Table of Contents

King County Housing Analysis

1. Motivation

Property adviser in a real estate company. Recommendations will be provided for real estate investors/companies to predict the price and incearse the sale price, inorder to increase profits and avoid financial loses.

2. Objectives

1-What are the important key factors to predict the price of the property?

2-How home interior can affect the house value?

3-How geographic features affect property values?

4-What are important features to increase the value of the property?

3. Data loading

In [3]:
```
#Import Libaries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats
import scipy.stats as stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.outliers_influence import variance_inflation_fac

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, make_scorer
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn import metrics

```

In [4]:
```
#Load data and check correct format
### Convert ? to  NaN
df= pd.read_csv('kc_house_data.csv', na_values='?')
```

#Data abbreviations description

Column names

id - unique identified for a house date - Date house was sold price - Price is prediction target bedrooms - Number of Bedrooms/House bathrooms - Number of bathrooms/bedrooms sqft_living - Square footage of the home sqft_lot - Square footage of the lot floors - Total floors (levels) in house waterfront - House which has a view to a waterfront view - If the house has a view or not condition - How good the condition is (Overall) grade - overall grade given to the housing unit, based on King County grading system sqft_above - square footage of house apart from basement sqft_basement - square footage of the basement yr_built - Built Year yr_renovated - Year when

house was renovated zipcode - zipcode of the house lat - Latitude coordinate long - Longitude coordinate sqft_living15 - The square footage of interior housing living space for the nearest 15 neighbors sqft_lot15 - The square footage of the land lots of the nearest 15 neighbors

In [1]:
```
#features = df.iloc[:,:-1]
#target=df['price']
#features.head(10)
```

In [5]:
```
df.head(20)
```

Out[5]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	
5	7237550310	5/12/2014	1230000.0	4	4.50	5420	101930	1.0	
6	1321400060	6/27/2014	257500.0	3	2.25	1715	6819	2.0	
7	2008000270	1/15/2015	291850.0	3	1.50	1060	9711	1.0	
8	2414600126	4/15/2015	229500.0	3	1.00	1780	7470	1.0	
9	3793500160	3/12/2015	323000.0	3	2.50	1890	6560	2.0	
10	1736800520	4/3/2015	662500.0	3	2.50	3560	9796	1.0	
11	9212900260	5/27/2014	468000.0	2	1.00	1160	6000	1.0	
12	114101516	5/28/2014	310000.0	3	1.00	1430	19901	1.5	
13	6054650070	10/7/2014	400000.0	3	1.75	1370	9680	1.0	
14	1175000570	3/12/2015	530000.0	5	2.00	1810	4850	1.5	
15	9297300055	1/24/2015	650000.0	4	3.00	2950	5000	2.0	
16	1875500060	7/31/2014	395000.0	3	2.00	1890	14040	2.0	
17	6865200140	5/29/2014	485000.0	4	1.00	1600	4300	1.5	
18	16000397	12/5/2014	189000.0	2	1.00	1200	9850	1.0	
19	7983200060	4/24/2015	230000.0	3	1.00	1250	9774	1.0	

20 rows × 21 columns

Information about Grade column

Represents the construction quality of improvements. Grades run from grade 1 to 13. Generally defined as:

1-3 = Falls short of minimum building standards. Normally cabin or inferior structure

4 = Generally older, low quality construction. Does not meet code.

5 = Low construction costs and workmanship. Small, simple design.

6 = Lowest grade currently meeting building code. Low quality materials and simple designs.

7 = Average grade of construction and design. Commonly seen in plats and older sub-divisions.

8 = Just above average in construction and design. Usually better materials in both the exterior and interior finish work.

9 = Better architectural design with extra interior and exterior design and quality.

10 = Homes of this quality generally have high quality features. Finish work is better and more design quality is seen in the floor plans. Generally have a larger square footage.

11 = Custom design and higher quality finish work with added amenities of solid woods, bathroom fixtures and more luxurious options.

12 = Custom design and excellent builders. All materials are of the highest quality and all conveniences are present.

13 = Generally custom designed and built. Mansion level. Large amount of highest quality cabinet work, wood trim, marble, entry ways etc.

Information derived from:
https://blue.kingcounty.com/Assessor/eRealProperty/ResidentialGlossary.aspx?idx=viewall&Parcel=7960900070&AreaReport=http://www.KingCounty.gov/depts/Assessor/Reports/areports/2019/residential-northeast/033.aspx#BuildingGrade
(https://blue.kingcounty.com/Assessor/eRealProperty/ResidentialGlossary.aspx?idx=viewall&Parcel=7960900070&AreaReport=http://www.KingCounty.gov/depts/Assessor/Reports/areports/2019/residential-northeast/033.aspx#BuildingGrade)

In [353]:
```
1 df.shape
```

Out[353]: (21597, 21)

4. Data cleaning

In [272]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 1   date           21597 non-null  object 
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64  
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64  
 6   sqft_lot       21597 non-null  int64  
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  float64
 9   view           21534 non-null  float64
 10  condition      21597 non-null  int64  
 11  grade          21597 non-null  int64  
 12  sqft_above     21597 non-null  int64  
 13  sqft_basement  21143 non-null  float64
 14  yr_built       21597 non-null  int64  
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64  
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64  
 20  sqft_lot15     21597 non-null  int64  
dtypes: float64(9), int64(11), object(1)
memory usage: 3.5+ MB
```

Columns with missing values: waterfront, view, sqft_basement, yr_renovated

In [9]:
```
#Checking for missing values
df.isna().any()
```

Out[9]:
```
id               False
date             False
price            False
bedrooms         False
bathrooms        False
sqft_living      False
sqft_lot         False
floors           False
waterfront        True
view              True
condition        False
grade            False
sqft_above       False
sqft_basement     True
yr_built         False
yr_renovated      True
zipcode          False
lat              False
long             False
sqft_living15    False
sqft_lot15       False
dtype: bool
```

In [9]:
```
df.isna().sum()
```

Out[9]:
```
id                0
date              0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront     2376
view             63
condition         0
grade             0
sqft_above        0
sqft_basement   454
yr_built          0
yr_renovated   3842
zipcode           0
lat               0
long              0
sqft_living15     0
sqft_lot15        0
dtype: int64
```

In [11]:
```
1 df.sqft_basement.value_counts()
```

Out[11]:
```
0.0       12826
600.0       217
500.0       209
700.0       208
800.0       201
           ...
915.0         1
295.0         1
1281.0        1
2130.0        1
906.0         1
Name: sqft_basement, Length: 303, dtype: int64
```

In [10]:
```
1 df.sqft_basement.unique()
```

Out[10]:
```
array([   0.,  400.,  910., 1530.,   nan,  730., 1700.,  300.,  970.,
        760.,  720.,  700.,  820.,  780.,  790.,  330., 1620.,  360.,
        588., 1510.,  410.,  990.,  600.,  560.,  550., 1000., 1600.,
        500., 1040.,  880., 1010.,  240.,  265.,  290.,  800.,  540.,
        710.,  840.,  380.,  770.,  480.,  570., 1490.,  620., 1250.,
       1270.,  120.,  650.,  180., 1130.,  450., 1640., 1460., 1020.,
       1030.,  750.,  640., 1070.,  490., 1310.,  630., 2000.,  390.,
        430.,  850.,  210., 1430., 1950.,  440.,  220., 1160.,  860.,
        580., 2060., 1820., 1180.,  200., 1150., 1200.,  680.,  530.,
       1450., 1170., 1080.,  960.,  280.,  870., 1100.,  460., 1400.,
        660., 1220.,  900.,  420., 1580., 1380.,  475.,  690.,  270.,
        350.,  935., 1370.,  980., 1470.,  160.,  950.,   50.,  740.,
       1780., 1900.,  340.,  470.,  370.,  140., 1760.,  130.,  520.,
        890., 1110.,  150., 1720.,  810.,  190., 1290.,  670., 1800.,
       1120., 1810.,   60., 1050.,  940.,  310.,  930., 1390.,  610.,
       1830., 1300.,  510., 1330., 1590.,  920., 1320., 1420., 1240.,
       1960., 1560., 2020., 1190., 2110., 1280.,  250., 2390., 1230.,
        170.,  830., 1260., 1410., 1340.,  590., 1500., 1140.,  260.,
        100.,  320., 1480., 1060., 1284., 1670., 1350., 2570., 1090.,
        110., 2500.,   90., 1940., 1550., 2350., 2490., 1481., 1360.,
       1135., 1520., 1850., 1660., 2130., 2600., 1690.,  243., 1210.,
       1024., 1798., 1610., 1440., 1570., 1650.,  704., 1910., 1630.,
       2360., 1852., 2090., 2400., 1790., 2150.,  230.,   70., 1680.,
       2100., 3000., 1870., 1710., 2030.,  875., 1540., 2850., 2170.,
        506.,  906.,  145., 2040.,  784., 1750.,  374.,  518., 2720.,
       2730., 1840., 3480., 2160., 1920., 2330., 1860., 2050., 4820.,
       1913.,   80., 2010., 3260., 2200.,  415., 1730.,  652., 2196.,
       1930.,  515.,   40., 2080., 2580., 1548., 1740.,  235.,  861.,
       1890., 2220.,  792., 2070., 4130., 2250., 2240., 1990.,  768.,
       2550.,  435., 1008., 2300., 2610.,  666., 3500.,  172., 1816.,
       2190., 1245., 1525., 1880.,  862.,  946., 1281.,  414., 2180.,
        276., 1248.,  602.,  516.,  176.,  225., 1275.,  266.,  283.,
         65., 2310.,   10., 1770., 2120.,  295.,  207.,  915.,  556.,
        417.,  143.,  508., 2810.,   20.,  274.,  248.])
```

In [3]:
```
df[df.bedrooms ==33]
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	w
15856	2402100895	6/25/2014	640000.0	33	1.75	1620	6000	1.0	

1 rows × 21 columns

In [6]:
```
#Change 33 bedrooms house to 3 bedrooms house
df['bedrooms'].replace(to_replace=33, value=3, inplace= True)
```

In [4]:
```
##Replace 0 with difference between sqft_above and sqft_living if any?
#df['sqft_basement'].replace(to_replace='0', value= abs(df['sqft_living
```

In [355]:
```
df['sqft_basement'].nunique()
```

Out[355]: 303

4.a. Missing values

In [7]:
```
##Replace missing values
features1=['waterfront', 'view']
features2=['yr_renovated', 'sqft_basement']

for value in features1:
    mode=df[value].mode()[0]
    df[value].fillna(mode, inplace=True)

for value in features2:
    median=df[value].median()
    df[value].fillna(median, inplace=True)
```

In [8]:
```
1 df.isna().any()
```

Out[8]:
```
id               False
date             False
price            False
bedrooms         False
bathrooms        False
sqft_living      False
sqft_lot         False
floors           False
waterfront       False
view             False
condition        False
grade            False
sqft_above       False
sqft_basement    False
yr_built         False
yr_renovated     False
zipcode          False
lat              False
long             False
sqft_living15    False
sqft_lot15       False
dtype: bool
```

In [20]:
```
1 df.isna().sum()
```

Out[20]:
```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

In [9]:
```
df['view'].isnull()
```

Out[9]:
```
0        False
1        False
2        False
3        False
4        False
         ...
21592    False
21593    False
21594    False
21595    False
21596    False
Name: view, Length: 21597, dtype: bool
```

In [22]:
```
df['view'].value_counts()
```

Out[22]:
```
0.0    19485
2.0      957
3.0      508
1.0      330
4.0      317
Name: view, dtype: int64
```

In [32]:
```
df['view'].nunique()
```

Out[32]:
```
5
```

In [30]:
```
df['waterfront'].value_counts()
```

Out[30]:
```
0.0    21451
1.0      146
Name: waterfront, dtype: int64
```

In [31]:
```
df['waterfront'].nunique()
```

Out[31]:
```
2
```

In [9]:
```
#transform date dtype 'object' to date fromat
df['date']=pd.to_datetime(df['date'])
```

In [11]:
```
df.info()
```

```
 3   bedrooms       21597 non-null  int64
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64
 6   sqft_lot       21597 non-null  int64
 7   floors         21597 non-null  float64
 8   waterfront     21597 non-null  float64
 9   view           21597 non-null  float64
 10  condition      21597 non-null  int64
 11  grade          21597 non-null  int64
 12  sqft_above     21597 non-null  int64
 13  sqft_basement  21597 non-null  float64
 14  yr_built       21597 non-null  int64
 15  yr_renovated   21597 non-null  float64
 16  zipcode        21597 non-null  int64
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64
 20  sqft_lot15     21597 non-null  int64
dtypes: datetime64[ns](1), float64(9), int64(11)
memory usage: 3.5 MB
```

In [10]:
```
#Drop id column
df=df.drop(['id'], axis=1)
```

In [27]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   date           21597 non-null  datetime64[ns]
 1   price          21597 non-null  float64       
 2   bedrooms       21597 non-null  int64         
 3   bathrooms      21597 non-null  float64       
 4   sqft_living    21597 non-null  int64         
 5   sqft_lot       21597 non-null  int64         
 6   floors         21597 non-null  float64       
 7   waterfront     21597 non-null  float64       
 8   view           21597 non-null  float64       
 9   condition      21597 non-null  int64         
 10  grade          21597 non-null  int64         
 11  sqft_above     21597 non-null  int64         
 12  sqft_basement  21597 non-null  float64       
 13  yr_built       21597 non-null  int64         
 14  yr_renovated   21597 non-null  float64       
 15  zipcode        21597 non-null  int64         
 16  lat            21597 non-null  float64       
 17  long           21597 non-null  float64       
 18  sqft_living15  21597 non-null  int64         
 19  sqft_lot15     21597 non-null  int64         
dtypes: datetime64[ns](1), float64(9), int64(10)
memory usage: 3.3 MB
```

In [28]:
```
1 df.head(10)
```

Out[28]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	con
0	2014-10-13	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	
1	2014-12-09	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	
2	2015-02-25	180000.0	2	1.00	770	10000	1.0	0.0	0.0	
3	2014-12-09	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	
4	2015-02-18	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	
5	2014-05-12	1230000.0	4	4.50	5420	101930	1.0	0.0	0.0	
6	2014-06-27	257500.0	3	2.25	1715	6819	2.0	0.0	0.0	
7	2015-01-15	291850.0	3	1.50	1060	9711	1.0	0.0	0.0	
8	2015-04-15	229500.0	3	1.00	1780	7470	1.0	0.0	0.0	
9	2015-03-12	323000.0	3	2.50	1890	6560	2.0	0.0	0.0	

In [33]:
```
1 #Double check if anything is missing
2 df.isna().sum().any().any()
```

Out[33]:
```
False
```

In [11]:
```
1 df.describe().round(2)
```

Out[11]:

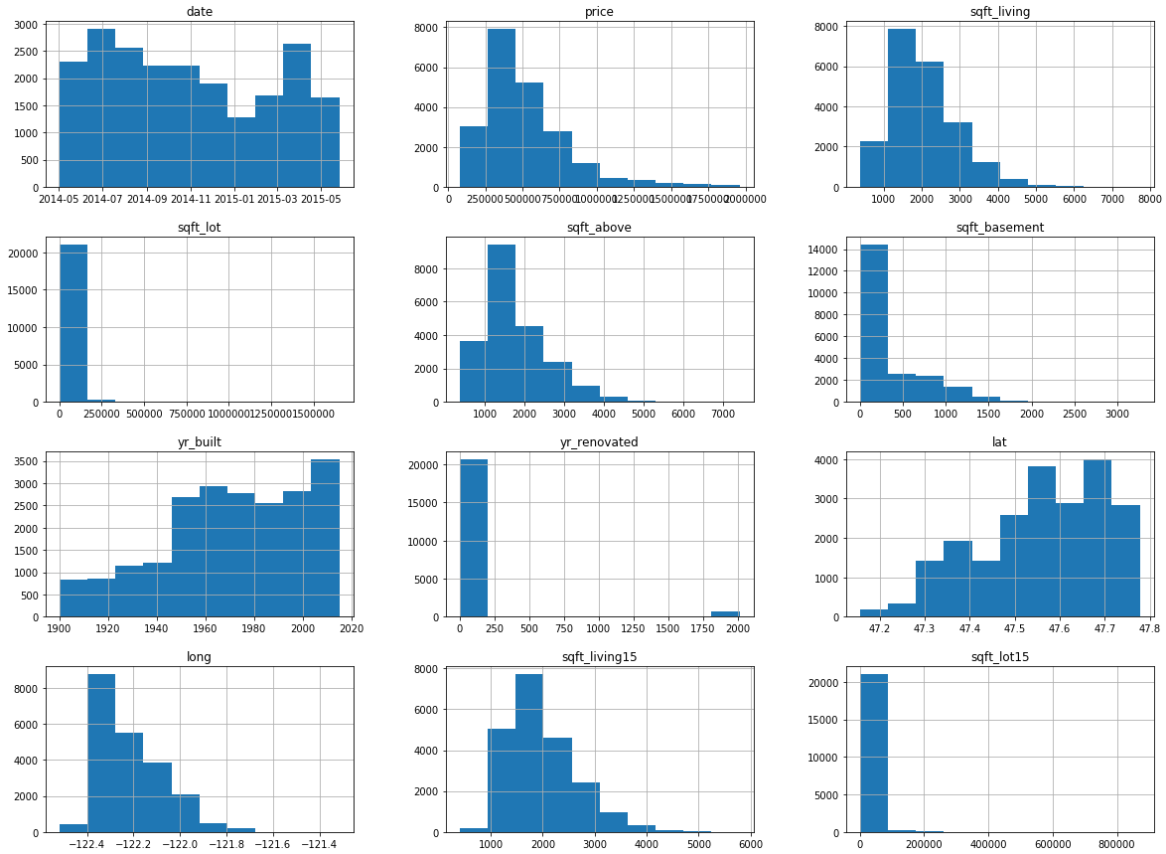
	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	vi
count	21597.00	21597.00	21597.00	21597.00	21597.00	21597.00	21597.00	21597
mean	540296.57	3.37	2.12	2080.32	15099.41	1.49	0.01	0
std	367368.14	0.90	0.77	918.11	41412.64	0.54	0.08	0
min	78000.00	1.00	0.50	370.00	520.00	1.00	0.00	0
25%	322000.00	3.00	1.75	1430.00	5040.00	1.00	0.00	0
50%	450000.00	3.00	2.25	1910.00	7618.00	1.50	0.00	0
75%	645000.00	4.00	2.50	2550.00	10685.00	2.00	0.00	0
max	7700000.00	11.00	8.00	13540.00	1651359.00	3.50	1.00	4

5. Data exploration

Check if the distributions of continuous variables are normal. We can check if there is a high skewness or/and kurtosis.

In [348]:

```
#Quick check for the distributions if it is normal or not
df.hist(figsize=(20, 15));
```

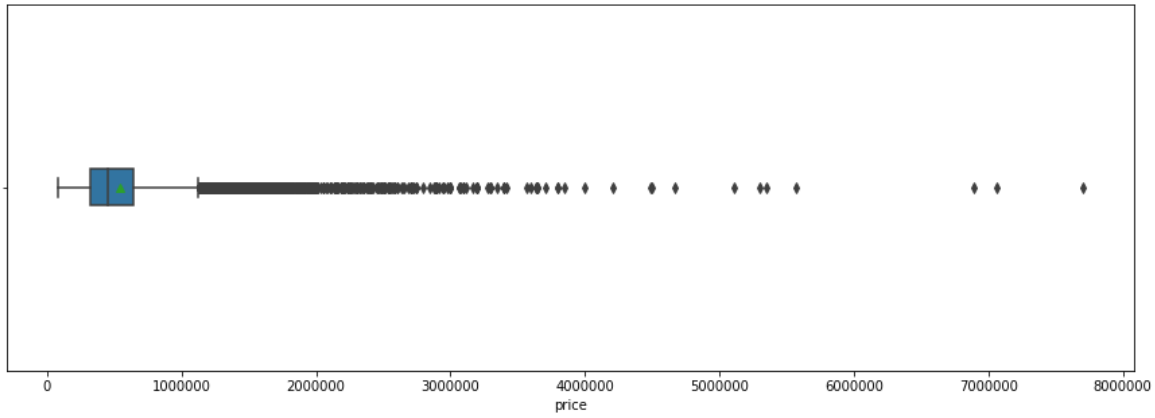


Most of the distributions are skewed and do not obey normal distribution.

