King County, WA, U.S.A.

1 Housing Guidance for King County, WA, U.S.A

- Student name: Vi Bui
- Student pace: Part-Time
- Scheduled project review date/time: 10/28/21
- Instructor name: Claude Fried
- Blog post URL: https://datasciish.com/ (https://datasciish.com/)

1.1 Overview [...]

Client: New WA state home buyers needing consultation on WA real estate market and expectations (price, size, location)

Data, Methodology, and Analysis: King County (WA, U.S.A.) housing data from 2014-2015

Results & Recommendations: After analyzing data and building models assessing relationships between price and square feet; price and bedrooms; and price to zip code, we've modeled the expectations for price range depending on square feet of living space, grade, condition, and renovation status

2 Data Exploration, Cleansing, Visualization, and Preparation

Data Exploration
Explore King County, WA, U.S.A. data from years 2014-2015

Data Cleansing
Check for duplicates (none); drop NaN values and unnecessary columns; continuously clean data as necessary

Data Visualization
Use visualizations to explore the data and determine how to further refine the dataset in order to prepare for modeling

Data Preparation

2.1 Data Exploration and Cleansing

Import data and all packages needed for data exploration and modeling

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

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns

import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

import scipy.stats as stats

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error, mean_squared_error

import os
import warnings
```

executed in 1.01s, finished 12:32:15 2021-11-07

Explore: columns, shape, info

In [2]:
```
df = pd.read_csv('data/kc_house_data.csv', index_col=0)
df.head()
```

executed in 50ms, finished 12:32:15 2021-11-07

Out[2]:

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

In [3]:
```
# Explore number of entries; which columns have missing data; and data type

df.info()
```

executed in 8ms, finished 12:32:15 2021-11-07

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21597 entries, 7129300520 to 1523300157
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           21597 non-null  object 
 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     19221 non-null  float64
 8   view           21534 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  object 
 13  yr_built       21597 non-null  int64  
 14  yr_renovated   17755 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: float64(8), int64(10), object(2)
memory usage: 3.5+ MB
```

In [4]:
```
# Check for duplicates

df.duplicated(keep='first').sum()
```

executed in 14ms, finished 12:32:15 2021-11-07

Out[4]:
```
0
```

In [5]:
```
# Check for NaN values

df.isna().sum()

# Columns and number of respective NaN values
# waterfront       2376
# view               63
# yr_renovated     3842
```

executed in 6ms, finished 12:32:15 2021-11-07

Out[5]:
```
date                0
price               0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront       2376
view               63
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_built            0
yr_renovated     3842
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
dtype: int64
```

In [6]:
```
# Explore columns

df.columns
```

executed in 3ms, finished 12:32:15 2021-11-07

Out[6]:
```
Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lo
t',
       'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_abov
e',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'lo
ng',
       'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

2.1.1 Understand Column Names and Descriptions for King County's Data Set

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

In [7]:
```
# Calculate the percentage of NaN

df['waterfront'].value_counts()

# Ask colleagues how to do this "smarter": (19075/(19075+146))
# (2376+19075)/(2376+19075+146)
```

executed in 3ms, finished 12:32:15 2021-11-07

Out[7]:
```
0.0    19075
1.0      146
Name: waterfront, dtype: int64
```

Observations after exploring waterfront data:

- 99.2% of houses (146 out of 19,221) do not have a waterfront view
- With 2376 entries with NaN values, imputing the NaN values to 0 makes no material difference
- Clean data: impute waterfront NaN values to 0 (represents no waterfront view)
- Resulting data: 99.3% of houses (21,451 out of 21,597) do not have a waterfront view

In [8]:
```
# Impute waterfront NaN values to 0

df['waterfront'] = df['waterfront'].fillna(0)
df['waterfront'].value_counts()
```

executed in 4ms, finished 12:32:15 2021-11-07

Out[8]:
```
0.0    21451
1.0      146
Name: waterfront, dtype: int64
```

In [9]:
```
# Double check for NaN values left

df['waterfront'].isna().sum()
```

executed in 3ms, finished 12:32:15 2021-11-07

Out[9]:
```
0
```

In [10]:
```
# Continue exploring other data that needs to be cleansed

df.info()
```

executed in 9ms, finished 12:32:15 2021-11-07

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21597 entries, 7129300520 to 1523300157
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           21597 non-null  object 
 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           21534 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  object 
 13  yr_built       21597 non-null  int64  
 14  yr_renovated   17755 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: float64(8), int64(10), object(2)
memory usage: 3.5+ MB
```

```
Write raw LaTeX or other formats here, for use with nbconvert. It will not be rendered in the notebook. When passing through nbconvert, a Raw Cell's content is added to the output unmodified.
```

In [11]:
```
df['yr_renovated'].value_counts()
```
executed in 5ms, finished 12:32:15 2021-11-07

Out[11]:
```
0.0       17011
2014.0       73
2003.0       31
2013.0       31
2007.0       30
            ...
1946.0        1
1959.0        1
1971.0        1
1951.0        1
1954.0        1
Name: yr_renovated, Length: 70, dtype: int64
```

'yr_renovated' data needs to be cleansed. Observations about 'yr_renovated':

- 'yr_renovated' has 3842 NaN values
- About the data: if house has been renovated, the year is entered. If not, 0 has been entered
- 95.8% of current data set (17,011 of 17,755 houses) have not been renovated
- Imputing the 3842 NaN values to 0 (not renovated) does not make a substantial difference
- Resulting data: 96.6% of new data set (20,853 of 21,597 houses) have not been renovated

In [12]:
```
df['yr_renovated'] = df['yr_renovated'].fillna(0)
df['yr_renovated'].value_counts()
```
executed in 5ms, finished 12:32:15 2021-11-07

Out[12]:
```
0.0       20853
2014.0       73
2003.0       31
2013.0       31
2007.0       30
            ...
1946.0        1
1959.0        1
1971.0        1
1951.0        1
1954.0        1
Name: yr_renovated, Length: 70, dtype: int64
```

In [13]:
```
# ask colleagues how you would get the sum of the value counts
# df['yr_renovated'].value_counts('0')
20853/21597
```
executed in 3ms, finished 12:32:15 2021-11-07

Out[13]:
```
0.9655507709404084
```

In [14]:
```
df.head()
```
executed in 12ms, finished 12:32:15 2021-11-07

Out[14]:

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

'sqft_basement' data needs to be cleansed. Observations about 'sqft_basement':

- Ran into an error with 'sqft_basement' data
- Found there are 454 entries with '?' symbols in the data
- 60.7% of current data set (12,826 of 21,143 houses) have 0 as entered for sqft_basement
- Imputing the 454 '?' entries to 0 does not make a substantial difference
- Resulting data: 61.5% of new data set (13,280 of 21,597 houses) have 0 sqft_basement

In [15]:
```
# Check how many entries for 'sqft_basement' are '?'
# Ask colleagues how to view the entire data set for sqft_basement

df['sqft_basement'].value_counts()
```
executed in 6ms, finished 12:32:15 2021-11-07

Out[15]:
```
0.0       12826
?           454
600.0       217
500.0       209
700.0       208
           ...
2240.0        1
2850.0        1
1135.0        1
3000.0        1
508.0         1
Name: sqft_basement, Length: 304, dtype: int64
```

In [16]:
```
# Impute 454 '?' entries to 0 values
# Transform data type from object to float

df['sqft_basement'] = df['sqft_basement'].apply(lambda x: 0 if x == '?' els
```
executed in 6ms, finished 12:32:15 2021-11-07

In [17]:
```
df.info()
```
executed in 8ms, finished 12:32:15 2021-11-07

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21597 entries, 7129300520 to 1523300157
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           21597 non-null  object 
 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           21534 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: float64(9), int64(10), object(1)
memory usage: 3.5+ MB
```

Continue cleaning data/transform data types:

- Transform data types
- Most importantly, convert zipcode from integer to string

In [18]:
```
# yr_renovated from float to integer (preference)
# zipcode from integer to string

df['yr_renovated'] = (df['yr_renovated'].astype(int))
df['zipcode'] = (df['zipcode'].astype(str))
```
executed in 16ms, finished 12:32:15 2021-11-07

In [19]:
```
df.info()
```

executed in 9ms, finished 12:32:15 2021-11-07

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21597 entries, 7129300520 to 1523300157
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           21597 non-null  object 
 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           21534 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  int64  
 15  zipcode        21597 non-null  object 
 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: float64(8), int64(10), object(2)
memory usage: 3.5+ MB
```

2.2 Create New Features

1. Year Sold (from date column)
2. Renovated (make renovated a binary value: renovated = 1; not renovated = 0)
3. Basement Present (make basement a binary value: renovated = 1; not renovated = 0)
4. Actual Age of Property (year sold - year built)
5. Bathrooms Per Bedroom (bathrooms/bedrooms)
6. Square Feet Living to Square Foot Lot (sqft_living/sqft_lot)

In [20]:
```
# Create new features

df['yr_sold'] = (df['date'].str[-4:].astype(int))
df['renovated'] = np.where(df['yr_renovated']!=0, 1,0)
df['basement_present'] = np.where(df['sqft_basement']!=0, 1,0)
df['actual_age_of_property'] = df['yr_sold']-df['yr_built']
df['bathrooms_per_bedroom'] = df['bathrooms']/df['bedrooms']
df['sqft_living_to_sqft_lot'] = df['sqft_living']/df['sqft_lot']
df.head()
```

executed in 27ms, finished 12:32:16 2021-11-07

Out[20]:

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

5 rows × 26 columns

In [21]:
```
# Check: data types
# Check: all value counts match

df.info()
```

executed in 11ms, finished 12:32:16 2021-11-07

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21597 entries, 7129300520 to 1523300157
Data columns (total 26 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   date                     21597 non-null  object 
 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                     21534 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  int64  
 15  zipcode                  21597 non-null  object 
 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  
 20  yr_sold                  21597 non-null  int64  
 21  renovated                21597 non-null  int64  
 22  basement_present         21597 non-null  int64  
 23  actual_age_of_property   21597 non-null  int64  
 24  bathrooms_per_bedroom    21597 non-null  float64
 25  sqft_living_to_sqft_lot  21597 non-null  float64
dtypes: float64(10), int64(14), object(2)
memory usage: 4.4+ MB
```

In [22]:
```
df.columns
```

executed in 3ms, finished 12:32:16 2021-11-07

Out[22]:
```
Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lo
t',
       'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_abov
e',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'lo
ng',
       'sqft_living15', 'sqft_lot15', 'yr_sold', 'renovated',
       'basement_present', 'actual_age_of_property', 'bathrooms_per_bedro
om',
       'sqft_living_to_sqft_lot'],
      dtype='object')
```

In [23]:
```
# Explore correlation for numerical values with .corr()

df.corr()
```

executed in 36ms, finished 12:32:16 2021-11-07

Out[23]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfro
price	1.000000	0.308787	0.525906	0.701917	0.089876	0.256804	0.2643
bedrooms	0.308787	1.000000	0.514508	0.578212	0.032471	0.177944	-0.0021
bathrooms	0.525906	0.514508	1.000000	0.755758	0.088373	0.502582	0.0636
sqft_living	0.701917	0.578212	0.755758	1.000000	0.173453	0.353953	0.1046
sqft_lot	0.089876	0.032471	0.088373	0.173453	1.000000	-0.004814	0.0214
floors	0.256804	0.177944	0.502582	0.353953	-0.004814	1.000000	0.0207
waterfront	0.264306	-0.002127	0.063629	0.104637	0.021459	0.020797	1.0000
view	0.395734	0.078523	0.186451	0.282532	0.075298	0.028436	0.3820
condition	0.036056	0.026496	-0.126479	-0.059445	-0.008830	-0.264075	0.0166
grade	0.667951	0.356563	0.665838	0.762779	0.114731	0.458794	0.0828
sqft_above	0.605368	0.479386	0.686668	0.876448	0.184139	0.523989	0.0717
sqft_basement	0.321108	0.297229	0.278485	0.428660	0.015031	-0.241866	0.0830
yr_built	0.053953	0.155670	0.507173	0.318152	0.052946	0.489193	-0.0244
yr_renovated	0.117855	0.017900	0.047177	0.051060	0.004979	0.003793	0.0739
lat	0.306692	-0.009951	0.024280	0.052155	-0.085514	0.049239	-0.0121
long	0.022036	0.132054	0.224903	0.241214	0.230227	0.125943	-0.0376
sqft_living15	0.585241	0.393406	0.569884	0.756402	0.144763	0.280102	0.0838
sqft_lot15	0.082845	0.030690	0.088303	0.184342	0.718204	-0.010722	0.0306
yr_sold	0.003727	-0.009949	-0.026577	-0.029014	0.005628	-0.022352	-0.0050
renovated	0.117543	0.017635	0.046742	0.050829	0.005091	0.003713	0.0742
basement_present	0.178264	0.158412	0.159863	0.201198	-0.034889	-0.252465	0.0392
actual_age_of_property	-0.053890	-0.155817	-0.507561	-0.318592	-0.052853	-0.489514	0.0244
bathrooms_per_bedroom	0.281227	-0.236129	0.652668	0.310690	0.063306	0.421169	0.0737
sqft_living_to_sqft_lot	0.123063	0.026798	0.287015	0.076988	-0.252601	0.556700	-0.0298

24 rows × 24 columns

- Explore descriptive statistics with .describe()
- Summarizes central tendency (mean), dispersion and shape of a dataset's distribution, excluding NaN values

In [24]:
```
# Explore descriptive statistics with .describe()
# Summarizes central tendency (mean), dispersion and shape of a dataset's d

df.describe()
```

executed in 56ms, finished 12:32:16 2021-11-07

Out[24]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
count	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	21597.000000	21597.
mean	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	1.494096	0.
std	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	0.539683	0.
min	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.000000	0.
25%	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1.000000	0.
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	2.000000	0.
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.

8 rows × 24 columns

In [25]:
```
# Explore distribution (value_counts) of bedroom data

df['bedrooms'].value_counts()
```

executed in 4ms, finished 12:32:16 2021-11-07

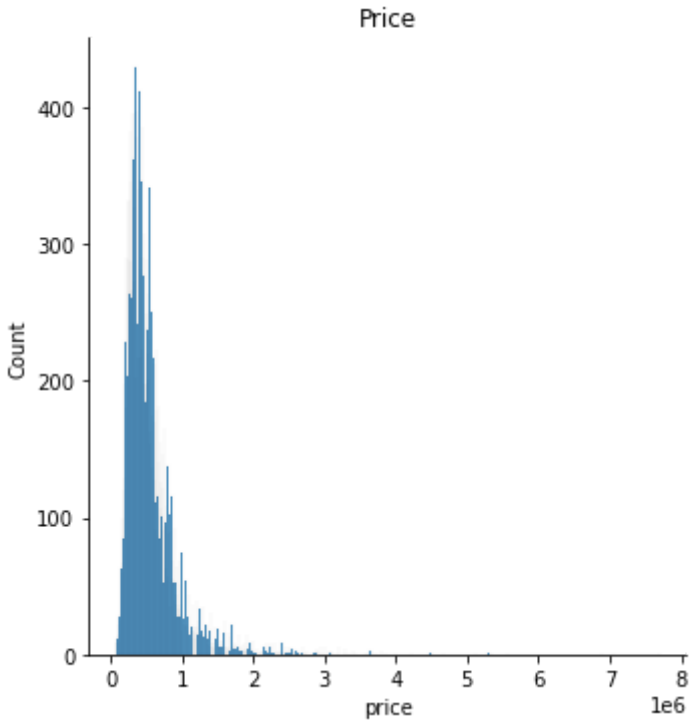
Out[25]:
```
3     9824
4     6882
2     2760
5     1601
6      272
1      196
7       38
8       13
9        6
10       3
11       1
33       1
Name: bedrooms, dtype: int64
```

2.3 Data Visualization

In [26]:
```
plt.figure(figsize=(12,8))
sns.displot(df['price'],bins=1000)
plt.title('Price')
plt.show();
```

executed in 1.34s, finished 12:32:17 2021-11-07

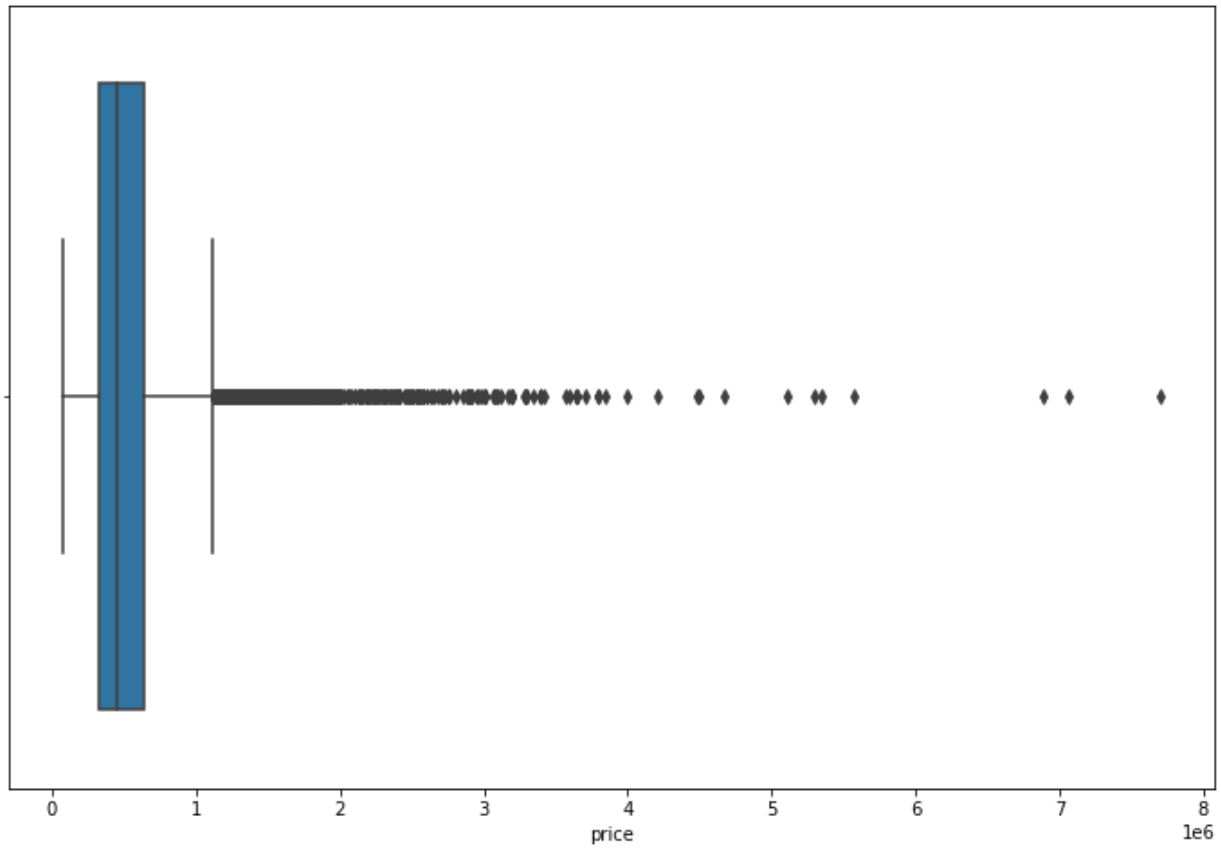
```
<Figure size 864x576 with 0 Axes>
```



In [27]:
```
fig, ax = plt.subplots(figsize=(12,8))
sns.boxplot(x='price', data=df, ax=ax)
```

executed in 154ms, finished 12:32:17 2021-11-07

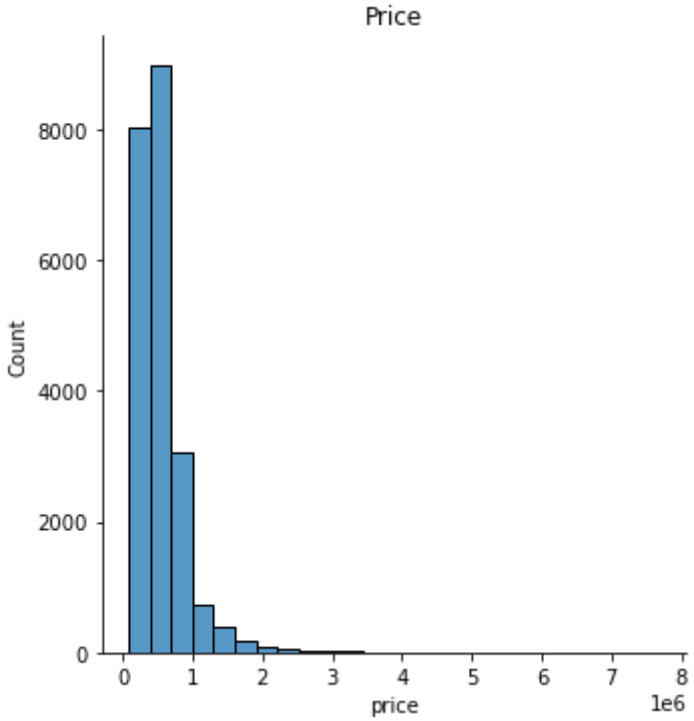
Out[27]: <AxesSubplot:xlabel='price'>



In [28]:
```
plt.figure(figsize=(12,8))
sns.displot(df['price'],bins=25)
plt.title('Price')
plt.show();
```

executed in 177ms, finished 12:32:17 2021-11-07

```
<Figure size 864x576 with 0 Axes>
```



In [29]:
```
df.describe()
```
executed in 55ms, finished 12:32:17 2021-11-07

Out[29]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
count	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	21597.000000	21597.
mean	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	1.494096	0.
std	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	0.539683	0.
min	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.000000	0.
25%	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1.000000	0.
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	2.000000	0.
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.

8 rows × 24 columns

Narrow price range to $175,000-$650,000

In [30]:
```
# Ask colleagues how to find where the "majority" of prices fall

df = df[df['price'].between(175_000,650_000)]
```
executed in 6ms, finished 12:32:17 2021-11-07

In [31]:
```
df.info()
```
executed in 8ms, finished 12:32:17 2021-11-07

```
 9   condition              15993 non-null  int64
 10  grade                  15993 non-null  int64
 11  sqft_above             15993 non-null  int64
 12  sqft_basement          15993 non-null  float64
 13  yr_built               15993 non-null  int64
 14  yr_renovated           15993 non-null  int64
 15  zipcode                15993 non-null  object
 16  lat                    15993 non-null  float64
 17  long                   15993 non-null  float64
 18  sqft_living15          15993 non-null  int64
 19  sqft_lot15             15993 non-null  int64
 20  yr_sold                15993 non-null  int64
 21  renovated              15993 non-null  int64
 22  basement_present       15993 non-null  int64
 23  actual_age_of_property 15993 non-null  int64
 24  bathrooms_per_bedroom  15993 non-null  float64
 25  sqft_living_to_sqft_lot 15993 non-null  float64
dtypes: float64(10), int64(14), object(2)
memory usage: 3.3+ MB
```

In [32]:
```
# Percentage of data that will be used

15_993/21_597
```

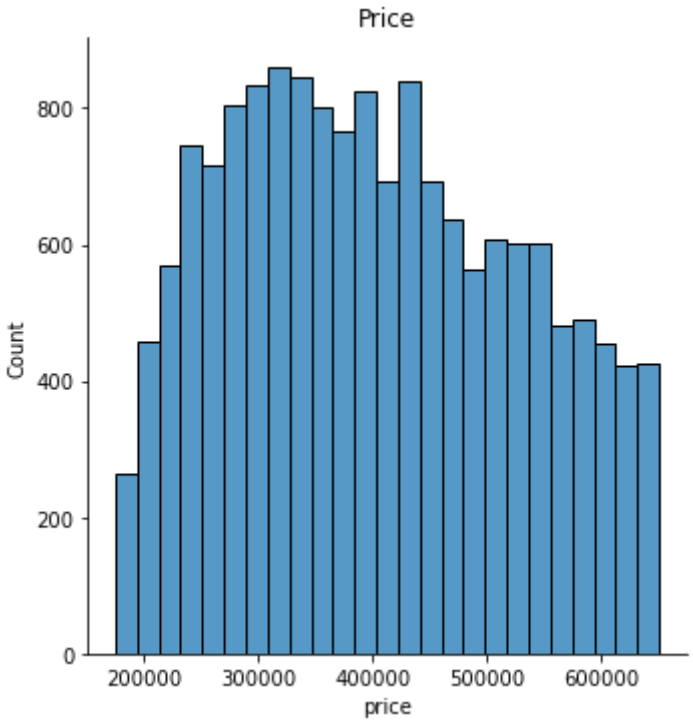
executed in 3ms, finished 12:32:17 2021-11-07

Out[32]:
```
0.7405195165995277
```

In [33]:
```
plt.figure(figsize=(12,8))
sns.displot(df['price'],bins=25)
plt.title('Price')
plt.show();
```

executed in 153ms, finished 12:32:18 2021-11-07

