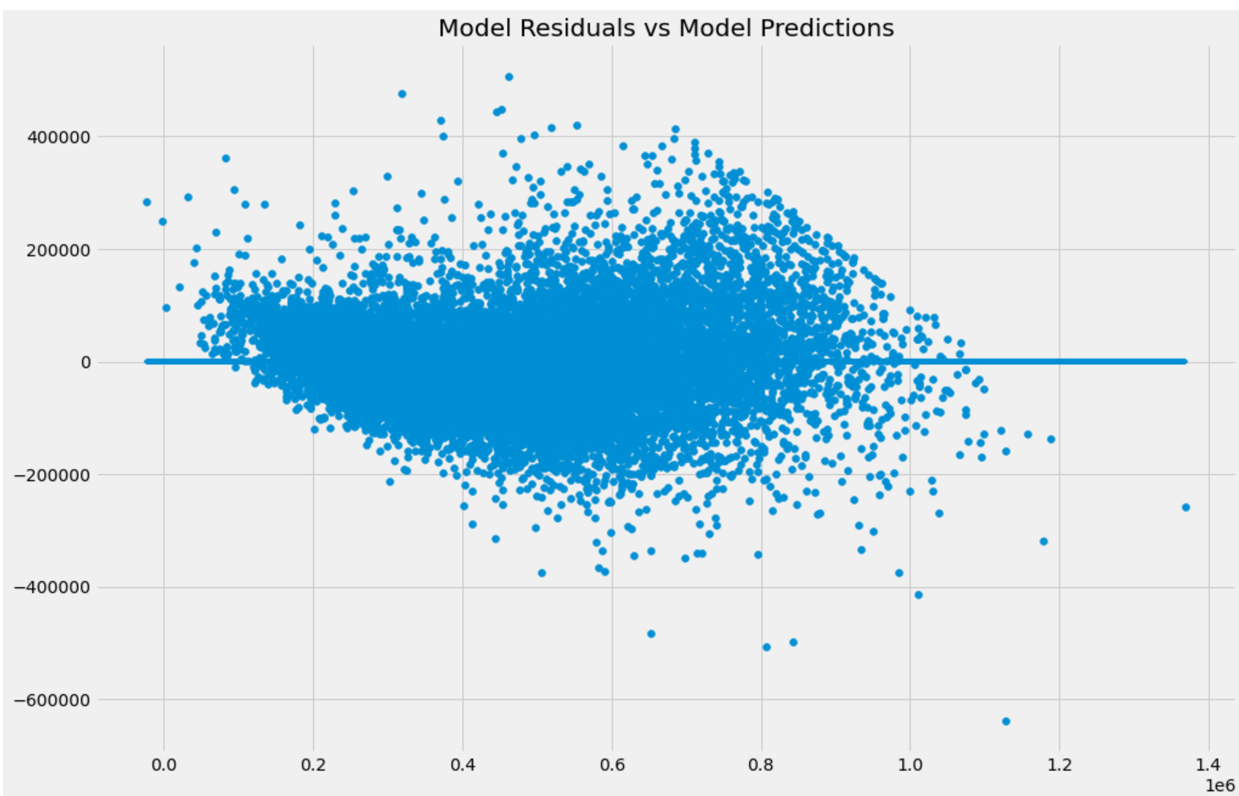
```
 ('zipcode_98052', 2.674567277935902),
 ('zipcode_98053', 1.833226675246918),
 ('zipcode_98055', 1.8333234239710965),
 ('zipcode_98056', 2.241852558567192),
 ('zipcode_98058', 2.249913964601556),
 ('zipcode_98059', 2.2780467793885153),
 ('zipcode_98065', 1.9618855775436468),
 ('zipcode_98070', 1.1242160426191574),
 ('zipcode_98072', 1.4636979700362966),
 ('zipcode_98074', 2.267454335188060 7),
 ('zipcode_98075', 1.8924929053004058),
 ('zipcode_98077', 1.1527421031409744),
 ('zipcode_98092', 1.8468759973994446),
 ('zipcode_98102', 1.3705935090352146),
 ('zipcode_98103', 3.271953378024299),
 ('zipcode_98105', 1.6954112730265174),
 ('zipcode_98106', 2.142590289848206),
 ('zipcode_98107', 2.024695303653884),
 ('zipcode_98108', 1.6578041383509365),
 ('zipcode_98109', 1.3634500314003661),
 ('zipcode_98112', 1.6953194305186934),
 ('zipcode_98115', 3.033105323032986),
 ('zipcode_98116', 2.1868822693808614),
 ('zipcode_98117', 3.022836102449557),
 ('zipcode_98118', 2.748780076090921),
 ('zipcode_98119', 1.6386611476887067),
 ('zipcode_98122', 2.0767775305505536),
 ('zipcode_98125', 2.3105206304820074),
 ('zipcode_98126', 2.2885481385771573),
 ('zipcode_98133', 2.6330496734718594),
 ('zipcode_98136', 1.932232055468873),
 ('zipcode_98144', 2.204035759657258),
 ('zipcode_98146', 1.9089124342559436),
 ('zipcode_98148', 1.1905640891017533),
 ('zipcode_98155', 2.3582785522923553),
 ('zipcode_98166', 1.6927193142677572),
 ('zipcode_98168', 1.8061273965283486),
 ('zipcode_98177', 1.7083072914652957),
 ('zipcode_98178', 1.8845792866402145),
 ('zipcode_98188', 1.4191782667855328),
 ('zipcode_98198', 1.8492854616301688),
 ('zipcode_98199', 2.025952142566707)]
```

```
Normality Test Results
================================================================================================
```






```
Homoscedasticity Test Results
==================================================================================================
```



