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:
- 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.

- **INTRODUCTION**
- **OBTAIN**
- **SCRUB**
- **EXPLORE**
- **MODEL**
- **INTERPRET**
- **RECOMMENDATIONS/CONCLUSIONS**

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 [344]:

```
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_inflatic
```

In [345]:

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

4.2 Global Functions

In [346]:

```
#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 [347]:

```
#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.head())
        print('\n')
    else:
        #display the df
        display(df.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 [348]:

```
#function to delete outliers using either iqr or std
def outliers(df, 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[col].mean()
        col_std = df[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[col] > lower_thresh_std) & (df[col] <
        print(f'There were {len(df) - len(df_new)} outliers remove
    elif outlier == 'iqr':
        #create outlier variables
        q25 = df[col].quantile(0.25)
        q75 = df[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[col] > lower_thresh_iqr) & (df[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 [349]:
```
#import the dataset from local csv
df_original = pd.read_csv('Data/kc_house_data.csv')
df_original
```

Out[349]:

aterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	
nan	0.0	3	7	1180	0.0	1955	0.0	98178	
0.0	0.0	3	7	2170	400.0	1951	1,991.0	98125	47.
0.0	0.0	3	6	770	0.0	1933	nan	98028	
0.0	0.0	5	7	1050	910.0	1965	0.0	98136	
0.0	0.0	3	8	1680	0.0	1987	0.0	98074	
...	...	...	...	...	...	...	...	...	
0.0	0.0	3	8	1530	0.0	2009	0.0	98103	
0.0	0.0	3	8	2310	0.0	2014	0.0	98146	
0.0	0.0	3	7	1020	0.0	2009	0.0	98144	
nan	0.0	3	8	1600	0.0	2004	0.0	98027	
0.0	0.0	3	7	1020	0.0	2008	0.0	98144	

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 [350]:
```
#column names
df_original.columns
```

Out[350]:
```
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 [351]:

```
#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 [352]:

```
#check for null values
df_original.isna().sum()/len(df_original)*100
```

Out[352]:

```
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             0.2917071815529934 7
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 [353]:
```
#check numeric data
df_original.describe()
```

Out[353]:

	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	367,368.1401013936	0.926298894542015	0.7689842966527002	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 [354]:
```
#create a copy of the original dataframe
df_scrub = df_original.copy()
df_scrub
```

Out[354]:

	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 [355]:

```
#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
         ...
243.0        1
946.0        1
508.0        1
935.0        1
417.0        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 [356]:

```
#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))
```

```
0.0       13280
600.0       217
500.0       209
700.0       208
800.0       201
          ...
792.0         1
2850.0        1
2350.0        1
1481.0        1
652.0         1
Name: sqft_basement, Length: 303, dtype: int64

21597
```

In [357]:

```
#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[357]:

```
True     21427
False      170
dtype: int64
```

In [358]:

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

Out[358]:

	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 [359]:
```
#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 [360]:
```
#check previous id where sqft_basement was a '?'
df_scrub.loc[df_scrub['id'] == 2525310310]
```

Out[360]:

	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 [361]:
```
#check the rows have been dropped
df_scrub.loc[df_scrub['sqft_basement'] == '?']['sqft_basement'].cc
```

Out[361]: 0

In [362]:
```
#check to ensure all descrepencies are gone
df_scrub.loc[(df_scrub['sqft_above'] + df_scrub['sqft_basement'] =
```

Out[362]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditio

In [363]:
```
#what is the median basement sqft
df_scrub.loc[df_scrub['sqft_basement'] > 0,'sqft_basement'].mediar
```

Out[363]:
```
700.0
```

ACTIONS

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

In [364]:
```
#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[364]:
```
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 [365]:
```
#how much more sqft does a house have when they have a basement?
df_scrub.groupby(by='basement')['sqft_living'].median()
```

Out[365]:
```
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 [366]:

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

Out[366]:

```
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 [367]:

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

Out[367]:

```
0
```

ACTIONS

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

In [368]:

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

Out[368]:

	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 [369]:
```
#explore data values
df_scrub['yr_built'].value_counts()
```

Out[369]:
```
0       3842
2014     457
2006     380
2005     378
2004     364
        ... 
1901      23
1933      22
1902      21
1934      16
1935      16
Name: yr_built, Length: 117, dtype: int64
```

OBSERVATIONS

- There are 3842 homes which have a `yr_built` of 0

ACTIONS

- I will investigate this further to see how to proceed

In [370]:

```
#view value_counts of all columns when yr_built is 0
df_scrub.loc[df_scrub['yr_built'] == 0].count()
```

Out[370]:

```
id               3842
date             3842
price            3842
bedrooms         3842
bathrooms        3842
sqft_living      3842
sqft_lot         3842
floors           3842
waterfront       3842
view             3842
condition        3842
grade            3842
sqft_above       3842
sqft_basement    3842
yr_built         3842
yr_renovated     3842
zipcode          3842
lat              3842
long             3842
sqft_living15    3842
sqft_lot15       3842
basement         3842
renovated        3842
dtype: int64
```

OBSERVATIONS

- All columns are 0 for these 3,842 homes.

ACTIONS

- I will drop these rows

In [371]:
```
#drop rows with 0's for values and check
df_scrub.drop(df_scrub.loc[df_scrub['yr_built'] == 0].index, inpla
df_scrub.loc[df_scrub['yr_built'] == 0].count()
```

Out[371]:
```
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
dtype: int64
```

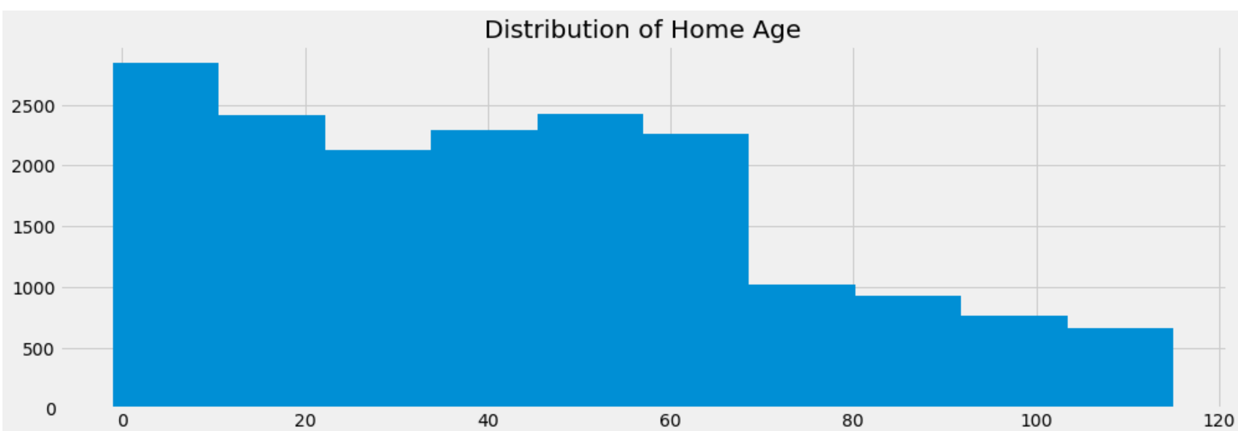
In [372]:
```
#create yr_sold column first
df_scrub['date'] = pd.to_datetime(df_scrub['date'])
df_scrub['yr_sold'] = df_scrub['date'].dt.year
df_scrub[['date','yr_built','yr_sold']]
```

Out[372]:

	date	yr_built	yr_sold
0	2014-10-13	1955	2014
1	2014-12-09	1951	2014
3	2014-12-09	1965	2014
4	2015-02-18	1987	2015
5	2014-05-12	2001	2014
...	...	...	...
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

17755 rows × 3 columns

In [373]:
```
#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 [374]:

```
#check data types
df_scrub.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17755 entries, 0 to 21596
Data columns (total 25 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             17755 non-null  int64
 1   date           17755 non-null  datetime64[ns]
 2   price          17755 non-null  float64
 3   bedrooms       17755 non-null  int64
 4   bathrooms      17755 non-null  float64
 5   sqft_living    17755 non-null  int64
 6   sqft_lot       17755 non-null  int64
 7   floors         17755 non-null  float64
 8   waterfront     15809 non-null  float64
 9   view           17704 non-null  float64
 10  condition      17755 non-null  int64
 11  grade          17755 non-null  int64
 12  sqft_above     17755 non-null  int64
 13  sqft_basement  17755 non-null  int64
 14  yr_built       17755 non-null  int64
 15  yr_renovated   17755 non-null  float64
 16  zipcode        17755 non-null  int64
 17  lat            17755 non-null  float64
 18  long           17755 non-null  float64
 19  sqft_living15  17755 non-null  int64
 20  sqft_lot15     17755 non-null  int64
 21  basement       17755 non-null  int64
 22  renovated      17755 non-null  int64
 23  yr_sold        17755 non-null  int64
 24  home_age       17755 non-null  int64
dtypes: datetime64[ns](1), float64(8), int64(16)
memory usage: 3.5 MB
```

OBSERVATIONS

- All data types seem good for the model

5.3 Null Values

In [375]:
```
#check for null values
df_scrub.isna().sum()
```

Out[375]:
```
id                 0
date               0
price              0
bedrooms           0
bathrooms          0
sqft_living        0
sqft_lot           0
floors             0
waterfront      1946
view              51
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 [376]:
```
#view values in waterfront column
df_scrub['waterfront'].value_counts(dropna=False)
```

Out[376]:
```
0.0    15688
nan     1946
1.0      121
Name: waterfront, dtype: int64
```

OBSERVATIONS:

- `waterfront` has 11% null values.

ACTION

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

In [377]:
```
#correlation of waterfront
df_scrub.corr()['waterfront']
```

Out[377]:
```
id              -0.0007271151406019631
price             0.28061779508670853
bedrooms        -0.0037972470805466155
bathrooms         0.06895849489521352
sqft_living       0.11490850577128048
sqft_lot          0.02592756366361523
floors           0.018991071659567576
waterfront                        1.0
view               0.4097734225678934
condition        0.017035059923196618
grade             0.08520732172037616
sqft_above        0.07953782583182091
sqft_basement     0.08954329499890055
yr_built        -0.023441718330979473
yr_renovated       0.0872436795030493
zipcode          0.029702414769883254
lat             -0.015771099373141872
long             -0.04205519654265491
sqft_living15     0.09252912301185473
sqft_lot15        0.02969156480538231
basement         0.042247678022296786
renovated         0.08763626037688341
yr_sold         -0.007725185091591844
home_age         0.023317154750412086
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 [378]:

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

Out[378]:

```
view
0.0      0.0
1.0      1.0
2.0      6.0
3.0     10.0
4.0    103.0
Name: waterfront, dtype: float64
```

OBSERVATIONS

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

In [379]:
```
#there are 19 null values in waterfront with a view of 4
df_scrub.loc[(df_scrub['waterfront'].isna()) & (df_scrub['view'] =
```

Out[379]:

	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	
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	
4422	7781600100	2014-09-05	1,340,000.0	3	2.75	2730	38869	1.5	

In [380]:
```
#there are 54 null values in waterfront with a view of 3
df_scrub.loc[(df_scrub['waterfront'].isna()) & (df_scrub['view'] =
```

Out[380]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterf
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	
707	4022907770	2014-10-14	550,000.0	4	1.75	2480	14782	1.0	
830	2061100570	2015-02-10	595,000.0	3	1.75	1910	5753	1.0	

ACTIONS

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

In [381]:
```
#fill in null where view is 4
df_scrub.loc[(df_scrub['waterfront'].isna()) & (df_scrub['view'] =
```

In [382]:
```
#fill in null where view is 3
df_scrub.loc[(df_scrub['waterfront'].isna()) & (df_scrub['view'] =
```

In [383]:
```
#check the changes
df_scrub.loc[(df_scrub['waterfront'].isna()) & (df_scrub['view'] =
```

Out[383]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditio

In [384]:
```
#check the changes
df_scrub.loc[(df_scrub['waterfront'].isna()) & (df_scrub['view'] =
```

Out[384]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditio

In [385]:
```
#view values in waterfront column
df_scrub['waterfront'].value_counts(dropna=False)/len(df_scrub)
```

Out[385]:
```
0.0     0.8835820895522388
nan     0.10537876654463531
1.0    0.011039143903125881
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 [386]:
```
#convert waterfront null values to 0
df_scrub.loc[df_scrub['waterfront'].isna(),['waterfront']] = 0
```

In [387]:
```
#check waterfront values
df_scrub['waterfront'].value_counts(dropna=False)
```

Out[387]:
```
0.0    17559
1.0      196
Name: waterfront, dtype: int64
```

In [388]:
```
#print number of rows in dataframe
print(f'The dataframe now has {len(df_scrub)} many rows.')
```

```
The dataframe now has 17755 many rows.
```

5.3.2 view Column

In [389]:
```
#view the values of the view column
df_scrub['view'].value_counts(dropna=False)
```

Out[389]:
```
0.0    15972
2.0      792
3.0      403
1.0      277
4.0      260
nan       51
Name: view, dtype: int64
```

ACTIONS

- I will drop the 39 null values

In [390]:
```
#drop rows
df_scrub.dropna(subset=['view'], inplace=True)
df_scrub['view'].isna().sum()
```

Out[390]:
```
0
```

In [391]:
```
#check the view column
df_scrub['view'].value_counts(dropna=False)
```

Out[391]:
```
0.0    15972
2.0      792
3.0      403
1.0      277
4.0      260
Name: view, dtype: int64
```

In [392]:
```
#recheck null values in dataframe
df_scrub.isna().sum()
```

Out[392]:
```
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 [393]:

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

Out[393]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfr
2494	1000102	2014-09-16	280,000.0	6	3.0	2400	9373	2.0	
2495	1000102	2015-04-22	300,000.0	6	3.0	2400	9373	2.0	
11421	109200390	2014-08-20	245,000.0	3	1.75	1480	3900	1.0	
11422	109200390	2014-10-20	250,000.0	3	1.75	1480	3900	1.0	
7786	251300110	2015-01-14	358,000.0	3	2.25	2510	12013	2.0	

In [394]:

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

Out[394]:

	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 [395]:
```
#look at columns
df_scrub.columns
```

Out[395]:
```
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 [396]:
```
#drop the sqft_basement column
df_scrub.drop(columns='sqft_basement', inplace=True)
df_scrub.columns
```

Out[396]:
```
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 [397]:
```
#drop columns
df_scrub.drop(columns=['sqft_living15','sqft_lot15'], inplace=True
df_scrub.columns
```

Out[397]:
```
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 [398]:
```
#drop the yr_renovated column
df_scrub.drop(columns='yr_renovated', inplace=True)
```

5.5.4 id Column

In [399]:
```
#drop the id column
df_scrub.drop(columns='id', inplace=True)
```

5.6 State of Dataframe

In [400]:

```
#state of the dataframe
df_scrub
```

Out[400]:

	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

6 Explore

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

In [401]:
```
#create a copy of the scrub dataframe
df_explore = df_scrub.copy()
df_explore.head()
```

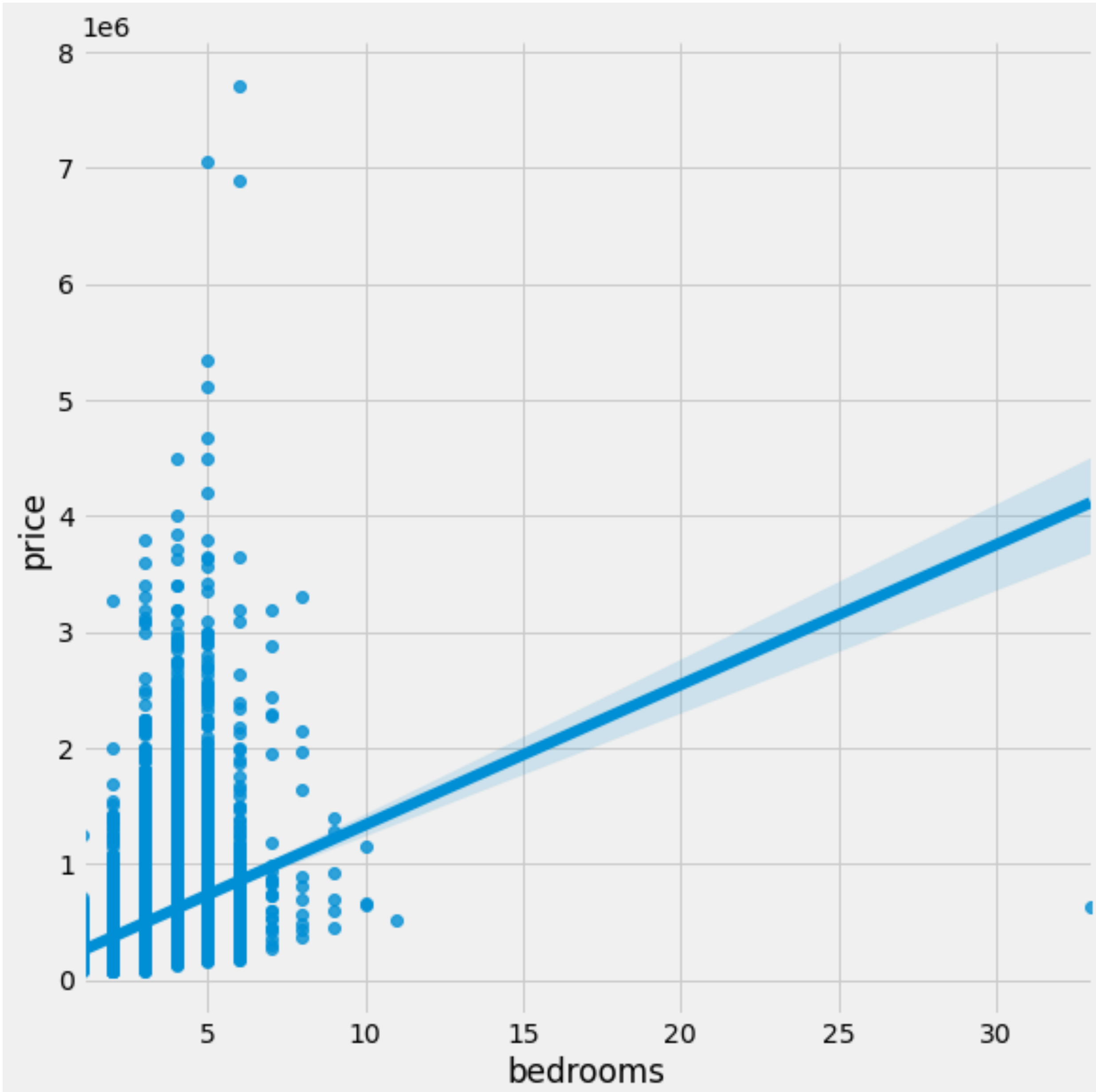
Out[401]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	con
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	

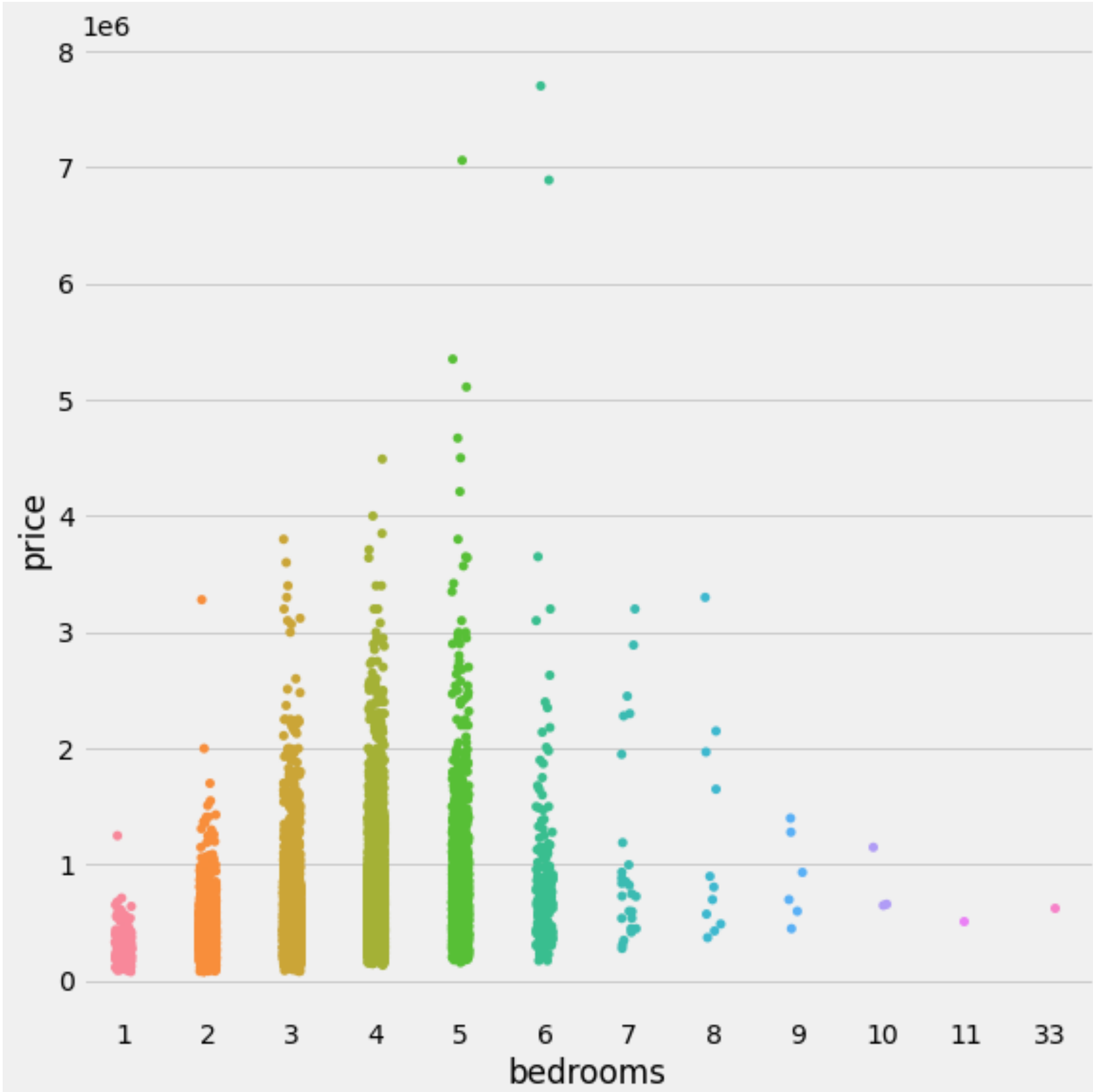
6.1 Linearity

6.1.1 bedrooms

In [402]:
```
#check linearity between bedrooms and price
sns.lmplot(data=df_explore, x='bedrooms', y='price', height=8);
```



In [403]:
```
#view price vs bedrooms another way
sns.catplot(data=df_explore, x='bedrooms', y='price', height=8);
```

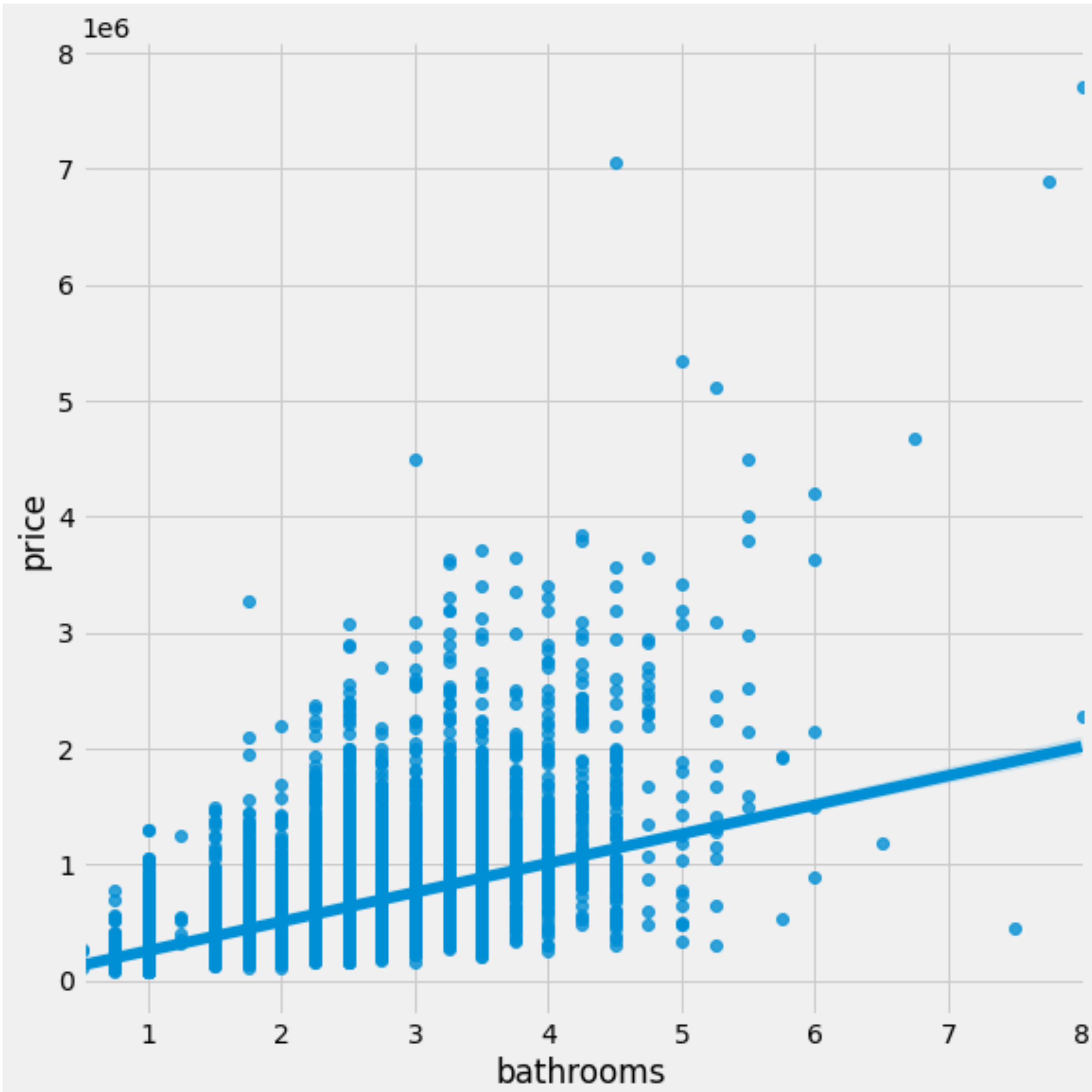


OBSERVATIONS

- 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 [404]:
```
#check linearity between bathrooms and price
sns.lmplot(data=df_explore, x='bathrooms', y='price', height=8);
```

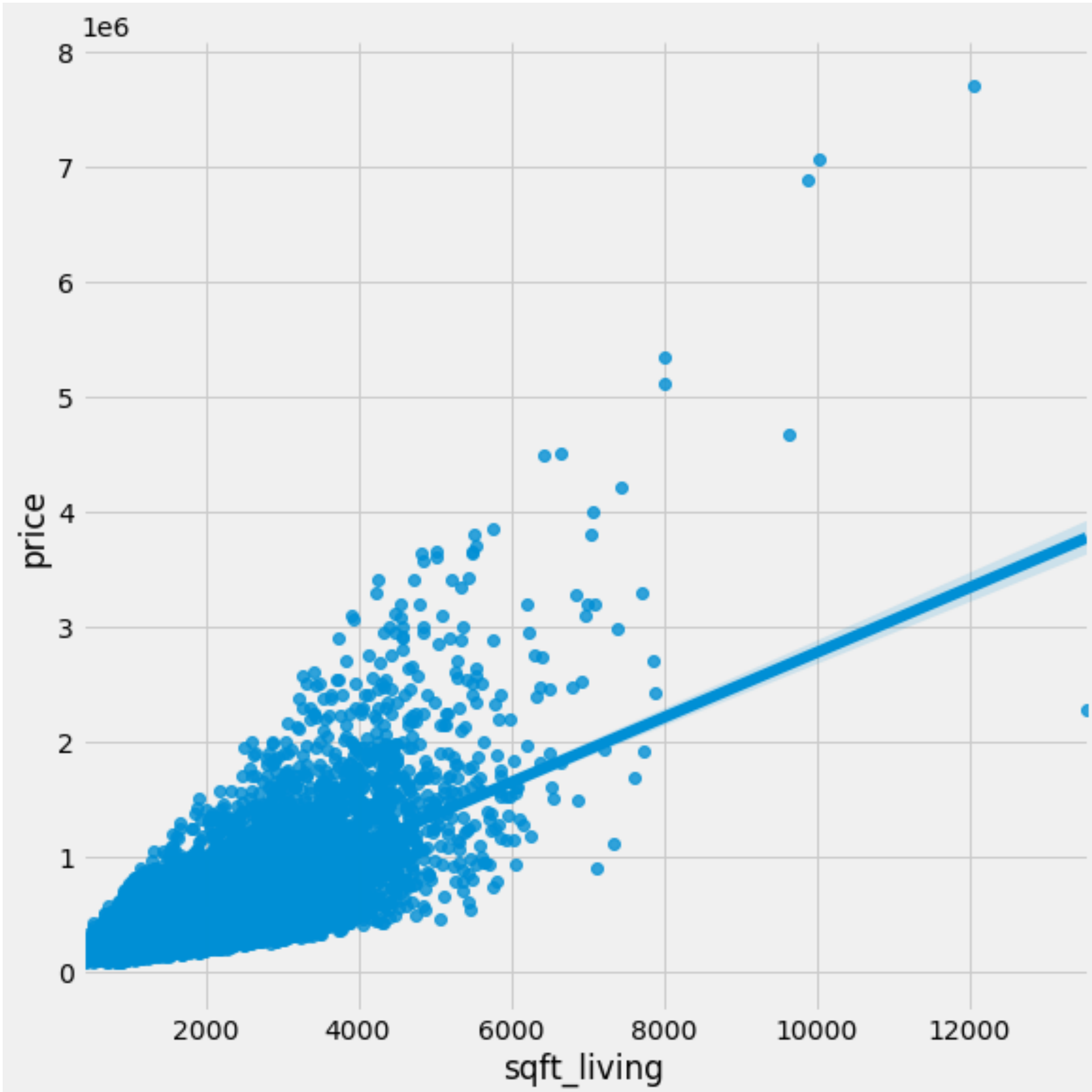


OBSERVATIONS

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

6.1.3 `sqft_living`

In [405]:
```
#check linearity between sqft_living and price
sns.lmplot(data=df_explore, x='sqft_living', y='price', height=8);
```



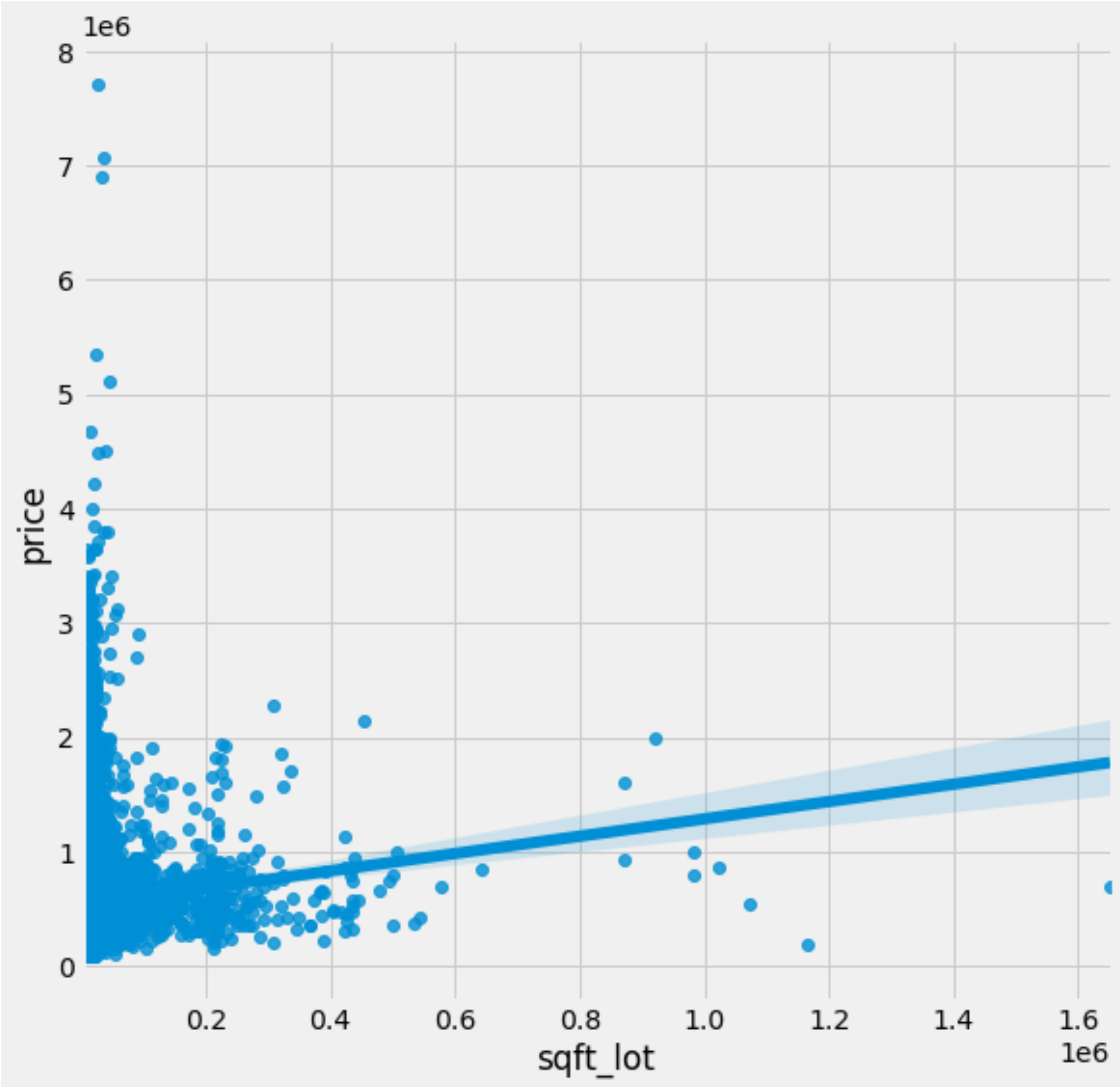
OBSERVATIONS

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

6.1.4 sqft_lot

In [406]:
```
#check linearity between sqft_lot and price
sns.lmplot(data=df_explore, x='sqft_lot', y='price', height=8)
```

Out[406]: <seaborn.axisgrid.FacetGrid at 0x7f92e0a6c3d0>

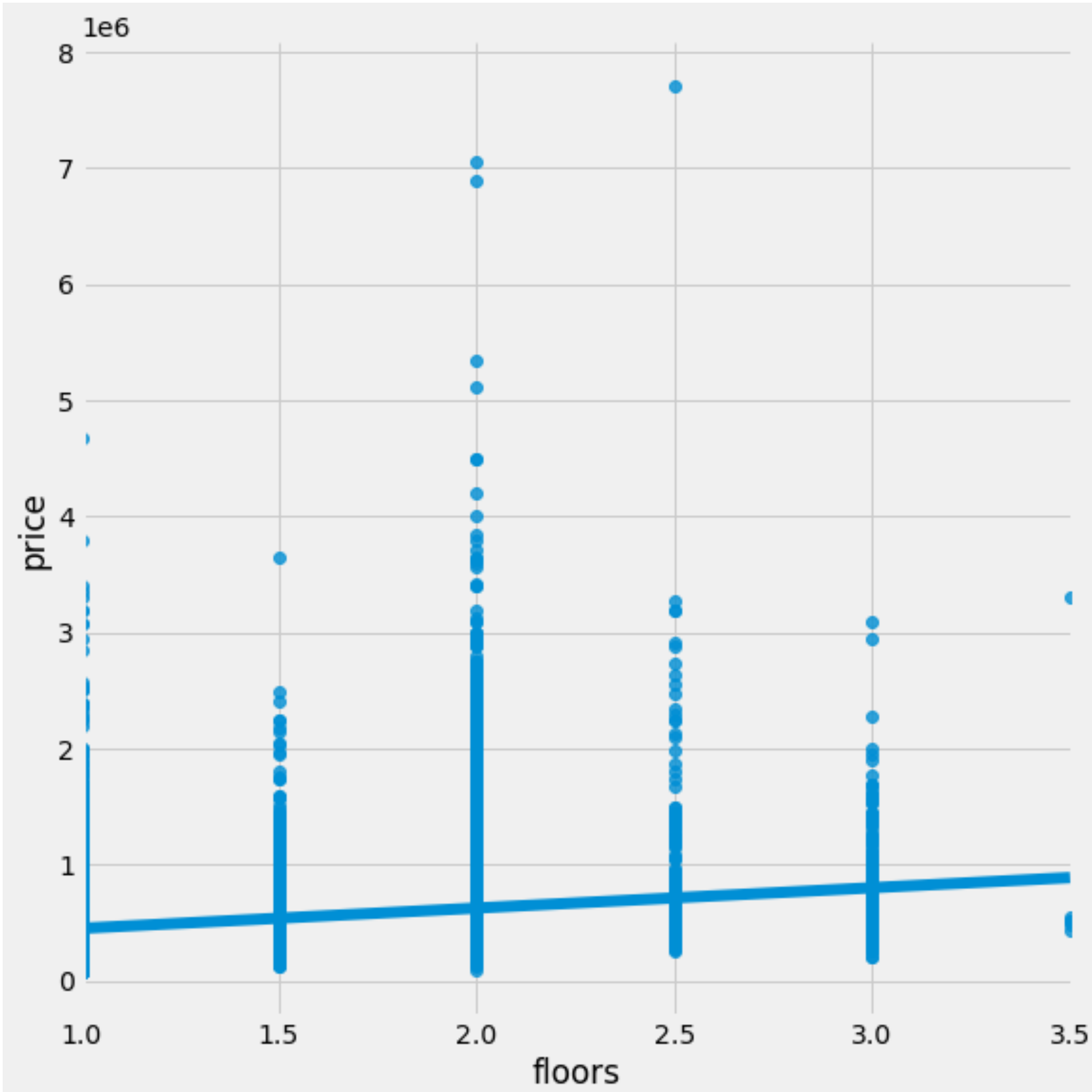


OBSERVATIONS

- 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 [407]:
```
#check linearity between floors and price
sns.lmplot(data=df_explore, x='floors', y='price', height=8);
```

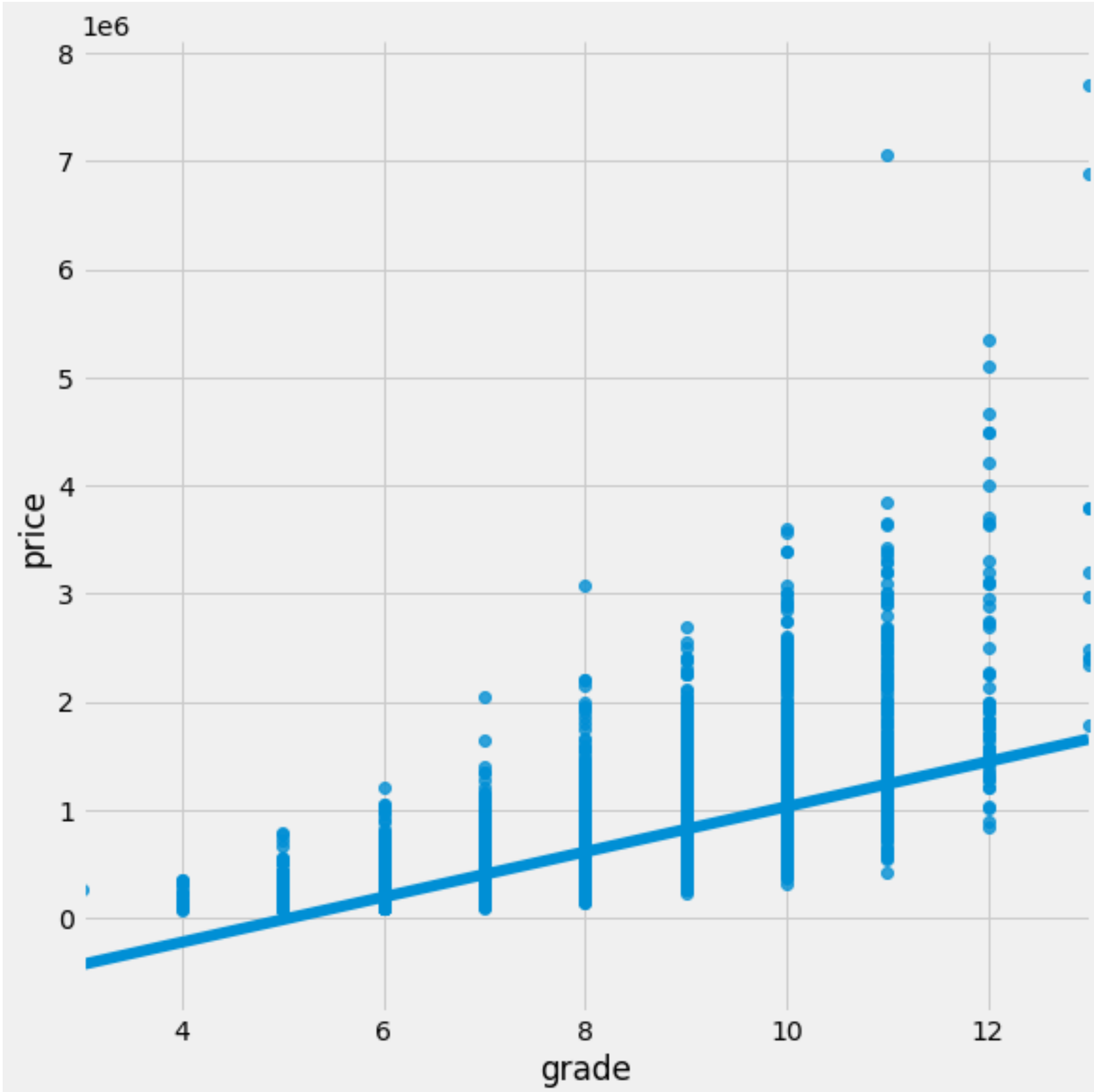


OBSERVATIONS

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

6.1.6 `grade`

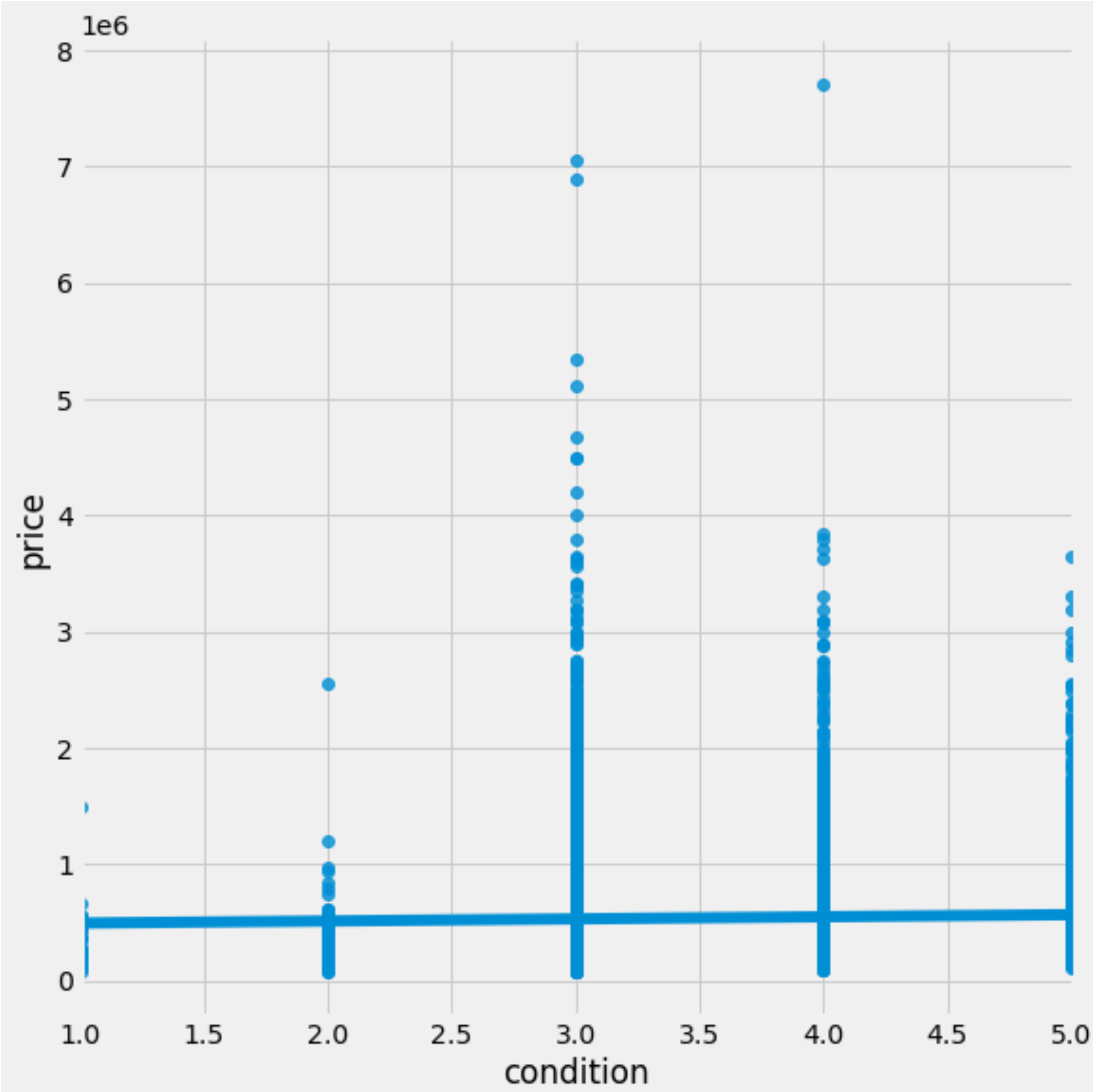
In [408]:
```
#check linearity between grade and price
sns.lmplot(data=df_explore, x='grade', y='price', height=8);
```



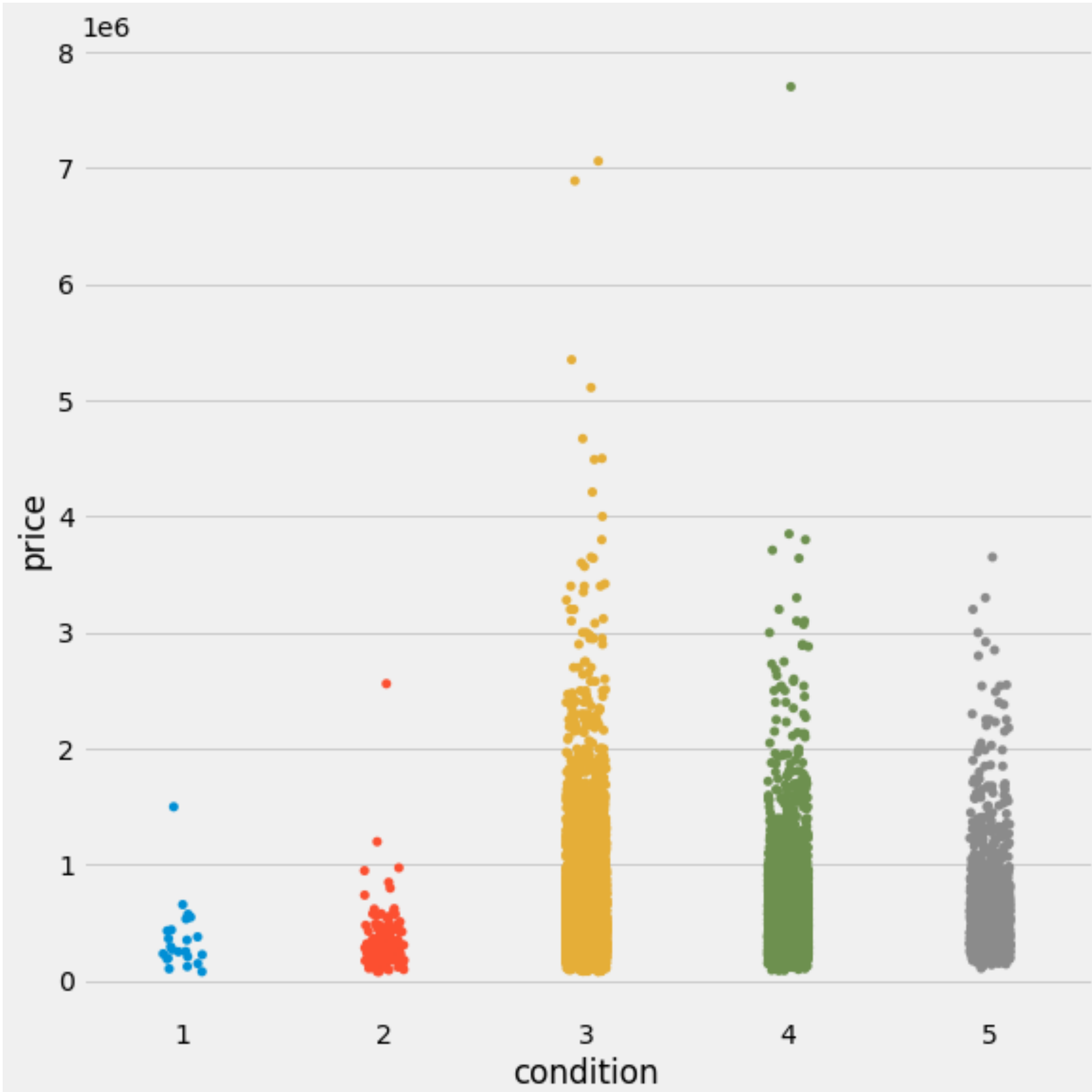
> **OBSERVATIONS**
>
> - There seems to be a linear relationship between `grade` and `price`.

6.1.7 `condition`

In [409]:
```
#check linearity between condition and price
sns.lmplot(data=df_explore, x='condition', y='price', height=8);
```



In [410]:
```
#check relationship another way
sns.catplot(data=df_explore, x='condition', y='price', height=8);
```

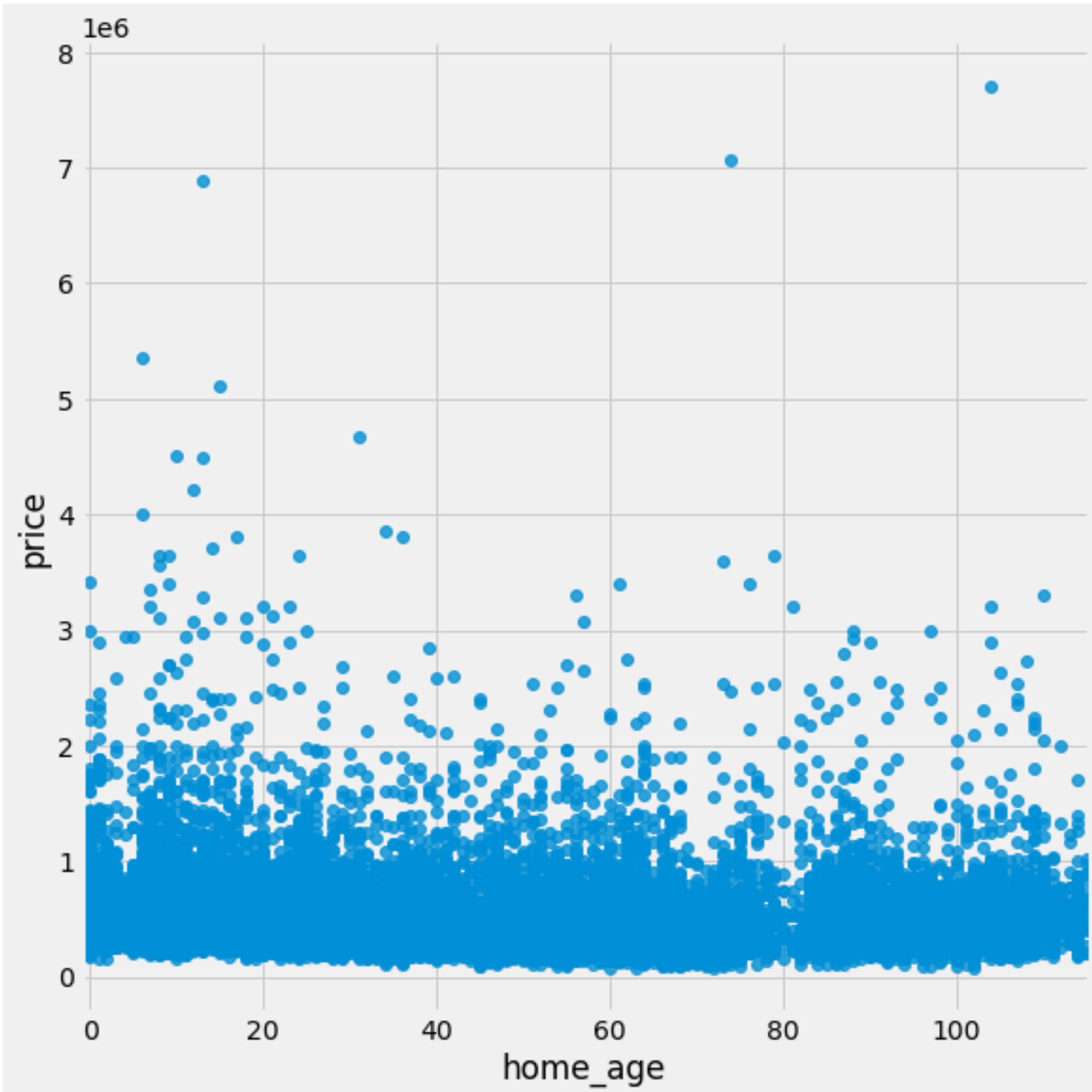


OBSERVATIONS

- 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 [411]:
```
#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