```
<Figure size 864x576 with 0 Axes>
```



In [34]:
```
# Explore the data - specifically bedrooms, bathrooms, and sqft_living

df.describe()
```

executed in 53ms, finished 12:32:18 2021-11-07

Out[34]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
count	15993.000000	15993.000000	15993.000000	15993.000000	1.599300e+04	15993.000000	1599
mean	400987.591009	3.244982	1.953667	1800.124742	1.317614e+04	1.431533	(
std	123893.841894	0.886582	0.660056	631.746037	3.310214e+04	0.537112	(
min	175000.000000	1.000000	0.500000	370.000000	5.720000e+02	1.000000	(
25%	299950.000000	3.000000	1.500000	1330.000000	5.000000e+03	1.000000	(
50%	392000.000000	3.000000	2.000000	1720.000000	7.439000e+03	1.000000	(
75%	499990.000000	4.000000	2.500000	2180.000000	9.968000e+03	2.000000	(
max	650000.000000	33.000000	7.500000	5461.000000	1.164794e+06	3.500000	

8 rows × 24 columns

In [35]:
```
# Explore correlation after narrowing data set

df.corr()
```

executed in 33ms, finished 12:32:18 2021-11-07

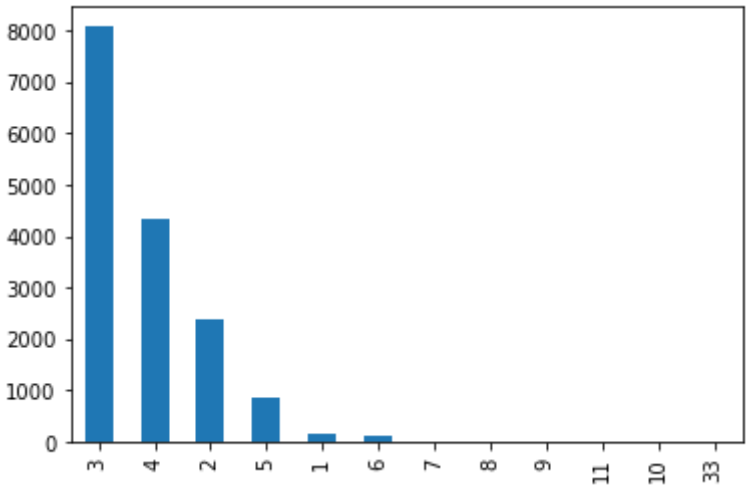
Out[35]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfr
price	1.000000	0.176433	0.315515	0.417273	0.071915	0.200133	0.0227
bedrooms	0.176433	1.000000	0.456624	0.589010	0.020603	0.102590	-0.0384
bathrooms	0.315515	0.456624	1.000000	0.666310	0.023810	0.492458	-0.0292
sqft_living	0.417273	0.589010	0.666310	1.000000	0.132353	0.268110	-0.0116
sqft_lot	0.071915	0.020603	0.023810	0.132353	1.000000	-0.052962	0.021(
floors	0.200133	0.102590	0.492458	0.268110	-0.052962	1.000000	-0.0151
waterfront	0.022750	-0.038404	-0.029209	-0.011641	0.021022	-0.015131	1.000(
view	0.126625	0.008418	0.038959	0.109977	0.105113	-0.030150	0.2654
condition	-0.003818	0.020562	-0.161222	-0.076913	0.015127	-0.291827	0.0204
grade	0.451436	0.256539	0.557766	0.586257	0.034926	0.424865	-0.0216
sqft_above	0.318006	0.449544	0.589683	0.811530	0.127287	0.493237	-0.0176
sqft_basement	0.196807	0.276462	0.188410	0.397274	0.020780	-0.319435	0.0087
yr_built	0.040321	0.162717	0.593987	0.353469	0.005088	0.547670	-0.0366
yr_renovated	0.027449	-0.008372	-0.012700	-0.001082	0.018037	-0.021295	0.0477
lat	0.479098	-0.107569	-0.112134	-0.143029	-0.107369	-0.017537	-0.0362
long	0.054300	0.132547	0.232980	0.256092	0.217447	0.105360	-0.0555
sqft_living15	0.382913	0.342631	0.478078	0.682108	0.152715	0.204379	0.0002
sqft_lot15	0.067769	0.015451	0.024263	0.143713	0.712409	-0.057095	0.0464
yr_sold	0.007087	-0.009127	-0.027191	-0.024446	-0.006834	-0.018975	-0.005(
renovated	0.027333	-0.008531	-0.013034	-0.001096	0.018080	-0.021294	0.0477
basement_present	0.174533	0.138967	0.121556	0.200849	-0.020799	-0.292454	0.0093
actual_age_of_property	-0.040202	-0.162862	-0.594414	-0.353860	-0.005201	-0.547963	0.0365
bathrooms_per_bedroom	0.184907	-0.306894	0.638663	0.181322	0.005793	0.435363	0.0068
sqft_living_to_sqft_lot	0.188304	-0.024121	0.328013	0.050878	-0.257394	0.610488	-0.0319

24 rows × 24 columns

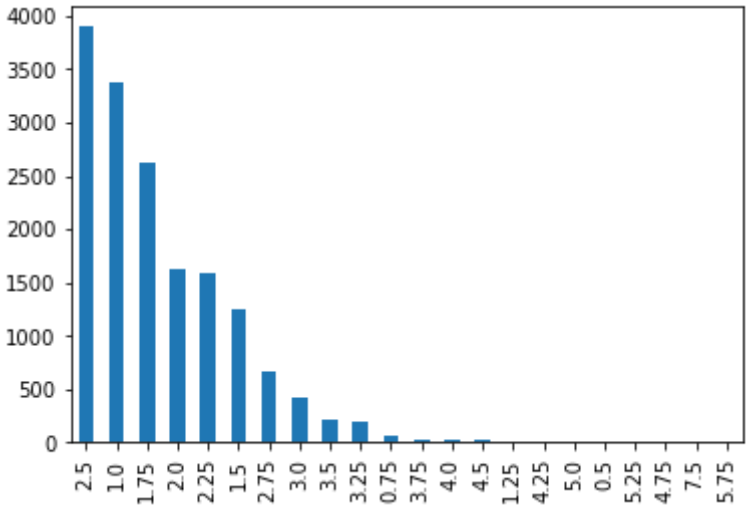
In [36]:
```
df['bedrooms'].value_counts().plot(kind='bar')
sns.despine;
```

executed in 139ms, finished 12:32:18 2021-11-07



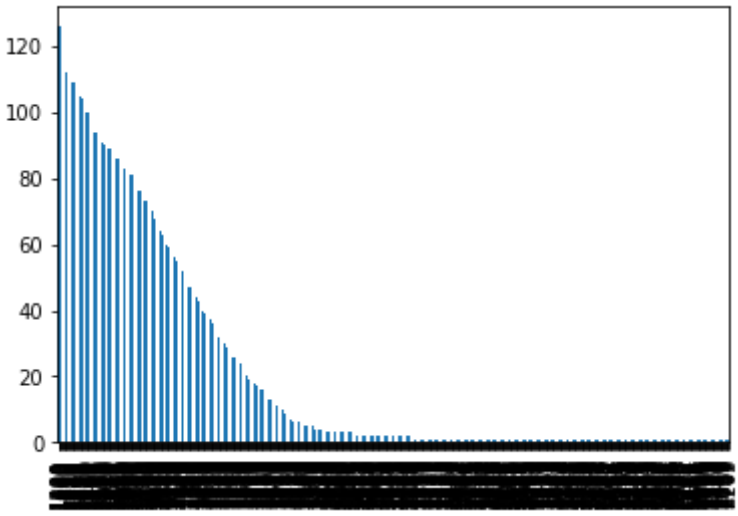
In [37]:
```
df['bathrooms'].value_counts().plot(kind='bar')
sns.despine;
```

executed in 156ms, finished 12:32:18 2021-11-07



In [38]:
```
df['sqft_living'].value_counts().plot(kind='bar')
sns.despine;
```

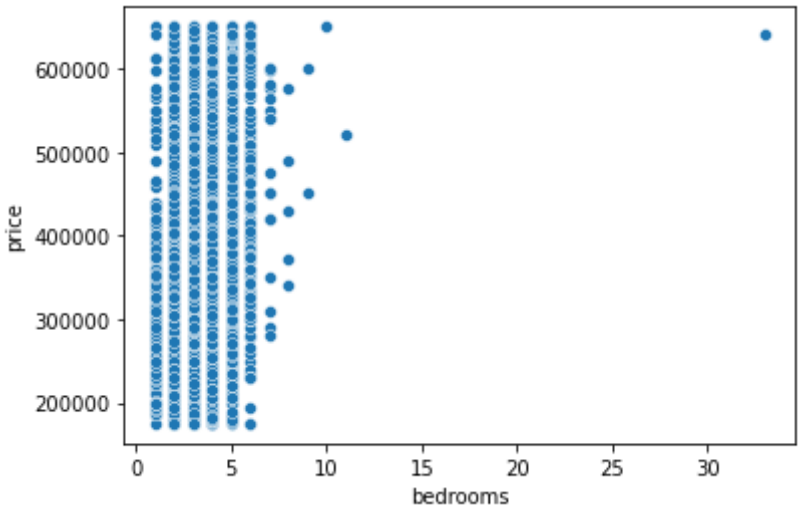
executed in 7.97s, finished 12:32:26 2021-11-07



In [39]:
```
sns.scatterplot(data=df, x='bedrooms', y='price')
```

executed in 130ms, finished 12:32:26 2021-11-07

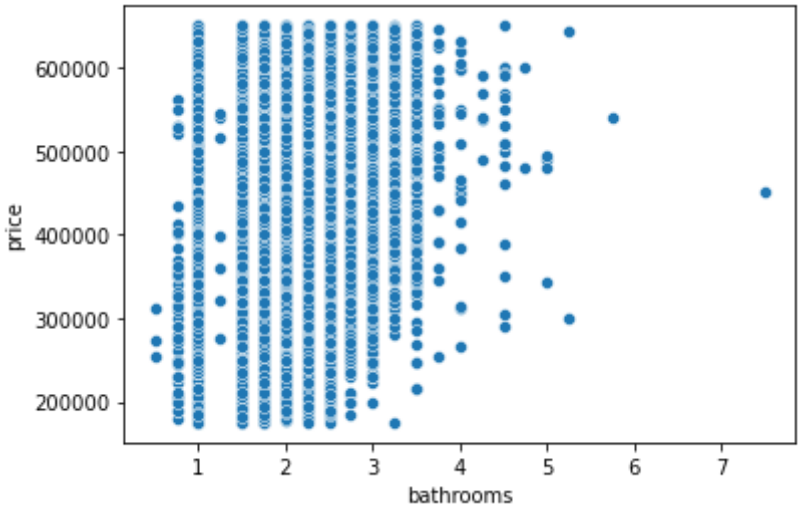
Out[39]: <AxesSubplot:xlabel='bedrooms', ylabel='price'>



In [40]:
```
sns.scatterplot(data=df, x='bathrooms', y='price')
```

executed in 124ms, finished 12:32:26 2021-11-07

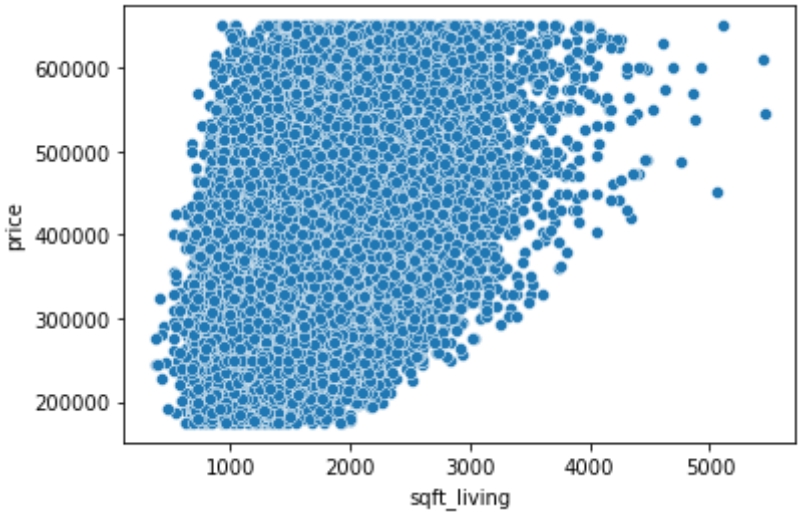
Out[40]: <AxesSubplot:xlabel='bathrooms', ylabel='price'>



In [41]:
```
sns.scatterplot(data=df, x='sqft_living', y='price')
```

executed in 127ms, finished 12:32:26 2021-11-07

Out[41]: <AxesSubplot:xlabel='sqft_living', ylabel='price'>



After seeing outliers in the data, refine data set to:

1. Bedrooms to 6 or less

2. sqft_living to 4000 or less

In [42]:
```
df.describe()
```
executed in 62ms, finished 12:32:26 2021-11-07

Out[42]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
count	15993.000000	15993.000000	15993.000000	15993.000000	1.599300e+04	15993.000000	1599
mean	400987.591009	3.244982	1.953667	1800.124742	1.317614e+04	1.431533	(
std	123893.841894	0.886582	0.660056	631.746037	3.310214e+04	0.537112	(
min	175000.000000	1.000000	0.500000	370.000000	5.720000e+02	1.000000	(
25%	299950.000000	3.000000	1.500000	1330.000000	5.000000e+03	1.000000	(
50%	392000.000000	3.000000	2.000000	1720.000000	7.439000e+03	1.000000	(
75%	499990.000000	4.000000	2.500000	2180.000000	9.968000e+03	2.000000	(
max	650000.000000	33.000000	7.500000	5461.000000	1.164794e+06	3.500000	

8 rows × 24 columns

In [43]:
```
df = df[df['bedrooms'] <= 6]
df.describe()
```
executed in 65ms, finished 12:32:26 2021-11-07

Out[43]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
count	15965.000000	15965.000000	15965.000000	15965.000000	1.596500e+04	15965.00000	15965.
mean	400835.351519	3.235766	1.951425	1797.901973	1.318139e+04	1.43135	0.
std	123867.266044	0.835528	0.656015	629.416330	3.312958e+04	0.53716	0.
min	175000.000000	1.000000	0.500000	370.000000	5.720000e+02	1.00000	0.
25%	299900.000000	3.000000	1.500000	1330.000000	5.000000e+03	1.00000	0.
50%	392000.000000	3.000000	2.000000	1720.000000	7.434000e+03	1.00000	0.
75%	499950.000000	4.000000	2.500000	2180.000000	9.966000e+03	2.00000	0.
max	650000.000000	6.000000	5.250000	5461.000000	1.164794e+06	3.50000	1.

8 rows × 24 columns

In [44]:
```
df = df[df['sqft_living'] <= 4000]
df.describe()
```

executed in 63ms, finished 12:32:26 2021-11-07

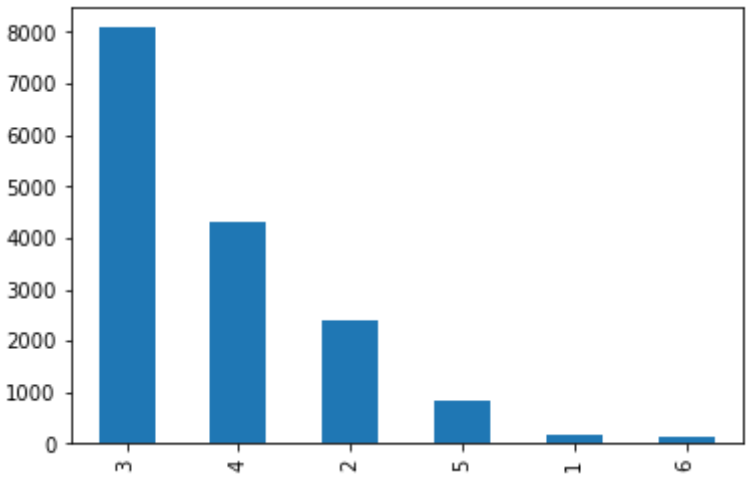
Out[44]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
count	15916.000000	15916.000000	15916.000000	15916.000000	1.591600e+04	15916.000000	15916
mean	400358.968145	3.232093	1.947427	1789.922971	1.310122e+04	1.430227	(
std	123707.148968	0.833128	0.652116	613.395670	3.301546e+04	0.536934	(
min	175000.000000	1.000000	0.500000	370.000000	5.720000e+02	1.000000	(
25%	299725.000000	3.000000	1.500000	1330.000000	5.000000e+03	1.000000	(
50%	390000.000000	3.000000	2.000000	1720.000000	7.420000e+03	1.000000	(
75%	499900.000000	4.000000	2.500000	2180.000000	9.936000e+03	2.000000	(
max	650000.000000	6.000000	5.250000	4000.000000	1.164794e+06	3.500000	

8 rows × 24 columns

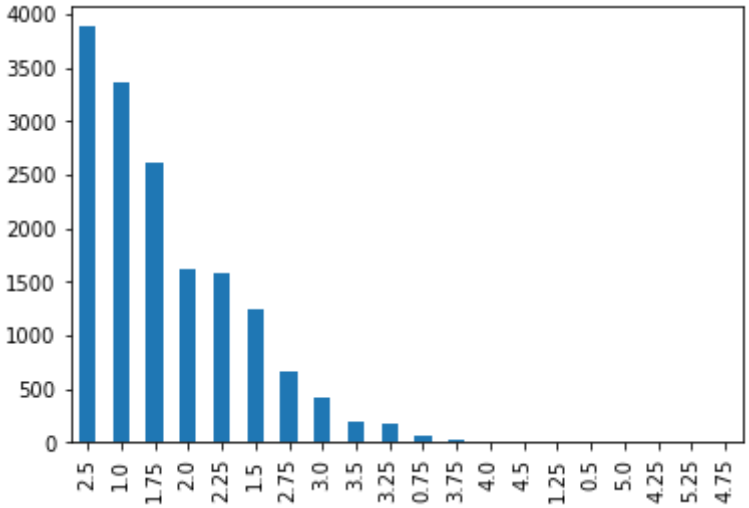
In [45]:
```
df['bedrooms'].value_counts().plot(kind='bar')
sns.despine;
```

executed in 99ms, finished 12:32:27 2021-11-07



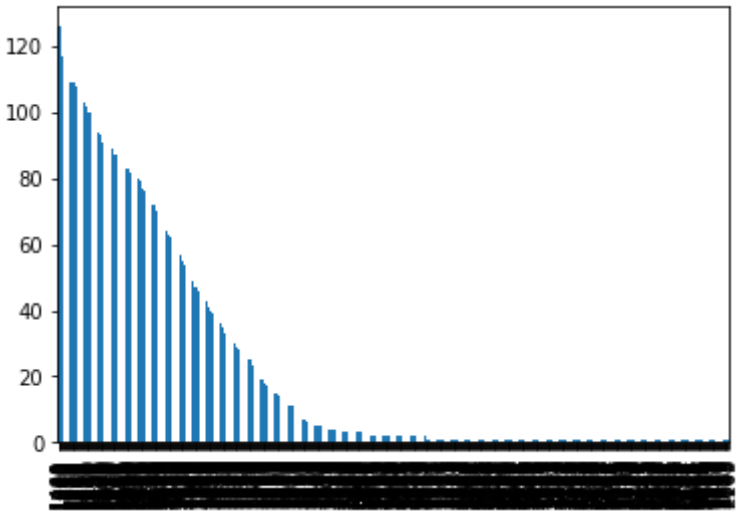
In [46]:
```
df['bathrooms'].value_counts().plot(kind='bar')
sns.despine;
```

executed in 161ms, finished 12:32:27 2021-11-07



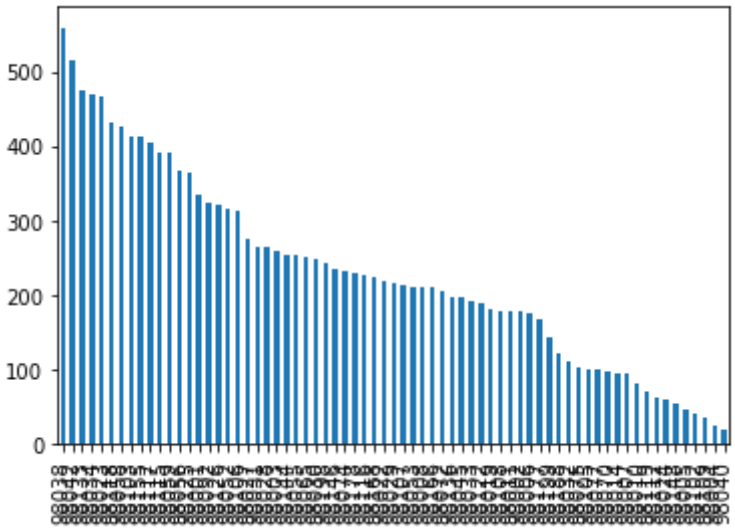
In [47]:
```
df['sqft_living'].value_counts().plot(kind='bar')
sns.despine;
```

executed in 7.55s, finished 12:32:34 2021-11-07



In [48]:
```
df['zipcode'].value_counts().plot(kind='bar')
sns.despine;
```

executed in 799ms, finished 12:32:35 2021-11-07



In [49]:
```
#pd.set_option("display.max_rows", None, "display.max_columns", None)
```

executed in 2ms, finished 12:32:35 2021-11-07

In [50]:
```
df['zipcode'].value_counts(ascending=True)
```

executed in 5ms, finished 12:32:35 2021-11-07

Out[50]:
```
98040     19
98004     23
98109     35
98102     40
98005     47
         ...
98023    466
98034    469
98133    475
98042    516
98038    559
Name: zipcode, Length: 69, dtype: int64
```

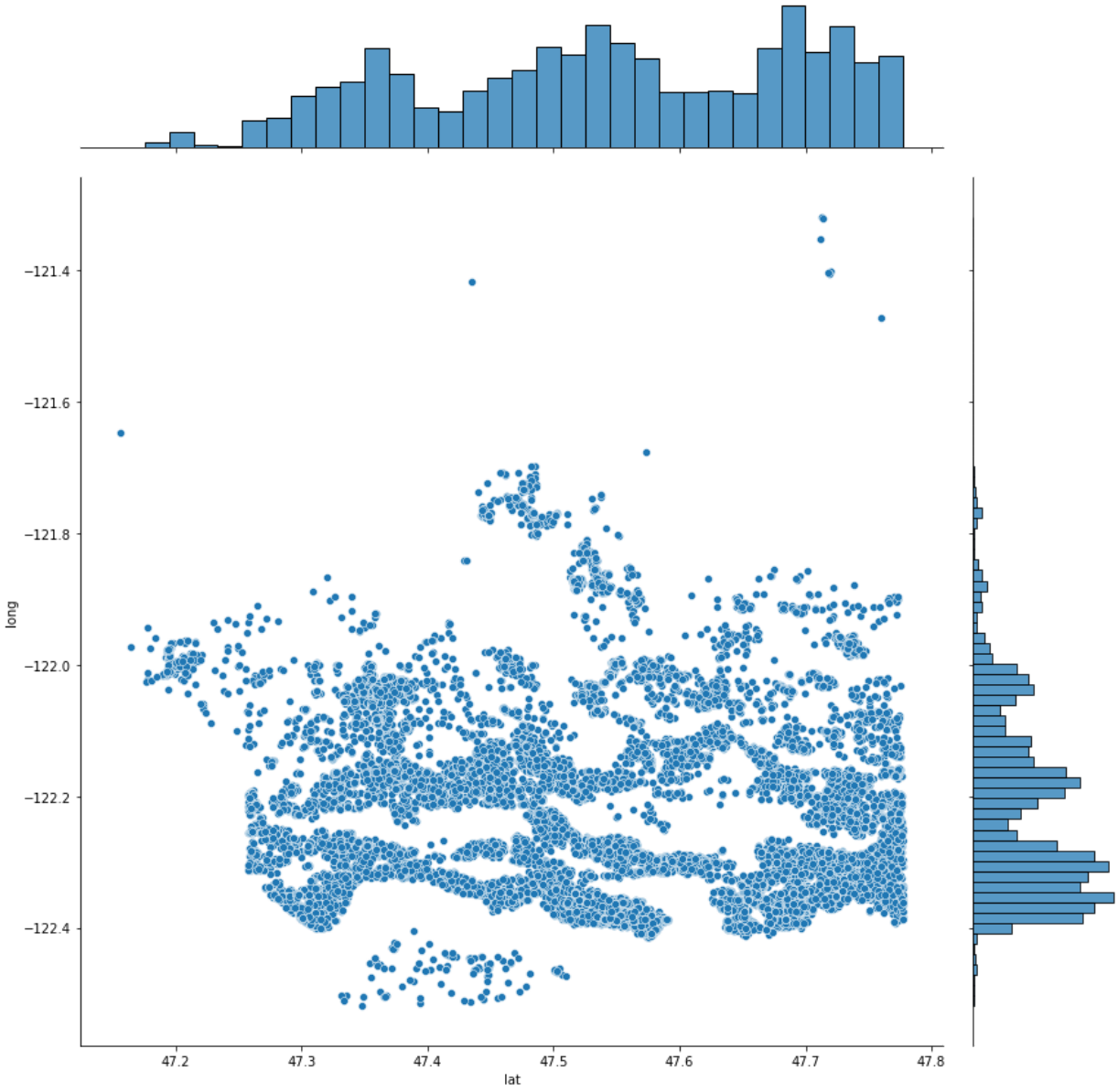
In [51]:
```
# For Future Work: explore correlation between these zip codes and price
# least_houses_zip = df[df['zipcode' == '98004', '98109', '98112', '98102',
# most_houses_zips = 98023, 98034, 98133, 98042, 98038
```

executed in 1ms, finished 12:32:35 2021-11-07

In [52]:
```
# For future work on zipcodes