In [114]:

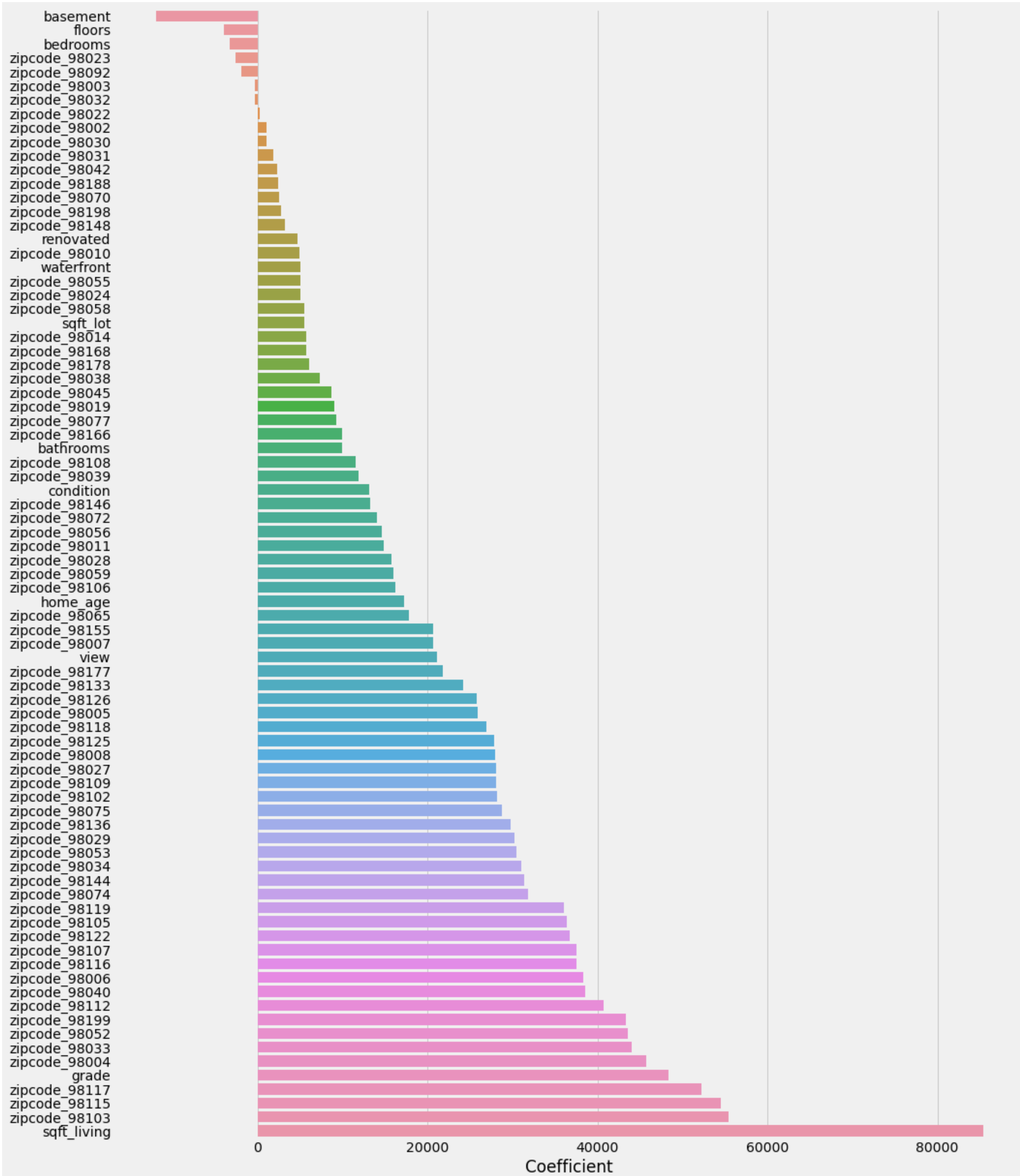
```
#create dataframe of feature coefficients
coefficients_m4_df = pd.DataFrame(model_4.params, columns=['Coeffi
coefficients_m4_df.drop('Intercept', inplace=True)
coefficients_m4_df = coefficients_m4_df.sort_values(by='Coefficien
```