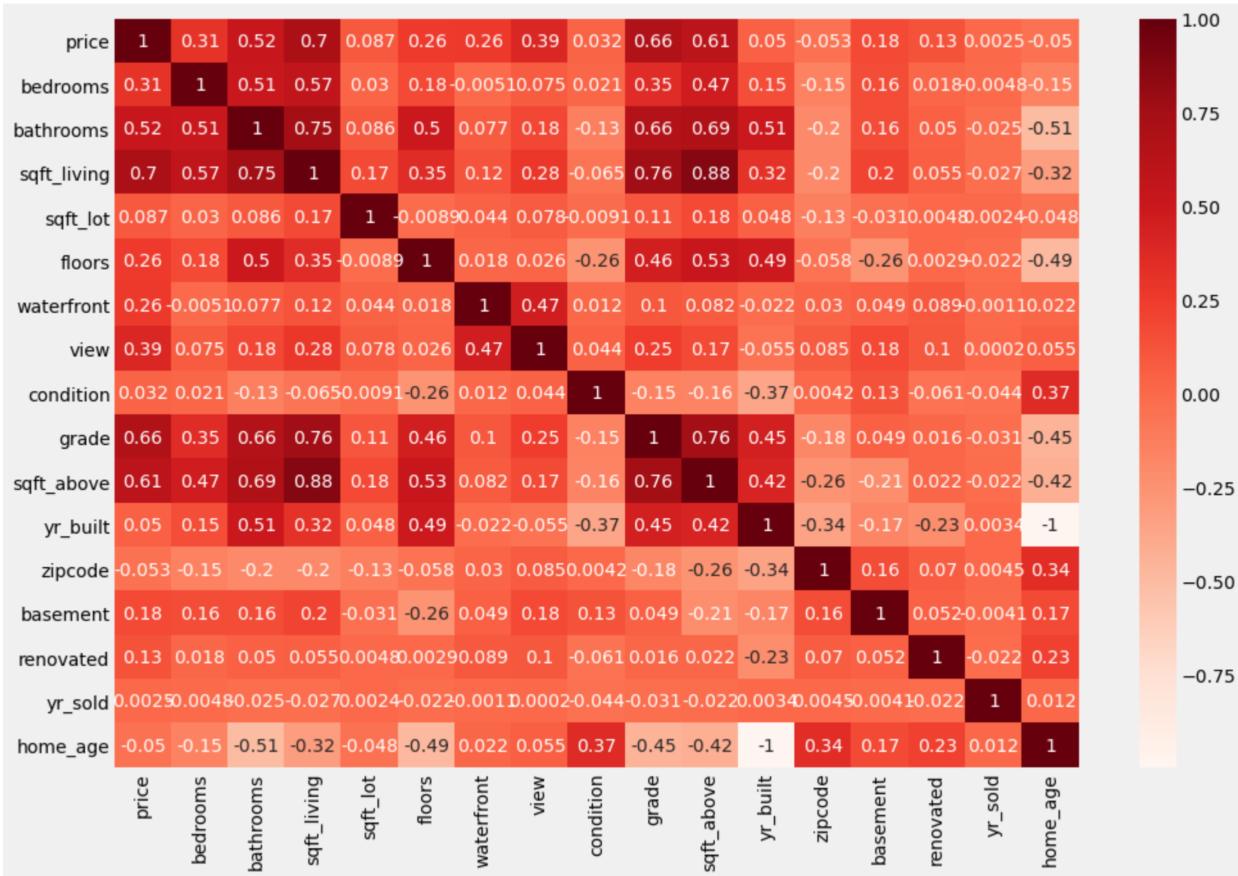
I want to check to see if the independent variables are truly independent from each other by checking for multicollinearity.

