# # King County Housing Predictions

**Mark Patterson, Matthew Zhang, Mel Friedman**

## ## Objective

```
M3 Consulting CO. is developing solutions to help average people
homes" and is working with
non-profit organizations to provide a service.
Recently, a family of 4 is looking to buy their first home in Kin
Consulting decides to analyze data
of people with median income to optimize home affordability. Addi
the impact of the technology boom
and rising home prices, the family of 4 is not sure where to look
-Predicting house prices under 500k (median)
-Identifying which parts of King County these houses exist
-"Good home" being specific statistically significant features
```

## ## Data Cleaning & EDA

```
The data used in this project is from the King County Housing Dat
```

In [1]:

```
# Import relevant and necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from scipy import stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats import diagnostic as diag
from statsmodels.stats.outliers_influence import variance_inflati

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_err
```

In [2]:

```
#Loading in the data set and setting style to Seaborn
df = pd.read_csv('data/kc_house_data.csv')
plt.style.use('seaborn')
```

In [3]:

```
# Review size and shape of the data (at this stage)
print(df.shape)
display(df.info())
display('-'*60)
display(df.isnull().any())
display(df.isnull().sum())
display('-'*60)
df.head()
```

```
(21597, 21)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
id               21597 non-null int64
date             21597 non-null object
price            21597 non-null float64
bedrooms         21597 non-null int64
bathrooms        21597 non-null float64
sqft_living      21597 non-null int64
sqft_lot         21597 non-null int64
floors           21597 non-null float64
waterfront       19221 non-null float64
view             21534 non-null float64
condition        21597 non-null int64
grade            21597 non-null int64
sqft_above       21597 non-null int64
sqft_basement    21597 non-null object
yr_built         21597 non-null int64
yr_renovated     17755 non-null float64
zipcode          21597 non-null int64
lat              21597 non-null float64
long             21597 non-null float64
sqft_living15    21597 non-null int64
sqft_lot15       21597 non-null int64
dtypes: float64(8), int64(11), object(2)
memory usage: 3.5+ MB


None

'------------------------------------------------------------'

id               False
date             False
price            False
bedrooms         False
bathrooms        False
sqft_living      False
sqft_lot         False
floors           False
waterfront        True
view              True
condition        False
```

```
grade            False
sqft_above       False
sqft_basement    False
yr_built         False
yr_renovated      True
zipcode          False
lat              False
long             False
sqft_living15    False
sqft_lot15       False
dtype: bool

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

'------------------------------------------------------------'
```

Out[3]:

| | id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors |
|---|---|---|---|---|---|---|---|---|
| **0** | 7129300520 | 10/13/2014 | 221900.0 | 3 | 1.00 | 1180 | 5650 | 1.0 |
| **1** | 6414100192 | 12/9/2014 | 538000.0 | 3 | 2.25 | 2570 | 7242 | 2.0 |
| **2** | 5631500400 | 2/25/2015 | 180000.0 | 2 | 1.00 | 770 | 10000 | 1.0 |
| **3** | 2487200875 | 12/9/2014 | 604000.0 | 4 | 3.00 | 1960 | 5000 | 1.0 |
| **4** | 1954400510 | 2/18/2015 | 510000.0 | 3 | 2.00 | 1680 | 8080 | 1.0 |

5 rows × 21 columns

In [4]:

```
#Identfied sqft_basement as an object type, so convert it to nume
df['sqft_basement'] = pd.to_numeric(df['sqft_basement'], errors='
display(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
id               21597 non-null int64
date             21597 non-null object
price            21597 non-null float64
bedrooms         21597 non-null int64
bathrooms        21597 non-null float64
sqft_living      21597 non-null int64
sqft_lot         21597 non-null int64
floors           21597 non-null float64
waterfront       19221 non-null float64
view             21534 non-null float64
condition        21597 non-null int64
grade            21597 non-null int64
sqft_above       21597 non-null int64
sqft_basement    21143 non-null float64
yr_built         21597 non-null int64
yr_renovated     17755 non-null float64
zipcode          21597 non-null int64
lat              21597 non-null float64
long             21597 non-null float64
sqft_living15    21597 non-null int64
sqft_lot15       21597 non-null int64
dtypes: float64(9), int64(11), object(1)
memory usage: 3.5+ MB


None
```

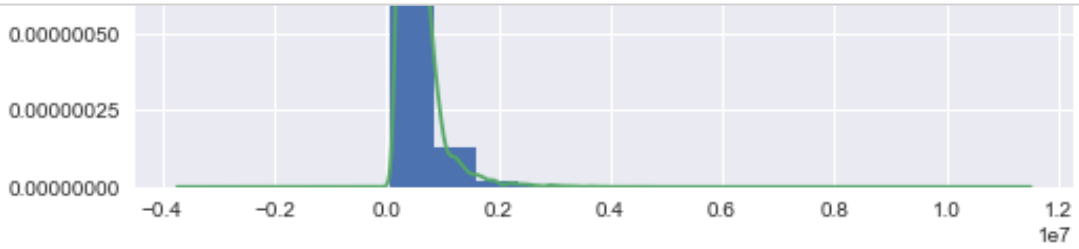
```
In order to explore the data, we first organized the data frame i
data frames to help understand
the visualizations.
```

In [5]:

```
# Seperate groups of features into seperate dataframes: counts, s
df_counts = df[['price','floors', 'bedrooms', 'bathrooms', 'water
df_condition = df[['price','condition', 'grade', 'yr_built', 'yr_
df_size = df[['price','sqft_lot', 'sqft_living', 'sqft_above', 's
```

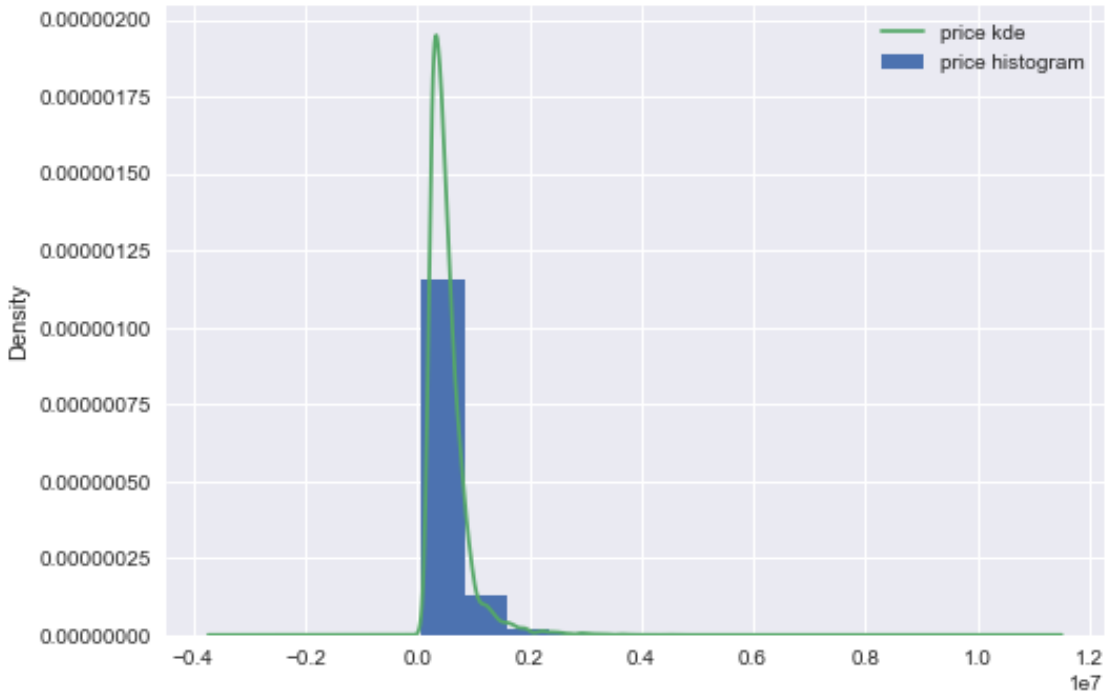
In [6]:

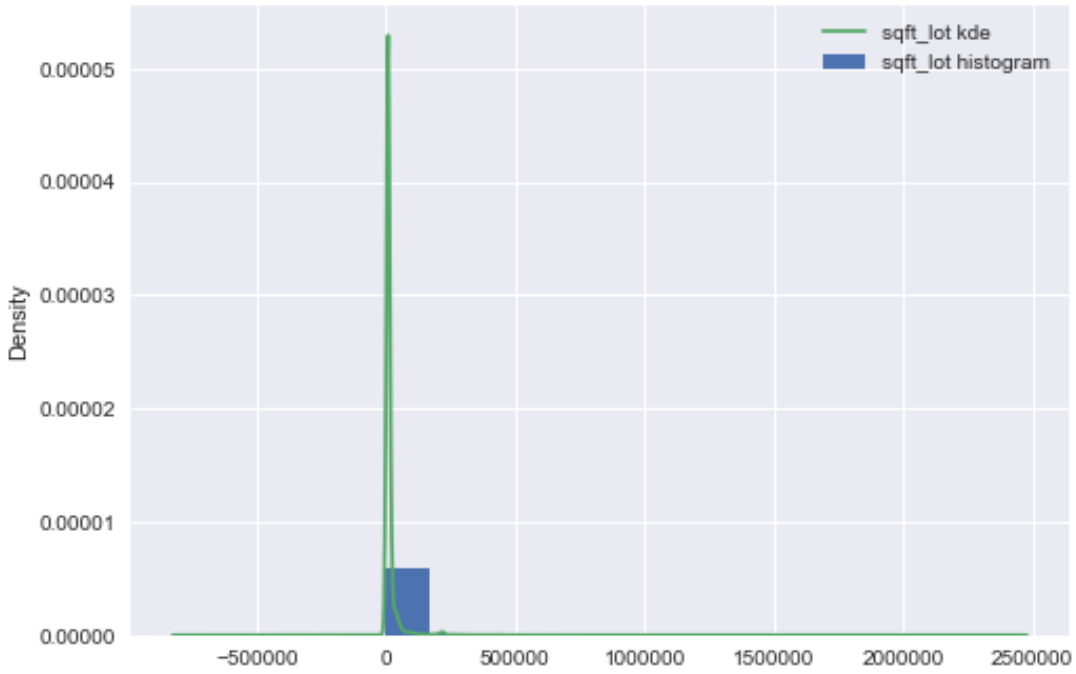
```
# Take a look at the distributions of some key features
for column in df_counts:
    df_counts[column].plot.hist(density=True, label = column+' hi
    df_counts[column].plot.kde(label =column+' kde')
    plt.legend()
    plt.show()
```

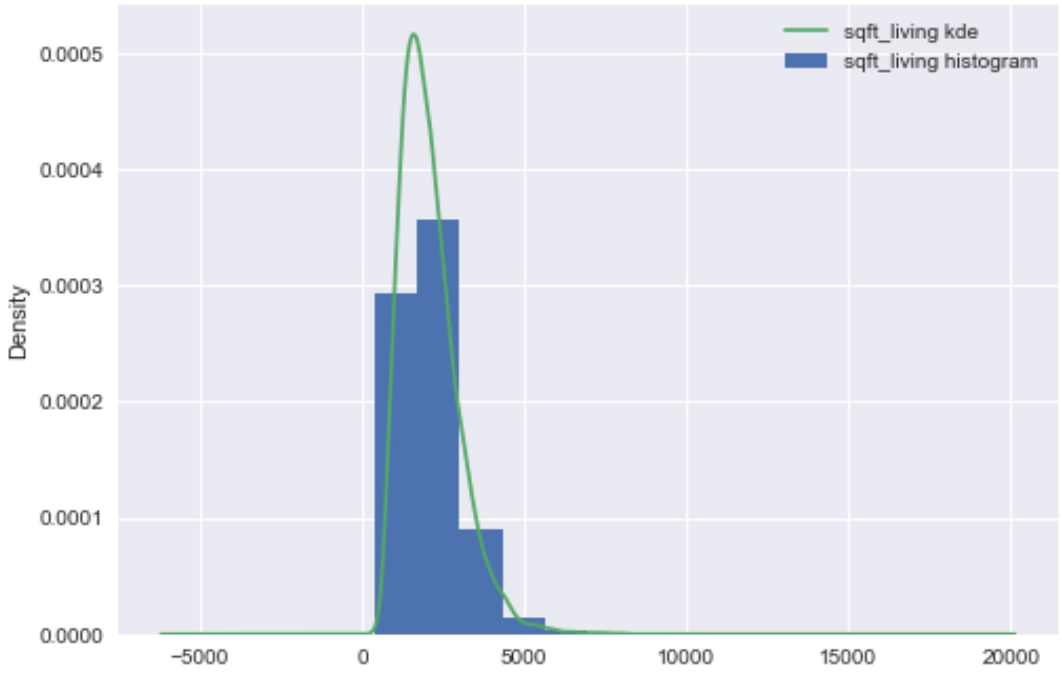


In [9]:

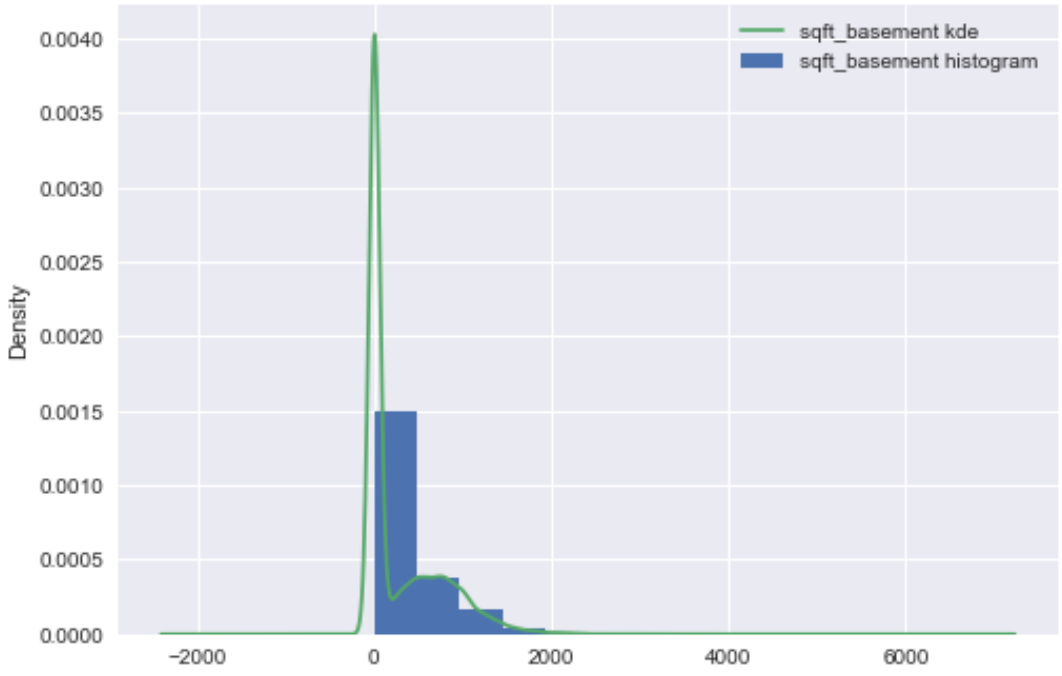
```
for column in df_size:
    df_size[column].plot.hist(density=True, label = column+' hist
    df_size[column].plot.kde(label =column+' kde')
    plt.legend()
    plt.show()
```

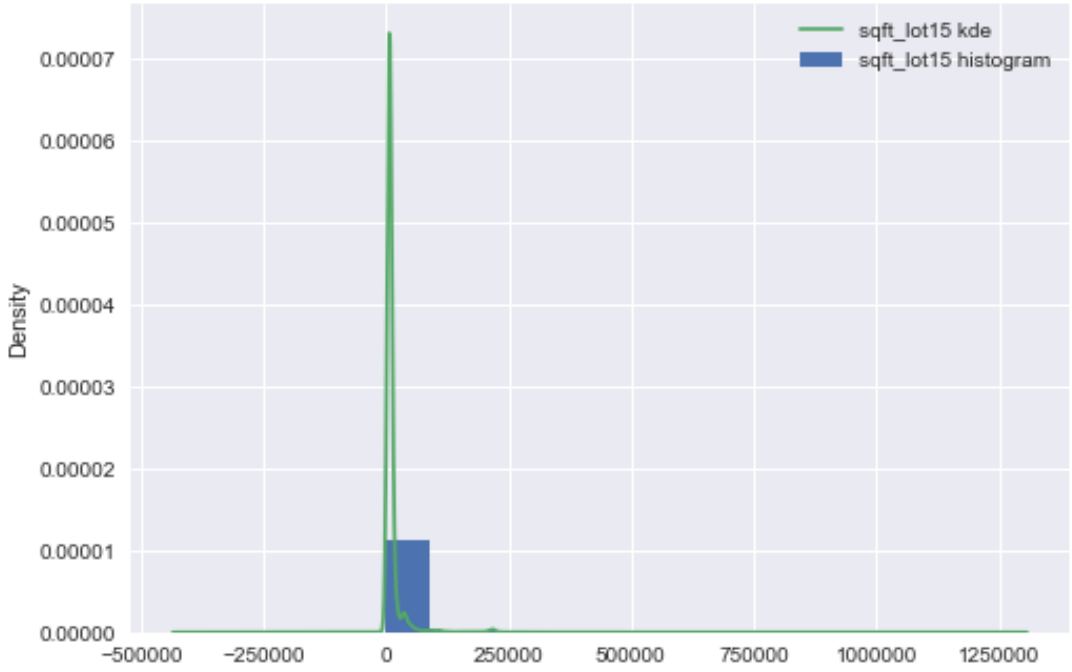



sqft_living kde
sqft_living histogram
Density
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
−5000
0
5000
10000
15000
20000