5.a. Outliers

There are a few of vey expensive homes as outliers. I decided to get rid of 0.1 % of the data

In [13]:
```
#Check for outliers
fig, ax=plt.subplots(figsize=(15,5))
sns.boxplot(x='price', data=df, orient='h', width=0.1, showmeans=True,

plt.show()
```



It looks like outliers represent homes that are more than 6,000000$

In [7]:
```
#Get rid of outliers
outliers=df[(df['price'])>=7500000].index
df.drop(outliers, inplace=True)
```

In [11]:
```
for i in range(80,100):
    q = i/100
    print("{} percentile: {}".format(q, df.price.quantile(q=q)))
```

```
0.8 percentile: 700435.9999999998
0.81 percentile: 718000.0
0.82 percentile: 730000.72
0.83 percentile: 749950.0
0.84 percentile: 760003.2
0.85 percentile: 779721.9999999991
0.86 percentile: 799000.0
0.87 percentile: 815000.0
0.88 percentile: 836739.9999999998
0.89 percentile: 859967.6
0.9 percentile: 887000.0
0.91 percentile: 919993.6
0.92 percentile: 950000.0
0.93 percentile: 997964.0000000002
0.94 percentile: 1060000.0
0.95 percentile: 1160000.0
0.96 percentile: 1260000.0
0.97 percentile: 1390000.0
0.98 percentile: 1600000.0
0.99 percentile: 1970000.0
```

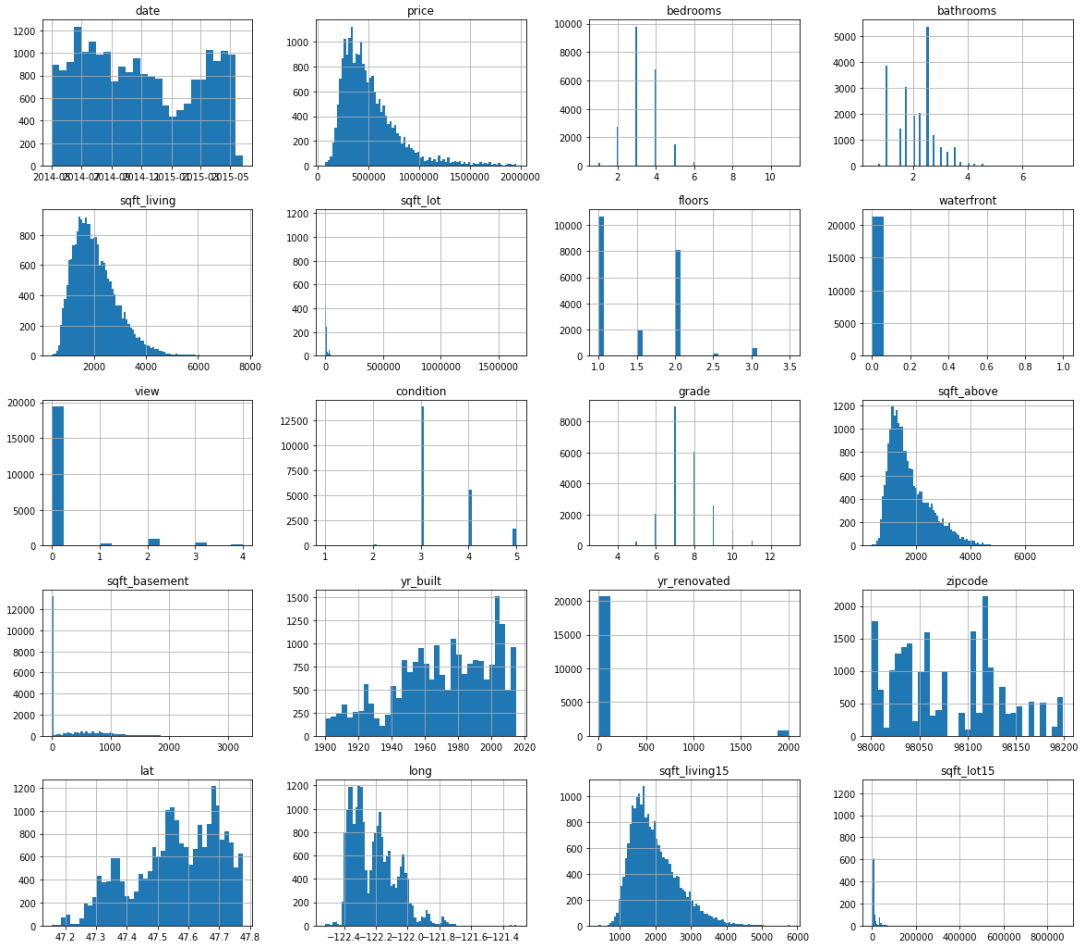
In [12]:

```
orig_tot = len(df)
df = df[df.price <1970000.0] # Subsetting to remove extreme outliers
print('Percent removed:', (orig_tot -len(df))/orig_tot)
```

```
Percent removed: 0.010047691809047552
```

In [19]:
```
#Plot again histograms for all variables in the dataset
df.hist(figsize=(20, 18), bins='auto')
```

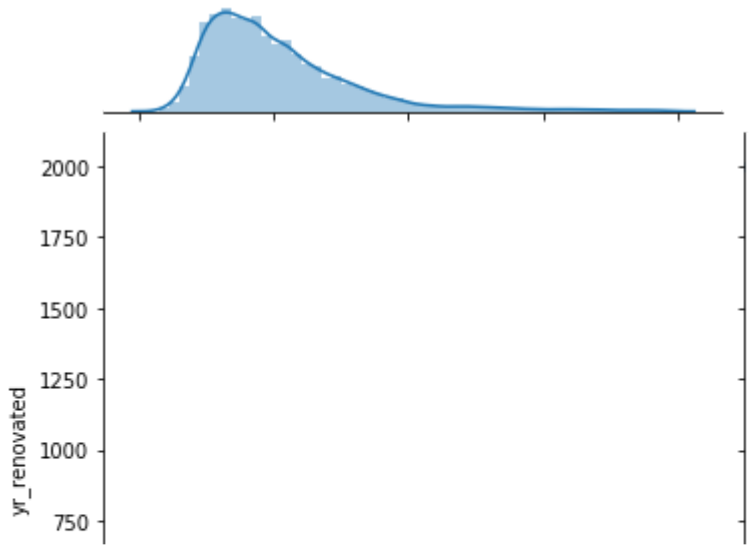
Out[19]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000219986FF688
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002199873C508
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021998AA7CC8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021998AE0708
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000021998B17F88
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021998B51908
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021998B8A2C8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021998BC0CC8
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000021998BCA8C8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021998C02A88
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021998C6F048
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021999C780C8
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000021999CB01C8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021999CE8308
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021999D21408
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021999D5B508
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000021999D935C8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021999DCC6C8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021999E047C8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021999E3C908
>]],
      dtype=object)
```



Most of these distributions are skewed, one way to ameloirate for skewed distribution is log transformations.

We can also distiguish continuous variables for categorials.

In [11]:
```
#check for linearity and plot some single variable regression plots of
sns.jointplot('price','sqft_living', data=df, kind='reg');
sns.jointplot('price','sqft_above', data=df, kind='reg');
sns.jointplot('price','sqft_lot', data=df, kind='reg');
sns.jointplot('price','sqft_basement', data=df, kind='reg');
sns.jointplot('price','lat', data=df, kind='reg');
sns.jointplot('price','long', data=df, kind='reg');
sns.jointplot('price','yr_built', data=df, kind='reg');
sns.jointplot('price','yr_renovated', data=df, kind='reg');
sns.jointplot('price','zipcode', data=df, kind='reg');
```



It looks like from initial investigations, sqft_living, sqft_above are linear.

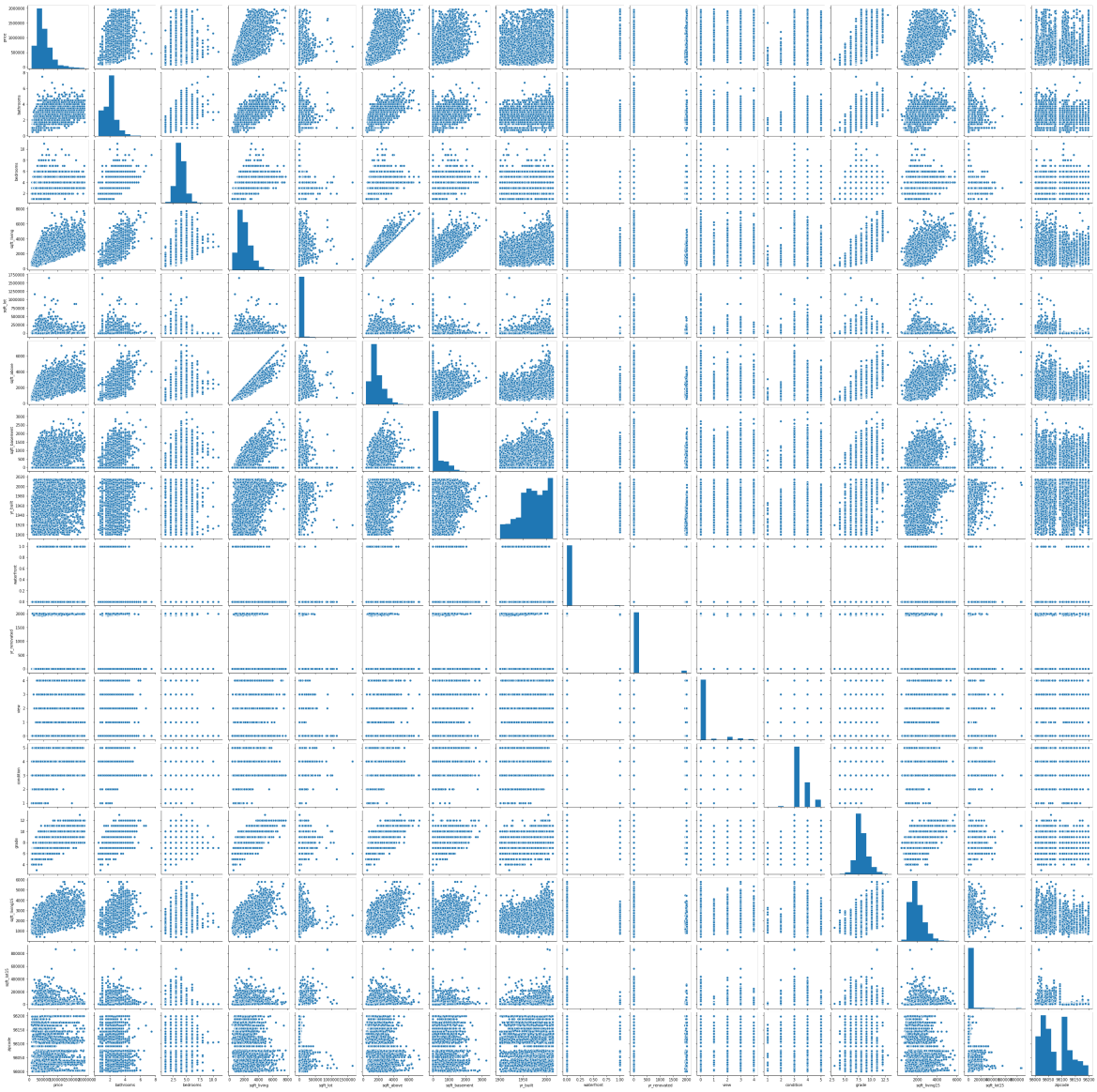
6. Feature Engineering

6.a. Identifying linearity

The linearity assumptions requires that there is a linear relationship between the response variable (Y) and predictor (X). Linear means that the change in Y by 1-unit change in X, is constant.

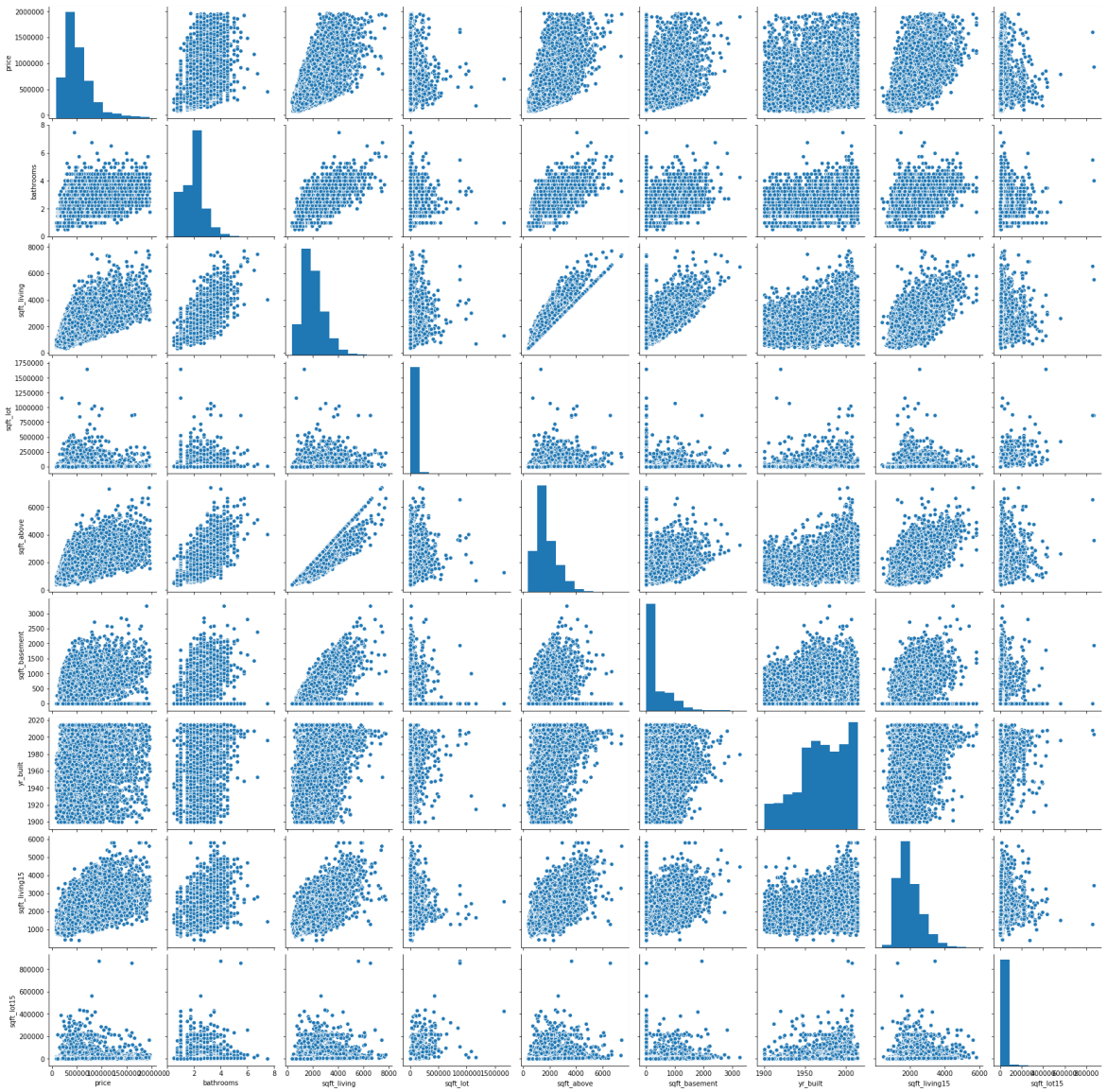
In [362]:

```
df_plot = df[['price', 'bathrooms','bedrooms' ,'sqft_living', 'sqft_lot
             'sqft_basement', 'yr_built','waterfront', "yr_renovated",
sns.pairplot(df_plot)
plt.show()
```



In [364]:

```
#Remove categorial variables for better visualization and linearity
df_plot = df[['price', 'bathrooms', 'sqft_living', 'sqft_lot', 'sqft_ab
              'sqft_basement', 'yr_built', 'sqft_living15', 'sqft_lot15
sns.pairplot(df_plot)
plt.show()
```



6.a.1. Linear relationship

When we check the linearity of parameters we see that 'price' has a linear relationship with other columns except 'yr_built'. There is a strong linear relationship between the price and sqft_living, sqft_above, sqft_basement, bathrooms, sqft_living15. We need to check the distributions one by one to avoid multicollinearity. We can eliminate these columns by checking their p values.

6.b. Identifying multicollinearity

Multicollinearity is the occurrence of high intercorrelations among two or more independent variables in a multiple regression model. Multicollinearity is a problem because it undermines the statistical significance of an independent variable. Other things being equal, the larger the standard error of a regression coefficient, the less likely it is that this coefficient will be statistically significant.

In [367]:
```
#Identifying multicollinearity
features = df.drop(['price'], axis=1)
target=df['price']
features.head(10)

```

Out[367]:

	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grad
0	2014-10-13	3	1.00	1180	5650	1.0	0.0	0.0	3	
1	2014-12-09	3	2.25	2570	7242	2.0	0.0	0.0	3	
2	2015-02-25	2	1.00	770	10000	1.0	0.0	0.0	3	
3	2014-12-09	4	3.00	1960	5000	1.0	0.0	0.0	5	
4	2015-02-18	3	2.00	1680	8080	1.0	0.0	0.0	3	
5	2014-05-12	4	4.50	5420	101930	1.0	0.0	0.0	3	1
6	2014-06-27	3	2.25	1715	6819	2.0	0.0	0.0	3	
7	2015-01-15	3	1.50	1060	9711	1.0	0.0	0.0	3	
8	2015-04-15	3	1.00	1780	7470	1.0	0.0	0.0	3	
9	2015-03-12	3	2.50	1890	6560	2.0	0.0	0.0	3	

7.b.1. Detection of correlated parameters

In [368]:

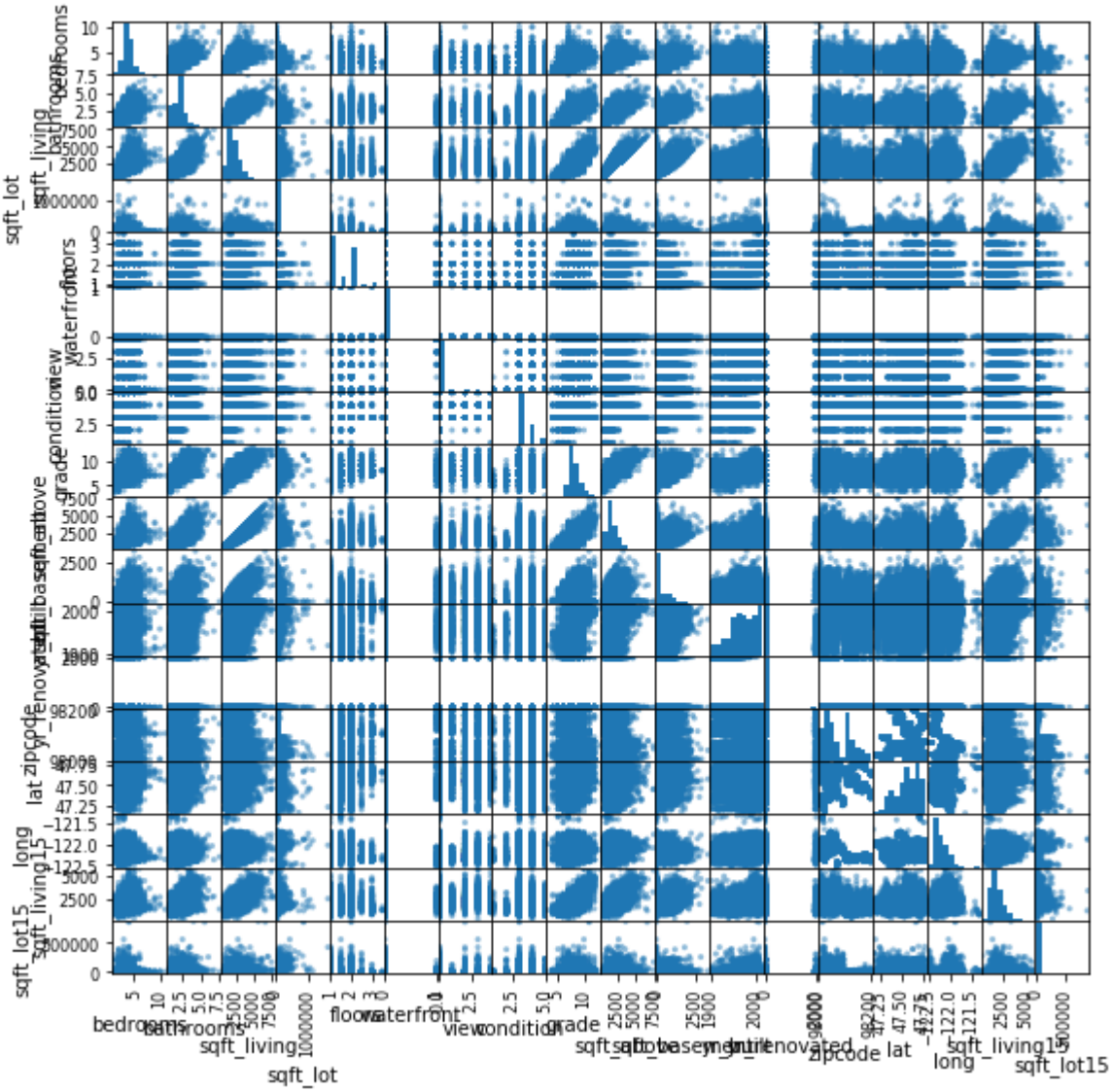
```
#Select only significant correlations greater than 0.75
abs(features.corr()) > 0.75
```

Out[368]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition
bedrooms	True	False	False	False	False	False	False	False
bathrooms	False	True	False	False	False	False	False	False
sqft_living	False	False	True	False	False	False	False	False
sqft_lot	False	False	False	True	False	False	False	False
floors	False	False	False	False	True	False	False	False
waterfront	False	False	False	False	False	True	False	False
view	False	False	False	False	False	False	True	False
condition	False	False	False	False	False	False	False	True
grade	False	False	False	False	False	False	False	False
sqft_above	False	False	True	False	False	False	False	False
sqft_basement	False	False	False	False	False	False	False	False
yr_built	False	False	False	False	False	False	False	False
yr_renovated	False	False	False	False	False	False	False	False
zipcode	False	False	False	False	False	False	False	False
lat	False	False	False	False	False	False	False	False
long	False	False	False	False	False	False	False	False
sqft_living15	False	False	True	False	False	False	False	False
sqft_lot15	False	False	False	False	False	False	False	False

In [369]:

```
pd.plotting.scatter_matrix(features,figsize  = [9, 9]);
plt.show()
```



In [16]:
```
1 features.corr()
```

Out[16]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
bedrooms	1.000000	0.520266	0.596904	0.033559	0.176499	-0.023221	0.063171
bathrooms	0.520266	1.000000	0.742046	0.085805	0.505652	0.016053	0.148946
sqft_living	0.596904	0.742046	1.000000	0.175440	0.353499	0.032656	0.236321
sqft_lot	0.033559	0.085805	0.175440	1.000000	-0.007385	0.020010	0.074192
floors	0.176499	0.505652	0.353499	-0.007385	1.000000	0.006936	0.014150
waterfront	-0.023221	0.016053	0.032656	0.020010	0.006936	1.000000	0.334788
view	0.063171	0.148946	0.236321	0.074192	0.014150	0.334788	1.000000
condition	0.020488	-0.133265	-0.067549	-0.007640	-0.268944	0.015470	0.040966
grade	0.354474	0.649476	0.748010	0.111272	0.459541	0.029890	0.209485
sqft_above	0.485556	0.668753	0.867559	0.184498	0.528760	0.014983	0.122372
sqft_basement	0.295411	0.250454	0.399329	0.011542	-0.261020	0.037216	0.243743
yr_built	0.164972	0.523841	0.338997	0.051817	0.496309	-0.036316	-0.059422
yr_renovated	0.015194	0.040291	0.038531	0.004892	0.001986	0.073972	0.085053
zipcode	-0.158676	-0.205942	-0.204762	-0.131171	-0.060486	0.043883	0.093153
lat	-0.016904	0.015291	0.041677	-0.087774	0.046495	-0.024307	-0.004155
long	0.141103	0.236582	0.263107	0.229421	0.129084	-0.049498	-0.077737
sqft_living15	0.396702	0.555616	0.754853	0.146153	0.276571	0.041711	0.247899
sqft_lot15	0.031123	0.085040	0.185815	0.713360	-0.012970	0.028138	0.070479

In [370]:
```
#include stack and zip to create a more robust solution that will retur
#the variable pairs from the correlation matrix that have correlations
df=features.corr().abs().stack().reset_index().sort_values(0, ascending

# zip the variable name columns (Which were only named level_0 and leve
df['pairs'] = list(zip(df.level_0, df.level_1))

# set index to pairs
df.set_index(['pairs'], inplace = True)

#d rop level columns
df.drop(columns=['level_1', 'level_0'], inplace = True)

