Final Project Submission

Please fill out:

- Student name: Robert Cauvy
- Student pace: Self paced
- Scheduled project review date/time: 11/05/21 10:00AM ET
- Instructor name: James Irving

TABLE OF CONTENTS

Click to jump to matching Markdown Header.

INTRODUCTION

Business Problem

King County Residents want to renovate their home to increase its resale value but don't know what factors are important for determining a home's value.

The AMCE Real Estate Advisory will provide the client with three things they can do to their property that can increase their property's resale value the most.

By analyzing real estate sales data from the county we will be able to provide the client with insights as to what renovation projects can most increase their property's value.

OBTAIN

Data Understanding

This project will leverage a data set containing sale prices of homes in King County, WA to advise homeowners decide which renovation projects would increase the resale potential of their home.

The data spans just over year, from May 2014 to 2015 and includes over 20,000 house sales.

Since our clients are most concerned with increasing their home's resale value Price will be used as the target variable, and will explore all of the other features of the dataset as predictor variables. These independent features include square footage, grade, condition, and number of bedrooms and bathrooms.

In [31]:
```
import pandas as pd
pd.set_option('display.max_columns',0)
pd.set_option('display.float_format', '{:.2f}'.format)
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter
%matplotlib inline
import seaborn as sns
plt.style.use('seaborn-darkgrid')
from scipy import stats
import math
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn.preprocessing import OneHotEncoder
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#import and read the dataset
df = pd.read_csv('data/kc_house_data.csv',index_col=0)
df
```

Out[2]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfro
7129300520	10/13/2014	221900.00	3	1.00	1180	5650	1.00	n
6414100192	12/9/2014	538000.00	3	2.25	2570	7242	2.00	0.(
5631500400	2/25/2015	180000.00	2	1.00	770	10000	1.00	0.(
2487200875	12/9/2014	604000.00	4	3.00	1960	5000	1.00	0.(
1954400510	2/18/2015	510000.00	3	2.00	1680	8080	1.00	0.(
...	...	...	...	...	...	...	...	
263000018	5/21/2014	360000.00	3	2.50	1530	1131	3.00	0.(
6600060120	2/23/2015	400000.00	4	2.50	2310	5813	2.00	0.(
1523300141	6/23/2014	402101.00	2	0.75	1020	1350	2.00	0.(
291310100	1/16/2015	400000.00	3	2.50	1600	2388	2.00	n
1523300157	10/15/2014	325000.00	2	0.75	1020	1076	2.00	0.(

21597 rows × 20 columns

In [3]:
```
df.info()
```

```
<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
```

Looks like there are a few missing values that must be addressed and some data types that will need to be changed.

In [4]:
```
#Convert to dates from string to datetime type
df['date'] = pd.to_datetime(df['date'])
```

In [5]:
```
#Examine the range of dates in the data
print(df['date'].min())
print(df['date'].max())
```

```
2014-05-02 00:00:00
2015-05-27 00:00:00
```

The data includes house sales in King County and spans roughly over a year from May 2014-2015. The data includes 21597 records.

According to the data glossary, 'view' refers to how many showings the property had prior to its sale. Since this is not a feature of the home we'll drop it.

'Condition' and 'Grade' definitions can be found on the The King County's official site. The glossary on page 26 describes condition and grade as the following:

- Condition: Relative to Age and Grade

  1= Poor Many repairs needed. Showing serious deterioration.

  2= Fair Some repairs needed immediately. Much deferred maintenance.

  3= Average Depending upon age of improvement; normal amount of upkeep for the age of the home.

4= Good Condition above the norm for the age of the home. Indicates extra attention and care has been taken to maintain.

5= Very Good Excellent maintenance and updating on home. Not a total renovation.

- Residential Building Grades

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

  Grade 4 Generally older low quality construction. Does not meet code.

  Grade 5 Lower construction costs and workmanship. Small, simple design.

  Grade 6 Lowest grade currently meeting building codes. Low quality materials, simple designs.

  Grade 7 Average grade of construction and design. Commonly seen in plats and older subdivisions.

  Grade 8 Just above average in construction and design. Usually better materials in both the exterior and interior finishes.

  Grade 9 Better architectural design, with extra exterior and interior design and quality.

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

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

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

  Grade 13 Generally custom designed and built. Approaching the Mansion level. Large amount of highest quality cabinet work, wood trim and marble; large entries

Source: https://www.kingcounty.gov/depts/assessor/Reports/area-reports/2020/residential-westcentral/~/media/depts/assessor/documents/AreaReports/2020/Residential/013.ashx

SCRUB

In [6]:
```
df.isna().sum()
```

Out[6]:
```
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
```

After checking for missing values. The columns that require scrubbing are _waterfront, view and yr*renovated*.

In [7]:
```
df['waterfront'] = df['waterfront'].fillna(0)
```

It's fair to assume records missing a value for *waterfront* do not contain this feature.

In [8]:
```
df['yr_renovated'] = df['yr_renovated'].fillna(0)
```

Values missing for _yr*renovated* can be assumed to have not been renovated.

In [9]:
```
df.drop(['date'],axis=1,inplace=True)
```

Since we already know the data set contains sale records from May 2014 to May 2015 we will not require this column.

In [10]:
```
df.drop(['view'],axis=1,inplace=True)
```

view is described by the number of showings for the property. This column will also be dropped.

In [11]:
```
df['sqft_basement'].replace('?',value=0,inplace=True)
```

There was a placeholder for unknown values in this column. If there is no value for basement footage it will be assumed to not have a basement.

In [12]:
```
df['sqft_basement'] = pd.to_numeric(df['sqft_basement'],errors='coerce')
```

In [13]:
```
pd.to_numeric(df['yr_built'],errors='coerce')
```

Out[13]:
```
id
7129300520    1955
6414100192    1951
5631500400    1933
2487200875    1965
1954400510    1987
              ...
263000018     2009
6600060120    2014
1523300141    2009
291310100     2004
1523300157    2008
Name: yr_built, Length: 21597, dtype: int64
```

In [14]:
```
pd.to_numeric(df['yr_renovated'],errors='coerce')
```

Out[14]:
```
id
7129300520      0.00
```

```
6414100192   1991.00
5631500400      0.00
2487200875      0.00
1954400510      0.00
                ...
263000018       0.00
6600060120      0.00
1523300141      0.00
291310100       0.00
1523300157      0.00
Name: yr_renovated, Length: 21597, dtype: float64
```

Changing these columns from to a numeric data types.

In [15]:
```
df['zipcode'].astype(str)
```

Out[15]:
```
id
7129300520    98178
6414100192    98125
5631500400    98028
2487200875    98136
1954400510    98074
              ...
263000018     98103
6600060120    98146
1523300141    98144
291310100     98027
1523300157    98144
Name: zipcode, Length: 21597, dtype: object
```

Since *zipcode* is categorical it will be transformed to a string.

Data Preparation

Here we will do some light feature engineering and prepare the data for initial modeling. This propecess will be iterated upon as the model is fine tuned.

In [16]:
```
df_cols = list(df.columns)
df_cols
```

Out[16]:
```
['price',
 'bedrooms',
 'bathrooms',
 'sqft_living',
 'sqft_lot',
 'floors',
 'waterfront',
 'condition',
 'grade',
 'sqft_above',
 'sqft_basement',
 'yr_built',
 'yr_renovated',
 'zipcode',
 'lat',
 'long',
 'sqft_living15',
 'sqft_lot15']
```

In [17]:
```
df['renovated'] = np.where(df['yr_renovated'] != 0, 1,0)
```

Creating a categorical variable for wheter a property has been renovated or not as an alternative

to renovation year.

In [18]:
```
df['renovated'].value_counts()
```

Out[18]:
```
0    20853
1      744
Name: renovated, dtype: int64
```

Only a small portion of the dataset has been previously renovated.

In [20]:
```
df['basement'] = np.where(df['sqft_basement'] != 0, 1,0)
```

Same process here for basement.

In [21]:
```
df['age'] = 2021 - df['yr_built']
```

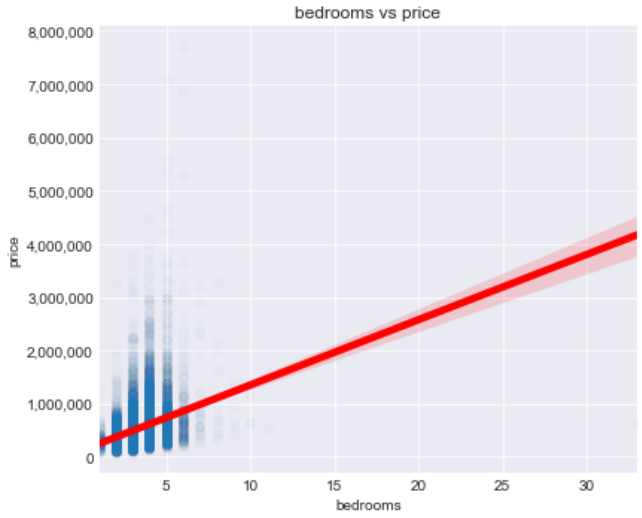
Using the year built feture to understand the age of the current age of the homes in the dataset.

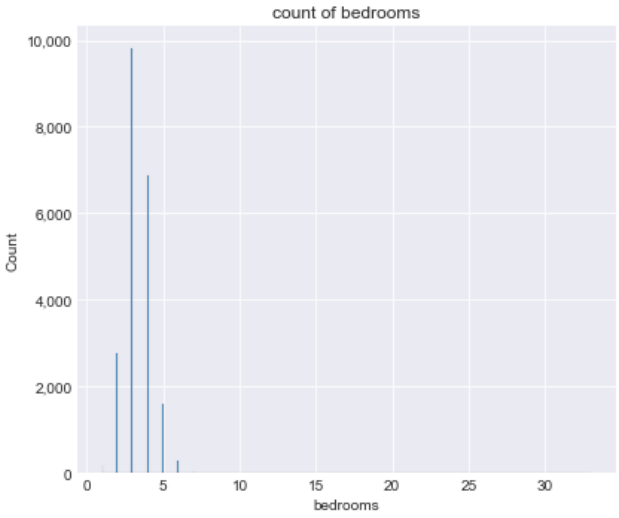
EXPLORE

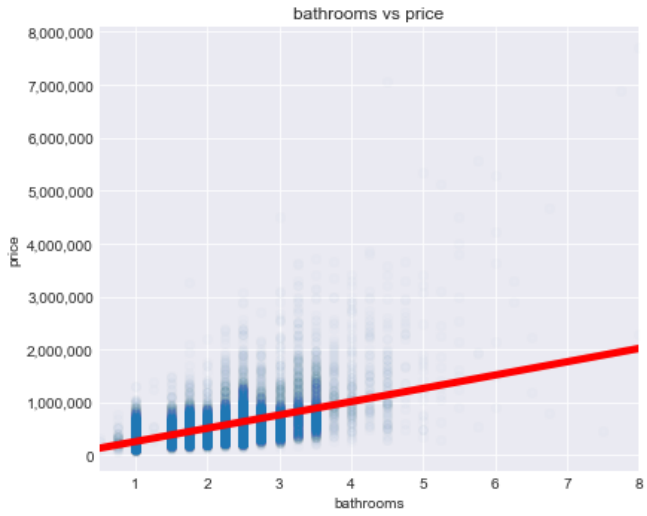
In [22]:
```
outcome = 'price'
x_cols = list(df.columns)
x_cols.remove('price')
```

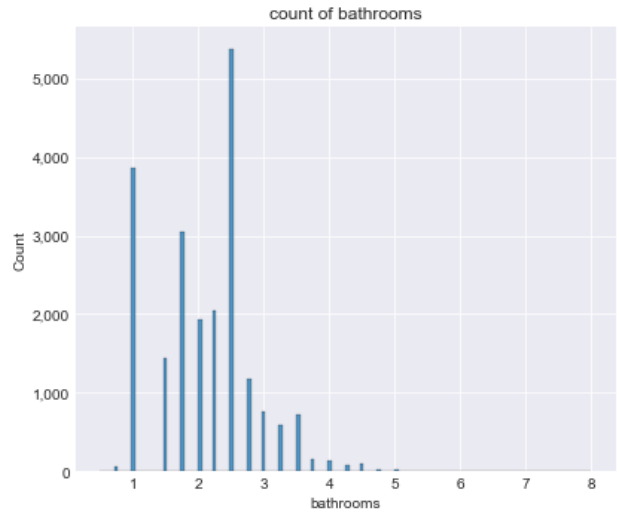
In [23]:
```
#Checking out features in the dataset against the target variable, Price

def plots(df, col, y='price'):

    fig, axes = plt.subplots(ncols=2,figsize=(12,5))

    sns.regplot(data=df,x=col, y=y,ax=axes[0],line_kws={'color':'red','lw':5},sc
    sns.histplot(data=df,x=col,discrete=False,ax=axes[1],kde=False)
    axes[0].set(title= col +' vs price', xlabel= col, ylabel='price')
    axes[1].set(title= 'count of ' + col, xlabel= col)
    axes[0].yaxis.set_major_formatter(FuncFormatter(lambda x, p: format(int(x),
    axes[1].yaxis.set_major_formatter(FuncFormatter(lambda x, p: format(int(x),
    plt.tight_layout();
```

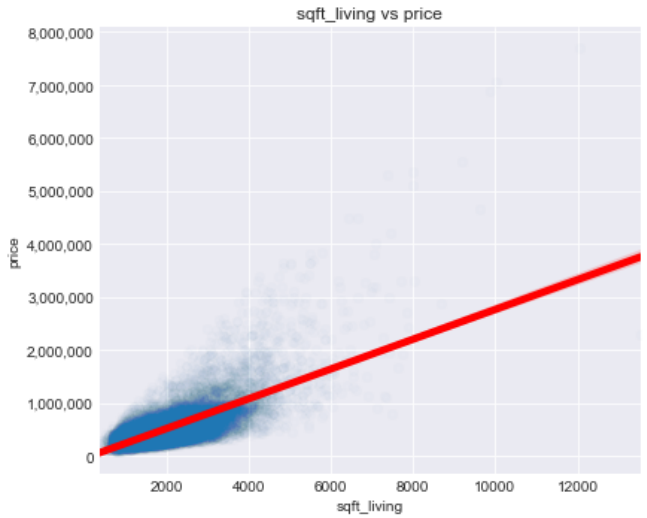
In [24]:
```
# Checking for Linearity and Distributions
for col in x_cols:
    plots(df, col, y='price')
```

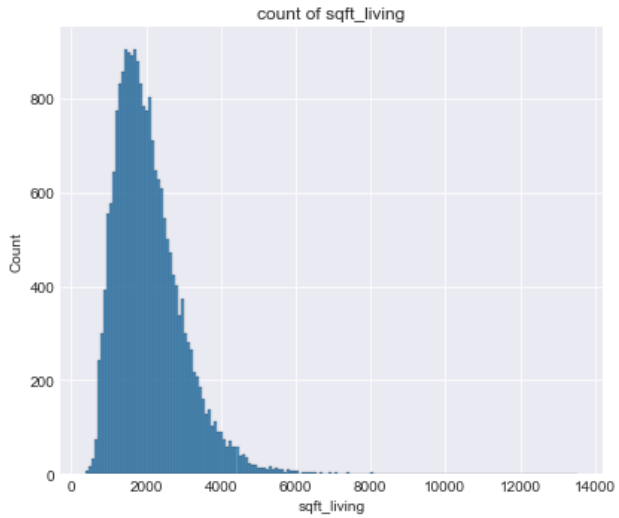

bedrooms vs price
price
bedrooms
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
5
10
15
20
25
30

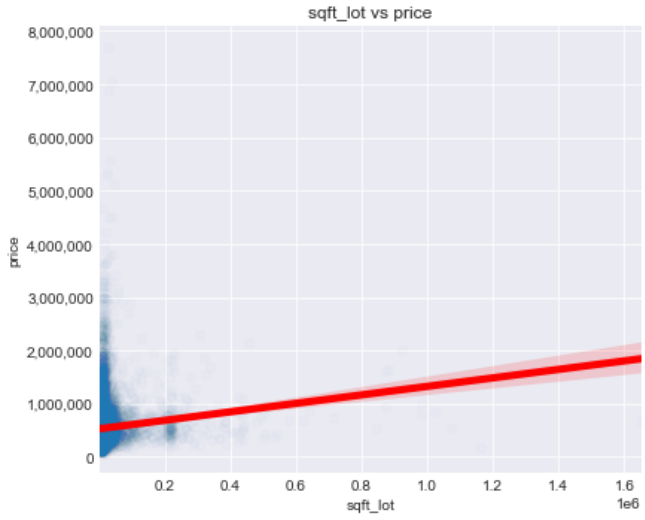

count of bedrooms
Count
bedrooms
10,000
8,000
6,000
4,000
2,000
0
0
5
10
15
20
25
30

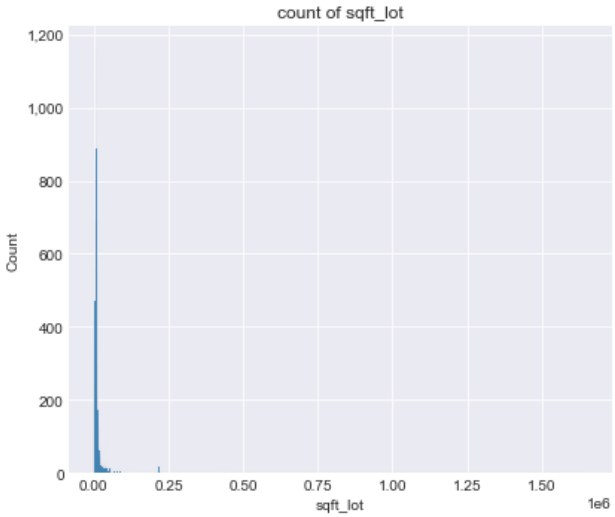

bathrooms vs price
price
bathrooms
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
1
2
3
4
5
6
7
8

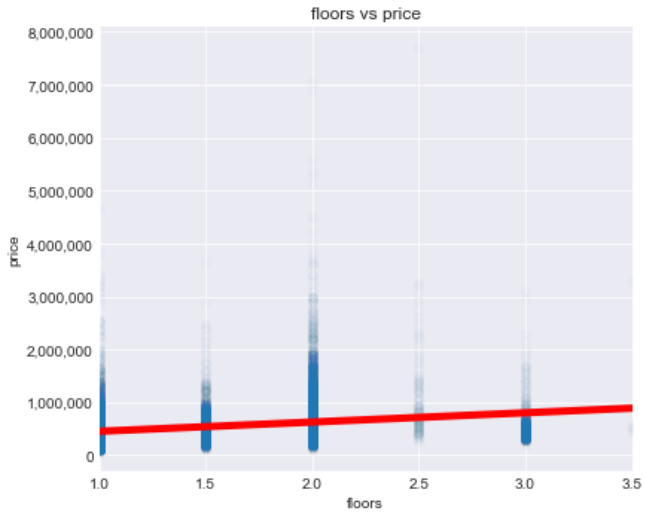

count of bathrooms
Count
bathrooms
5,000
4,000
3,000
2,000
1,000
0
1
2
3
4
5
6
7
8

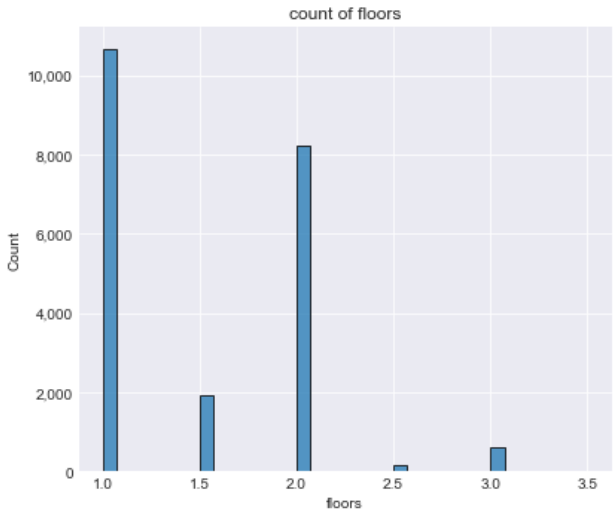

sqft_living vs price
price
sqft_living
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
2000
4000
6000
8000
10000
12000

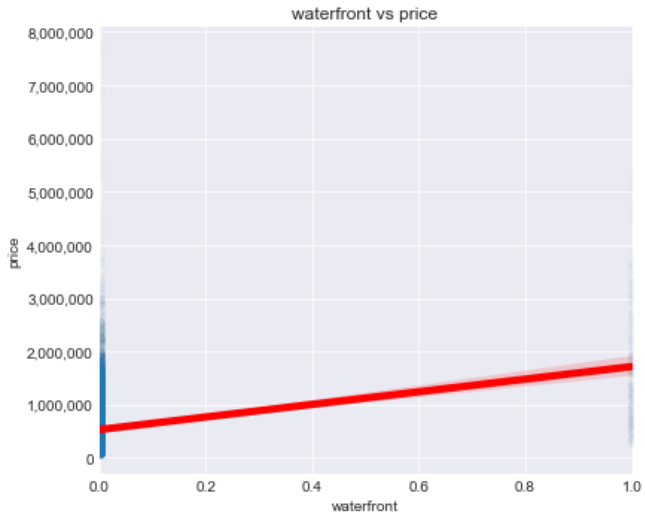

count of sqft_living
Count
sqft_living
800
600
400
200
0
0
2000
4000
6000
8000
10000
12000
14000

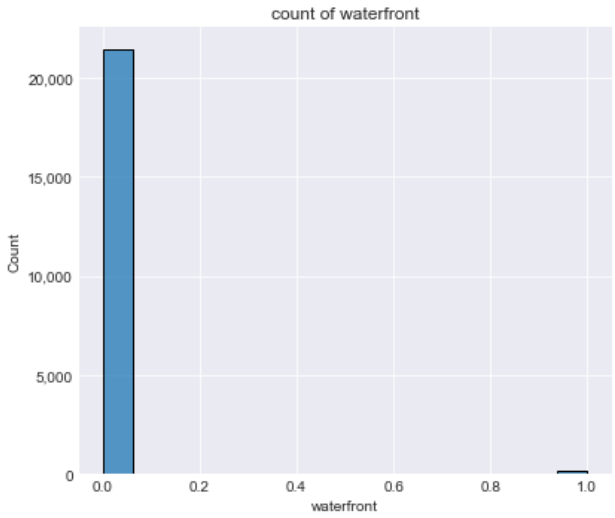

sqft_lot vs price
price
sqft_lot
1e6

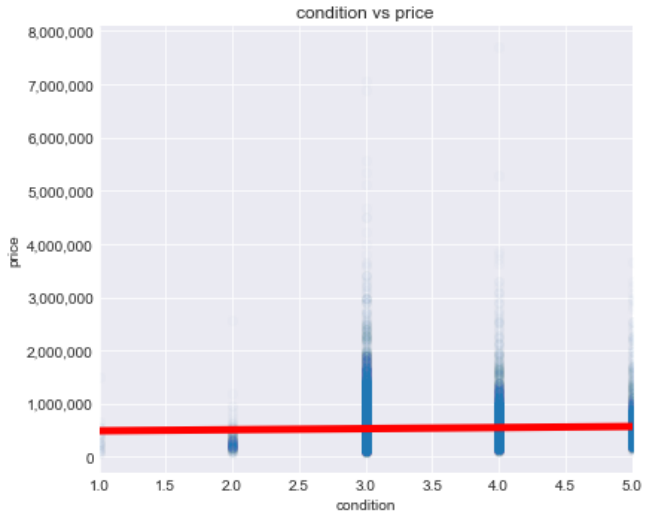

count of sqft_lot
Count
sqft_lot
1e6

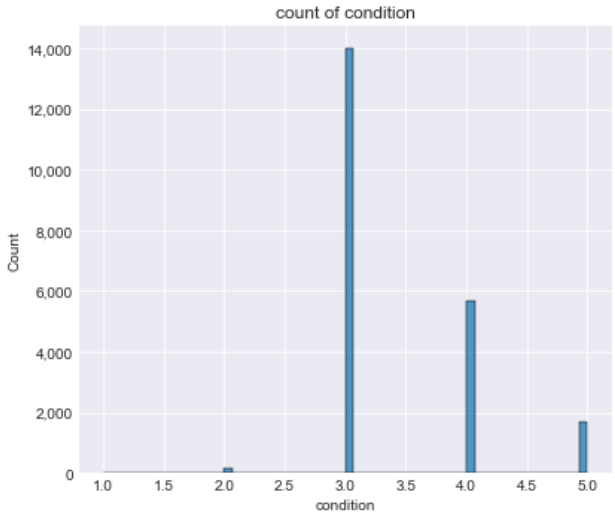

floors vs price
price
floors

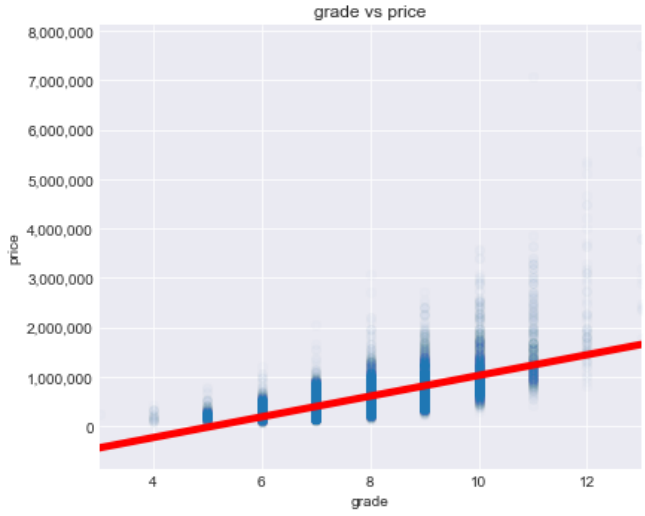

count of floors
Count
floors

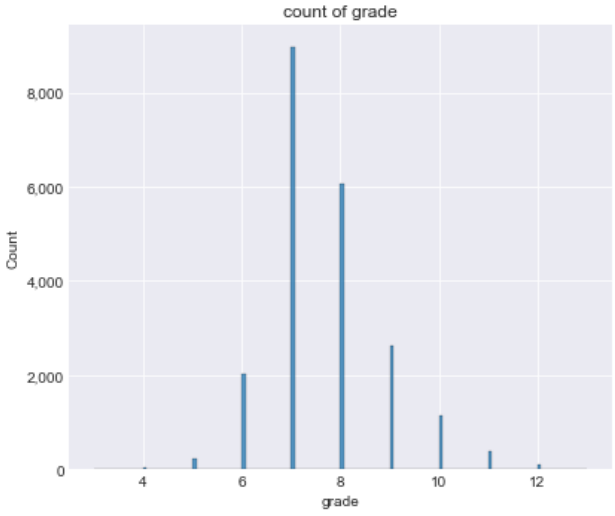

waterfront vs price
price
waterfront

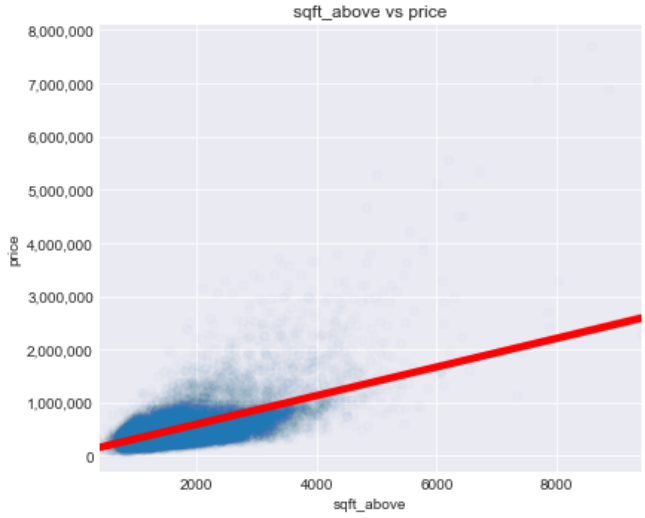

count of waterfront
Count
waterfront

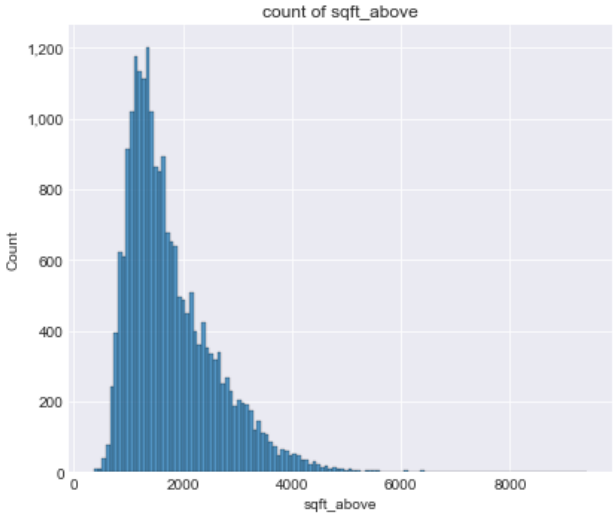

condition vs price
price
condition
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