6.2.1 Two Variable Multicollinearity Check

In [412]:
```
#remove lat and long columns
df_explore.drop(columns=['lat','long'], inplace=True)
```

In [413]:

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



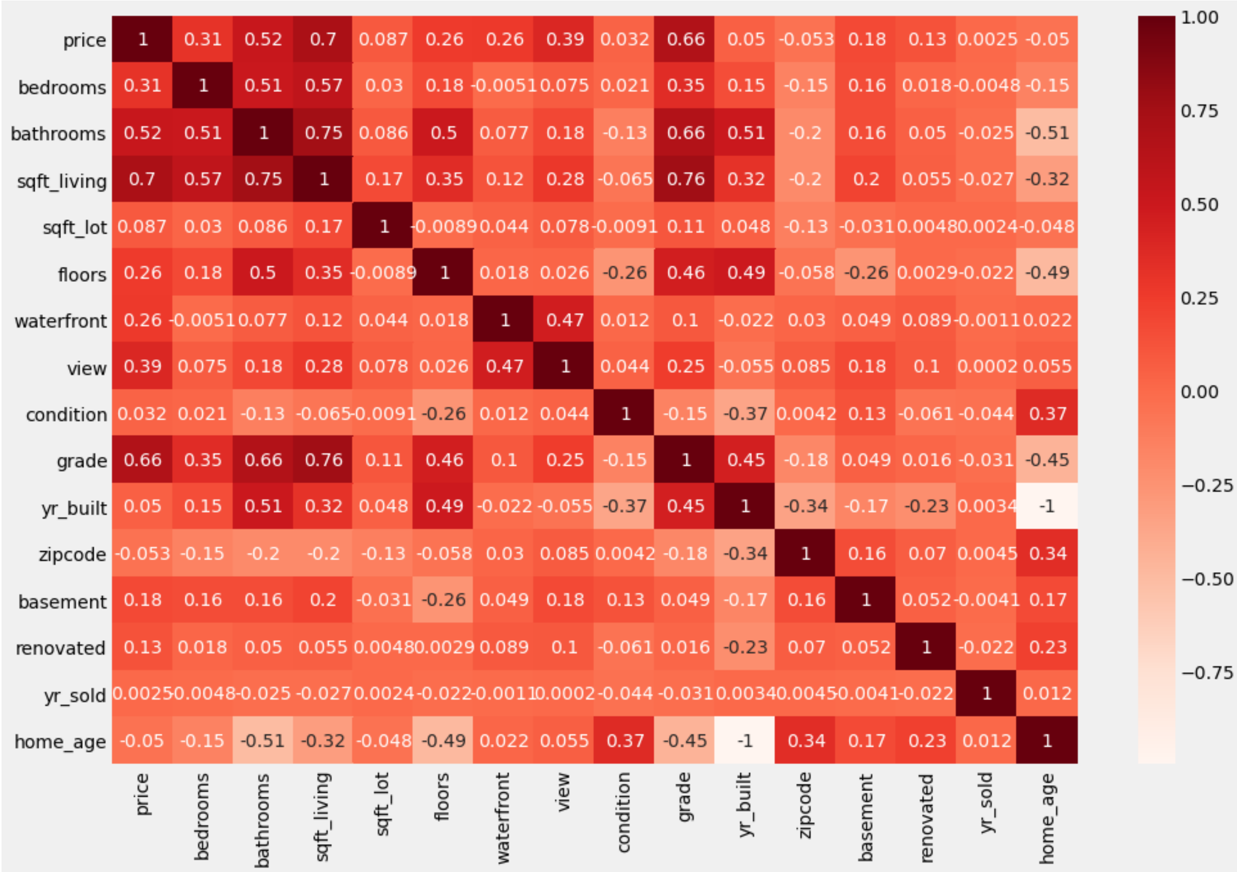
ACTIONS

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

In [414]:

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

In [415]:
```
corr = df_explore.corr()
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(corr, cmap='Reds', annot=True, ax=ax);
```



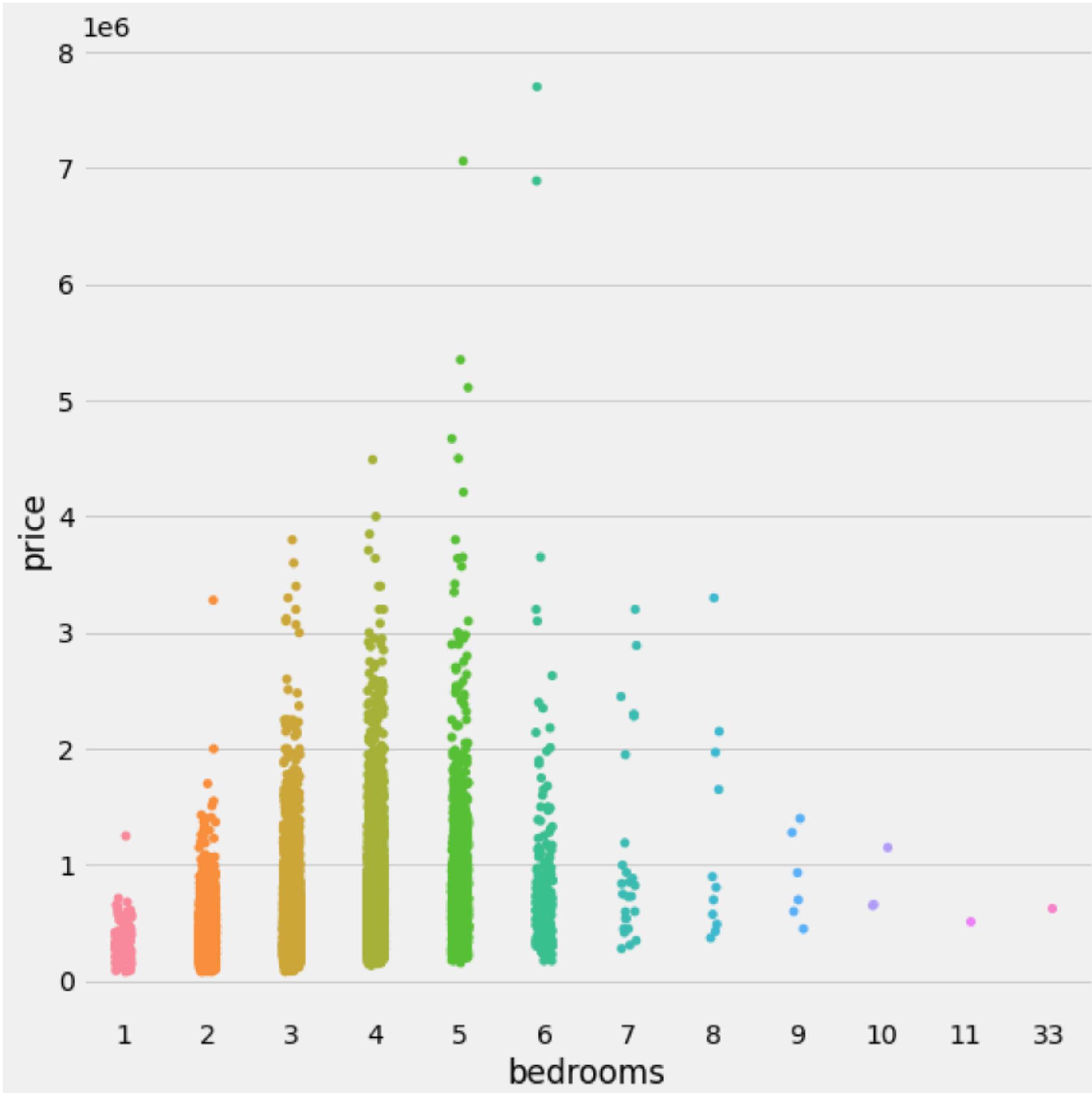
OBSERVATIONS

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

6.3 Outlier Removal

6.3.1 `bedrooms`

In [416]:
```
#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 [417]:
```
#remove outliers
df_explore = df_explore.loc[df_explore['bedrooms'] <= 5]
len(df_explore)
```

Out[417]: 17424

7 Model

In [418]:

```
#create a copy of the explore dataframe
df_model_base = df_explore.copy()
df_model_base
```

Out[418]:

	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

17424 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 [419]:
```
#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 [420]:
```
#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 [421]:
```
#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 [422]:
```
#create a new model dataframe
df_model_1 = df_model_base.copy()
df_model_1
```

Out[422]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditi
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	
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	
5	1,230,000.0	4	4.5	5420	101930	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	

17424 rows × 14 columns

In [423]:
```
#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	
0	221,900.0	-0.39834574796776623	-1.4761162000797934	-0.9902889880360612	-0.22724992
1	538,000.0	-0.39834574796776623	0.19611041462901635	0.5685541395261385	-0.189438771
3	604,000.0	0.8097963429966133	1.199446383454302	-0.1155424847997189	-0.242687898
4	510,000.0	-0.39834574796776623	-0.13833490831274559	-0.4295540500640469	-0.169535663
5	1,230,000.0	0.8097963429966133	3.206118321104874	3.76474328596662	2.05946995

```
                            OLS Regression Results
==============================================================================
=========
```

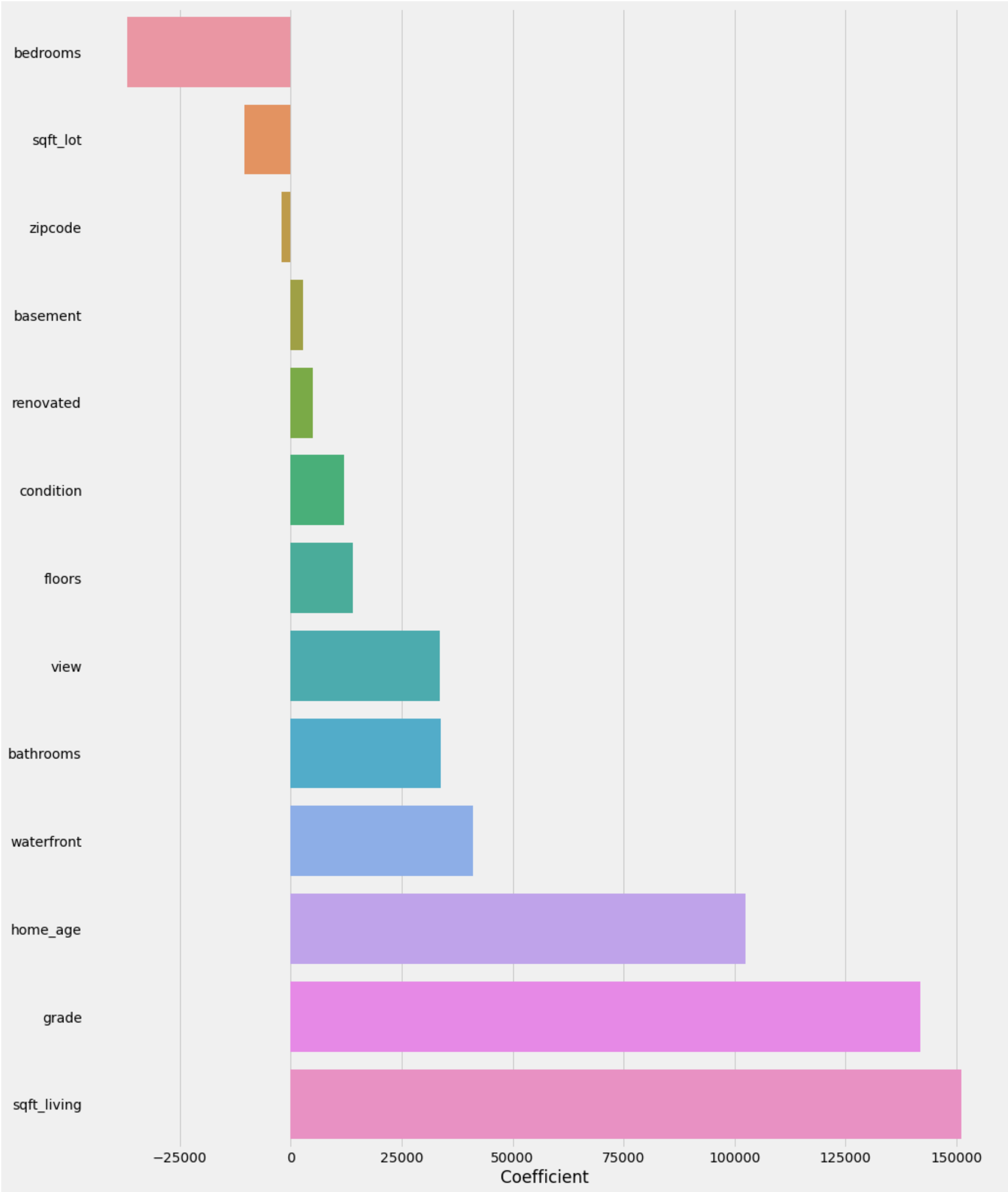
```
Dep. Variable:                  price   R-squared:
0.645
Model:                            OLS   Adj. R-squared:
0.645
Method:                 Least Squares   F-statistic:
2431.
Date:                Sat, 17 Apr 2021   Prob (F-statistic):
0.00
Time:                        23:40:35   Log-Likelihood:            -2
.3834e+05
No. Observations:               17424   AIC:
4.767e+05
Df Residuals:                   17410   BIC:
4.768e+05
Df Model:                          13
Covariance Type:            nonrobust
=====================================================================
==========
                  coef    std err          t      P>|t|      [0.025
0.975]
---------------------------------------------------------------------
----------
Intercept    5.353e+05   1599.894    334.574      0.000    5.32e+05
5.38e+05
bedrooms    -3.689e+04   2065.413    -17.863      0.000   -4.09e+04
-3.28e+04
bathrooms    3.389e+04   2939.233     11.529      0.000    2.81e+04
3.96e+04
sqft_living  1.511e+05   3296.690     45.839      0.000    1.45e+05
1.58e+05
sqft_lot    -1.045e+04   1649.271     -6.335      0.000   -1.37e+04
-7216.189
floors       1.394e+04   2174.766      6.408      0.000    9673.075
1.82e+04
waterfront   4.106e+04   1817.782     22.589      0.000    3.75e+04
4.46e+04
view         3.364e+04   1927.367     17.455      0.000    2.99e+04
3.74e+04
condition    1.192e+04   1783.540      6.684      0.000    8425.023
1.54e+04
grade        1.419e+05   2759.328     51.436      0.000    1.37e+05
1.47e+05
zipcode     -2002.1504   1810.953     -1.106      0.269   -5551.799
1547.498
basement     2824.2373   1869.360      1.511      0.131    -839.896
6488.371
renovated    4977.5044   1709.075      2.912      0.004    1627.546
8327.462
home_age     1.026e+05   2405.692     42.630      0.000    9.78e+04
1.07e+05
```

```
==============================================================================
=========
Omnibus:                    11594.178   Durbin-Watson:
1.975
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            5
30871.059
Skew:                           2.616   Prob(JB):
0.00
Kurtosis:                      29.530   Cond. No.
4.89
==============================================================================
=========

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

In [424]:

```
#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 [425]:
```
#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

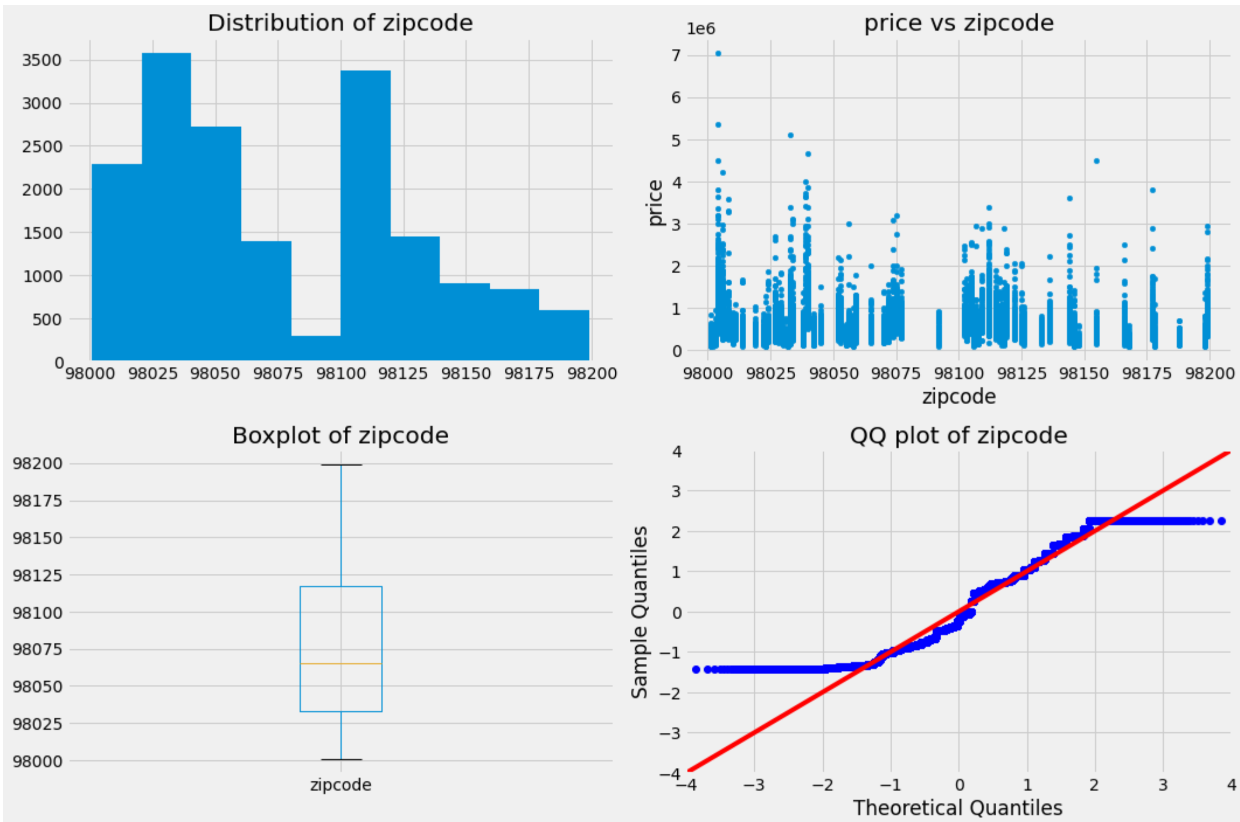
I will evaluate `zipcode` and `grade` for OHE in order to better model this feature.