# rename correlation column as cc rather than 0
df.columns = ['cc']

# drop duplicates. This could be dangerous if you have variables perfec
# for the sake of exercise, kept it in.
df.drop_duplicates(inplace=True)
```

In [371]:
```
df[(df.cc>.75) & (df.cc <1)]
```

Out[371]:

	cc
pairs	
(sqft_living, sqft_above)	0.867559
(sqft_living15, sqft_living)	0.754853

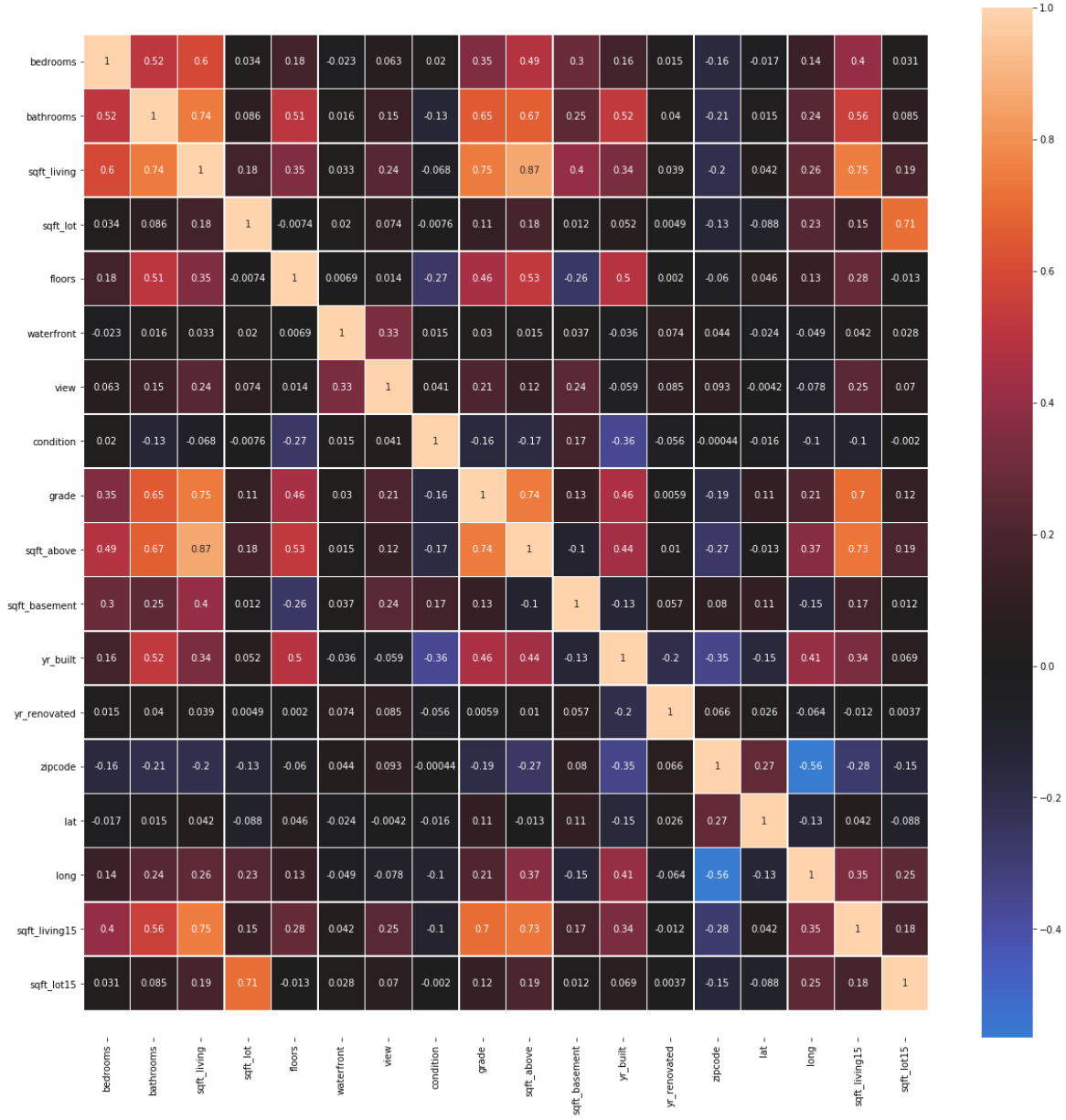
6.b.1.1. Data decision to address colinearity

There are three sets of variales that are highly correlated. we can remove sqft_living15 and sqft_above from data set df.drop(columns=['sqft_living15',"sqft_above"], inplace=True). For now I will keep all the features and test the performance of the model.

6.b.2 Heat map

In [374]:

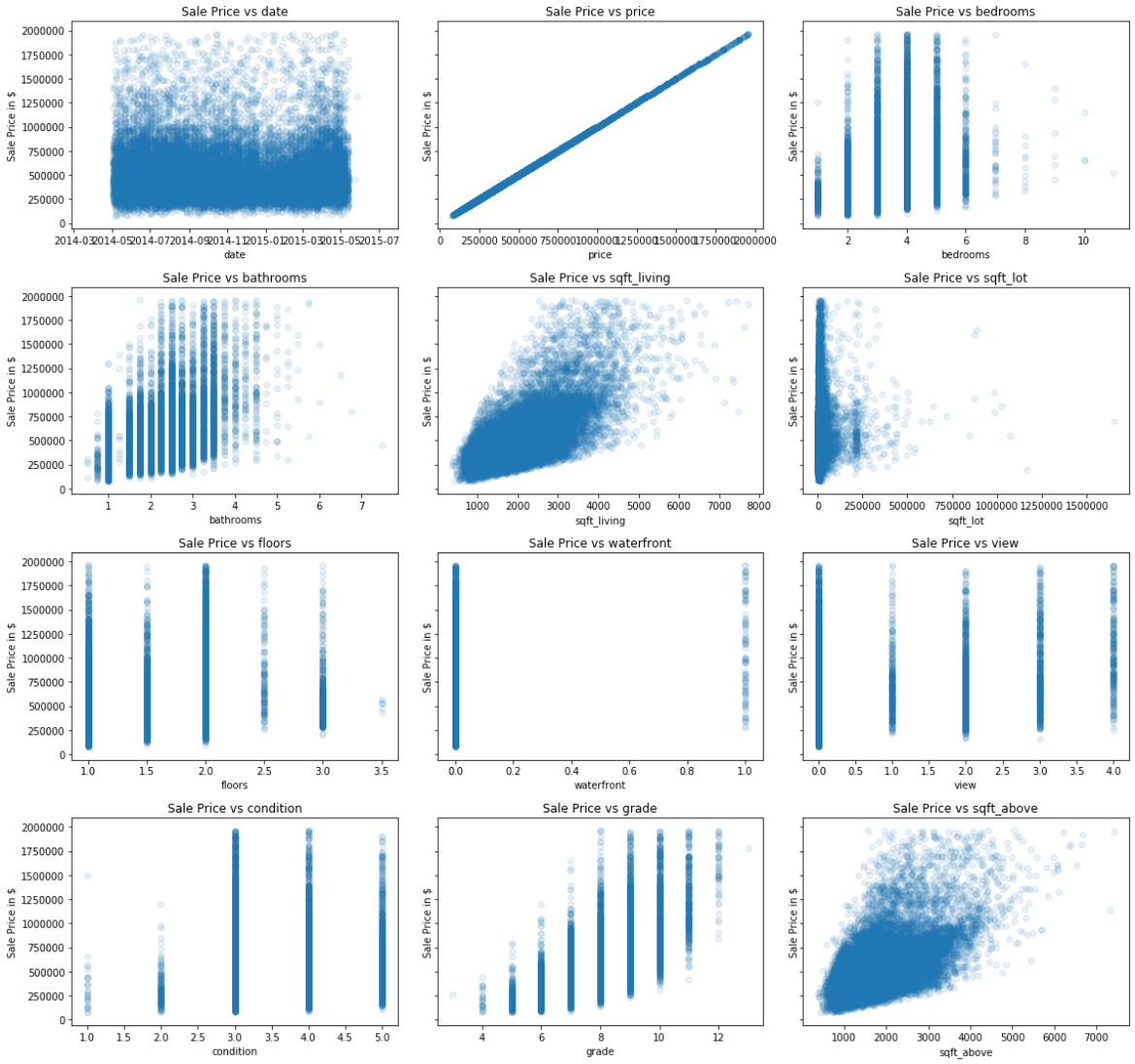
```
#Heat map
plt.figure(figsize=(20, 20))
ax = sns.heatmap(features.corr(), center=0, linewidths=.5, annot=True)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.show()
```



6.c. Identify categorial and continuous variables

In [384]:

```
fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(16,15), sharey=True

for ax, column in zip(axes.flatten(), df.columns):
    ax.scatter(df[column], df['price'] , label=column, alpha=.1)
    ax.set_title(f'Sale Price vs {column}')
    ax.set_xlabel(column)
    ax.set_ylabel('Sale Price in $')

fig.tight_layout()
```



6.d. Log transformation and data normalization

Log transformation is an effective technique we can use to improve the performance of linear regression models. We make the distributions look muck normal and improve our models' predictive performance.

In [13]:

```
#Transformation non-normal features
#take out sqft_basement,yr_renovated, these two features after log and
#or infinity values I tried diffrent normalization methods and still sa
#I decided to remove sqft_basement,yr_renovated sicnce the correlation

continuous = ['price', 'sqft_living', 'sqft_lot', 'sqft_above',
              'yr_built', 'sqft_living15', 'sqft_lot15' ]


categoricals= ['floors', 'condition','bedrooms', 'view', 'waterfront',

#for feat in non_normal:
    #df[feat]=df[feat].map(lambda x: np.log(x))


kc_cont = df[continuous]

#log features
log_names = [f'{column}_log' for column in kc_cont.columns]

kc_log = np.log(kc_cont)
kc_log.columns = log_names



# normalize (subract mean and divide by std)

def normalize(feature):
    return (feature - feature.mean()) / feature.std()

kc_log_norm =kc_log.apply(normalize)


```

6.e. One-Hot Encoding Categorical Columns

Categorical columns: Bedrooms, bathrooms,floors, waterfront, view, grade, condition and zipcode. We change their type to 'category' and make new columns by using pandas get_dummies() function.

In [14]:
```
#Categorial features
df['bedrooms'] = df['bedrooms'].astype('category')
df['floors'] = df['floors'].astype('category')
df['waterfront'] = df['waterfront'].astype('category')
df['view'] = df['view'].astype('category')
df['condition'] = df['condition'].astype('category')
df['grade'] = df['grade'].astype('category')
df['zipcode']=df['zipcode'].astype('category')
df['bathrooms']=df['bathrooms'].astype('category')

df_dummy = pd.get_dummies(df[categoricals], prefix=categoricals, drop_f
```

In [42]:
```
#Data normalization
#def normalize(feature):
   # return (feature - feature.mean())/feature.std()


#for feat in ['bathrooms', 'sqft_living', 'sqft_lot', 'sqft_above', 'sq
            # 'yr_built', 'yr_renovated', 'sqft_living15', 'sqft_lot15'
    #df[feat] = norm_feat(df[feat])


```

In [15]:
```
#Combine categorial and continuous features
preprocessed = pd.concat([kc_log_norm, df_dummy], axis=1)
preprocessed.head()
```

Out[15]:

	price_log	sqft_living_log	sqft_lot_log	sqft_above_log	yr_built_log	sqft_living15_log	sqft_
0	-1.444792	-1.127694	-0.381175	-0.746778	-0.538117	-1.031533	
1	0.330701	0.747609	-0.105607	0.705282	-0.675344	-0.312979	
2	-1.864331	-2.156133	0.252602	-1.764245	-1.296368	1.160623	
3	0.562685	0.094808	-0.516846	-1.024990	-0.196274	-0.985659	
4	0.223550	-0.276573	0.015941	0.095264	0.549696	-0.117724	

5 rows × 135 columns

In [43]:
```
# Dealing with Categorial columns
#Make sure Python recognizes these strings as category.

#df['bedrooms'] = df['bedrooms'].astype('category')
#df['floors'] = df['floors'].astype('category')
#df['waterfront'] = df['waterfront'].astype('category')
#df['view'] = df['view'].astype('category')
#df['condition'] = df['condition'].astype('category')
#df['grade'] = df['grade'].astype('category')
#df['bathrooms']=df['bathrooms'].astype('category')
#df['zipcode'] = df['zipcode'].astype('category')
#df['date'] = df['date'].astype('category')
```

In [12]:
```
#Create dummy variables for the categorial features
#feats = [ 'bedrooms', 'bathrooms',  'floors',
        # 'waterfront', 'view', 'condition', 'grade']
#df_feats = df[feats]
#df_feats_zipcode = df_feats.copy()
#df_feats = pd.get_dummies(df_feats, drop_first=True)
```

In [44]:
```
#floor_dummies = pd.get_dummies(df['floors'], prefix='floors', drop_fir
#condition_dummies = pd.get_dummies(df['condition'], prefix='con', drop
#grade_dummies = pd.get_dummies(df['grade'], prefix='grade', drop_first
#bedrooms_dummies = pd.get_dummies(df['bedrooms'], prefix='bed', drop_f
#bathrooms_dummies = pd.get_dummies(df['bathrooms'], prefix='bath', dro
#waterfront_dummies = pd.get_dummies(df['waterfront'], prefix='water',
#view_dummies = pd.get_dummies(df['view'], prefix='view', drop_first=Tr
```

In [45]:
```
#df = df.drop(['floors', 'condition','bedrooms', 'view', 'waterfront',
```

In [14]:
```
#df = pd.concat([df, view_dummies, bathrooms_dummies, waterfront_dummie
                         #floor_dummies,
                         #condition_dummies,
                        # grade_dummies,
                         #bedrooms_dummies
                         #], axis=1)
#df.head()
```

In [32]:
```
1 df.dtypes
```

Out[32]:
```
date           datetime64[ns]
price                 float64
sqft_living           float64
sqft_lot              float64
sqft_above            float64
                       ...
bed_7                   uint8
bed_8                   uint8
bed_9                   uint8
bed_10                  uint8
bed_11                  uint8
Length: 75, dtype: object
```

In [388]:
```
1 df.keys()
```

Out[388]:
```
Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'lon
g',
       'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

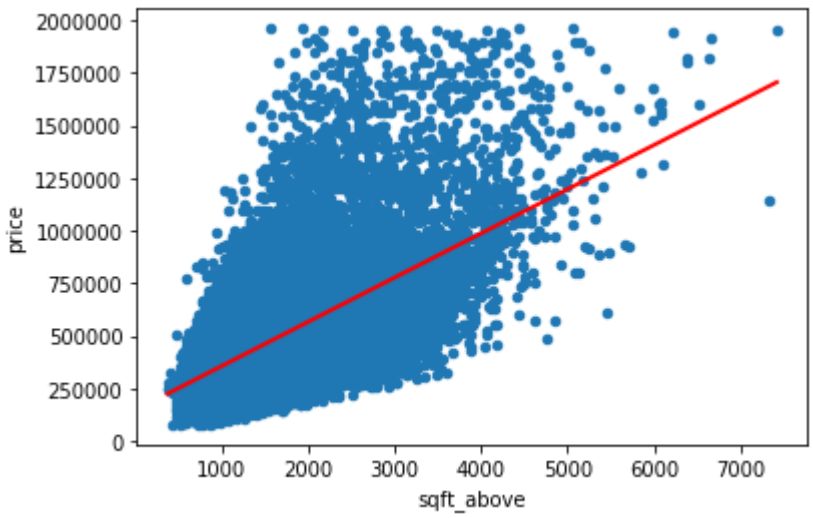
7. Linear regression

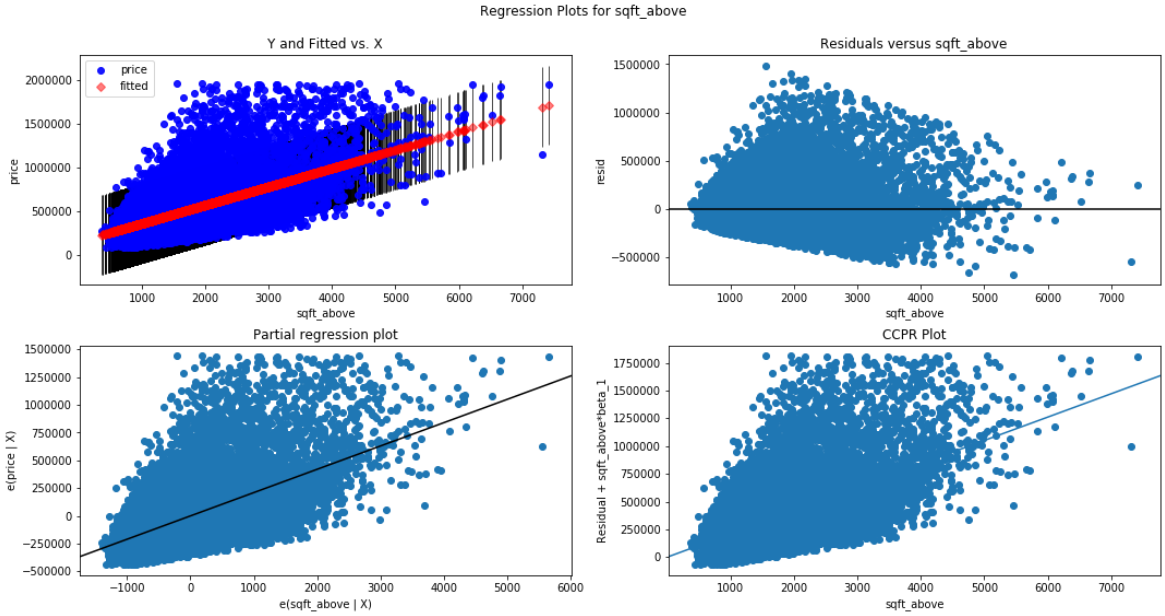
Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. We will start modelling our data using two variables: the target (price) and other features to check if there is a linear relationship. The weakness of the linear regression model Weaknessesis when it performs poorly when there are non-linear relationships.

In [389]:

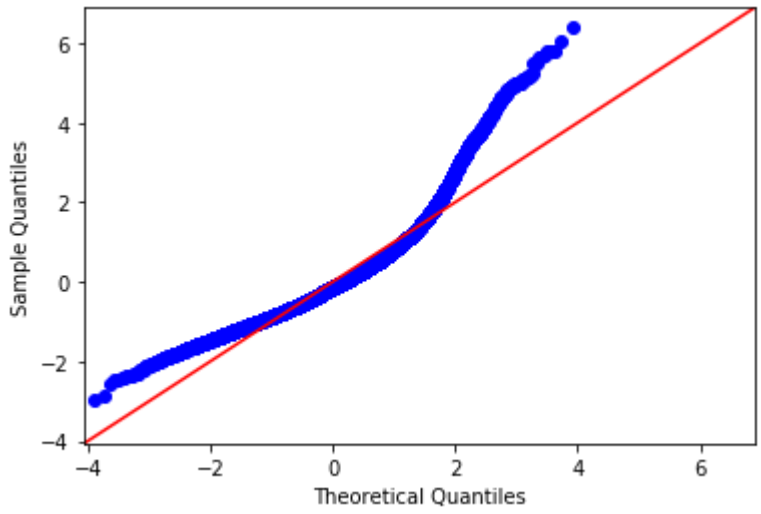
```
#Single linear regression
import statsmodels.formula.api as smf
f = 'price~sqft_above'
model = smf.ols(formula=f, data=df).fit()
print ('R-Squared:',model.rsquared)
print (model.params)
X_new = pd.DataFrame({'sqft_above': [df.sqft_above.min(), df.sqft_above
preds = model.predict(X_new)
df.plot(kind='scatter', x='sqft_above', y='price');
plt.plot(X_new, preds, c='red', linewidth=2);
plt.show()
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_above", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=Tru
fig.show()
```

```
R-Squared: 0.34062439602857164
Intercept     147248.157357
sqft_above       210.060588
dtype: float64
```





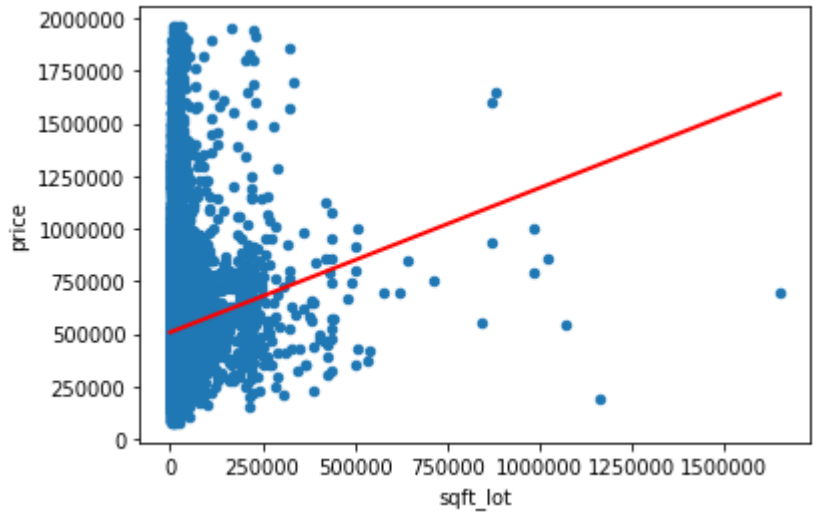

```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\ipykernel_launcher.py:1
8: UserWarning: Matplotlib is currently using module://ipykernel.pylab.back
end_inline, which is a non-GUI backend, so cannot show the figure.
```

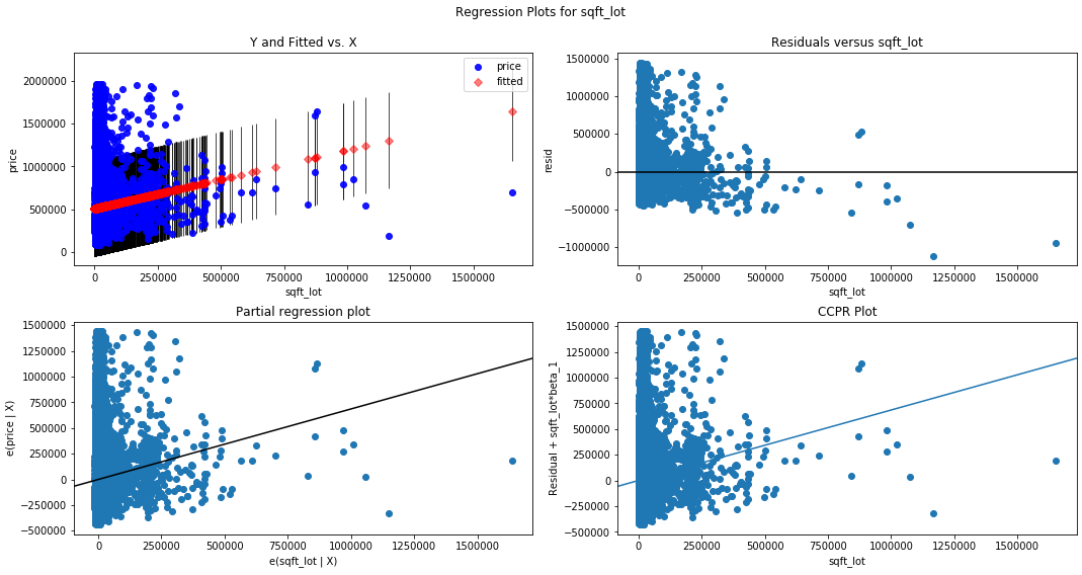


In [14]:

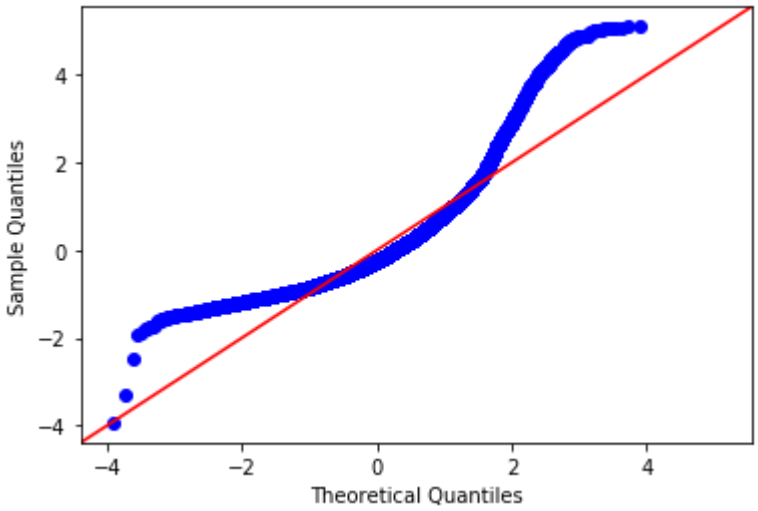
```
f = 'price~sqft_lot'
model = smf.ols(formula=f, data=df).fit()
print ('R-Squared:',model.rsquared)
print (model.params)
X_new = pd.DataFrame({'sqft_lot': [df.sqft_lot.min(), df.sqft_lot.max()
preds = model.predict(X_new)
df.plot(kind='scatter', x='sqft_lot', y='price');
plt.plot(X_new, preds, c='red', linewidth=2);
plt.show()
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_lot", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=Tru
fig.show()
```

```
R-Squared: 0.009728480794731209
Intercept    508026.025566
sqft_lot          0.685566
dtype: float64
```





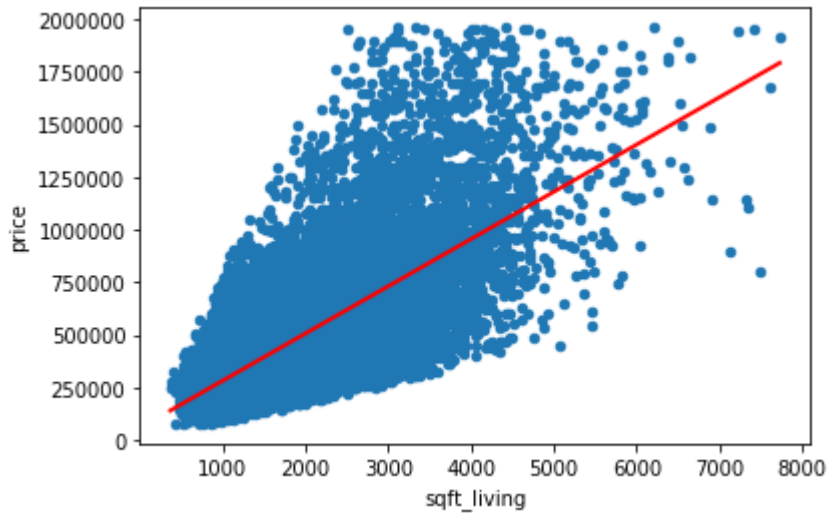
```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\ipykernel_launcher.py:1
6: UserWarning: Matplotlib is currently using module://ipykernel.pylab.back
end_inline, which is a non-GUI backend, so cannot show the figure.
  app.launch_new_instance()
```

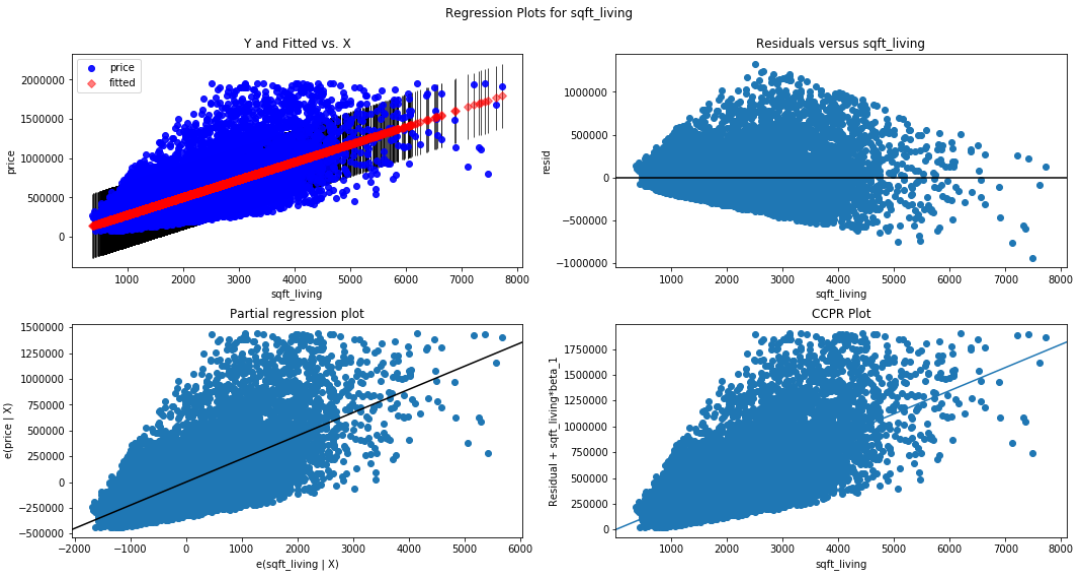


In [390]:

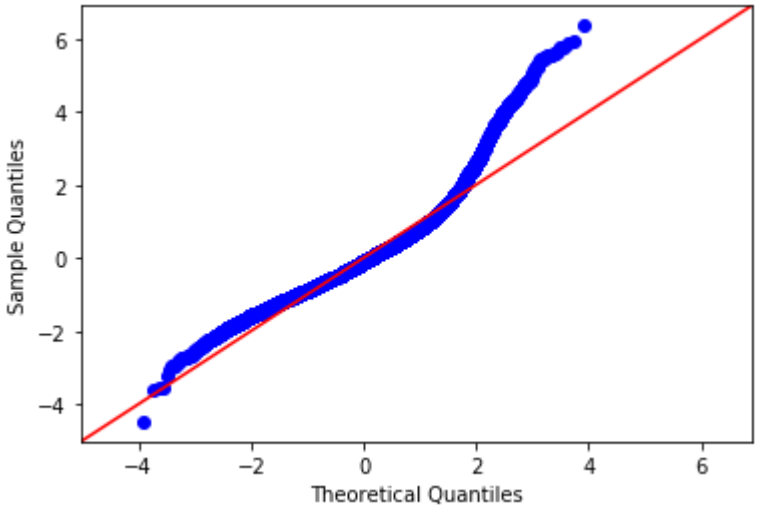
```
f = 'price~sqft_living'
model = smf.ols(formula=f, data=df).fit()
print ('R-Squared:',model.rsquared)
print (model.params)
X_new = pd.DataFrame({'sqft_living': [df.sqft_living.min(), df.sqft_liv
preds = model.predict(X_new)
df.plot(kind='scatter', x='sqft_living', y='price');
plt.plot(X_new, preds, c='red', linewidth=2);
plt.show()
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_living", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=Tru
fig.show()
```

```
R-Squared: 0.46050140020301855
Intercept      57988.127483
sqft_living      224.444098
dtype: float64
```





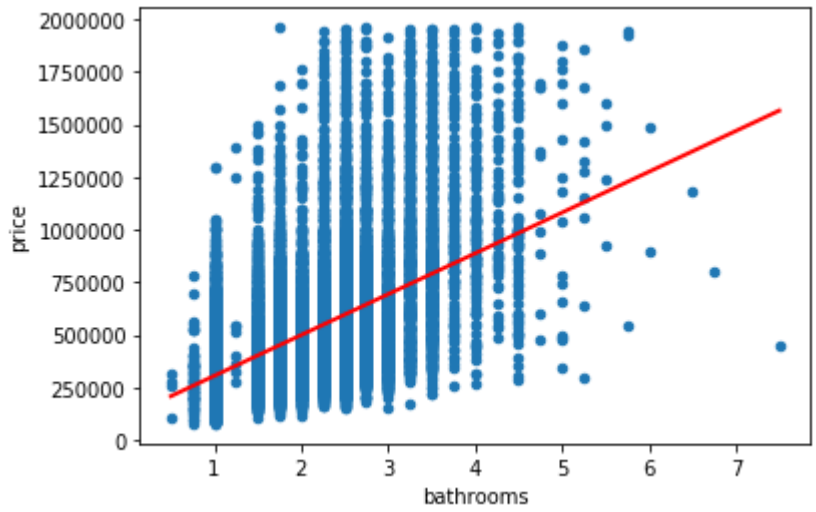
```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\ipykernel_launcher.py:1
6: UserWarning: Matplotlib is currently using module://ipykernel.pylab.back
end_inline, which is a non-GUI backend, so cannot show the figure.
  app.launch_new_instance()
```

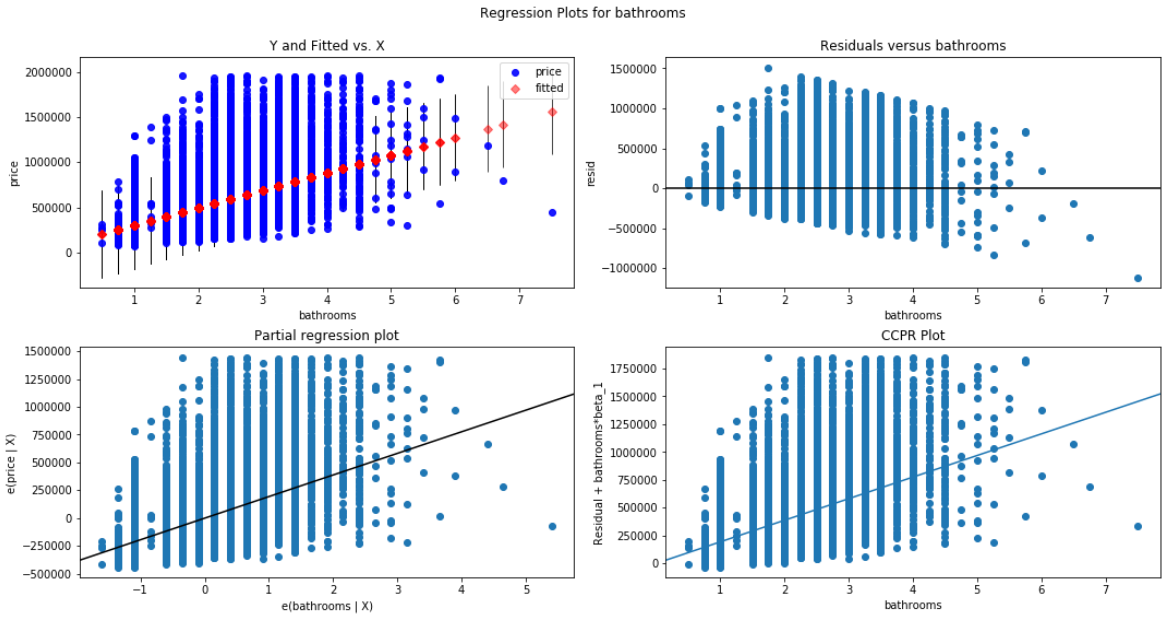


In [392]:

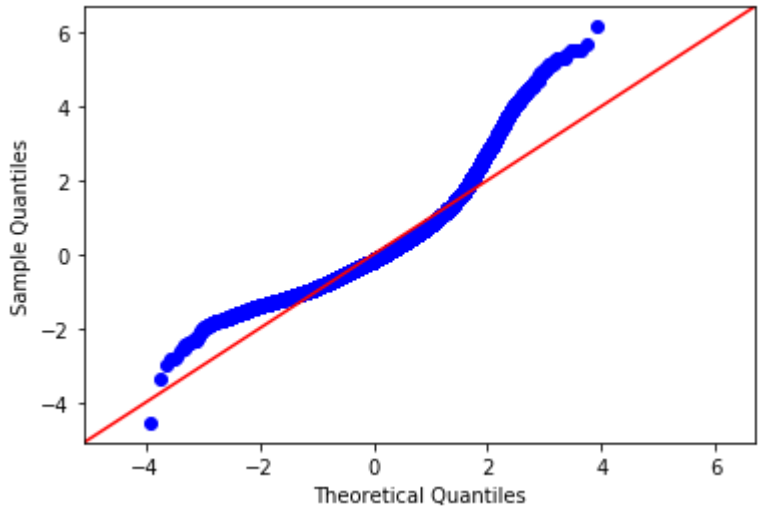
```
f = 'price~bathrooms'
model = smf.ols(formula=f, data=df).fit()
print ('R-Squared:',model.rsquared)
print (model.params)
X_new = pd.DataFrame({'bathrooms': [df.bathrooms.min(), df.bathrooms.ma
preds = model.predict(X_new)
df.plot(kind='scatter', x='bathrooms', y='price');
plt.plot(X_new, preds, c='red', linewidth=2);
plt.show()
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "bathrooms", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=Tru
fig.show()
```

```
R-Squared: 0.25765215401678143
Intercept    111508.817838
bathrooms    193845.888862
dtype: float64
```





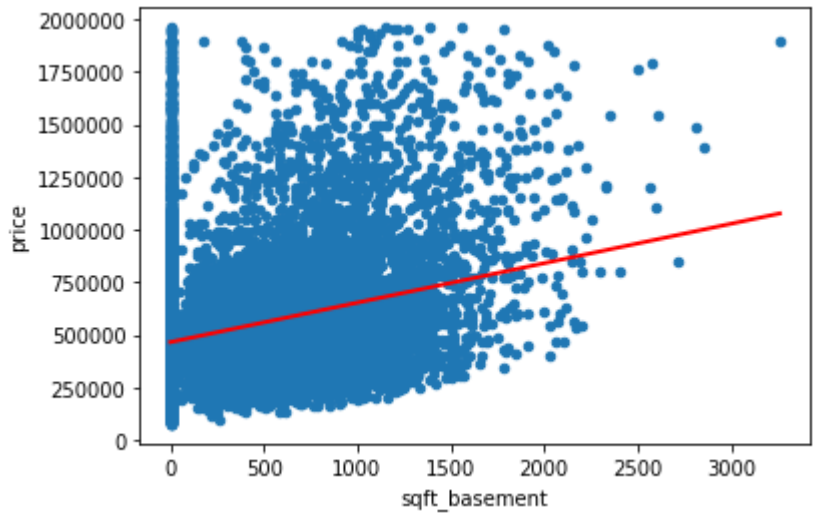

```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\ipykernel_launche
r.py:16: UserWarning: Matplotlib is currently using module://ipykerne
l.pylab.backend_inline, which is a non-GUI backend, so cannot show the
figure.
  app.launch_new_instance()
```

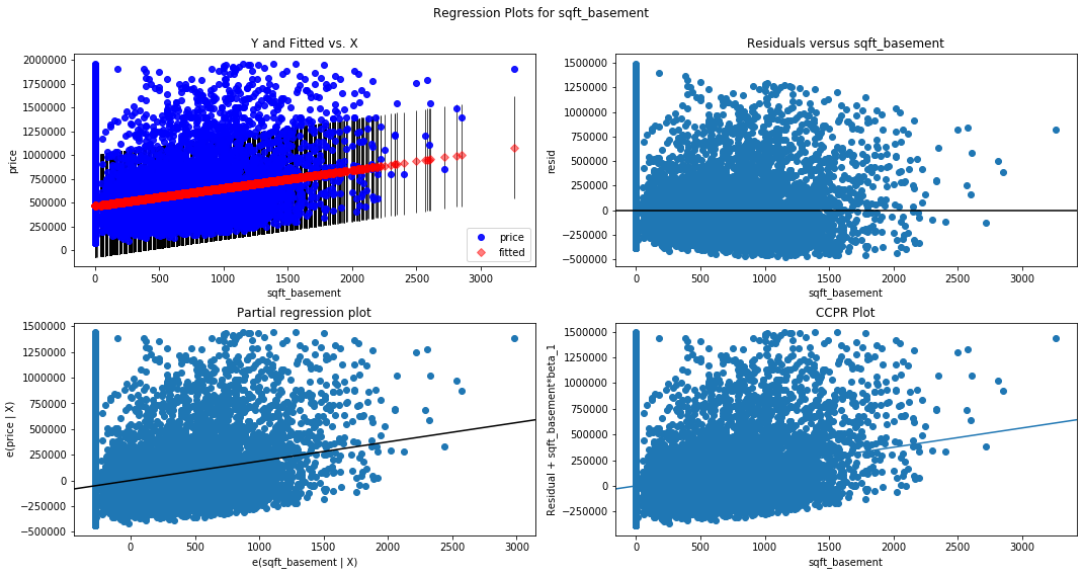


In [394]:

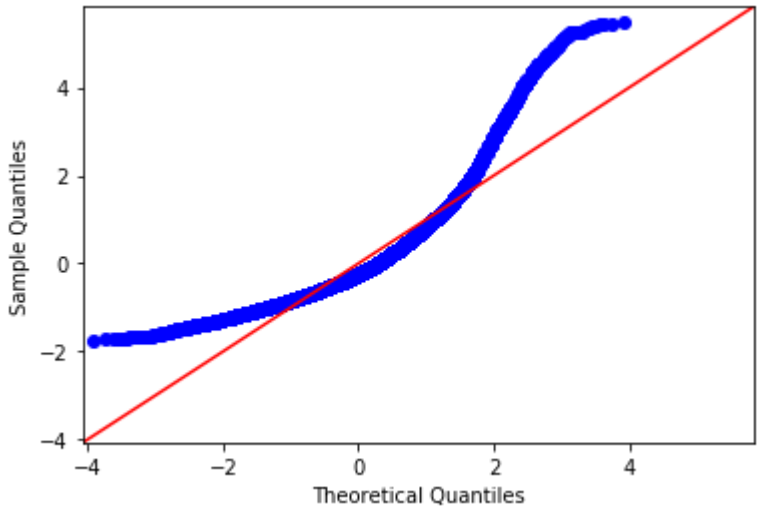
```
f = 'price~sqft_basement'
model = smf.ols(formula=f, data=df).fit()
print ('R-Squared:',model.rsquared)
print (model.params)
X_new = pd.DataFrame({'sqft_basement': [df.sqft_basement.min(), df.sqft
preds = model.predict(X_new)
df.plot(kind='scatter', x='sqft_basement', y='price');
plt.plot(X_new, preds, c='red', linewidth=2);
plt.show()
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_basement", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=Tru
fig.show()
```

```
R-Squared: 0.07939243950679753
Intercept        466034.390734
sqft_basement       187.596030
dtype: float64
```





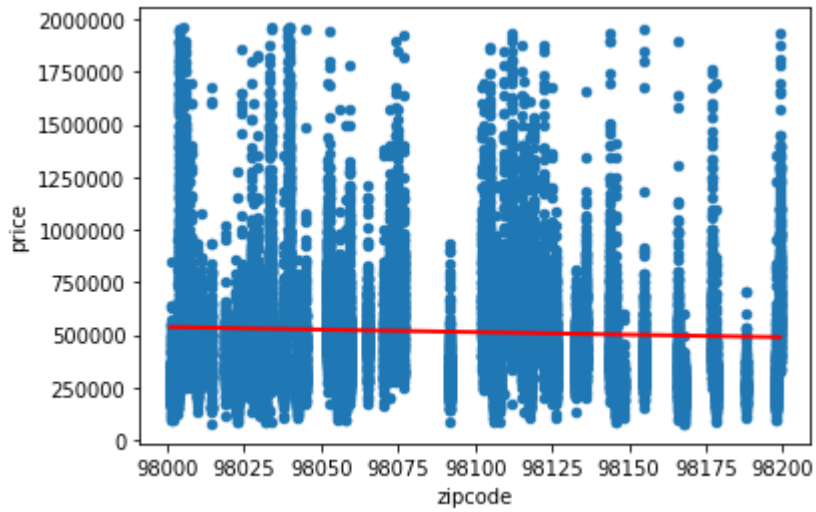
```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\ipykernel_launcher.py:1
6: UserWarning: Matplotlib is currently using module://ipykernel.pylab.back
end_inline, which is a non-GUI backend, so cannot show the figure.
  app.launch_new_instance()
```

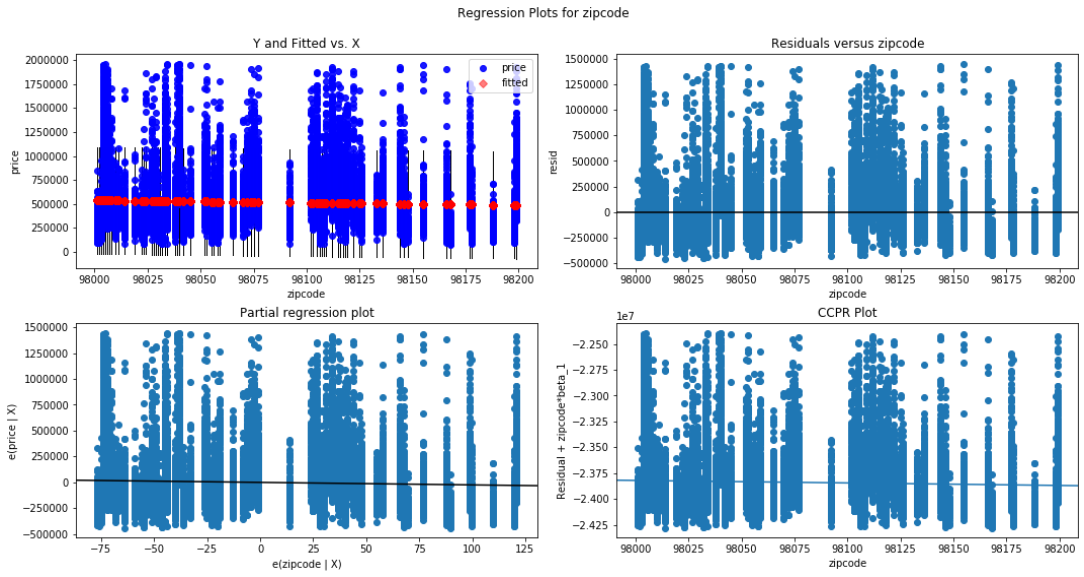


In [396]:

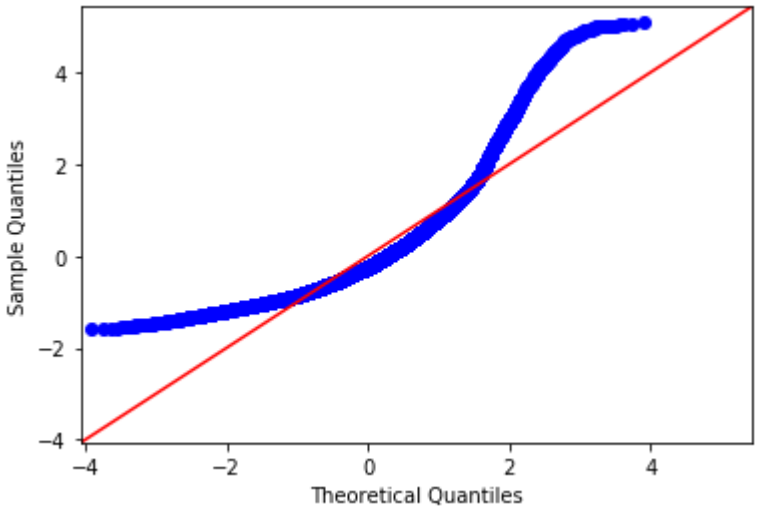
```
f = 'price~zipcode'
model = smf.ols(formula=f, data=df).fit()
print ('R-Squared:',model.rsquared)
print (model.params)
X_new = pd.DataFrame({'zipcode': [df.zipcode.min(), df.zipcode.max()]})
preds = model.predict(X_new)
df.plot(kind='scatter', x='zipcode', y='price');
plt.plot(X_new, preds, c='red', linewidth=2);
plt.show()
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "zipcode", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=Tru
fig.show()
```

```
R-Squared: 0.002083310448225628
Intercept    2.435655e+07
zipcode     -2.430538e+02
dtype: float64
```





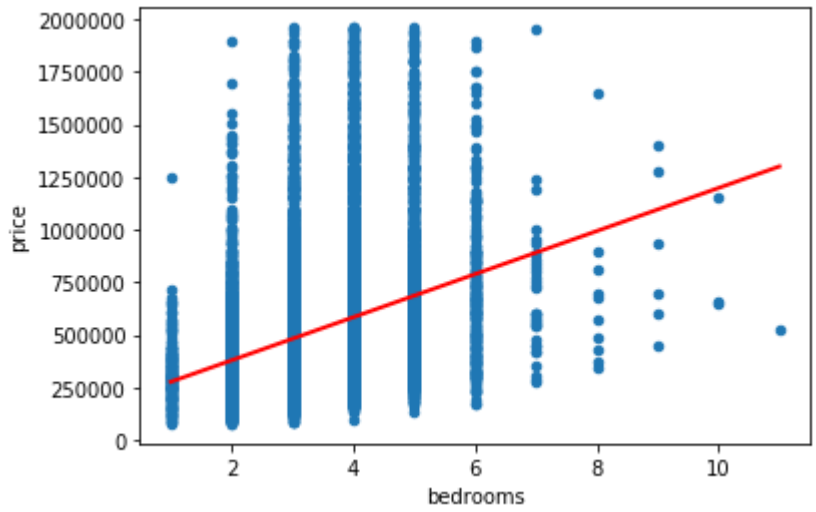
```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\ipykernel_launcher.py:1
6: UserWarning: Matplotlib is currently using module://ipykernel.pylab.back
end_inline, which is a non-GUI backend, so cannot show the figure.
  app.launch_new_instance()
```

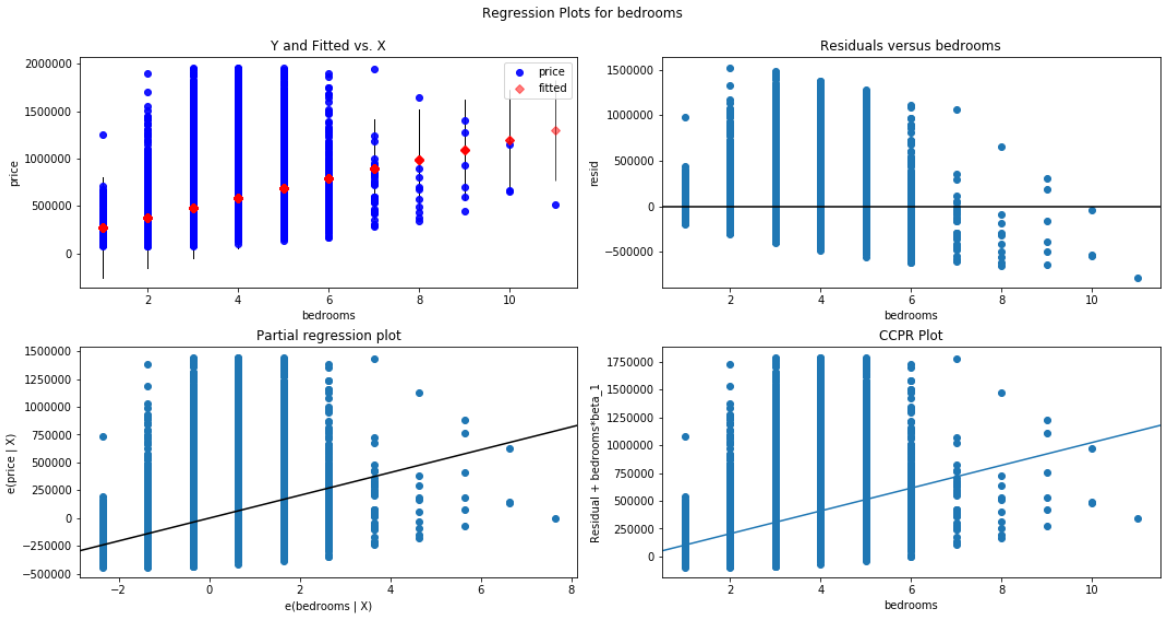


In [398]:

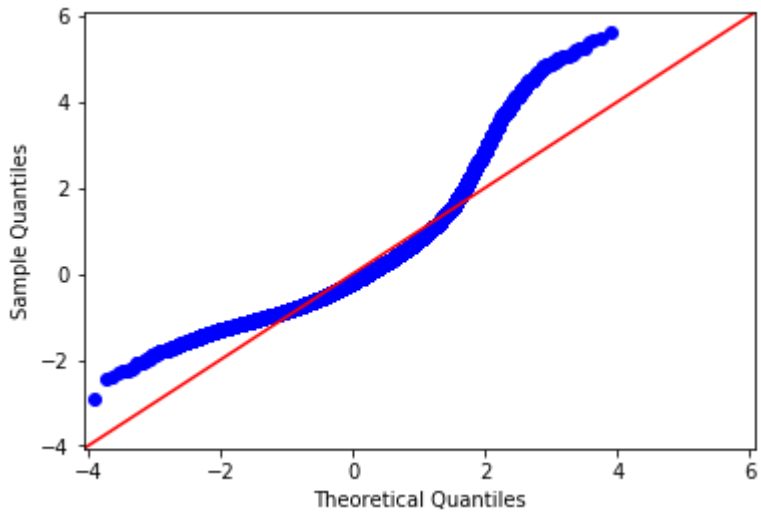
```
f = 'price~bedrooms'
model = smf.ols(formula=f, data=df).fit()
print ('R-Squared:',model.rsquared)
print (model.params)
X_new = pd.DataFrame({'bedrooms': [df.bedrooms.min(), df.bedrooms.max()
preds = model.predict(X_new)
df.plot(kind='scatter', x='bedrooms', y='price');
plt.plot(X_new, preds, c='red', linewidth=2);
plt.show()
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "bedrooms", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=Tru
fig.show()
```

```
R-Squared: 0.10403493835924071
Intercept    174215.931999
bedrooms     102348.574257
dtype: float64
```






```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\ipykernel_launche
r.py:16: UserWarning: Matplotlib is currently using module://ipykerne
l.pylab.backend_inline, which is a non-GUI backend, so cannot show the
figure.
  app.launch_new_instance()
```



7.a. Preliminary analysis

As preliminary analysis, sqft_living, sqft_above, bathrooms are among the features best performing for linear regression model. Other variables had non-linear relationship explaned by low r-squared.

8. Multiple linear regression model

Multiple linear regression is a model for predicting the value of one dependent variable based on two or more independent variables.

8.a. Baseline model using statsmodel

In this first modeling attempt we push all parameters to the model. We see how it works. Then we are going to drop some parameters according to the p-values and variance inflation factors.

In [399]:

```
#Multiple linear regression model with statsmodels

X = preprocessed.drop('price_log', axis=1)
y = preprocessed['price_log']
X_int = sm.add_constant(X)
model = sm.OLS(y,X_int).fit()
model.summary()
```

sqft_living_log	0.2421	0.007	34.688	0.000	0.228	0.256
sqft_lot_log	0.1273	0.007	19.580	0.000	0.115	0.140
sqft_above_log	0.1101	0.007	16.476	0.000	0.097	0.123
yr_built_log	-0.0169	0.004	-3.760	0.000	-0.026	-0.008
sqft_living15_log	0.0926	0.004	21.327	0.000	0.084	0.101
sqft_lot15_log	-0.0328	0.006	-5.104	0.000	-0.045	-0.020
floors_1.5	0.0132	0.010	1.308	0.191	-0.007	0.033
floors_2.0	-0.0182	0.009	-2.038	0.042	-0.036	-0.001
floors_2.5	-0.0036	0.031	-0.114	0.909	-0.065	0.058
floors_3.0	-0.1490	0.019	-7.803	0.000	-0.186	-0.112
floors_3.5	-0.1616	0.146	-1.109	0.267	-0.447	0.124
condition_2	0.3032	0.072	4.227	0.000	0.163	0.444
condition_3	0.5775	0.067	8.650	0.000	0.447	0.708

8.b. sklearn- Baseline Model

Before modelling, I will split he data into train and test. Model using only the train data. Test the model on test data. For model refinement, I will select features with hight coefficents that can negatively or positively influence the price.

In [44]:

```
def filter_columns(df):
    '''Return DataFrame with only certain columns
    '''
    columns_to_keep = ['price',
        'bedrooms',
        'bathrooms',
        'sqft_living',
        'sqft_lot',
        'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_abc
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', '
       'sqft_living15', 'sqft_lot15'
    ]

    return df[columns_to_keep]

```

In [45]:

```
df_filtered = filter_columns(df)
df_filtered.head()
```

Out[45]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	g
0	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	3	
1	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	3	
2	180000.0	2	1.00	770	10000	1.0	0.0	0.0	3	
3	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	5	
4	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	3	

In [46]:
```
1 df_filtered.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21380 entries, 0 to 21596
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   price          21380 non-null  float64
 1   bedrooms       21380 non-null  category
 2   bathrooms      21380 non-null  category
 3   sqft_living    21380 non-null  int64
 4   sqft_lot       21380 non-null  int64
 5   floors         21380 non-null  category
 6   waterfront     21380 non-null  category
 7   view           21380 non-null  category
 8   condition      21380 non-null  category
 9   grade          21380 non-null  category
 10  sqft_above     21380 non-null  int64
 11  sqft_basement  21380 non-null  float64
 12  yr_built       21380 non-null  int64
 13  yr_renovated   21380 non-null  float64
 14  zipcode        21380 non-null  category
 15  lat            21380 non-null  float64
 16  long           21380 non-null  float64
 17  sqft_living15  21380 non-null  int64
 18  sqft_lot15     21380 non-null  int64
dtypes: category(8), float64(5), int64(6)
memory usage: 2.1 MB
```

In [47]:
```
1 target = df_filtered['price']
2 features = df_filtered.drop('price', axis=1)
```

In [48]:
```
1 features.head()
```

Out[48]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft
0	3	1.00	1180	5650	1.0	0.0	0.0	3	7	
1	3	2.25	2570	7242	2.0	0.0	0.0	3	7	
2	2	1.00	770	10000	1.0	0.0	0.0	3	6	
3	4	3.00	1960	5000	1.0	0.0	0.0	5	7	
4	3	2.00	1680	8080	1.0	0.0	0.0	3	8	

In [49]:
```
1 #Split the data into training and test sets. Set the seed to 42 and the
2 X_train , X_test, y_train, y_test = train_test_split(features, target,
```

In [50]:
```
# Transform with MinMaxScaler, I used the MinMaxScaler to scale the tra
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

In [51]:
```
#Transform the test data (X_test) using the same scaler:
X_test_scaled = scaler.transform(X_test)
```

Baseline Model

In [52]:
```
#Fit a regression model to the training data
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

Out[52]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [53]:
```
linreg.coef_
```

Out[53]:
```
array([-177534.16001614,  212546.62793383,  456457.40389517,
        353445.31345925,   70007.61378   ,  304679.74801362,
        187564.84208087,  114947.86171879,  819454.68134789,
        325533.1706828 ,  116172.06258861, -261647.03175563,
         49463.90321689,  -89655.58395639,  360854.40707749,
       -185063.18602069,  261758.41506393, -237703.04748898])
```

In [54]:
```
linreg.intercept_
```

Out[54]:
```
-251320.09667274018
```

In [483]:
```
pd.DataFrame(data=linreg.coef_.reshape(1,-1), columns= features.columns
```

Out[483]:

floors	waterfront	view	condition	grade	sqft_above	sqft
70007.61378	304679.748014	187564.842081	114947.861719	819454.681348	325533.170683	116

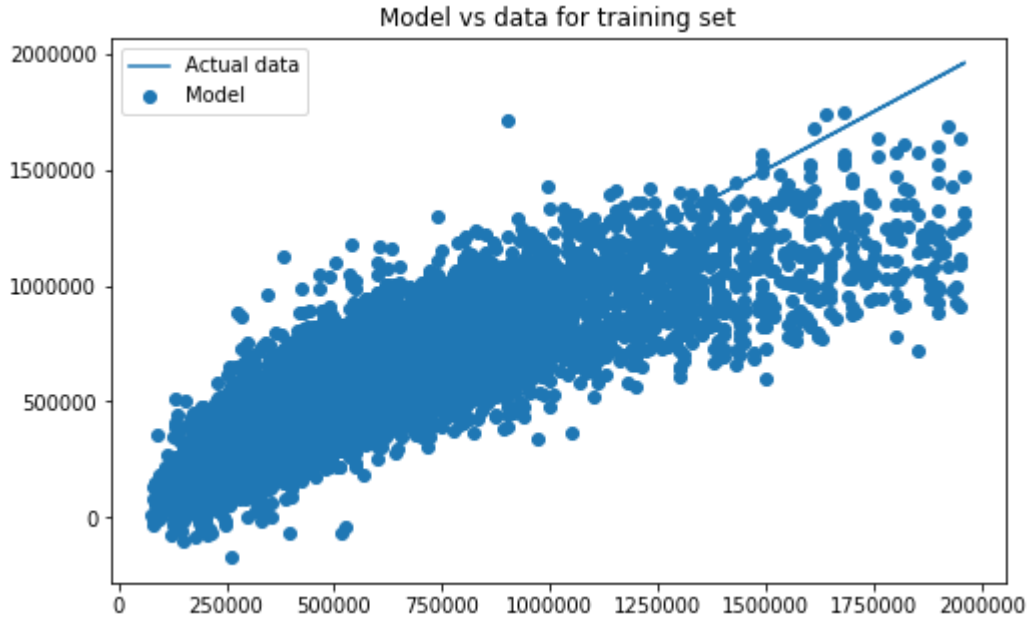
Model to make predictions on both the training and test sets:

In [55]:

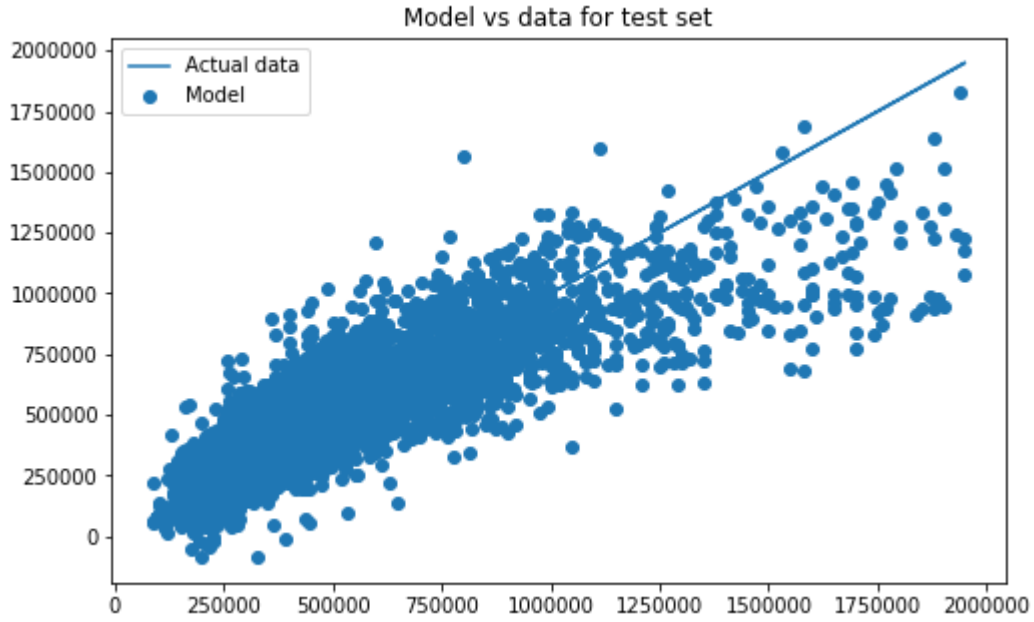
```
# Training set predictions
lm_train_predictions = linreg.predict(X_train_scaled)

# Test set predictions
lm_test_predictions = linreg.predict(X_test_scaled)
```

In [469]:

```
# vertical distance between the points and the line denote the errors
#Plot predictions for the training set against the actual data:
plt.figure(figsize=(8, 5))
plt.scatter(y_train, lm_train_predictions, label='Model')
plt.plot(y_train, y_train, label='Actual data')
plt.title('Model vs data for training set')
plt.legend();
```



In [470]:
```
#Plot predictions for the test set against the actual data:

plt.figure(figsize=(8, 5))
plt.scatter(y_test, lm_test_predictions, label='Model')
plt.plot(y_test, y_test, label='Actual data')
plt.title('Model vs data for test set')
plt.legend();
```



In [471]:
```
# Make a prediction with cross validation (cv)
scores = cross_val_score(linreg, X_train, y_train)

scores.mean(), scores.std()
```

Out[471]: (0.7167860536786433, 0.008310682108915357)

In [481]:
```
mean_squared_error(y_train, lm_train_predictions)
r2_score(y_train, lm_train_predictions)
```

Out[481]: 0.7178885833452935

In [32]:
```
#K-fold cros validation
# train {K} linear regression models on the data, with each linear mode
#and all other sections combined as the training set.
# then average the individual results frome each of these linear models
#perform 5-fold cross-validation to get the mean squared error through

scores = cross_val_score( linreg, X_train, y_train, cv=10, scoring="neg


rmse_scores = np.sqrt(-scores)
display(rmse_scores)
display(rmse_scores.mean())
display(rmse_scores.std())
```

```
array([156203.49062643, 148178.08376976, 154577.0397996 , 154540.56951064,
       150660.46849994, 150647.01902292, 158600.85033595, 152671.14655593,
       145374.48106365, 140238.66931809])

151169.18185029193

5180.177432405825
```

In [207]:
```
#Bias
def bias(y, y_hat):
    return np.mean(y_hat - y)
```

In [210]:
```
#Variance
def variance(y_hat):
    return np.mean([yi**2 for yi in y_hat]) - np.mean(y_hat)**2
```

In [211]:
```
#Calculate bias and variance
# Bias and variance for training set
b = bias(y_train, lm_train_predictions)
v =variance(lm_train_predictions)
print('Train bias: {} \nTrain variance: {}'.format(b, v))
```

```
Train bias: 7.000165469630842e-11
Train variance: 57967332005.25546
```

In [212]:
```
# Bias and variance for test set
b = bias(y_test, lm_test_predictions)
v = variance(lm_test_predictions)
print('Test bias: {} \nTest variance: {}'.format(b, v))
```

```
Test bias: -831.4950545169168
Test variance: 59263212959.64935
```

In [56]:

```
#Mean square error for train set
#Print result of MAE
print(metrics.mean_absolute_error(y_train, lm_train_predictions ))
#Print result of MSE
print(metrics.mean_squared_error(y_train, lm_train_predictions ))
#Print result of RMSE
print(np.sqrt(metrics.mean_squared_error(y_train, lm_train_predictions
```

```
105978.26806727248
22779643709.462173
150929.26723953235
```

Baseline Model comments

This model fit scores is good of 0.72 with standard deviation of 0.008. The mean rmse is 151169 and rmse standard deviation of 5180. However, there are too many features and it needs refinement.