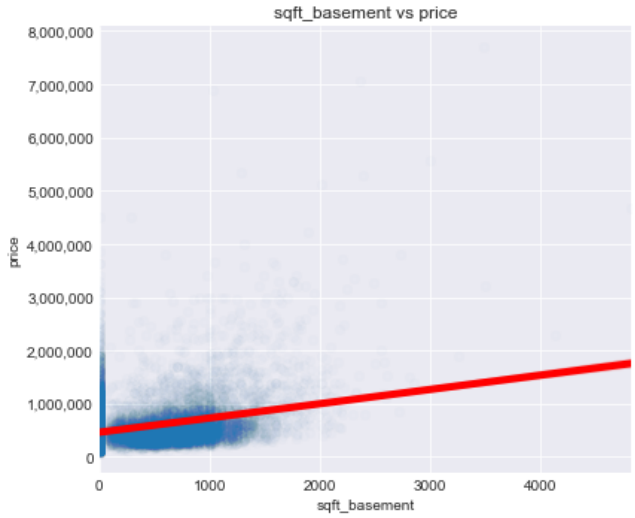

count of condition
Count
condition
14,000
12,000
10,000
8,000
6,000
4,000
2,000
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

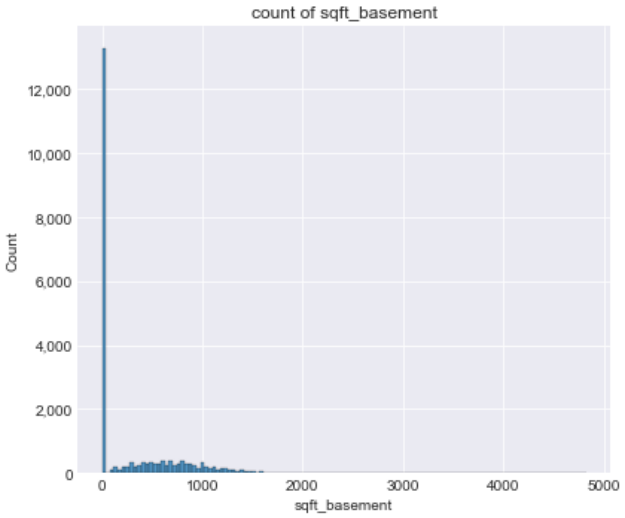

grade vs price
price
grade
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
4
6
8
10
12

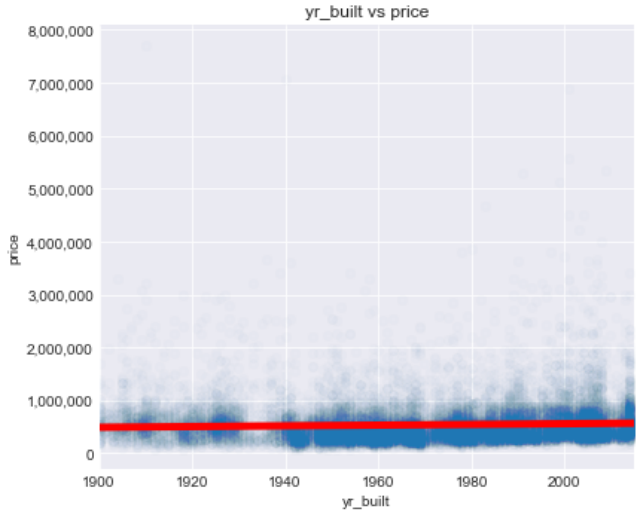

count of grade
Count
grade
8,000
6,000
4,000
2,000
0
4
6
8
10
12

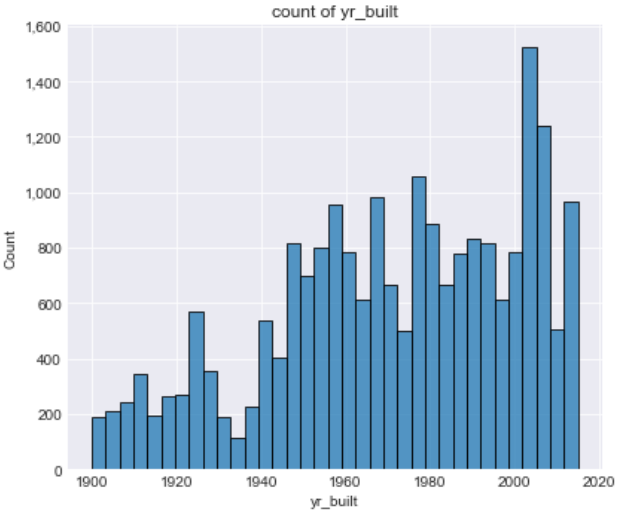

sqft_above vs price
price
sqft_above
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
2000
4000
6000
8000

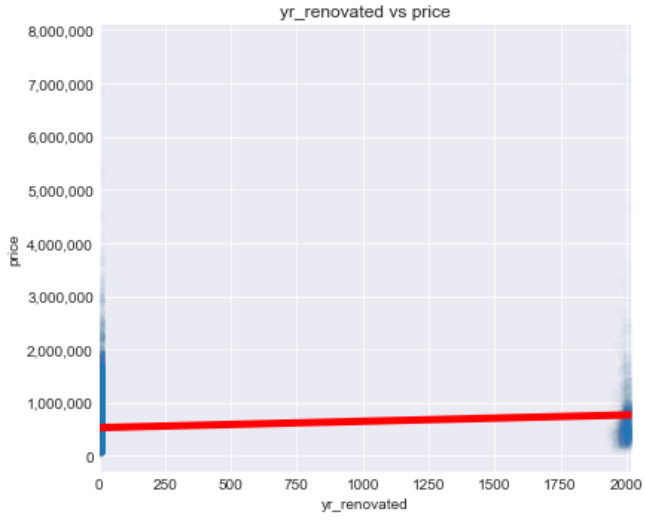

count of sqft_above
Count
sqft_above
1,200
1,000
800
600
400
200
0
0
2000
4000
6000
8000

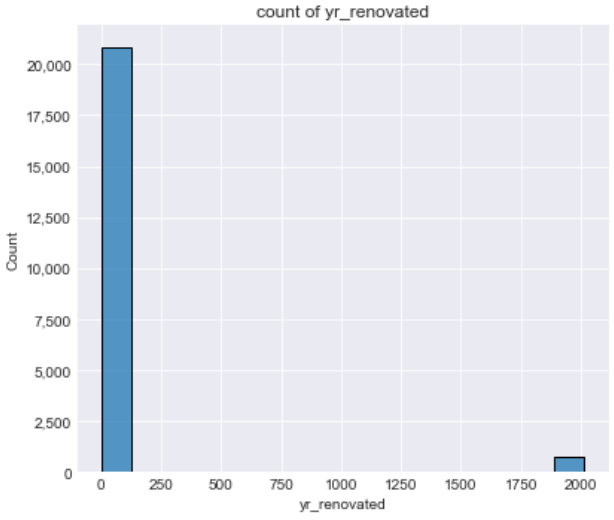

sqft_basement vs price
price
sqft_basement
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
0
1000
2000
3000
4000

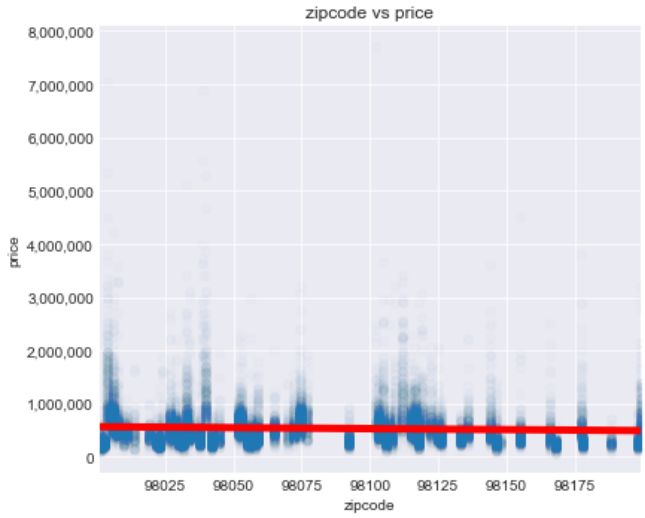

count of sqft_basement
Count
sqft_basement
12,000
10,000
8,000
6,000
4,000
2,000
0
0
1000
2000
3000
4000
5000

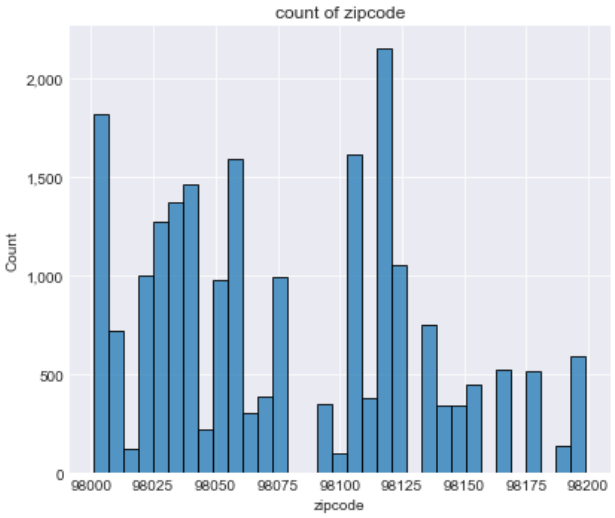

yr_built vs price
price
yr_built
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
1900
1920
1940
1960
1980
2000


count of yr_built
Count
yr_built
1,600
1,400
1,200
1,000
800
600
400
200
0
1900
1920
1940
1960
1980
2000
2020

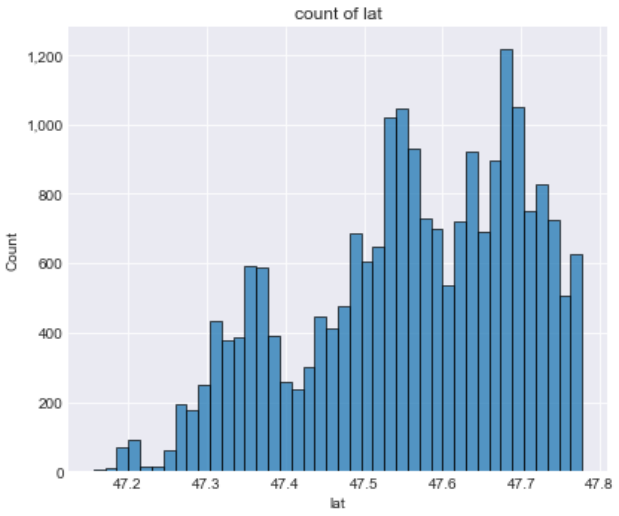

yr_renovated vs price
price
yr_renovated
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
0
250
500
750
1000
1250
1500
1750
2000

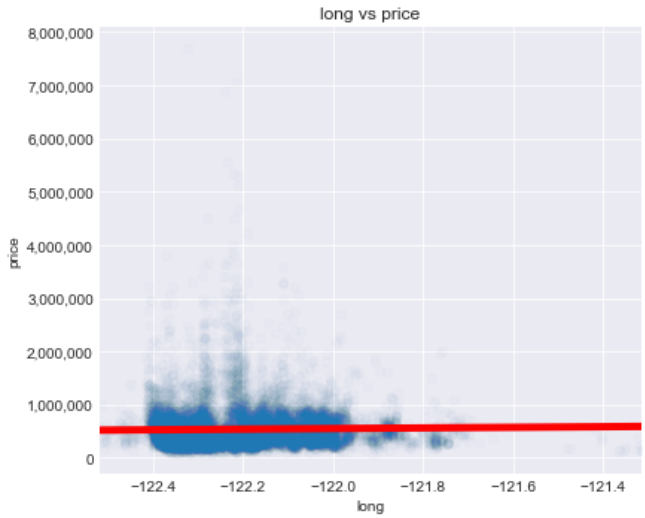

count of yr_renovated
Count
yr_renovated
20,000
17,500
15,000
12,500
10,000
7,500
5,000
2,500
0
0
250
500
750
1000
1250
1500
1750
2000

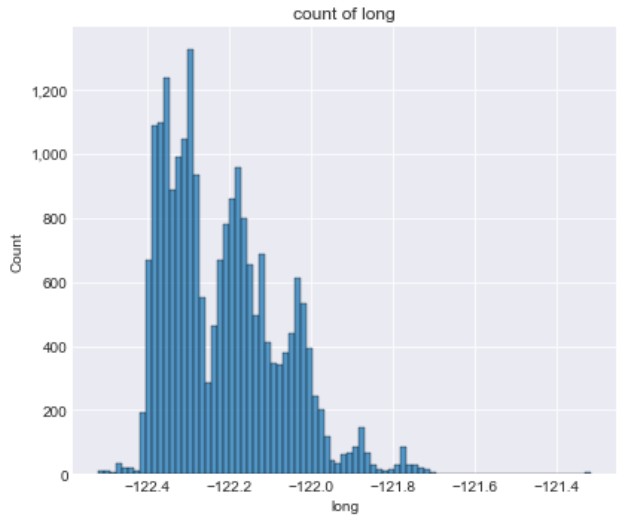

zipcode vs price
price
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
98025
98050
98075
98100
98125
98150
98175
zipcode

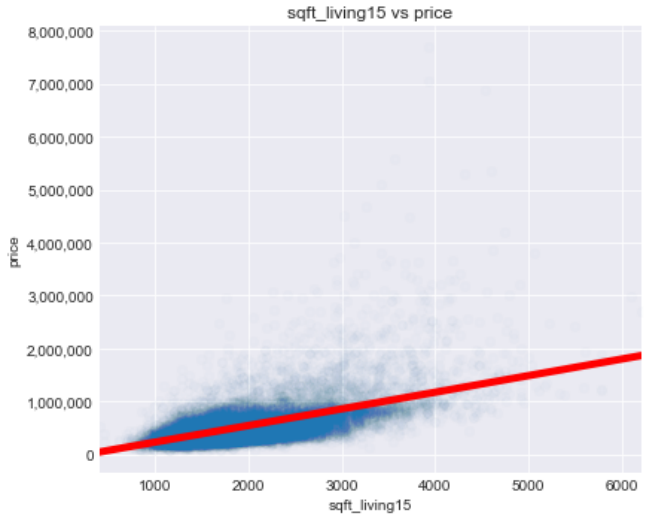

count of zipcode
Count
2,000
1,500
1,000
500
0
98000
98025
98050
98075
98100
98125
98150
98175
98200
zipcode

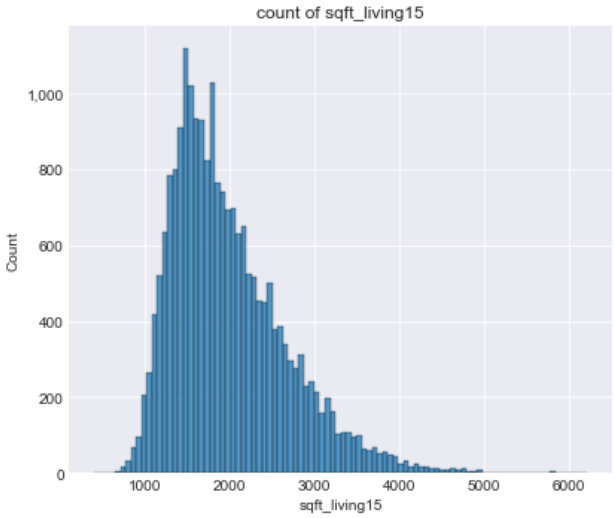

lat vs price
price
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
47.2
47.3
47.4
47.5
47.6
47.7
lat

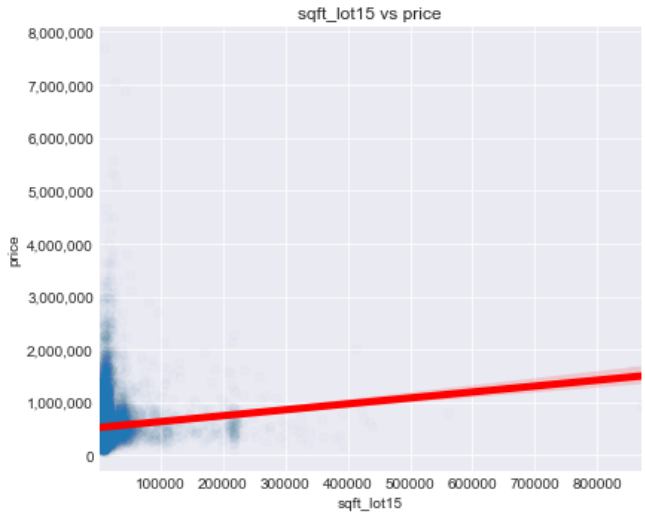

count of lat
Count
1,200
1,000
800
600
400
200
0
47.2
47.3
47.4
47.5
47.6
47.7
47.8
lat

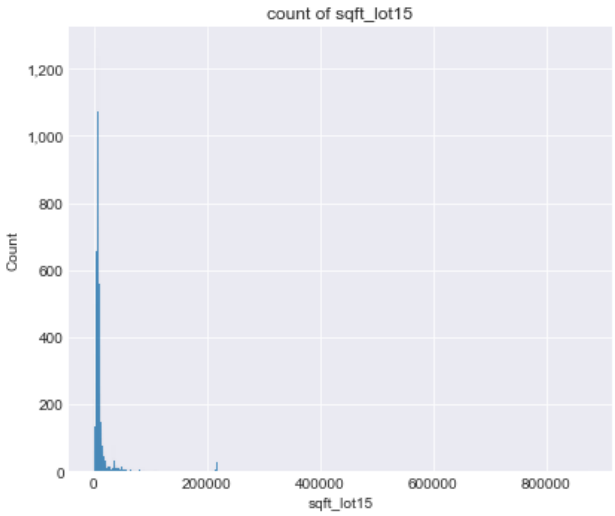

long vs price
price
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
−122.4
−122.2
−122.0
−121.8
−121.6
−121.4
long

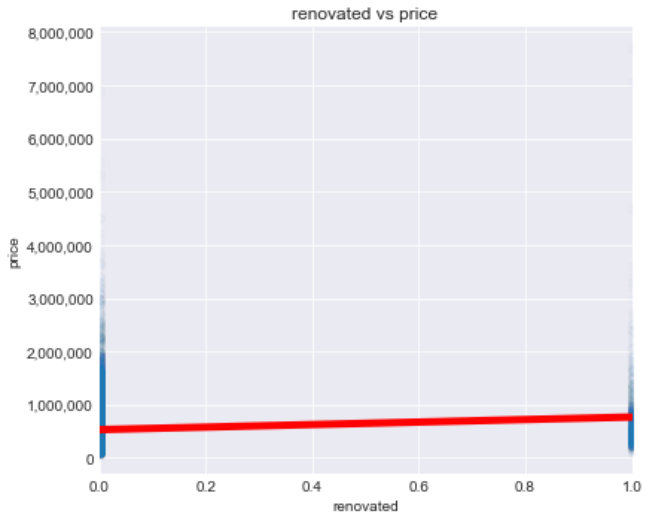

count of long
Count
1,200
1,000
800
600
400
200
0
−122.4
−122.2
−122.0
−121.8
−121.6
−121.4
long

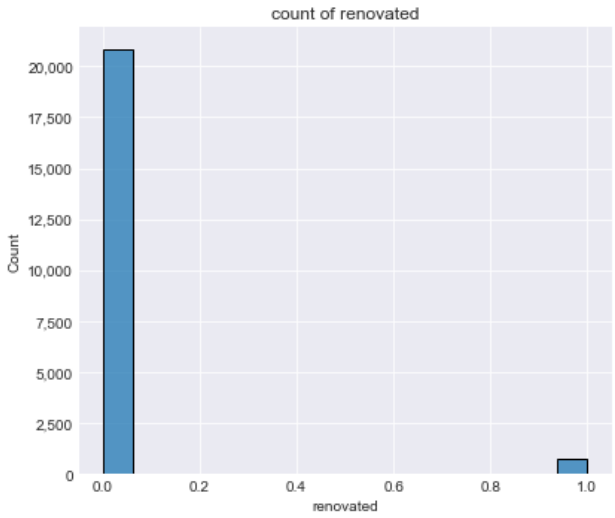

sqft_living15 vs price
price
sqft_living15
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
1000
2000
3000
4000
5000
6000

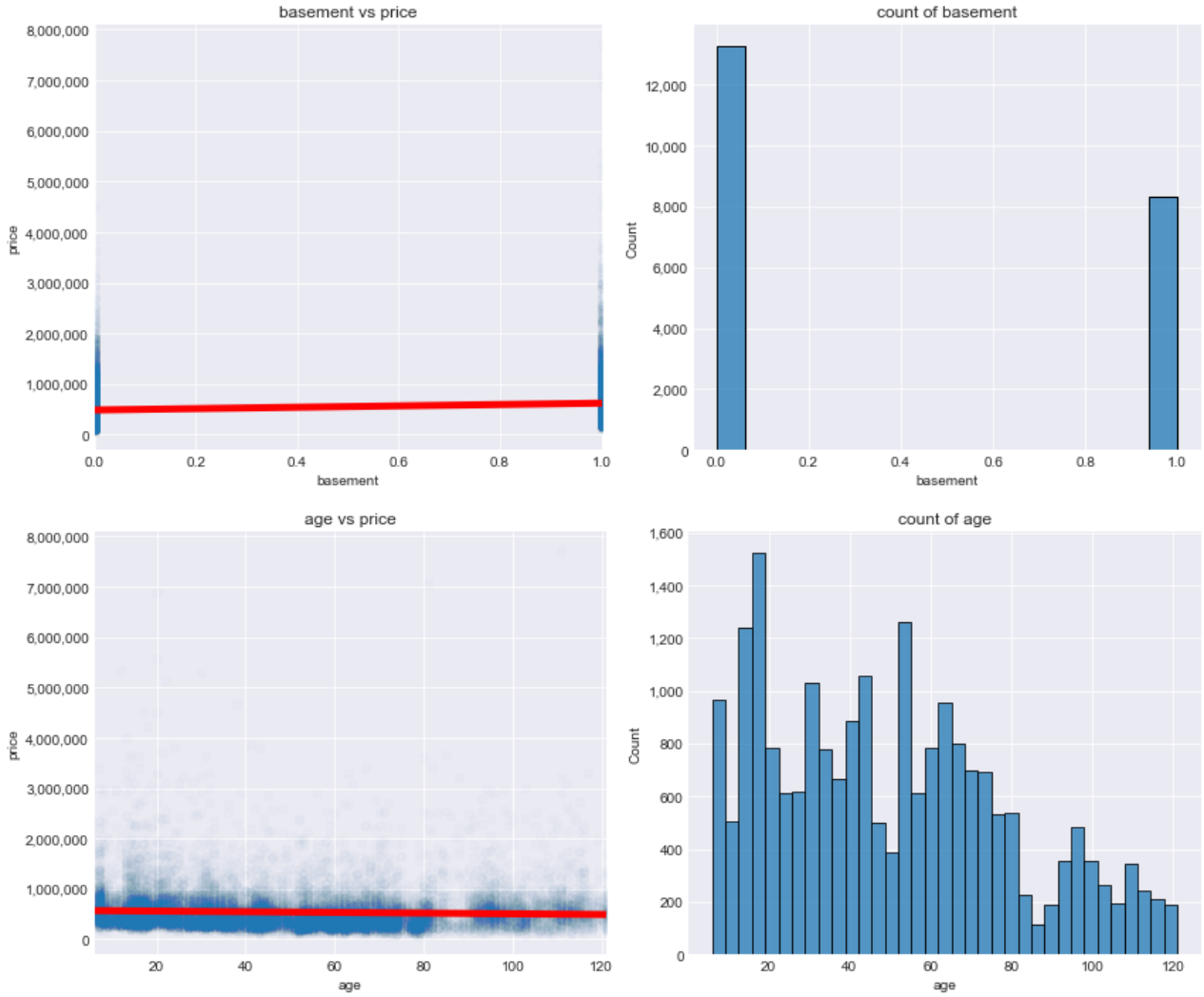

count of sqft_living15
Count
sqft_living15
1,000
800
600
400
200
0
1000
2000
3000
4000
5000
6000


sqft_lot15 vs price
price
sqft_lot15
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
100000
200000
300000
400000
500000
600000
700000
800000


count of sqft_lot15
Count
sqft_lot15
1,200
1,000
800
600
400
200
0
0
200000
400000
600000
800000


renovated vs price
price
renovated
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
0.0
0.2
0.4
0.6
0.8
1.0


count of renovated
Count
renovated
20,000
17,500
15,000
12,500
10,000
7,500
5,000
2,500
0
0.0
0.2
0.4
0.6
0.8
1.0



Upon investigating the linear relationships between the target and predictors, it is clear that some features are categorical (zipcode, condition, waterfront and the features we engineered earlier -- basement, renovated and age.) Other features such as bedrooms, bathrooms, grade and square footages have a positively correlated relationship with price. The others have a complex relationship that need to be further investigated to confirm whether they violate the linearity assumption? There also appears to quite outliers that must be removed. Only the square footage features seem to be somewhat normally distributed but still skew to the right.