plt.figure(figsize=(12,12))
sns.jointplot(x=df['lat'], y=df['long'], size=12)
plt.xlabel('Latitude', fontsize=11)
plt.ylabel('Longitude', fontsize=11)
plt.show()
sns.despine;
```

executed in 474ms, finished 12:32:36 2021-11-07

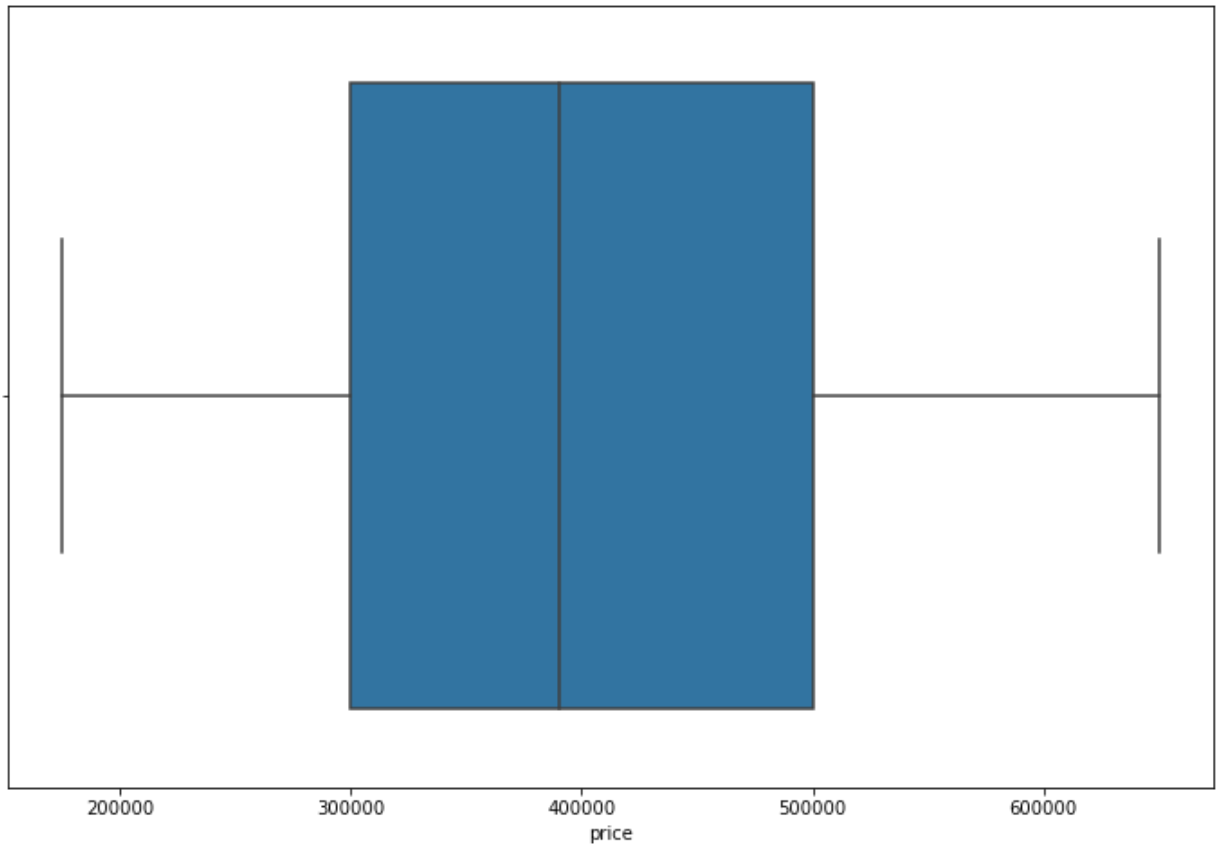
```
/Users/v/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/seabor
n/axisgrid.py:2015: UserWarning: The `size` parameter has been renamed to
`height`; please update your code.
  warnings.warn(msg, UserWarning)

<Figure size 864x864 with 0 Axes>
```



In [53]:
```
# Boxplot for price

fig, ax = plt.subplots(figsize=(12,8))
sns.boxplot(x='price', data=df, ax=ax)
```

executed in 80ms, finished 12:32:36 2021-11-07

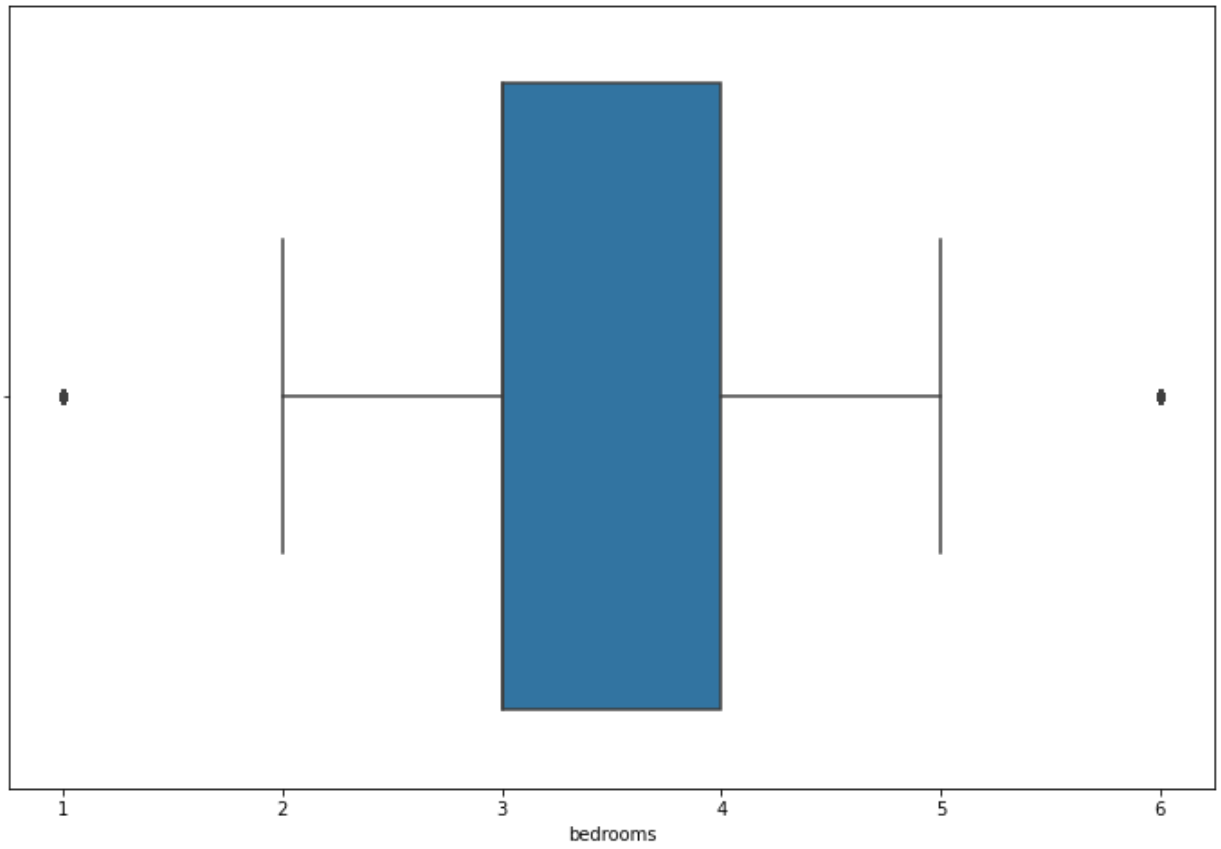
Out[53]: <AxesSubplot:xlabel='price'>



In [54]:
```
# Boxplot for bedrooms

fig, ax = plt.subplots(figsize=(12,8))
sns.boxplot(x='bedrooms', data=df, ax=ax)
```

executed in 196ms, finished 12:32:36 2021-11-07

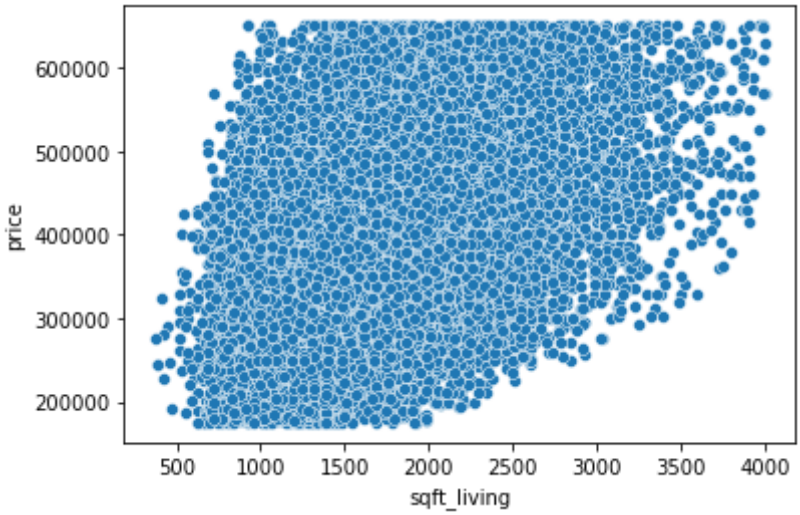
Out[54]:
```
<AxesSubplot:xlabel='bedrooms'>
```



In [55]:
```
# Scatterplot for sqft_living and price

sns.scatterplot(data=df, x='sqft_living', y='price')
```

executed in 136ms, finished 12:32:36 2021-11-07

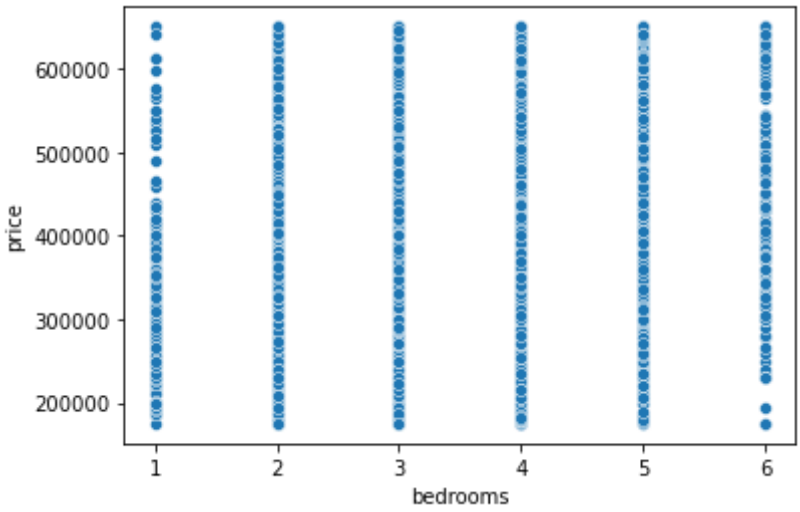
Out[55]: <AxesSubplot:xlabel='sqft_living', ylabel='price'>



In [56]:
```
# Scatterplot for bedrooms and price

sns.scatterplot(data=df, x='bedrooms', y='price')
```

executed in 127ms, finished 12:32:36 2021-11-07

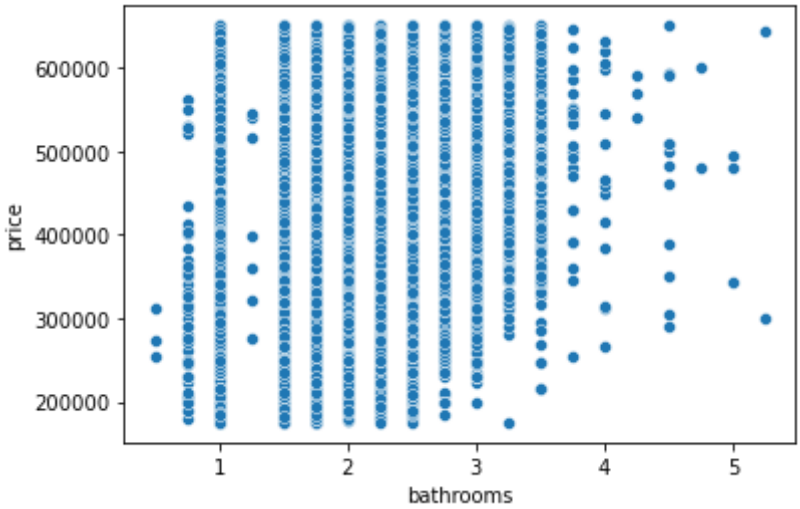
Out[56]: <AxesSubplot:xlabel='bedrooms', ylabel='price'>



In [57]:
```
# Scatterplot for bathrooms and price

sns.scatterplot(data=df, x='bathrooms', y='price')
```

executed in 124ms, finished 12:32:36 2021-11-07

Out[57]: <AxesSubplot:xlabel='bathrooms', ylabel='price'>



2.4 Data Preparation

Start dropping columns that will not be used

1. 'view'
2. 'date'

In [58]:
```
# Will not use 'view' (# of times the house has been viewed) for analysis

if 'view' in df.columns:
    df.drop('view', axis=1, inplace=True)
```

executed in 4ms, finished 12:32:36 2021-11-07

In [59]:
```
# Drop date

if 'date' in df.columns:
    df.drop('date', axis=1, inplace=True)
```

executed in 4ms, finished 12:32:36 2021-11-07

2.4.1 Create Target and Explore Data with More Visualizations

TARGET is price

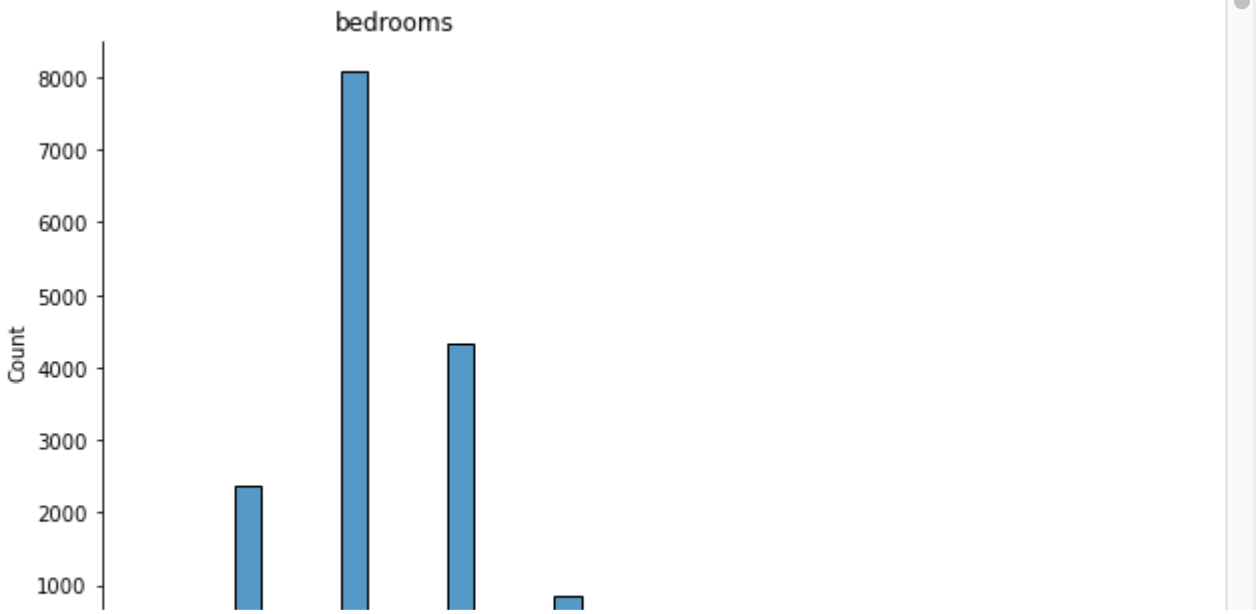
In [60]:
```
# Target is price
# X values is everything else