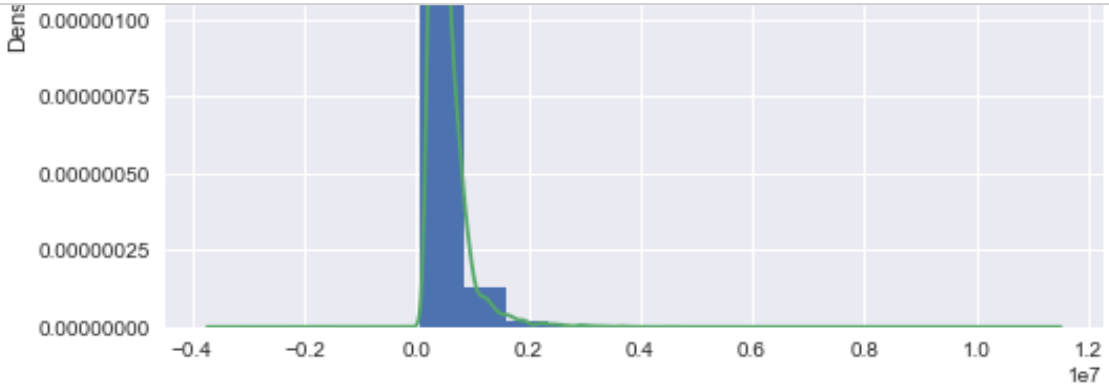

sqft_above kde
sqft_above histogram
Density
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
−5000
−2500
0
2500
5000
7500
10000
12500


sqft_basement kde
sqft_basement histogram
Density
0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
−2000
0
2000
4000
6000


sqft_lot15 kde
sqft_lot15 histogram
Density
0.00007
0.00006
0.00005
0.00004
0.00003
0.00002
0.00001
0.00000
−500000
−250000
0
250000
500000
750000
1000000
1250000


In [7]:

```
for column in df_condition:
    df_condition[column].plot.hist(density=True, label = column+'
    df_condition[column].plot.kde(label =column+' kde')
    plt.legend()
    plt.show()
```

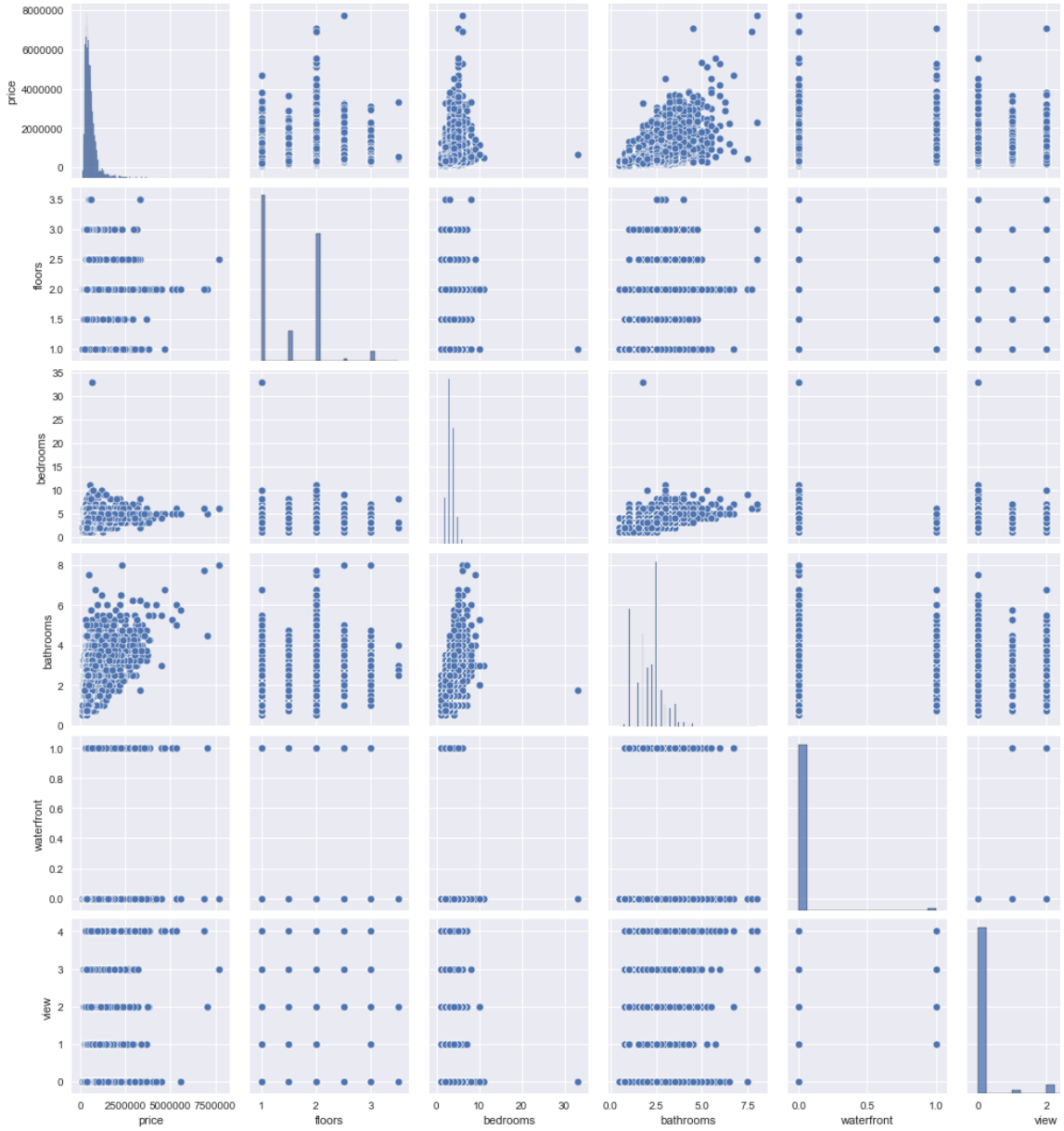


In [8]:

```
# Take a look at scatter plots and look for linear relationships
# sns.pairplot(data[['GrLivArea', 'OverallQual', 'TotRmsAbvGrd',
sns.pairplot(df_counts)
```

Out[8]:

```
<seaborn.axisgrid.PairGrid at 0x1a238afc88>
```



In [9]:

```
sns.pairplot(df_condition)
```

Out[9]:

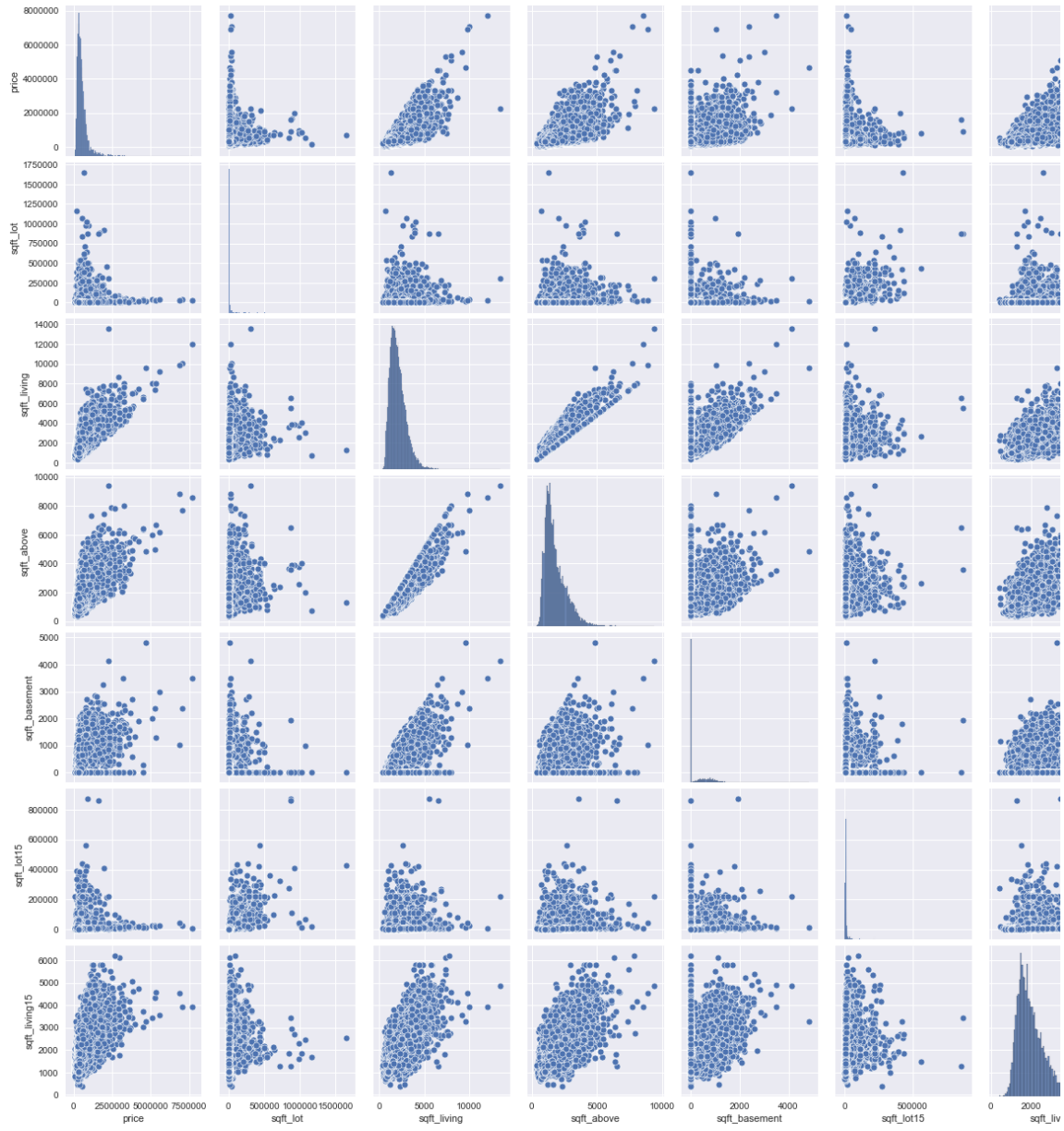
```
<seaborn.axisgrid.PairGrid at 0x1a262b3400>
```