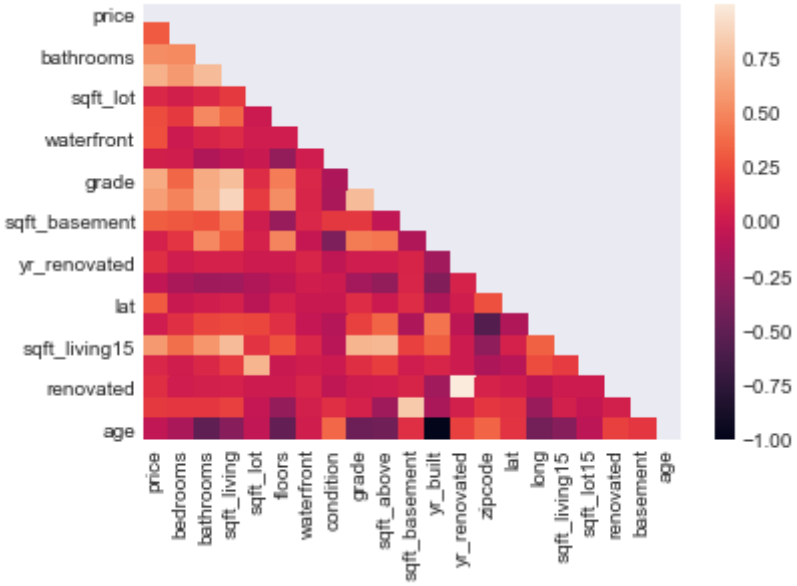
In [25]:
```
corr = df.corr()
corr
```

Out[25]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition
price	1.00	0.31	0.53	0.70	0.09	0.26	0.26	0.04
bedrooms	0.31	1.00	0.51	0.58	0.03	0.18	-0.00	0.03
bathrooms	0.53	0.51	1.00	0.76	0.09	0.50	0.06	-0.13
sqft_living	0.70	0.58	0.76	1.00	0.17	0.35	0.10	-0.06
sqft_lot	0.09	0.03	0.09	0.17	1.00	-0.00	0.02	-0.01
floors	0.26	0.18	0.50	0.35	-0.00	1.00	0.02	-0.26
waterfront	0.26	-0.00	0.06	0.10	0.02	0.02	1.00	0.02

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition
condition	0.04	0.03	-0.13	-0.06	-0.01	-0.26	0.02	1.00
grade	0.67	0.36	0.67	0.76	0.11	0.46	0.08	-0.15
sqft_above	0.61	0.48	0.69	0.88	0.18	0.52	0.07	-0.16
sqft_basement	0.32	0.30	0.28	0.43	0.02	-0.24	0.08	0.17
yr_built	0.05	0.16	0.51	0.32	0.05	0.49	-0.02	-0.36
yr_renovated	0.12	0.02	0.05	0.05	0.00	0.00	0.07	-0.06
zipcode	-0.05	-0.15	-0.20	-0.20	-0.13	-0.06	0.03	0.00
lat	0.31	-0.01	0.02	0.05	-0.09	0.05	-0.01	-0.02
long	0.02	0.13	0.22	0.24	0.23	0.13	-0.04	-0.11
sqft_living15	0.59	0.39	0.57	0.76	0.14	0.28	0.08	-0.09
sqft_lot15	0.08	0.03	0.09	0.18	0.72	-0.01	0.03	-0.00
renovated	0.12	0.02	0.05	0.05	0.01	0.00	0.07	-0.06
basement	0.18	0.16	0.16	0.20	-0.03	-0.25	0.04	0.13
age	-0.05	-0.16	-0.51	-0.32	-0.05	-0.49	0.02	0.36

In [26]:
```
sns.heatmap(corr, mask=np.triu(np.ones_like(corr, dtype=bool)));
```



In [27]:
```
price_corrs = df.corr()['price'].map(abs).sort_values(ascending=False)
price_corrs
```

Out[27]:
```
price            1.00
sqft_living      0.70
grade            0.67
sqft_above       0.61
sqft_living15    0.59
bathrooms        0.53
sqft_basement    0.32
bedrooms         0.31
lat              0.31
waterfront       0.26
```

```
floors          0.26
basement        0.18
yr_renovated    0.12
renovated       0.12
sqft_lot        0.09
sqft_lot15      0.08
age             0.05
yr_built        0.05
zipcode         0.05
condition       0.04
long            0.02
Name: price, dtype: float64
```

Heatmap and correlation table confirms initial views from the regression plots above. Square footage, grade, bathrooms and bedrooms have the highest correlation with the dependent variable.

MODEL

Fit Initial Model

In [28]:
```
# Writing a function to re-use for fitting and iterating models
outcome='price'
cat_cols =[]
def make_model(df,outcome='price', cat_cols=[]):
    x_cols = list(df.columns)
    x_cols.remove(outcome)


    predictors = '+'.join(x_cols)
    formula = outcome + '~' + predictors
    formula

    for col in cat_cols:
        formula = formula.replace(col,f'C({col})')

    model = smf.ols(formula=formula, data=df).fit()
    display(model.summary2())
    return model
```

In [29]:
```
# A function that produces a QQ-plot and Regression plot to evaluate model
def evaluation(model, df, y='price'):

    fig, axes = plt.subplots(ncols=2,figsize=(12,5))

    sm.graphics.qqplot(model.resid, dist=stats.norm, line='45', ax=axes[0], fit=
    sns.regplot(model.predict(df), model.resid,ax=axes[1])
    plt.tight_layout();
```

Model 1

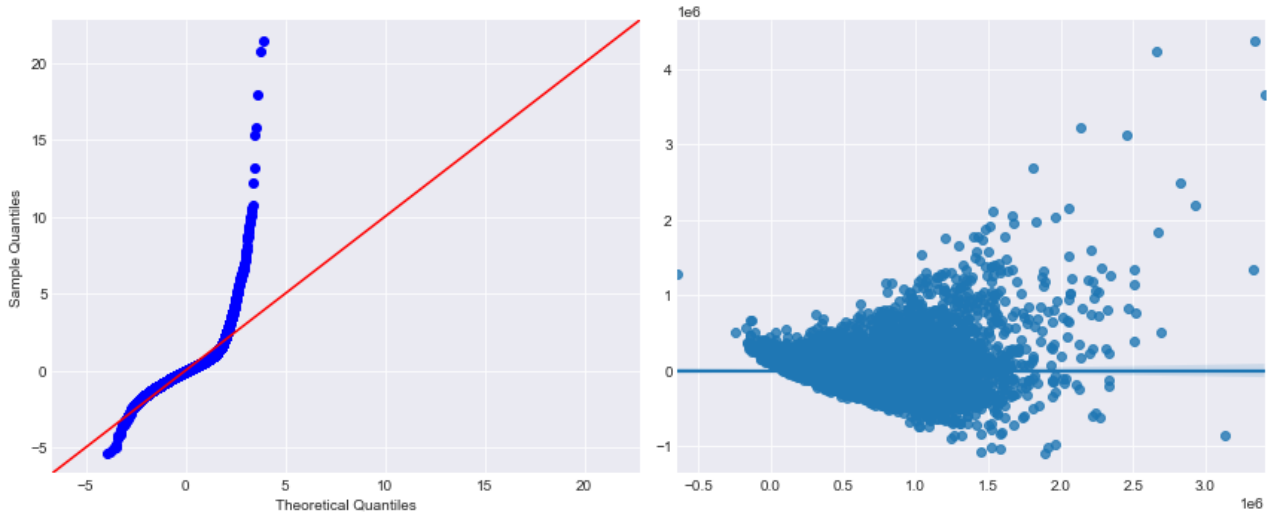
In [32]:
```
# Fitting the first model from the baseline dataset
model1 = make_model(df)
evaluation(model1, df)
```

Model:	OLS	Adj. R-squared:	0.692

Dependent Variable:	price	AIC:	589383.0327
Date:	2021-11-04 12:20	BIC:	589542.6389
No. Observations:	21597	Log-Likelihood:	-2.9467e+05
Df Model:	19	F-statistic:	2552.
Df Residuals:	21577	Prob (F-statistic):	0.00
R-squared:	0.692	Scale:	4.1595e+10

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
Intercept	-2.2355	1.4408	-1.5516	0.1208	-5.0597	0.5886
bedrooms	-39778.2166	1920.5593	-20.7118	0.0000	-43542.6548	-36013.7785
bathrooms	43599.3677	3334.3457	13.0758	0.0000	37063.8036	50134.9319
sqft_living	115.8913	18.3419	6.3184	0.0000	79.9398	151.8427
sqft_lot	0.1728	0.0485	3.5601	0.0004	0.0777	0.2680
floors	9535.7170	3649.3607	2.6130	0.0090	2382.7002	16688.7339
waterfront	782630.9328	17220.6172	45.4473	0.0000	748877.2498	816384.6157
condition	28308.0964	2383.2140	11.8781	0.0000	23636.8207	32979.3721
grade	101076.8431	2191.0299	46.1321	0.0000	96782.2625	105371.4237
sqft_above	62.1337	18.2972	3.3958	0.0007	26.2699	97.9976
sqft_basement	60.6977	18.9781	3.1983	0.0014	23.4993	97.8962
yr_built	-3657.1531	1444.1394	-2.5324	0.0113	-6487.7732	-826.5330
yr_renovated	3282.5333	482.7420	6.7998	0.0000	2336.3233	4228.7433
zipcode	-506.4916	33.3341	-15.1944	0.0000	-571.8290	-441.1543
lat	577280.3782	10844.5887	53.2321	0.0000	556024.1825	598536.5739
long	-235060.5549	13351.0540	-17.6061	0.0000	-261229.6079	-208891.5019
sqft_living15	36.0459	3.4536	10.4371	0.0000	29.2765	42.8153
sqft_lot15	-0.3797	0.0743	-5.1087	0.0000	-0.5254	-0.2340
renovated	-6500001.8055	963504.5879	-6.7462	0.0000	-8388542.0345	-4611461.5765
basement	-12555.0869	5368.7615	-2.3385	0.0194	-23078.2565	-2031.9173
age	-860.8773	1468.5680	-0.5862	0.5577	-3739.3792	2017.6245

Omnibus:	18149.321	Durbin-Watson:	1.986
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1708549.923
Skew:	3.520	Prob(JB):	0.000
Kurtosis:	46.001	Condition No.:	897063612119817650176



The first model has an .69 R-squared. Not bad but could be improved. QQ Plot shows the residuals are not normally distributed and violated the assumptions of heteroscedasticity.

Model 2

OneHotEncoding Categorical Variables

In [33]:
```
cat_cols = ['zipcode','condition']
```

Pulling out the predictors identified above as categorical. The other categorical features are already binary or ordinal.

In [34]:
```
encoder = OneHotEncoder(sparse=False, drop='first')
encoder
```

Out[34]:
```
OneHotEncoder(drop='first', sparse=False)
```

In [35]:
```
data_ohe = encoder.fit_transform(df[cat_cols])
df_ohe = pd.DataFrame(data_ohe,columns=encoder.get_feature_names(cat_cols),
                      index=df.index)
df_ohe
```

Out[35]:

id	zipcode_98002	zipcode_98003	zipcode_98004	zipcode_98005	zipcode_98006
7129300520	0.00	0.00	0.00	0.00	0.00
6414100192	0.00	0.00	0.00	0.00	0.00
5631500400	0.00	0.00	0.00	0.00	0.00
2487200875	0.00	0.00	0.00	0.00	0.00
1954400510	0.00	0.00	0.00	0.00	0.00
...	...	...	...	...	...
263000018	0.00	0.00	0.00	0.00	0.00
6600060120	0.00	0.00	0.00	0.00	0.00

id	zipcode_98002	zipcode_98003	zipcode_98004	zipcode_98005	zipcode_98006
1523300141	0.00	0.00	0.00	0.00	0.00
291310100	0.00	0.00	0.00	0.00	0.00
1523300157	0.00	0.00	0.00	0.00	0.00

21597 rows × 73 columns

In [36]:
```
df_model2 = pd.concat([df.copy(),df_ohe], axis=1,join='inner')
df_model2.drop(columns=['condition','zipcode'],inplace=True)
df_model2
```

Out[36]:

id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	grade	s
7129300520	221900.00	3	1.00	1180	5650	1.00	0.00	7	
6414100192	538000.00	3	2.25	2570	7242	2.00	0.00	7	
5631500400	180000.00	2	1.00	770	10000	1.00	0.00	6	
2487200875	604000.00	4	3.00	1960	5000	1.00	0.00	7	
1954400510	510000.00	3	2.00	1680	8080	1.00	0.00	8	
...	...	...	...	...	...	...	...	...	...
263000018	360000.00	3	2.50	1530	1131	3.00	0.00	8	
6600060120	400000.00	4	2.50	2310	5813	2.00	0.00	8	
1523300141	402101.00	2	0.75	1020	1350	2.00	0.00	7	
291310100	400000.00	3	2.50	1600	2388	2.00	0.00	8	
1523300157	325000.00	2	0.75	1020	1076	2.00	0.00	7	

21597 rows × 92 columns

In [37]:
```
model2 = make_model(df_model2)
evaluation(model2, df_model2)
```

Model:	OLS	Adj. R-squared:	0.799
Dependent Variable:	price	AIC:	580256.9951
Date:	2021-11-04 12:32	BIC:	580983.2033
No. Observations:	21597	Log-Likelihood:	-2.9004e+05
Df Model:	90	F-statistic:	952.9
Df Residuals:	21506	Prob (F-statistic):	0.00
R-squared:	0.800	Scale:	2.7171e+10

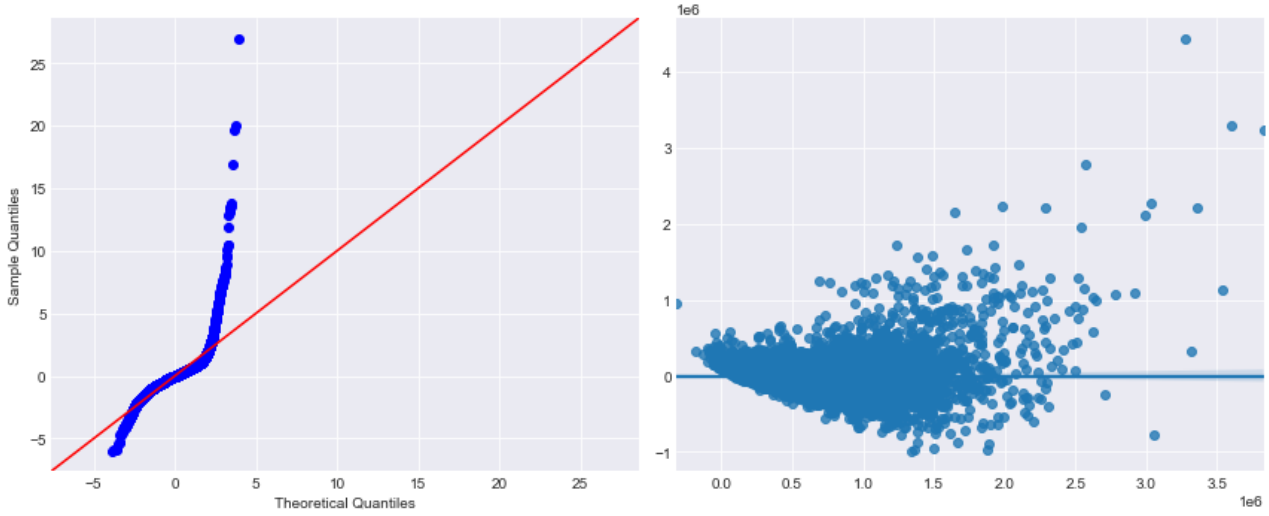
	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
Intercept	-12.6937	3.0846	-4.1152	0.0000	-18.7398	-6.6476

bedrooms	-30634.7015	1574.1831	-19.4607	0.0000	-33720.2173	-27549.1856
bathrooms	27037.0695	2721.3822	9.9351	0.0000	21702.9581	32371.1808
sqft_living	108.0823	14.8649	7.2710	0.0000	78.9460	137.2185
sqft_lot	0.2771	0.0394	7.0275	0.0000	0.1998	0.3543
floors	-43029.5527	3251.7709	-13.2327	0.0000	-49403.2653	-36655.8402
waterfront	851457.4647	14181.4144	60.0404	0.0000	823660.8388	879254.0906
grade	63550.4818	1870.1570	33.9814	0.0000	59884.8351	67216.1285
sqft_above	94.2188	14.8612	6.3399	0.0000	65.0897	123.3478
sqft_basement	64.3772	15.3859	4.1842	0.0000	34.2197	94.5347
yr_built	-13267.5439	3116.1166	-4.2577	0.0000	-19375.3640	-7159.7238
yr_renovated	2649.1114	391.4468	6.7675	0.0000	1881.8466	3416.3763
lat	160775.2945	65069.6262	2.4708	0.0135	33233.9927	288316.5963
long	-152265.8787	46727.9087	-3.2586	0.0011	-243856.0515	-60675.7059
sqft_living15	27.7759	2.9215	9.5075	0.0000	22.0496	33.5021
sqft_lot15	-0.1385	0.0621	-2.2315	0.0257	-0.2602	-0.0168
renovated	-5245969.8264	781288.5930	-6.7145	0.0000	-6777353.5168	-3714586.1359
basement	-32849.3685	4399.4964	-7.4666	0.0000	-41472.7084	-24226.0287
age	-12386.4442	3118.2607	-3.9722	0.0001	-18498.4669	-6274.4214
zipcode_98002	39255.6774	14846.4809	2.6441	0.0082	10155.4717	68355.8831
zipcode_98003	-19398.8109	13273.0430	-1.4615	0.1439	-45414.9613	6617.3394
zipcode_98004	719660.6734	24130.2164	29.8240	0.0000	672363.6565	766957.6904
zipcode_98005	242853.1568	25790.3167	9.4164	0.0000	192302.2199	293404.0937
zipcode_98006	241464.0648	21085.4777	11.4517	0.0000	200134.9619	282793.1678
zipcode_98007	199515.6053	26616.0840	7.4961	0.0000	147346.1031	251685.1075
zipcode_98008	229420.9099	25269.2326	9.0791	0.0000	179891.3367	278950.4832
zipcode_98010	103576.0282	22636.2497	4.5757	0.0000	59207.2969	147944.7595
zipcode_98011	49643.3258	32872.4026	1.5102	0.1310	-14789.0258	114075.6773
zipcode_98014	104667.7570	36108.0701	2.8987	0.0038	33893.2568	175442.2572
zipcode_98019	60501.8499	35612.1058	1.6989	0.0894	-9300.5234	130304.2232
zipcode_98022	69128.8434	19646.3008	3.5187	0.0004	30620.6342	107637.0526
zipcode_98023	-50197.8174	12213.5924	-4.1100	0.0000	-74137.3658	-26258.2689
zipcode_98024	167934.1676	31777.5423	5.2847	0.0000	105647.8238	230220.5115
zipcode_98027	156289.8123	21641.9102	7.2216	0.0000	113870.0604	198709.5642
zipcode_98028	47413.8340	31928.9475	1.4850	0.1376	-15169.2753	109996.9434
zipcode_98029	198964.7664	24714.5927	8.0505	0.0000	150522.3284	247407.2045
zipcode_98030	3783.5412	14594.4773	0.2592	0.7954	-24822.7187	32389.8011

zipcode_98031	5421.4135	15206.4248	0.3565	0.7215	-24384.3089	35227.1360
zipcode_98032	-5300.4941	17646.6272	-0.3004	0.7639	-39889.1945	29288.2064
zipcode_98033	314509.8177	27391.4717	11.4820	0.0000	260820.4980	368199.1374
zipcode_98034	146897.1208	29365.0268	5.0025	0.0000	89339.4865	204454.7552
zipcode_98038	49304.4939	16388.2659	3.0085	0.0026	17182.2752	81426.7126
zipcode_98039	1248012.1174	32598.6446	38.2842	0.0000	1184116.3521	1311907.8828
zipcode_98040	472202.3528	21342.1407	22.1254	0.0000	430370.1713	514034.5343
zipcode_98042	13036.5319	13969.0808	0.9332	0.3507	-14343.9043	40416.9681
zipcode_98045	145874.5892	30266.2554	4.8197	0.0000	86550.4799	205198.6984
zipcode_98052	174638.5612	27967.3038	6.2444	0.0000	119820.5679	229456.5546
zipcode_98053	151985.1455	29963.6044	5.0723	0.0000	93254.2547	210716.0364
zipcode_98055	32443.8026	16926.9954	1.9167	0.0553	-734.3661	65621.9713
zipcode_98056	76037.2854	18388.9564	4.1349	0.0000	39993.5647	112081.0062
zipcode_98058	16371.7049	15996.1101	1.0235	0.3061	-14981.8595	47725.2692
zipcode_98059	60709.6467	18041.9015	3.3649	0.0008	25346.1792	96073.1141
zipcode_98065	108578.1106	27908.6848	3.8905	0.0001	53875.0147	163281.2064
zipcode_98070	-31306.8359	21266.1274	-1.4721	0.1410	-72990.0257	10376.3539
zipcode_98072	85491.6807	32706.7930	2.6139	0.0090	21383.9364	149599.4249
zipcode_98074	136260.0147	26476.6135	5.1464	0.0000	84363.8850	188156.1444
zipcode_98075	141085.0604	25460.5593	5.5413	0.0000	91180.4724	190989.6484
zipcode_98077	49077.9285	34027.3947	1.4423	0.1492	-17618.2933	115774.1502
zipcode_98092	-25910.9626	13272.6351	-1.9522	0.0509	-51926.3136	104.3884
zipcode_98102	448578.1723	28283.5825	15.8600	0.0000	393140.2491	504016.0955
zipcode_98103	274174.1781	26482.4310	10.3531	0.0000	222266.6458	326081.7105
zipcode_98105	406595.6476	27203.4253	14.9465	0.0000	353274.9130	459916.3823
zipcode_98106	109926.8385	19624.8981	5.6014	0.0000	71460.5801	148393.0968
zipcode_98107	282506.2532	27305.8386	10.3460	0.0000	228984.7808	336027.7257
zipcode_98108	85552.5194	21665.0557	3.9489	0.0001	43087.4006	128017.6382
zipcode_98109	439444.9091	28135.8379	15.6187	0.0000	384296.5764	494593.2418
zipcode_98112	547551.9698	24973.8058	21.9251	0.0000	498601.4550	596502.4847
zipcode_98115	270297.2970	26925.2712	10.0388	0.0000	217521.7650	323072.8290
zipcode_98116	256252.0133	21903.2542	11.6993	0.0000	213320.0078	299184.0189
zipcode_98117	247450.8687	27265.9692	9.0754	0.0000	194007.5432	300894.1942
zipcode_98118	143711.7417	19134.7052	7.5105	0.0000	106206.2979	181217.1855
zipcode_98119	428722.3188	26580.6007	16.1291	0.0000	376622.3665	480822.2711
zipcode_98122	288024.6356	23714.8177	12.1453	0.0000	241541.8311	334507.4402

zipcode_98125	140241.2438	29071.1978	4.8241	0.0000	83259.5362	197222.9514
zipcode_98126	162455.7351	20112.9889	8.0772	0.0000	123032.7825	201878.6878
zipcode_98133	88697.3158	30022.0035	2.9544	0.0031	29851.9584	147542.6731
zipcode_98136	219925.3757	20620.0377	10.6656	0.0000	179508.5699	260342.1816
zipcode_98144	244442.8302	22041.9041	11.0899	0.0000	201239.0604	287646.6000
zipcode_98146	89290.7557	18401.7001	4.8523	0.0000	53222.0563	125359.4551
zipcode_98148	40202.3925	25061.1100	1.6042	0.1087	-8919.2452	89324.0302
zipcode_98155	72764.7793	31213.4297	2.3312	0.0198	11584.1379	133945.4207
zipcode_98166	37497.0786	16850.1941	2.2253	0.0261	4469.4463	70524.7109
zipcode_98168	47153.1604	17806.8539	2.6480	0.0081	12250.4037	82055.9171
zipcode_98177	158217.3420	31322.4199	5.0512	0.0000	96823.0717	219611.6122
zipcode_98178	30499.4616	18381.5321	1.6592	0.0971	-5529.7070	66528.6301
zipcode_98188	19055.4721	18878.7554	1.0094	0.3128	-17948.2911	56059.2353
zipcode_98198	4736.5979	14288.6965	0.3315	0.7403	-23270.3089	32743.5047
zipcode_98199	326245.7442	25890.3440	12.6011	0.0000	275498.7464	376992.7420
condition_2	61693.3313	33247.3490	1.8556	0.0635	-3473.9429	126860.6056
condition_3	49007.7395	30868.6784	1.5876	0.1124	-11497.1637	109512.6426
condition_4	71029.5240	30876.5630	2.3004	0.0214	10509.1665	131549.8816
condition_5	111032.0481	31061.5686	3.5746	0.0004	50149.0658	171915.0303

Omnibus:	20612.577	Durbin-Watson:	1.983
Prob(Omnibus):	0.000	Jarque-Bera (JB):	3761631.516
Skew:	4.131	Prob(JB):	0.000
Kurtosis:	67.124	Condition No.:	514357015562337255424



Breaking out categorical variable improved R-Squared to from .69 to .80. Some of the zipcode categories has large p-values but most do not so they can remain in the model. The distribution

of errors and homoscedascity must still be improved. Curiously bedrooms and floors have a negative coefficients. Something to consider while fine tuning the model.