9. Model 1- Predict Price Model

In [294]:
```
#After couple model refinements and selection of high coeffient feature
#I Selected 2 features only, with the higest coefficients
#Model refinement
columns_to_keep =['sqft_living', 'condition', "view",


        "grade", "lat", "waterfront",

    ]









features=df[columns_to_keep]
target=df['price']
X_train , X_test, y_train, y_test = train_test_split(features, target,
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

Out[294]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [295]:
```
#coefficients
display(linreg.coef_)
display(linreg.intercept_)
```

```
array([905697.10060232, 200382.82260338, 245455.47525068, 792450.47435524,
       400991.04902388, 315302.39168487])

-491014.6446427151
```

In [282]:
```
pd.DataFrame(data=linreg.coef_.reshape(1,-1), columns= features.columns
```

Out[282]:

	sqft_living	condition	view	grade	lat	waterfront
0	905697.100602	200382.822603	245455.475251	792450.474355	400991.049024	315302.391685

In [296]:
```
lm_train_predictions = linreg.predict(X_train_scaled)
SS_Residual = np.sum((y_train-lm_train_predictions )**2)
SS_Total = np.sum((y_train-np.mean(y_train))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
r_squared
```

Out[296]:
```
0.6749169538418713
```

In [297]:
```
scores = cross_val_score(linreg, X_train, y_train)

scores.mean(), scores.std()
```

Out[297]:
```
(0.6743174870839045, 0.010215740183141272)
```

In [298]:
```
#K-fold cross validation

scores = cross_val_score( linreg, X_train, y_train, cv=10, scoring="neg


rmse_scores = np.sqrt(-scores)

display(rmse_scores)
display(rmse_scores.mean())
display(rmse_scores.std())
```

```
array([169356.87415222, 157702.4813901 , 163271.80059323, 166196.0450632 ,
       164521.39598969, 161087.81975332, 167574.80213484, 167215.76728896,
       155119.32400467, 148759.28557418])

162080.55959444106

6163.473163616742
```

In [299]:
```
# Test set predictions
lm_test_predictions = linreg.predict(X_test_scaled)
```

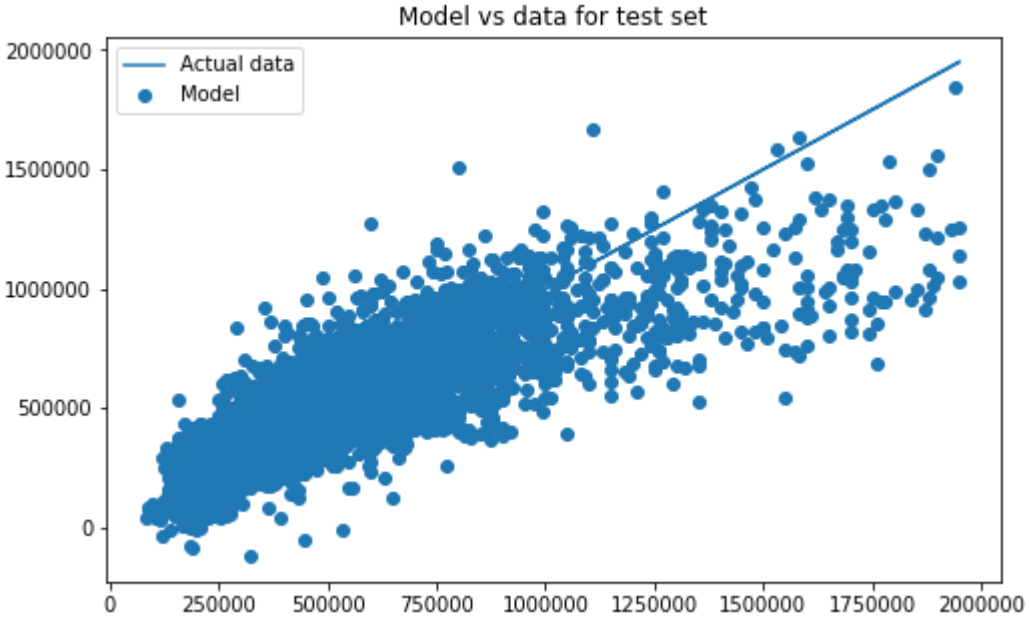
In [300]:
```
#residuals
resid = lm_test_predictions - y_test
resid
```

Out[300]:
```
10391    -46411.467572
12559    115969.361880
7540       6841.175006
10860     -2401.427738
15427      -119.244021
              ...
3974     159096.591627
10220     23810.375591
945      -11732.070773
9906    -241132.398644
18760    118206.592557
Name: price, Length: 5345, dtype: float64
```

In [301]:
```
# Training set predictions
lm_train_predictions = linreg.predict(X_train_scaled)
```

In [288]:
```
# vertical distance between the points and the line denote the errors
#Plot predictions for the training set against the actual data:
plt.figure(figsize=(8, 5))
plt.scatter(y_train, lm_train_predictions, label='Model')
plt.plot(y_train, y_train, label='Actual data')
plt.title('Model vs data for training set')
plt.legend();
```

In [302]:
```
#Plot predictions for the test set against the actual data:

plt.figure(figsize=(8, 5))
plt.scatter(y_test, lm_test_predictions, label='Model')
plt.plot(y_test, y_test, label='Actual data')
plt.title('Model vs data for test set')
plt.legend();
```



In [303]:
```
train_mse = mean_squared_error(y_train, lm_train_predictions)
test_mse = mean_squared_error(y_test,lm_test_predictions)
print('Train Mean Squarred Error:', train_mse)
print('Test Mean Squarred Error:', test_mse)
```

```
Train Mean Squarred Error: 26249472833.396843
Test Mean Squarred Error: 27097251884.640495
```

In [304]:
```
#Mean square error for test set
#Print result of MAE
print(metrics.mean_absolute_error(y_train, lm_train_predictions ))
#Print result of MSE
print(metrics.mean_squared_error(y_train, lm_train_predictions ))
#Print result of RMSE
print(np.sqrt(metrics.mean_squared_error(y_train, lm_train_predictions
```

```
113333.60950096983
26249472833.396843
162016.8905805714
```

In [305]:
```
#Mean square error for test set
#Print result of MAE
print(metrics.mean_absolute_error(y_test, lm_test_predictions ))
#Print result of MSE
print(metrics.mean_squared_error(y_test, lm_test_predictions ))
#Print result of RMSE
print(np.sqrt(metrics.mean_squared_error(y_test, lm_test_predictions ))
```

```
114489.20290064665
27097251884.640495
164612.42931395094
```

```
#### Model 1- Predict Price Model comments
This is a good model with 6 features,  with 3 most imortant features of sqft_living, grade and latitude. The mean square error of this model is 114489. Model 1 predicts home prices within $ 114,489 of the true home price on average.

```

10. Model 2 - Home Interior Model

In [215]:
```
#Initial features input into the model for internal home features,  [ '
#'floors',  'condition', 'grade', 'sqft_above',  'sqft_basement', 'yr_r
#inorder to answer the question, Does home interior features affect the
#Model 2 for home interior features
#Model refinement, keep 2 home interior features with the highest coeff

columns_to_keep = [
        'sqft_living',

          'grade',

      ]

features=df[columns_to_keep]
target=df['price']
X_train , X_test, y_train, y_test = train_test_split(features, target,
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

Out[215]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [216]:
```
#coefficients

display(linreg.coef_)
display(linreg.intercept_)
pd.DataFrame(data=linreg.coef_.reshape(1,-1), columns= features.columns
```

```
array([937246.96873716, 883375.60203995])

-148422.63626300823
```

Out[216]:

	sqft_living	grade
0	937246.968737	883375.60204

In [217]:
```
lm_train_predictions = linreg.predict(X_train_scaled)
SS_Residual = np.sum((y_train-lm_train_predictions )**2)
SS_Total = np.sum((y_train-np.mean(y_train))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
r_squared
```

Out[217]: 0.526628853093052

In [218]:
```
scores = cross_val_score(linreg, X_train, y_train)

scores.mean(), scores.std()
```

Out[218]: (0.5262447018503691, 0.005727569627331013)

In [219]:
```
#10-fold cross validation

scores = cross_val_score( linreg, X_train, y_train, cv=10, scoring="neg


rmse_scores = np.sqrt(-scores)
display(rmse_scores)
display(rmse_scores.mean())
display(rmse_scores.std())
```

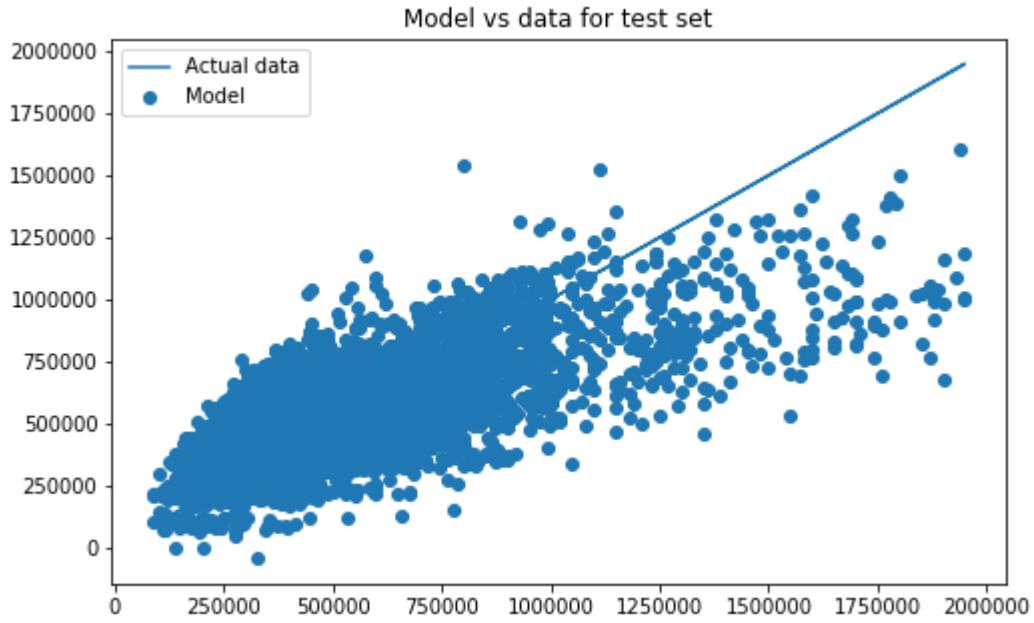
```
array([202129.01637213, 190284.60563276, 191365.34560495, 201400.81098047,
       193290.69738204, 196137.66038432, 200967.77160241, 200901.92576441,
       195101.33269128, 183201.30014586])

195478.04665606347

5820.964052264255
```

In [220]:

```
#model to make predictions on both the training and test sets:
# Training set predictions
lm_train_predictions = linreg.predict(X_train_scaled)

# Test set predictions
lm_test_predictions = linreg.predict(X_test_scaled)
# vertical distance between the points and the line denote the errors
#model to make predictions on both the training and test sets:
plt.figure(figsize=(8, 5))
plt.scatter(y_train, lm_train_predictions, label='Model')
plt.plot(y_train, y_train, label='Actual data')
plt.title('Model vs data for training set')
plt.legend();
```

In [221]:

```
#model to make predictions on both the training and test sets:
plt.figure(figsize=(8, 5))
plt.scatter(y_test, lm_test_predictions, label='Model')
plt.plot(y_test, y_test, label='Actual data')
plt.title('Model vs data for test set')
plt.legend();
```



In [222]:
```
#train residuals

resid_train = lm_train_predictions - y_train
resid_train
```

Out[222]:
```
11447    -57987.503981
7294     332433.594515
5566      40255.668695
9818     123584.485404
2032      56670.129436
              ...
11387    -69905.557964
12074    -10489.352989
5448      78670.129436
872     -128766.263167
15945     79571.209895
Name: price, Length: 16035, dtype: float64
```

In [224]:
```
#test residuals

resid_test = lm_test_predictions - y_test
resid_test
```

Out[224]:
```
10391     40440.127495
12559     69623.116426
7540      32042.535505
10860      6740.544178
15427   -157524.560360
              ...
3974     390158.428571
10220    117307.849591
945      166537.472393
9906    -290046.322631
18760    -89383.795693
Name: price, Length: 5345, dtype: float64
```

In [225]:
```
#train and test mean squared error
train_mse = mean_squared_error(y_train, lm_train_predictions)
test_mse = mean_squared_error(y_test,lm_test_predictions)
print('Train Mean Squarred Error:', train_mse)
print('Test Mean Squarred Error:', test_mse)

```

```
Train Mean Squarred Error: 38223288503.34336
Test Mean Squarred Error: 38288073770.42306
```

In [226]:
```
#Mean square error for train set
#Print result of MAE
print(metrics.mean_absolute_error(y_train, lm_train_predictions ))
#Print result of MSE
print(metrics.mean_squared_error(y_train, lm_train_predictions ))
#Print result of RMSE
print(np.sqrt(metrics.mean_squared_error(y_train, lm_train_predictions
```

```
142784.6086207621
38223288503.34336
195507.77095385073
```

In [227]:
```
#Mean square error for test set
#Print result of MAE
print(metrics.mean_absolute_error(y_test, lm_test_predictions ))
#Print result of MSE
print(metrics.mean_squared_error(y_test, lm_test_predictions ))
#Print result of RMSE
print(np.sqrt(metrics.mean_squared_error(y_test, lm_test_predictions ))
```

```
143763.09406045912
38288073770.42306
195673.38544222884
```

Model 2 for home interior comments

This is a good model with only 2 features grade and sqft_living with mean square error of 143763. Model 2 predicts home prices within $ 143, 763 of the true home price on average.

11. Model 3 - Exterior Geographic Features Model

In [236]:
```
#After couple model refinemnts of selecting  features with highest coef
#Model refinement

columns_to_keep =[ 'view', 'sqft_lot',
        'lat']









features=df[columns_to_keep]
target=df['price']
X_train , X_test, y_train, y_test = train_test_split(features, target,
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

Out[236]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [237]:
```
#coefficients
display(linreg.coef_)
display(linreg.intercept_)
pd.DataFrame(data=linreg.coef_.reshape(1,-1), columns= features.columns
```

```
array([ 536252.00264864, 1153399.45466326,  470349.50326379])

173216.25029527873
```

Out[237]:

	view	sqft_lot	lat
0	536252.002649	1.153399e+06	470349.503264

In [238]:
```
lm_train_predictions = linreg.predict(X_train_scaled)
SS_Residual = np.sum((y_train-lm_train_predictions )**2)
SS_Total = np.sum((y_train-np.mean(y_train))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
r_squared
```

Out[238]:
```
0.2620998111976154
```

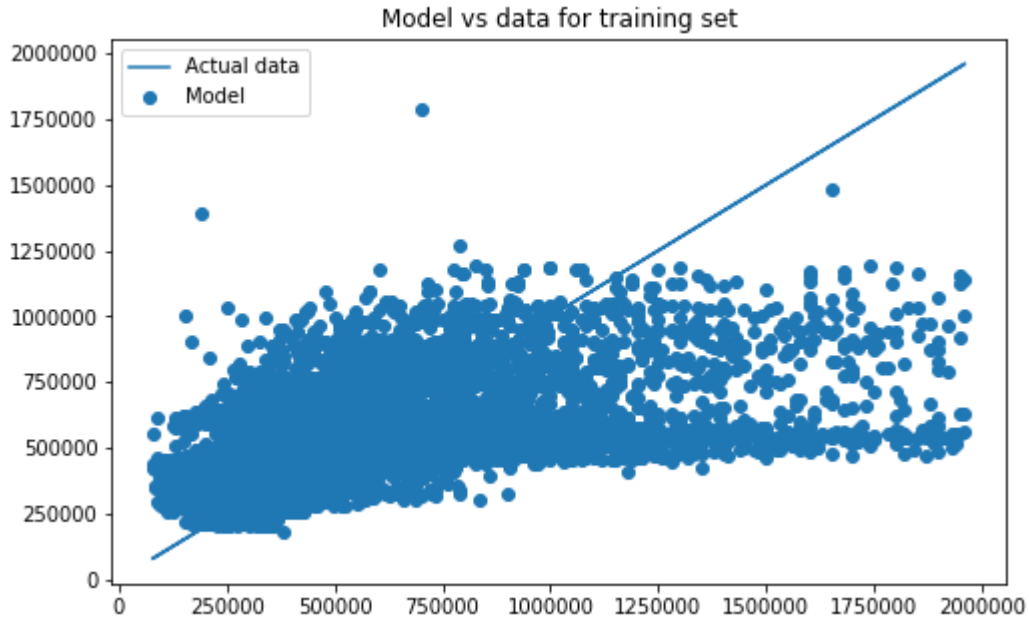
In [239]:
```
scores = cross_val_score(linreg, X_train, y_train)

scores.mean(), scores.std()
```

Out[239]: (0.2620760345034613, 0.0203713947604395)

In [240]:
```
# 10-fold cross validation
scores = cross_val_score( linreg, X_train, y_train, cv=10, scoring="neg


rmse_scores = np.sqrt(-scores)
display(rmse_scores)
display(rmse_scores.mean())
display(rmse_scores.std())
```

```
array([251305.82743652, 235154.85306336, 239875.71818249, 256775.65931571,
       247996.29229683, 248274.22442847, 255039.1161614 , 246327.10195405,
       235511.77245482, 224510.71787373])

244077.12831673754

9618.61935479307
```

In [241]:
```
#model to make predictions on both the training and test sets:
# Training set predictions
lm_train_predictions = linreg.predict(X_train_scaled)

# Test set predictions
lm_test_predictions = linreg.predict(X_test_scaled)
# vertical distance between the points and the line denote the errors
#model to make predictions on both the training and test sets:
plt.figure(figsize=(8, 5))
plt.scatter(y_train, lm_train_predictions, label='Model')
plt.plot(y_train, y_train, label='Actual data')
plt.title('Model vs data for training set')
plt.legend();
```



In [68]:
```
#train residuals
resid_train = lm_train_predictions - y_train
resid_train
```

Out[68]:
```
11447    -51065.135745
7294     -57549.963834
5566     298715.465064
9818      54631.294583
2032     -77934.417359
              ...
11387    -91596.208322
12074    191083.796798
5448     170588.037002
872       40828.705617
15945    385013.191001
Name: price, Length: 16035, dtype: float64
```

In [69]:
```
#test residuals
resid_test = lm_test_predictions - y_test
resid_test
```

Out[69]:
```
10391     11256.098149
12559    -41812.613023
7540     304642.876006
10860    311911.651715
15427    206861.433110
             ...
3974     -75762.786065
10220     74359.058770
945       47903.659938
9906     -79130.527973
18760     94142.083784
Name: price, Length: 5345, dtype: float64
```

In [70]:
```
#train and test mean square error
train_mse = mean_squared_error(y_train, lm_train_predictions)
test_mse = mean_squared_error(y_test,lm_test_predictions)
print('Train Mean Squarred Error:', train_mse)
print('Test Mean Squarred Error:', test_mse)

```

```
Train Mean Squarred Error: 59583208625.111694
Test Mean Squarred Error: 60620443237.071495
```

In [242]:
```
#Mean square error for train set
#Print result of MAE
print(metrics.mean_absolute_error(y_train, lm_train_predictions ))
#Print result of MSE
print(metrics.mean_squared_error(y_train, lm_train_predictions ))
#Print result of RMSE
print(np.sqrt(metrics.mean_squared_error(y_train, lm_train_predictions
```

```
172013.443811179
59583208625.111694
244096.71981637052
```

In [243]:
```
#Mean square error for test set
#Print result of MAE
print(metrics.mean_absolute_error(y_test, lm_test_predictions ))
#Print result of MSE
print(metrics.mean_squared_error(y_test, lm_test_predictions ))
#Print result of RMSE
print(np.sqrt(metrics.mean_squared_error(y_test, lm_test_predictions ))
```

```
173343.6517758979
60620443237.07147
246212.19148748802
```

```
#### Model 3 for home exterior geographic features comments
This model did not perform well with mean absolute error of 173343. As result, we can only depend on geographic features to predict the house sale price. There are other category of features which impact the prices.

```

12. Model 4- Increase Sale Price Model

Select the best performing features from all above models (i.e.geographic models, interior features models, predict price model).

In [138]:

```
z_keep4=[

'sqft_living15_log',
'floors_2.0',

'waterfront_1.0',
'bedrooms_6',
'view_1.0',
'view_2.0',
'view_3.0',
'view_4.0',

'bathrooms_3.75',
'bathrooms_4.0',
'bathrooms_4.25',
'bathrooms_4.5',
'bathrooms_4.75',
'bathrooms_5.0',
'bathrooms_5.25',
'bathrooms_5.5',


'grade_8',
'grade_11',
'grade_10',
'grade_12',
'zipcode_98004',
'zipcode_98005',
'zipcode_98006',
'zipcode_98007',
'zipcode_98008',
'zipcode_98010',
'zipcode_98011',
'zipcode_98014',
'zipcode_98019',
'zipcode_98022',
'zipcode_98024',
'zipcode_98027',
'zipcode_98028',
'zipcode_98029',
'zipcode_98030',
'zipcode_98031',
'zipcode_98033',
'zipcode_98034',
'zipcode_98038',
'zipcode_98039',
'zipcode_98040',
'zipcode_98042',
'zipcode_98045',
'zipcode_98052',
'zipcode_98053',
'zipcode_98055',
'zipcode_98056',
'zipcode_98058',
'zipcode_98059',
'zipcode_98065',
```

```
'zipcode_98070',
'zipcode_98072',
'zipcode_98074',
'zipcode_98075',
'zipcode_98077',
'zipcode_98092',
'zipcode_98102',
'zipcode_98103',
'zipcode_98105',
'zipcode_98106',
'zipcode_98107',
'zipcode_98108',
'zipcode_98109',
'zipcode_98112',
'zipcode_98115',
'zipcode_98116',
'zipcode_98117',
'zipcode_98118',
'zipcode_98119',
'zipcode_98122',
'zipcode_98125',
'zipcode_98126',
'zipcode_98133',
'zipcode_98136',
'zipcode_98144',
'zipcode_98146',
'zipcode_98148',
'zipcode_98155',
'zipcode_98166',
'zipcode_98168',
'zipcode_98177',
'zipcode_98178',
'zipcode_98188',
'zipcode_98198',
'zipcode_98199']

features=preprocessed[z_keep4]

target = preprocessed['price_log']
X_train , X_test, y_train, y_test = train_test_split(features, target,


linreg = LinearRegression()
linreg.fit(X_train, y_train)
```

Out[138]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [139]:
```
#coefficients
display(linreg.coef_)
display(linreg.intercept_)
zip(features, linreg.coef_)

pd.DataFrame(data=linreg.coef_.reshape(1,-1), columns= features.columns
```

```
array([0.40005897, 0.18592817, 0.82743031, 0.1794887 , 0.26945887,
       0.29248928, 0.47393881, 0.71007598, 0.39390517, 0.23138531,
       0.33228994, 0.29890138, 0.39366115, 0.47620188, 0.48127859,
       0.46212901, 0.10362594, 0.67620977, 0.48263451, 0.79527207,
       2.25944771, 1.63420031, 1.32659829, 1.35900112, 1.36156064,
       0.67389767, 0.89332701, 0.67088642, 0.62897841, 0.21558682,
       1.02663216, 1.01827949, 0.82804404, 1.19286312, 0.07706159,
       0.1318008 , 1.59511382, 1.08446303, 0.30496519, 2.69116869,
       1.83749981, 0.13528154, 0.72665948, 1.29425883, 1.25414328,
       0.23838845, 0.66213743, 0.34279552, 0.69113504, 0.72249967,
       0.77646417, 1.02879997, 1.13659892, 1.09488571, 1.04662212,
       0.09290605, 1.87350568, 1.69501166, 1.96776143, 0.65712102,
       1.67471657, 0.74177943, 1.98597411, 2.12684453, 1.66018987,
       1.55161427, 1.67930107, 0.84833144, 1.91818114, 1.59128495,
       1.12382492, 1.0909434 , 0.93384376, 1.30829334, 1.2875893 ,
       0.53608423, 0.2313844 , 0.88060702, 0.69702653, 0.0357514 ,
       1.20849307, 0.24524252, 0.23417898, 0.07923256, 1.72956193])