In [426]:
```
#check column names
df_model_base.columns
```

Out[426]:
```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', '
floors',
       'waterfront', 'view', 'condition', 'grade', 'zipcode', 'baseme
nt',
       'renovated', 'home_age'],
      dtype='object')
```

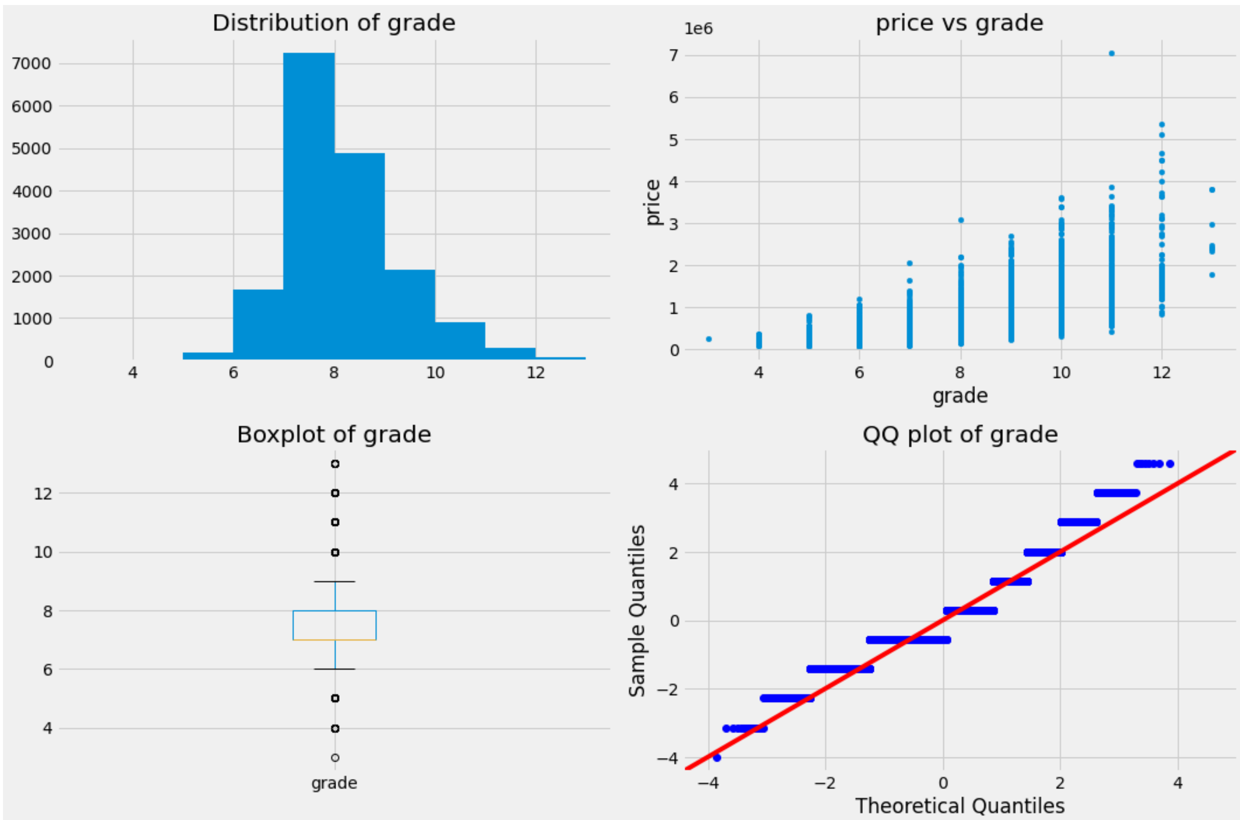
In [427]:
```
#investigate zipcode
get_plots(df_model_base,'zipcode')
```

```
count           17,424.0
mean     98,077.70443067035
std       53.47790092647879
min             98,001.0
25%             98,033.0
50%             98,065.0
75%             98,117.0
max             98,199.0
Name: zipcode, dtype: float64
```



In [428]:
```
#investigate zipcode
get_plots(df_model_base,'grade')
```

```
count              17,424.0
mean       7.65426997245179
std       1.164861827514171
min                     3.0
25%                     7.0
50%                     7.0
75%                     8.0
max                    13.0
Name: grade, dtype: float64
```



OBSERVATIONS

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

In [429]:
```
#fit the data
cat_zipcode = ['zipcode']
encoder = OneHotEncoder(drop='first', sparse=False)
encoder.fit(df_model_base[cat_zipcode])
```

Out[429]:
```
OneHotEncoder(drop='first', sparse=False)
```

In [430]:
```
#transform the data
ohe_vars = encoder.transform(df_model_base[cat_zipcode])
ohe_vars
```

Out[430]:
```
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 [431]:
```
#get the features
encoder.get_feature_names(cat_zipcode)
```

Out[431]:
```
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 [432]:

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

Out[432]:

	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	
3	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	
5	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	

17424 rows × 69 columns

In [433]:

```
#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[433]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditi
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	
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	
5	1,230,000.0	4	4.5	5420	101930	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	

17424 rows × 82 columns

7.3 Model 2

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

In [434]:
```
#copy the base model with the OHE column
df_model_2 = df_model_base.copy()
df_model_2
```

Out[434]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditi
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	
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	
5	1,230,000.0	4	4.5	5420	101930	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	

17424 rows × 82 columns

7.3.1 Model Creation

In [435]:
```
#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	
0	221,900.0	-0.39834574796776623	-1.4761162000797934	-0.9902889880360612	-0.22724992
1	538,000.0	-0.39834574796776623	0.19611041462901635	0.5685541395261385	-0.189438771
3	604,000.0	0.8097963429966133	1.199446383454302	-0.1155424847997189	-0.242687898
4	510,000.0	-0.39834574796776623	-0.13833490831274559	-0.4295540500640469	-0.169535663
5	1,230,000.0	0.8097963429966133	3.206118321104874	3.76474328596662	2.05946995

5 rows × 82 columns

```
                           OLS Regression Results
======================================================================
=========
Dep. Variable:                  price   R-squared:
0.804
Model:                            OLS   Adj. R-squared:
0.803
Method:                 Least Squares   F-statistic:
877.0
Date:                Sat, 17 Apr 2021   Prob (F-statistic):
0.00
Time:                        23:40:37   Log-Likelihood:            -2
.3317e+05
No. Observations:               17424   AIC:
4.665e+05
Df Residuals:                   17342   BIC:
4.671e+05
Df Model:                          81
Covariance Type:            nonrobust
======================================================================
============
                    coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
------------
Intercept      5.353e+05   1191.443    449.273      0.000    5.33e+05
5.38e+05
bedrooms      -2.383e+04   1564.307    -15.237      0.000   -2.69e+04
-2.08e+04
bathrooms      1.983e+04   2208.535      8.980      0.000    1.55e+04
2.42e+04
sqft_living    1.718e+05   2577.845     66.659      0.000    1.67e+05
1.77e+05
sqft_lot       7960.0706   1312.149      6.066      0.000    5388.127
1.05e+04
floors        -1.945e+04   1802.915    -10.790      0.000    -2.3e+04
-1.59e+04
waterfront     4.657e+04   1373.600     33.904      0.000    4.39e+04
4.93e+04
view           4.053e+04   1474.862     27.482      0.000    3.76e+04
4.34e+04
condition       1.41e+04   1362.419     10.349      0.000    1.14e+04
1.68e+04
grade          7.142e+04   2232.315     31.995      0.000     6.7e+04
7.58e+04
basement      -2.747e+04   1504.135    -18.266      0.000   -3.04e+04
-2.45e+04
renovated      6304.1397   1284.039      4.910      0.000    3787.294
```

```
8820.985
home_age        2.168e+04   2079.948     10.424      0.000    1.76e+04
2.58e+04
zipcode_98002  2847.4567   1486.308      1.916      0.055     -65.858
5760.771
zipcode_98003 -1836.3316   1606.122     -1.143      0.253   -4984.493
1311.830
zipcode_98004  9.203e+04   1655.281     55.597      0.000    8.88e+04
9.53e+04
zipcode_98005  2.664e+04   1453.363     18.328      0.000    2.38e+04
2.95e+04
zipcode_98006  3.998e+04   1856.570     21.535      0.000    3.63e+04
4.36e+04
zipcode_98007  1.884e+04   1403.818     13.419      0.000    1.61e+04
2.16e+04
zipcode_98008  2.927e+04   1607.379     18.211      0.000    2.61e+04
3.24e+04
zipcode_98010  4869.4235   1355.127      3.593      0.000    2213.237
7525.610
zipcode_98011  1.171e+04   1478.967      7.915      0.000    8806.611
1.46e+04
zipcode_98014  7851.6245   1406.108      5.584      0.000    5095.511
1.06e+04
zipcode_98019  7635.2857   1463.016      5.219      0.000    4767.627
1.05e+04
zipcode_98022 -3412.0977   1548.000     -2.204      0.028   -6446.334
-377.861
zipcode_98023 -4233.5881   1849.144     -2.289      0.022   -7858.097
-609.079
zipcode_98024  9254.4859   1323.053      6.995      0.000    6661.169
1.18e+04
zipcode_98027  2.348e+04   1757.710     13.356      0.000       2e+04
2.69e+04
zipcode_98028  1.346e+04   1599.981      8.413      0.000    1.03e+04
1.66e+04
zipcode_98029  2.577e+04   1669.723     15.432      0.000    2.25e+04
2.9e+04
zipcode_98030   524.2864   1549.340      0.338      0.735   -2512.576
3561.149
zipcode_98031  1887.8637   1581.064      1.194      0.232   -1211.180
4986.908
zipcode_98032   311.8326   1395.817      0.223      0.823   -2424.110
3047.776
zipcode_98033  5.161e+04   1781.445     28.971      0.000    4.81e+04
5.51e+04
zipcode_98034  3.271e+04   1880.092     17.398      0.000     2.9e+04
3.64e+04
zipcode_98038  5506.2634   1951.699      2.821      0.005    1680.736
9331.790
zipcode_98039  5.924e+04   1280.335     46.269      0.000    5.67e+04
```

```
6.18e+04
zipcode_98040  5.861e+04   1628.824     35.984      0.000    5.54e+04
6.18e+04
zipcode_98042   934.6431   1905.133      0.491      0.624   -2799.609
4668.895
zipcode_98045  9364.5603   1537.995      6.089      0.000    6349.935
1.24e+04
zipcode_98052  3.837e+04   1918.412     20.002      0.000    3.46e+04
4.21e+04
zipcode_98053  2.702e+04   1754.526     15.400      0.000    2.36e+04
3.05e+04
zipcode_98055  5351.4293   1579.503      3.388      0.001    2255.445
8447.414
zipcode_98056  1.343e+04   1745.033      7.694      0.000       1e+04
1.68e+04
zipcode_98058  4621.1727   1784.024      2.590      0.010    1124.306
8118.039
zipcode_98059  1.309e+04   1801.962      7.265      0.000    9559.501
1.66e+04
zipcode_98065  1.077e+04   1631.794      6.600      0.000    7570.936
1.4e+04
zipcode_98070  1586.2173   1403.857      1.130      0.259   -1165.484
4337.918
zipcode_98072  1.775e+04   1596.931     11.113      0.000    1.46e+04
2.09e+04
zipcode_98074  2.518e+04   1792.038     14.049      0.000    2.17e+04
2.87e+04
zipcode_98075  2.337e+04   1711.877     13.651      0.000       2e+04
2.67e+04
zipcode_98077  1.265e+04   1520.040      8.325      0.000    9674.231
1.56e+04
zipcode_98092 -4603.3229   1691.148     -2.722      0.006   -7918.143
-1288.503
zipcode_98102  3.109e+04   1362.006     22.829      0.000    2.84e+04
3.38e+04
zipcode_98103  5.381e+04   1989.762     27.043      0.000    4.99e+04
5.77e+04
zipcode_98105  4.618e+04   1548.344     29.823      0.000    4.31e+04
4.92e+04
zipcode_98106  1.847e+04   1647.751     11.212      0.000    1.52e+04
2.17e+04
zipcode_98107   3.96e+04   1628.434     24.317      0.000    3.64e+04
4.28e+04
zipcode_98108  1.131e+04   1472.762      7.677      0.000    8418.988
1.42e+04
zipcode_98109  3.204e+04   1385.802     23.119      0.000    2.93e+04
3.48e+04
zipcode_98112   6.82e+04   1639.720     41.591      0.000     6.5e+04
7.14e+04
zipcode_98115  5.385e+04   1947.131     27.657      0.000       5e+04
```

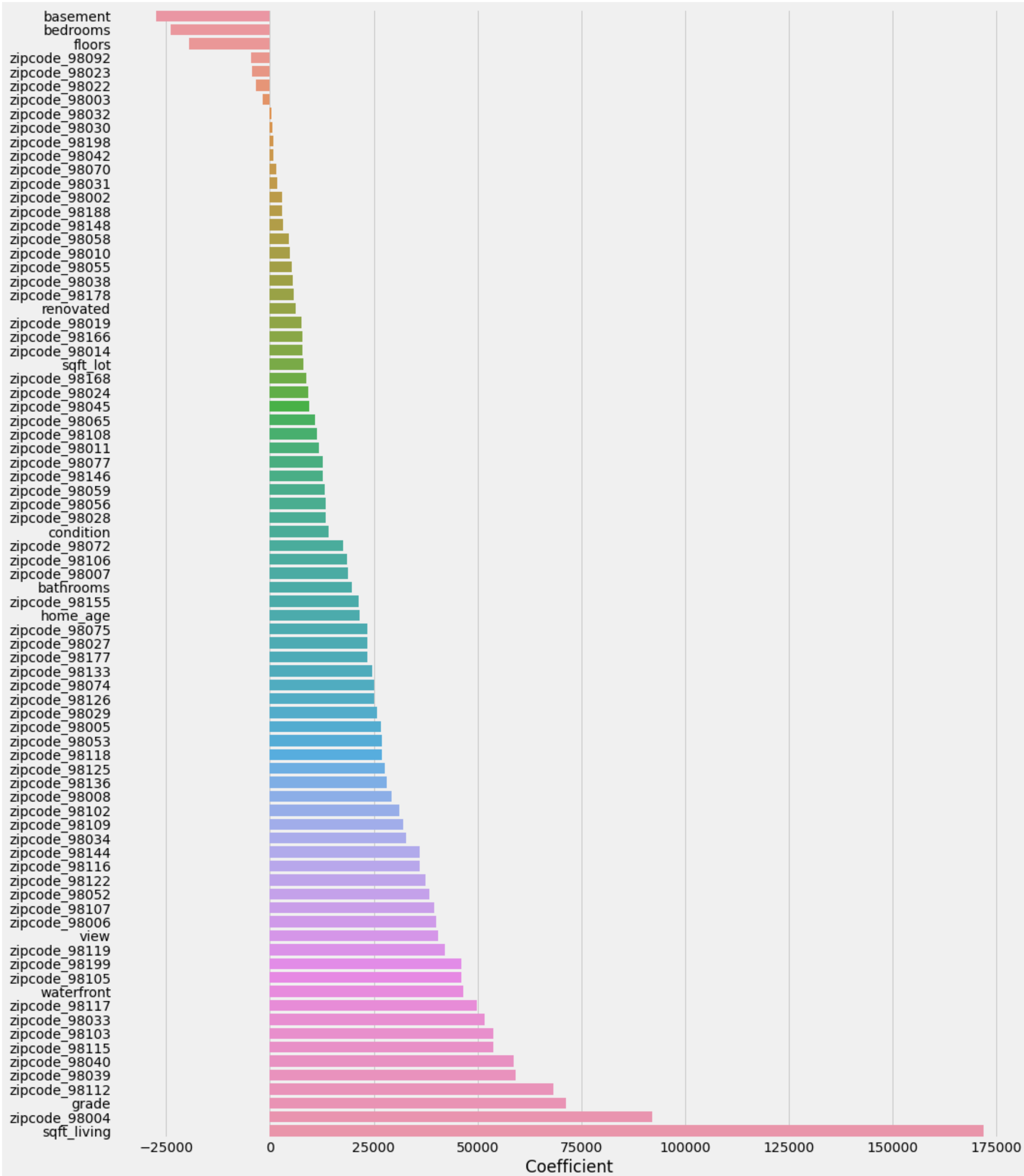
```
5.77e+04
zipcode_98116  3.604e+04   1702.718     21.169      0.000    3.27e+04
3.94e+04
zipcode_98117  4.974e+04   1935.598     25.697      0.000    4.59e+04
5.35e+04
zipcode_98118  2.703e+04   1871.971     14.438      0.000    2.34e+04
3.07e+04
zipcode_98119  4.218e+04   1499.555     28.127      0.000    3.92e+04
4.51e+04
zipcode_98122  3.742e+04   1638.874     22.834      0.000    3.42e+04
4.06e+04
zipcode_98125  2.773e+04   1739.234     15.942      0.000    2.43e+04
3.11e+04
zipcode_98126  2.518e+04   1695.782     14.850      0.000    2.19e+04
2.85e+04
zipcode_98133  2.454e+04   1847.002     13.285      0.000    2.09e+04
2.82e+04
zipcode_98136   2.82e+04   1600.486     17.620      0.000    2.51e+04
3.13e+04
zipcode_98144  3.598e+04   1718.334     20.938      0.000    3.26e+04
3.93e+04
zipcode_98146  1.266e+04   1606.515      7.883      0.000    9514.835
1.58e+04
zipcode_98148  3154.6956   1287.167      2.451      0.014     631.718
5677.673
zipcode_98155  2.127e+04   1767.994     12.028      0.000    1.78e+04
2.47e+04
zipcode_98166  7764.0640   1567.793      4.952      0.000    4691.032
1.08e+04
zipcode_98168  8721.1891   1594.137      5.471      0.000    5596.521
1.18e+04
zipcode_98177   2.35e+04   1564.720     15.017      0.000    2.04e+04
2.66e+04
zipcode_98178  5651.5335   1574.308      3.590      0.000    2565.732
8737.335
zipcode_98188  2967.7182   1398.834      2.122      0.034     225.862
5709.574
zipcode_98198   825.7395   1606.333      0.514      0.607   -2322.835
3974.314
zipcode_98199  4.603e+04   1666.930     27.616      0.000    4.28e+04
4.93e+04
======================================================================
=========
Omnibus:                    15159.032   Durbin-Watson:
1.999
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           18
86413.841
Skew:                           3.630   Prob(JB):
0.00
Kurtosis:                      53.454   Cond. No.
```

```
15.4
==============================================================================
=========

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

In [436]:

```
#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='Coefficien
```

In [437]:
```
#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 0.803
- 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

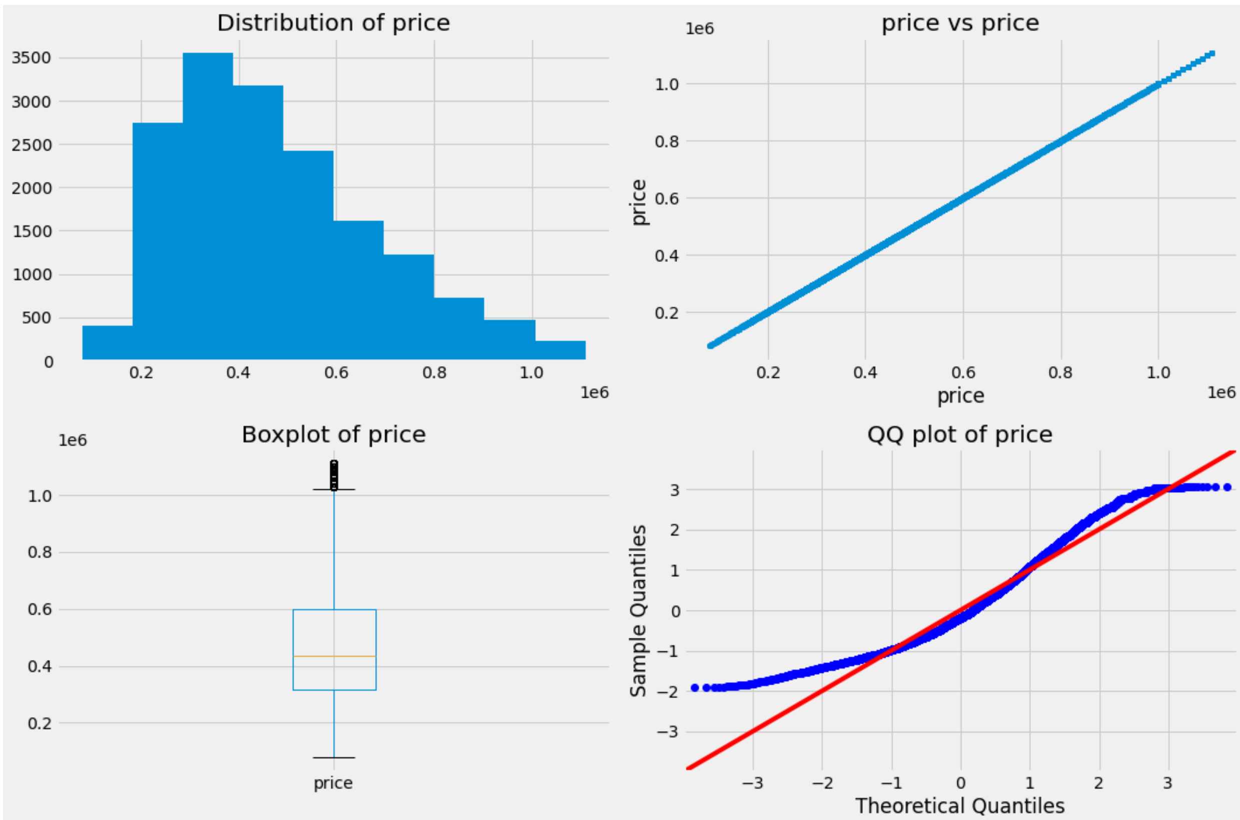
7.3.3 Model Tuning

7.3.3.1 `price` Outlier Removal

I will investigate `price` for outliers.

In [438]:
```
get_plots(df_model_base,'price',outlier='iqr')
```

```
The number of rows removed is 897
count            17,424.0
mean     535,283.093721304
std      354,215.472622472
min              80,000.0
25%             320,000.0
50%             450,000.0
75%             639,912.5
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 [439]:

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

Out[439]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conditi
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	
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	
5	1,230,000.0	4	4.5	5420	101930	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	

17424 rows × 82 columns

In [440]:

```
#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 897 outliers removed.
```

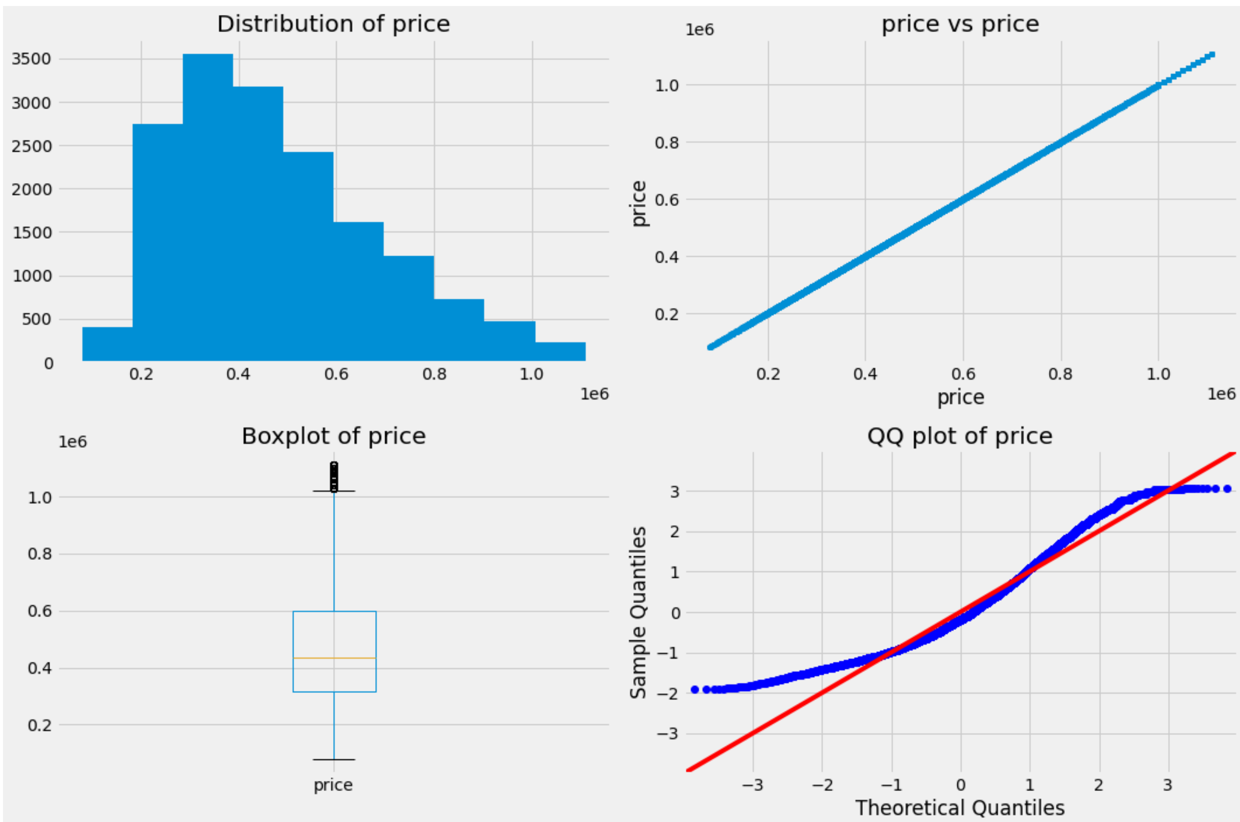
Out[440]:

	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	
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	
6	257,500.0	3	2.25	1715	6819	2.0	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	

16527 rows × 82 columns

In [441]:
```
#recheck the price column
get_plots(df_model_3,'price',outlier='none')
```

```
count                  16,527.0
mean     475,336.27548859443
std      206,876.3808524632
min                  80,000.0
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 [442]:
```
#view model_3 dataframe
df_model_3
```

Out[442]:

	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	
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	
6	257,500.0	3	2.25	1715	6819	2.0	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	

16527 rows × 82 columns

7.4.1 Model Creation

In [443]:
```
#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	
0	221,900.0	-0.35980469755201844	-1.5005865056176506	-1.026945274052607	-0.221893473
1	538,000.0	-0.35980469755201844	0.2961643839368538	0.7922061878543151	-0.183066035
3	604,000.0	0.8635016666615617	1.3742149176695566	-0.006126468090449303	-0.23774638
4	510,000.0	-0.35980469755201844	-0.0631857939740471	-0.3725742445896854	-0.16262797
6	257,500.0	-0.35980469755201844	0.2961643839368538	-0.3267682725272809	-0.19338262

5 rows × 82 columns

```
                            OLS Regression Results
======================================================================
=========
Dep. Variable:                  price   R-squared:
0.828
Model:                            OLS   Adj. R-squared:
0.827
Method:                 Least Squares   F-statistic:
977.1
Date:                Sat, 17 Apr 2021   Prob (F-statistic):
0.00
Time:                        23:40:39   Log-Likelihood:             -2
.1119e+05
No. Observations:               16527   AIC:
4.226e+05
Df Residuals:                   16445   BIC:
4.232e+05
Df Model:                          81
Covariance Type:            nonrobust
======================================================================
============
                    coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
------------
Intercept      4.753e+05    669.087    710.426      0.000    4.74e+05
4.77e+05
bedrooms      -3738.0637    882.895     -4.234      0.000   -5468.633
-2007.494
bathrooms      1.031e+04   1185.810      8.692      0.000    7982.441
1.26e+04
sqft_living    8.805e+04   1339.342     65.745      0.000    8.54e+04
9.07e+04
sqft_lot       1.084e+04    735.303     14.741      0.000    9397.481
1.23e+04
floors        -5570.7182   1025.588     -5.432      0.000   -7580.981
-3560.455
waterfront     4413.4761    741.636      5.951      0.000    2959.789
5867.163
view           2.132e+04    769.366     27.705      0.000    1.98e+04
2.28e+04
condition      1.285e+04    763.833     16.829      0.000    1.14e+04
1.44e+04
grade          5.242e+04   1148.530     45.639      0.000    5.02e+04
5.47e+04
basement      -1.268e+04    848.013    -14.958      0.000   -1.43e+04
-1.1e+04
renovated      5463.9265    714.646      7.646      0.000    4063.143
6864.710
```

```
home_age       1.649e+04   1183.305     13.938      0.000    1.42e+04
1.88e+04
zipcode_98002   690.7288    834.744      0.827      0.408    -945.460
2326.918
zipcode_98003  -605.6287    901.805     -0.672      0.502   -2373.264
1162.006
zipcode_98004  4.562e+04    807.908     56.467      0.000     4.4e+04
4.72e+04
zipcode_98005  2.823e+04    803.956     35.110      0.000    2.67e+04
2.98e+04
zipcode_98006  3.809e+04    986.989     38.592      0.000    3.62e+04
4e+04
zipcode_98007  1.944e+04    783.234     24.822      0.000    1.79e+04
2.1e+04
zipcode_98008  2.729e+04    890.585     30.646      0.000    2.55e+04
2.9e+04
zipcode_98010  6484.5758    761.260      8.518      0.000    4992.424
7976.728
zipcode_98011  1.439e+04    830.624     17.320      0.000    1.28e+04
1.6e+04
zipcode_98014  8866.8036    785.714     11.285      0.000    7326.720
1.04e+04
zipcode_98019  9474.0582    821.762     11.529      0.000    7863.316
1.11e+04
zipcode_98022  -532.3810    870.675     -0.611      0.541   -2238.998
1174.236
zipcode_98023 -2784.1068   1036.566     -2.686      0.007   -4815.889
-752.325
zipcode_98024  8986.9320    736.719     12.199      0.000    7542.884
1.04e+04
zipcode_98027  2.723e+04    976.128     27.892      0.000    2.53e+04
2.91e+04
zipcode_98028  1.594e+04    896.656     17.778      0.000    1.42e+04
1.77e+04
zipcode_98029  2.914e+04    933.954     31.202      0.000    2.73e+04
3.1e+04
zipcode_98030   981.6896    869.852      1.129      0.259    -723.314
2686.693
zipcode_98031  1671.2277    887.657      1.883      0.060     -68.676
3411.132
zipcode_98032  -674.2099    783.882     -0.860      0.390   -2210.703
862.283
zipcode_98033  4.387e+04    959.024     45.744      0.000     4.2e+04
4.57e+04
zipcode_98034  3.009e+04   1045.968     28.767      0.000     2.8e+04
3.21e+04
zipcode_98038  7819.8287   1093.407      7.152      0.000    5676.632
9963.025
zipcode_98039  7973.8217    672.811     11.851      0.000    6655.039
9292.605
```

```
zipcode_98040  3.918e+04   821.418     47.700      0.000    3.76e+04
4.08e+04
zipcode_98042  2228.5210   1069.250     2.084      0.037     132.675
4324.367
zipcode_98045   1.15e+04   862.742     13.332      0.000    9810.810
1.32e+04
zipcode_98052  4.335e+04   1069.763    40.520      0.000    4.13e+04
4.54e+04
zipcode_98053  3.382e+04   970.708     34.844      0.000    3.19e+04
3.57e+04
zipcode_98055  4815.7805   886.855      5.430      0.000    3077.448
6554.113
zipcode_98056  1.443e+04   977.320     14.765      0.000    1.25e+04
1.63e+04
zipcode_98058  5715.4576   1000.673     5.712      0.000    3754.031
7676.885
zipcode_98059  1.643e+04   1002.332    16.392      0.000    1.45e+04
1.84e+04
zipcode_98065  1.696e+04   913.756     18.559      0.000    1.52e+04
1.87e+04
zipcode_98070  9444.5175   793.138     11.908      0.000    7889.881
1.1e+04
zipcode_98072  2.116e+04   889.769     23.780      0.000    1.94e+04
2.29e+04
zipcode_98074  3.224e+04   995.027     32.401      0.000    3.03e+04
3.42e+04
zipcode_98075  3.227e+04   947.100     34.075      0.000    3.04e+04
3.41e+04
zipcode_98077  1.854e+04   846.730     21.893      0.000    1.69e+04
2.02e+04
zipcode_98092 -1408.3070   949.737     -1.483      0.138   -3269.895
453.281
zipcode_98102  2.411e+04   751.211     32.092      0.000    2.26e+04
2.56e+04
zipcode_98103  4.981e+04   1115.002    44.672      0.000    4.76e+04
5.2e+04
zipcode_98105  3.305e+04   830.852     39.781      0.000    3.14e+04
3.47e+04
zipcode_98106  1.427e+04   926.022     15.406      0.000    1.25e+04
1.61e+04
zipcode_98107  3.536e+04   913.607     38.703      0.000    3.36e+04
3.71e+04
zipcode_98108   1.01e+04   827.480     12.203      0.000    8475.491
1.17e+04
zipcode_98109  2.597e+04   760.328     34.162      0.000    2.45e+04
2.75e+04
zipcode_98112  3.791e+04   835.360     45.385      0.000    3.63e+04
3.96e+04
zipcode_98115  5.014e+04   1080.673    46.393      0.000     4.8e+04
5.23e+04
```

```
zipcode_98116  3.548e+04    947.005     37.464      0.000    3.36e+04
3.73e+04
zipcode_98117  4.791e+04   1084.578     44.170      0.000    4.58e+04
5e+04
zipcode_98118  2.467e+04   1049.473     23.502      0.000    2.26e+04
2.67e+04
zipcode_98119  3.306e+04    816.276     40.502      0.000    3.15e+04
3.47e+04
zipcode_98122  3.297e+04    910.289     36.214      0.000    3.12e+04
3.47e+04
zipcode_98125   2.61e+04    972.910     26.824      0.000    2.42e+04
2.8e+04
zipcode_98126   2.38e+04    953.613     24.963      0.000    2.19e+04
2.57e+04
zipcode_98133  2.193e+04   1037.495     21.139      0.000    1.99e+04
2.4e+04
zipcode_98136  2.804e+04    894.795     31.331      0.000    2.63e+04
2.98e+04
zipcode_98144  2.961e+04    948.821     31.205      0.000    2.77e+04
3.15e+04
zipcode_98146  1.204e+04    898.807     13.390      0.000    1.03e+04
1.38e+04
zipcode_98148  2492.2259    722.879      3.448      0.001    1075.304
3909.147
zipcode_98155  1.911e+04    989.753     19.309      0.000    1.72e+04
2.11e+04
zipcode_98166  1.135e+04    874.371     12.986      0.000    9640.420
1.31e+04
zipcode_98168  5306.7810    895.606      5.925      0.000    3551.296
7062.266
zipcode_98177  2.182e+04    858.887     25.409      0.000    2.01e+04
2.35e+04
zipcode_98178  5993.3084    883.791      6.781      0.000    4260.982
7725.635
zipcode_98188  2323.7967    785.505      2.958      0.003     784.122
3863.471
zipcode_98198  2460.5070    900.742      2.732      0.006     694.956
4226.058
zipcode_98199  3.997e+04    903.833     44.228      0.000    3.82e+04
4.17e+04
======================================================================
=========
Omnibus:                     1493.017   Durbin-Watson:
1.987
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
5641.662
Skew:                           0.406   Prob(JB):
0.00
Kurtosis:                       5.744   Cond. No.
14.9
```

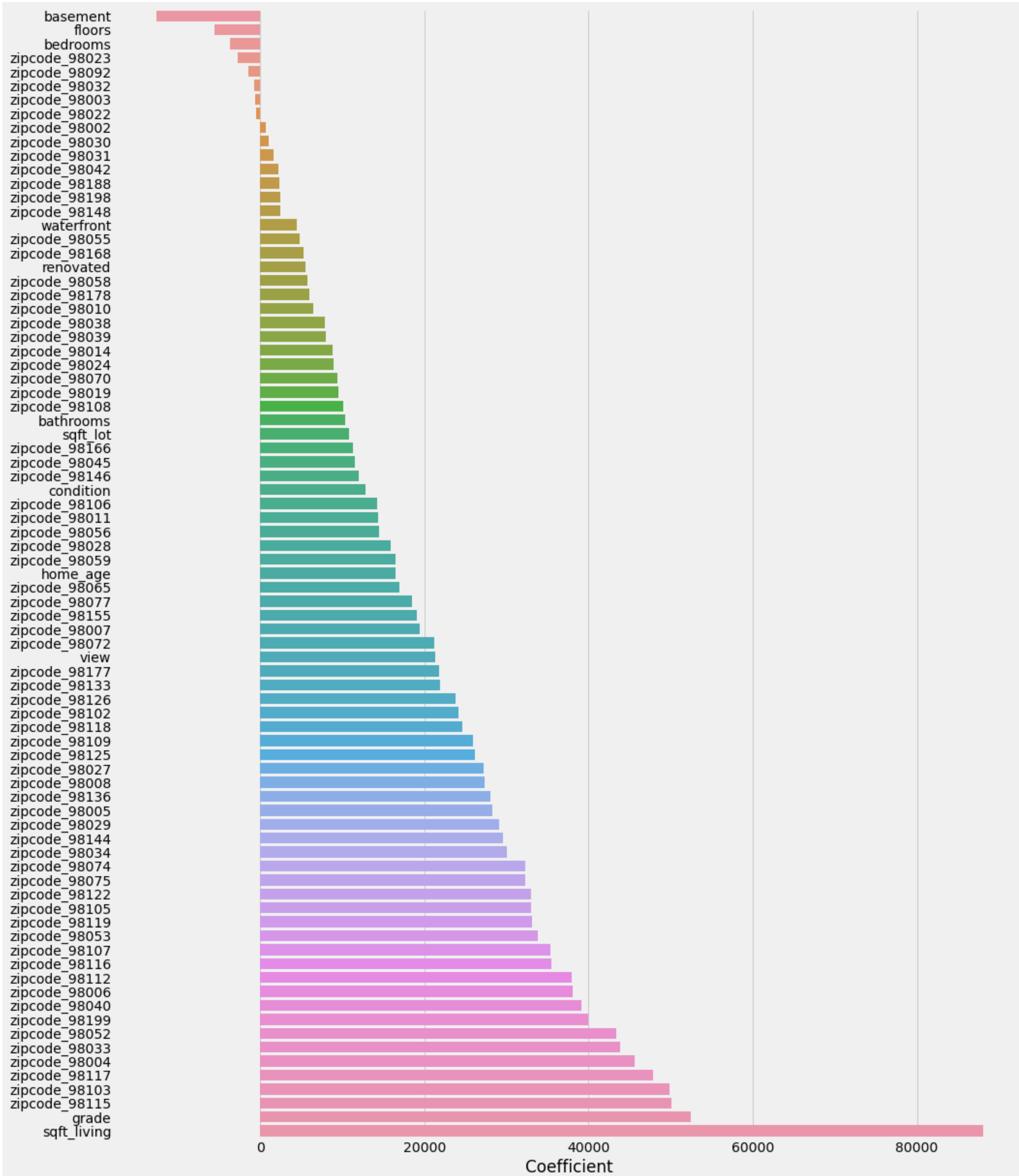
```
==============================================================================
=========