Model 3

Adressing Multicolinearity

In [38]:
```
## Get the correlation matrix for our model_df (without the target)
corr = df.drop('price',axis=1).corr()
corr.round(2)
```

Out[38]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	grade
bedrooms	1.00	0.51	0.58	0.03	0.18	-0.00	0.03	0.36
bathrooms	0.51	1.00	0.76	0.09	0.50	0.06	-0.13	0.67
sqft_living	0.58	0.76	1.00	0.17	0.35	0.10	-0.06	0.76
sqft_lot	0.03	0.09	0.17	1.00	-0.00	0.02	-0.01	0.11
floors	0.18	0.50	0.35	-0.00	1.00	0.02	-0.26	0.46
waterfront	-0.00	0.06	0.10	0.02	0.02	1.00	0.02	0.08
condition	0.03	-0.13	-0.06	-0.01	-0.26	0.02	1.00	-0.15
grade	0.36	0.67	0.76	0.11	0.46	0.08	-0.15	1.00
sqft_above	0.48	0.69	0.88	0.18	0.52	0.07	-0.16	0.76
sqft_basement	0.30	0.28	0.43	0.02	-0.24	0.08	0.17	0.17
yr_built	0.16	0.51	0.32	0.05	0.49	-0.02	-0.36	0.45
yr_renovated	0.02	0.05	0.05	0.00	0.00	0.07	-0.06	0.02
zipcode	-0.15	-0.20	-0.20	-0.13	-0.06	0.03	0.00	-0.19
lat	-0.01	0.02	0.05	-0.09	0.05	-0.01	-0.02	0.11
long	0.13	0.22	0.24	0.23	0.13	-0.04	-0.11	0.20
sqft_living15	0.39	0.57	0.76	0.14	0.28	0.08	-0.09	0.71
sqft_lot15	0.03	0.09	0.18	0.72	-0.01	0.03	-0.00	0.12
renovated	0.02	0.05	0.05	0.01	0.00	0.07	-0.06	0.02
basement	0.16	0.16	0.20	-0.03	-0.25	0.04	0.13	0.05
age	-0.16	-0.51	-0.32	-0.05	-0.49	0.02	0.36	-0.45

In [39]:
```
abs(corr.round(2)) > 0.75
```

Out[39]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	grade
bedrooms	True	False	False	False	False	False	False	False
bathrooms	False	True	True	False	False	False	False	False
sqft_living	False	True	True	False	False	False	False	True

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

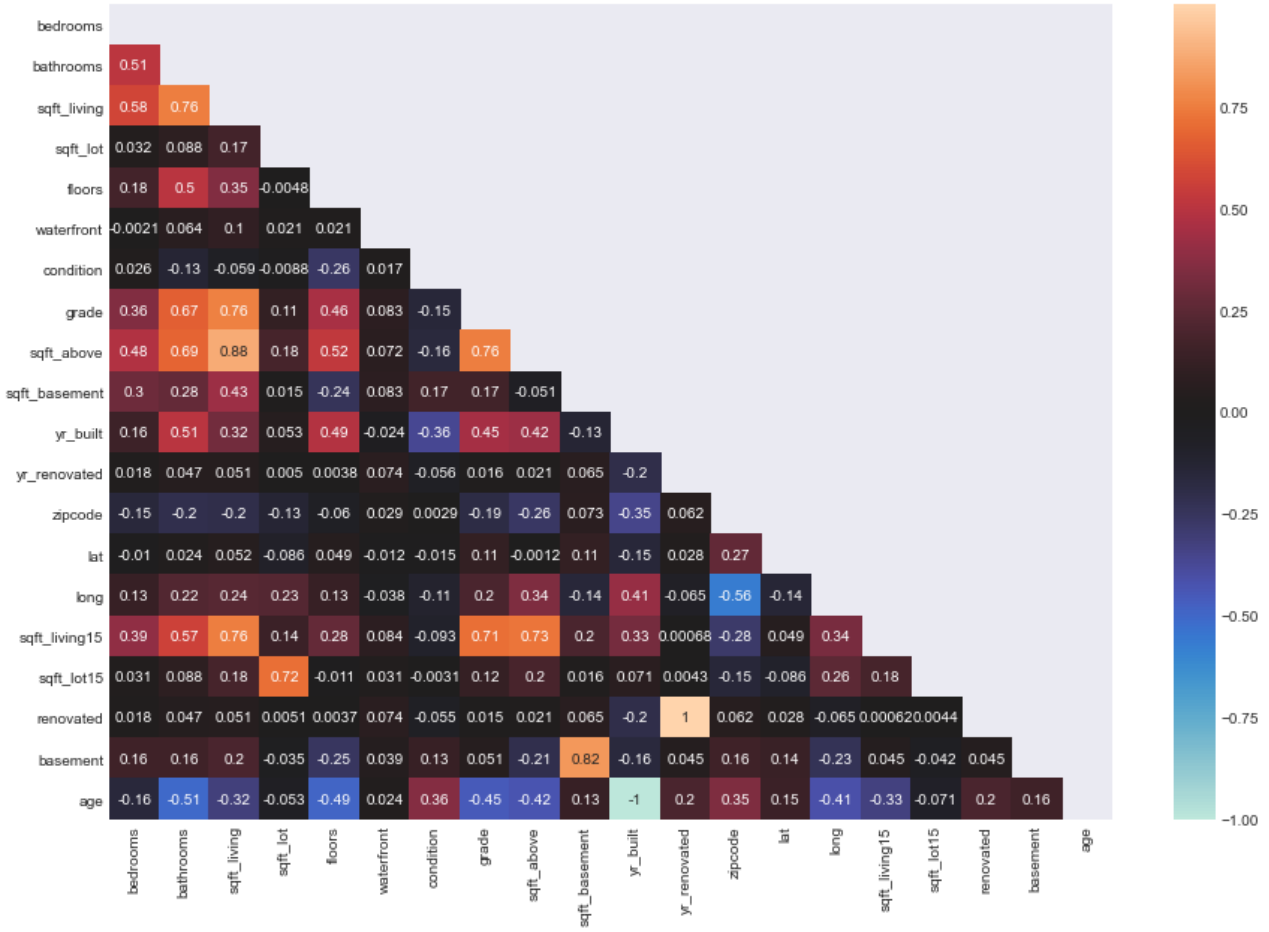
In [40]:
```
# stacks the row:column pairs into a multindex
# reset the index to set the multindex to seperate columns
# sort values. 0 is the column automatically generated by the stacking
df_corr = corr.abs().stack().reset_index().sort_values(0, ascending=False)

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

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

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

# rename correlation column as cc rather than 0
df_corr.columns = ['cc']
df_corr.drop_duplicates(inplace=True)

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

Out[40]:

	cc
pairs	
(yr_renovated, renovated)	1.00
(sqft_above, sqft_living)	0.88
(sqft_basement, basement)	0.82

	cc
pairs	
(grade, sqft_living)	0.76
(sqft_living15, sqft_living)	0.76
(grade, sqft_above)	0.76
(sqft_living, bathrooms)	0.76

In [41]:
```
## Plot this as a heatmap

fig, ax = plt.subplots(figsize=(15,10))

sns.heatmap(corr,ax=ax, center=0,annot=True,mask=np.triu(np.ones_like(corr, dtyp
```



In [42]:
```
df_trimmed = df.copy()
# Trim for multicolinearty
df_trimmed.drop(columns=['sqft_living','yr_renovated','yr_built','sqft_basement'
```

In [44]:
```
model3 = make_model(df_trimmed,cat_cols=cat_cols)
evaluation(model3,df_trimmed)
```

Model:	OLS	Adj. R-squared:	0.787
Dependent Variable:	price	AIC:	581450.4938
Date:	2021-11-04 12:35	BIC:	582144.7808

No. Observations:	21597	Log-Likelihood:	-2.9064e+05
Df Model:	86	F-statistic:	929.9
Df Residuals:	21510	Prob (F-statistic):	0.00
R-squared:	0.788	Scale:	2.8720e+10

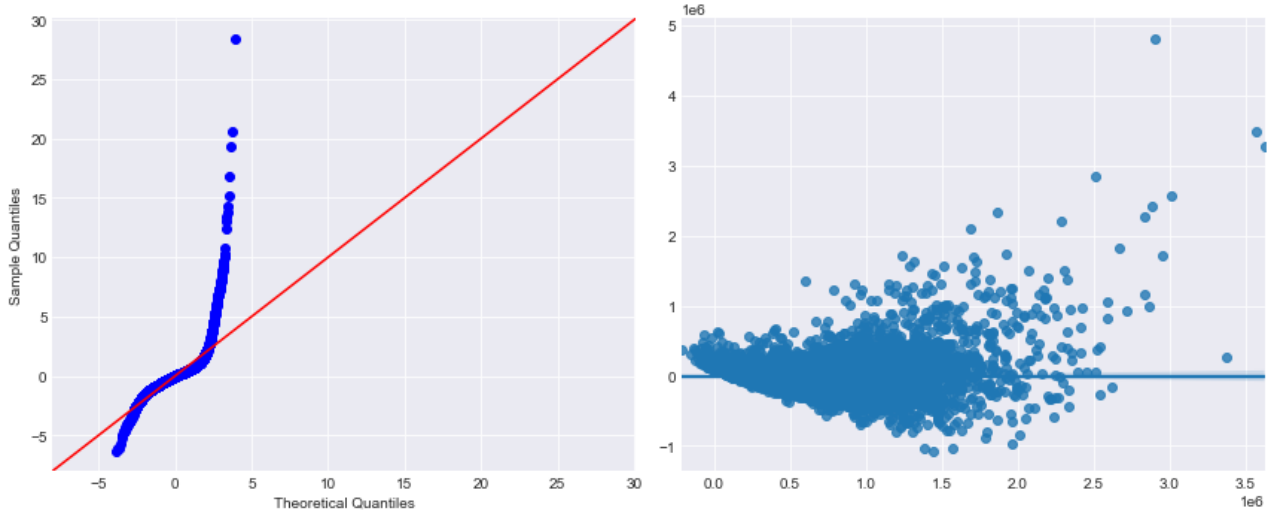
	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
Intercept	-26362798.0347	6467151.7865	-4.0764	0.0000	-39038895.9003	-13686700.1690
C(condition) [T.2]	49437.0569	34178.4511	1.4464	0.1481	-17555.2459	116429.3597
C(condition) [T.3]	38910.8335	31733.6525	1.2262	0.2201	-23289.4825	101111.1495
C(condition) [T.4]	64973.7892	31741.3153	2.0470	0.0407	2758.4536	127189.1249
C(condition) [T.5]	110519.4061	31931.7018	3.4611	0.0005	47930.8988	173107.9133
C(zipcode) [T.98002]	38817.1719	15263.2341	2.5432	0.0110	8900.0993	68734.2445
C(zipcode) [T.98003]	-22732.0139	13645.2706	-1.6659	0.0957	-49477.7578	4013.7301
C(zipcode) [T.98004]	725074.8572	24805.5328	29.2304	0.0000	676454.1704	773695.5440
C(zipcode) [T.98005]	239599.0847	26512.7758	9.0371	0.0000	187632.0749	291566.0945
C(zipcode) [T.98006]	251611.6140	21669.6156	11.6113	0.0000	209137.5579	294085.6701
C(zipcode) [T.98007]	196717.5783	27359.3830	7.1901	0.0000	143091.1554	250344.0011
C(zipcode) [T.98008]	229666.0419	25973.8421	8.8422	0.0000	178755.3821	280576.7017
C(zipcode) [T.98010]	96170.0557	23262.8247	4.1341	0.0000	50573.1915	141766.9200
C(zipcode) [T.98011]	47606.0384	33793.2379	1.4087	0.1589	-18631.2179	113843.2947
C(zipcode) [T.98014]	108809.7134	37108.9836	2.9322	0.0034	36073.3492	181546.0776
C(zipcode) [T.98019]	61354.6876	36610.9094	1.6759	0.0938	-10405.4141	133114.7893
C(zipcode) [T.98022]	60900.3753	20195.3444	3.0156	0.0026	21316.0003	100484.7504
C(zipcode) [T.98023]	-57293.7007	12553.1705	-4.5641	0.0000	-81898.8474	-32688.5541
C(zipcode) [T.98024]	157947.2092	32654.2069	4.8370	0.0000	93942.5382	221951.8803
C(zipcode) [T.98027]	154160.8158	22245.9010	6.9299	0.0000	110557.1974	197764.4341

C(zipcode) [T.98028]	52679.7623	32821.4576	1.6050	0.1085	-11652.7324	117012.2571
C(zipcode) [T.98029]	194476.8181	25393.3130	7.6586	0.0000	144704.0385	244249.5977
C(zipcode) [T.98030]	-1943.2510	15000.2417	-0.1295	0.8969	-31344.8389	27458.3370
C(zipcode) [T.98031]	1842.9095	15626.1159	0.1179	0.9061	-28785.4384	32471.2573
C(zipcode) [T.98032]	-2530.6096	18142.2863	-0.1395	0.8891	-38090.8383	33029.6191
C(zipcode) [T.98033]	314849.9146	28158.0156	11.1815	0.0000	259658.1126	370041.7167
C(zipcode) [T.98034]	143305.0095	30186.5932	4.7473	0.0000	84137.0446	202472.9743
C(zipcode) [T.98038]	47134.9937	16843.6414	2.7984	0.0051	14120.2054	80149.7820
C(zipcode) [T.98039]	1247016.6716	33514.3596	37.2084	0.0000	1181326.0374	1312707.3059
C(zipcode) [T.98040]	482086.5559	21938.2300	21.9747	0.0000	439085.9957	525087.1161
C(zipcode) [T.98042]	9533.0780	14357.1376	0.6640	0.5067	-18607.9781	37674.1341
C(zipcode) [T.98045]	144286.9433	31101.3435	4.6393	0.0000	83325.9999	205247.8866
C(zipcode) [T.98052]	167529.8961	28744.9379	5.8282	0.0000	111187.6826	223872.1095
C(zipcode) [T.98053]	150608.4236	30803.2557	4.8894	0.0000	90231.7544	210985.0927
C(zipcode) [T.98055]	31816.1807	17398.9825	1.8286	0.0675	-2287.1175	65919.4788
C(zipcode) [T.98056]	79407.9496	18900.8885	4.2013	0.0000	42360.8043	116455.0949
C(zipcode) [T.98058]	11711.5091	16438.7134	0.7124	0.4762	-20509.5902	43932.6084
C(zipcode) [T.98059]	53751.2016	18542.9155	2.8987	0.0038	17405.7098	90096.6933
C(zipcode) [T.98065]	107651.9369	28663.1731	3.7558	0.0002	51469.9885	163833.8852
C(zipcode) [T.98070]	-36510.9812	21695.4103	-1.6829	0.0924	-79035.5970	6013.6346
C(zipcode) [T.98072]	80447.9067	33623.7572	2.3926	0.0167	14542.8450	146352.9683
C(zipcode) [T.98074]	129713.5775	27210.9477	4.7670	0.0000	76378.0988	183049.0562
C(zipcode) [T.98075]	129817.9754	26160.1775	4.9624	0.0000	78542.0844	181093.8664
C(zipcode)	38381.8994	34979.2711	1.0973	0.2725	-30180.0701	106943.8690

[T.98077]						
C(zipcode) [T.98092]	-32317.9551	13637.1621	-2.3698	0.0178	-59047.8059	-5588.1044
C(zipcode) [T.98102]	442325.8079	29071.8371	15.2149	0.0000	385342.8478	499308.7681
C(zipcode) [T.98103]	279143.9278	27223.4842	10.2538	0.0000	225783.8766	332503.9790
C(zipcode) [T.98105]	396609.7256	27961.8089	14.1840	0.0000	341802.5032	451416.9479
C(zipcode) [T.98106]	108877.5294	20173.4584	5.3971	0.0000	69336.0526	148419.0063
C(zipcode) [T.98107]	282108.4322	28069.0518	10.0505	0.0000	227091.0059	337125.8586
C(zipcode) [T.98108]	89695.1718	22272.1382	4.0272	0.0001	46040.1266	133350.2171
C(zipcode) [T.98109]	437310.6335	28918.0191	15.1224	0.0000	380629.1681	493992.0990
C(zipcode) [T.98112]	544855.7373	25670.3984	21.2251	0.0000	494539.8496	595171.6249
C(zipcode) [T.98115]	271854.1421	27677.4143	9.8222	0.0000	217604.3543	326103.9300
C(zipcode) [T.98116]	256766.8990	22513.1098	11.4052	0.0000	212639.5316	300894.2663
C(zipcode) [T.98117]	249691.0473	28027.9636	8.9086	0.0000	194754.1569	304627.9377
C(zipcode) [T.98118]	148120.3656	19669.4226	7.5305	0.0000	109566.8363	186673.8948
C(zipcode) [T.98119]	420205.3388	27316.8005	15.3827	0.0000	366662.3808	473748.2968
C(zipcode) [T.98122]	280907.2114	24374.7183	11.5245	0.0000	233130.9530	328683.4698
C(zipcode) [T.98125]	145368.0996	29885.1451	4.8642	0.0000	86790.9955	203945.2037
C(zipcode) [T.98126]	164508.9032	20672.9743	7.9577	0.0000	123988.3381	205029.4683
C(zipcode) [T.98133]	92695.6072	30863.9817	3.0034	0.0027	32199.9106	153191.3039
C(zipcode) [T.98136]	216510.0781	21193.7257	10.2158	0.0000	174968.8015	258051.3548
C(zipcode) [T.98144]	248603.8298	22653.5150	10.9742	0.0000	204201.2579	293006.4018
C(zipcode) [T.98146]	91796.4138	18914.9025	4.8531	0.0000	54721.7998	128871.0277
C(zipcode) [T.98148]	34985.9235	25763.4953	1.3580	0.1745	-15512.4410	85484.2879
C(zipcode) [T.98155]	76697.9488	32088.4188	2.3902	0.0168	13802.2644	139593.6332

C(zipcode) [T.98166]	37103.6733	17319.0209	2.1424	0.0322	3157.1059	71050.2407
C(zipcode) [T.98168]	53610.3287	18303.8611	2.9289	0.0034	17733.4014	89487.2561
C(zipcode) [T.98177]	162509.1327	32198.5781	5.0471	0.0000	99397.5279	225620.7374
C(zipcode) [T.98178]	41349.1119	18893.4258	2.1885	0.0286	4316.5940	78381.6299
C(zipcode) [T.98188]	25138.1417	19407.0921	1.2953	0.1952	-12901.2003	63177.4837
C(zipcode) [T.98198]	6897.8881	14688.9854	0.4696	0.6386	-21893.6143	35689.3906
C(zipcode) [T.98199]	325715.4837	26614.6770	12.2382	0.0000	273548.7399	377882.2275
bedrooms	-19969.3144	1586.4360	-12.5875	0.0000	-23078.8468	-16859.7819
bathrooms	48509.3918	2719.3507	17.8386	0.0000	43179.2625	53839.5211
sqft_lot	0.2581	0.0308	8.3787	0.0000	0.1977	0.3184
floors	-63726.9516	3281.6960	-19.4189	0.0000	-70159.3196	-57294.5836
waterfront	880760.5774	14533.0978	60.6038	0.0000	852274.6262	909246.5286
grade	70402.7909	1910.7733	36.8452	0.0000	66657.5333	74148.0485
sqft_above	185.1275	3.2154	57.5746	0.0000	178.8250	191.4300
lat	153662.9282	66895.3432	2.2971	0.0216	22543.0867	284782.7696
long	-150609.2241	47970.3517	-3.1396	0.0017	-244634.6766	-56583.7716
sqft_living15	41.4629	2.9640	13.9888	0.0000	35.6532	47.2725
renovated	46652.4423	6751.0028	6.9104	0.0000	33419.9753	59884.9092
basement	69115.3855	3175.7695	21.7634	0.0000	62890.6413	75340.1297
age	1044.2850	68.0173	15.3532	0.0000	910.9661	1177.6039

Omnibus:	21732.790	Durbin-Watson:	1.985
Prob(Omnibus):	0.000	Jarque-Bera (JB):	4745561.238
Skew:	4.479	Prob(JB):	0.000
Kurtosis:	75.065	Condition No.:	247297793



The issue of multicollinearity may have been solved but R-Squared value fell from .80 to .78. Small decrease but the residuals are less normally distributed. The condition category has large p-values so will drop as a categorical feature and try as an ordinal feature.

Model 4

Removing Outliers

In [54]:
```
df_scaled = df_trimmed.copy()
df_scaled.drop(columns=['renovated'],inplace=True)
```

In [55]:
```
ss = StandardScaler()
```

In [56]:
```
scaled_cols = list(df_scaled.columns)
# outlier_cols = ('waterfront','condition','grade','zipcode','renovated','baseme
# filter_cols = [col for col in scaled_cols if col not in outlier_cols]
```

In [57]:
```
# Dataframe scaled by z-score
df_z = pd.DataFrame(ss.fit_transform(df_scaled),columns=scaled_cols,index=df_sca
df_z
```

Out[57]:

id	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	condition	grade	sqft_al
7129300520	-0.87	-0.40	-1.45	-0.23	-0.92	-0.08	-0.63	-0.56	-
6414100192	-0.01	-0.40	0.17	-0.19	0.94	-0.08	-0.63	-0.56	
5631500400	-0.98	-1.48	-1.45	-0.12	-0.92	-0.08	-0.63	-1.41	-
2487200875	0.17	0.68	1.15	-0.24	-0.92	-0.08	2.44	-0.56	-
1954400510	-0.08	-0.40	-0.15	-0.17	-0.92	-0.08	-0.63	0.29	-
...	...	...	...	...	...	...	...	...	
263000018	-0.49	-0.40	0.50	-0.34	2.79	-0.08	-0.63	0.29	-
6600060120	-0.38	0.68	0.50	-0.22	0.94	-0.08	-0.63	0.29	
1523300141	-0.38	-1.48	-1.78	-0.33	0.94	-0.08	-0.63	-0.56	-

id	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	condition	grade	sqft_al
291310100	-0.38	-0.40	0.50	-0.31	0.94	-0.08	-0.63	0.29	-
1523300157	-0.59	-1.48	-1.78	-0.34	0.94	-0.08	-0.63	-0.56	-

21597 rows × 15 columns

In [58]:
```
df_z.describe()
```

Out[58]:

	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	condition	grade	sqft_
count	21597.00	21597.00	21597.00	21597.00	21597.00	21597.00	21597.00	21597.00	21
mean	-0.00	-0.00	0.00	0.00	-0.00	-0.00	0.00	0.00	
std	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	
min	-1.26	-2.56	-2.10	-0.35	-0.92	-0.08	-3.70	-3.97	
25%	-0.59	-0.40	-0.48	-0.24	-0.92	-0.08	-0.63	-0.56	
50%	-0.25	-0.40	0.17	-0.18	0.01	-0.08	-0.63	-0.56	
75%	0.29	0.68	0.50	-0.11	0.94	-0.08	0.91	0.29	
max	19.49	31.98	7.65	39.51	3.72	12.12	2.44	4.55	

Looking at the max and min values there are clearly some large outliers that are skewing the data and therefore should be removed.

In [59]:
```
idx_outliers = (df_z[scaled_cols] < -3) | (df_z[scaled_cols] > 3)
idx_outliers
```

Out[59]:

id	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	condition	grade	sqft_al
7129300520	False	False	False	False	False	False	False	False	F
6414100192	False	False	False	False	False	False	False	False	F
5631500400	False	False	False	False	False	False	False	False	F
2487200875	False	False	False	False	False	False	False	False	F
1954400510	False	False	False	False	False	False	False	False	F
...	...	...	...	...	...	...	...	...	
263000018	False	False	False	False	False	False	False	False	F
6600060120	False	False	False	False	False	False	False	False	F
1523300141	False	False	False	False	False	False	False	False	F
291310100	False	False	False	False	False	False	False	False	F
1523300157	False	False	False	False	False	False	False	False	F

21597 rows × 15 columns

In [60]:
```
idx_outliers.sum()
```

Out[60]:
```
price            406
bedrooms          62
bathrooms        187
sqft_lot         346
floors             7
waterfront       146
condition         29
grade            130
sqft_above       253
zipcode            0
lat                0
long             232
sqft_living15    236
basement           0
age                0
dtype: int64
```

In [61]:
```
idx_outliers_any = idx_outliers.any(axis=1)