-1.1387758613960475
```

Out[139]:

	sqft_living15_log	floors_2.0	waterfront_1.0	bedrooms_6	view_1.0	view_2.0	view_3.0	vie
0	0.400059	0.185928	0.82743	0.179489	0.269459	0.292489	0.473939	0.7

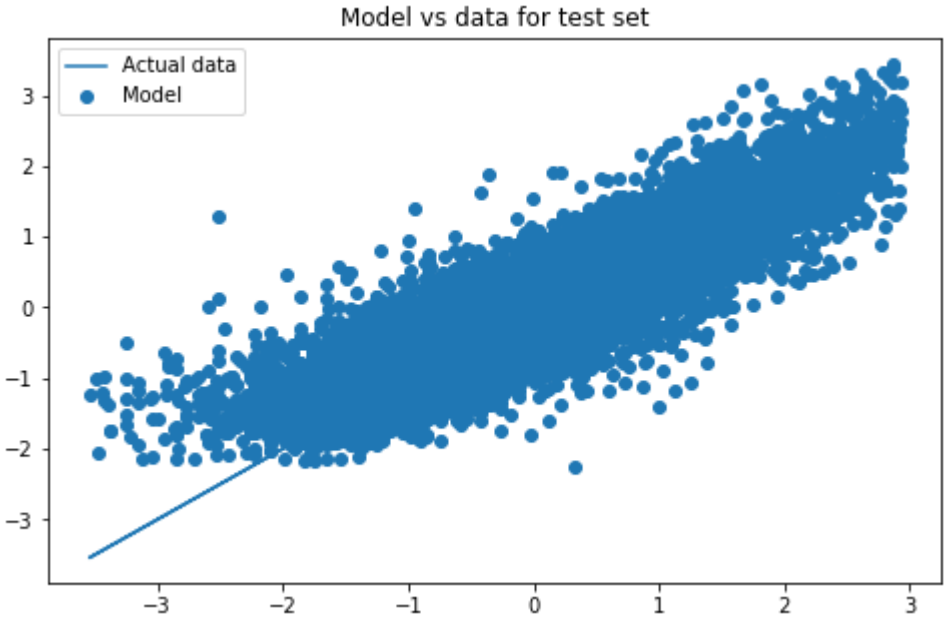
1 rows × 85 columns

In [140]:
```
#10-fold cross validation
scores = cross_val_score(linreg, X_train, y_train)

scores.mean(), scores.std()
```

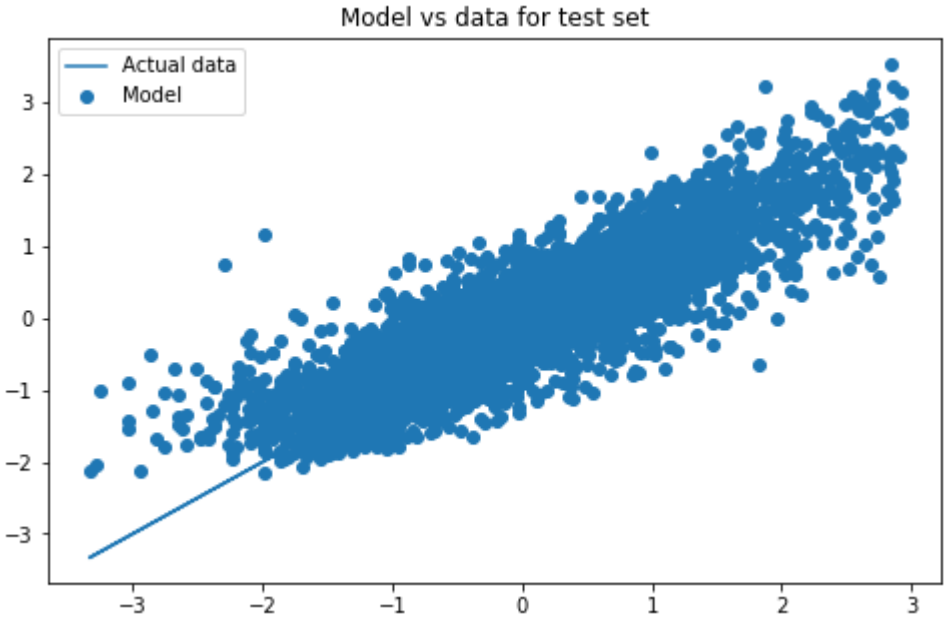
Out[140]: (0.7777327489780969, 0.004963194045755207)

In [141]:

```
#model to make predictions on both the training and test sets:
# Training set predictions
lm_train_predictions = linreg.predict(X_train)

# Test set predictions
lm_test_predictions = linreg.predict(X_test)

# vertical distance between the points and the line denote the errors
#model to make predictions on both the training and test sets:
plt.figure(figsize=(8, 5))
plt.scatter(y_train, lm_train_predictions, label='Model')
plt.plot(y_train, y_train, label='Actual data')
plt.title('Model vs data for test set')
plt.legend();
```



In [143]:
```
# vertical distance between the points and the line denote the errors
#model to make predictions on both the training and test sets:
plt.figure(figsize=(8, 5))
plt.scatter(y_test, lm_test_predictions, label='Model')
plt.plot(y_test, y_test, label='Actual data')
plt.title('Model vs data for test set')
plt.legend();
```



In [144]:
```
#Mean square error for train data

#Print result of MAE
print(metrics.mean_absolute_error(y_train, lm_train_predictions ))
#Print result of MSE
print(metrics.mean_squared_error(y_train, lm_train_predictions ))
#Print result of RMSE
print(np.sqrt(metrics.mean_squared_error(y_train, lm_train_predictions
```

```
0.3471787980994624
0.21809099362404463
0.46700213449624045
```

In [142]:

```
#Mean square error for test data

#Print result of MAE
print(metrics.mean_absolute_error(y_test, lm_test_predictions ))
#Print result of MSE
print(metrics.mean_squared_error(y_test, lm_test_predictions ))
#Print result of RMSE
print(np.sqrt(metrics.mean_squared_error(y_test, lm_test_predictions ))
```

```
0.35092959198009366
0.22496289652609278
0.474302536917201
```

Model 4 comments

Model 4 is the best performing model with mean absolute error of 0.35. Model 4 predicts home price within $0.35 of the true home price on average.

13. Data interpretation

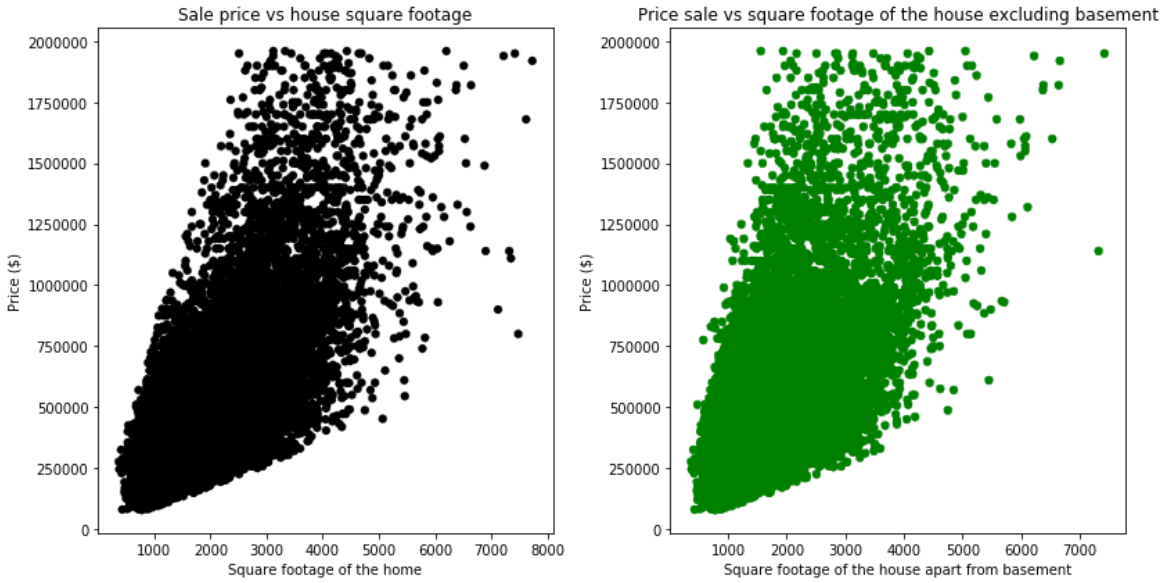
13.a. Model 1- Sale Price Prediction

The most important two features to predict the price sale are: square footage of the house , grade (how good and well maintained is the house condition). Other impotant features: location (lat, long), view (does the home have a geographic view), condition of the house, and the number of floors.

13.a.1. Visualization for the house square footage

In [657]:

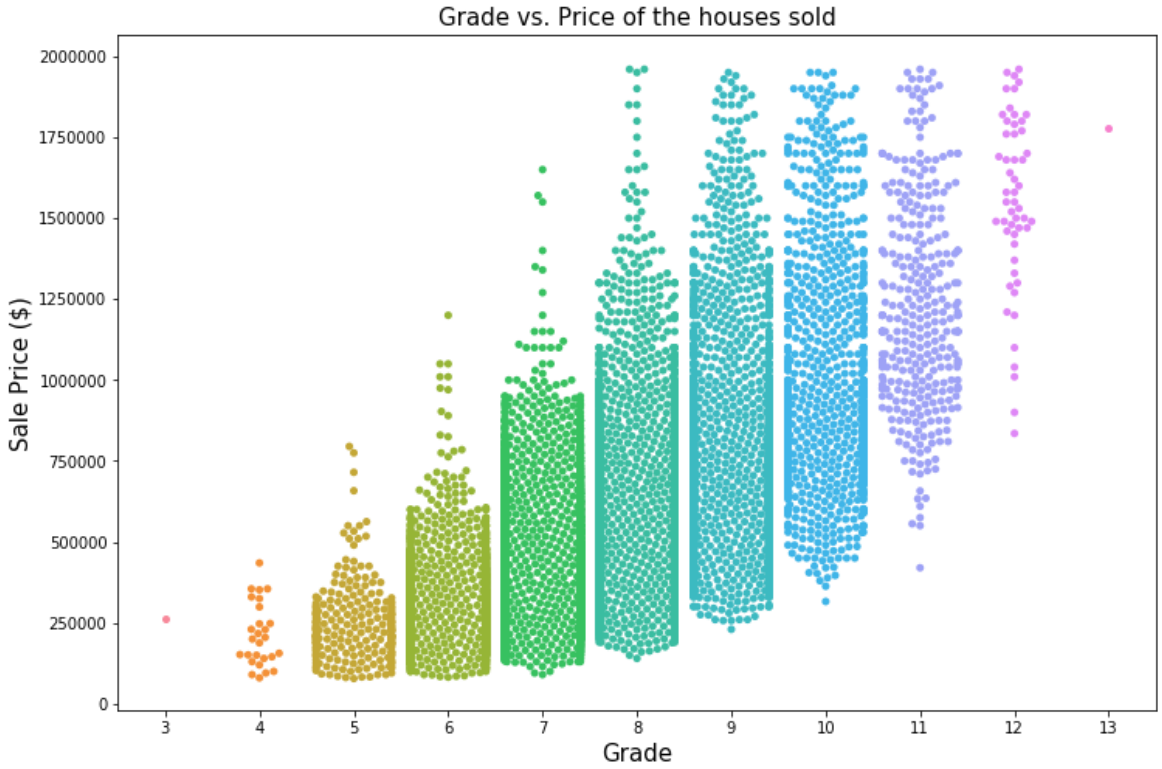
```
new_figure, (ax1, ax2) = plt.subplots(figsize=(12,6), ncols=2)
new_figure.set_tight_layout(True)

ax1.scatter(x, y, color='black', linewidth=0.5, linestyle = ':')

ax2.scatter(z,y, color='green', linewidth=0.5, linestyle = '-.')

ax1.set_xlabel('Square footage of the home')
ax1.set_ylabel('Price ($)')
ax1.set_title ('Sale price vs house square footage')

ax2.set_xlabel('Square footage of the house apart from basement')
ax2.set_ylabel('Price ($)')
ax2.set_title ('Price sale vs square footage of the house excluding bas
```



13.a.2. Visualization for the house grade

In [721]:
```
fig=plt.figure(figsize=(12,8))
sns.swarmplot(x='grade', y='price', data=df);
plt.title('Grade vs. Price of the houses sold', fontsize=15)
plt.xlabel("Grade", fontsize=15)
plt.ylabel('Sale Price ($)', fontsize=15)
```

Out[721]: Text(0, 0.5, 'Sale Price ($)')



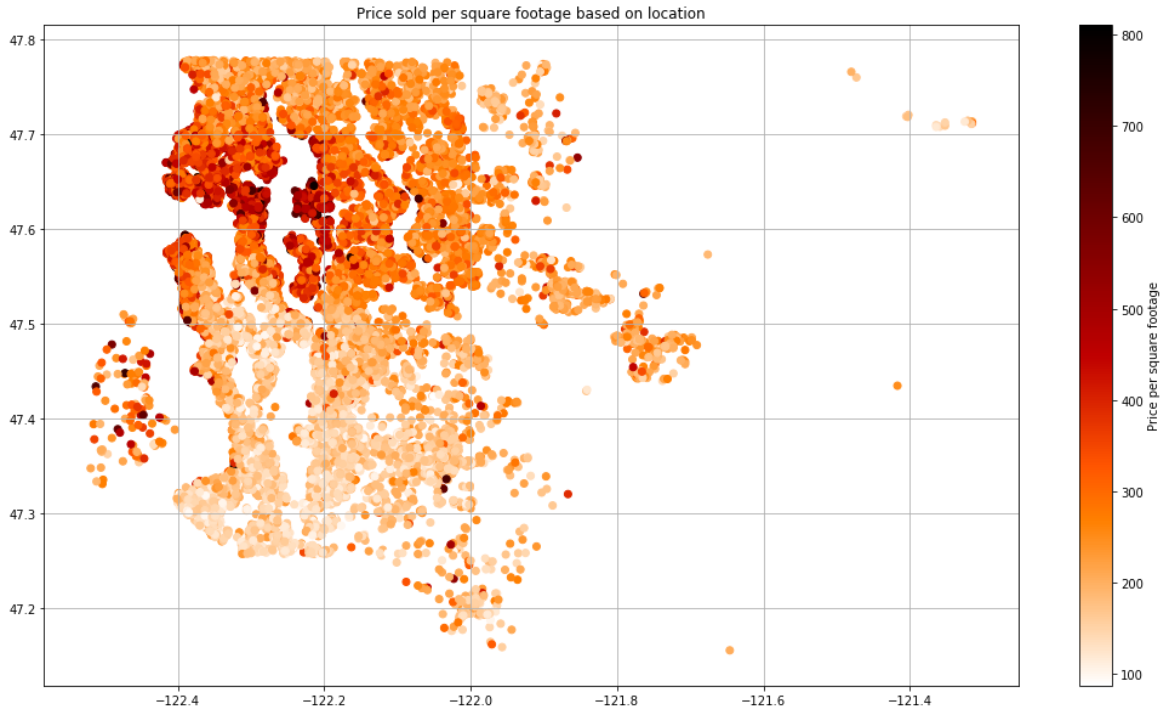
13.b. Model 3- Important geographic features vs price sale.

The most two important features are square foot of the whole house lot and view (if the home has a beautiful landscape view). By descending order: square foot of the whole house lot, view (if the home has a beautiful landscape view),location, and waterfront (if the home is facing a lake, river..ect).

13.b.1. Visualization for the location of the house

In [688]:
```
plt.figure(figsize=(18,10))
plt.scatter(df['long'], df['lat'], c=df['price']/df['sqft_living'], cma
cbar = plt.colorbar()
cbar.set_label('Price per square footage')
plt.grid(which='both')
plt.title('Price sold per square footage based on location')
plt.show
```

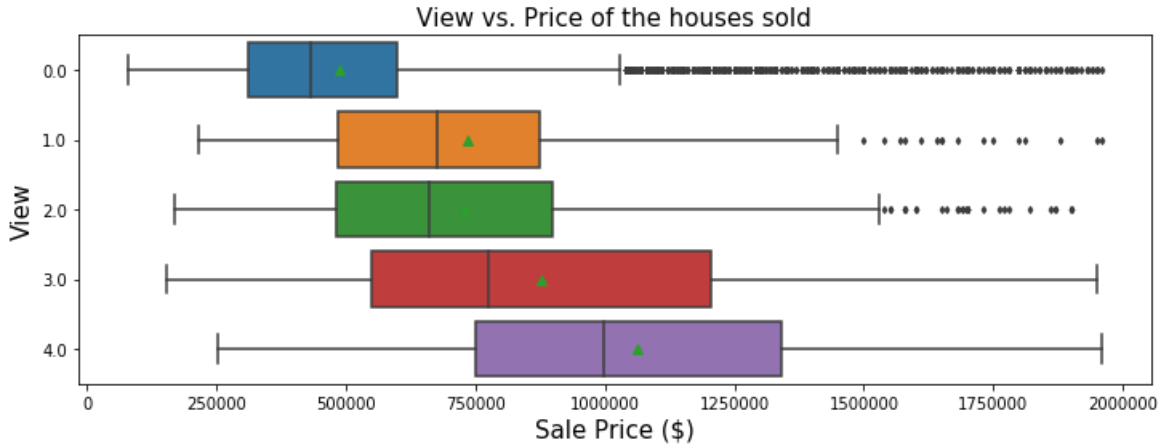
Out[688]: <function matplotlib.pyplot.show(*args, **kw)>



13.b.2 Visualization for house view

In [11]:

```
fig, ax = plt.subplots(figsize=(12,4))
sns.boxplot(y = 'view',
            x = 'price',
            data = df,
            width = 0.8,
            orient = 'h',
            showmeans = True,
            fliersize = 3,
            ax = ax)
plt.xlabel("Sale Price ($)", fontsize=15)
plt.ylabel('View', fontsize=15)
plt.title('View vs. Price of the houses sold', fontsize=15)
plt.show()
```



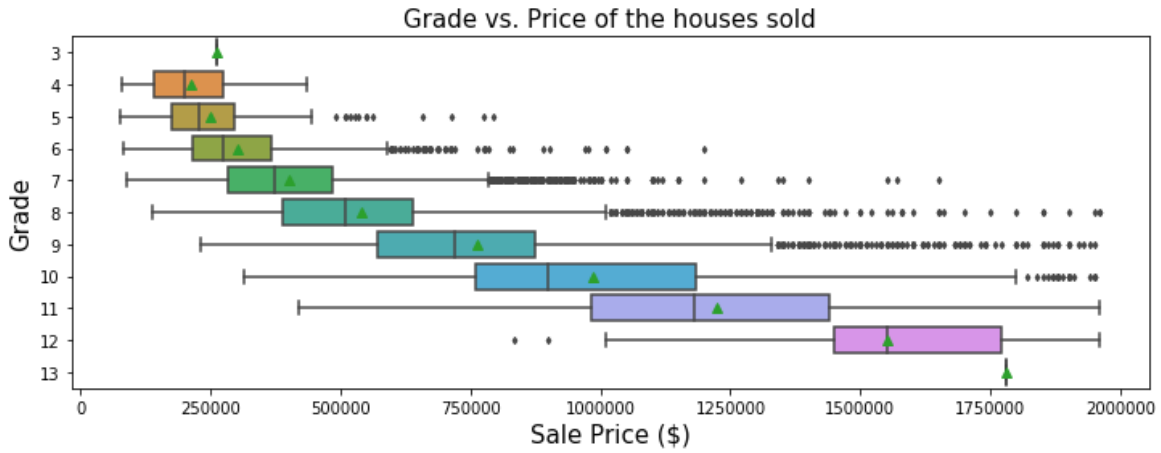
13.c. Model 2- Important house interior features vs price sale

The most important two features are grade (overall grade of the housing unit) and square footage of the home. Therefore, the most important features to predict the price sale based on home interior features are by descending order: Grade (overall grade of the housing unit), square footage of the home, total floors in the house, the square footage of interior housing living space for the nearest 15 neighbors, how good is the overall conditon of the house, square footage of the basement and the year when the house is renovated.

13.c.1. Visualization for the house grade

In [12]:

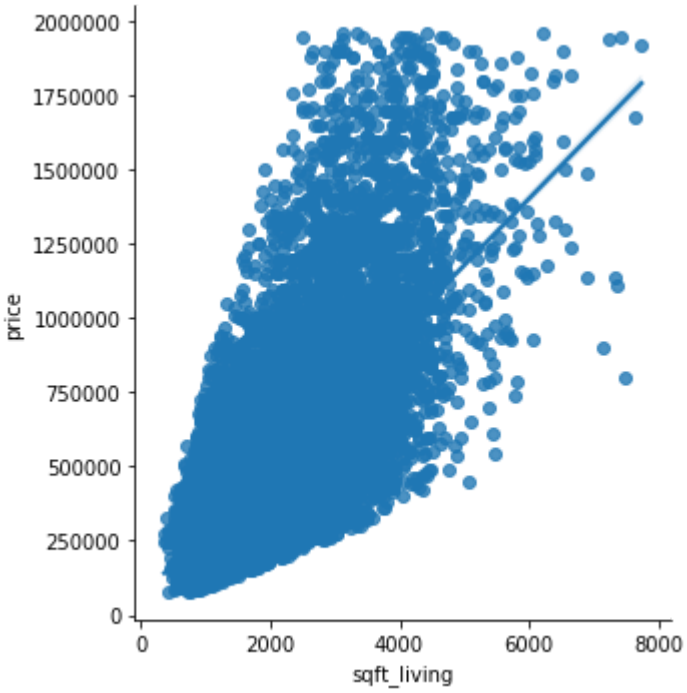
```
fig, ax = plt.subplots(figsize=(12,4))
sns.boxplot(y = 'grade',
            x = 'price',
            data = df,
            width = 0.8,
            orient = 'h',
            showmeans = True,
            fliersize = 3,
            ax = ax)
plt.title('Grade vs. Price of the houses sold', fontsize=15)
plt.xlabel("Sale Price ($)", fontsize=15)
plt.ylabel('Grade', fontsize=15)
plt.show()
```

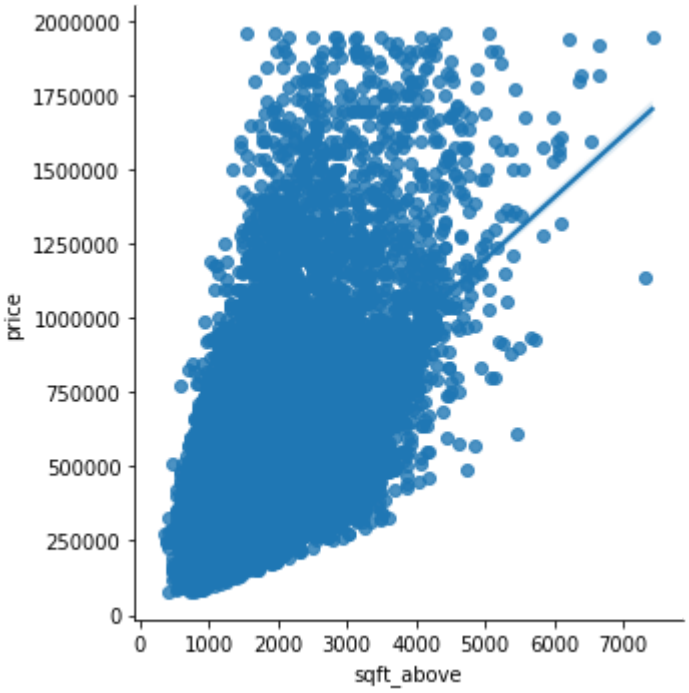


13.c.2. Visualization of the square footage of the house

In [647]:
```
sns.lmplot(x="sqft_living", y="price",data=df,)
sns.lmplot(x="sqft_above", y="price", data=df)
```

Out[647]: <seaborn.axisgrid.FacetGrid at 0x1f51f204dc8>

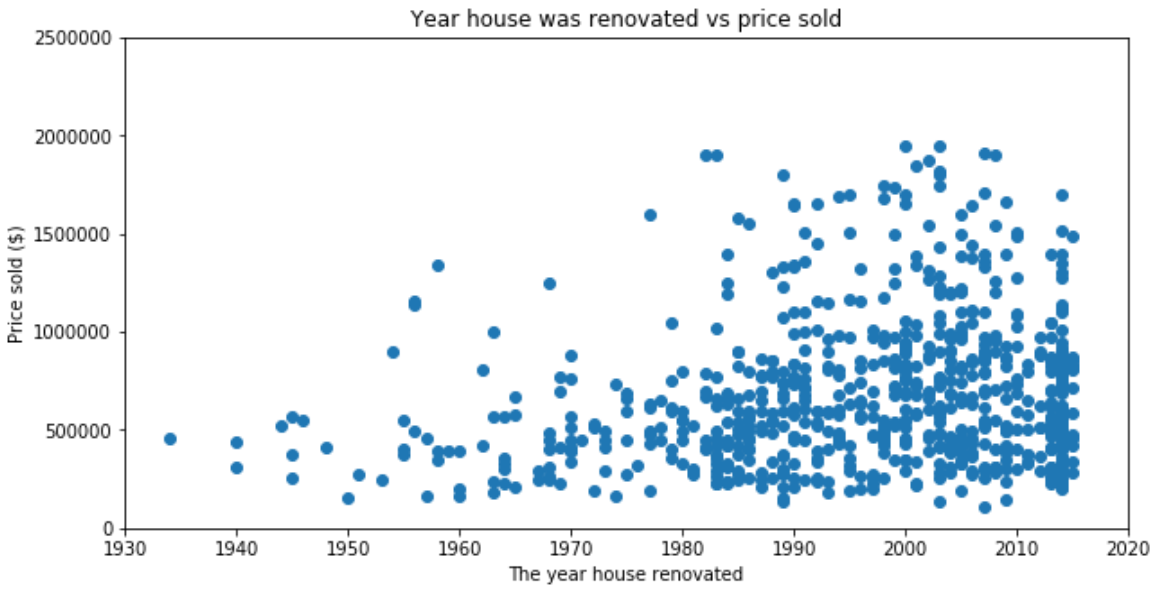




13.c.3. Vsualization of year renovation of the house

In [686]:

```
x=df['yr_renovated']
y=df['price']

new_figure = plt.figure(figsize=(10, 5))
ax = new_figure.add_subplot(111)

plt.scatter(x, y, label='Sample Data')
ax.set_xlim(1930, 2020), ax.set_ylim(0,2500000)
ax.set_xlabel('The year house renovated')
ax.set_ylabel('Price sold ($)')
ax.set_title('Year house was renovated vs price sold');
plt.show()
```



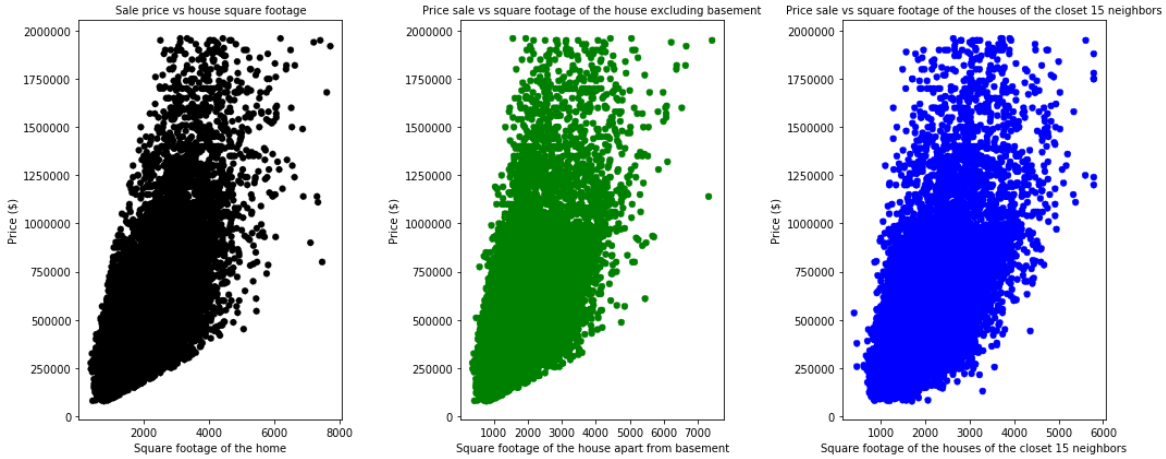
13.d. Model 4- Important features which increase the price sale

The most two important features are square footage of house a and overall grade given to the housing unit, based on King County grading system. Therefore, the most important features which can potentially increase the price sale: square footage of house apart from basement, overall grade given to the housing unit, based on King County grading system, latitude coordinate (location). Other features: square footage of the basement, house which has a view to a waterfront, If the house has a view or not, How good is the overall condition.

13.d.1. Visualization of the square footage of the house

In [266]:

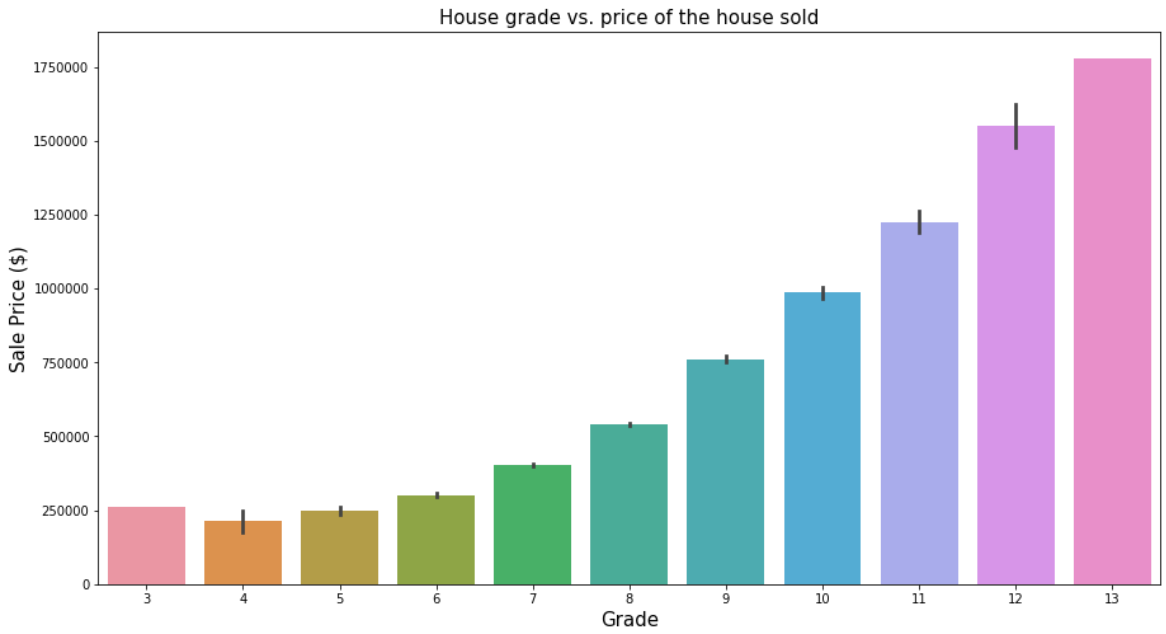
```
x=df['sqft_living']
y=df['price']
z=df['sqft_above']
m=df['sqft_living15']

new_figure, (ax1, ax2, ax3) = plt.subplots(figsize=(15,6), ncols=3)
new_figure.set_tight_layout(True)

ax1.scatter(x, y, color='black', linewidth=0.5, linestyle = ':')

ax2.scatter(z,y, color='green', linewidth=0.5, linestyle = '-.')

ax3.scatter(m,y, color='blue', linewidth=0.5, linestyle = '-.')


ax1.set_xlabel('Square footage of the home')
ax1.set_ylabel('Price ($)')
ax1.set_title ('Sale price vs house square footage', fontsize=10)

ax2.set_xlabel('Square footage of the house apart from basement')
ax2.set_ylabel('Price ($)')
ax2.set_title ('Price sale vs square footage of the house excluding bas

ax3.set_xlabel('Square footage of the houses of the closet 15 neighbors
ax3.set_ylabel('Price ($)')
ax3.set_title ('Price sale vs square footage of the houses of the close
```



13.d.2. Visualization of the house grade

In [717]:

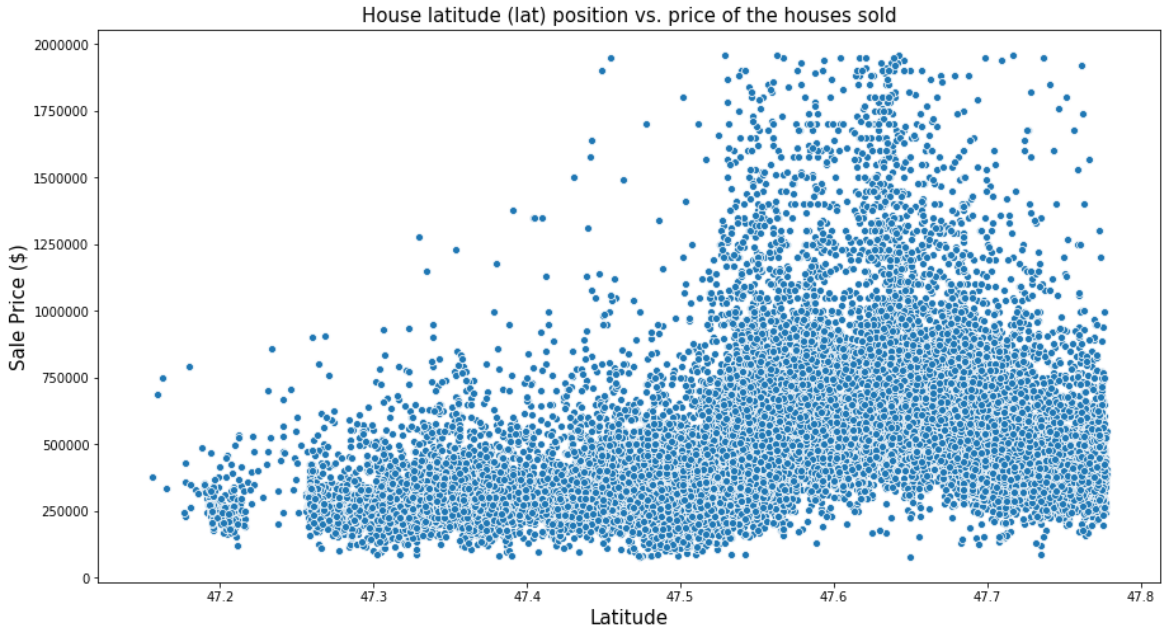
```
fig = plt.figure(figsize=(15, 8))
sns.barplot(data=df, y='price', x='grade')
plt.title('House grade vs. price of the house sold', fontsize=15)
plt.xlabel("Grade", fontsize=15)
plt.ylabel('Sale Price ($)', fontsize=15)
plt.show()
```



13.d.3. Visualization of the house latitude location

In [716]:

```
fig = plt.figure(figsize=(15, 8))
sns.scatterplot(data=df, y='price', x='lat')
plt.title('House latitude (lat) position vs. price of the houses sold',
plt.xlabel("Latitude", fontsize=15)
plt.ylabel('Sale Price ($)', fontsize=15)
plt.show()
```

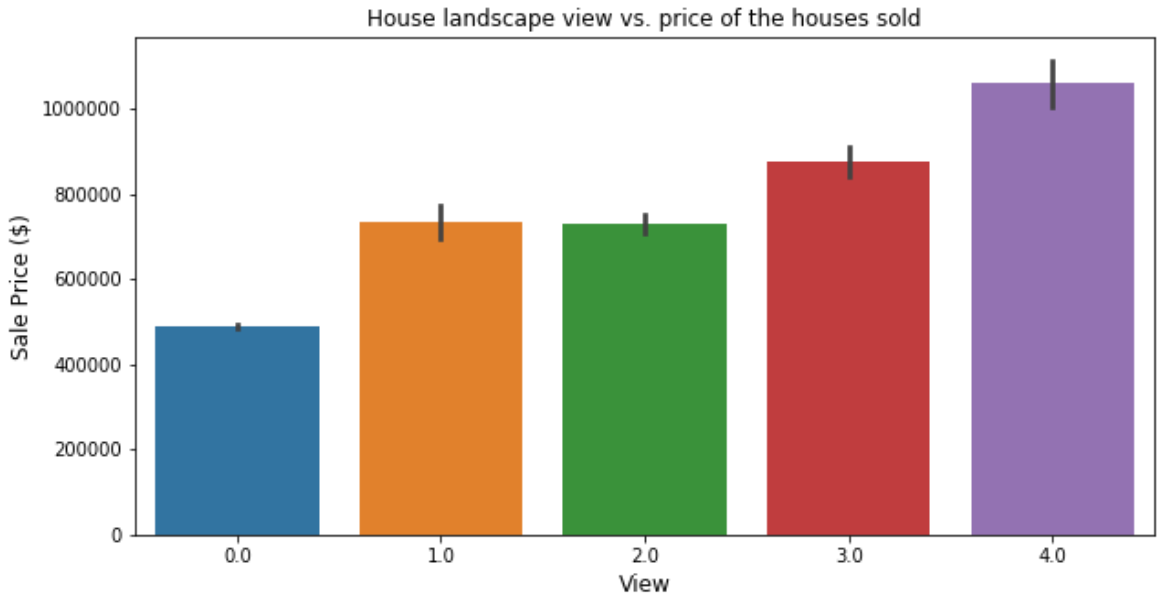


13.d.4 Visualization of the house view

In [714]:

```
fig = plt.figure(figsize=(10, 5))
sns.barplot(data=df, y='price', x='view')
plt.title('House landscape view vs. price of the houses sold', fontsize
plt.xlabel("View", fontsize=12)
plt.ylabel('Sale Price ($)', fontsize=12)
plt.show()
```

```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\pandas\io\formats\forma
t.py:1403: FutureWarning: Index.ravel returning ndarray is deprecated; in a
future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
```

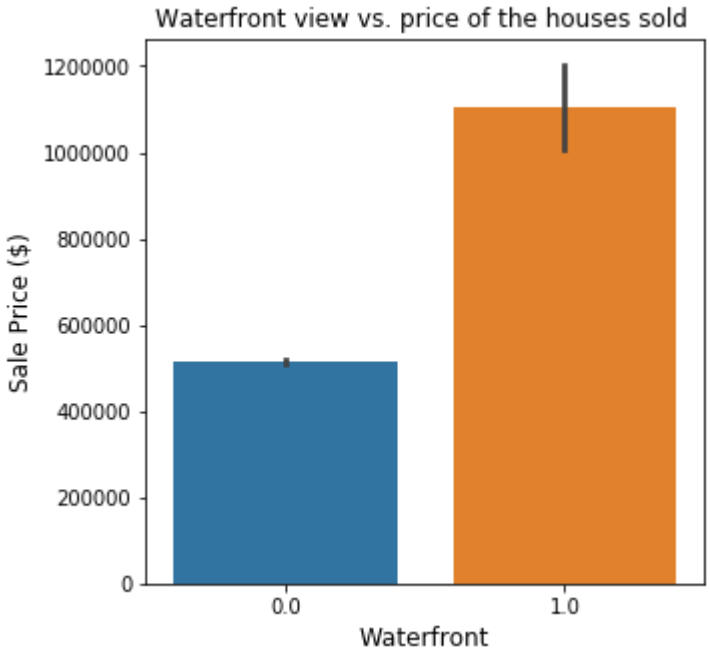


13.d.5. Visualization of the house waterfront

In [713]:

```
1
2 fig = plt.figure(figsize=(5, 5))
3 sns.barplot(data=df, y='price', x='waterfront')
4 plt.title('Waterfront view vs. price of the houses sold ', fontsize=12)
5 plt.xlabel("Waterfront", fontsize=12)
6 plt.ylabel('Sale Price ($)', fontsize=12)
7 plt.show()
```

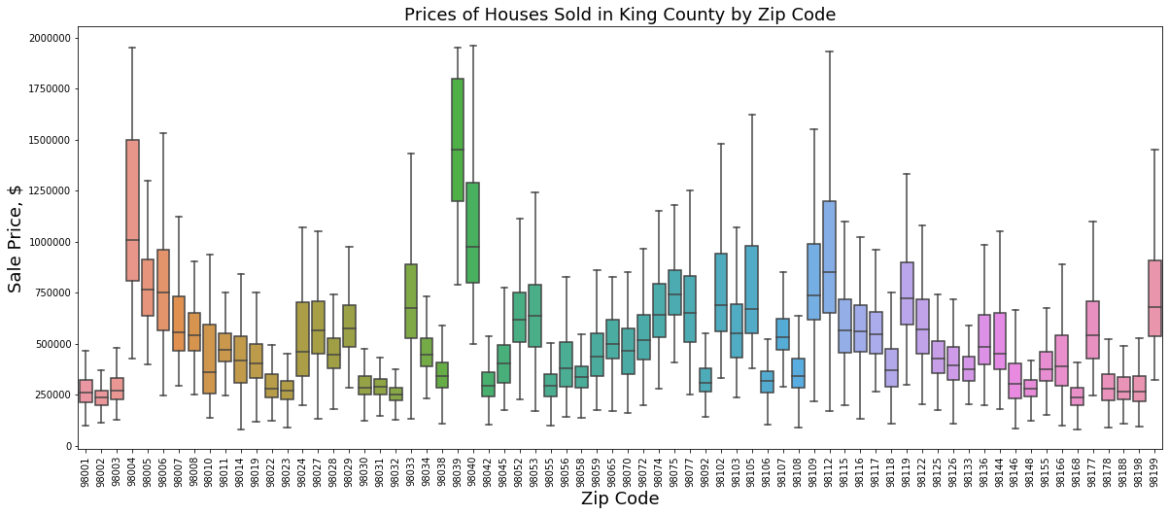
```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\pandas\io\formats\format.py:1403: FutureWarning: Index.ravel returning ndarray is deprecated; in a future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
```



13.d.6. Visualization of the houses zipcode

In [248]:

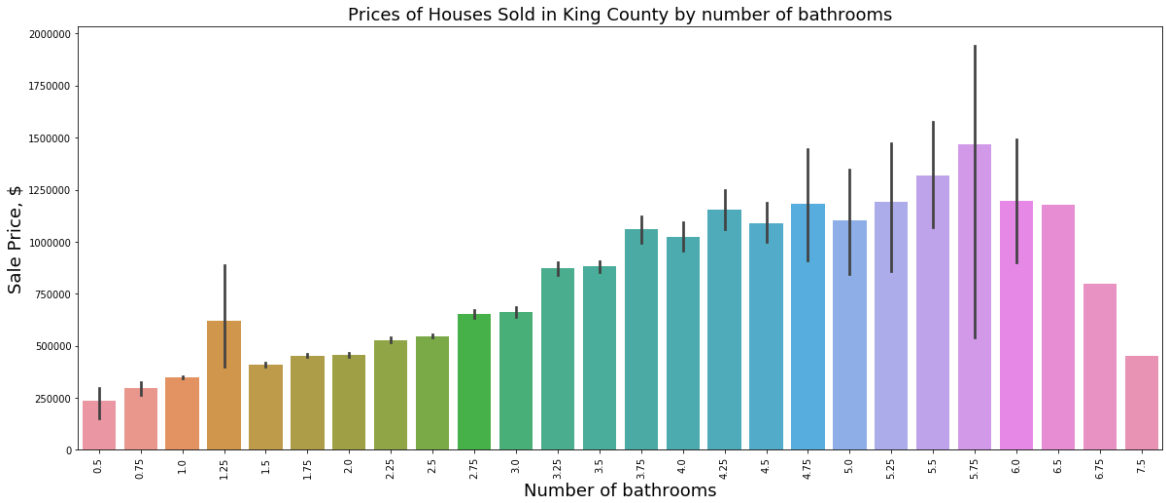
```
fig = plt.figure(figsize=(20, 8))
fig = sns.boxplot(x='zipcode', y='price', data=df, showfliers=False)
plt.title('Prices of Houses Sold in King County by Zip Code', fontsize=
plt.xlabel("Zip Code", fontsize=18)
plt.xticks(rotation=90)
plt.ylabel('Sale Price, $', fontsize=18)
plt.show()
```



13.d.7. Visualization of the number of bathrooms per house

In [252]:
```
fig = plt.figure(figsize=(20, 8))
fig = sns.barplot(x='bathrooms', y='price', data=df)
plt.title('Prices of Houses Sold in King County by number of bathrooms'
plt.xlabel("Number of bathrooms", fontsize=18)
plt.xticks(rotation=90)
plt.ylabel('Sale Price, $', fontsize=18)
plt.show()
```

```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\pandas\io\formats\forma
t.py:1403: FutureWarning: Index.ravel returning ndarray is deprecated; in a
future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\pandas\io\formats\forma
t.py:1403: FutureWarning: Index.ravel returning ndarray is deprecated; in a
future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
```



14. Conclusion

We answered the four questions by optimizing 4 multiple linear regression models. These models can give insights for specific house prediction.

The most important features to predict the sale price of the house in King County , Seattle are square footage,the gradefollowed by the location of the house.

The best performing model with 6 features is Model 1. This is a good model with 6 features, with 3 most imortant features of sqft_living, grade and latitude to predict the sale price. The mean square error of this model is 114489. Model 1 predicts home prices within $ 114,489 of the true home price on average.

The best performing multiple regression model in this project with multiple features to predict the increase in sale price is Model 4. Model 4 is the best performing model with mean absolute error of 0.35. Model 4 predicts home price within $0.35 of the true home price on average.

Other features that can increase the sale price i.e. location (which region the house is located in King County), waterfront for the house and a natural view for the house (i.e. overlooking a mountain, hill, valley, forest..ect).

Overall, according to our models to predict the price, the most precise features that can increase the sale price are: square footage of the house of the 15 closet neighbors, a house with two floors, 4 views, 1 waterfront, 5.25 bathrooms, grade 12 or 13 located in certion locations (zipcode 9814, 98011, 9802, 98033, 98102...ect).

More data in different years is needed to have better prediction in order to have better visualozations for the chage of the sale price with time. If time data is provided, we can predict if sale price increase/decrease by time and estimate the value increase or decrease of the properties with time in certain areas. This information is vital because it allows real estate investors to invest in the right properties that will have profits.

Other features are missing from the data, like school district, the education quality of the schooling system, house upgrades (wood flooring, marble countertops, wood decoration, stone facade...ect). These features can influentially affect the sale price and thir incoporation into the model in essential for future consideration.

In [ ]:

```
1
```