TARGET = 'price'
X_VALS = [c for c in df.columns if c != TARGET]
TARGET in X_VALS
```

executed in 3ms, finished 12:32:36 2021-11-07

Out[60]: False

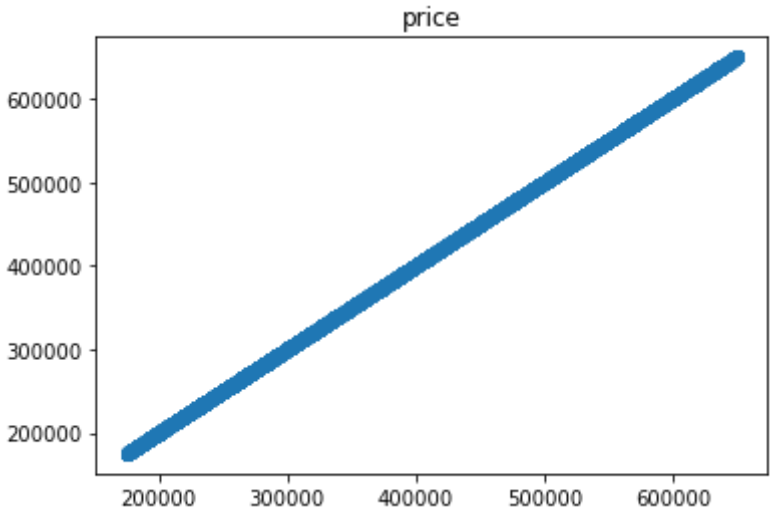
In [61]:
```
for col in df.columns:
    plt.figure(figsize=(12,8))
    sns.displot(df[col],bins=20)
    plt.title(col)
    plt.show();
```

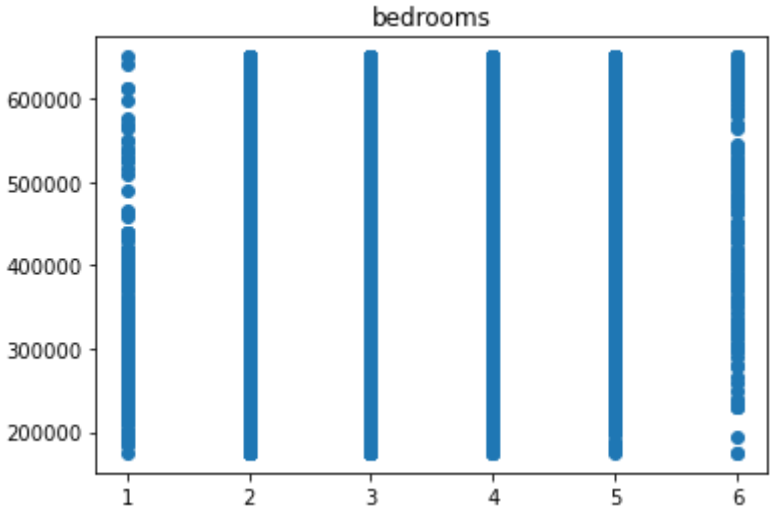
executed in 4.98s, finished 12:32:41 2021-11-07

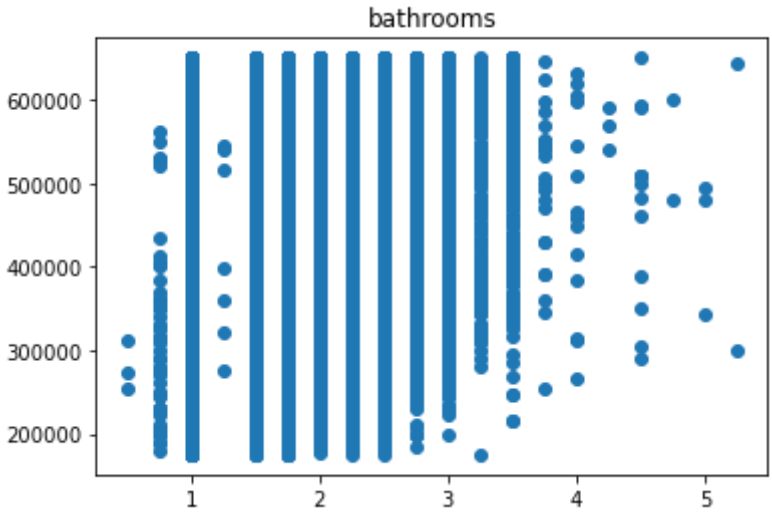


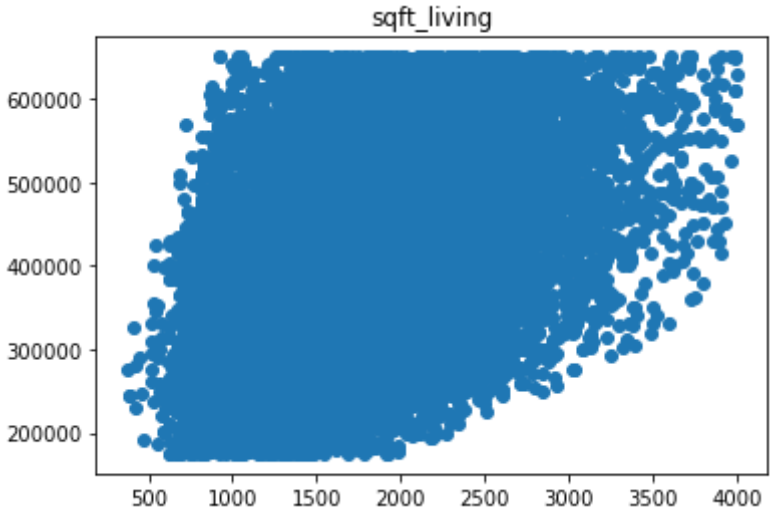
In [62]:
```
for col in df.columns:
    plt.scatter(df[col], df[TARGET])
    plt.title(col)
    plt.show()
```

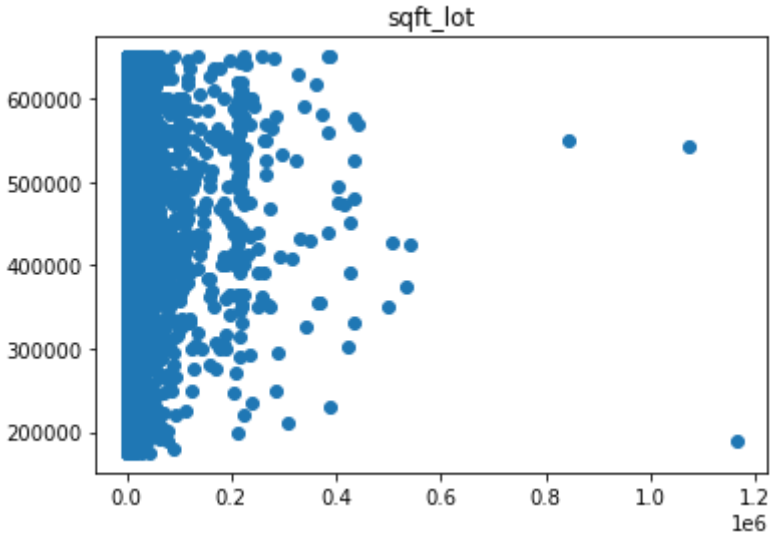
executed in 3.05s, finished 12:32:44 2021-11-07

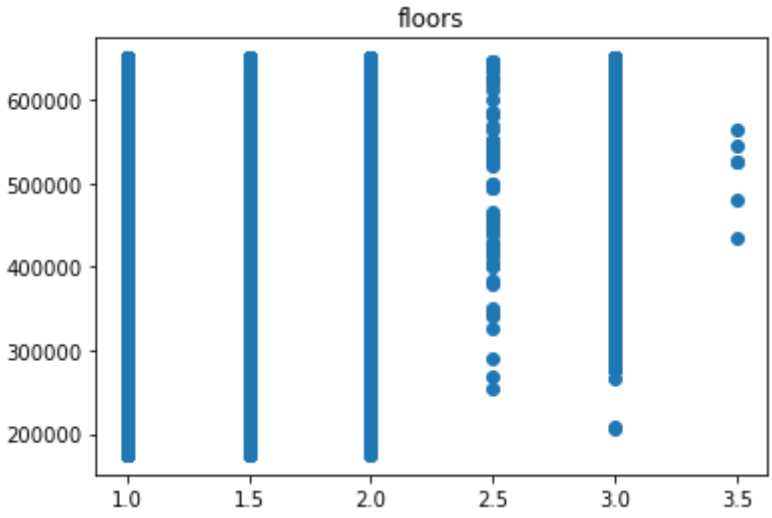


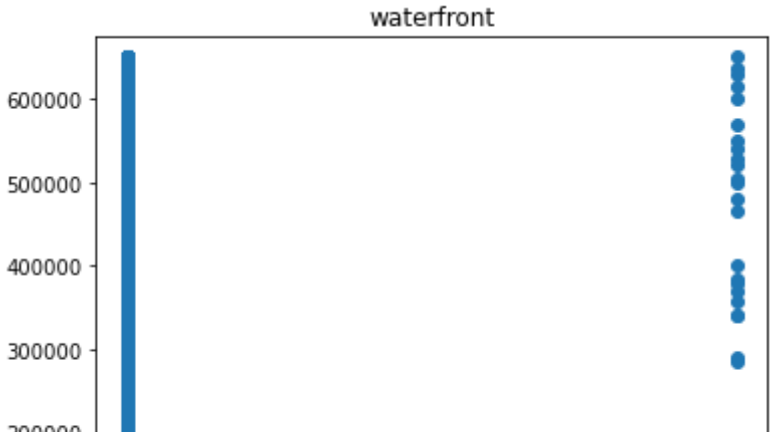


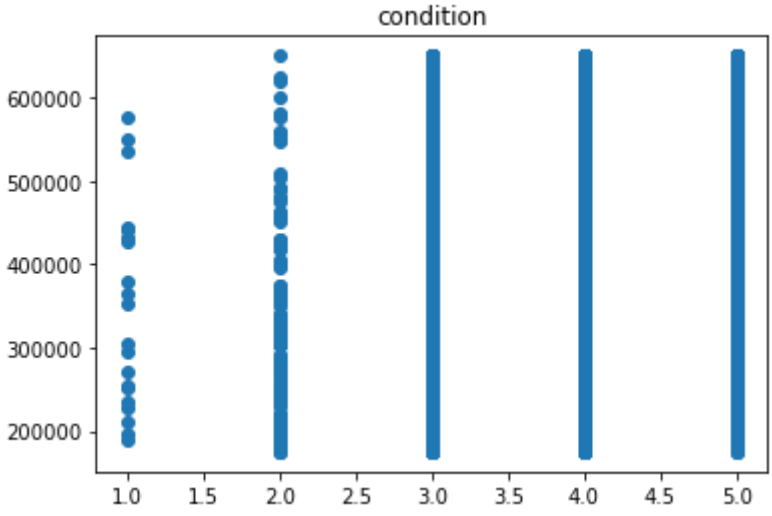


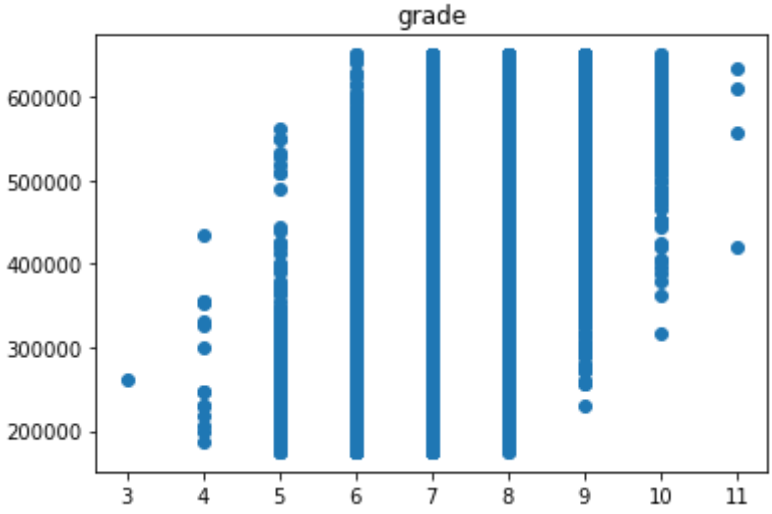

sqft_living
600000
500000
400000
300000
200000
500
1000
1500
2000
2500
3000
3500
4000

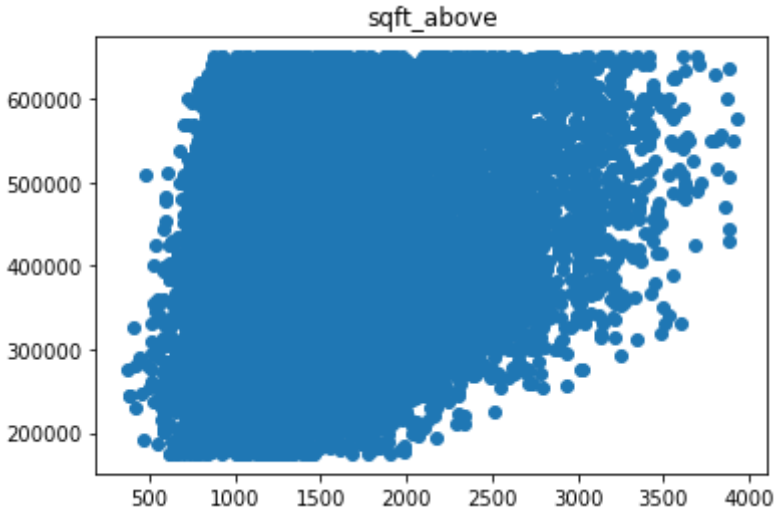

sqft_lot
600000
500000
400000
300000
200000
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1e6

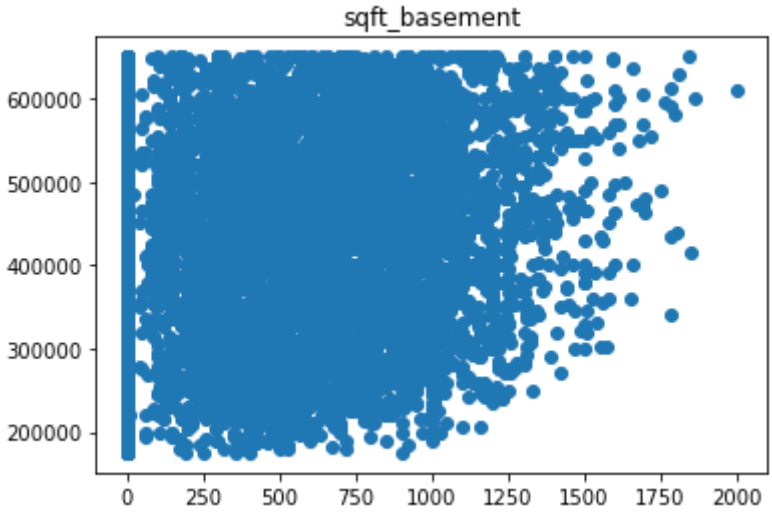

floors
600000
500000
400000
300000
200000
1.0
1.5
2.0
2.5
3.0
3.5

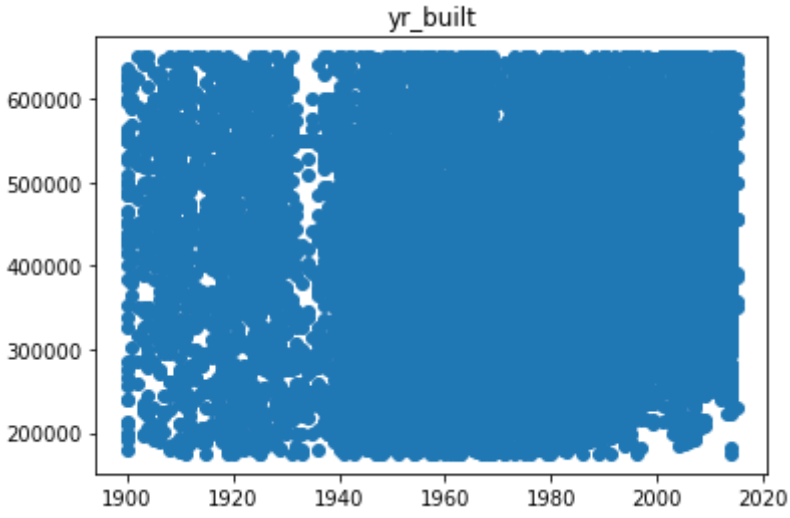

waterfront
600000
500000
400000
300000

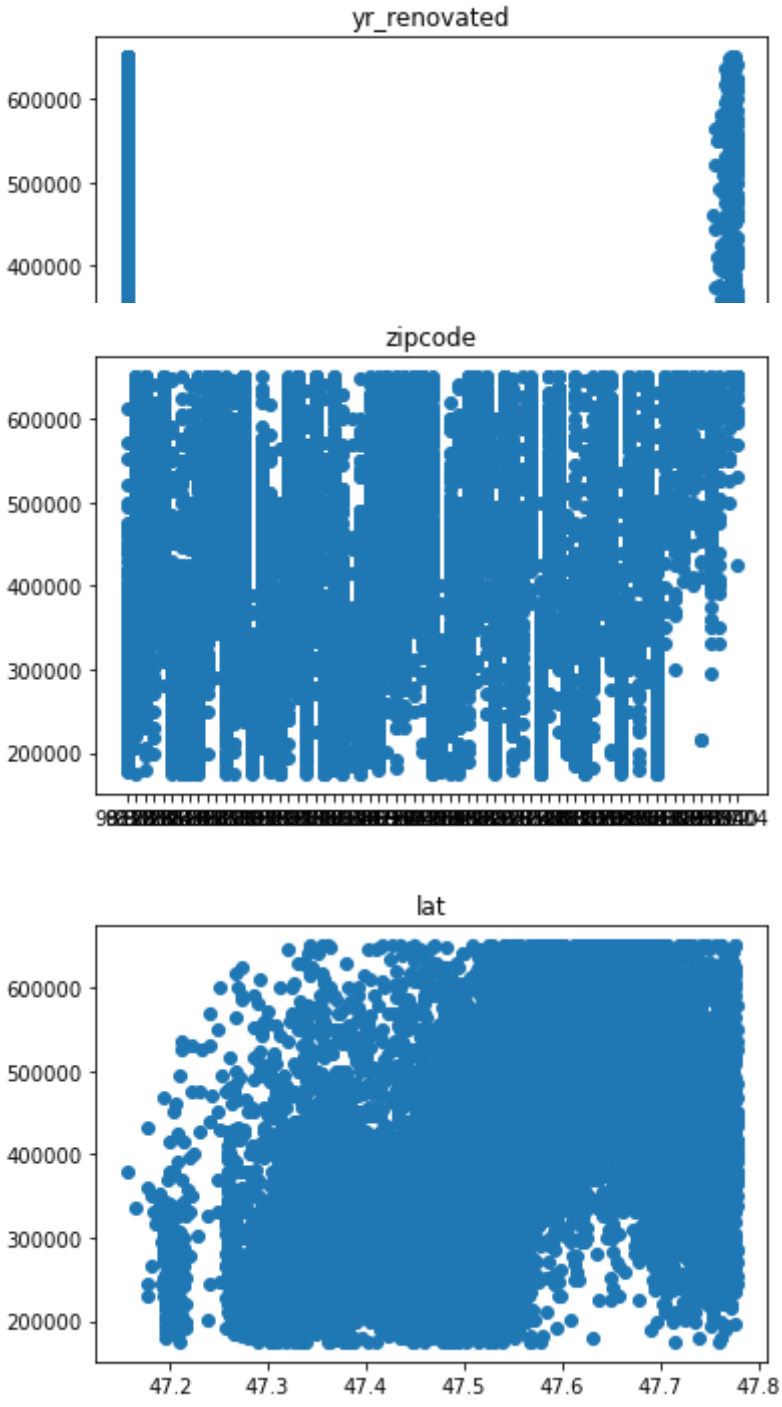

condition
600000
500000
400000
300000
200000
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

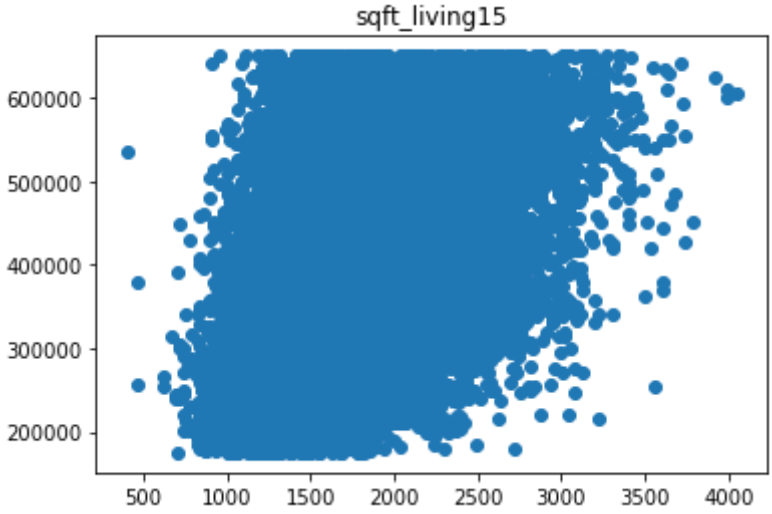

grade
600000
500000
400000
300000
200000
3
4
5
6
7
8
9
10
11

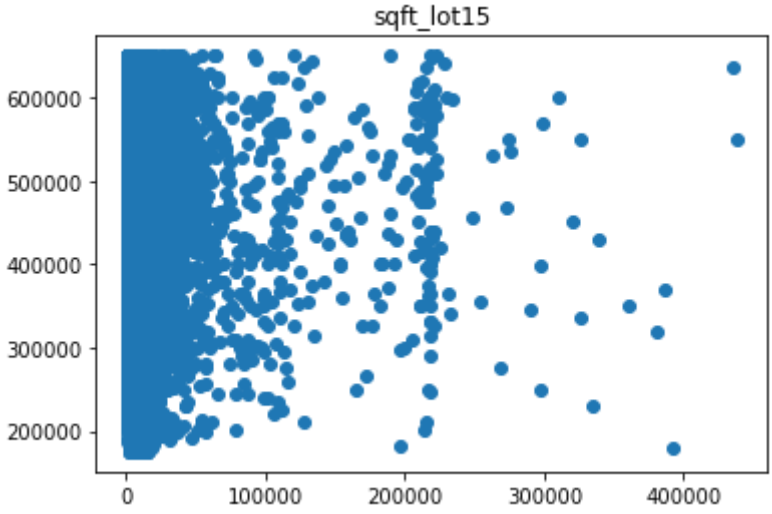

sqft_above
600000
500000
400000
300000
200000
500
1000
1500
2000
2500
3000
3500
4000

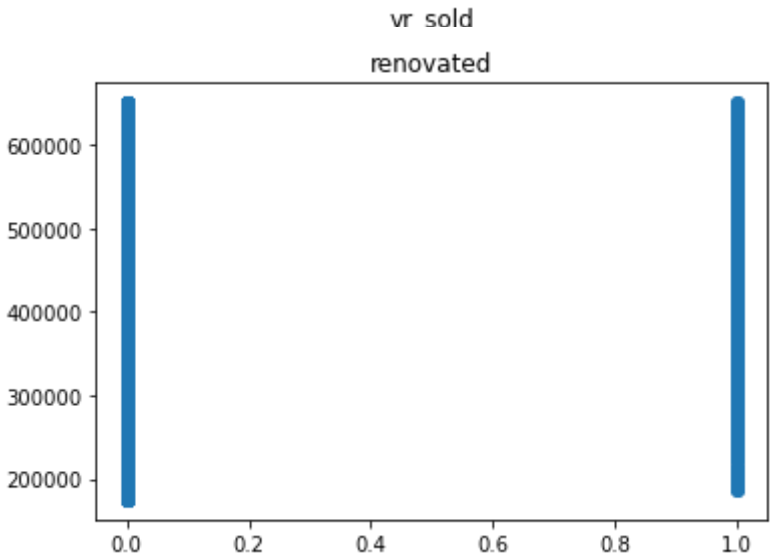

sqft_basement
600000
500000
400000
300000
200000
0
250
500
750
1000
1250
1500
1750
2000

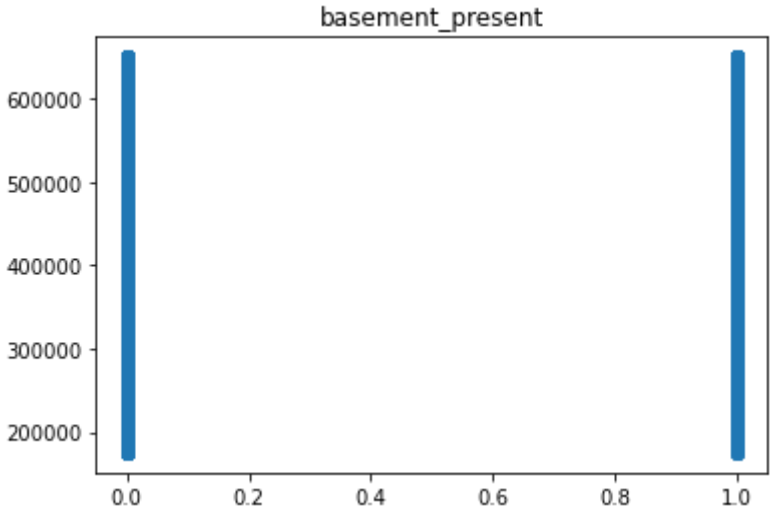

yr_built
600000
500000
400000
300000
200000
1900
1920
1940
1960
1980
2000
2020


yr_renovated
600000
500000
400000
zipcode
600000
500000
400000
300000
200000
lat
600000
500000
400000
300000
200000
47.2
47.3
47.4
47.5
47.6
47.7
47.8

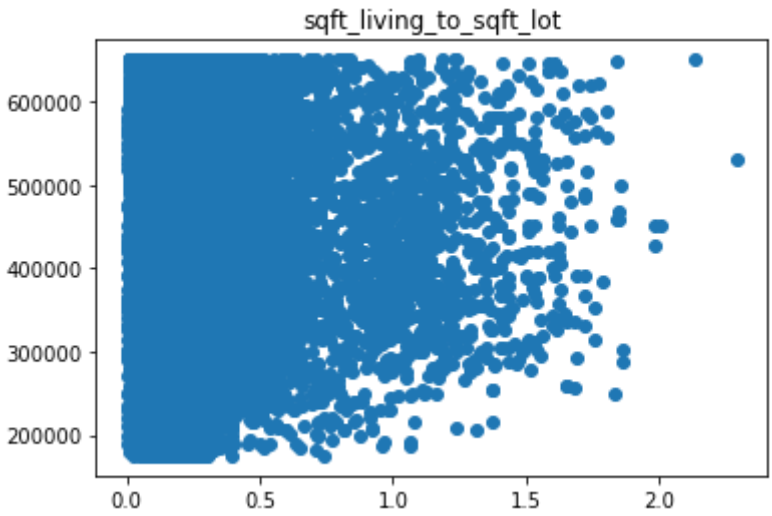

sqft_living15
600000
500000
400000
300000
200000
500
1000
1500
2000
2500
3000
3500
4000


sqft_lot15
600000
500000
400000
300000
200000
0
100000
200000
300000
400000


yr_sold
renovated
600000
500000
400000
300000
200000
0.0
0.2
0.4
0.6
0.8
1.0

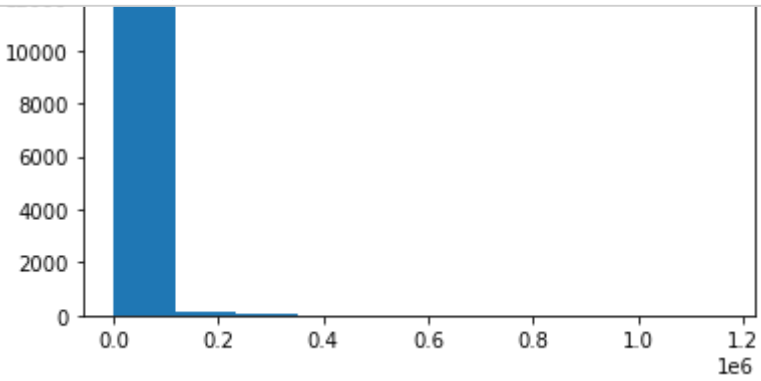

basement_present
600000
500000
400000
300000
200000
0.0
0.2
0.4
0.6
0.8
1.0

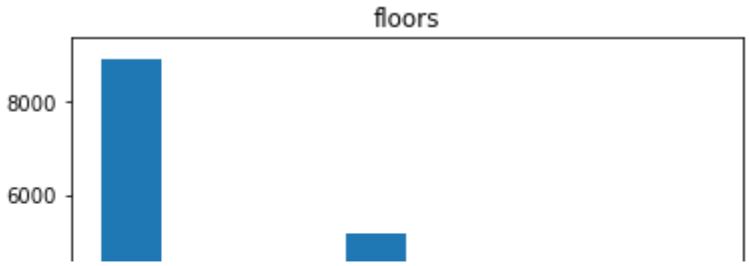

actual_age_of_property
bathrooms_per_bedroom
600000
500000
400000
300000
200000
0.5
1.0
1.5
2.0
2.5


sqft_living_to_sqft_lot
600000
500000
400000
300000
200000
0.0
0.5
1.0
1.5
2.0

In [63]:
```
for col in df.columns:
    plt.hist(df[col])
    plt.title(col)
    plt.show()
```

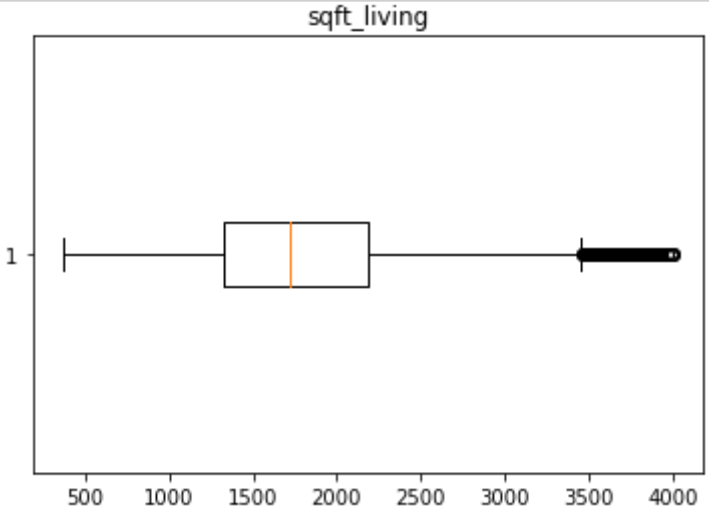
executed in 4.00s, finished 12:32:48 2021-11-07





In [64]:
```
for col in df.select_dtypes('number').columns:
    plt.boxplot(df[col], vert=False)
    plt.title(col)
    plt.show()
```

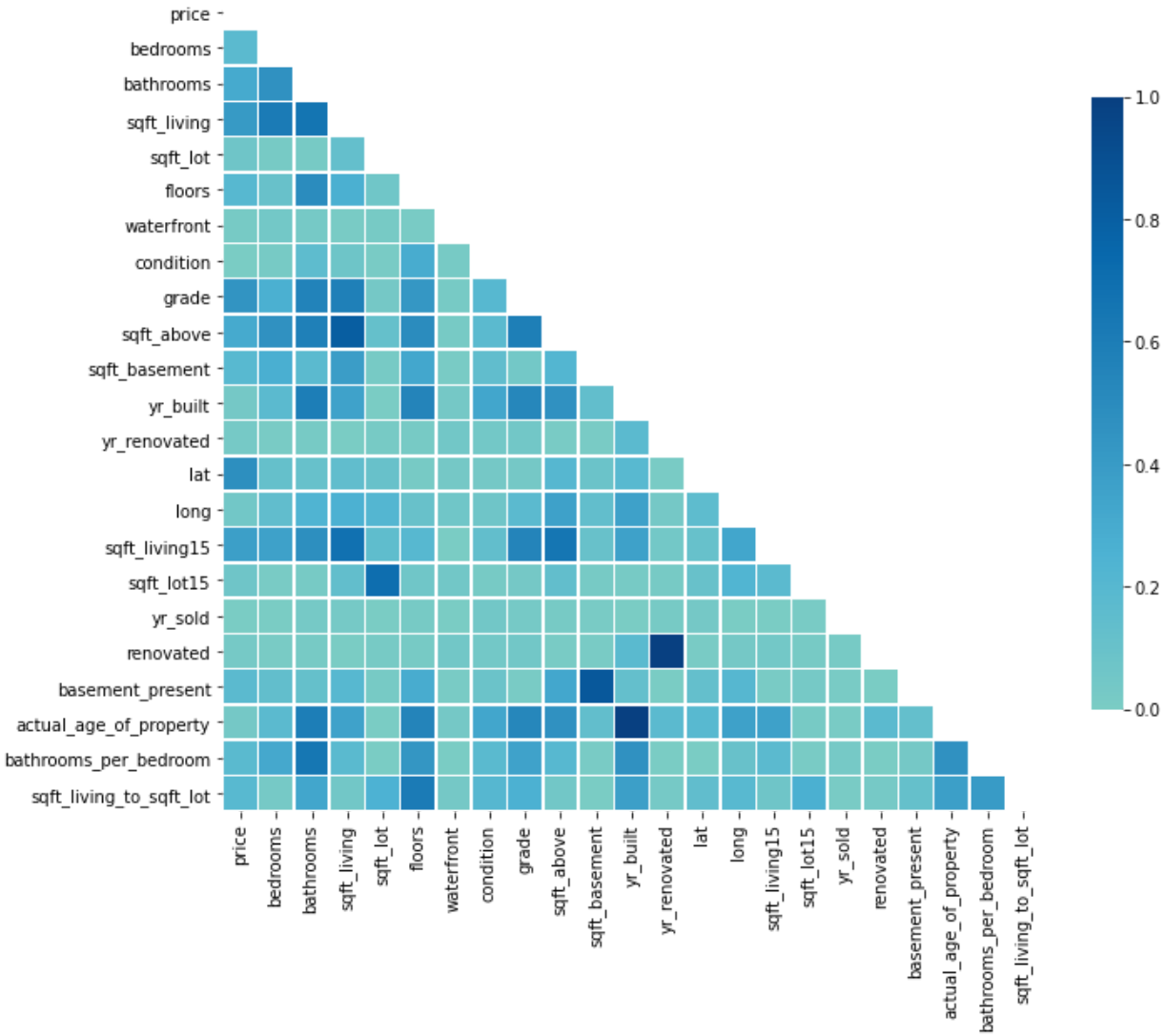
executed in 2.54s, finished 12:32:51 2021-11-07



In [65]:
```
corr = df.corr().abs()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
# GnBu is your color preference
sns.heatmap(corr, mask=mask, cmap="GnBu", vmin=0, vmax=1.0, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .75})
```

executed in 577ms, finished 12:32:51 2021-11-07

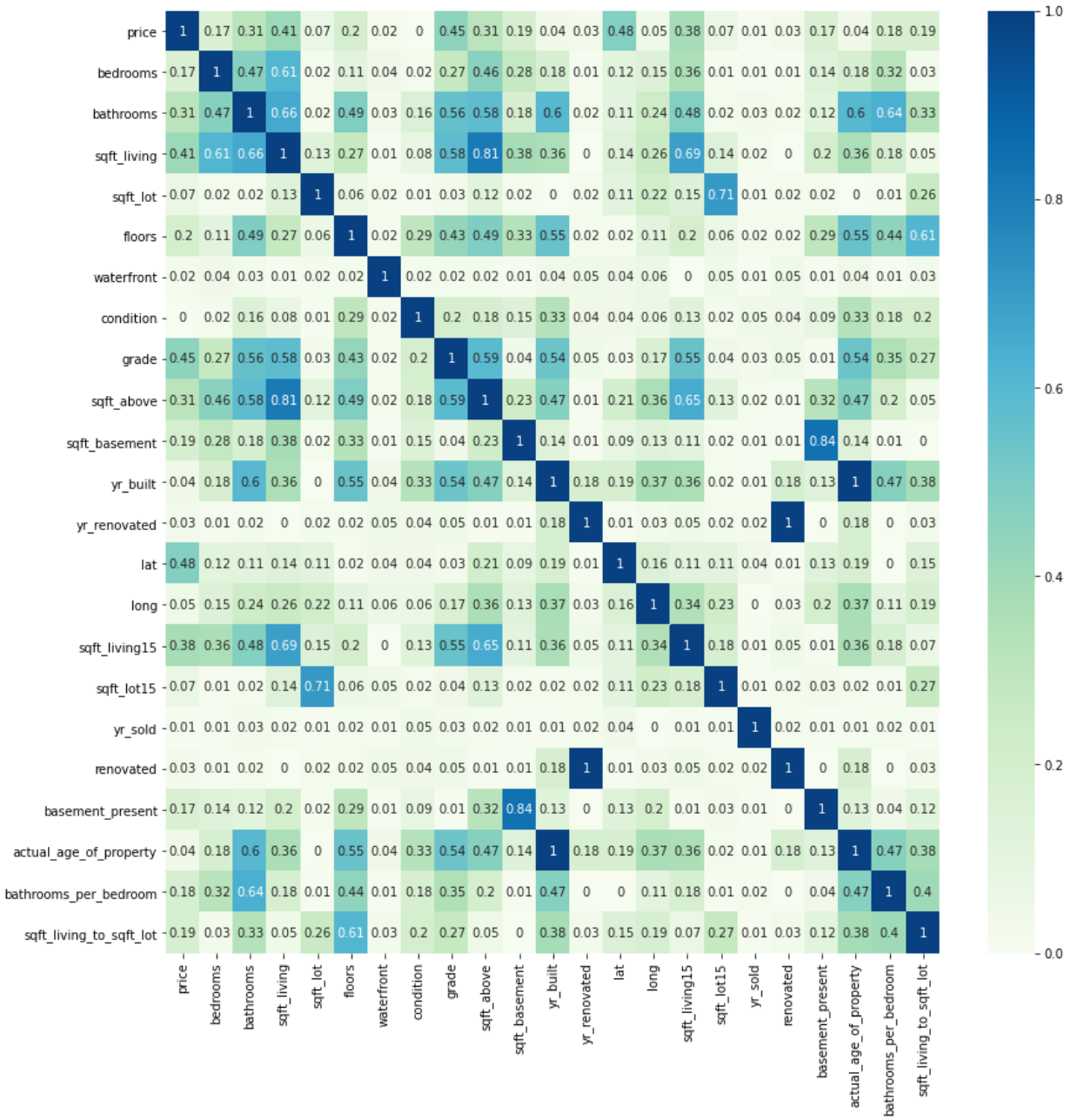
Out[65]: <AxesSubplot:>



In [66]:
```
# Create a correlation heatmap grid with data