idx_outliers_any.sum()
```

Out[61]:
```
1423
```

The dataset has a total of 1423 outliers. Since this is only a small portion of the full dataset we can remove all outlier records from the dataset.

In [63]:
```
df_no_outliers = df_scaled[~idx_outliers_any].copy()
df_no_outliers
```

Out[63]:

id	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	condition	grade	sq
7129300520	221900.00	3	1.00	5650	1.00	0.00	3	7	
6414100192	538000.00	3	2.25	7242	2.00	0.00	3	7	
5631500400	180000.00	2	1.00	10000	1.00	0.00	3	6	
2487200875	604000.00	4	3.00	5000	1.00	0.00	5	7	
1954400510	510000.00	3	2.00	8080	1.00	0.00	3	8	
...	...	...	...	...	...	...	...	...	...
263000018	360000.00	3	2.50	1131	3.00	0.00	3	8	
6600060120	400000.00	4	2.50	5813	2.00	0.00	3	8	
1523300141	402101.00	2	0.75	1350	2.00	0.00	3	7	
291310100	400000.00	3	2.50	2388	2.00	0.00	3	8	
1523300157	325000.00	2	0.75	1076	2.00	0.00	3	7	

20174 rows × 15 columns

The resulting dataset still contains over 20,000 records.

In [64]:
```
df_no_outliers.describe()
```

Out[64]:

	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	condition	grade
count	20174.00	20174.00	20174.00	20174.00	20174.00	20174.00	20174.00	20174.00
mean	496900.73	3.34	2.06	10415.41	1.48	0.00	3.42	7.57
std	247415.84	0.86	0.70	13136.77	0.54	0.00	0.65	1.05
min	82000.00	1.00	0.50	520.00	1.00	0.00	2.00	5.00
25%	317000.00	3.00	1.50	5000.00	1.00	0.00	3.00	7.00
50%	440000.00	3.00	2.25	7415.00	1.00	0.00	3.00	7.00
75%	618000.00	4.00	2.50	10046.75	2.00	0.00	4.00	8.00
max	1640000.00	6.00	4.25	138085.00	3.00	0.00	5.00	11.00

In [65]:
```
# All waterfront properties are outliers
df_no_outliers.drop(columns=['waterfront'],axis=1,inplace=True)
```

In [92]:
```
# Try the model again with outliers removed
model4 = make_model(df_no_outliers,cat_cols=['zipcode','grade','condition','base
evaluation(model4, df_no_outliers)
```

Model:	OLS	Adj. R-squared:	0.819
Dependent Variable:	price	AIC:	523892.0228
Date:	2021-11-04 12:57	BIC:	524596.2042
No. Observations:	20174	Log-Likelihood:	-2.6186e+05
Df Model:	88	F-statistic:	1041.
Df Residuals:	20085	Prob (F-statistic):	0.00
R-squared:	0.820	Scale:	1.1058e+10

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
Intercept	-28247902.7563	5241549.0588	-5.3892	0.0000	-38521769.2591	-17974036.2535
C(condition)[T.3]	30126.3432	8693.6335	3.4653	0.0005	13086.1077	47166.5787
C(condition)[T.4]	52273.5051	8729.6001	5.9881	0.0000	35162.7723	69384.2380
C(condition)[T.5]	90231.3151	9023.7689	9.9993	0.0000	72543.9871	107918.6431
C(grade)[T.6]	7053.4299	7703.9636	0.9156	0.3599	-8046.9712	22153.8311
C(grade)[T.7]	29461.7668	7606.6013	3.8732	0.0001	14552.2037	44371.3298
C(grade)[T.8]	77069.8998	7947.0851	9.6979	0.0000	61492.9606	92646.8390
C(grade)[T.9]	171763.3781	8528.4850	20.1400	0.0000	155046.8472	188479.9090
C(grade)[T.10]	256939.8341	9386.0292	27.3747	0.0000	238542.4462	275337.2220

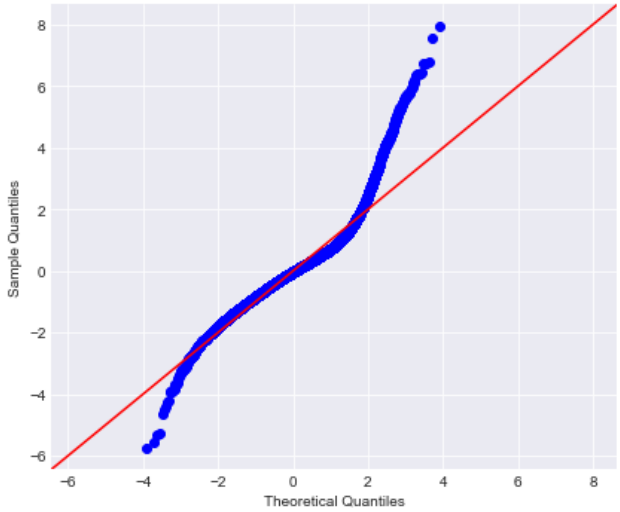
C(grade) [T.11]	362351.2223	12372.6308	29.2865	0.0000	338099.8501	386602.5945
C(zipcode) [T.98002]	24783.0620	9706.8224	2.5532	0.0107	5756.8932	43809.2309
C(zipcode) [T.98003]	-10131.5184	8570.0225	-1.1822	0.2371	-26929.4662	6666.4294
C(zipcode) [T.98004]	593424.7101	16812.1472	35.2974	0.0000	560471.5212	626377.8989
C(zipcode) [T.98005]	312970.3720	17749.3888	17.6327	0.0000	278180.1126	347760.6314
C(zipcode) [T.98006]	285342.8262	14928.2424	19.1143	0.0000	256082.2453	314603.4070
C(zipcode) [T.98007]	251665.9700	18426.6555	13.6577	0.0000	215548.2123	287783.7276
C(zipcode) [T.98008]	265064.2361	17916.4697	14.7944	0.0000	229946.4845	300181.9878
C(zipcode) [T.98010]	129998.0561	16993.9522	7.6497	0.0000	96688.5145	163307.5977
C(zipcode) [T.98011]	119182.2794	22449.1829	5.3090	0.0000	75180.0379	163184.5210
C(zipcode) [T.98014]	163503.1194	26663.0797	6.1322	0.0000	111241.2942	215764.9447
C(zipcode) [T.98019]	130580.2546	26204.4900	4.9831	0.0000	79217.3026	181943.2065
C(zipcode) [T.98022]	70371.5082	14945.1936	4.7086	0.0000	41077.7017	99665.3146
C(zipcode) [T.98023]	-46459.4826	8285.3406	-5.6074	0.0000	-62699.4305	-30219.5347
C(zipcode) [T.98024]	188778.2039	25365.7056	7.4423	0.0000	139059.3383	238497.0694
C(zipcode) [T.98027]	208650.3310	16206.2883	12.8747	0.0000	176884.6754	240415.9866
C(zipcode) [T.98028]	112939.6625	21628.0029	5.2219	0.0000	70547.0010	155332.3240
C(zipcode) [T.98029]	254560.4183	18644.4651	13.6534	0.0000	218015.7360	291105.1006
C(zipcode) [T.98030]	15160.6069	9797.8099	1.5473	0.1218	-4043.9050	34365.1187
C(zipcode) [T.98031]	22652.4510	10278.3668	2.2039	0.0275	2506.0082	42798.8937
C(zipcode) [T.98032]	-5033.4100	11344.7553	-0.4437	0.6573	-27270.0618	17203.2418
C(zipcode) [T.98033]	340930.0896	18906.4289	18.0325	0.0000	303871.9366	377988.2426
C(zipcode) [T.98034]	176889.2776	20149.5608	8.7788	0.0000	137394.4841	216384.0711

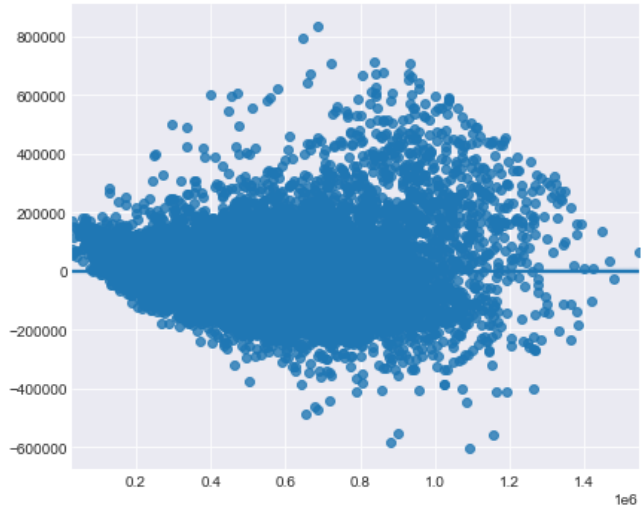
C(zipcode) [T.98038]	77784.9088	12750.5686	6.1005	0.0000	52792.7475	102777.0701
C(zipcode) [T.98039]	799003.3429	29864.7139	26.7541	0.0000	740466.0516	857540.6342
C(zipcode) [T.98040]	463902.7641	14670.2237	31.6221	0.0000	435147.9212	492657.6069
C(zipcode) [T.98042]	31948.1183	10198.2645	3.1327	0.0017	11958.6825	51937.5541
C(zipcode) [T.98045]	169453.2027	37643.7141	4.5015	0.0000	95668.4325	243237.9729
C(zipcode) [T.98052]	244980.1726	19757.4210	12.3994	0.0000	206254.0054	283706.3398
C(zipcode) [T.98053]	251020.4736	22064.0498	11.3769	0.0000	207773.1245	294267.8226
C(zipcode) [T.98055]	46324.3324	11424.4696	4.0548	0.0001	23931.4341	68717.2307
C(zipcode) [T.98056]	102362.7313	12676.7751	8.0748	0.0000	77515.2113	127210.2512
C(zipcode) [T.98058]	43922.7340	11320.8514	3.8798	0.0001	21732.9358	66112.5323
C(zipcode) [T.98059]	102885.3243	12793.5419	8.0420	0.0000	77808.9319	127961.7167
C(zipcode) [T.98065]	188527.6551	22307.4544	8.4513	0.0000	144803.2129	232252.0972
C(zipcode) [T.98070]	84648.4191	16316.7326	5.1878	0.0000	52666.2835	116630.5547
C(zipcode) [T.98072]	157338.8273	22823.0615	6.8939	0.0000	112603.7529	202073.9017
C(zipcode) [T.98074]	205287.7790	19436.3605	10.5620	0.0000	167190.9167	243384.6413
C(zipcode) [T.98075]	222980.4298	18961.4129	11.7597	0.0000	185814.5038	260146.3559
C(zipcode) [T.98077]	133606.1646	24338.6053	5.4895	0.0000	85900.5000	181311.8292
C(zipcode) [T.98092]	-7706.4714	9053.5880	-0.8512	0.3947	-25452.2471	10039.3044
C(zipcode) [T.98102]	398614.8983	18891.3658	21.1004	0.0000	361586.2702	435643.5264
C(zipcode) [T.98103]	300792.9486	17730.0860	16.9651	0.0000	266040.5243	335545.3730
C(zipcode) [T.98105]	399969.3124	18308.9777	21.8455	0.0000	364082.2130	435856.4119
C(zipcode) [T.98106]	94284.5268	13043.5116	7.2285	0.0000	68718.1731	119850.8805
C(zipcode) [T.98107]	286623.9081	18227.9466	15.7244	0.0000	250895.6361	322352.1801
C(zipcode)	96401.8410	14269.3866	6.7559	0.0000	68432.6716	124371.0104

[T.98108]						
C(zipcode) [T.98109]	432178.5753	18731.9539	23.0717	0.0000	395462.4078	468894.7429
C(zipcode) [T.98112]	478099.6180	16877.8917	28.3270	0.0000	445017.5644	511181.6715
C(zipcode) [T.98115]	307888.6864	18083.1181	17.0263	0.0000	272444.2902	343333.0825
C(zipcode) [T.98116]	278706.7996	14644.8065	19.0311	0.0000	250001.7765	307411.8227
C(zipcode) [T.98117]	276626.6271	18259.2628	15.1499	0.0000	240836.9728	312416.2814
C(zipcode) [T.98118]	155492.4663	12786.1767	12.1610	0.0000	130430.5102	180554.4225
C(zipcode) [T.98119]	410346.5356	17750.2019	23.1179	0.0000	375554.6826	445138.3886
C(zipcode) [T.98122]	299518.4950	15846.7256	18.9010	0.0000	268457.6117	330579.3784
C(zipcode) [T.98125]	176932.7345	19537.5576	9.0560	0.0000	138637.5175	215227.9516
C(zipcode) [T.98126]	167878.6908	13386.0521	12.5413	0.0000	141640.9297	194116.4520
C(zipcode) [T.98133]	119747.0228	20136.0795	5.9469	0.0000	80278.6536	159215.3919
C(zipcode) [T.98136]	233252.0281	13753.5555	16.9594	0.0000	206293.9302	260210.1260
C(zipcode) [T.98144]	245176.4809	14714.6982	16.6620	0.0000	216334.4643	274018.4976
C(zipcode) [T.98146]	96084.1525	12217.3598	7.8646	0.0000	72137.1242	120031.1809
C(zipcode) [T.98148]	36230.8245	16325.2761	2.2193	0.0265	4231.9429	68229.7061
C(zipcode) [T.98155]	110180.5503	21026.3828	5.2401	0.0000	68967.1137	151393.9869
C(zipcode) [T.98166]	84964.5236	11174.3622	7.6035	0.0000	63061.8563	106867.1909
C(zipcode) [T.98168]	34045.8041	11718.8990	2.9052	0.0037	11075.7999	57015.8084
C(zipcode) [T.98177]	196501.2677	21002.6896	9.3560	0.0000	155334.2717	237668.2637
C(zipcode) [T.98178]	54214.4636	12243.7859	4.4279	0.0000	30215.6379	78213.2893
C(zipcode) [T.98188]	30589.4689	12287.2724	2.4895	0.0128	6505.4063	54673.5315
C(zipcode) [T.98198]	26076.7335	9343.8520	2.7908	0.0053	7762.0165	44391.4505
C(zipcode) [T.98199]	336646.4756	17366.8823	19.3844	0.0000	302605.9605	370686.9907

C(basement)[T.1]	50629.5497	2071.8950	24.4363	0.0000	46568.4654	54690.6340
bedrooms	1201.2316	1124.6494	1.0681	0.2855	-1003.1736	3405.6368
bathrooms	36050.8546	1829.4366	19.7060	0.0000	32465.0087	39636.7006
sqft_lot	0.6234	0.0673	9.2658	0.0000	0.4915	0.7553
floors	-27633.8202	2183.1951	-12.6575	0.0000	-31913.0618	-23354.5785
sqft_above	109.9584	2.3495	46.8017	0.0000	105.3532	114.5635
lat	58051.0604	44339.9035	1.3092	0.1905	-28858.7909	144960.9117
long	-207178.2981	41662.6757	-4.9728	0.0000	-288840.5631	-125516.0331
sqft_living15	44.5409	2.1090	21.1198	0.0000	40.4072	48.6747
age	758.9964	42.8129	17.7282	0.0000	675.0796	842.9132

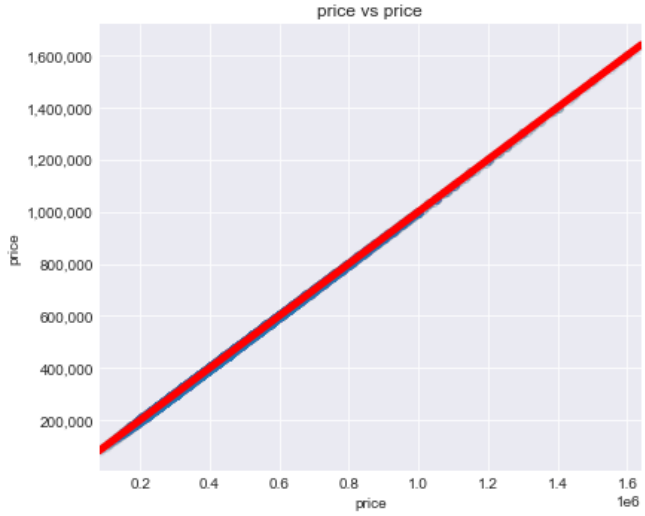
Omnibus:	5208.593	Durbin-Watson:	1.975
Prob(Omnibus):	0.000	Jarque-Bera (JB):	29965.045
Skew:	1.118	Prob(JB):	0.000
Kurtosis:	8.536	Condition No.:	119370568

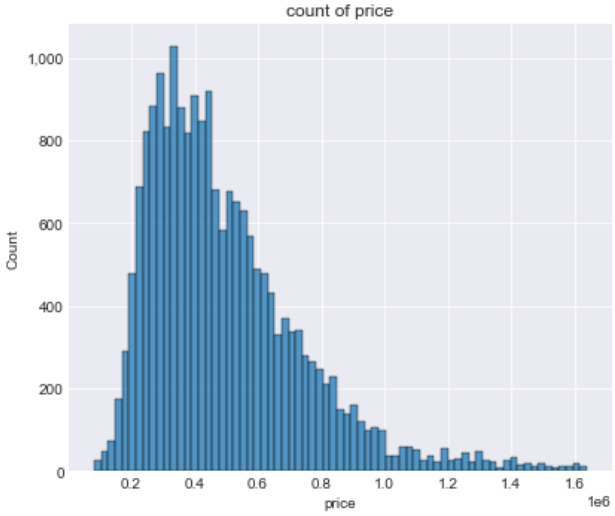


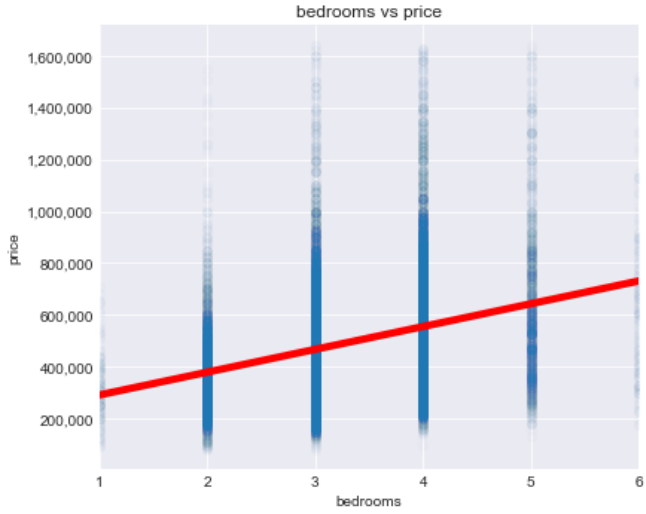


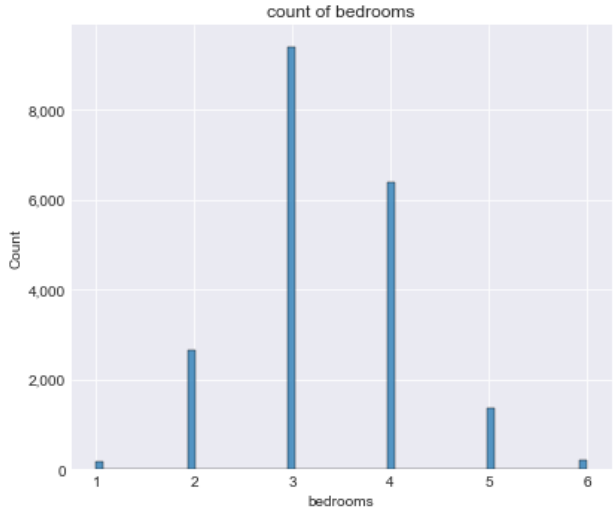
R-squared improves to .82. The homoscedasticity assumptions and distribution of residuals is looking much more normally distributed than before, can still be improved.

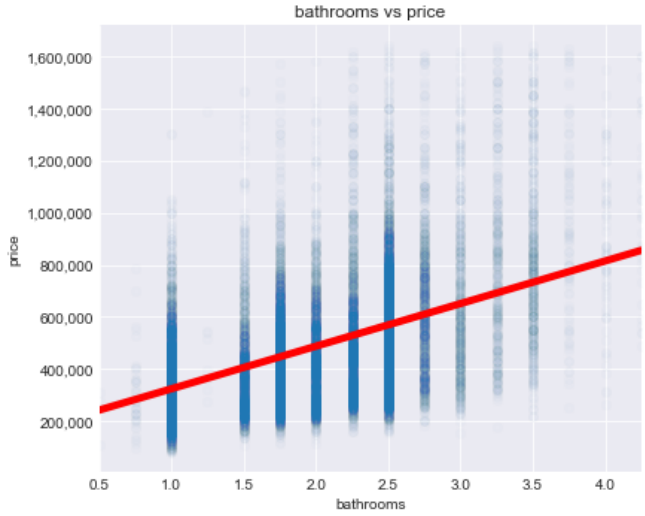
In [67]:
```
# Checking for Linearity with outliers removed
refined_cols = list(df_no_outliers.columns)