In [11]:

```
sns.pairplot(df_size)
```

Out[11]:

```
<seaborn.axisgrid.PairGrid at 0x1a303907f0>
```



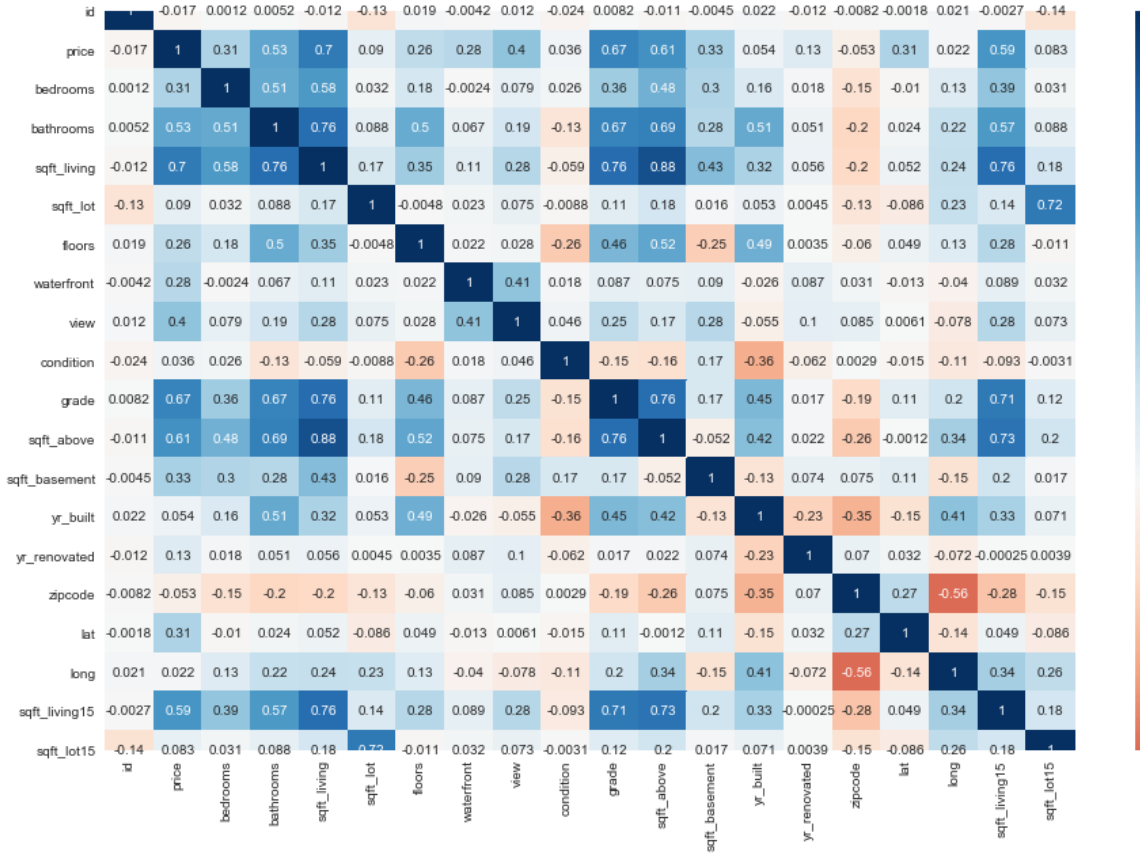
In [12]:

```
# Take a look at correlations between variables (multicollinearit
corr = df.corr()
display(corr)
plt.figure(figsize=(16,10))
sns.heatmap(corr, cmap='RdBu', annot=True, center=0)
#So we have a few collerated features
```

| | id | price | bedrooms | bathrooms | sqft_living | sqft_lot | f |
|---|---|---|---|---|---|---|---|
| **id** | 1.000000 | -0.016772 | 0.001150 | 0.005162 | -0.012241 | -0.131911 | 0.01 |
| **price** | -0.016772 | 1.000000 | 0.308787 | 0.525906 | 0.701917 | 0.089876 | 0.25 |
| **bedrooms** | 0.001150 | 0.308787 | 1.000000 | 0.514508 | 0.578212 | 0.032471 | 0.17 |
| **bathrooms** | 0.005162 | 0.525906 | 0.514508 | 1.000000 | 0.755758 | 0.088373 | 0.50 |
| **sqft_living** | -0.012241 | 0.701917 | 0.578212 | 0.755758 | 1.000000 | 0.173453 | 0.35 |
| **sqft_lot** | -0.131911 | 0.089876 | 0.032471 | 0.088373 | 0.173453 | 1.000000 | -0.00 |
| **floors** | 0.018608 | 0.256804 | 0.177944 | 0.502582 | 0.353953 | -0.004814 | 1.00 |
| **waterfront** | -0.004176 | 0.276295 | -0.002386 | 0.067282 | 0.110230 | 0.023143 | 0.02 |
| **view** | 0.011592 | 0.395734 | 0.078523 | 0.186451 | 0.282532 | 0.075298 | 0.02 |
| **condition** | -0.023803 | 0.036056 | 0.026496 | -0.126479 | -0.059445 | -0.008830 | -0.26 |
| **grade** | 0.008188 | 0.667951 | 0.356563 | 0.665838 | 0.762779 | 0.114731 | 0.45 |
| **sqft_above** | -0.010799 | 0.605368 | 0.479386 | 0.686668 | 0.876448 | 0.184139 | 0.52 |
| **sqft_basement** | -0.004548 | 0.325008 | 0.302683 | 0.282693 | 0.434576 | 0.015533 | -0.24 |
| **yr_built** | 0.021617 | 0.053953 | 0.155670 | 0.507173 | 0.318152 | 0.052946 | 0.48 |
| **yr_renovated** | -0.012010 | 0.129599 | 0.018495 | 0.051050 | 0.055660 | 0.004513 | 0.00 |
| **zipcode** | -0.008211 | -0.053402 | -0.154092 | -0.204786 | -0.199802 | -0.129586 | -0.05 |
| **lat** | -0.001798 | 0.306692 | -0.009951 | 0.024280 | 0.052155 | -0.085514 | 0.04 |
| **long** | 0.020672 | 0.022036 | 0.132054 | 0.224903 | 0.241214 | 0.230227 | 0.12 |
| **sqft_living15** | -0.002701 | 0.585241 | 0.393406 | 0.569884 | 0.756402 | 0.144763 | 0.28 |
| **sqft_lot15** | -0.138557 | 0.082845 | 0.030690 | 0.088303 | 0.184342 | 0.718204 | -0.01 |

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a382bf390>
```




```
Removing null values: Identified that some columns had null value
unianimously decided it was best to convert
these values to zeroes. We did not want to remove too many rows a
inspecting the columns, lots of values were
already zero so converting nulls to zeroes would not be impactful
```

In [13]:

```
n = df.nunique(axis=0)
print("No.of.unique values in each column :\n", n)
```

```
No.of.unique values in each column :
 id               21420
date               372
price             3622
bedrooms            12
bathrooms           29
sqft_living       1034
sqft_lot          9776
floors               6
waterfront           2
view                 5
condition            5
grade               11
sqft_above         942
sqft_basement      303
yr_built           116
yr_renovated        70
zipcode             70
lat               5033
long               751
sqft_living15      777
sqft_lot15        8682
dtype: int64
```

In [14]:

```
# Change nulls to 0 for 4 columns
df = df.replace(np.NaN, 0.0)
display(df.isna().sum())
```

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

In [15]:

```
# Change "?" to 0
df = df.replace("?", 0.0)
display(df.isna().sum())
```

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

```
Outliers: Start by cleaning price into 2 standard deviations from
the stats.zscore.
```

In [16]:

```
# get rid of outliers for price (2 SD from mean)
df2 = df[(np.abs(stats.zscore(df['price']) < 2))]
df2.head(2)
```

Out[16]:

| | id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors |
|---|---|---|---|---|---|---|---|---|
| **0** | 7129300520 | 10/13/2014 | 221900.0 | 3 | 1.00 | 1180 | 5650 | 1.0 |
| **1** | 6414100192 | 12/9/2014 | 538000.0 | 3 | 2.25 | 2570 | 7242 | 2.0 |

2 rows × 21 columns

In [17]:

```
df2['bedrooms'].value_counts()
```

Out[17]:

```
3     9666
4     6470
2     2744
5     1392
6      238
1      196
7       32
8        9
9        4
10       3
11       1
33       1
Name: bedrooms, dtype: int64
```

In [18]:

```
#Remove obvious outliers using .loc
df2 = df2.loc[df2['bedrooms'] < 33]
```

In [19]:

```
display(df2['grade'].value_counts())
display(df2['view'].value_counts())
display(df2['condition'].value_counts())
display(df2['waterfront'].value_counts())
display(df2['floors'].value_counts())
```

```
7     8966
8     5996
9     2421
6     2038
10     861
5      242
11     195
4       27
12       8
3        1
Name: grade, dtype: int64

0.0    19066
2.0      843
3.0      383
1.0      286
4.0      177
Name: view, dtype: int64

3    13481
4     5486
5     1591
2      169
1       28
Name: condition, dtype: int64

0.0    20693
1.0       62
Name: waterfront, dtype: int64

1.0    10501
2.0     7691
1.5     1859
3.0      578
2.5      120
3.5        6
Name: floors, dtype: int64
```

## ### Model Approach A

```
Our first approach had initial thoughts of either focusing on low
median priced homes. Felt that such a large price range of homes
well, so split into 3 and decided to focus on middle priced homes
$605K).
```

In [20]:

```
df2_high = df2.loc[df2['price'] > 605001]
df2_low = df2.loc[df2['price'] < 315000]
df2_mid = df2.loc[(df2['price'] > 315000) & (df2['price'] < 60500
```

In [21]:

```
#This approach focused on df2_mid with house prices ranging from
df2_mid.head(2)
```

Out[21]:

| | id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors |
|---|---|---|---|---|---|---|---|---|
| **1** | 6414100192 | 12/9/2014 | 538000.0 | 3 | 2.25 | 2570 | 7242 | 2.0 |
| **3** | 2487200875 | 12/9/2014 | 604000.0 | 4 | 3.00 | 1960 | 5000 | 1.0 |

2 rows × 21 columns

In [22]:

```
#Drop unnecessary columns
df2_mid = df2_mid.drop(['id', 'date', 'zipcode', 'lat', 'long'],
```

In [23]:

```
df2_mid.head(2)
```

Out[23]:

| | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condi |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 538000.0 | 3 | 2.25 | 2570 | 7242 | 2.0 | 0.0 | 0.0 | |
| **3** | 604000.0 | 4 | 3.00 | 1960 | 5000 | 1.0 | 0.0 | 0.0 | |

In [24]:

```
# Model 1: Playing around with different features, averaging an r
f = 'price~bathrooms+yr_built+yr_renovated+grade'
model = ols(formula=f, data=df2_mid).fit()
model.summary()
```

Out[24]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | price | **R-squared:** | 0.145 |
| **Model:** | OLS | **Adj. R-squared:** | 0.144 |
| **Method:** | Least Squares | **F-statistic:** | 434.2 |
| **Date:** | Sun, 29 Nov 2020 | **Prob (F-statistic):** | 0.00 |
| **Time:** | 20:41:39 | **Log-Likelihood:** | -1.3009e+05 |
| **No. Observations:** | 10279 | **AIC:** | 2.602e+05 |
| **Df Residuals:** | 10274 | **BIC:** | 2.602e+05 |
| **Df Model:** | 4 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **Intercept** | 2.138e+06 | 6.51e+04 | 32.829 | 0.000 | 2.01e+06 | 2.27e+06 |
| **bathrooms** | 2.398e+04 | 1510.640 | 15.875 | 0.000 | 2.1e+04 | 2.69e+04 |
| **yr_built** | -1027.1826 | 35.083 | -29.278 | 0.000 | -1095.953 | -958.413 |
| **yr_renovated** | -2.5192 | 2.441 | -1.032 | 0.302 | -7.304 | 2.266 |
| **grade** | 3.83e+04 | 1180.910 | 32.429 | 0.000 | 3.6e+04 | 4.06e+04 |

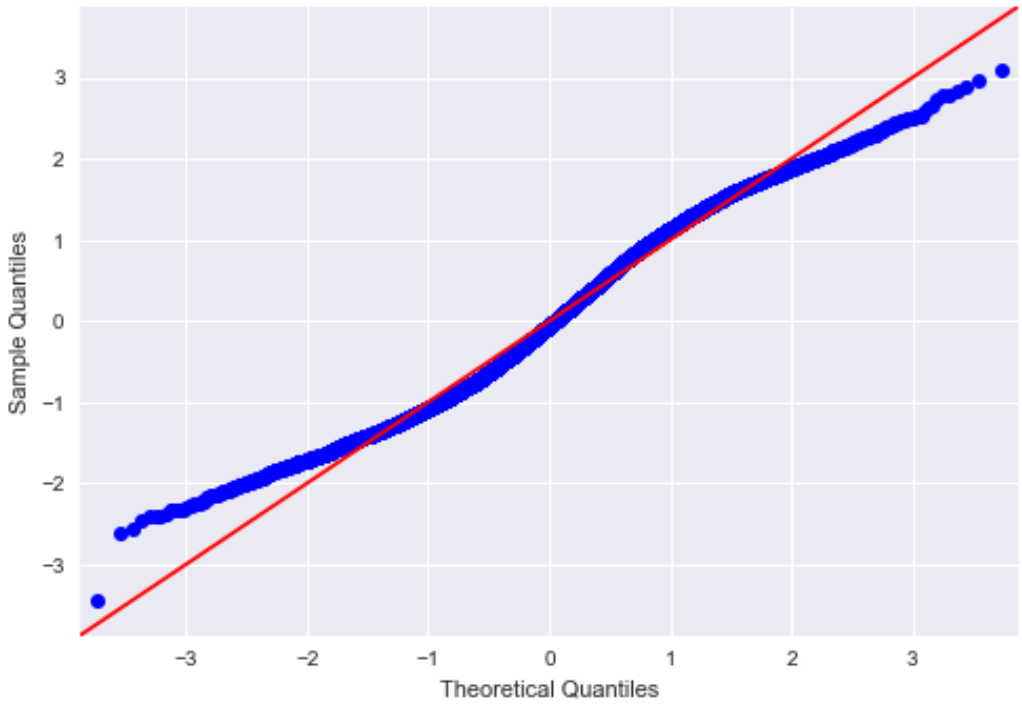
| | | | |
|---|---|---|---|
| **Omnibus:** | 985.935 | **Durbin-Watson:** | 1.996 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 325.568 |
| **Skew:** | 0.151 | **Prob(JB):** | 2.01e-71 |
| **Kurtosis:** | 2.182 | **Cond. No.** | 1.72e+05 |

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifiec
[2] The condition number is large, 1.72e+05. This might indicate that there are strong multicollinearity or other numerical problems.

In [25]:

```
# QQ-Plot
import scipy.stats as stats
residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', f
fig.show()
```

```
/Users/MZhang/opt/anaconda3/envs/learn-env/lib/python3.6/site-pac
s/ipykernel_launcher.py:5: UserWarning: Matplotlib is currently u
module://ipykernel.pylab.backend_inline, which is a non-GUI backe
so cannot show the figure.
  """
```



In [26]:

```
#This looks at residual distributions for individual features(pred
#Remember to stop this cell manually if you don't run through each
import statsmodels.stats.api as sms
results = []
for idx, column in enumerate(df2_mid.columns):
    print (f"regression for king county housing for mid income~{co

    f = f'price~{column}'
    model = ols(formula=f, data=df2_mid).fit()

    fig, axes = plt.subplots(figsize=(15,12))
    fig = sm.graphics.plot_regress_exog(model, column, fig=fig)
    fig = sm.graphics.qqplot(model.resid, dist=stats.norm, line='4
    fig.tight_layout()
    plt.show()

    results.append([column, model.rsquared, model.params[0], model
    input("Press Enter to continue...")
```

```
regression for king county housing for mid income~price
```

## ### Approach B