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

In [444]:

```
#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 [445]:

```
#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

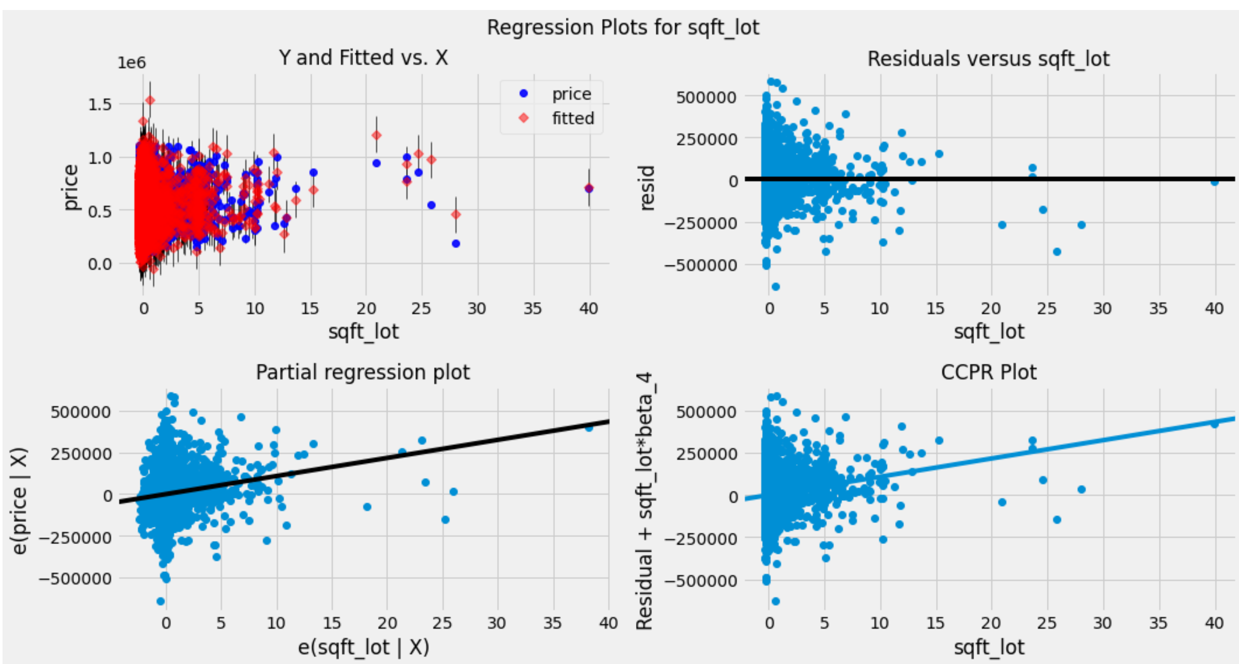
ACTIONS

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

7.4.3 Model Tuning

In [446]:

```
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

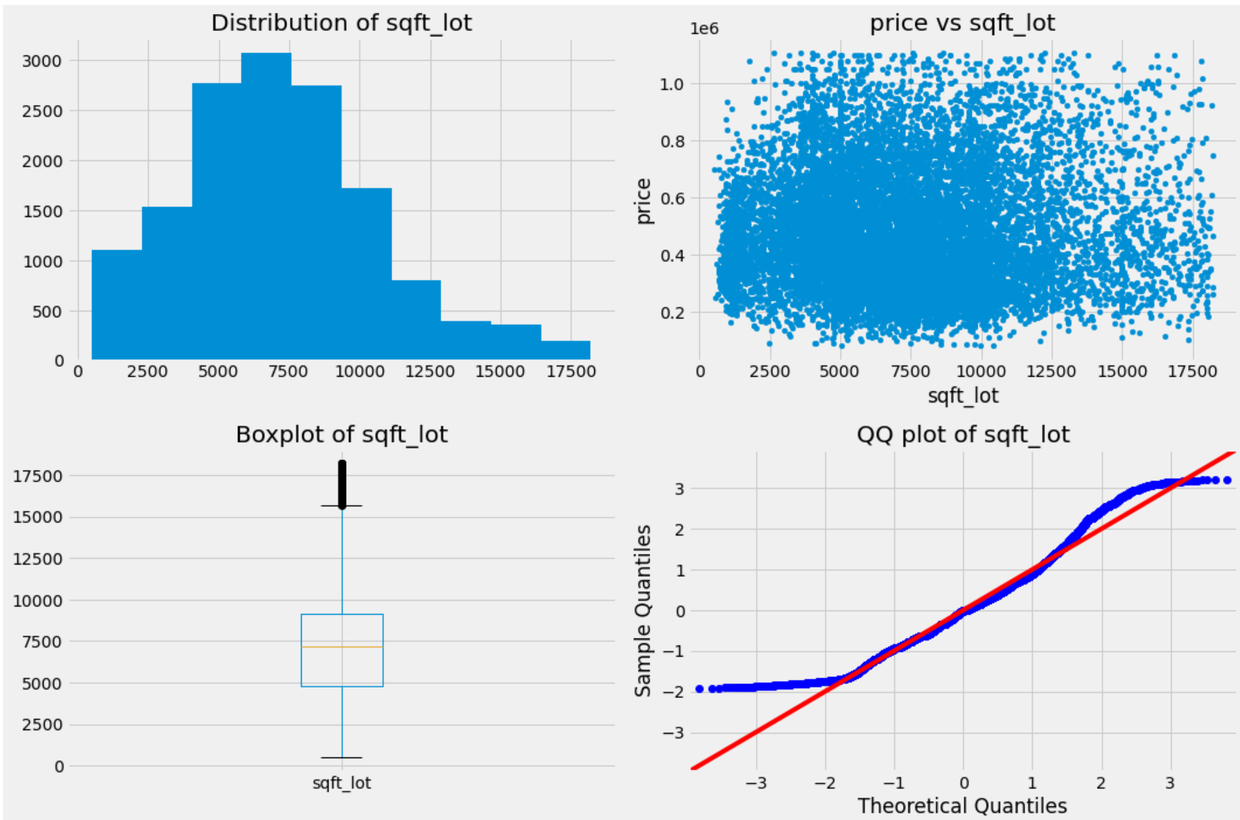
- Will remove the outliers of sqft_lot first and maybe and see if there is any improvement in the overall model.

7.4.3.1 `sqft_lot` Outlier Removal

I will investigate `sqft_lot` for outliers.

In [447]:
```
get_plots(df_model_base,'sqft_lot',outlier='iqr')
```

```
The number of rows removed is 1814
count          16,527.0
mean    14,748.061293640709
std      41,001.93285064867
min               520.0
25%             5,000.0
50%             7,500.0
75%            10,283.5
max         1,651,359.0
Name: sqft_lot, dtype: float64
```



OBSERVATIONS

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

In [448]:
```
#create a copy of model_2 to set up model_3
df_model_4 = df_model_base.copy()
df_model_4
```

Out[448]:

	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	
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	
6	257,500.0	3	2.25	1715	6819	2.0	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	

16527 rows × 82 columns

In [449]:

```
#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 1814 outliers removed.
```

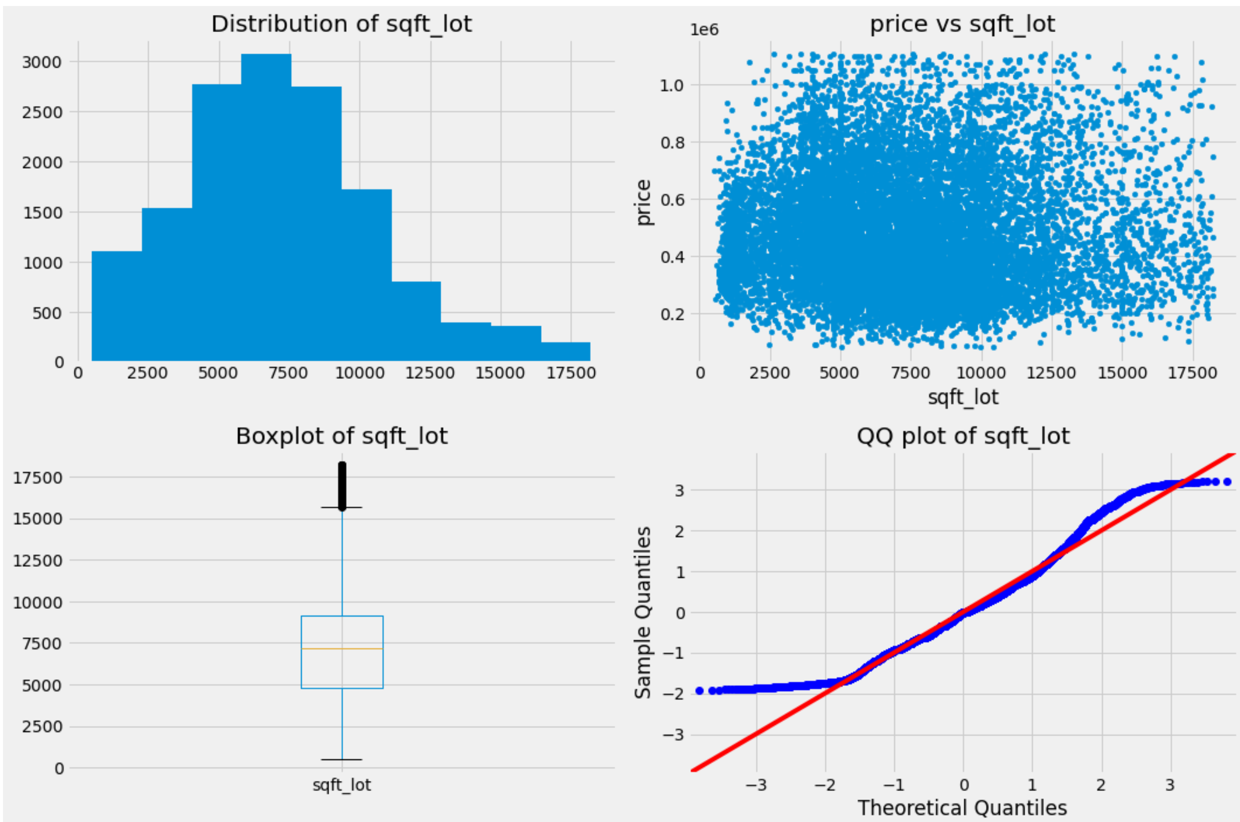
Out[449]:

	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	
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	
6	257,500.0	3	2.25	1715	6819	2.0	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	

14713 rows × 82 columns

In [450]:
```
#recheck the sqft_lot column
get_plots(df_model_4,'sqft_lot',outlier='none')
```

```
count          14,713.0
mean     7,188.183307279277
std      3,443.210270301906
min               520.0
25%             4,800.0
50%             7,161.0
75%             9,163.0
max            18,200.0
Name: sqft_lot, dtype: float64
```



7.5 Model 4

In [451]:

```
#view model_3 dataframe
df_model_4
```

Out[451]:

	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	
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	
6	257,500.0	3	2.25	1715	6819	2.0	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	

14713 rows × 82 columns

7.5.1 Model Creation

In [452]:

```
#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	
0	221,900.0	-0.3393806244251373	-1.4785611049109177	-1.0026384463856017	-0.44672941
1	538,000.0	-0.3393806244251373	0.33264793499935746	0.9343861946228085	0.0156298014
3	604,000.0	0.88267202539548	1.4193733589455224	0.08432502123062846	-0.63550
4	510,000.0	-0.3393806244251373	-0.029593872982697576	-0.30586699278545415	0.25900732
6	257,500.0	-0.3393806244251373	0.33264793499935746	-0.25709299103344385	-0.107220668

5 rows × 82 columns

```
                            OLS Regression Results
======================================================================
=========
Dep. Variable:                  price   R-squared:
0.837
Model:                            OLS   Adj. R-squared:
0.836
Method:                 Least Squares   F-statistic:
928.9
Date:                Sat, 17 Apr 2021   Prob (F-statistic):
0.00
Time:                        23:40:42   Log-Likelihood:             -1
.8736e+05
No. Observations:               14713   AIC:
3.749e+05
Df Residuals:                   14631   BIC:
3.755e+05
Df Model:                          81
Covariance Type:            nonrobust
======================================================================
============
                    coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
------------
Intercept      4.655e+05    678.507    686.053      0.000    4.64e+05
4.67e+05
bedrooms      -3462.2915    911.665     -3.798      0.000   -5249.269
-1675.314
bathrooms      1.058e+04   1205.433      8.774      0.000    8213.190
1.29e+04
sqft_living    8.499e+04   1359.254     62.528      0.000    8.23e+04
8.77e+04
sqft_lot       5391.0531    981.067      5.495      0.000    3468.038
7314.069
floors        -4425.0934   1110.844     -3.984      0.000   -6602.487
-2247.699
waterfront     4949.2598    738.321      6.703      0.000    3502.057
6396.463
view           2.091e+04    769.983     27.152      0.000    1.94e+04
2.24e+04
condition      1.307e+04    778.881     16.775      0.000    1.15e+04
1.46e+04
grade          4.826e+04   1159.868     41.605      0.000     4.6e+04
5.05e+04
basement      -1.177e+04    871.659    -13.507      0.000   -1.35e+04
-1.01e+04
renovated      5284.2503    725.461      7.284      0.000    3862.256
6706.244
```

```
home_age        1.68e+04   1246.391     13.482      0.000    1.44e+04
1.92e+04
zipcode_98002  1087.8128    874.680      1.244      0.214    -626.669
2802.295
zipcode_98003  -542.2482    940.987     -0.576      0.564   -2386.702
1302.206
zipcode_98004  4.626e+04    829.253     55.791      0.000    4.46e+04
4.79e+04
zipcode_98005  2.629e+04    799.913     32.872      0.000    2.47e+04
2.79e+04
zipcode_98006  3.831e+04   1021.295     37.513      0.000    3.63e+04
4.03e+04
zipcode_98007  2.012e+04    810.831     24.820      0.000    1.85e+04
2.17e+04
zipcode_98008   2.81e+04    931.968     30.148      0.000    2.63e+04
2.99e+04
zipcode_98010  4985.2323    729.898      6.830      0.000    3554.540
6415.925
zipcode_98011  1.425e+04    854.139     16.688      0.000    1.26e+04
1.59e+04
zipcode_98014  5697.5583    733.193      7.771      0.000    4260.408
7134.708
zipcode_98019  7917.4331    812.397      9.746      0.000    6325.032
9509.834
zipcode_98022   -12.0443    827.327     -0.015      0.988   -1633.709
1609.620
zipcode_98023 -2526.7942   1085.697     -2.327      0.020   -4654.897
-398.691
zipcode_98024  4845.0438    709.578      6.828      0.000    3454.181
6235.907
zipcode_98027  2.802e+04    927.609     30.208      0.000    2.62e+04
2.98e+04
zipcode_98028  1.572e+04    928.267     16.939      0.000    1.39e+04
1.75e+04
zipcode_98029  3.057e+04    983.389     31.082      0.000    2.86e+04
3.25e+04
zipcode_98030   743.1074    903.763      0.822      0.411   -1028.382
2514.597
zipcode_98031  1537.9587    917.478      1.676      0.094    -260.413
3336.331
zipcode_98032  -295.0154    806.958     -0.366      0.715   -1876.754
1286.723
zipcode_98033  4.432e+04   1003.964     44.148      0.000    4.24e+04
4.63e+04
zipcode_98034   3.14e+04   1106.189     28.390      0.000    2.92e+04
3.36e+04
zipcode_98038  7207.1707   1109.938      6.493      0.000    5031.553
9382.789
zipcode_98039  8416.9041    683.032     12.323      0.000    7078.074
9755.734
```