In [115]:

```
#create csv to export for data viz
# coefficients_df.to_csv(r'Model Coefficients (scaled) for Tableau
```

In [116]:

```
#bar plot showing coefficients
fig, ax = plt.subplots(figsize=(15,20))
sns.barplot(data = coefficients_m4_df, y=coefficients_m4_df.index,
```



7.5.2 Model Interpretation

OBSERVATIONS

- R^2 is 0.835
- The normality and homoscedasticity of the residuals are acceptable and therefore there will not be another iteration of the model.
- All features except for some zipcodes are statistically significant.
- Most negatively correlated features with price are `basement`, `floors` and `bedrooms`
- Most positively correlated features with price are `sqft_living`, `grade` and `view`

8 Interpret

The final model was created after 4 total iterations. Each iteration highlighted issues within the model that affected the accuracy of the model or its significance. Before the first model, I had evaluated linearity of features and multicollinearity between features. These were dealt with and remedied prior to running the first model. I also initially had zipcode not being OHE but this was difficult for the model to deal with because each zipcode has a lot of variation in how it affects home price. OHE zipcode jumped the R^2 significantly, however, the residuals of the model still showed room for improvement. This was primarily because of outliers in price and sqft_lot which were remedied in model iteration 3 and 4. Model 4 (final model) showed a R^2 of .835 with significant features and almost normal and homoscedastic residuals. The final model highlighted a few insights:

- Square Footage is the feature which best predicts home price (highest normalized coefficient).
- Zipcodes vary widely in their influence on home price
- The grade of construction is a highly influential feature for home price but it depends on whether or not you have high or low construction quality.
- The view is of the home is an extremely important feature when predicting price but is mostly an uncontrollable feature for a home owner.
- Bathrooms are also very important to the overall home price
- It is important to stay away from adding bedrooms or floors

9 Recommendations and Conclusions

Based on what the model showed were significantly impactful features, I recommend the following actions for any renovator looking to make smart decisions that will add value to their home:

- Add a full size bathroom (~60 square feet) with above average construction quality to improve home value
- If the house has an unfinished basement of more than 350 square feet, finish the basement to get the extra square feet. This will offset the fact that the model views having a basement negatively affects the home value when considered by itself. However, if a home has an unfinished basement around the median size of the area, 700 square feet, then it will end up being a large value increase to the value of the home. Again, utilizing above average construction quality will add additional value.

10 Appendix

10.1 Dataset for Tableau

In [3555]:
```
#view the dataframe
df_scrub
```

Out[3555]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
0	2014-10-13	221,900.0	3	1.0	1180	5650	1.0	0.0	0.0
1	2014-12-09	538,000.0	3	2.25	2570	7242	2.0	0.0	0.0
3	2014-12-09	604,000.0	4	3.0	1960	5000	1.0	0.0	0.0
4	2015-02-18	510,000.0	3	2.0	1680	8080	1.0	0.0	0.0
5	2014-05-12	1,230,000.0	4	4.5	5420	101930	1.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...
21592	2014-05-21	360,000.0	3	2.5	1530	1131	3.0	0.0	0.0
21593	2015-02-23	400,000.0	4	2.5	2310	5813	2.0	0.0	0.0
21594	2014-06-23	402,101.0	2	0.75	1020	1350	2.0	0.0	0.0
21595	2015-01-16	400,000.0	3	2.5	1600	2388	2.0	0.0	0.0
21596	2014-10-15	325,000.0	2	0.75	1020	1076	2.0	0.0	0.0

17704 rows × 20 columns

In [1003]:
```
#remove outliers
df_scrub = df_scrub.loc[df_scrub['bedrooms'] <= 5]
```

In [1004]:
```
#remove outliers based off iqr
df_scrub = outliers(df_scrub, 'price', 'iqr')
```

```
There were 897 outliers removed.
```

In [1005]:
```
#remove outliers based off iqr
df_scrub = outliers(df_scrub, 'sqft_lot', 'iqr')
```

```
There were 1814 outliers removed.
```

In [1007]:
```
#write csv file for tableau
# df_scrub.to_csv(r'Scrubbed Housing Data for Tableau.csv',index=F
```