```
Our second approach took a previous df (df2_low - 20,452 x 39) an
a low end price range of $154K to $315K. Had dummies for 4 variab
grade, bedrooms, bath-rooms).
```

In [27]:

```
# Create dummies for grade
grade_dummies = pd.get_dummies(df2['grade'], prefix='grade', drop
df2 = pd.concat([df2, grade_dummies], axis=1)
df2.head()
```

Out[27]:

| | id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors |
|---|---|---|---|---|---|---|---|---|
| **0** | 7129300520 | 10/13/2014 | 221900.0 | 3 | 1.00 | 1180 | 5650 | 1.0 |
| **1** | 6414100192 | 12/9/2014 | 538000.0 | 3 | 2.25 | 2570 | 7242 | 2.0 |
| **2** | 5631500400 | 2/25/2015 | 180000.0 | 2 | 1.00 | 770 | 10000 | 1.0 |
| **3** | 2487200875 | 12/9/2014 | 604000.0 | 4 | 3.00 | 1960 | 5000 | 1.0 |
| **4** | 1954400510 | 2/18/2015 | 510000.0 | 3 | 2.00 | 1680 | 8080 | 1.0 |

5 rows × 30 columns

In [28]:

```
# Cut outliers for bathroom, then create dummies
# Went up to around 8.0 with 1 count so we removed those values
df2 = df2[(np.abs(stats.zscore(df2['bathrooms']) < 2))]
df2['bathrooms'].value_counts()
```

Out[28]:

```
2.50    5270
1.00    3849
1.75    3029
2.25    1988
2.00    1917
1.50    1439
2.75    1134
3.00     695
3.50     585
3.25     463
0.75      71
1.25       8
0.50       4
Name: bathrooms, dtype: int64
```

In [29]:

```
# Create dummies for bathrooms
bathrooms_dummies = pd.get_dummies(df2['bathrooms'], prefix='bath
# df2_remove = train4.drop(['OverallQual'], axis=1)
df2 = pd.concat([df2, bathrooms_dummies], axis=1)
df2.head()
```

Out[29]:

| | id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors |
|---|---|---|---|---|---|---|---|---|
| **0** | 7129300520 | 10/13/2014 | 221900.0 | 3 | 1.00 | 1180 | 5650 | 1.0 |
| **1** | 6414100192 | 12/9/2014 | 538000.0 | 3 | 2.25 | 2570 | 7242 | 2.0 |
| **2** | 5631500400 | 2/25/2015 | 180000.0 | 2 | 1.00 | 770 | 10000 | 1.0 |
| **3** | 2487200875 | 12/9/2014 | 604000.0 | 4 | 3.00 | 1960 | 5000 | 1.0 |
| **4** | 1954400510 | 2/18/2015 | 510000.0 | 3 | 2.00 | 1680 | 8080 | 1.0 |

5 rows × 42 columns

In [30]:

```
df2 = df2.drop(['bathrooms', 'grade'], axis=1)
pd.set_option('display.max_columns', None)
```

In [31]:

```
df2.head(2)
```

Out[31]:

| | id | date | price | bedrooms | sqft_living | sqft_lot | floors | waterfront |
|---|---|---|---|---|---|---|---|---|
| **0** | 7129300520 | 10/13/2014 | 221900.0 | 3 | 1180 | 5650 | 1.0 | 0.0 |
| **1** | 6414100192 | 12/9/2014 | 538000.0 | 3 | 2570 | 7242 | 2.0 | 0.0 |

In [32]:

```
# Had to convert the "." to "_" because when selecting these spec
# when using "." in a variable/column name.
df2 = df2.rename(columns={'bath_0.75':'bath_0_75', 'bath_1.0':'ba
                          'bath_1.5':'bath_1_5', 'b
                          'bath_2.25':'bath_2_25',
                          'bath_3.0':'bath_3_0', 'b
```

In [33]:

```
# Create dummies for condition
condition_dummies = pd.get_dummies(df2['condition'], prefix='cond
df2 = pd.concat([df2, condition_dummies], axis=1)
df2.head()
```

Out[33]:

| | id | date | price | bedrooms | sqft_living | sqft_lot | floors | waterfront |
|---|---|---|---|---|---|---|---|---|
| **0** | 7129300520 | 10/13/2014 | 221900.0 | 3 | 1180 | 5650 | 1.0 | 0.0 |
| **1** | 6414100192 | 12/9/2014 | 538000.0 | 3 | 2570 | 7242 | 2.0 | 0.0 |
| **2** | 5631500400 | 2/25/2015 | 180000.0 | 2 | 770 | 10000 | 1.0 | 0.0 |
| **3** | 2487200875 | 12/9/2014 | 604000.0 | 4 | 1960 | 5000 | 1.0 | 0.0 |
| **4** | 1954400510 | 2/18/2015 | 510000.0 | 3 | 1680 | 8080 | 1.0 | 0.0 |

In [34]:

```
# Create dummies for bedrooms
bedrooms_dummies = pd.get_dummies(df2['bedrooms'], prefix='bed',
df2 = pd.concat([df2, bedrooms_dummies], axis=1)
df2.head()
```

Out[34]:

| | id | date | price | bedrooms | sqft_living | sqft_lot | floors | waterfront |
|---|---|---|---|---|---|---|---|---|
| **0** | 7129300520 | 10/13/2014 | 221900.0 | 3 | 1180 | 5650 | 1.0 | 0.0 |
| **1** | 6414100192 | 12/9/2014 | 538000.0 | 3 | 2570 | 7242 | 2.0 | 0.0 |
| **2** | 5631500400 | 2/25/2015 | 180000.0 | 2 | 770 | 10000 | 1.0 | 0.0 |
| **3** | 2487200875 | 12/9/2014 | 604000.0 | 4 | 1960 | 5000 | 1.0 | 0.0 |
| **4** | 1954400510 | 2/18/2015 | 510000.0 | 3 | 1680 | 8080 | 1.0 | 0.0 |

In [35]:

```
df2 = df2.drop(['condition', 'bedrooms'], axis=1)
```

In [36]:

```
#Drop unnecessary columns
df2 = df2.drop(['id', 'date', 'zipcode', 'lat', 'long'], axis=1)
```

In [37]:

```
df2.head(2)
```

Out[37]:

| | price | sqft_living | sqft_lot | floors | waterfront | view | sqft_above | sqft_basement | y |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 221900.0 | 1180 | 5650 | 1.0 | 0.0 | 0.0 | 1180 | 0.0 | |
| **1** | 538000.0 | 2570 | 7242 | 2.0 | 0.0 | 0.0 | 2170 | 400.0 | |

In [38]:

```
#Create a low priced df, values decided off of the median and a l
#not too low such as $50,000.
df3 = df2.loc[(df2['price'] >= 154000) & (df2['price'] <= 315000)
```

In [39]:

```
# Cut years outliers on low end (only include if >1940)
df3 = df3.loc[df3['yr_built'] > 1940]
df3.shape
```

Out[39]:

```
(4379, 47)
```

In [40]:

```
# Look at descriptives for new df with +- 2 SD bands added
df3.describe()
desc_df = df3.describe()
desc_df.loc['+2_std'] = desc_df.loc['mean'] + (desc_df.loc['std']
desc_df.loc['-2_std'] = desc_df.loc['mean'] - (desc_df.loc['std']
desc_df
```

Out[40]:

| | price | sqft_living | sqft_lot | floors | waterfront | |
|---|---|---|---|---|---|---|
| **count** | 4379.000000 | 4379.000000 | 4379.000000 | 4379.000000 | 4379.000000 | 4379.000 |
| **mean** | 254876.091573 | 1506.313999 | 9689.443709 | 1.284654 | 0.000228 | 0.032 |
| **std** | 39819.607680 | 447.856694 | 13555.403862 | 0.462610 | 0.015112 | 0.265 |
| **min** | 154000.000000 | 380.000000 | 572.000000 | 1.000000 | 0.000000 | 0.000 |
| **25%** | 226000.000000 | 1180.000000 | 6229.500000 | 1.000000 | 0.000000 | 0.000 |
| **50%** | 260000.000000 | 1470.000000 | 7904.000000 | 1.000000 | 0.000000 | 0.000 |
| **75%** | 288000.000000 | 1800.000000 | 9690.500000 | 2.000000 | 0.000000 | 0.000 |
| **max** | 315000.000000 | 3340.000000 | 306848.000000 | 3.000000 | 1.000000 | 3.000 |
| **+2_std** | 334515.306933 | 2402.027387 | 36800.251433 | 2.209875 | 0.030452 | 0.564 |
| **-2_std** | 175236.876214 | 610.600611 | -17421.364015 | 0.359433 | -0.029995 | -0.499 |

In [41]:

```
df3.head(2)
```

Out[41]:

| | price | sqft_living | sqft_lot | floors | waterfront | view | sqft_above | sqft_basement | y |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 221900.0 | 1180 | 5650 | 1.0 | 0.0 | 0.0 | 1180 | 0.0 | |
| **6** | 257500.0 | 1715 | 6819 | 2.0 | 0.0 | 0.0 | 1715 | 0.0 | |

In [42]:

```
# Check for duplicates
duplicate_rows_df = df3[df3.duplicated()]
print(duplicate_rows_df.shape)
duplicate_rows_df
```

```
(1, 47)
```

Out[42]:

| | price | sqft_living | sqft_lot | floors | waterfront | view | sqft_above | sqft_basemen |
|---|---|---|---|---|---|---|---|---|
| **4348** | 259950.0 | 1070 | 649 | 2.0 | 0.0 | 0.0 | 720 | 350.( |

In [43]:

```
# Examine yr_built data via histogram...
plt.hist(df3.yr_built, bins=30, density=True, alpha=0.5, histtype
```

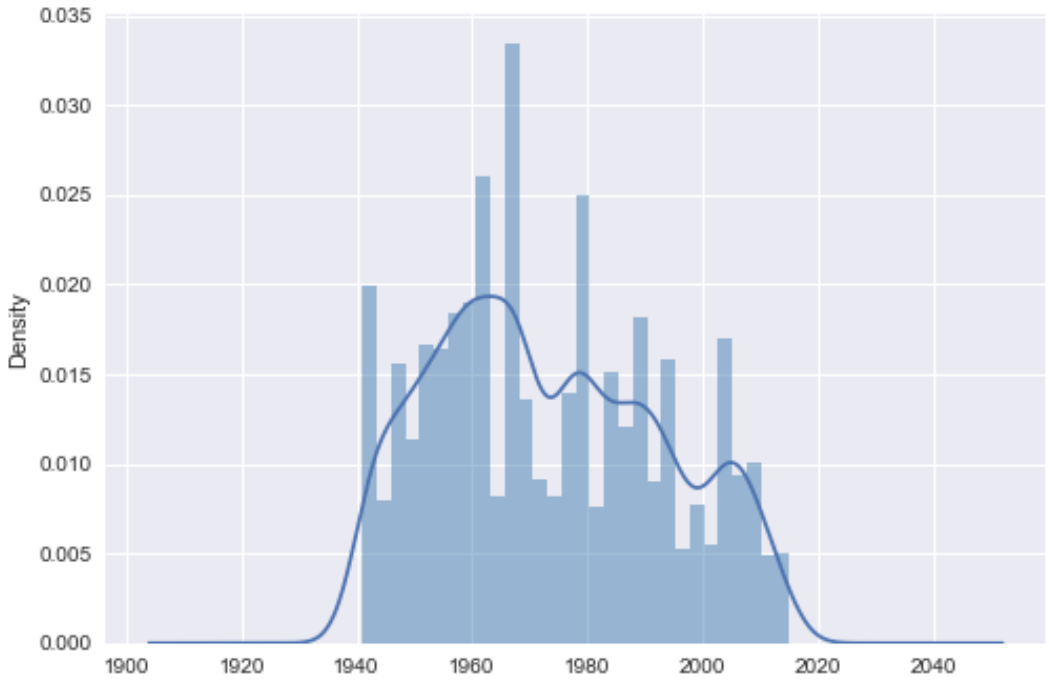
```
        0.00907279, 0.01583109, 0.00527703, 0.00777667, 0.0054621
        0.01703462, 0.00935052, 0.01009116, 0.00490671, 0.0049992
 array([1941.        , 1943.46666667, 1945.93333333, 1948.4
        1950.86666667, 1953.33333333, 1955.8       , 1958.2666666
        1960.73333333, 1963.2       , 1965.66666667, 1968.1333333
        1970.6       , 1973.06666667, 1975.53333333, 1978.
        1980.46666667, 1982.93333333, 1985.4       , 1987.8666666
        1990.33333333, 1992.8       , 1995.26666667, 1997.7333333
        2000.2       , 2002.66666667, 2005.13333333, 2007.6
        2010.06666667, 2012.53333333, 2015.        ]),
 <a list of 1 Patch objects>)
```

In [44]:

```
# Examine yr_built data via histogram...
plt.hist(df3.yr_built, bins=30, density=True, alpha=0.5, histtype
df3['yr_built'].plot.kde(label ='kde')
```

Out[44]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a3d30cdd8>
```




In [45]:

```
# Create train, text split
train, test = train_test_split(df3, test_size=0.25, random_state=
```

In [46]:

```
train.shape
```

Out[46]:

```
(3284, 47)
```

In [47]:

```
from statsmodels.formula.api import ols
f = 'price~grade_6+grade_7+grade_8+sqft_living+bed_2+bed_3+bed_4'
# create a fitted model in one line
model = ols(formula=f, data=train).fit()
model.summary()
```

Out[47]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | price | **R-squared:** | 0.152 |
| **Model:** | OLS | **Adj. R-squared:** | 0.150 |
| **Method:** | Least Squares | **F-statistic:** | 83.93 |
| **Date:** | Sun, 29 Nov 2020 | **Prob (F-statistic):** | 1.37e-112 |
| **Time:** | 20:43:11 | **Log-Likelihood:** | -39184. |
| **No. Observations:** | 3284 | **AIC:** | 7.838e+04 |
| **Df Residuals:** | 3276 | **BIC:** | 7.843e+04 |
| **Df Model:** | 7 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **Intercept** | 1.946e+05 | 5867.285 | 33.175 | 0.000 | 1.83e+05 | 2.06e+05 |
| **grade_6** | -233.8749 | 4363.136 | -0.054 | 0.957 | -8788.626 | 8320.876 |
| **grade_7** | 9034.7580 | 4210.551 | 2.146 | 0.032 | 779.179 | 1.73e+04 |
| **grade_8** | 1.626e+04 | 4587.610 | 3.545 | 0.000 | 7267.145 | 2.53e+04 |
| **sqft_living** | 31.0042 | 1.921 | 16.140 | 0.000 | 27.238 | 34.771 |
| **bed_2** | 1.218e+04 | 3882.032 | 3.137 | 0.002 | 4565.131 | 1.98e+04 |
| **bed_3** | 4984.7312 | 3379.697 | 1.475 | 0.140 | -1641.802 | 1.16e+04 |
| **bed_4** | 3710.9878 | 3479.060 | 1.067 | 0.286 | -3110.365 | 1.05e+04 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 66.208 | **Durbin-Watson:** | 1.988 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 59.455 |
| **Skew:** | -0.278 | **Prob(JB):** | 1.23e-13 |
| **Kurtosis:** | 2.645 | **Cond. No.** | 2.08e+04 |

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifiec

[2] The condition number is large, 2.08e+04. This might indicate that there are

In [48]:

```
# Split data into x and y versions of train and test
y_test = test[['price']]
X_test = test.drop(['price'],axis=1)
y_train = train[['price']]
X_train = train.drop(['price'],axis=1)
```

In [49]:

```
# Try and baseline this to the test data set (before changes)
from sklearn.linear_model import LinearRegression

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

y_hat_train = linreg.predict(X_train)
y_hat_test = linreg.predict(X_test)
```

In [50]:

```
from sklearn.metrics import mean_squared_error

train_mse = mean_squared_error(y_train, y_hat_train)
test_mse = mean_squared_error(y_test, y_hat_test)

print('Train Root Mean Squarred Error:', train_mse**0.5)
print('Test Root Mean Squarred Error:', test_mse**0.5)
```

```
Train Root Mean Squarred Error: 35566.276817254875
Test Root Mean Squarred Error: 35041.22814893627
```

## ### Approach C

```
At this point, we were still using df6_low (10,013 x 21); and dec
do some transformations on the data to see if that would help. We
grade. And then performed min-max scaling on most of the other pr
variables (13) with the exception of price, yr_built, and yr_reno
```

In [51]:

```
dfx = pd.read_csv('data/kc_house_data.csv')
```

In [52]:

```
dfx['sqft_basement'] = pd.to_numeric(dfx['sqft_basement'], errors
display(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
id               21597 non-null int64
date             21597 non-null object
price            21597 non-null float64
bedrooms         21597 non-null int64
bathrooms        21597 non-null float64
sqft_living      21597 non-null int64
sqft_lot         21597 non-null int64
floors           21597 non-null float64
waterfront       21597 non-null float64
view             21597 non-null float64
condition        21597 non-null int64
grade            21597 non-null int64
sqft_above       21597 non-null int64
sqft_basement    21597 non-null float64
yr_built         21597 non-null int64
yr_renovated     21597 non-null float64
zipcode          21597 non-null int64
lat              21597 non-null float64
long             21597 non-null float64
sqft_living15    21597 non-null int64
sqft_lot15       21597 non-null int64
dtypes: float64(9), int64(11), object(1)
memory usage: 3.5+ MB

None
```

In [53]:

```
# Change nulls to 0 for 4 columns
dfy = dfx.replace(np.NaN, 0.0)
display(dfy.isnull().sum())
```