for col in refined_cols:
    plots(df_no_outliers, col, y='price')
```

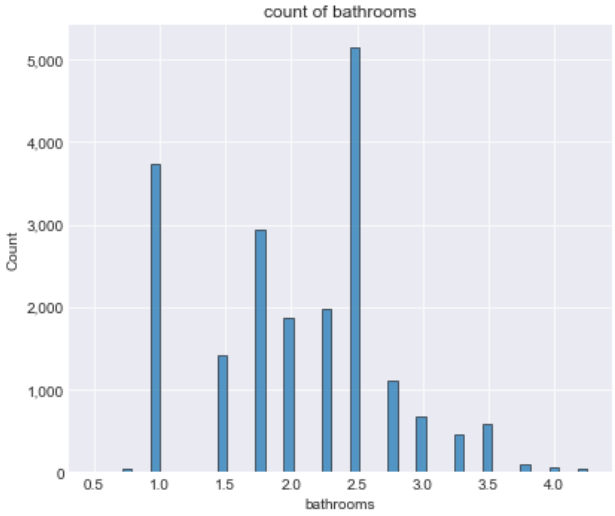

price vs price
price
price
1e6

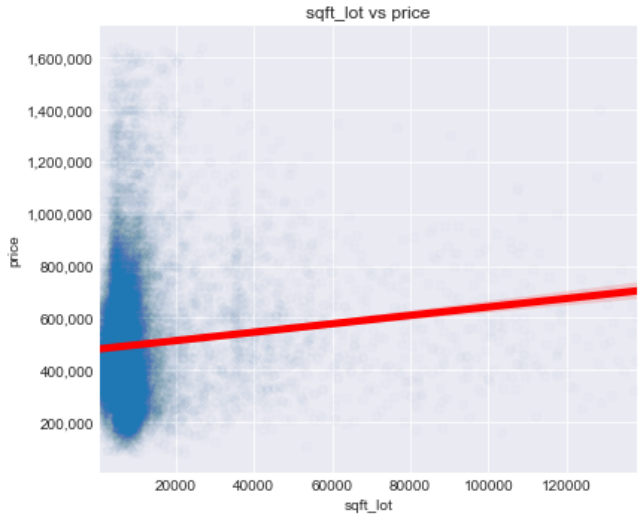

count of price
Count
price
1e6

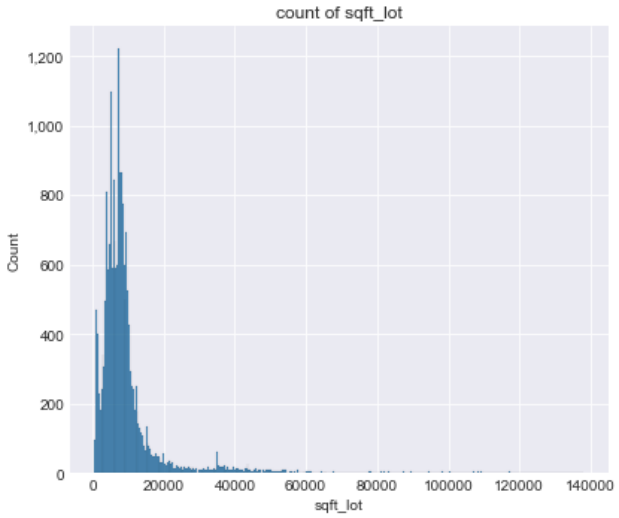

bedrooms vs price
price
bedrooms

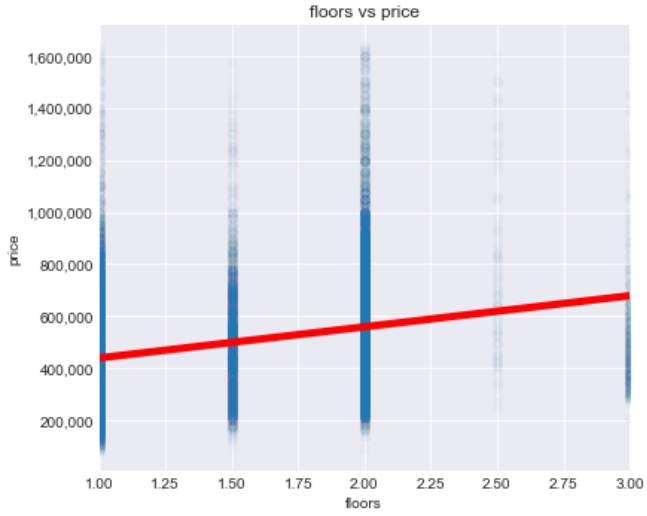

count of bedrooms
Count
bedrooms

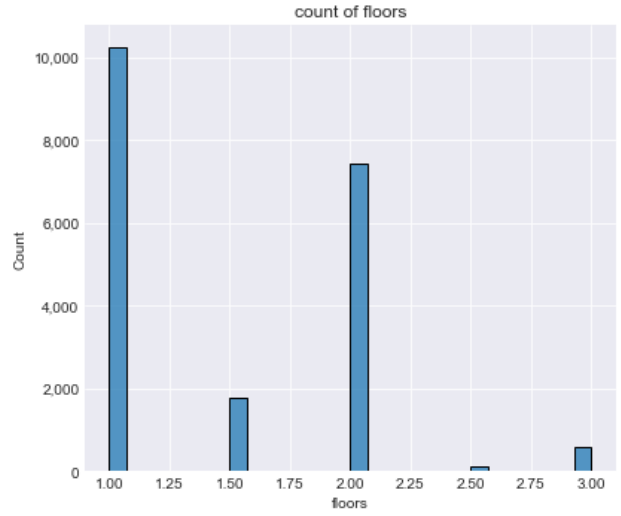

bathrooms vs price
price
bathrooms

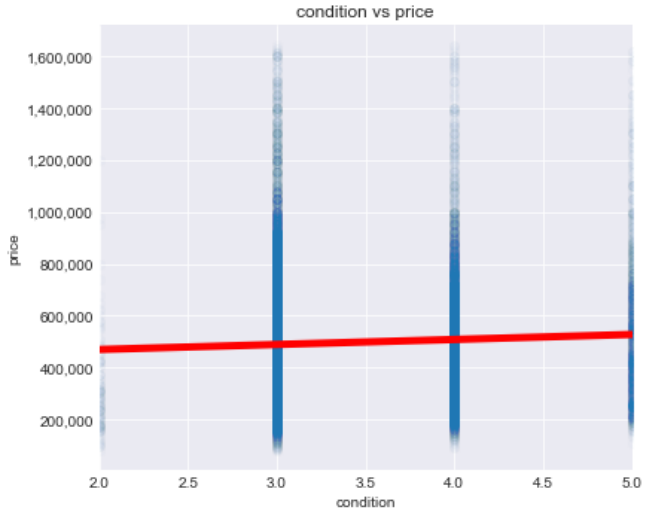

count of bathrooms
Count
bathrooms

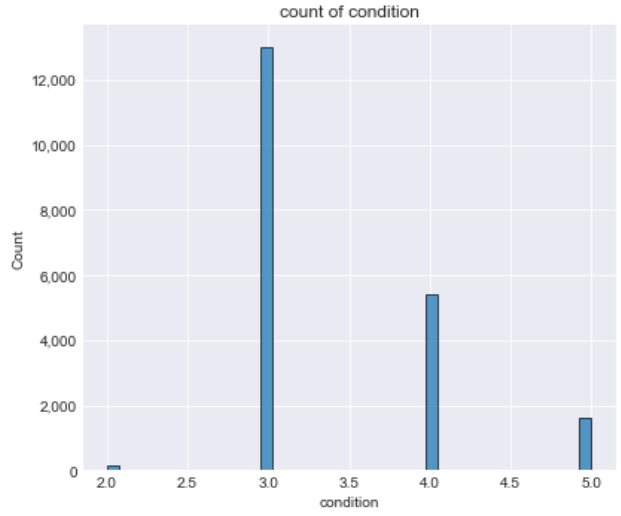

sqft_lot vs price
price
sqft_lot
1,600,000
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
20000
40000
60000
80000
100000
120000

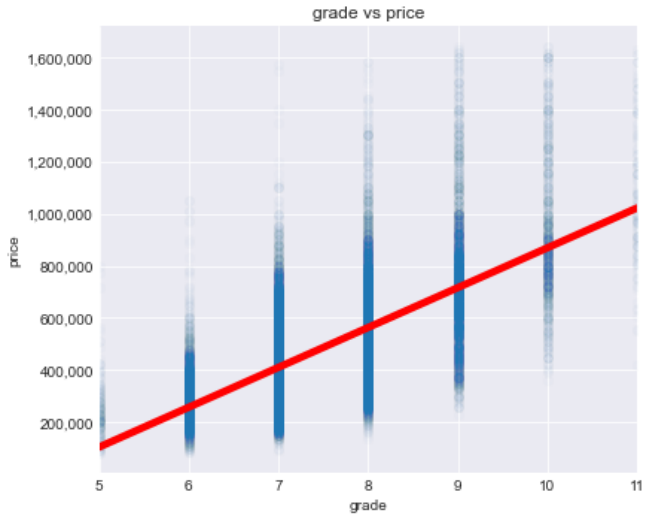

count of sqft_lot
Count
sqft_lot
1,200
1,000
800
600
400
200
0
0
20000
40000
60000
80000
100000
120000
140000

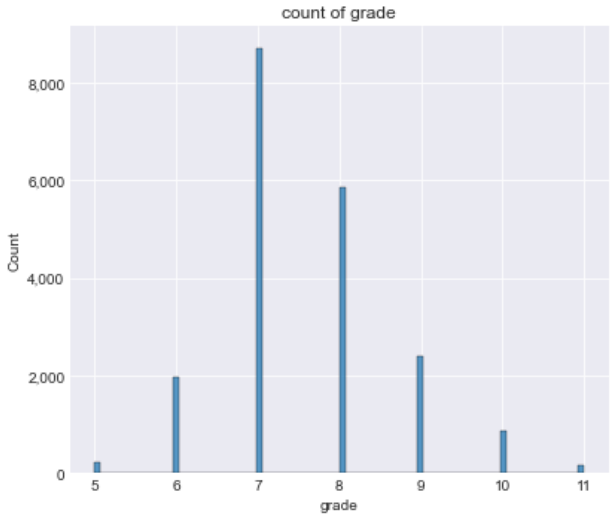

floors vs price
price
floors
1,600,000
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00

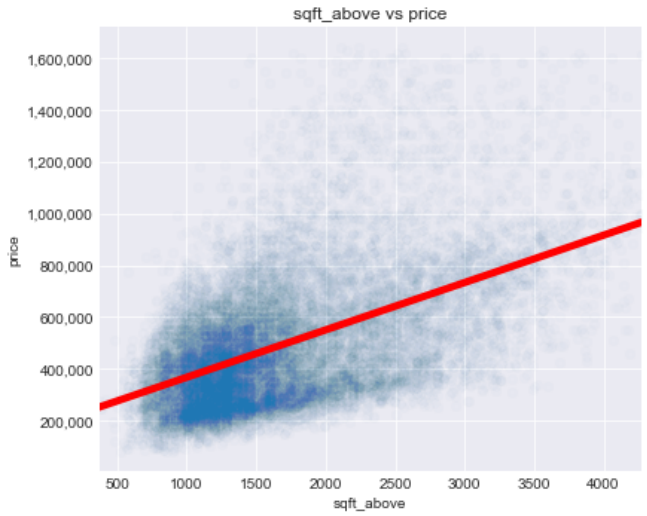

count of floors
Count
floors
10,000
8,000
6,000
4,000
2,000
0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00

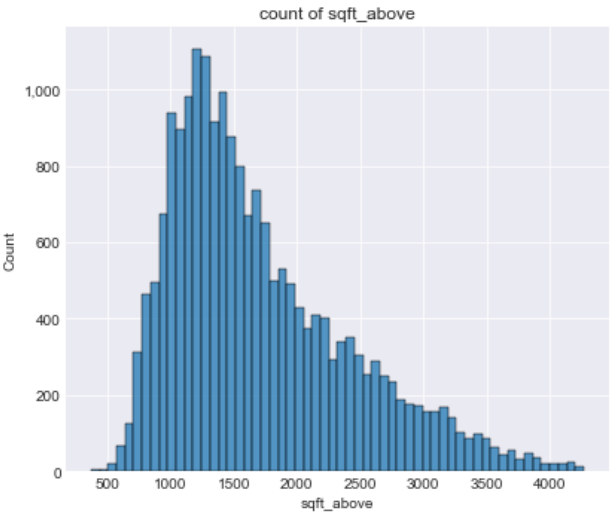

condition vs price
price
condition
1,600,000
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
2.0
2.5
3.0
3.5
4.0
4.5
5.0

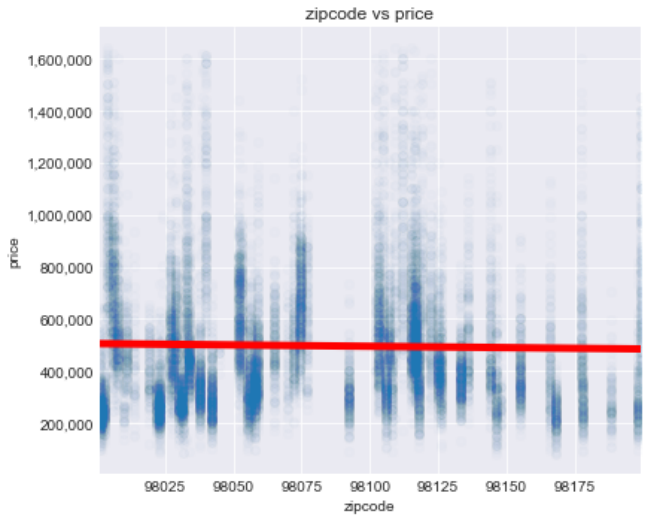

count of condition
Count
condition
12,000
10,000
8,000
6,000
4,000
2,000
0
2.0
2.5
3.0
3.5
4.0
4.5
5.0

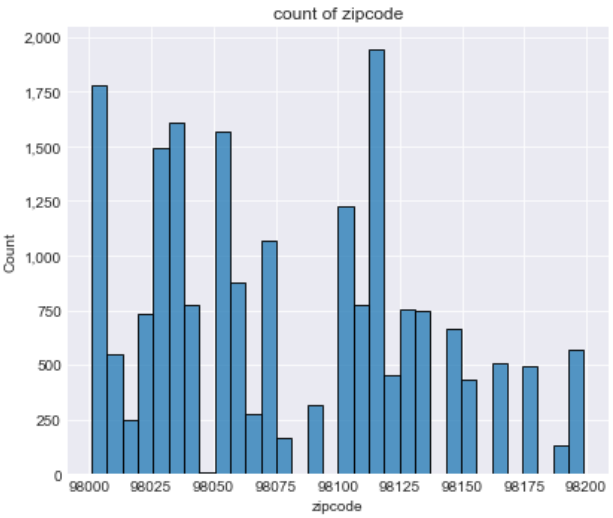

grade vs price
price
grade
1,600,000
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
5
6
7
8
9
10
11

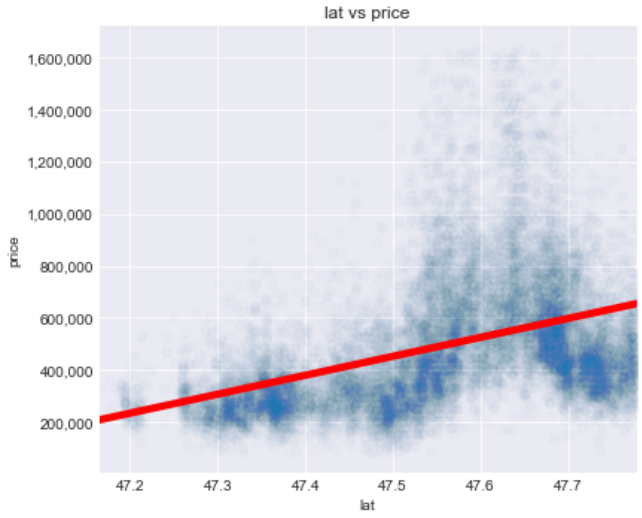

count of grade
Count
grade
8,000
6,000
4,000
2,000
0
5
6
7
8
9
10
11

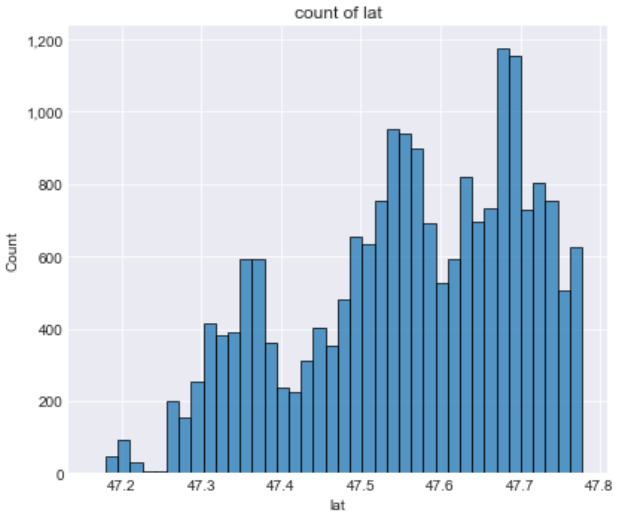

sqft_above vs price
price
sqft_above
1,600,000
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
500
1000
1500
2000
2500
3000
3500
4000

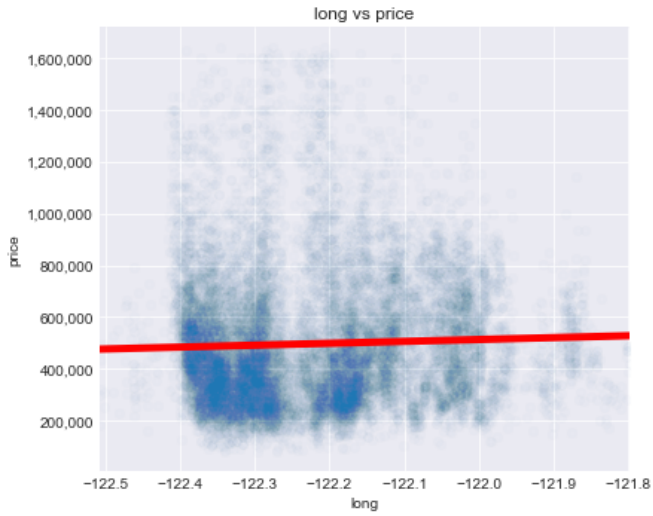

count of sqft_above
Count
sqft_above
1,000
800
600
400
200
0
500
1000
1500
2000
2500
3000
3500
4000

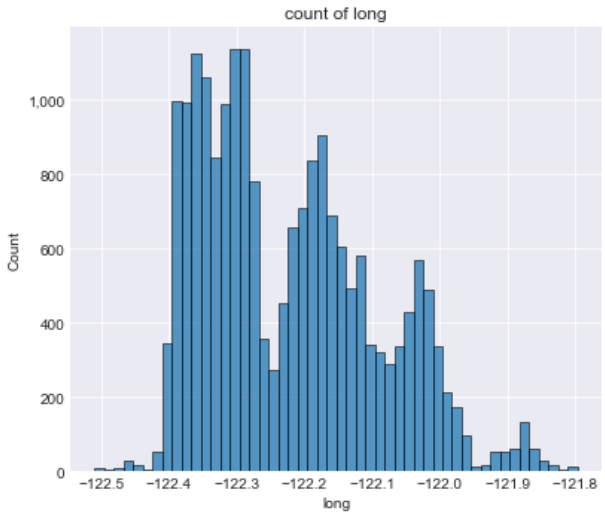

zipcode vs price
price
zipcode
1,600,000
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
98025
98050
98075
98100
98125
98150
98175

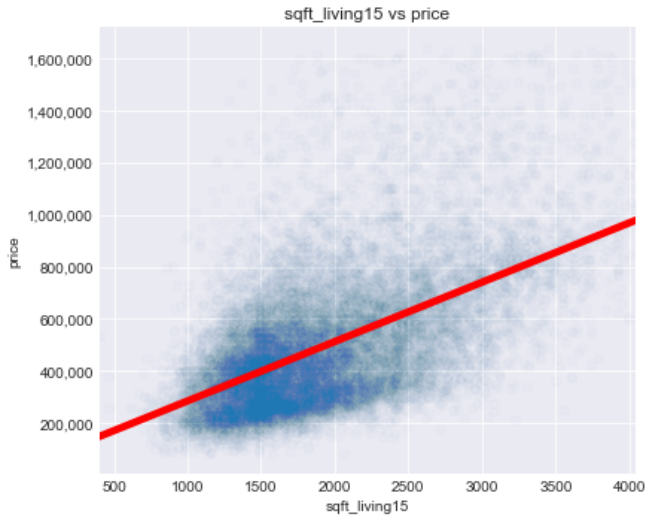

count of zipcode
Count
zipcode
2,000
1,750
1,500
1,250
1,000
750
500
250
0
98000
98025
98050
98075
98100
98125
98150
98175
98200

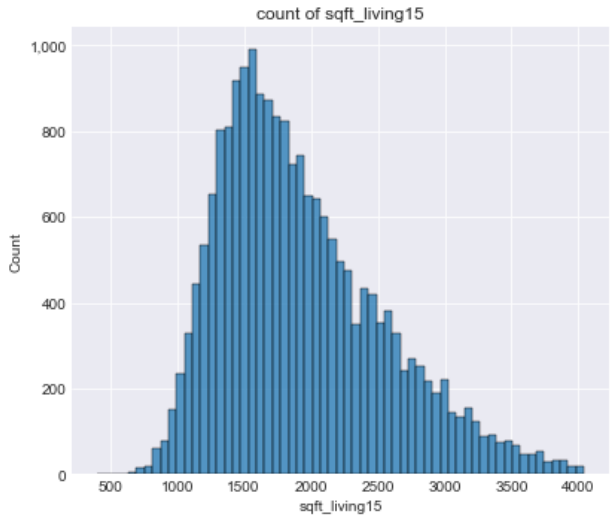

lat vs price
price
lat
1,600,000
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
47.2
47.3
47.4
47.5
47.6
47.7

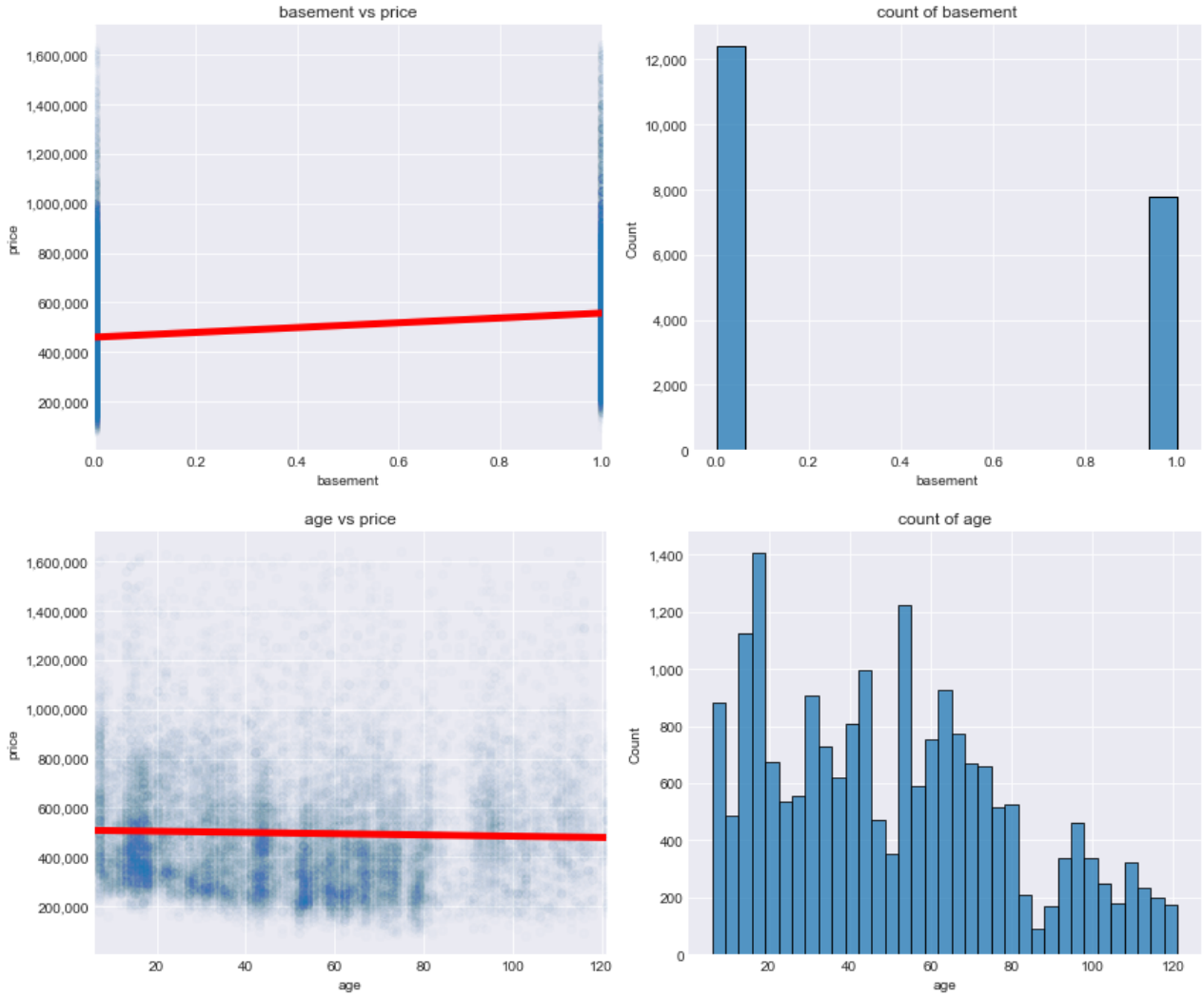

count of lat
Count
lat
1,200
1,000
800
600
400
200
0
47.2
47.3
47.4
47.5
47.6
47.7
47.8


long vs price
price
long
1,600,000
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
−122.5
−122.4
−122.3
−122.2
−122.1
−122.0
−121.9
−121.8


count of long
Count
long
1,000
800
600
400
200
0
−122.5
−122.4
−122.3
−122.2
−122.1
−122.0
−121.9
−121.8


sqft_living15 vs price
price
sqft_living15
1,600,000
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
500
1000
1500
2000
2500
3000
3500
4000


count of sqft_living15
Count
sqft_living15
1,000
800
600
400
200
0
500
1000
1500
2000
2500
3000
3500
4000



Assumptions of linearity are much more clearer now and the continuous variables are more normally distributed, but still slightly skewed.

Checking for multicollinearity again

In [73]:
```
## Get the correlation matrix for our model_df (without the target)
corr = df_no_outliers.drop('price',axis=1).corr()
corr.round(2)
```

Out[73]:

	bedrooms	bathrooms	sqft_lot	floors	condition	grade	sqft_above	zipcode	la
bedrooms	1.00	0.50	0.09	0.16	0.03	0.34	0.48	-0.16	-0.0
bathrooms	0.50	1.00	0.06	0.50	-0.14	0.62	0.63	-0.21	0.0
sqft_lot	0.09	0.06	1.00	-0.09	0.04	0.13	0.21	-0.18	-0.0
floors	0.16	0.50	-0.09	1.00	-0.28	0.45	0.53	-0.06	0.0
condition	0.03	-0.14	0.04	-0.28	1.00	-0.17	-0.17	-0.01	-0.0
grade	0.34	0.62	0.13	0.45	-0.17	1.00	0.71	-0.18	0.1
sqft_above	0.48	0.63	0.21	0.53	-0.17	0.71	1.00	-0.27	-0.0
zipcode	-0.16	-0.21	-0.18	-0.06	-0.01	-0.18	-0.27	1.00	0.2
lat	-0.03	0.00	-0.08	0.04	-0.01	0.11	-0.02	0.28	1.0

	bedrooms	bathrooms	sqft_lot	floors	condition	grade	sqft_above	zipcode	la
long	0.17	0.26	0.28	0.14	-0.09	0.23	0.40	-0.58	-0.1
sqft_living15	0.40	0.53	0.26	0.26	-0.10	0.68	0.71	-0.29	0.0
basement	0.15	0.15	-0.02	-0.27	0.13	0.03	-0.26	0.16	0.1
age	-0.16	-0.54	-0.03	-0.50	0.37	-0.46	-0.45	0.34	0.1

In [74]:
```
abs(corr.round(2)) > 0.75
```

Out[74]:

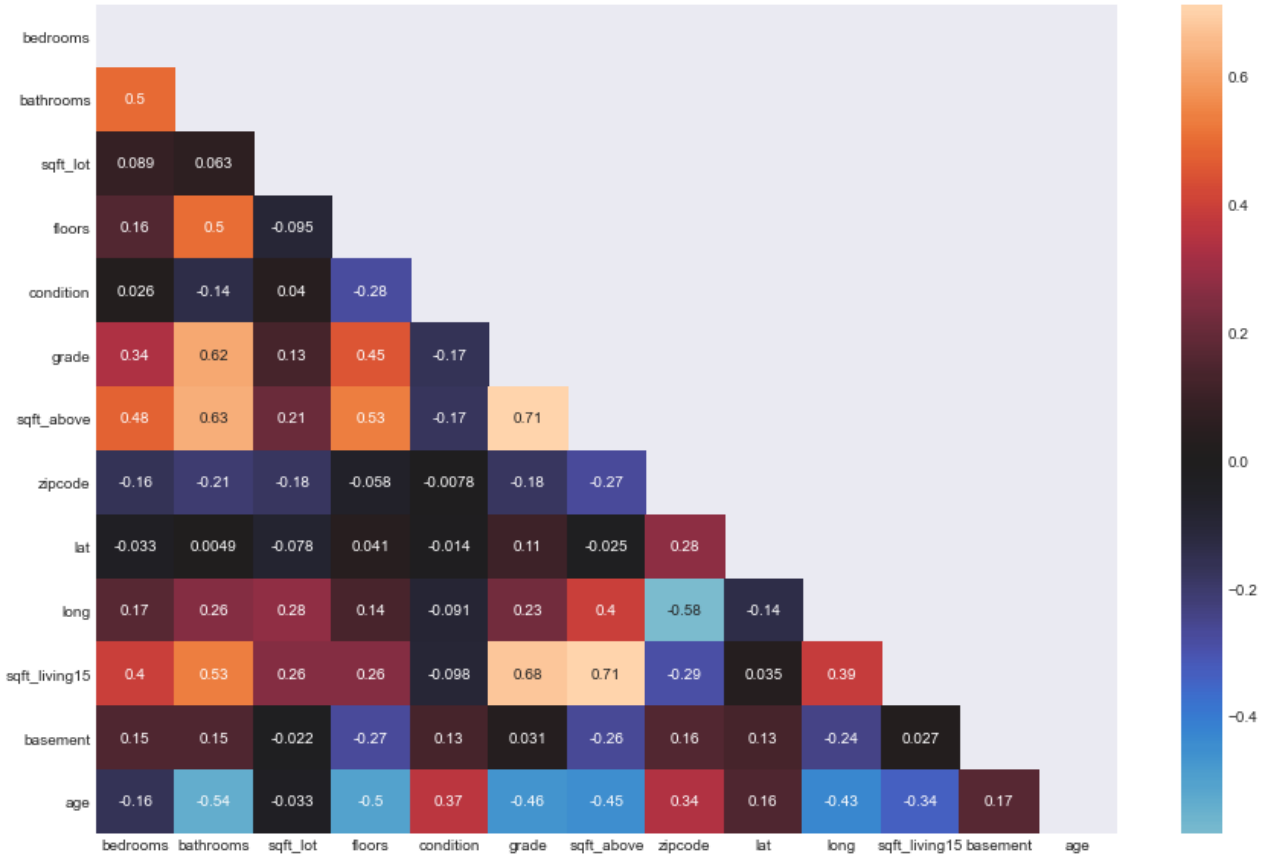
	bedrooms	bathrooms	sqft_lot	floors	condition	grade	sqft_above	zipcode	la
bedrooms	True	False	False	False	False	False	False	False	Fals
bathrooms	False	True	False	False	False	False	False	False	Fals
sqft_lot	False	False	True	False	False	False	False	False	Fals
floors	False	False	False	True	False	False	False	False	Fals
condition	False	False	False	False	True	False	False	False	Fals
grade	False	False	False	False	False	True	False	False	Fals
sqft_above	False	False	False	False	False	False	True	False	Fals
zipcode	False	False	False	False	False	False	False	True	Fals
lat	False	False	False	False	False	False	False	False	Tru
long	False	False	False	False	False	False	False	False	Fals
sqft_living15	False	False	False	False	False	False	False	False	Fals
basement	False	False	False	False	False	False	False	False	Fals
age	False	False	False	False	False	False	False	False	Fals

In [75]:
```
# stacks the row:column pairs into a multindex
# reset the index to set the multindex to seperate columns
# sort values. 0 is the column automatically generated by the stacking
df_corr = corr.abs().stack().reset_index().sort_values(0, ascending=False)