plt.figure(figsize=(14, 14))
corr_matrix = df.corr().abs().round(2)
sns.heatmap(data=corr_matrix,cmap="GnBu",annot=True)
```

executed in 2.49s, finished 12:32:54 2021-11-07

Out[66]: <AxesSubplot:>



In [67]:
```
df.head()
```
executed in 16ms, finished 12:32:54 2021-11-07

Out[67]:

id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	gra
7129300520	221900.0	3	1.00	1180	5650	1.0	0.0	3	
6414100192	538000.0	3	2.25	2570	7242	2.0	0.0	3	
5631500400	180000.0	2	1.00	770	10000	1.0	0.0	3	
2487200875	604000.0	4	3.00	1960	5000	1.0	0.0	5	
1954400510	510000.0	3	2.00	1680	8080	1.0	0.0	3	

5 rows × 24 columns

2.4.2 Clean Data Before Modeling

- drop sqft_basement, sqft_basement15, sqft_lot, sqft_lot15, yr_renovated
- drop longitude, latitude,

In [68]:
```
if 'sqft_basement' in df.columns:
    df.drop('sqft_basement', axis=1, inplace=True)
```
executed in 3ms, finished 12:32:54 2021-11-07

In [69]:
```
if 'sqft_basement15' in df.columns:
    df.drop('sqft_basement15', axis=1, inplace=True)
```
executed in 2ms, finished 12:32:54 2021-11-07

In [70]:
```
if 'sqft_lot' in df.columns:
    df.drop('sqft_lot', axis=1, inplace=True)
```
executed in 4ms, finished 12:32:54 2021-11-07

In [71]:
```
if 'sqft_lot15' in df.columns:
    df.drop('sqft_lot15', axis=1, inplace=True)
```
executed in 4ms, finished 12:32:54 2021-11-07

In [72]:
```
if 'sqft_living15' in df.columns:
    df.drop('sqft_living15', axis=1, inplace=True)
```
executed in 4ms, finished 12:32:54 2021-11-07

In [73]:
```
if 'yr_renovated' in df.columns:
    df.drop('yr_renovated', axis=1, inplace=True)
```
executed in 3ms, finished 12:32:54 2021-11-07

In [74]:
```
if 'lat' in df.columns:
    df.drop('lat', axis=1, inplace=True)
```
executed in 3ms, finished 12:32:54 2021-11-07

In [75]:
```
if 'long' in df.columns:
    df.drop('long', axis=1, inplace=True)
```
executed in 3ms, finished 12:32:54 2021-11-07

After checking for overfitting in models, drop additional features

In [76]:
```
# after checking for overfitting in models, drop additional features

#'const': 49.808021804391046,
#'bedrooms': 8.124041148684404,
#'bathrooms': 13.373648045281776,
#'sqft_living': 10.9560606782623,
#'floors': 3.316672951320918,
#'waterfront': 1.078179446917037,
#'condition': 1.2848095397229615,
#'grade': 2.273801501092483,
#'sqft_above': 11.76560610464051,
#'yr_built': inf,
#'yr_sold': inf,
#'renovated': 1.0815968469358994,
#'basement_present': 3.8066176363082005,
#'actual_age_of_property': inf,
#'bathrooms_per_bedroom': 10.77082956774539,
#'sqft_living_to_sqft_lot': 3.1142842893817773,

if 'sqft_above' in df.columns:
    df.drop('sqft_above', axis=1, inplace=True)
```
executed in 4ms, finished 12:32:54 2021-11-07

In [77]:
```
if 'bathrooms_per_bedroom' in df.columns:
    df.drop('bathrooms_per_bedroom', axis=1, inplace=True)
```
executed in 3ms, finished 12:32:54 2021-11-07

In [78]:
```
if 'sqft_living_to_sqft_lot' in df.columns:
    df.drop('sqft_living_to_sqft_lot', axis=1, inplace=True)
```
executed in 4ms, finished 12:32:54 2021-11-07

In [79]:
```
if 'yr_built' in df.columns:
    df.drop('yr_built', axis=1, inplace=True)
```
executed in 3ms, finished 12:32:54 2021-11-07

In [80]:
```
if 'yr_sold' in df.columns:
    df.drop('yr_sold', axis=1, inplace=True)
```
executed in 3ms, finished 12:32:54 2021-11-07

In [81]:
```
if 'actual_age_of_property' in df.columns:
    df.drop('actual_age_of_property', axis=1, inplace=True)
```

executed in 4ms, finished 12:32:54 2021-11-07

In [82]:
```
df.head()
```

executed in 9ms, finished 12:32:54 2021-11-07

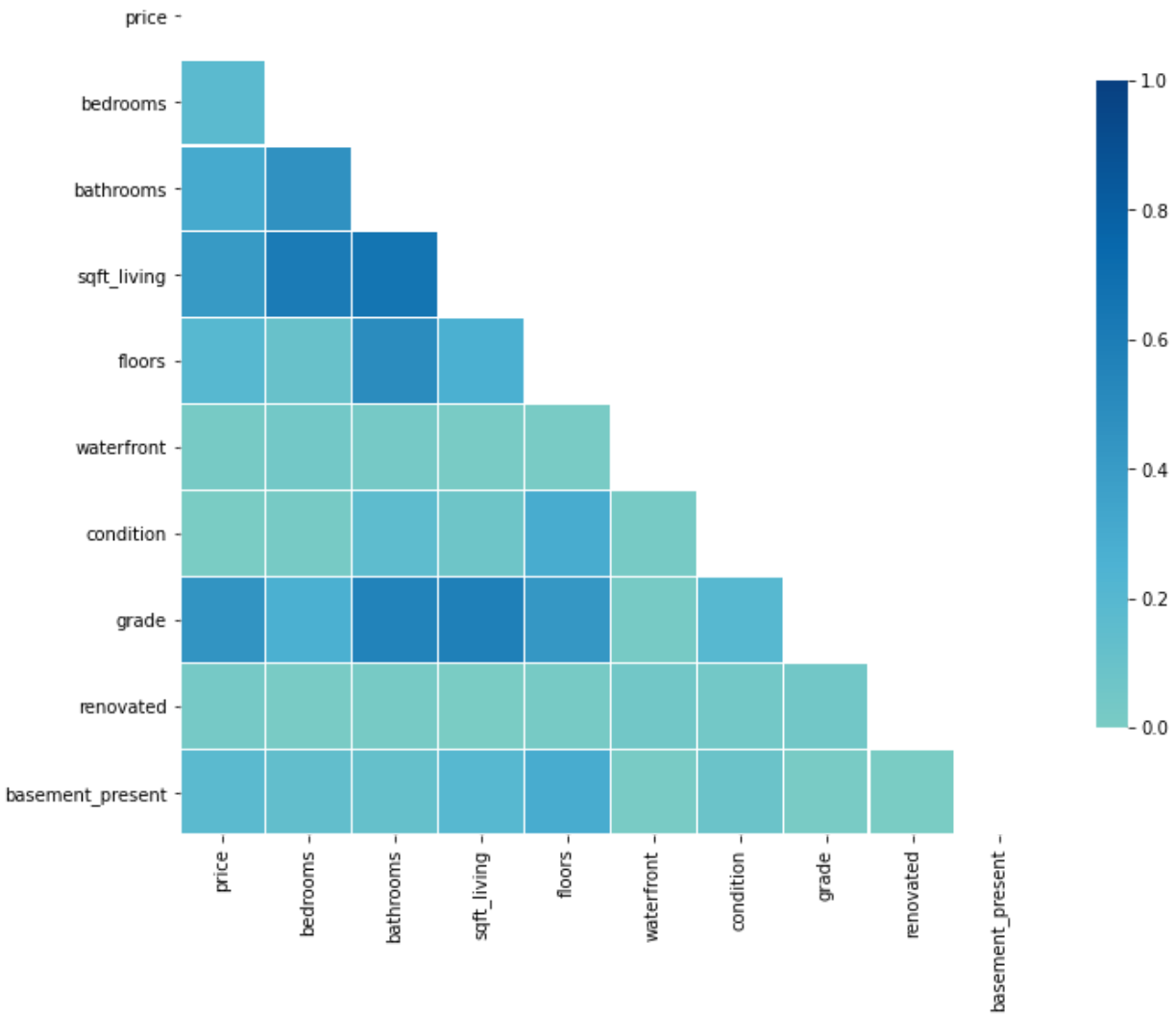
Out[82]:

id	price	bedrooms	bathrooms	sqft_living	floors	waterfront	condition	grade	zipco
7129300520	221900.0	3	1.00	1180	1.0	0.0	3	7	981
6414100192	538000.0	3	2.25	2570	2.0	0.0	3	7	981
5631500400	180000.0	2	1.00	770	1.0	0.0	3	6	980
2487200875	604000.0	4	3.00	1960	1.0	0.0	5	7	981
1954400510	510000.0	3	2.00	1680	1.0	0.0	3	8	980

In [83]:
```
# See if correlation heatmap changes with clean data

corr = df.corr().abs()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap="GnBu", vmin=0, vmax=1.0, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .75})
```

executed in 232ms, finished 12:32:54 2021-11-07

Out[83]: <AxesSubplot:>



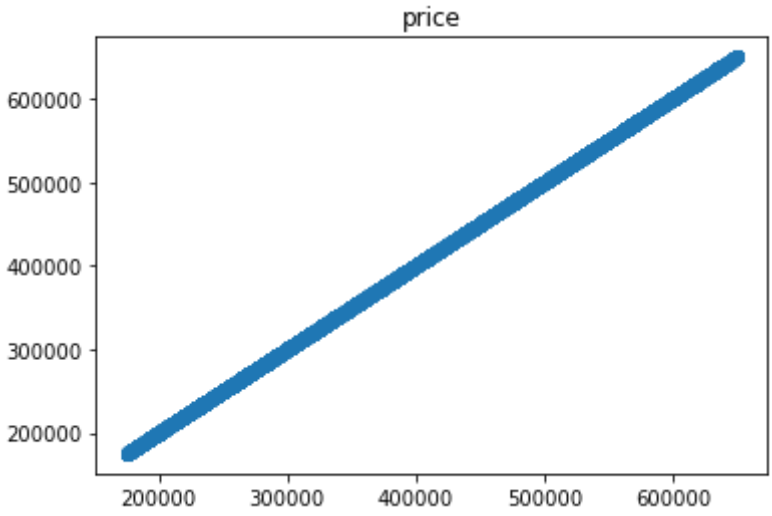
In [84]:
```
# Create a correlation heatmap grid with data

plt.figure(figsize=(14, 14))
corr_matrix = df.corr().abs().round(2)
sns.heatmap(data=corr_matrix,cmap="GnBu",annot=True)
```

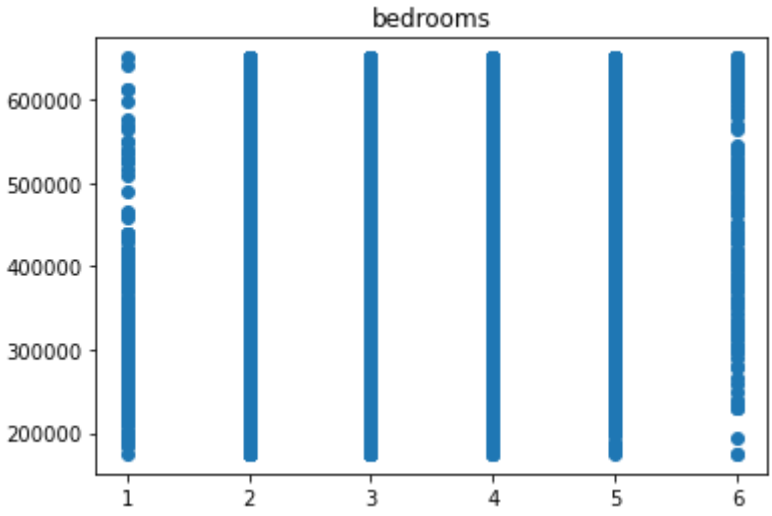
executed in 664ms, finished 12:32:55 2021-11-07

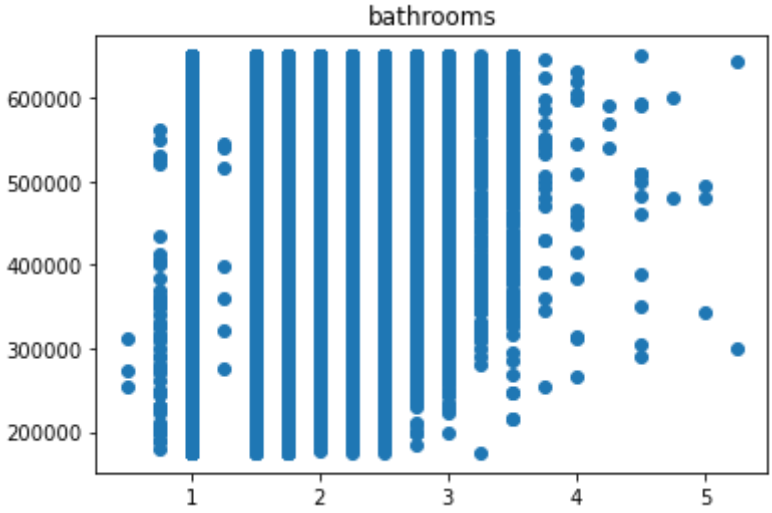
Out[84]: <AxesSubplot:>

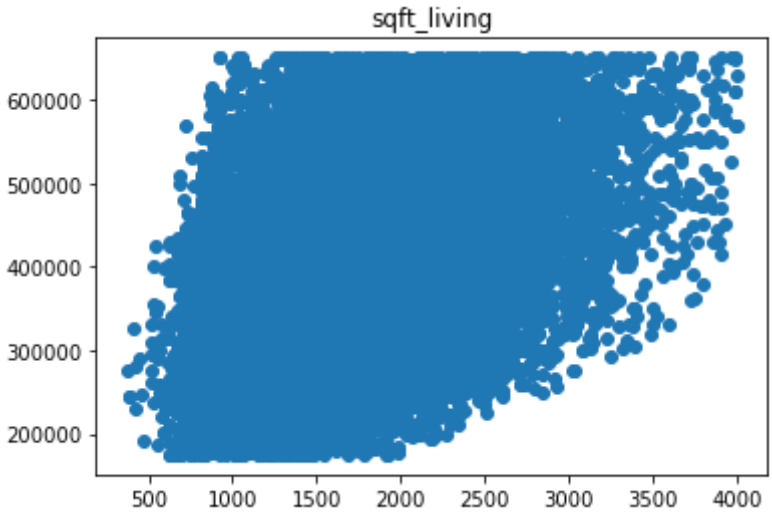
In [85]:
```
# Check data one more time

for col in df.columns:
    plt.scatter(df[col], df[TARGET])
    plt.title(col)
    plt.show()
```

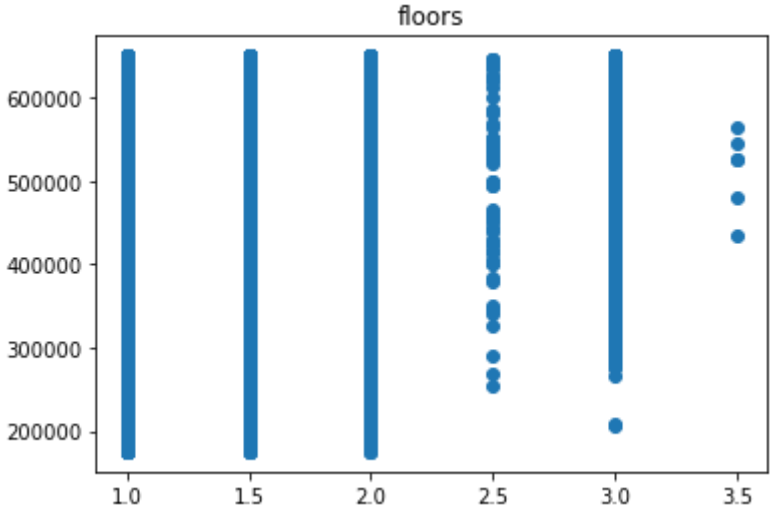
executed in 1.78s, finished 12:32:57 2021-11-07

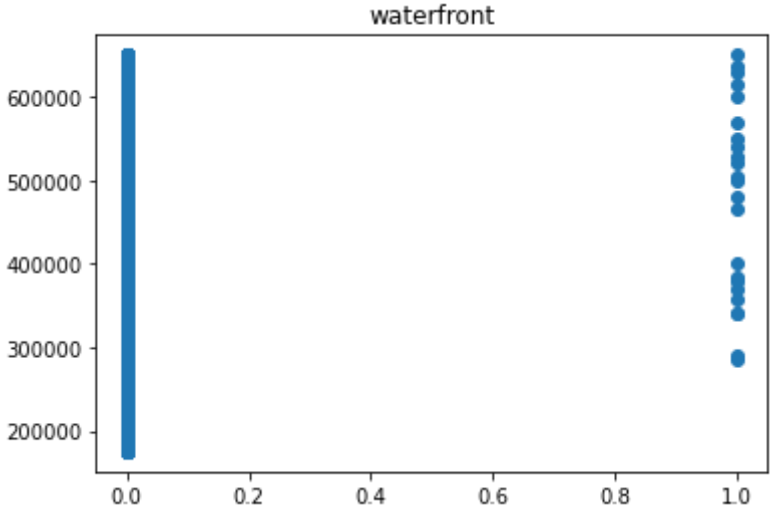


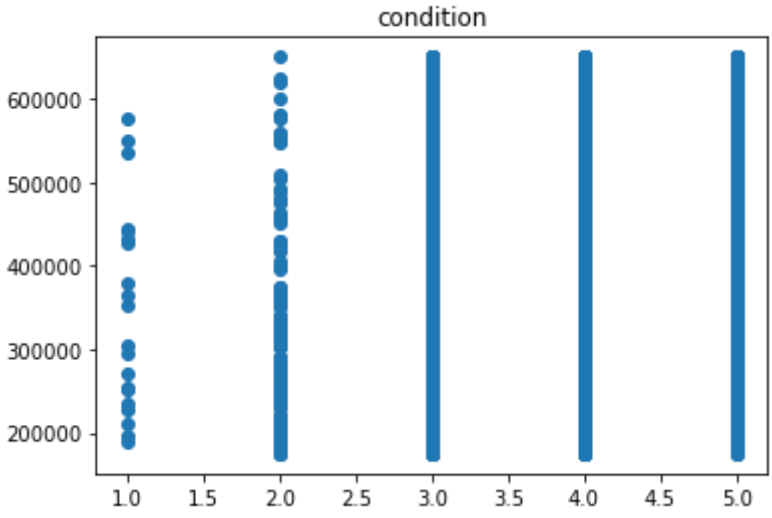


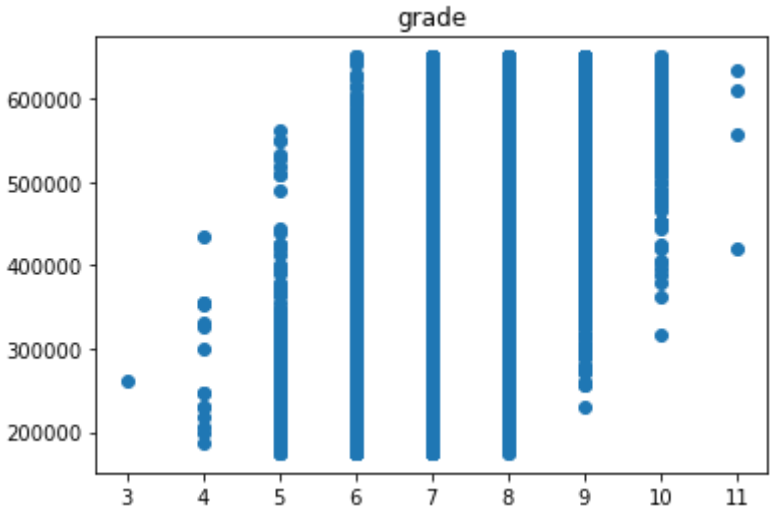


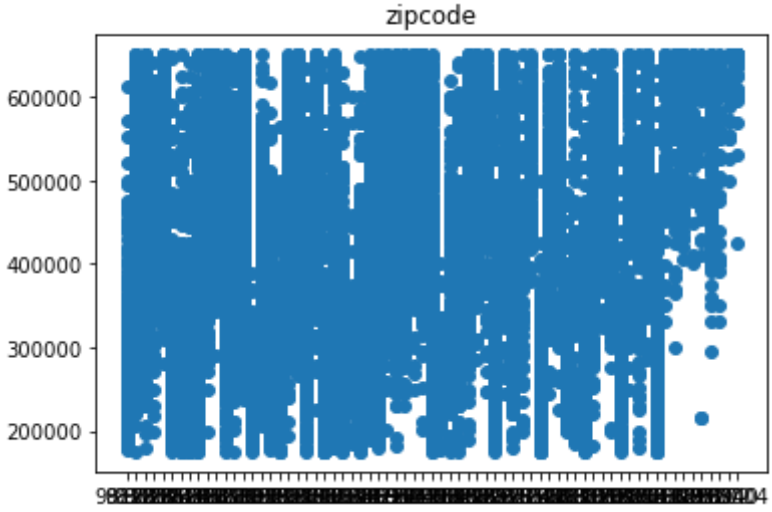

sqft_living
600000
500000
400000
300000
200000
500
1000
1500
2000
2500
3000
3500
4000

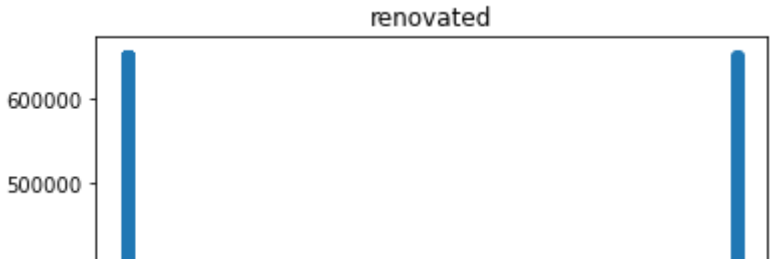

floors
600000
500000
400000
300000
200000
1.0
1.5
2.0
2.5
3.0
3.5

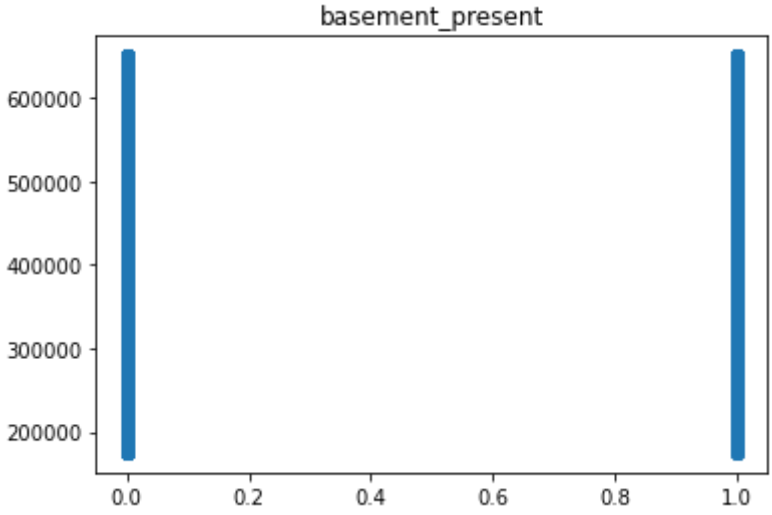

waterfront
600000
500000
400000
300000
200000
0.0
0.2
0.4
0.6
0.8
1.0


condition
600000
500000
400000
300000
200000
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


grade
600000
500000
400000
300000
200000
3
4
5
6
7
8
9
10
11

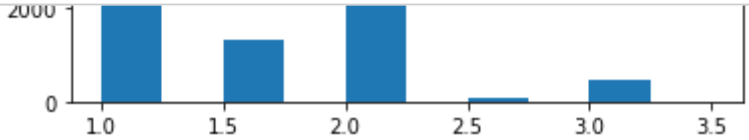

zipcode
600000
500000
400000
300000
200000

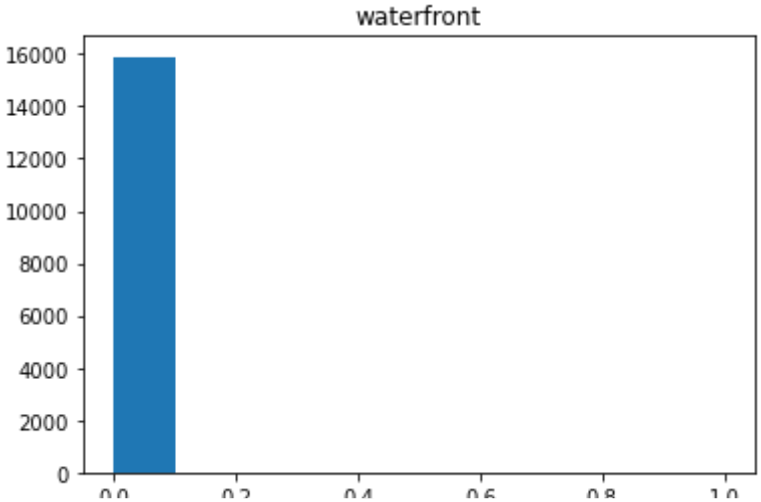




In [86]:
```
for col in df.columns:
    plt.hist(df[col])
    plt.title(col)
    plt.show()
```

executed in 2.23s, finished 12:32:59 2021-11-07





3 Start Building Model

- Create dependent (y) and independent (x) variables
- Create Train and Test data subsets

3.1 Create TARGET and Independent Variables for Model

In [87]:
```
# Dependent variable(y) is price as previously defined as TARGET
# Independent variables(X) are all variables that are not price