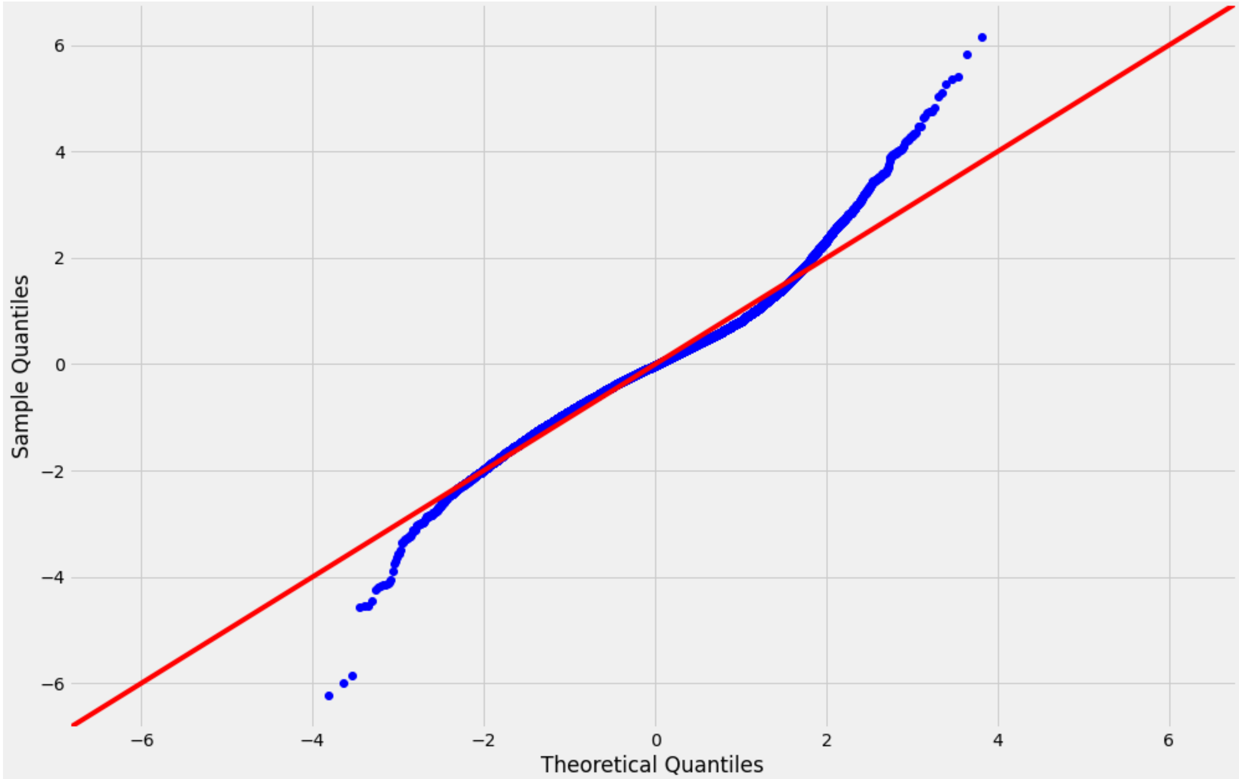
```
zipcode_98040  3.993e+04   847.465     47.117      0.000    3.83e+04
4.16e+04
zipcode_98042  2037.9571  1074.556      1.897      0.058     -68.308
4144.222
zipcode_98045  9037.1926   834.596     10.828      0.000    7401.280
1.07e+04
zipcode_98052   4.37e+04  1113.604     39.240      0.000    4.15e+04
4.59e+04
zipcode_98053  3.044e+04   927.713     32.814      0.000    2.86e+04
3.23e+04
zipcode_98055  5141.5935   926.141      5.552      0.000    3326.239
6956.947
zipcode_98056  1.456e+04  1023.003     14.230      0.000    1.26e+04
1.66e+04
zipcode_98058  5262.7359  1023.052      5.144      0.000    3257.424
7268.048
zipcode_98059  1.558e+04  1024.500     15.211      0.000    1.36e+04
1.76e+04
zipcode_98065  1.762e+04   952.583     18.497      0.000    1.58e+04
1.95e+04
zipcode_98070  2771.4419   725.416      3.820      0.000    1349.535
4193.349
zipcode_98072  1.387e+04   826.418     16.783      0.000    1.22e+04
1.55e+04
zipcode_98074  3.153e+04  1019.614     30.919      0.000    2.95e+04
3.35e+04
zipcode_98075  2.975e+04   947.370     31.404      0.000    2.79e+04
3.16e+04
zipcode_98077  9811.2057   732.329     13.397      0.000    8375.749
1.12e+04
zipcode_98092 -2017.1068   930.761     -2.167      0.030   -3841.517
-192.697
zipcode_98102  2.605e+04   782.619     33.282      0.000    2.45e+04
2.76e+04
zipcode_98103  5.405e+04  1222.330     44.220      0.000    5.17e+04
5.64e+04
zipcode_98105  3.564e+04   882.277     40.391      0.000    3.39e+04
3.74e+04
zipcode_98106  1.576e+04   985.344     15.999      0.000    1.38e+04
1.77e+04
zipcode_98107  3.846e+04   981.760     39.175      0.000    3.65e+04
4.04e+04
zipcode_98108  1.118e+04   872.810     12.810      0.000    9469.713
1.29e+04
zipcode_98109  2.808e+04   796.107     35.266      0.000    2.65e+04
2.96e+04
zipcode_98112  4.084e+04   888.936     45.942      0.000    3.91e+04
4.26e+04
zipcode_98115  5.384e+04  1174.536     45.843      0.000    5.15e+04
5.61e+04
```

```
zipcode_98116  3.816e+04   1018.795     37.459      0.000    3.62e+04
4.02e+04
zipcode_98117   5.18e+04   1184.587     43.732      0.000    4.95e+04
5.41e+04
zipcode_98118  2.693e+04   1135.281     23.717      0.000    2.47e+04
2.92e+04
zipcode_98119  3.569e+04    865.510     41.235      0.000     3.4e+04
3.74e+04
zipcode_98122  3.588e+04    979.136     36.646      0.000     3.4e+04
3.78e+04
zipcode_98125   2.75e+04   1032.765     26.631      0.000    2.55e+04
2.95e+04
zipcode_98126    2.6e+04   1024.962     25.365      0.000     2.4e+04
2.8e+04
zipcode_98133  2.369e+04   1108.626     21.365      0.000    2.15e+04
2.59e+04
zipcode_98136  3.024e+04    950.359     31.820      0.000    2.84e+04
3.21e+04
zipcode_98144  3.223e+04   1026.274     31.402      0.000    3.02e+04
3.42e+04
zipcode_98146  1.271e+04    939.827     13.519      0.000    1.09e+04
1.45e+04
zipcode_98148  2724.9929    741.438      3.675      0.000    1271.681
4178.305
zipcode_98155  1.992e+04   1037.138     19.210      0.000    1.79e+04
2.2e+04
zipcode_98166  9923.5322    885.575     11.206      0.000    8187.694
1.17e+04
zipcode_98168  5758.3648    917.839      6.274      0.000    3959.285
7557.444
zipcode_98177  2.156e+04    886.032     24.335      0.000    1.98e+04
2.33e+04
zipcode_98178  6421.5638    933.172      6.881      0.000    4592.430
8250.698
zipcode_98188  2224.2459    808.527      2.751      0.006     639.430
3809.061
zipcode_98198  2832.4799    930.130      3.045      0.002    1009.307
4655.653
zipcode_98199  4.284e+04    967.289     44.286      0.000    4.09e+04
4.47e+04
======================================================================
=========
Omnibus:                     1111.490   Durbin-Watson:
1.991
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
3643.258
Skew:                           0.364   Prob(JB):
0.00
Kurtosis:                       5.326   Cond. No.
15.4
```

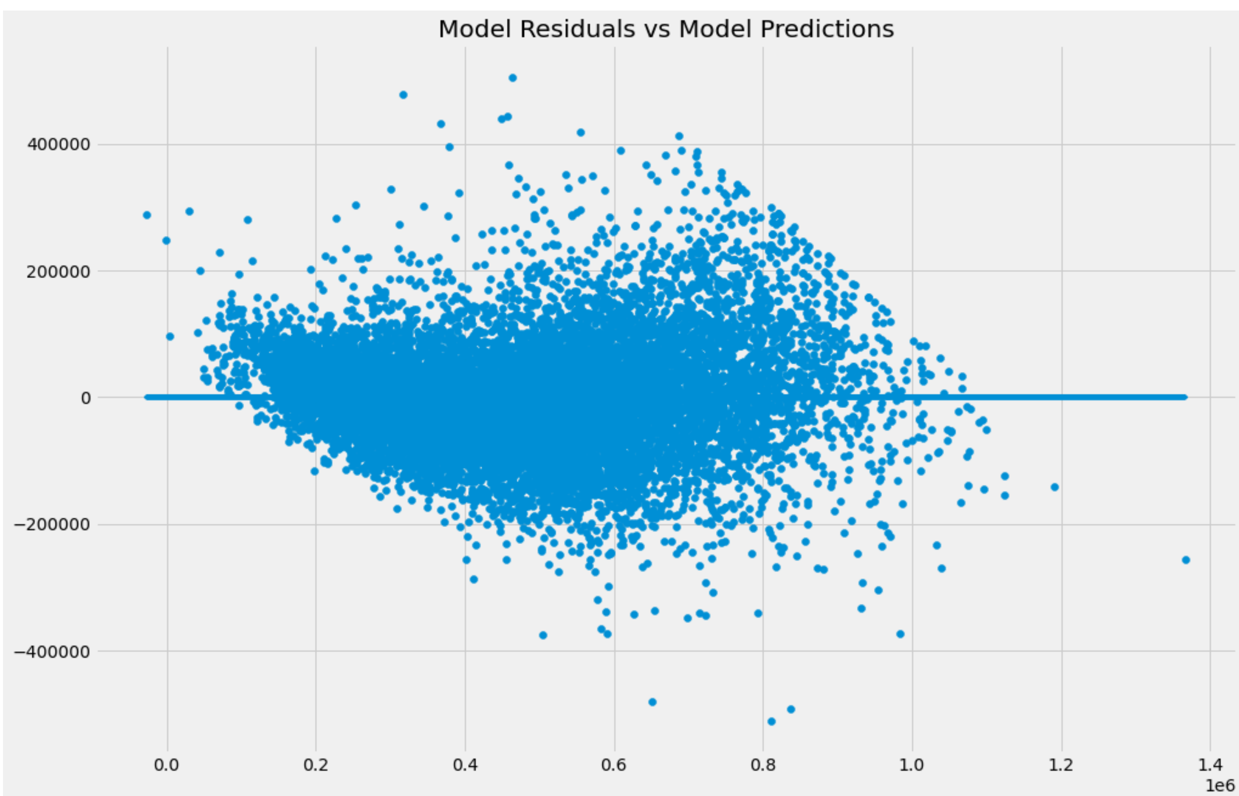
```
==============================================================================
=========

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


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

[('bedrooms', 1.8052258460138977),
 ('bathrooms', 3.1560787120331084),
 ('sqft_living', 4.012941576892276),
 ('sqft_lot', 2.090542150108155),
 ('floors', 2.6802014238503222),
 ('waterfront', 1.1840010763445867),
 ('view', 1.2877257008684329),
 ('condition', 1.3176616917253916),
 ('grade', 2.921989992063223),
 ('basement', 1.6502669654756037),
 ('renovated', 1.143112521172393),
 ('home_age', 3.3741918215413835),
 ('zipcode_98002', 1.6617258029584627),
 ('zipcode_98003', 1.9232201966915534),
 ('zipcode_98004', 1.4936048169063694),
 ('zipcode_98005', 1.3897811759551244),
 ('zipcode_98006', 2.2654970339341958),
 ('zipcode_98007', 1.4279809229577263),
 ('zipcode_98008', 1.8865269808841547),
 ('zipcode_98010', 1.1571404780548797),
 ('zipcode_98011', 1.5845967782458514),
 ('zipcode_98014', 1.1676093769575406),
 ('zipcode_98019', 1.433501415885143),
 ('zipcode_98022', 1.486673275547539),
 ('zipcode_98023', 2.5602279321654233),
 ('zipcode_98024', 1.0936091358571864),
 ('zipcode_98027', 1.8689220994735234),
 ('zipcode_98028', 1.8715755169655057),
 ('zipcode_98029', 2.1004495967023833),
 ('zipcode_98030', 1.774069262301884),
 ('zipcode_98031', 1.828320959299267),
 ('zipcode_98032', 1.4143690903184738),
 ('zipcode_98033', 2.1892604924059134),
 ('zipcode_98034', 2.657786186068738),
 ('zipcode_98038', 2.675831188551039),
 ('zipcode_98039', 1.0133141908844567),
 ('zipcode_98040', 1.5599276738774221),
 ('zipcode_98042', 2.5079527305462688),
```

```
 ('zipcode_98045', 1.5129120598504768),
 ('zipcode_98052', 2.6935348582190257),
 ('zipcode_98053', 1.869343274227229),
 ('zipcode_98055', 1.8630136567100917),
 ('zipcode_98056', 2.2730830459620006),
 ('zipcode_98058', 2.2733024689353742),
 ('zipcode_98059', 2.2797409357861045),
 ('zipcode_98065', 1.9709104371783528),
 ('zipcode_98070', 1.142972093759657),
 ('zipcode_98072', 1.4834098805528684),
 ('zipcode_98074', 2.2580453146287804),
 ('zipcode_98075', 1.949398271127825),
 ('zipcode_98077', 1.1648601472289926),
 ('zipcode_98092', 1.8816472473492962),
 ('zipcode_98102', 1.3303395753245912),
 ('zipcode_98103', 3.2451753616688555),
 ('zipcode_98105', 1.690717992420347),
 ('zipcode_98106', 2.1088086170398457),
 ('zipcode_98107', 2.0934963327977436),
 ('zipcode_98108', 1.6546304843976902),
 ('zipcode_98109', 1.376587366905492),
 ('zipcode_98112', 1.716337658147101),
 ('zipcode_98115', 2.9963631679233114),
 ('zipcode_98116', 2.2544209259137618),
 ('zipcode_98117', 3.0478617449740577),
 ('zipcode_98118', 2.7994216335510944),
 ('zipcode_98119', 1.6270675377665895),
 ('zipcode_98122', 2.0823196206571866),
 ('zipcode_98125', 2.316672245674781),
 ('zipcode_98126', 2.2817982723431034),
 ('zipcode_98133', 2.669507821832827),
 ('zipcode_98136', 1.9617170169617866),
 ('zipcode_98144', 2.2876414123383713),
 ('zipcode_98146', 1.9184778942422824),
 ('zipcode_98148', 1.1940191590201628),
 ('zipcode_98155', 2.3363328091193654),
 ('zipcode_98166', 1.7033817215030915),
 ('zipcode_98168', 1.8297601939446475),
 ('zipcode_98177', 1.7051424981098284),
 ('zipcode_98178', 1.8914042810811187),
 ('zipcode_98188', 1.419877315729031),
 ('zipcode_98198', 1.8790960306169342),
 ('zipcode_98199', 2.0322353245607885)]


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

QQPlot for Model Residuals



Homoscedasticity Test Results
==



In [453]:

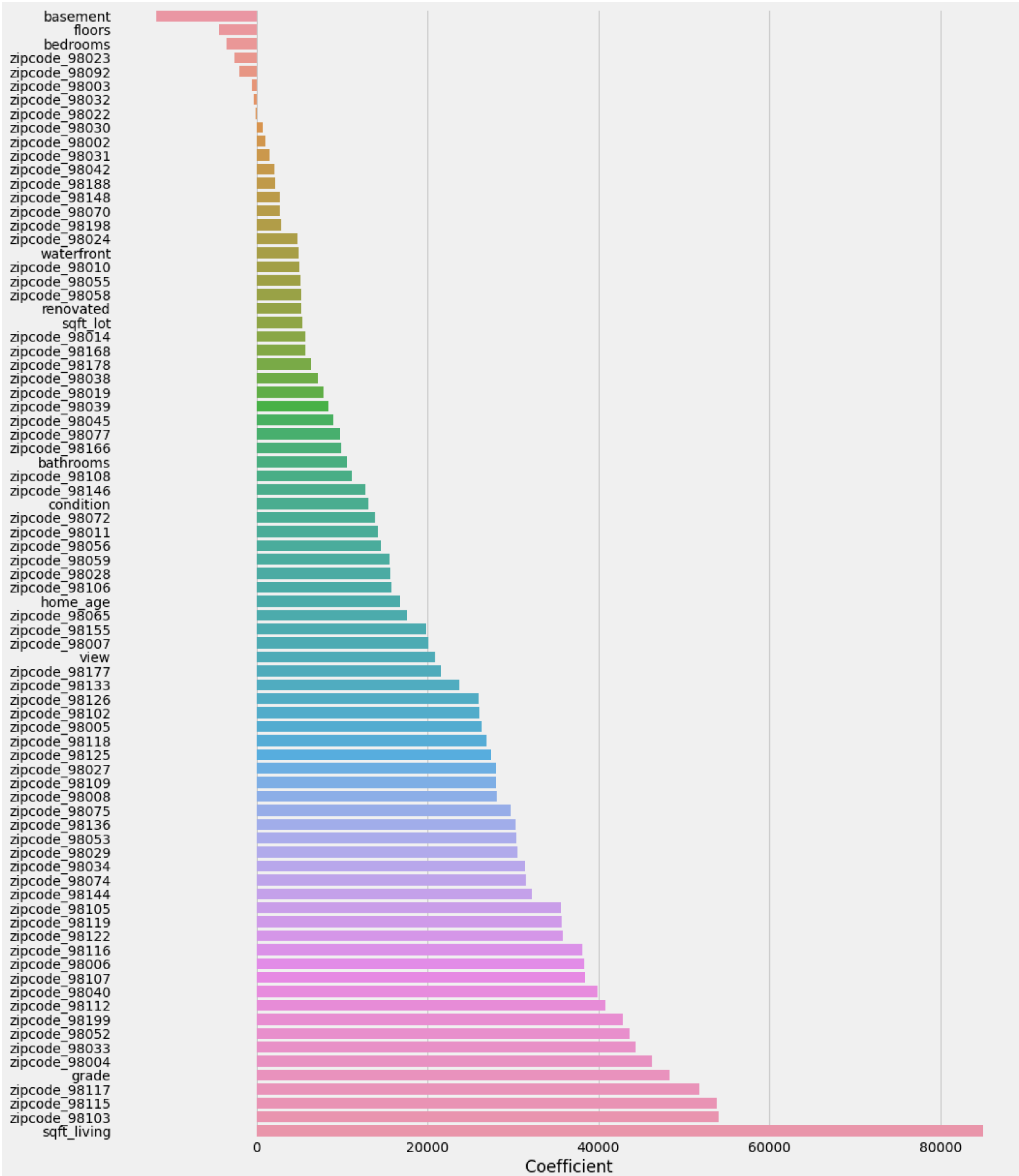
```
#create dataframe of feature coefficients
coefficients_m4_df = pd.DataFrame(model_4.params, columns=['Coeffi
coefficients_m4_df.drop('Intercept', inplace=True)
coefficients_m4_df = coefficients_m4_df.sort_values(by='Coefficier
```

In [3552]:

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

In [455]:

```
#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 0.836
- 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 .836 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
```