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

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

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

# rename correlation column as cc rather than 0
df_corr.columns = ['cc']
df_corr.drop_duplicates(inplace=True)

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

Out[75]:

	cc
pairs	

In [76]:
```
fig, ax = plt.subplots(figsize=(15,10))

sns.heatmap(corr,ax=ax, center=0,annot=True,mask=np.triu(np.ones_like(corr, dtyp
```



Looks much better now with outliers and columns trimmed.

Variance Inflation Factor

In [77]:
```
no_outlier_cols = list(df_no_outliers)
no_outlier_cols.remove('price')
no_outlier_cols.remove('condition')
no_outlier_cols.remove('grade')
no_outlier_cols.remove('zipcode')
X = df_no_outliers[no_outlier_cols]
vif = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
list(zip(no_outlier_cols, vif))
```

Out[77]:
```
[('bedrooms', 25.28541109934226),
 ('bathrooms', 27.518417279134713),
 ('sqft_lot', 1.8499121447213551),
 ('floors', 16.542890722160347),
 ('sqft_above', 28.03668664381894),
 ('lat', 118211.82978832249),
 ('long', 117587.1885724088),
 ('sqft_living15', 26.861119382963764),
 ('basement', 2.6475073164886216),
 ('age', 6.612972935123147)]
```

All of the features besides sqft_lot, basement and age have large VIFs.

In [78]:
```
price_corrs2 = df_no_outliers.corr()['price'].map(abs).sort_values(ascending=Fal
price_corrs2
```

Out[78]:
```
price           1.00
grade           0.65
sqft_living15   0.56
sqft_above      0.52
bathrooms       0.46
lat             0.41
bedrooms        0.31
floors          0.26
basement        0.19
sqft_lot        0.09
condition       0.05
long            0.04
age             0.03
zipcode         0.02
Name: price, dtype: float64
```

Now it appears the top 3 features most correlated with price are grade, sqft, and bathrooms.

Model 5

Feature Engineering

In [101…
```
# bathrooms per sqft and bedroom
df_bath_m5 = df_no_outliers.copy()
df_bath_m5['sqft_per_bath'] = df_bath_m5['sqft_above'] / df_bath_m5['bathrooms']
df_bath_m5
```

Out[101…

id	price	bedrooms	bathrooms	sqft_lot	floors	condition	grade	sqft_above	z
7129300520	221900.00	3	1.00	5650	1.00	3	7	1180	
6414100192	538000.00	3	2.25	7242	2.00	3	7	2170	
5631500400	180000.00	2	1.00	10000	1.00	3	6	770	
2487200875	604000.00	4	3.00	5000	1.00	5	7	1050	
1954400510	510000.00	3	2.00	8080	1.00	3	8	1680	
…	…	…	…	…	…	…	…	…	…
263000018	360000.00	3	2.50	1131	3.00	3	8	1530	
6600060120	400000.00	4	2.50	5813	2.00	3	8	2310	
1523300141	402101.00	2	0.75	1350	2.00	3	7	1020	
291310100	400000.00	3	2.50	2388	2.00	3	8	1600	
1523300157	325000.00	2	0.75	1076	2.00	3	7	1020	

20174 rows × 15 columns

In [104…
```
model5 = make_model(df_bath_m5,cat_cols=['zipcode','grade','condition','basement
evaluation(model5,df_bath_m5)
```

Model:	OLS	Adj. R-squared:	0.820
Dependent Variable:	price	AIC:	523833.5860

Date:	2021-11-04 13:08	BIC:	524545.6794
No. Observations:	20174	Log-Likelihood:	-2.6183e+05
Df Model:	89	F-statistic:	1033.
Df Residuals:	20084	Prob (F-statistic):	0.00
R-squared:	0.821	Scale:	1.1025e+10

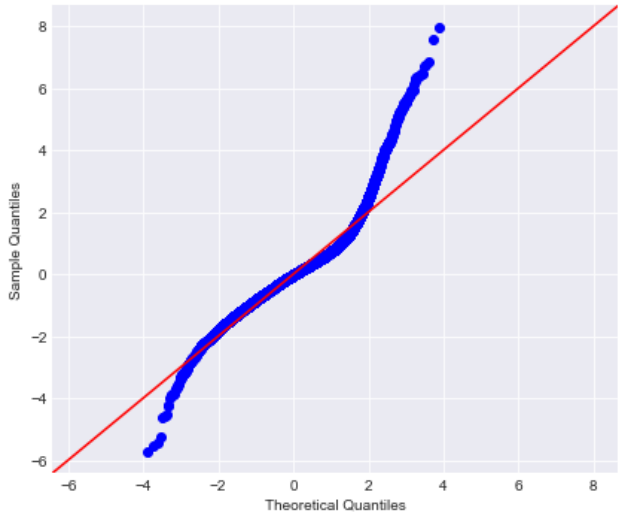
	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
Intercept	-27964463.8238	5233961.3316	-5.3429	0.0000	-38223457.7891	-17705469.8584
C(condition)[T.3]	30480.0521	8680.9569	3.5111	0.0004	13464.6639	47495.4404
C(condition)[T.4]	52763.6972	8716.9796	6.0530	0.0000	35677.7014	69849.6930
C(condition)[T.5]	91082.0123	9011.1532	10.1077	0.0000	73419.4122	108744.6124
C(grade)[T.6]	4550.7704	7699.3770	0.5911	0.5545	-10540.6408	19642.1816
C(grade)[T.7]	25599.9700	7611.6801	3.3632	0.0008	10680.4521	40519.4880
C(grade)[T.8]	72835.3933	7954.1154	9.1569	0.0000	57244.6741	88426.1125
C(grade)[T.9]	168368.1527	8527.1567	19.7449	0.0000	151654.2255	185082.0799
C(grade)[T.10]	254193.1582	9378.8907	27.1027	0.0000	235809.7622	272576.5541
C(grade)[T.11]	361019.4835	12355.6105	29.2191	0.0000	336801.4724	385237.4946
C(zipcode)[T.98002]	24322.3409	9692.7165	2.5093	0.0121	5323.8207	43320.8612
C(zipcode)[T.98003]	-9990.7149	8557.4275	-1.1675	0.2430	-26763.9754	6782.5455
C(zipcode)[T.98004]	592009.1702	16788.3916	35.2630	0.0000	559102.5442	624915.7962
C(zipcode)[T.98005]	311581.6815	17724.1661	17.5795	0.0000	276840.8605	346322.5024
C(zipcode)[T.98006]	284311.2664	14906.8617	19.0725	0.0000	255092.5934	313529.9393
C(zipcode)[T.98007]	251200.9712	18399.6306	13.6525	0.0000	215136.1845	287265.7580
C(zipcode)[T.98008]	264316.3952	17890.3577	14.7742	0.0000	229249.8252	299382.9652
C(zipcode)[T.98010]	129104.8887	16969.3287	7.6081	0.0000	95843.6111	162366.1664
C(zipcode)[T.98011]	118433.1277	22416.3474	5.2833	0.0000	74495.2462	162371.0092
C(zipcode)[T.98014]	161387.9491	26625.2283	6.0615	0.0000	109200.3154	213575.5828
C(zipcode)[T.98019]	128727.5431	26167.0079	4.9195	0.0000	77438.0591	180017.0271

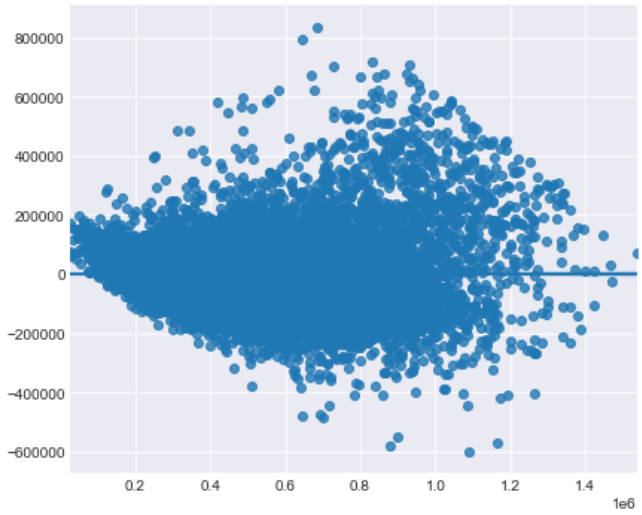
C(zipcode) [T.98022]	69554.1530	14923.5671	4.6607	0.0000	40302.7362	98805.5697
C(zipcode) [T.98023]	-45392.8252	8274.2864	-5.4860	0.0000	-61611.1060	-29174.5445
C(zipcode) [T.98024]	186532.1624	25330.0221	7.3641	0.0000	136883.2392	236181.0856
C(zipcode) [T.98027]	206653.6823	16184.4781	12.7686	0.0000	174930.7763	238376.5883
C(zipcode) [T.98028]	111508.9738	21596.9549	5.1632	0.0000	69177.1689	153840.7787
C(zipcode) [T.98029]	253380.1574	18617.6429	13.6097	0.0000	216888.0486	289872.2663
C(zipcode) [T.98030]	15104.3770	9783.3911	1.5439	0.1226	-4071.8728	34280.6267
C(zipcode) [T.98031]	22998.6121	10263.3348	2.2409	0.0250	2881.6331	43115.5910
C(zipcode) [T.98032]	-5227.9381	11328.0845	-0.4615	0.6444	-27431.9139	16976.0377
C(zipcode) [T.98033]	339608.1044	18879.3685	17.9883	0.0000	302602.9920	376613.2168
C(zipcode) [T.98034]	175535.5022	20120.6582	8.7241	0.0000	136097.3600	214973.6445
C(zipcode) [T.98038]	77924.6850	12731.8136	6.1205	0.0000	52969.2849	102880.0851
C(zipcode) [T.98039]	797678.8803	29821.2438	26.7487	0.0000	739226.7938	856130.9668
C(zipcode) [T.98040]	462448.4747	14649.8284	31.5668	0.0000	433733.6082	491163.3412
C(zipcode) [T.98042]	32266.2097	10183.3360	3.1685	0.0015	12306.0350	52226.3845
C(zipcode) [T.98045]	167660.7853	37589.0151	4.4604	0.0000	93983.2293	241338.3412
C(zipcode) [T.98052]	244255.4636	19728.5607	12.3808	0.0000	205585.8647	282925.0625
C(zipcode) [T.98053]	250206.9010	22031.8227	11.3566	0.0000	207022.7195	293391.0826
C(zipcode) [T.98055]	45228.9431	11408.5265	3.9645	0.0001	22867.2945	67590.5918
C(zipcode) [T.98056]	101308.9148	12658.8440	8.0030	0.0000	76496.5412	126121.2884
C(zipcode) [T.98058]	43659.7257	11304.2389	3.8622	0.0001	21502.4893	65816.9622
C(zipcode) [T.98059]	102926.5059	12774.7120	8.0571	0.0000	77887.0214	127965.9903
C(zipcode) [T.98065]	186751.3931	22275.7951	8.3836	0.0000	143089.0056	230413.7806
C(zipcode)	84335.7965	16292.7656	5.1763	0.0000	52400.6382	116270.9549

[T.98070]						
C(zipcode) [T.98072]	156142.9851	22789.9887	6.8514	0.0000	111472.7360	200813.2341
C(zipcode) [T.98074]	204811.9895	19407.8487	10.5530	0.0000	166771.0126	242852.9665
C(zipcode) [T.98075]	222655.7476	18933.5496	11.7599	0.0000	185544.4358	259767.0594
C(zipcode) [T.98077]	132949.8219	24302.9281	5.4705	0.0000	85314.0873	180585.5564
C(zipcode) [T.98092]	-8030.0165	9040.3580	-0.8882	0.3744	-25749.8604	9689.8274
C(zipcode) [T.98102]	396337.1280	18865.8415	21.0082	0.0000	359358.5296	433315.7264
C(zipcode) [T.98103]	299514.7900	17704.7546	16.9172	0.0000	264812.0172	334217.5627
C(zipcode) [T.98105]	399448.0888	18282.1518	21.8491	0.0000	363613.5702	435282.6074
C(zipcode) [T.98106]	94017.7300	13024.3581	7.2186	0.0000	68488.9187	119546.5413
C(zipcode) [T.98107]	285756.5279	18201.4597	15.6996	0.0000	250080.1724	321432.8834
C(zipcode) [T.98108]	96028.1135	14248.4647	6.7395	0.0000	68099.9527	123956.2743
C(zipcode) [T.98109]	429646.6550	18707.2258	22.9669	0.0000	392978.9565	466314.3536
C(zipcode) [T.98112]	476280.8422	16854.6776	28.2581	0.0000	443244.2902	509317.3942
C(zipcode) [T.98115]	307656.1554	18056.5262	17.0385	0.0000	272263.8815	343048.4294
C(zipcode) [T.98116]	278643.0285	14623.2529	19.0548	0.0000	249980.2521	307305.8050
C(zipcode) [T.98117]	276684.0660	18232.3883	15.1754	0.0000	240947.0879	312421.0442
C(zipcode) [T.98118]	154413.2937	12768.1135	12.0937	0.0000	129386.7429	179439.8445
C(zipcode) [T.98119]	408645.9945	17725.4289	23.0542	0.0000	373902.6983	443389.2906
C(zipcode) [T.98122]	297740.4438	15825.0584	18.8145	0.0000	266722.0298	328758.8577
C(zipcode) [T.98125]	175497.0944	19509.6767	8.9954	0.0000	137256.5262	213737.6627
C(zipcode) [T.98126]	168200.2740	13366.4132	12.5838	0.0000	142001.0066	194399.5414
C(zipcode) [T.98133]	118595.3758	20106.9884	5.8982	0.0000	79184.0276	158006.7239
C(zipcode) [T.98136]	233292.0266	13733.3124	16.9873	0.0000	206373.6066	260210.4465

C(zipcode) [T.98144]	244506.2955	14693.2932	16.6407	0.0000	215706.2344	273306.3565
C(zipcode) [T.98146]	95428.7725	12199.6691	7.8222	0.0000	71516.4193	119341.1257
C(zipcode) [T.98148]	37129.1874	16301.6576	2.2776	0.0228	5176.6000	69081.7748
C(zipcode) [T.98155]	108894.5901	20996.0874	5.1864	0.0000	67740.5349	150048.6453
C(zipcode) [T.98166]	85000.8513	11157.9155	7.6180	0.0000	63130.4207	106871.2818
C(zipcode) [T.98168]	32775.7342	11702.7937	2.8007	0.0051	9837.2978	55714.1707
C(zipcode) [T.98177]	194564.4661	20973.2597	9.2768	0.0000	153455.1551	235673.7771
C(zipcode) [T.98178]	53911.9973	12225.8263	4.4097	0.0000	29948.3739	77875.6206
C(zipcode) [T.98188]	29458.3241	12270.0519	2.4008	0.0164	5408.0149	53508.6332
C(zipcode) [T.98198]	25855.4264	9330.1422	2.7712	0.0056	7567.5816	44143.2712
C(zipcode) [T.98199]	336306.2104	17341.3751	19.3933	0.0000	302315.6913	370296.7296
C(basement) [T.1]	50925.3515	2069.1963	24.6112	0.0000	46869.5569	54981.1460
bedrooms	205.7359	1130.2928	0.1820	0.8556	-2009.7309	2421.2027
bathrooms	61346.0739	3735.7147	16.4215	0.0000	54023.7664	68668.3814
sqft_lot	0.6047	0.0672	8.9951	0.0000	0.4729	0.7364
floors	-28613.4768	2183.6316	-13.1036	0.0000	-32893.5740	-24333.3795
sqft_above	83.1924	4.1705	19.9479	0.0000	75.0179	91.3669
lat	61050.5096	44276.3252	1.3789	0.1680	-25734.7232	147835.7425
long	-203291.8462	41604.3646	-4.8863	0.0000	-284839.8168	-121743.8756
sqft_living15	45.2884	2.1081	21.4835	0.0000	41.1564	49.4204
age	734.0585	42.8704	17.1227	0.0000	650.0289	818.0880
sqft_per_bath	59.4098	7.6534	7.7626	0.0000	44.4086	74.4111

Omnibus:	5127.428	Durbin-Watson:	1.976
Prob(Omnibus):	0.000	Jarque-Bera (JB):	29314.896
Skew:	1.101	Prob(JB):	0.000
Kurtosis:	8.480	Condition No.:	119444398





No significant changes.

iNTERPRET

In [106…
```
coeffs = model5.params.sort_values().to_frame('coeffs')
coeffs['abs'] = coeffs['coeffs'].abs()
coeffs.sort_values('abs', ascending=False, inplace=True)
```

In [107…
```
coeffs[~coeffs.index.str.startswith('C(zipcode)')]
```

Out[107…

	coeffs	abs
Intercept	-27964463.82	27964463.82
C(grade)[T.11]	361019.48	361019.48
C(grade)[T.10]	254193.16	254193.16
long	-203291.85	203291.85
C(grade)[T.9]	168368.15	168368.15
C(condition)[T.5]	91082.01	91082.01
C(grade)[T.8]	72835.39	72835.39
bathrooms	61346.07	61346.07
lat	61050.51	61050.51
C(condition)[T.4]	52763.70	52763.70
C(basement)[T.1]	50925.35	50925.35
C(condition)[T.3]	30480.05	30480.05
floors	-28613.48	28613.48
C(grade)[T.7]	25599.97	25599.97
C(grade)[T.6]	4550.77	4550.77
age	734.06	734.06
bedrooms	205.74	205.74
sqft_above	83.19	83.19

	coeffs	abs
sqft_per_bath	59.41	59.41
sqft_living15	45.29	45.29
sqft_lot	0.60	0.60

The following features together explain 82% of the variation in house sale price:

- Number of bathrooms
- Living space (sqft) above ground
- Grade
- Condition
- Size of houses in the neighborhood
- Age of house
- Basement
- Zipcode
- Location (Latitude and Longitude)

All the numerical features (aside form location) have a positive relationship with the price of the home. We can interpret this as for every unit of increase, the dependant variable (price) will increase by the predicting variables coefficient.

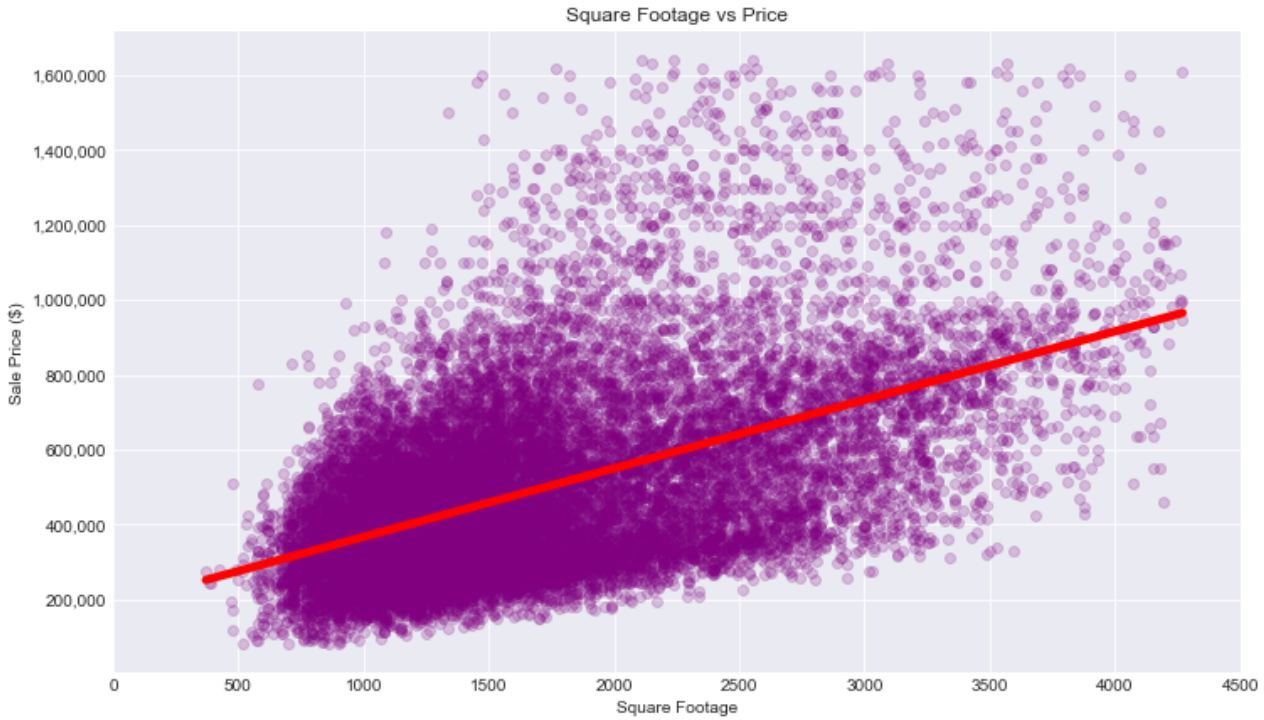
It is critical to understand that location has a great affect on price. This can best be understood by the difference in coefficients in zipcodes. Some zipcodes can greatly increase price while others may have a negative affect on a home's price.