y = df[TARGET]
X = df.drop(columns=[TARGET])
X
```

executed in 15ms, finished 12:32:59 2021-11-07

Out[87]:

id	bedrooms	bathrooms	sqft_living	floors	waterfront	condition	grade	zipcode	renova
7129300520	3	1.00	1180	1.0	0.0	3	7	98178	
6414100192	3	2.25	2570	2.0	0.0	3	7	98125	
5631500400	2	1.00	770	1.0	0.0	3	6	98028	
2487200875	4	3.00	1960	1.0	0.0	5	7	98136	
1954400510	3	2.00	1680	1.0	0.0	3	8	98074	
...	...	...	...	...	...	...	...	...	
263000018	3	2.50	1530	3.0	0.0	3	8	98103	
6600060120	4	2.50	2310	2.0	0.0	3	8	98146	
1523300141	2	0.75	1020	2.0	0.0	3	7	98144	
291310100	3	2.50	1600	2.0	0.0	3	8	98027	
1523300157	2	0.75	1020	2.0	0.0	3	7	98144	

15916 rows × 10 columns

In [88]:
```
y.describe()
```

executed in 6ms, finished 12:32:59 2021-11-07

Out[88]:
```
count     15916.000000
mean     400358.968145
std      123707.148968
min      175000.000000
25%      299725.000000
50%      390000.000000
75%      499900.000000
max      650000.000000
Name: price, dtype: float64
```

3.2 Create Train and Test Data Subsets

In [89]:
```
# Create Train and Test data subsets using train_test_split
# Check shape of each data set

X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=100)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

executed in 6ms, finished 12:32:59 2021-11-07

Out[89]:
```
((11937, 10), (3979, 10), (11937,), (3979,))
```

In [90]:
```
# Check percentage of data that is Train data
# Train data is 75% of data; Test data is 25% of data

11_937/(11_937+3979)
```

executed in 2ms, finished 12:32:59 2021-11-07

Out[90]:
```
0.75
```

In [91]:
```
# Reset index on Train and Test data

X_train.reset_index(drop=True, inplace=True)
X_test.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)
```

executed in 2ms, finished 12:32:59 2021-11-07

Create Number and Category Column Variables

In [92]:
```
# Create variable for "Number" columns (integers, floats)
# Create variable for "Category" columns (objects, strings)
# Check Category Columns

NUMBER_COLS = X_train.select_dtypes('number').columns
CATEGORY_COLS = X_train.select_dtypes('object').columns
CATEGORY_COLS
```

executed in 6ms, finished 12:32:59 2021-11-07

Out[92]:
```
Index(['zipcode'], dtype='object')
```

3.3 One Hot Encode Category Columns (zipcode)

In [93]:
```
# ONE HOT ENCODE
# zipcode is the only category column
# One Hot Encode zipcodes so you can use the data in model

ohe = OneHotEncoder(drop='first', sparse=False)
X_train_ohe = ohe.fit_transform(X_train[CATEGORY_COLS])
X_test_ohe = ohe.transform(X_test[CATEGORY_COLS])

X_train_ohe = pd.DataFrame(X_train_ohe, columns=ohe.get_feature_names(CATEG
X_test_ohe = pd.DataFrame(X_test_ohe, columns=ohe.get_feature_names(CATEGOR

X_train_ohe.columns = [c.lower() for c in X_train_ohe]
X_test_ohe.columns = [c.lower() for c in X_test_ohe]
```

executed in 12ms, finished 12:32:59 2021-11-07

In [94]:
```
# Check one hot encoding of zipcodes

X_train_ohe.head()
```

executed in 17ms, finished 12:32:59 2021-11-07

Out[94]:

	zipcode_98002	zipcode_98003	zipcode_98004	zipcode_98005	zipcode_98006	zipcode_98007	zip
0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	1.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 68 columns

In [95]:
```
# Concatenate Number Columns with One Hot Encoded Columns

X_train_raw = pd.concat([X_train[NUMBER_COLS],
                         X_train_ohe],
                         axis=1)
X_test_raw = pd.concat([X_test[NUMBER_COLS],
                         X_test_ohe],
                         axis=1)
```

executed in 10ms, finished 12:32:59 2021-11-07

3.4 Scale the Data

In [96]:
```
# Scale the Data using StandardScaler()

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train[NUMBER_COLS])
X_test_scaled = scaler.transform(X_test[NUMBER_COLS])

X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train[NUMBER_COLS].
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test[NUMBER_COLS].col
```

executed in 11ms, finished 12:32:59 2021-11-07

In [97]:
```
# Check the shape of the data

X_train_scaled.shape, X_test_scaled.shape
```

executed in 3ms, finished 12:32:59 2021-11-07

Out[97]:
```
((11937, 9), (3979, 9))
```

In [98]:
```
# Concatenate Scaled data with One Hot Encoded data

X_train_scaled = pd.concat([X_train_scaled,
                            X_train_ohe],
                            axis=1)
X_test_scaled = pd.concat([X_test_scaled,
                           X_test_ohe],
                           axis=1)
```

executed in 9ms, finished 12:32:59 2021-11-07

In [99]:
```
# Check X_train_scaled data

X_train_scaled
```

executed in 24ms, finished 12:32:59 2021-11-07

Out[99]:

	bedrooms	bathrooms	sqft_living	floors	waterfront	condition	grade	renovated	ba
0	-1.484449	-1.457646	-1.647388	-0.796457	-0.034267	-0.633146	-1.566432	-0.153275	
1	0.914667	0.465379	1.044130	1.066624	-0.034267	-0.633146	0.855183	-0.153275	
2	0.914667	0.849984	0.489515	-0.796457	-0.034267	2.512689	0.855183	-0.153275	
3	0.914667	-1.457646	-1.272207	-0.796457	-0.034267	0.939771	-1.566432	-0.153275	
4	-1.484449	-1.457646	-0.652342	-0.796457	-0.034267	-0.633146	-0.355625	-0.153275	
...	...	...	...	...	...	...	...	...	
11932	2.114224	1.234589	2.185987	-0.796457	-0.034267	0.939771	0.855183	-0.153275	
11933	2.114224	0.849984	2.626417	-0.796457	-0.034267	0.939771	0.855183	-0.153275	
11934	-0.284891	0.465379	-0.211911	-0.796457	-0.034267	0.939771	0.855183	-0.153275	
11935	-0.284891	0.465379	-0.521844	1.066624	-0.034267	-0.633146	0.855183	-0.153275	
11936	-0.284891	-0.303831	-0.472907	-0.796457	-0.034267	-0.633146	-0.355625	-0.153275	

11937 rows × 77 columns

3.5 Add an Intercept

In [100]:
```
# Add an intercept with sm.add_constant()
# The b in y = mx + b

X_train_scaled = sm.add_constant(X_train_scaled)
X_test_scaled = sm.add_constant(X_test_scaled)
```

executed in 32ms, finished 12:32:59 2021-11-07

In [101]:
```
X_train_scaled = sm.add_constant(X_train_scaled)
X_test_scaled = sm.add_constant(X_test_scaled)
```

executed in 34ms, finished 12:32:59 2021-11-07

4 MODELS

4.1 Model 1: Everything

- Used for exploration

In [102]:
```
model1 = sm.OLS(y_train, X_train_scaled).fit()
model1.summary()
```

executed in 84ms, finished 12:32:59 2021-11-07

Out[102]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.742
Model:	OLS	**Adj. R-squared:**	0.740
Method:	Least Squares	**F-statistic:**	442.0
Date:	Sun, 07 Nov 2021	**Prob (F-statistic):**	0.00
Time:	12:32:59	**Log-Likelihood:**	-1.4886e+05
No. Observations:	11937	**AIC:**	2.979e+05
Df Residuals:	11859	**BIC:**	2.985e+05
Df Model:	77		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.735e+05	4044.287	67.617	0.000	2.66e+05	2.81e+05
bedrooms	-3777.9235	766.377	-4.930	0.000	-5280.148	-2275.698
bathrooms	4006.3214	919.168	4.359	0.000	2204.602	5808.041
sqft_living	5.751e+04	1008.240	57.044	0.000	5.55e+04	5.95e+04
floors	-7602.7410	836.628	-9.087	0.000	-9242.669	-5962.812
waterfront	5088.3292	600.906	8.468	0.000	3910.454	6266.204
condition	9541.0686	629.450	15.158	0.000	8307.244	1.08e+04
grade	2.607e+04	818.004	31.876	0.000	2.45e+04	2.77e+04
renovated	1469.4406	585.502	2.510	0.012	321.761	2617.121
basement_present	-5546.0095	721.806	-7.684	0.000	-6960.868	-4131.151
zipcode_98002	-1.29e+04	6672.443	-1.933	0.053	-2.6e+04	178.832
zipcode_98003	1740.5176	6073.043	0.287	0.774	-1.02e+04	1.36e+04
zipcode_98004	3.714e+05	1.63e+04	22.743	0.000	3.39e+05	4.03e+05
zipcode_98005	2.74e+05	1.1e+04	24.807	0.000	2.52e+05	2.96e+05
zipcode_98006	2.052e+05	6840.113	30.006	0.000	1.92e+05	2.19e+05
zipcode_98007	2.173e+05	8409.334	25.836	0.000	2.01e+05	2.34e+05
zipcode_98008	2.181e+05	6419.708	33.974	0.000	2.06e+05	2.31e+05
zipcode_98010	8.115e+04	8837.877	9.182	0.000	6.38e+04	9.85e+04
zipcode_98011	1.529e+05	6972.915	21.932	0.000	1.39e+05	1.67e+05
zipcode_98014	1.097e+05	8779.574	12.491	0.000	9.25e+04	1.27e+05
zipcode_98019	1.098e+05	6680.290	16.431	0.000	9.67e+04	1.23e+05

zipcode_98022	2.712e+04	6326.221	4.288	0.000	1.47e+04	3.95e+04
zipcode_98023	-1.575e+04	5246.049	-3.002	0.003	-2.6e+04	-5466.508
zipcode_98024	1.462e+05	1.03e+04	14.240	0.000	1.26e+05	1.66e+05
zipcode_98027	1.763e+05	5977.162	29.489	0.000	1.65e+05	1.88e+05
zipcode_98028	1.363e+05	6044.350	22.553	0.000	1.24e+05	1.48e+05
zipcode_98029	2.123e+05	6367.167	33.347	0.000	2e+05	2.25e+05
zipcode_98030	7039.3493	6236.424	1.129	0.259	-5185.064	1.93e+04
zipcode_98031	8528.6375	5947.097	1.434	0.152	-3128.648	2.02e+04
zipcode_98032	-1.223e+04	8048.185	-1.519	0.129	-2.8e+04	3547.731
zipcode_98033	2.437e+05	6552.817	37.187	0.000	2.31e+05	2.57e+05
zipcode_98034	1.645e+05	5296.904	31.057	0.000	1.54e+05	1.75e+05
zipcode_98038	4.63e+04	5160.252	8.973	0.000	3.62e+04	5.64e+04
zipcode_98040	3.26e+05	1.59e+04	20.537	0.000	2.95e+05	3.57e+05
zipcode_98042	1.271e+04	5169.826	2.458	0.014	2571.503	2.28e+04
zipcode_98045	9.882e+04	6641.193	14.880	0.000	8.58e+04	1.12e+05
zipcode_98052	2.154e+05	5726.400	37.616	0.000	2.04e+05	2.27e+05
zipcode_98053	2.064e+05	6463.692	31.929	0.000	1.94e+05	2.19e+05
zipcode_98055	3.959e+04	6104.436	6.485	0.000	2.76e+04	5.16e+04
zipcode_98056	8.944e+04	5560.338	16.086	0.000	7.85e+04	1e+05
zipcode_98058	4.089e+04	5385.866	7.593	0.000	3.03e+04	5.15e+04
zipcode_98059	9.496e+04	5502.125	17.258	0.000	8.42e+04	1.06e+05
zipcode_98065	1.48e+05	6142.417	24.090	0.000	1.36e+05	1.6e+05
zipcode_98070	1.408e+05	8714.996	16.152	0.000	1.24e+05	1.58e+05
zipcode_98072	1.642e+05	6510.426	25.221	0.000	1.51e+05	1.77e+05
zipcode_98074	2.013e+05	6243.559	32.239	0.000	1.89e+05	2.14e+05
zipcode_98075	2.343e+05	8190.912	28.608	0.000	2.18e+05	2.5e+05
zipcode_98077	1.627e+05	8442.365	19.277	0.000	1.46e+05	1.79e+05
zipcode_98092	1469.8656	5764.394	0.255	0.799	-9829.292	1.28e+04
zipcode_98102	2.971e+05	1.17e+04	25.400	0.000	2.74e+05	3.2e+05
zipcode_98103	2.441e+05	5508.917	44.306	0.000	2.33e+05	2.55e+05
zipcode_98105	3.051e+05	8436.754	36.162	0.000	2.89e+05	3.22e+05
zipcode_98106	9.325e+04	5853.373	15.931	0.000	8.18e+04	1.05e+05
zipcode_98107	2.819e+05	6487.600	43.455	0.000	2.69e+05	2.95e+05
zipcode_98108	1.028e+05	6800.267	15.124	0.000	8.95e+04	1.16e+05
zipcode_98109	3.119e+05	1.29e+04	24.230	0.000	2.87e+05	3.37e+05
zipcode_98112	3.033e+05	9963.820	30.437	0.000	2.84e+05	3.23e+05

zipcode_98115	2.569e+05	5559.862	46.199	0.000	2.46e+05	2.68e+05
zipcode_98116	2.537e+05	6430.978	39.453	0.000	2.41e+05	2.66e+05
zipcode_98117	2.582e+05	5454.138	47.348	0.000	2.48e+05	2.69e+05
zipcode_98118	1.346e+05	5496.994	24.481	0.000	1.24e+05	1.45e+05
zipcode_98119	2.984e+05	9594.954	31.095	0.000	2.8e+05	3.17e+05
zipcode_98122	2.415e+05	6810.371	35.462	0.000	2.28e+05	2.55e+05
zipcode_98125	1.686e+05	5663.514	29.765	0.000	1.57e+05	1.8e+05
zipcode_98126	1.61e+05	5796.992	27.777	0.000	1.5e+05	1.72e+05
zipcode_98133	1.264e+05	5269.619	23.988	0.000	1.16e+05	1.37e+05
zipcode_98136	2.102e+05	6579.194	31.956	0.000	1.97e+05	2.23e+05
zipcode_98144	1.814e+05	6201.612	29.254	0.000	1.69e+05	1.94e+05
zipcode_98146	8.473e+04	6252.114	13.552	0.000	7.25e+04	9.7e+04
zipcode_98148	3.585e+04	1.09e+04	3.286	0.001	1.45e+04	5.72e+04
zipcode_98155	1.244e+05	5408.652	23.005	0.000	1.14e+05	1.35e+05
zipcode_98166	1.012e+05	6410.773	15.789	0.000	8.87e+04	1.14e+05
zipcode_98168	2.8e+04	6444.679	4.345	0.000	1.54e+04	4.06e+04
zipcode_98177	1.859e+05	7048.310	26.374	0.000	1.72e+05	2e+05
zipcode_98178	4.12e+04	6272.872	6.569	0.000	2.89e+04	5.35e+04
zipcode_98188	2.428e+04	7679.720	3.161	0.002	9223.028	3.93e+04
zipcode_98198	3.161e+04	6226.177	5.077	0.000	1.94e+04	4.38e+04
zipcode_98199	2.673e+05	7404.979	36.100	0.000	2.53e+05	2.82e+05

Omnibus:	561.223	**Durbin-Watson:**	2.024
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1486.527
Skew:	0.243	**Prob(JB):**	0.00
Kurtosis:	4.659	**Cond. No.**	100.

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

In [103]:
```
# Results sorted by coefficients descending

results1_as_html = model1.summary().tables[1].as_html()
results1 = pd.read_html(results1_as_html, header=0, index_col=0)[0]
results1.sort_values('coef', ascending=False)#.set_option('display.max_rows
```

executed in 100ms, finished 12:32:59 2021-11-07

Out[103]:

	coef	std err	t	P>|t|	[0.025	0.975]
zipcode_98004	371400.0000	16300.000	22.743	0.000	339000.000	403000.000
zipcode_98040	326000.0000	15900.000	20.537	0.000	295000.000	357000.000
zipcode_98109	311900.0000	12900.000	24.230	0.000	287000.000	337000.000
zipcode_98105	305100.0000	8436.754	36.162	0.000	289000.000	322000.000
zipcode_98112	303300.0000	9963.820	30.437	0.000	284000.000	323000.000
...	...	...	...	...	...	...
basement_present	-5546.0095	721.806	-7.684	0.000	-6960.868	-4131.151
floors	-7602.7410	836.628	-9.087	0.000	-9242.669	-5962.812
zipcode_98032	-12230.0000	8048.185	-1.519	0.129	-28000.000	3547.731
zipcode_98002	-12900.0000	6672.443	-1.933	0.053	-26000.000	178.832
zipcode_98023	-15750.0000	5246.049	-3.002	0.003	-26000.000	-5466.508

78 rows × 6 columns

4.1.1 Check Linear Model Assumptions

1. Linearity

2. Residual Normality
sm.graphics.qqplot(model.resid, dist=stats.norm, line='45', fit=True)
Omnibus Value

3. Homoskedasticity
Durbin-Watson: range of 1.5 to 2.5 is relatively normal

4. Multicollinearity
VIF (variance_inflation_factor())

Also check p-value
A p-value less than 0.05 (typically ≤ 0.05) is statistically significant. It indicates strong evidence against the null hypothesis, as there is less than a 5% probability the null is correct (and the results are random).