Findings Visualized

Square Footage Above Ground

In [230…

```
fig, ax = plt.subplots(figsize=(12,7))
sns.regplot(x='sqft_above', y='price', data=df_no_outliers,line_kws={'color':'re
ax.set(title= 'Square Footage vs Price', xlabel= 'Square Footage', ylabel='Sale
ax.xaxis.set_ticks(range(0,5000,500))
ax.yaxis.set_major_formatter(FuncFormatter(lambda x, p: format(int(x), ',')))
```



The scatter plot above displays the positive relationship between above ground square footage and sale price. As the square footage of a home increases, so does its price.

Basements

In [109…
```
df_basement = df_no_outliers.loc[:,['price','basement']]
df_basement_price = df_basement.groupby('basement')['price'].mean()
```

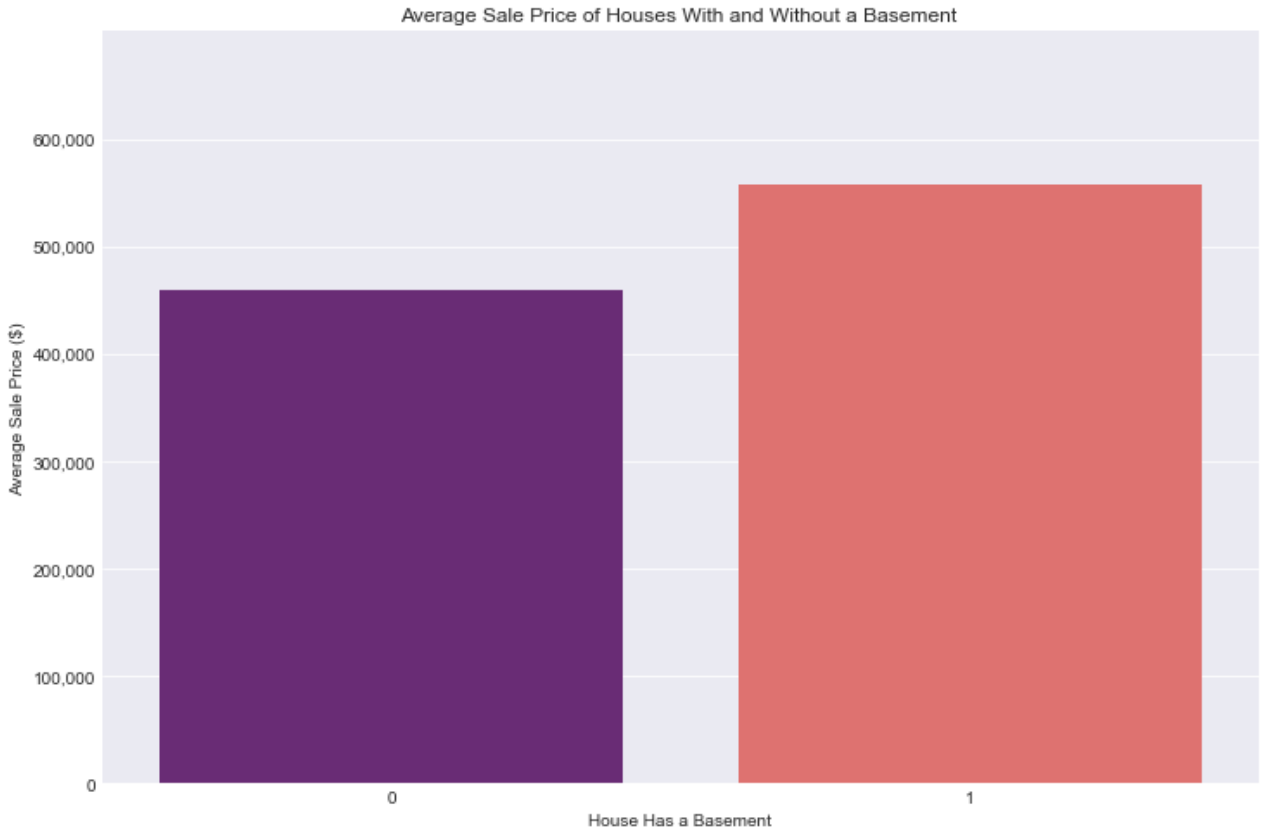
In [110…
```
df_basement_price = df_basement_price.to_frame().reset_index()
```

In [111…
```
df_basement_price
```

Out[111…

	basement	price
0	0	459560.88
1	1	556647.39

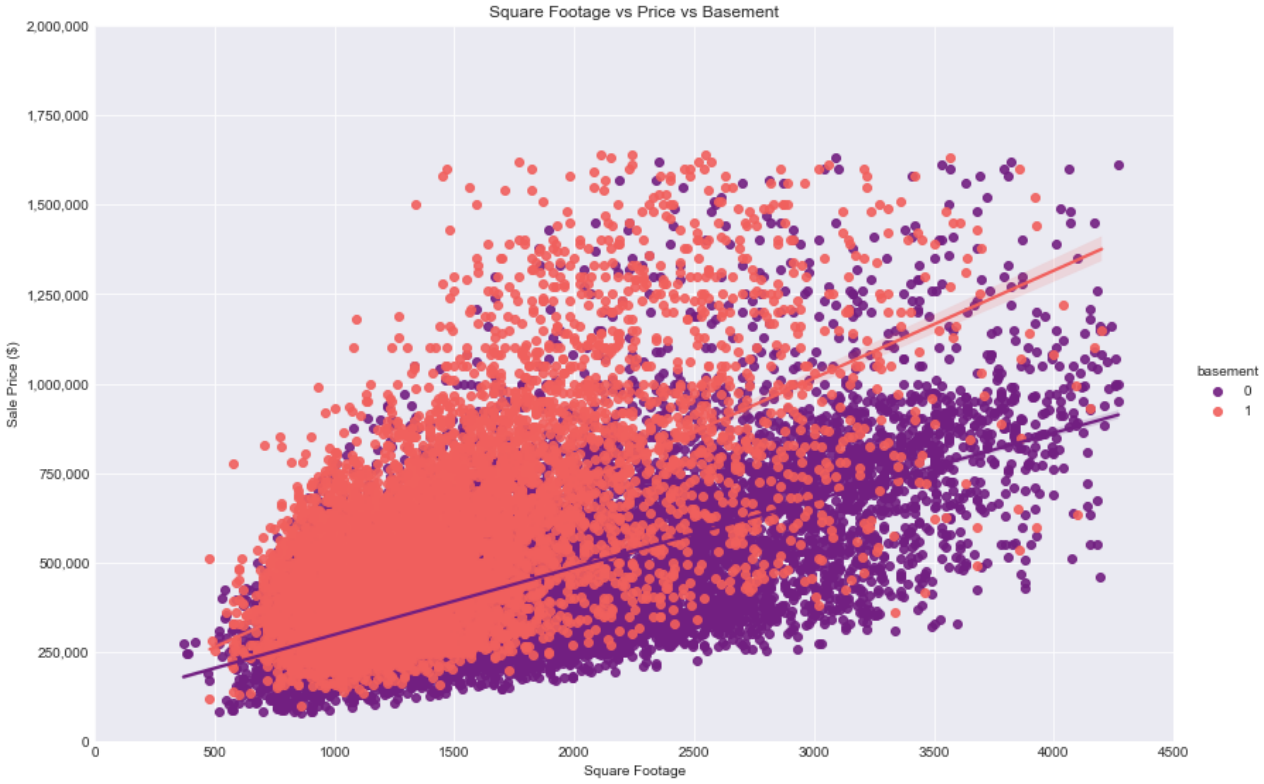
In [121…
```
fig, ax = plt.subplots(figsize=(12,8))
sns.barplot(data =df_basement_price, x='basement', y= 'price',palette='magma')
ax.set_xlabel('House Has a Basement')
ax.set_ylabel('Average Sale Price ($)')
ax.set_title('Average Sale Price of Houses With and Without a Basement ')
ax.set_ylim(0,700000)
ax.yaxis.set_major_formatter(FuncFormatter(lambda x, p: format(int(x), ',')))
ax.set_yticks(range(0,700000,100000));
```



Houses with a basement on average have a higher sale price ($556,647) compared to those without (\$459,560).

In [214…

```
sns.lmplot(x='sqft_above', y='price', data=df_model6, hue='basement',
           aspect=2, scatter_kws=dict(alpha=0.9), palette="magma")
ax = plt.gca()
ax.set_xlabel('Square Footage')
ax.set_ylabel('Sale Price ($)')
ax.set_title("Square Footage vs Price vs Basement")
ax.yaxis.set_major_formatter(FuncFormatter(lambda x, p: format(int(x), ',')))
ax.xaxis.set_ticks(range(0,5000,500))
ax.set_ylim(0, 2000000)
plt.gcf().set_size_inches(12,8);
```



In the plot above it is evident that houses with a similar above ground square footage but have a basement tend to have a higher sale price compared to similarly sized homes without a basement.

Grades

In [172…
```
df_no_outliers['grade'].value_counts()
```

Out[172…
```
7     8714
8     5858
9     2393
6     1977
10     862
5      212
11     158
Name: grade, dtype: int64
```

In [173…
```
df_grade = df_no_outliers.loc[:,['price','grade']]
df_grade
```

Out[173…

id	price	grade
7129300520	221900.00	7
6414100192	538000.00	7
5631500400	180000.00	6
2487200875	604000.00	7
1954400510	510000.00	8
...	...	...

id	price	grade
263000018	360000.00	8
6600060120	400000.00	8
1523300141	402101.00	7
291310100	400000.00	8
1523300157	325000.00	7

20174 rows × 2 columns

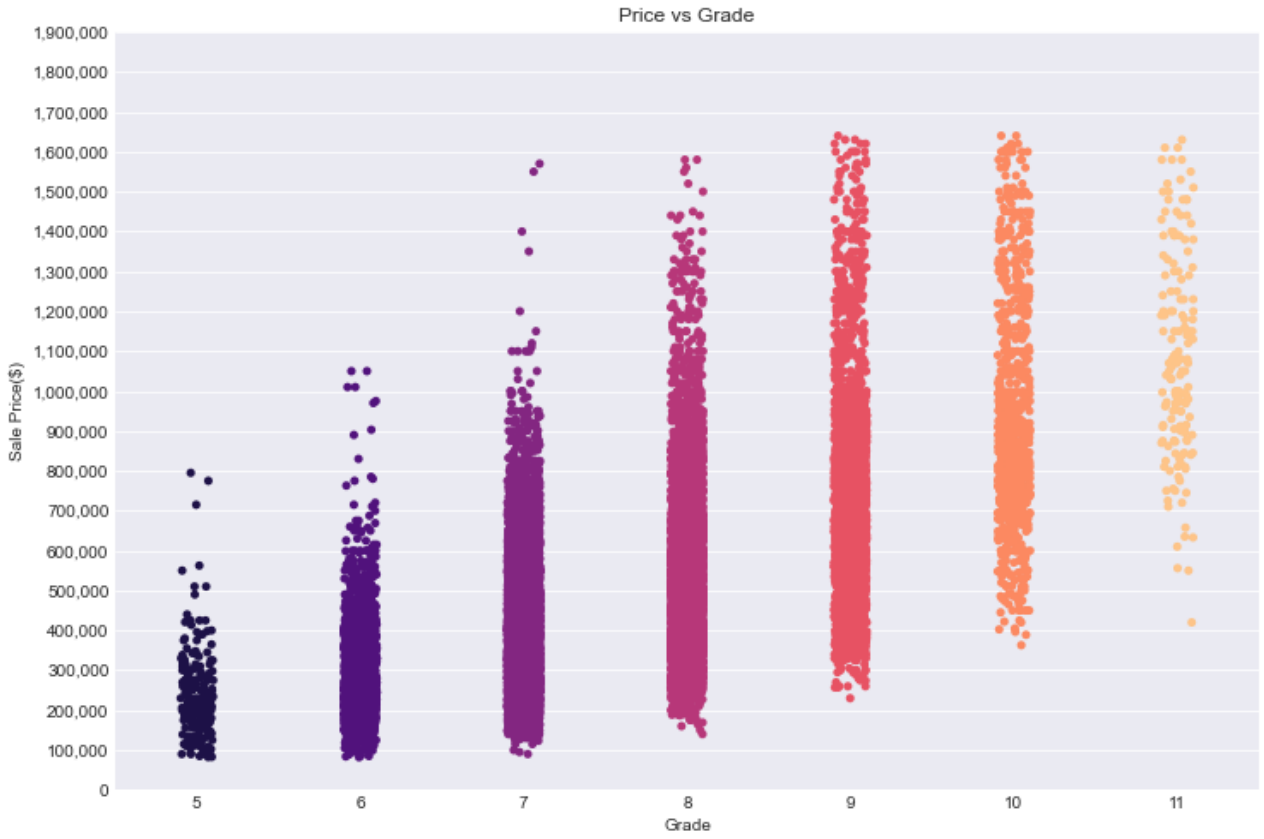
In [174…
```
mean_price_per_grade = df_grade.groupby('grade')['price'].mean().reset_index()
mean_price_per_grade
```

Out[174…

	grade	price
0	5	242883.33
1	6	299883.79
2	7	401311.94
3	8	536465.48
4	9	741211.06
5	10	922544.18
6	11	1085542.78

The average sale prices by grade show that it is very clearly evident that higher grade leads to a higher sale price.

In [191…
```
sns.catplot(data = df_grade_sqft, x='grade', y='price', aspect=1.5,palette='magm
ax = plt.gca()
fig = plt.gcf()
ax.set_yticks(range(0,2000000,100000))
ax.yaxis.set_major_formatter(FuncFormatter(lambda x, p: format(int(x), ',')))
ax.set_ylabel('Sale Price($)')
fig.set_size_inches(10, 7)
ax.set_xlabel('Grade')
ax.set_title('Price vs Grade');
```



In [155…
```
df_grade_sqft = pd.concat([df_grade, pd.DataFrame(df_no_outliers['sqft_above'])]
df_grade_sqft
```

Out[155…

id	price	grade	sqft_above
7129300520	221900.00	7	1180
6414100192	538000.00	7	2170
5631500400	180000.00	6	770
2487200875	604000.00	7	1050
1954400510	510000.00	8	1680
...	...	...	...
263000018	360000.00	8	1530
6600060120	400000.00	8	2310
1523300141	402101.00	7	1020
291310100	400000.00	8	1600
1523300157	325000.00	7	1020

20174 rows × 3 columns

Plotting grade by category displays the relationship between a home's grade and its sale price. Although there is an overlap which can be attributed to other features such as size and location, it is clearly evident that as grade increase price will trend upwards. The biggest increase can be seen in a jump from grade 7 (average grade) to grade 8 (just above average grade).

Bathrooms

In [176…
```
df_bath = df_no_outliers.loc[:,['price','bathrooms']]
df_bath
```

Out[176…

id	price	bathrooms
7129300520	221900.00	1.00
6414100192	538000.00	2.25
5631500400	180000.00	1.00
2487200875	604000.00	3.00
1954400510	510000.00	2.00
...	...	...
263000018	360000.00	2.50
6600060120	400000.00	2.50
1523300141	402101.00	0.75
291310100	400000.00	2.50
1523300157	325000.00	0.75

20174 rows × 2 columns

In [177…
```
df_bath['bathrooms'].value_counts()
```

Out[177…
```
2.50    5154
1.00    3741
1.75    2932
2.25    1974
2.00    1862
1.50    1412
2.75    1107
3.00     678
3.50     591
3.25     462
3.75     101
4.00      67
0.75      47
4.25      34
1.25       8
0.50       4
Name: bathrooms, dtype: int64
```

In [182…
```
df_bath_sqft = pd.concat([df_bath, pd.DataFrame(df_no_outliers['sqft_above'])],
df_bath_sqft
```

Out[182…

id	price	bathrooms	sqft_above
7129300520	221900.00	1.00	1180
6414100192	538000.00	2.25	2170

id	price	bathrooms	sqft_above
5631500400	180000.00	1.00	770
2487200875	604000.00	3.00	1050
1954400510	510000.00	2.00	1680
...	...	...	...
263000018	360000.00	2.50	1530
6600060120	400000.00	2.50	2310
1523300141	402101.00	0.75	1020
291310100	400000.00	2.50	1600
1523300157	325000.00	0.75	1020

20174 rows × 3 columns

In [183…
```
df_bath_sqft['full_baths'] = round(df_bath['bathrooms'],0)
df_bath_sqft = df_bath_sqft[df_bath_sqft['full_baths'] > 0]
df_bath_sqft
```

Out[183…

id	price	bathrooms	sqft_above	full_baths
7129300520	221900.00	1.00	1180	1.00
6414100192	538000.00	2.25	2170	2.00
5631500400	180000.00	1.00	770	1.00
2487200875	604000.00	3.00	1050	3.00
1954400510	510000.00	2.00	1680	2.00
...	...	...	...	...
263000018	360000.00	2.50	1530	2.00
6600060120	400000.00	2.50	2310	2.00
1523300141	402101.00	0.75	1020	1.00
291310100	400000.00	2.50	1600	2.00
1523300157	325000.00	0.75	1020	1.00

20170 rows × 4 columns

In [184…
```
mean_price_per_bath = df_bath_sqft.groupby('full_baths')['price'].mean().reset_i
mean_price_per_bath
```

Out[184…

	full_baths	price
0	1.00	348174.80
1	2.00	490042.26

	full_baths	price
2	3.00	667363.12
3	4.00	842453.93

In [187…
```
fig, ax = plt.subplots(figsize=(12,7))
sns.barplot(data = mean_price_per_bath, x='full_baths', y= 'price',palette='magm
ax.set_xlabel('Number of Bathrooms')
ax.set_ylabel('Average Sale Price ($)')
ax.set_title('Average Sale Price per Number of Bathrooms')
ax.set_ylim(0,1200000)
ax.yaxis.set_major_formatter(FuncFormatter(lambda x, p: format(int(x), ',')));
ax.set_yticks(range(0,1300000,100000));
```

In [215…
```
sns.lmplot(x='sqft_above', y='price', data=df_bath_sqft, hue='full_baths',
           aspect=2, scatter_kws=dict(alpha=0.9), palette="magma")
ax = plt.gca()
ax.set_xlabel('Square Footage')
ax.set_ylabel('Sale Price ($)')
ax.set_title("Square Footage vs Price vs Bathrooms")
ax.yaxis.set_major_formatter(FuncFormatter(lambda x, p: format(int(x), ',')))
ax.set_ylim(0, 2000000)
ax.xaxis.set_ticks(range(0,5000,500))
plt.gcf().set_size_inches(12,8);
```

CONCLUSIONS & RECOMMENDATIONS

Even though not all features of a home are controllable by the homeowner it is important that they are included in the model. This allows the model to reach its largest R-squared value, which can in turn explain the variance in the dependent variable that the independent variables explain collectively. From there we can take the largest coefficients of the features that can be changed (ie bedrooms, bathrooms, square footage, condition, etc.).

The results of the best performing model suggests that the three most salient renovations King County homeowners can do to increase the resale value of their homes are:

- Increasing grade
- Adding a bathroom
- Adding a basement

It's also worth noting increasing the square footage above ground can increase a home's value \$116 - \$95 per square foot.

Limitations & Next Steps

Given more time and information about what the homeowner's renovation budget would be, we would have wanted to analyze whether these top 3 parameters would truly be the most effective in bringing a net value increase since a renovation such as adding a basement to a home would be very costly and may not end up returning a net value increase. Additionally, the construction costs in the state of Washington may be higher than other states due to factors such as permitting, material costs, logistical challenges etc. which may effect the net value increase as well. Furthermore, having information about whether the homeowner is thinking about living in

the renovated house or renting it out would allow us to fine tune our analysis and bring more valuable insight.

Further Analysis

Multiple linear regression is a highly interpretable model so it serves as a great choice for problems such as predicting housing prices and determining which features affect price.

With more time and resource allocated to this project, the model could be improved by including renovation project budgets and job costs to get an accurate net value increase.

The dataset also does not include any information about how the home was marketed and or sold which could have had a significant impact in the sale price even though it is not a feature of the home.

And lastly, this model does not take into account any code or zoning regulations before determining which features should be renovated and some suggestions may not be possible, legal or is cost prohibitive.

APPENDIX

Recursive Feature Elimination

In [ ]:
```
lr_rfe = LinearRegression()
select = RFE(lr_rfe, n_features_to_select=3)
```

In [ ]:
```
df_scaled1 = df.copy()
df_scaled1
```

In [ ]:
```
ss = StandardScaler()
```

In [ ]:
```
df_z = pd.DataFrame(ss.fit_transform(df_scaled1),columns=df_cols)
df_z
```

In [ ]:
```
df_refined = df_z[(df_z[df_cols] > -3) & (df_z[df_cols] < 3)]

df_refined
```

In [ ]:
```
df_refined.isna().sum()
```

In [ ]:
```
df_refined2 = df_refined.dropna(axis=0)
df_refined2
```

In [ ]:
```
df_refined3 = pd.DataFrame(ss.inverse_transform(df_refined2),columns=df_cols)
df_refined3.describe()
```

In [ ]:
```
df_refined3.head(30)
```

In [ ]:
```
# ss = StandardScaler()
# ss.fit(df.drop('price', axis=1))
```

```
# df_scaled = pd.DataFrame(ss.transform(df.drop('price',axis=1)))
```

In [ ]:
```
select.fit(X=df_scaled,y=df['price'])
```

In [ ]:
```
select.support_
```

In [ ]:
```
df_scaled.columns[select.support_]
```

In [ ]:
```
select.ranking_
```

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

In [ ]:
```
# formula = 'price ~ grade + bathrooms + sqft_living'
# model = smf.ols(formula,df_norm).fit()
# model.summary2()
```

Feature Engineering

In [ ]:
```
model2 = make_model(ohe_cols,df_ohe)
evaluation(model2, df_ohe)
```

In [ ]:
```
df['sqft_living'].hist(bins='auto',);
```

In [ ]:
```
df['large_homes'] = df['sqft_living'] > 2000
```

In [ ]:
```
df.corr()['price']['large_homes']
```

In [ ]:
```
df['small_homes'] = df['sqft_living'] < 2000
```

In [ ]:
```
df.corr()['price']['small_homes']
```

In [ ]:
```
df['bathrooms'].hist(bins='auto');
```

In [ ]:
```
df['many_baths'] = df['bathrooms'] > 2
```

In [ ]:
```
df.corr()['price']['many_baths']
```

In [ ]:
```
df['few_baths'] = df['bathrooms'] < 2
```

In [ ]:
```
df.corr()['price']['few_baths']
```

In [ ]:
```
df['grade'].hist(bins=10);
```

In [ ]:
```
df['yr_built'].hist(bins='auto');
```

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

In [ ]:
```
df['new_homes'] = df['yr_built'] > 2000
```

In [ ]:
```
df.corr()['price']['new_homes']
```

In [ ]:
```
def evaluation(df, x, y='price'):

    fig, axes = plt.subplots(ncols=2,figsize=(12,5))

    sm.graphics.qqplot(model.resid, dist=stats.norm, line='45', ax=axes[0], fit=
    sns.regplot(model.predict(df), model.resid,ax=axes[1])
    plt.plot(model.predict(df[x_cols]), [0 for i in range(len(df))])


    axes[0].set(title= x+' vs price', xlabel= x, ylabel='price')
    axes[1].set(title= 'count of ' + x, xlabel= x)
    plt.tight_layout();
```

In [ ]:
```
evaluation(df, x='sqft_living', y='price')
```

In [ ]:
```
model.pvalues[model.pvalues > .05]
```

Model Refinement

In [ ]:
```
#Finding a cutoff point
for i in range(90, 99):
#     q = i / 100
    print('{} percentile: {}'.format(q, df['price'].quantile(q=q)))
```

In [ ]:
```
# subset = df[df['price'] < 1600000]
# print('Percent removed:',(len(df) - len(subset))/len(df))
# outcome = 'price'
# x_cols = ['bathrooms', 'sqft_living', 'grade']
# predictors = '+'.join(x_cols)
# formula = outcome + '~' + predictors
# model = smf.ols(formula=formula, data=subset).fit()
# model.summary()
```

In [ ]:
```
# outcome = 'price'
# x_cols = ['bathrooms', 'grade', 'sqft_living']
# predictors = '+'.join(x_cols)
# formula = outcome + '~' + predictors
# model = smf.ols(formula=formula, data=df).fit()
# model.summary()
```

Model Validation

In [ ]:
```
# X = df[['sqft_living','bathrooms']]
# y = df['price']
```

In [ ]:
```
# # Model Validation
# X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

In [ ]:
```
# X_train.head()
```

In [ ]:
```

```