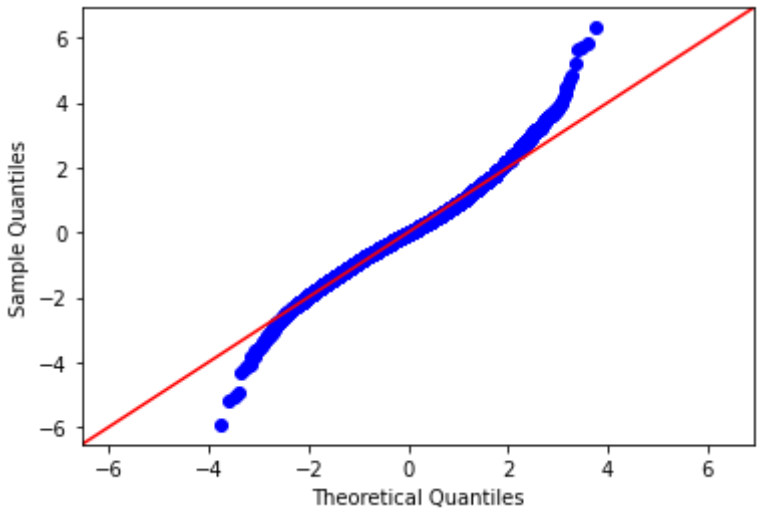
Check for overfitting

- Mean Absolute Error (MAE)

- Mean Sqaured Error (MSE)
- Root Mean Sqaured Error (RMSE)

In [104]:
```
# Check linearity and residual normality

sm.graphics.qqplot(model1.resid, dist=stats.norm, line='45', fit=True);
```

executed in 111ms, finished 12:32:59 2021-11-07



In [105]:
```
# Check for Multicollinearity

def create_vif_dct(dataframe, const_col_name='const'):

    if const_col_name not in dataframe.columns:
        dataframe = sm.add_constant(dataframe)

    # Dummy-checking
    df = dataframe.select_dtypes('number')
    if df.shape != dataframe.shape:
        warnings.warn('\n\nThere are non-numerical columns trying to be pas
    if df.isna().sum().any():
        raise ValueError('There may not be any missing values in the datafr

    # Creating VIF Dictionary
    vif_dct = {}

    # Loop through each row and set the variable name to the VIF
    for i in range(len(df.columns)):
        vif = variance_inflation_factor(df.values, i)
        v = df.columns[i]
        vif_dct[v] = vif

    return vif_dct
```

executed in 5ms, finished 12:32:59 2021-11-07

In [106]:
```
# Check for multicolinearity - anything over 10 is not good!

create_vif_dct(X_train_scaled)
```

executed in 2.31s, finished 12:33:02 2021-11-07

Out[106]:
```
{'const': 48.804888593432594,
 'bedrooms': 1.7525266115446538,
 'bathrooms': 2.520977861021298,
 'sqft_living': 3.0332438247636637,
 'floors': 2.0885480888237606,
 'waterfront': 1.0774400957148569,
 'condition': 1.1822282938705688,
 'grade': 1.996598740009202,
 'renovated': 1.0229074802205151,
 'basement_present': 1.554607741508546,
 'zipcode_98002': 1.572374400097395,
 'zipcode_98003': 1.7683916273984623,
 'zipcode_98004': 1.065100539678895,
 'zipcode_98005': 1.1552525010993069,
 'zipcode_98006': 1.5495760787475503,
 'zipcode_98007': 1.2999857058283855,
 'zipcode_98008': 1.6562704322361503,
 'zipcode_98010': 1.2621828886369568,
 'zipcode_98011': 1.4914193011733052,
 'zipcode_98014': 1.264640922782956,
 'zipcode_98019': 1.5651862005750914,
 'zipcode_98022': 1.6764572920324579,
 'zipcode_98023': 2.401885345658481,
 'zipcode_98024': 1.1811733768623345,
 'zipcode_98027': 1.8251705776763347,
 'zipcode_98028': 1.7782165922916775,
 'zipcode_98029': 1.688382860504492,
 'zipcode_98030': 1.695277852470107,
 'zipcode_98031': 1.8239180673146302,
 'zipcode_98032': 1.3345289185972118,
 'zipcode_98033': 1.6002524120784773,
 'zipcode_98034': 2.336472752494877,
 'zipcode_98038': 2.529762947645856,
 'zipcode_98040': 1.0695100160422801,
 'zipcode_98042': 2.48919498398572,
 'zipcode_98045': 1.5684405390947247,
 'zipcode_98052': 1.9670133379258323,
 'zipcode_98053': 1.6383968948327174,
 'zipcode_98055': 1.7506671291350397,
 'zipcode_98056': 2.08370547501707,
 'zipcode_98058': 2.230646819307026,
 'zipcode_98059': 2.1268217530668387,
 'zipcode_98065': 1.772519797918147,
 'zipcode_98070': 1.3587003781601552,
 'zipcode_98072': 1.610588276152504,
 'zipcode_98074': 1.7180666400873077,
 'zipcode_98075': 1.316109939412652,
 'zipcode_98077': 1.2926215328810982,
 'zipcode_98092': 1.9374125918788754,
 'zipcode_98102': 1.1593679933535124,
 'zipcode_98103': 2.3265534775693357,
```

```
 'zipcode_98105': 1.3084770106186392,
 'zipcode_98106': 1.9318565444070566,
 'zipcode_98107': 1.7119511051672336,
 'zipcode_98108': 1.5428738521577468,
 'zipcode_98109': 1.1155508114641512,
 'zipcode_98112': 1.2110043603791862,
 'zipcode_98115': 2.164337059909012,
 'zipcode_98116': 1.6821988102483125,
 'zipcode_98117': 2.280513896798148,
 'zipcode_98118': 2.1946814064237246,
 'zipcode_98119': 1.2370712597262954,
 'zipcode_98122': 1.5701214411554572,
 'zipcode_98125': 2.0238874056833405,
 'zipcode_98126': 1.9755145425871357,
 'zipcode_98133': 2.4169932736595734,
 'zipcode_98136': 1.6342507660544252,
 'zipcode_98144': 1.7882327681733308,
 'zipcode_98146': 1.713299220605527,
 'zipcode_98148': 1.1564530569895037,
 'zipcode_98155': 2.228779395730326,
 'zipcode_98166': 1.651663330462584,
 'zipcode_98168': 1.6590811305558477,
 'zipcode_98177': 1.487356216897902,
 'zipcode_98178': 1.7056064002303757,
 'zipcode_98188': 1.3748946030663856,
 'zipcode_98198': 1.7179126771024968,
 'zipcode_98199': 1.4534675049413095}
```

In [107]:
```
# Predictions on the training set

y_train_pred = model1.predict(X_train_scaled)
y_test_pred = model1.predict(X_test_scaled)
```

executed in 4ms, finished 12:33:02 2021-11-07

In [108]:
```
# Regression metrics for Train data
# Mean Absolute Error (MAE)
# Mean Sqaured Error (MSE)
# Root Mean Sqaured Error (RMSE)
# Is this only done once?

train_mae = mean_absolute_error(y_train, y_train_pred)
train_mse = mean_squared_error(y_train, y_train_pred)
train_rmse = np.sqrt(train_mse)

print('Train MAE:', train_mae)
print('Train MSE:', train_mse)
print('Train RMSE:', train_rmse)
```

executed in 4ms, finished 12:33:02 2021-11-07

```
Train MAE: 47495.546251964
Train MSE: 3974372692.5231338
Train RMSE: 63042.62599640924
```

In [109]:
```
# Regression metrics for Test data

test_mae = mean_absolute_error(y_test, y_test_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
test_rmse = np.sqrt(test_mse)

print('Test MAE:', test_mae)
print('Test MSE:', test_mse)
print('Test RMSE:', test_rmse)
```

executed in 5ms, finished 12:33:02 2021-11-07

```
Test MAE: 48683.51726441247
Test MSE: 4208824782.898014
Test RMSE: 64875.45593595481
```

4.2 Model 2: All Features Excluding Zipcode

In [110]:
```
# Model without zipcodes

[c for c in X_train_scaled.columns if not c.startswith('zipcode')]
```

executed in 4ms, finished 12:33:02 2021-11-07

Out[110]:
```
['const',
 'bedrooms',
 'bathrooms',
 'sqft_living',
 'floors',
 'waterfront',
 'condition',
 'grade',
 'renovated',
 'basement_present']
```

In [111]:
```
model2 = sm.OLS(y_train, X_train_scaled[[c for c in X_train_scaled.columns
model2.summary2()
```

executed in 35ms, finished 12:33:02 2021-11-07

Out[111]:

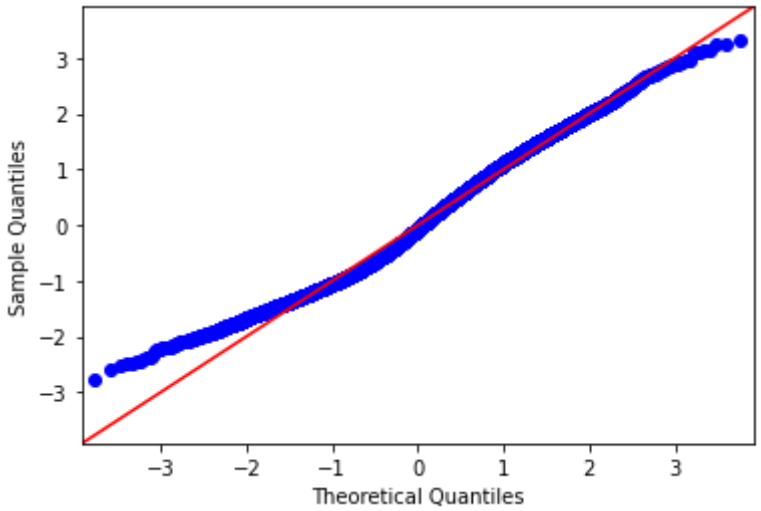
Model:	OLS	Adj. R-squared:	0.274
Dependent Variable:	price	AIC:	310058.2259
Date:	2021-11-07 12:33	BIC:	310132.0999
No. Observations:	11937	Log-Likelihood:	-1.5502e+05
Df Model:	9	F-statistic:	502.4
Df Residuals:	11927	Prob (F-statistic):	0.00
R-squared:	0.275	Scale:	1.1163e+10

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
const	399995.9067	967.0205	413.6375	0.0000	398100.3890	401891.4243
bedrooms	-9259.6531	1252.8124	-7.3911	0.0000	-11715.3696	-6803.9366
bathrooms	-7685.7973	1492.9577	-5.1480	0.0000	-10612.2376	-4759.3570
sqft_living	32467.9265	1589.2035	20.4303	0.0000	29352.8288	35583.0242
floors	13156.1308	1279.6647	10.2809	0.0000	10647.7796	15664.4820
waterfront	3319.7723	968.5369	3.4276	0.0006	1421.2822	5218.2624
condition	10388.1898	1020.9067	10.1755	0.0000	8387.0464	12389.3333
grade	40496.1086	1306.6716	30.9918	0.0000	37934.8194	43057.3978
renovated	4306.2156	970.8841	4.4354	0.0000	2403.1245	6209.3066
basement_present	20118.1663	1080.4635	18.6199	0.0000	18000.2819	22236.0508

Omnibus:	560.278	Durbin-Watson:	2.020
Prob(Omnibus):	0.000	Jarque-Bera (JB):	331.293
Skew:	0.266	Prob(JB):	0.000
Kurtosis:	2.381	Condition No.:	3

In [112]:
```
# Check linearity and residual normality

sm.graphics.qqplot(model2.resid, dist=stats.norm, line='45', fit=True);
```

executed in 119ms, finished 12:33:02 2021-11-07



In [113]:
```
create_vif_dct(X_train_scaled[[c for c in X_train_scaled.columns if not c.s
```

executed in 44ms, finished 12:33:02 2021-11-07

Out[113]:
```
{'const': 1.0,
 'bedrooms': 1.6784204890659655,
 'bathrooms': 2.3835466342659077,
 'sqft_living': 2.7007704622217075,
 'floors': 1.7511405980093482,
 'waterfront': 1.003138717164317,
 'condition': 1.114553090585022,
 'grade': 1.8258350881548409,
 'renovated': 1.0080067669410961,
 'basement_present': 1.2483858731730055}
```

4.3 MODEL VI - Combining Features Into One Model

In [114]:
```
model_vi = sm.OLS(y_train, X_train_scaled[['const', 'sqft_living','floors',
        'zipcode_98023', 'zipcode_98034','zipcode_98133','zipcode_98042',
model_vi.summary()
```

executed in 21ms, finished 12:33:02 2021-11-07

zipcode_98023	-1.487e+05	5477.516	-27.149	0.000	-1.59e+05	-1.38e+05
zipcode_98034	3.045e+04	5589.444	5.447	0.000	1.95e+04	4.14e+04
zipcode_98133	-1.6e+04	5469.389	-2.925	0.003	-2.67e+04	-5277.816
zipcode_98042	-1.179e+05	5291.954	-22.277	0.000	-1.28e+05	-1.08e+05
zipcode_98038	-8.43e+04	5273.910	-15.985	0.000	-9.46e+04	-7.4e+04

Omnibus:	253.525	**Durbin-Watson:**	2.020
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	194.004
Skew:	0.225	**Prob(JB):**	7.46e-43
Kurtosis:	2.567	**Cond. No.**	8.78

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

In [115]:
```
model_vii = sm.OLS(y_train, X_train_scaled[
                                          ['const', 'bedrooms', 'bathroom
                                           'condition','grade','renovated
                                           'zipcode_98038', 'zipcode_9804
                                           'zipcode_98023', 'zipcode_9811
                                           'zipcode_98155', 'zipcode_9811
                                           'zipcode_98125', 'zipcode_9809
                                           'zipcode_98059', 'zipcode_9810
                                           'zipcode_98031', 'zipcode_9800
                                           'zipcode_98065', 'zipcode_9803
                                           'zipcode_98074', 'zipcode_9817
                                           'zipcode_98022', 'zipcode_9802
                                           'zipcode_98053', 'zipcode_9800
                                           ]]).fit()
model_vii.summary()
```

executed in 53ms, finished 12:33:02 2021-11-07

zipcode_98107	1.172e+05	6450.715	18.164	0.000	1.05e+05	1.3e+05
zipcode_98166	-4.721e+04	6394.965	-7.382	0.000	-5.97e+04	-3.47e+04
zipcode_98053	6.066e+04	6506.307	9.324	0.000	4.79e+04	7.34e+04
zipcode_98008	7.098e+04	6420.251	11.056	0.000	5.84e+04	8.36e+04
zipcode_98072	1.54e+04	6564.236	2.345	0.019	2528.185	2.83e+04

Omnibus:	270.308	**Durbin-Watson:**	2.016
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	488.488
Skew:	0.178	**Prob(JB):**	8.44e-107
Kurtosis:	3.925	**Cond. No.**	25.7

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

In [116]:
```
model_viii = sm.OLS(y_train, X_train_scaled[
                                          ['const', 'bedrooms', 'bathroom
                                           'condition','grade',
                                           'zipcode_98038', 'zipcode_9804
                                           'zipcode_98023', 'zipcode_9811
                                           'zipcode_98155', 'zipcode_9811
                                           'zipcode_98125', 'zipcode_9809
                                           'zipcode_98059', 'zipcode_9810
                                           'zipcode_98031', 'zipcode_9800
                                           'zipcode_98065', 'zipcode_9803
                                           'zipcode_98074', 'zipcode_9817
                                           'zipcode_98022', 'zipcode_9802
                                           'zipcode_98053', 'zipcode_9800
                                           ]]).fit()
model_viii.summary()
```

executed in 48ms, finished 12:33:02 2021-11-07

Out[116]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.594
Model:	OLS	**Adj. R-squared:**	0.593
Method:	Least Squares	**F-statistic:**	378.6
Date:	Sun, 07 Nov 2021	**Prob (F-statistic):**	0.00
Time:	12:33:02	**Log-Likelihood:**	-1.5155e+05
No. Observations:	11937	**AIC:**	3.032e+05
Df Residuals:	11890	**BIC:**	3.035e+05
Df Model:	46		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.243e+05	1514.371	280.169	0.000	4.21e+05	4.27e+05
bedrooms	-5846.3693	950.573	-6.150	0.000	-7709.647	-3983.092
bathrooms	1885.2190	1116.215	1.689	0.091	-302.745	4073.183
sqft_living	4.915e+04	1208.369	40.673	0.000	4.68e+04	5.15e+04
floors	-4210.8972	931.176	-4.522	0.000	-6036.154	-2385.640
waterfront	4848.5357	728.354	6.657	0.000	3420.842	6276.229
condition	9661.6760	777.909	12.420	0.000	8136.847	1.12e+04
grade	3.363e+04	1006.737	33.409	0.000	3.17e+04	3.56e+04
zipcode_98038	-9.973e+04	4317.858	-23.098	0.000	-1.08e+05	-9.13e+04
zipcode_98042	-1.34e+05	4323.998	-30.987	0.000	-1.42e+05	-1.26e+05
zipcode_98133	-2.647e+04	4458.504	-5.936	0.000	-3.52e+04	-1.77e+04
zipcode_98034	1.513e+04	4550.279	3.325	0.001	6211.093	2.4e+04
zipcode_98023	-1.658e+05	4457.706	-37.198	0.000	-1.75e+05	-1.57e+05

zipcode_98118	-1.777e+04	4852.669	-3.662	0.000	-2.73e+04	-8256.803
zipcode_98058	-1.073e+05	4717.716	-22.753	0.000	-1.17e+05	-9.81e+04
zipcode_98103	8.382e+04	4826.380	17.366	0.000	7.44e+04	9.33e+04
zipcode_98155	-2.506e+04	4740.521	-5.286	0.000	-3.44e+04	-1.58e+04
zipcode_98117	1.018e+05	4736.929	21.492	0.000	9.25e+04	1.11e+05
zipcode_98115	1.007e+05	4929.541	20.436	0.000	9.11e+04	1.1e+05
zipcode_98056	-5.739e+04	5024.052	-11.424	0.000	-6.72e+04	-4.75e+04
zipcode_98125	1.575e+04	5165.362	3.048	0.002	5621.331	2.59e+04
zipcode_98092	-1.464e+05	5375.938	-27.234	0.000	-1.57e+05	-1.36e+05
zipcode_98126	6264.9652	5358.860	1.169	0.242	-4239.276	1.68e+04
zipcode_98052	6.385e+04	5295.982	12.056	0.000	5.35e+04	7.42e+04
zipcode_98059	-4.989e+04	4933.774	-10.112	0.000	-5.96e+04	-4.02e+04
zipcode_98106	-6.055e+04	5457.551	-11.095	0.000	-7.13e+04	-4.99e+04
zipcode_98027	2.441e+04	5708.051	4.276	0.000	1.32e+04	3.56e+04
zipcode_98028	-1.237e+04	5837.607	-2.119	0.034	-2.38e+04	-926.365
zipcode_98031	-1.398e+05	5676.382	-24.631	0.000	-1.51e+05	-1.29e+05
zipcode_98003	-1.488e+05	5874.459	-25.332	0.000	-1.6e+05	-1.37e+05
zipcode_98055	-1.087e+05	5930.030	-18.325	0.000	-1.2e+05	-9.7e+04
zipcode_98144	2.343e+04	5980.197	3.918	0.000	1.17e+04	3.52e+04
zipcode_98065	4519.2460	6009.425	0.752	0.452	-7260.210	1.63e+04
zipcode_98030	-1.404e+05	6149.610	-22.837	0.000	-1.52e+05	-1.28e+05
zipcode_98198	-1.168e+05	6112.938	-19.110	0.000	-1.29e+05	-1.05e+05
zipcode_98146	-6.337e+04	6146.651	-10.309	0.000	-7.54e+04	-5.13e+04
zipcode_98074	4.809e+04	6145.960	7.824	0.000	3.6e+04	6.01e+04
zipcode_98178	-1.06e+05	6188.362	-17.136	0.000	-1.18e+05	-9.39e+04
zipcode_98116	9.647e+04	6356.201	15.177	0.000	8.4e+04	1.09e+05
zipcode_98168	-1.206e+05	6451.844	-18.687	0.000	-1.33e+05	-1.08e+05
zipcode_98022	-1.19e+05	6270.925	-18.980	0.000	-1.31e+05	-1.07e+05
zipcode_98029	5.718e+04	6339.675	9.019	0.000	4.47e+04	6.96e+04
zipcode_98107	1.202e+05	6428.753	18.698	0.000	1.08e+05	1.33e+05
zipcode_98166	-4.647e+04	6401.602	-7.260	0.000	-5.9e+04	-3.39e+04
zipcode_98053	5.755e+04	6488.655	8.870	0.000	4.48e+04	7.03e+04
zipcode_98008	6.966e+04	6425.601	10.841	0.000	5.71e+04	8.23e+04
zipcode_98072	1.44e+04	6571.852	2.191	0.028	1519.519	2.73e+04

Omnibus:	274.230	**Durbin-Watson:**	2.017

Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	495.916
Skew:	0.181	**Prob(JB):**	2.06e-108
Kurtosis:	3.930	**Cond. No.**	25.7

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

In [117]:
```
create_vif_dct(X_train_scaled[['const', 'bedrooms', 'bathrooms', 'sqft_livi
                                                'condition','grade',
                                                'zipcode_98038', 'zipcode_9804
                                                'zipcode_98023', 'zipcode_9811
                                                'zipcode_98155', 'zipcode_9811
                                                'zipcode_98125', 'zipcode_9809
                                                'zipcode_98059', 'zipcode_9810
                                                'zipcode_98031', 'zipcode_9800
                                                'zipcode_98065', 'zipcode_9803
                                                'zipcode_98074', 'zipcode_9817
                                                'zipcode_98022', 'zipcode_9802
                                                'zipcode_98053', 'zipcode_9800
```

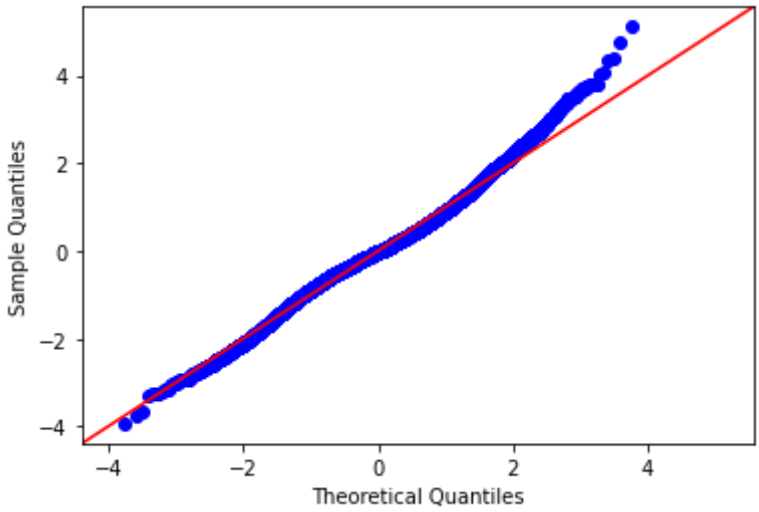
executed in 790ms, finished 12:33:03 2021-11-07

Out[117]:
```
{'const': 4.369725983879623,
 'bedrooms': 1.7217107677678187,
 'bathrooms': 2.3740262731332087,
 'sqft_living': 2.7822034535701605,
 'floors': 1.6521639283198817,
 'waterfront': 1.0108229358922625,
 'condition': 1.1530477044209724,
 'grade': 1.9311768338639281,
 'zipcode_98038': 1.1310582307350479,
 'zipcode_98042': 1.1119573653335004,
 'zipcode_98133': 1.1048559490438092,
 'zipcode_98034': 1.1010391596644689,
 'zipcode_98023': 1.1074410168301358,
 'zipcode_98118': 1.0921759799392237,
 'zipcode_98058': 1.0929327058184868,
 'zipcode_98103': 1.1403385012782687,
 'zipcode_98155': 1.0933301221801928,
 'zipcode_98117': 1.0984603918015572,
 'zipcode_98115': 1.0864756529302824,
 'zipcode_98056': 1.0863067171350993,
 'zipcode_98125': 1.0750438394752115,
 'zipcode_98092': 1.076052729999994,
 'zipcode_98126': 1.0780274053189864,
 'zipcode_98052': 1.0743536168784191,
 'zipcode_98059': 1.0920411077296286,
 'zipcode_98106': 1.0724286270863268,
 'zipcode_98027': 1.062916758805177,
 'zipcode_98028': 1.059169145684753,
 'zipcode_98031': 1.0610813451510335,
 'zipcode_98003': 1.0566026776285689,
 'zipcode_98055': 1.0549609390062185,
 'zipcode_98144': 1.061832605422847,
 'zipcode_98065': 1.083399255936633,
 'zipcode_98030': 1.0526281865251654,
 'zipcode_98198': 1.0574706129067932,
 'zipcode_98146': 1.0574667844366896,
 'zipcode_98074': 1.0630781153416873,
 'zipcode_98178': 1.0600041905229836,
 'zipcode_98116': 1.049370290096881,
 'zipcode_98168': 1.0618004887674302,
 'zipcode_98022': 1.051906702404663,
 'zipcode_98029': 1.0688645161908064,
 'zipcode_98107': 1.0734626251617383,
```

```
 'zipcode_98166': 1.051691153998909,
 'zipcode_98053': 1.0543317642868377,
 'zipcode_98008': 1.059591481845864,
 'zipcode_98072': 1.047976247703205}
```

In [118]:
```
# Check linearity and residual normality

sm.graphics.qqplot(model_viii.resid, dist=stats.norm, line='45', fit=True);
```

executed in 100ms, finished 12:33:03 2021-11-07



5 FINAL MODEL

** Final Model includes:

1. Bedrooms
2. Bathrooms
3. Square Feet Living
4. Floors
5. Waterfront
6. Condition
7. Grade
8. Renovation Status
9. Basement Present
10. All Zipcodes

Excluding zipcodes, **Square Feet Living, Grade, and Condition are the strongest determinants of price**

In [119]:
```
model_final = sm.OLS(y_train, X_train_scaled).fit()
model_final.summary2()
```

executed in 122ms, finished 12:33:03 2021-11-07

Out[119]:

Model:	OLS	Adj. R-squared:	0.740
Dependent Variable:	price	AIC:	297876.8343
Date:	2021-11-07 12:33	BIC:	298453.0513
No. Observations:	11937	Log-Likelihood:	-1.4886e+05
Df Model:	77	F-statistic:	442.0
Df Residuals:	11859	Prob (F-statistic):	0.00
R-squared:	0.742	Scale:	4.0005e+09

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
const	273461.5916	4044.2866	67.6168	0.0000	265534.1265	281389.0568
bedrooms	-3777.9235	766.3772	-4.9296	0.0000	-5280.1485	-2275.6984
bathrooms	4006.3214	919.1677	4.3586	0.0000	2204.6018	5808.0409
sqft_living	57514.2433	1008.2400	57.0442	0.0000	55537.9275	59490.5591
floors	-7602.7410	836.6282	-9.0874	0.0000	-9242.6695	-5962.8125
waterfront	5088.3292	600.9064	8.4678	0.0000	3910.4541	6266.2043
condition	9541.0686	629.4496	15.1578	0.0000	8307.2442	10774.8930
grade	26074.4898	818.0044	31.8757	0.0000	24471.0669	27677.9126
renovated	1469.4406	585.5021	2.5097	0.0121	321.7605	2617.1207
basement_present	-5546.0095	721.8063	-7.6835	0.0000	-6960.8682	-4131.1507
zipcode_98002	-12900.2509	6672.4427	-1.9334	0.0532	-25979.3332	178.8315
zipcode_98003	1740.5176	6073.0427	0.2866	0.7744	-10163.6424	13644.6776
zipcode_98004	371396.6894	16329.9337	22.7433	0.0000	339387.3405	403406.0382
zipcode_98005	274013.6629	11045.7926	24.8071	0.0000	252362.0975	295665.2283
zipcode_98006	205246.6476	6840.1134	30.0063	0.0000	191838.9032	218654.3920
zipcode_98007	217263.7113	8409.3343	25.8360	0.0000	200780.0365	233747.3860
zipcode_98008	218106.1136	6419.7078	33.9745	0.0000	205522.4332	230689.7940
zipcode_98010	81149.0139	8837.8775	9.1820	0.0000	63825.3243	98472.7036
zipcode_98011	152927.1362	6972.9153	21.9316	0.0000	139259.0784	166595.1941
zipcode_98014	109662.0470	8779.5741	12.4906	0.0000	92452.6414	126871.4525
zipcode_98019	109764.3159	6680.2900	16.4311	0.0000	96669.8517	122858.7801
zipcode_98022	27124.5154	6326.2211	4.2876	0.0000	14724.0843	39524.9465
zipcode_98023	-15749.6244	5246.0488	-3.0022	0.0027	-26032.7406	-5466.5082
zipcode_98024	146195.0964	10266.6456	14.2398	0.0000	126070.7867	166319.4060
zipcode_98027	176260.9931	5977.1620	29.4891	0.0000	164544.7751	187977.2110

zipcode_98028	136320.8952	6044.3505	22.5534	0.0000	124472.9768	148168.8137
zipcode_98029	212325.4807	6367.1670	33.3469	0.0000	199844.7890	224806.1725
zipcode_98030	7039.3493	6236.4236	1.1287	0.2590	-5185.0639	19263.7626
zipcode_98031	8528.6375	5947.0968	1.4341	0.1516	-3128.6478	20185.9228
zipcode_98032	-12228.0316	8048.1848	-1.5194	0.1287	-28003.7941	3547.7309
zipcode_98033	243677.5076	6552.8165	37.1867	0.0000	230832.9123	256522.1030
zipcode_98034	164507.8973	5296.9041	31.0574	0.0000	154125.0964	174890.6982
zipcode_98038	46302.0157	5160.2523	8.9728	0.0000	36187.0748	56416.9567
zipcode_98040	326034.5377	15875.7881	20.5366	0.0000	294915.3887	357153.6868
zipcode_98042	12705.2104	5169.8264	2.4576	0.0140	2571.5026	22838.9181
zipcode_98045	98823.0902	6641.1928	14.8803	0.0000	85805.2629	111840.9175
zipcode_98052	215402.6310	5726.3997	37.6157	0.0000	204177.9482	226627.3139
zipcode_98053	206380.7334	6463.6923	31.9292	0.0000	193710.8361	219050.6307
zipcode_98055	39586.2889	6104.4362	6.4848	0.0000	27620.5925	51551.9853
zipcode_98056	89444.2979	5560.3379	16.0861	0.0000	78545.1235	100343.4723
zipcode_98058	40893.6048	5385.8663	7.5928	0.0000	30336.4233	51450.7863
zipcode_98059	94956.6570	5502.1248	17.2582	0.0000	84171.5898	105741.7241
zipcode_98065	147969.0730	6142.4173	24.0897	0.0000	135928.9274	160009.2186
zipcode_98070	140768.2153	8714.9958	16.1524	0.0000	123685.3938	157851.0367
zipcode_98072	164199.6052	6510.4262	25.2210	0.0000	151438.1018	176961.1086
zipcode_98074	201284.7308	6243.5589	32.2388	0.0000	189046.3312	213523.1305
zipcode_98075	234327.9686	8190.9122	28.6083	0.0000	218272.4370	250383.5001
zipcode_98077	162743.2665	8442.3655	19.2770	0.0000	146194.8452	179291.6877
zipcode_98092	1469.8656	5764.3941	0.2550	0.7987	-9829.2924	12769.0236
zipcode_98102	297088.5562	11696.2982	25.4002	0.0000	274161.8930	320015.2194
zipcode_98103	244078.3724	5508.9174	44.3061	0.0000	233279.9906	254876.7543
zipcode_98105	305088.9218	8436.7538	36.1619	0.0000	288551.5003	321626.3433
zipcode_98106	93248.0402	5853.3725	15.9307	0.0000	81774.4699	104721.6106
zipcode_98107	281918.8926	6487.5995	43.4550	0.0000	269202.1333	294635.6520
zipcode_98108	102847.1230	6800.2672	15.1240	0.0000	89517.4839	116176.7622
zipcode_98109	311859.4786	12871.0024	24.2296	0.0000	286630.2026	337088.7547
zipcode_98112	303265.9515	9963.8199	30.4367	0.0000	283735.2300	322796.6731
zipcode_98115	256862.6724	5559.8617	46.1995	0.0000	245964.4313	267760.9134
zipcode_98116	253719.7551	6430.9779	39.4527	0.0000	241113.9835	266325.5268
zipcode_98117	258242.1319	5454.1378	47.3479	0.0000	247551.1272	268933.1366
zipcode_98118	134570.4171	5496.9936	24.4807	0.0000	123795.4079	145345.4263

zipcode_98119	298357.7687	9594.9537	31.0953	0.0000	279550.0854	317165.4520
zipcode_98122	241509.4396	6810.3713	35.4620	0.0000	228159.9947	254858.8845
zipcode_98125	168574.7553	5663.5136	29.7650	0.0000	157473.3396	179676.1710
zipcode_98126	161024.4470	5796.9920	27.7772	0.0000	149661.3916	172387.5023
zipcode_98133	126406.2882	5269.6185	23.9877	0.0000	116076.9714	136735.6050
zipcode_98136	210242.1271	6579.1944	31.9556	0.0000	197345.8268	223138.4275
zipcode_98144	181420.9715	6201.6122	29.2538	0.0000	169264.7942	193577.1488
zipcode_98146	84728.6154	6252.1137	13.5520	0.0000	72473.4470	96983.7839
zipcode_98148	35848.0297	10909.3826	3.2860	0.0010	14463.8502	57232.2092
zipcode_98155	124424.9608	5408.6518	23.0048	0.0000	113823.1159	135026.8057
zipcode_98166	101217.5922	6410.7730	15.7887	0.0000	88651.4254	113783.7589
zipcode_98168	27999.3152	6444.6792	4.3446	0.0000	15366.6869	40631.9436
zipcode_98177	185891.7024	7048.3104	26.3739	0.0000	172075.8577	199707.5471
zipcode_98178	41204.4668	6272.8717	6.5687	0.0000	28908.6093	53500.3244
zipcode_98188	24276.5388	7679.7197	3.1611	0.0016	9223.0284	39330.0491
zipcode_98198	31612.4188	6226.1773	5.0773	0.0000	19408.0899	43816.7477
zipcode_98199	267319.0664	7404.9794	36.0999	0.0000	252804.0919	281834.0408

Omnibus:	561.223	Durbin-Watson:	2.024
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1486.527
Skew:	0.243	Prob(JB):	0.000
Kurtosis:	4.659	Condition No.:	100

In [120]:
```
# Results sorted by coefficients descending

results_as_html = model_final.summary().tables[1].as_html()
results = pd.read_html(results1_as_html, header=0, index_col=0)[0]
results.sort_values('coef', ascending=False)
```

executed in 39ms, finished 12:33:03 2021-11-07

Out[120]:

	coef	std err	t	P>|t|	[0.025	0.975]
zipcode_98004	371400.0000	16300.000	22.743	0.000	339000.000	403000.000
zipcode_98040	326000.0000	15900.000	20.537	0.000	295000.000	357000.000
zipcode_98109	311900.0000	12900.000	24.230	0.000	287000.000	337000.000
zipcode_98105	305100.0000	8436.754	36.162	0.000	289000.000	322000.000
zipcode_98112	303300.0000	9963.820	30.437	0.000	284000.000	323000.000
...	...	...	...	...	...	...
basement_present	-5546.0095	721.806	-7.684	0.000	-6960.868	-4131.151
floors	-7602.7410	836.628	-9.087	0.000	-9242.669	-5962.812
zipcode_98032	-12230.0000	8048.185	-1.519	0.129	-28000.000	3547.731
zipcode_98002	-12900.0000	6672.443	-1.933	0.053	-26000.000	178.832
zipcode_98023	-15750.0000	5246.049	-3.002	0.003	-26000.000	-5466.508

78 rows × 6 columns

5.0.1 Check Linear Model Assumptions

1. Linearity

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

2. Residual Normality

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

3. Homoskedasticity

- Durbin-Watson: range of 1.5 to 2.5 is relatively normal

4. Multicollinearity

- VIF (variance_inflation_factor())
- Anything above 10 needs to be removed

Also check p-value

- A p-value less than 0.05 (typically ≤ 0.05) is statistically significant. It indicates strong evidence against the null hypothesis, as there is less than a 5% probability the null is correct (and the

results are random).

Check for overfitting

- Test vs. Train Data, compare:

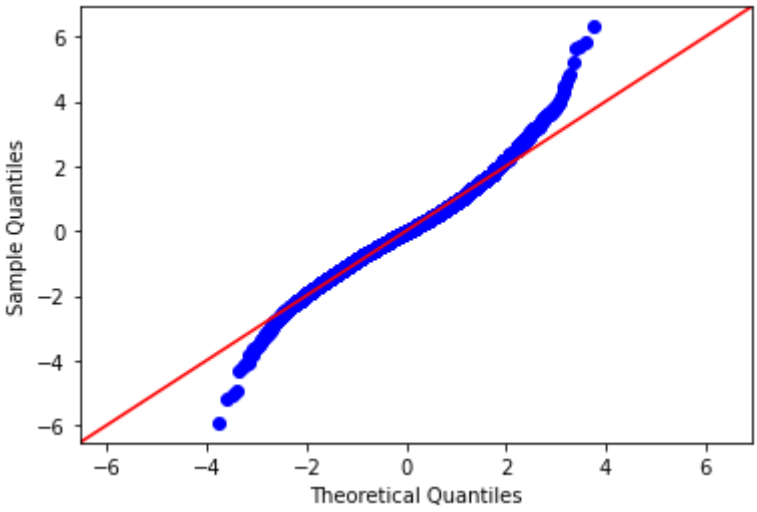
1. Mean Absolute Error (MAE)
2. Mean Sqaured Error (MSE)
3. Root Mean Sqaured Error (RMSE)

5.1 Check for Linearity and Residual Normality using Q-Q Plot

- There are some tails but overall residuals appear normal

In [121]:
```
# Check Linearity and Residual Normality

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

executed in 115ms, finished 12:33:03 2021-11-07



5.2 Check for Multicolinearity using VIF (Variance Inflation Factor)

- Removed all variance inflation factors above 10
- All remaining are below 10

In [122]:
```
# Check for multicolinearity - anything over 10 is not good!

create_vif_dct(X_train_scaled)
```

executed in 2.27s, finished 12:33:06 2021-11-07

Out[122]:
```
{'const': 48.804888593432594,
 'bedrooms': 1.7525266115446538,
 'bathrooms': 2.520977861021298,
 'sqft_living': 3.0332438247636637,
 'floors': 2.0885480888237606,
 'waterfront': 1.0774400957148569,
 'condition': 1.1822282938705688,
 'grade': 1.996598740009202,
 'renovated': 1.0229074802205151,
 'basement_present': 1.554607741508546,
 'zipcode_98002': 1.572374400097395,
 'zipcode_98003': 1.7683916273984623,
 'zipcode_98004': 1.065100539678895,
 'zipcode_98005': 1.1552525010993069,
 'zipcode_98006': 1.5495760787475503,
 'zipcode_98007': 1.2999857058283855,
 'zipcode_98008': 1.6562704322361503,
 'zipcode_98010': 1.2621828886369568,
 'zipcode_98011': 1.4914193011733052,
 'zipcode_98014': 1.264640922782956,
 'zipcode_98019': 1.5651862005750914,
 'zipcode_98022': 1.6764572920324579,
 'zipcode_98023': 2.401885345658481,
 'zipcode_98024': 1.1811733768623345,
 'zipcode_98027': 1.8251705776763347,
 'zipcode_98028': 1.7782165922916775,
 'zipcode_98029': 1.688382860504492,
 'zipcode_98030': 1.695277852470107,
 'zipcode_98031': 1.8239180673146302,
 'zipcode_98032': 1.3345289185972118,
 'zipcode_98033': 1.6002524120784773,
 'zipcode_98034': 2.336472752494877,
 'zipcode_98038': 2.529762947645856,
 'zipcode_98040': 1.0695100160422801,
 'zipcode_98042': 2.48919498398572,
 'zipcode_98045': 1.5684405390947247,
 'zipcode_98052': 1.9670133379258323,
 'zipcode_98053': 1.6383968948327174,
 'zipcode_98055': 1.7506671291350397,
 'zipcode_98056': 2.08370547501707,
 'zipcode_98058': 2.230646819307026,
 'zipcode_98059': 2.1268217530668387,
 'zipcode_98065': 1.772519797918147,
 'zipcode_98070': 1.3587003781601552,
 'zipcode_98072': 1.610588276152504,
 'zipcode_98074': 1.7180666400873077,
 'zipcode_98075': 1.316109939412652,
 'zipcode_98077': 1.2926215328810982,
 'zipcode_98092': 1.9374125918788754,
 'zipcode_98102': 1.1593679933535124,
 'zipcode_98103': 2.3265534775693357,
 'zipcode_98105': 1.3084770106186392,
```

```
'zipcode_98106': 1.9318565444070566,
'zipcode_98107': 1.7119511051672336,
'zipcode_98108': 1.5428738521577468,
'zipcode_98109': 1.1155508114641512,
'zipcode_98112': 1.2110043603791862,
'zipcode_98115': 2.164337059909012,
'zipcode_98116': 1.6821988102483125,
'zipcode_98117': 2.280513896798148,
'zipcode_98118': 2.1946814064237246,
'zipcode_98119': 1.2370712597262954,
'zipcode_98122': 1.5701214411554572,
'zipcode_98125': 2.0238874056833405,
'zipcode_98126': 1.9755145425871357,
'zipcode_98133': 2.4169932736595734,
'zipcode_98136': 1.6342507660544252,
'zipcode_98144': 1.7882327681733308,
'zipcode_98146': 1.713299220605527,
'zipcode_98148': 1.1564530569895037,
'zipcode_98155': 2.228779395730326,
'zipcode_98166': 1.651663330462584,
'zipcode_98168': 1.6590811305558477,
'zipcode_98177': 1.487356216897902,
'zipcode_98178': 1.7056064002303757,
'zipcode_98188': 1.3748946030663856,
'zipcode_98198': 1.7179126771024968,
'zipcode_98199': 1.4534675049413095}
```

5.3 Check for Homoskedasticity

- Durbin-Watson: range of 1.5 to 2.5 is relatively normal
- Model's Durbin-Watson is 2.024

5.4 Check for Over-fitting

- Check expected vs. predicted errors of Train Test sets
- Train and Test data are within range of each other
- The average expected error (mean absolute error) of the Train data is $47,496 while the aveage expected error of the Test data is $48,684

In [123]:
```
# Predictions on the training set

y_train_pred = model1.predict(X_train_scaled)
y_test_pred = model1.predict(X_test_scaled)

# Regression metrics for Train data

train_mae = mean_absolute_error(y_train, y_train_pred)
train_mse = mean_squared_error(y_train, y_train_pred)
train_rmse = np.sqrt(train_mse)

# Regression metrics for Test data

test_mae = mean_absolute_error(y_test, y_test_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
test_rmse = np.sqrt(test_mse)

print('Train MAE:', train_mae)
print('Test MAE:', test_mae)

print('Train MSE:', train_mse)
print('Test MSE:', test_mse)

print('Train RMSE:', train_rmse)
print('Test RMSE:', test_rmse)
```

executed in 8ms, finished 12:33:06 2021-11-07

```
Train MAE: 47495.546251964
Test MAE: 48683.51726441247
Train MSE: 3974372692.5231338
Test MSE: 4208824782.898014
Train RMSE: 63042.62599640924
Test RMSE: 64875.45593595481
```

5.5 Look at Unscaled/Raw Data for Price Expectations for Each Feature

In [124]:
```
X_train_raw = sm.add_constant(X_train_raw)
X_test_raw = sm.add_constant(X_test_raw)

model_unscaled = sm.OLS(y_train, X_train_raw).fit()
model_unscaled.summary2()
```

executed in 138ms, finished 12:33:06 2021-11-07

Out[124]:

Model:	OLS	Adj. R-squared:	0.740
Dependent Variable:	price	AIC:	297876.8343
Date:	2021-11-07 12:33	BIC:	298453.0513
No. Observations:	11937	Log-Likelihood:	-1.4886e+05
Df Model:	77	F-statistic:	442.0
Df Residuals:	11859	Prob (F-statistic):	0.00
R-squared:	0.742	Scale:	4.0005e+09

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
const	-149137.4013	8569.2272	-17.4038	0.0000	-165934.4924	-132340.3102
bedrooms	-4531.8375	919.3137	-4.9296	0.0000	-6333.8432	-2729.8318
bathrooms	6163.4048	1414.0660	4.3586	0.0000	3391.6034	8935.2061
sqft_living	93.8186	1.6447	57.0442	0.0000	90.5948	97.0424
floors	-14164.5223	1558.7061	-9.0874	0.0000	-17219.8420	-11109.2026
waterfront	148666.6664	17556.7947	8.4678	0.0000	114252.4686	183080.8641
condition	15007.3150	990.0723	15.1578	0.0000	13066.6108	16948.0191
grade	31571.1934	990.4461	31.8757	0.0000	29629.7566	33512.6302
renovated	9812.2096	3909.6980	2.5097	0.0121	2148.5602	17475.8591
basement_present	-11543.8567	1502.4188	-7.6835	0.0000	-14488.8439	-8598.8695
zipcode_98002	-12900.2509	6672.4427	-1.9334	0.0532	-25979.3332	178.8315
zipcode_98003	1740.5176	6073.0427	0.2866	0.7744	-10163.6424	13644.6776
zipcode_98004	371396.6894	16329.9337	22.7433	0.0000	339387.3405	403406.0382
zipcode_98005	274013.6629	11045.7926	24.8071	0.0000	252362.0975	295665.2283
zipcode_98006	205246.6476	6840.1134	30.0063	0.0000	191838.9032	218654.3920
zipcode_98007	217263.7113	8409.3343	25.8360	0.0000	200780.0365	233747.3860
zipcode_98008	218106.1136	6419.7078	33.9745	0.0000	205522.4332	230689.7940
zipcode_98010	81149.0139	8837.8775	9.1820	0.0000	63825.3243	98472.7036
zipcode_98011	152927.1362	6972.9153	21.9316	0.0000	139259.0784	166595.1941
zipcode_98014	109662.0470	8779.5741	12.4906	0.0000	92452.6414	126871.4525
zipcode_98019	109764.3159	6680.2900	16.4311	0.0000	96669.8517	122858.7801
zipcode_98022	27124.5154	6326.2211	4.2876	0.0000	14724.0843	39524.9465
zipcode_98023	-15749.6244	5246.0488	-3.0022	0.0027	-26032.7406	-5466.5082

zipcode_98024	146195.0964	10266.6456	14.2398	0.0000	126070.7867	166319.4060
zipcode_98027	176260.9931	5977.1620	29.4891	0.0000	164544.7751	187977.2110
zipcode_98028	136320.8952	6044.3505	22.5534	0.0000	124472.9768	148168.8137
zipcode_98029	212325.4807	6367.1670	33.3469	0.0000	199844.7890	224806.1725
zipcode_98030	7039.3493	6236.4236	1.1287	0.2590	-5185.0639	19263.7626
zipcode_98031	8528.6375	5947.0968	1.4341	0.1516	-3128.6478	20185.9228
zipcode_98032	-12228.0316	8048.1848	-1.5194	0.1287	-28003.7941	3547.7309
zipcode_98033	243677.5076	6552.8165	37.1867	0.0000	230832.9123	256522.1030
zipcode_98034	164507.8973	5296.9041	31.0574	0.0000	154125.0964	174890.6982
zipcode_98038	46302.0157	5160.2523	8.9728	0.0000	36187.0748	56416.9567
zipcode_98040	326034.5377	15875.7881	20.5366	0.0000	294915.3887	357153.6868
zipcode_98042	12705.2104	5169.8264	2.4576	0.0140	2571.5026	22838.9181
zipcode_98045	98823.0902	6641.1928	14.8803	0.0000	85805.2629	111840.9175
zipcode_98052	215402.6310	5726.3997	37.6157	0.0000	204177.9482	226627.3139
zipcode_98053	206380.7334	6463.6923	31.9292	0.0000	193710.8361	219050.6307
zipcode_98055	39586.2889	6104.4362	6.4848	0.0000	27620.5925	51551.9853
zipcode_98056	89444.2979	5560.3379	16.0861	0.0000	78545.1235	100343.4723
zipcode_98058	40893.6048	5385.8663	7.5928	0.0000	30336.4233	51450.7863
zipcode_98059	94956.6570	5502.1248	17.2582	0.0000	84171.5898	105741.7241
zipcode_98065	147969.0730	6142.4173	24.0897	0.0000	135928.9274	160009.2186
zipcode_98070	140768.2153	8714.9958	16.1524	0.0000	123685.3938	157851.0367
zipcode_98072	164199.6052	6510.4262	25.2210	0.0000	151438.1018	176961.1086
zipcode_98074	201284.7308	6243.5589	32.2388	0.0000	189046.3312	213523.1305
zipcode_98075	234327.9686	8190.9122	28.6083	0.0000	218272.4370	250383.5001
zipcode_98077	162743.2665	8442.3655	19.2770	0.0000	146194.8452	179291.6877
zipcode_98092	1469.8656	5764.3941	0.2550	0.7987	-9829.2924	12769.0236
zipcode_98102	297088.5562	11696.2982	25.4002	0.0000	274161.8930	320015.2194
zipcode_98103	244078.3724	5508.9174	44.3061	0.0000	233279.9906	254876.7543
zipcode_98105	305088.9218	8436.7538	36.1619	0.0000	288551.5003	321626.3433
zipcode_98106	93248.0402	5853.3725	15.9307	0.0000	81774.4699	104721.6106
zipcode_98107	281918.8926	6487.5995	43.4550	0.0000	269202.1333	294635.6520
zipcode_98108	102847.1230	6800.2672	15.1240	0.0000	89517.4839	116176.7622
zipcode_98109	311859.4786	12871.0024	24.2296	0.0000	286630.2026	337088.7547
zipcode_98112	303265.9515	9963.8199	30.4367	0.0000	283735.2300	322796.6731
zipcode_98115	256862.6724	5559.8617	46.1995	0.0000	245964.4313	267760.9134
zipcode_98116	253719.7551	6430.9779	39.4527	0.0000	241113.9835	266325.5268

zipcode_98117	258242.1319	5454.1378	47.3479	0.0000	247551.1272	268933.1366
zipcode_98118	134570.4171	5496.9936	24.4807	0.0000	123795.4079	145345.4263
zipcode_98119	298357.7687	9594.9537	31.0953	0.0000	279550.0854	317165.4520
zipcode_98122	241509.4396	6810.3713	35.4620	0.0000	228159.9947	254858.8845
zipcode_98125	168574.7553	5663.5136	29.7650	0.0000	157473.3396	179676.1710
zipcode_98126	161024.4470	5796.9920	27.7772	0.0000	149661.3916	172387.5023
zipcode_98133	126406.2882	5269.6185	23.9877	0.0000	116076.9714	136735.6050
zipcode_98136	210242.1271	6579.1944	31.9556	0.0000	197345.8268	223138.4275
zipcode_98144	181420.9715	6201.6122	29.2538	0.0000	169264.7942	193577.1488
zipcode_98146	84728.6154	6252.1137	13.5520	0.0000	72473.4470	96983.7839
zipcode_98148	35848.0297	10909.3826	3.2860	0.0010	14463.8502	57232.2092
zipcode_98155	124424.9608	5408.6518	23.0048	0.0000	113823.1159	135026.8057
zipcode_98166	101217.5922	6410.7730	15.7887	0.0000	88651.4254	113783.7589
zipcode_98168	27999.3152	6444.6792	4.3446	0.0000	15366.6869	40631.9436
zipcode_98177	185891.7024	7048.3104	26.3739	0.0000	172075.8577	199707.5471
zipcode_98178	41204.4668	6272.8717	6.5687	0.0000	28908.6093	53500.3244
zipcode_98188	24276.5388	7679.7197	3.1611	0.0016	9223.0284	39330.0491
zipcode_98198	31612.4188	6226.1773	5.0773	0.0000	19408.0899	43816.7477
zipcode_98199	267319.0664	7404.9794	36.0999	0.0000	252804.0919	281834.0408

Omnibus:	561.223	Durbin-Watson:	2.024
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1486.527
Skew:	0.243	Prob(JB):	0.000
Kurtosis:	4.659	Condition No.:	111890

6 Evaluation and Conclusions

After building models to evaluate the relationship between price and several factors, we can offer guidance to new home buyers in WA State about the expectation of price relative to square feet of living, waterfront views, condition, and grade.

**** Important note: the results are best suited for home buyers seeking homes with a maximum of 6 bedrooms, 4000 square feet, and a budget ranging from $175,000 to $650,000

Conclusions

- **The most important factors in our model, besides zipcode, are: Square Feet Living, Grade, and Condition**
- The **average price for a home in King County, WA is approximately $400,358**
- **Every additional square feet of space costs approximately $94. Note: other models showed this cost could be up to $200 per square feet in the densest zipcodes**

- The grade of a home (1-13) is a strong determinant of price. **Every grade increase costs approximately $31,571**
- The condition (1-5) is also a strong determinant; **Every condition level increase costs approximately $15,007**
- **If the home has been renovated, the price is expected to be approximately $9812 more**

7 Future Work

Future work:

- Refine existing models and expand dataset for different types of home buyers
- Explore relationship of price to zip code
- Build models for Suburbs (Medina, WA) vs. City (Seattle, WA)
- Build more comprehensive models considering other factors such as location, renovations, waterfront view