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

In [54]:

```
# get rid of outliers for price (2 SD from mean)
dfz = dfy[(np.abs(stats.zscore(dfy['price']) < 2))]
```

In [55]:

```
dfz = dfz.loc[dfz['bedrooms'] < 33]
```

In [56]:

```
# Cut outliers for bathroom
df5 = dfz[(np.abs(stats.zscore(dfz['bathrooms']) < 2))]
df5['bathrooms'].value_counts()
```

Out[56]:

```
2.50    5270
1.00    3849
1.75    3029
2.25    1988
2.00    1917
1.50    1439
2.75    1134
3.00     695
3.50     585
3.25     463
0.75      71
1.25       8
0.50       4
Name: bathrooms, dtype: int64
```

In [57]:

```
# Create the low priced df... use < 438K and also > 154K, based o
df6_low = df5.loc[(df5['price'] >= 154000) & (df5['price'] <= 438
df6_low.head()
```

Out[57]:

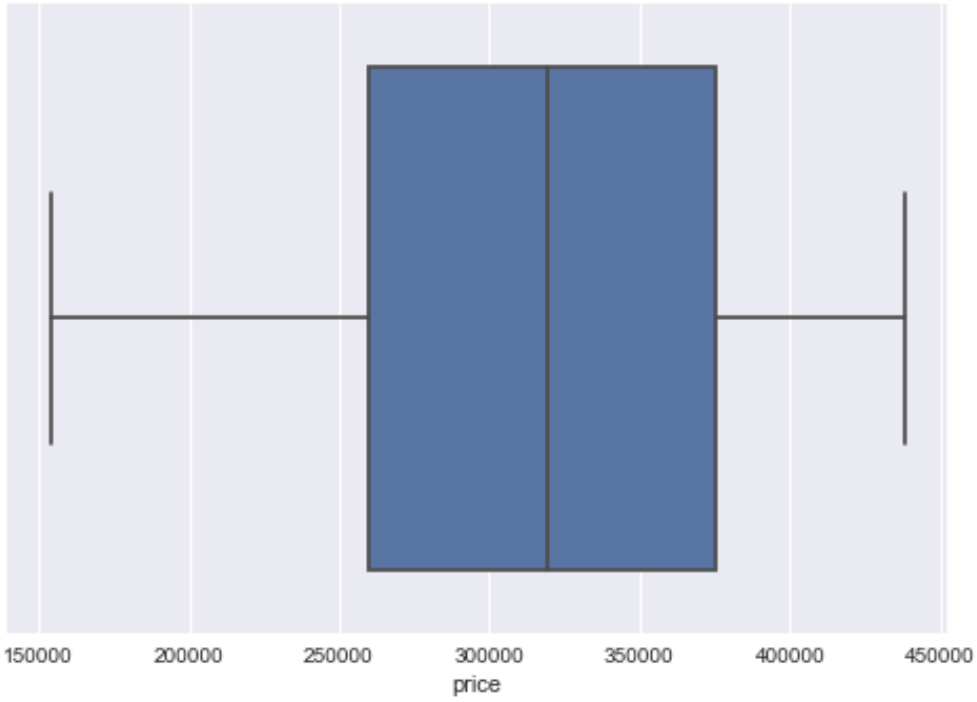
| | id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors |
|---|---|---|---|---|---|---|---|---|
| **0** | 7129300520 | 10/13/2014 | 221900.0 | 3 | 1.00 | 1180 | 5650 | 1.0 |
| **2** | 5631500400 | 2/25/2015 | 180000.0 | 2 | 1.00 | 770 | 10000 | 1.0 |
| **6** | 1321400060 | 6/27/2014 | 257500.0 | 3 | 2.25 | 1715 | 6819 | 2.0 |
| **7** | 2008000270 | 1/15/2015 | 291850.0 | 3 | 1.50 | 1060 | 9711 | 1.0 |
| **8** | 2414600126 | 4/15/2015 | 229500.0 | 3 | 1.00 | 1780 | 7470 | 1.0 |

In [58]:

```
sns.boxplot(x=df6_low['price'])
```

Out[58]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a400d0c50>
```

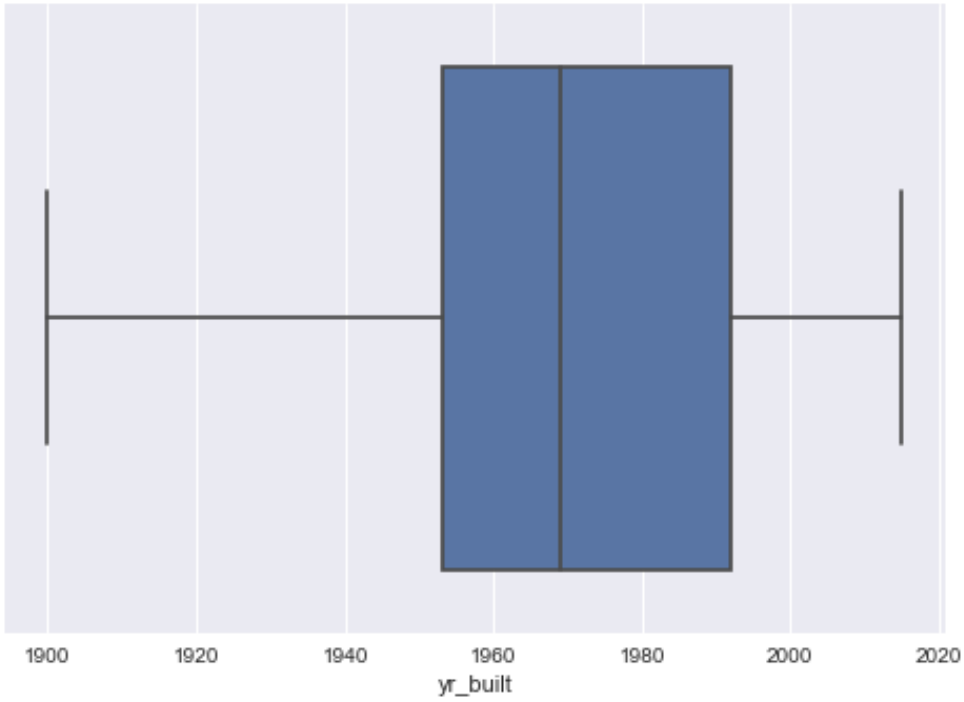


In [96]:

```
sns.boxplot(x=df6_low['yr_built'])
```

Out[96]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a372a9828>
```

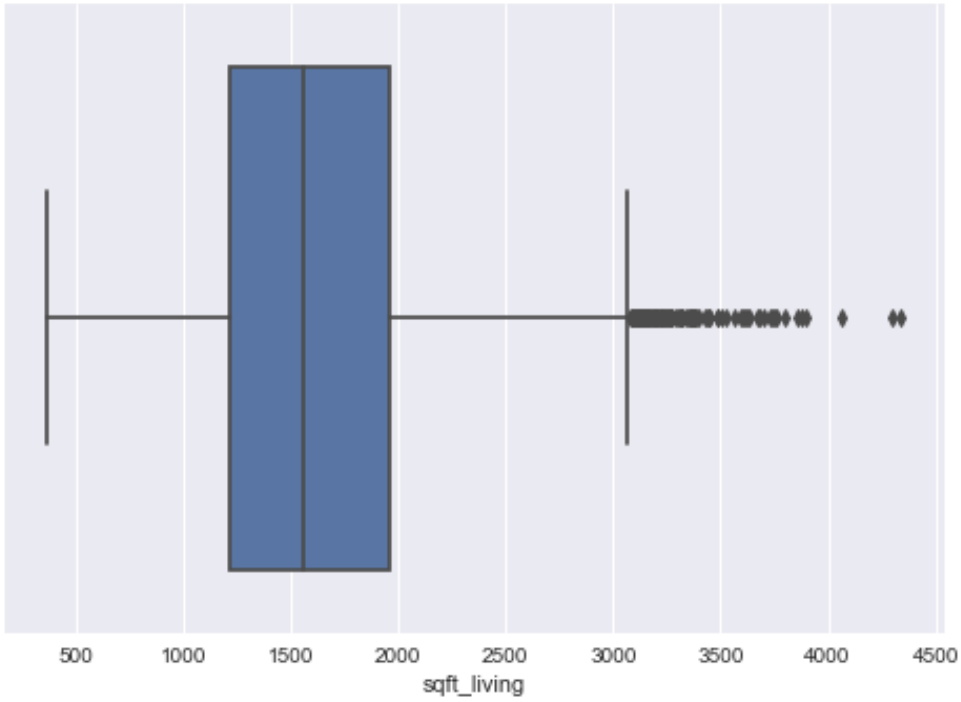


In [97]:

```
sns.boxplot(x=df6_low['sqft_living'])
```

Out[97]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a372faef0>
```

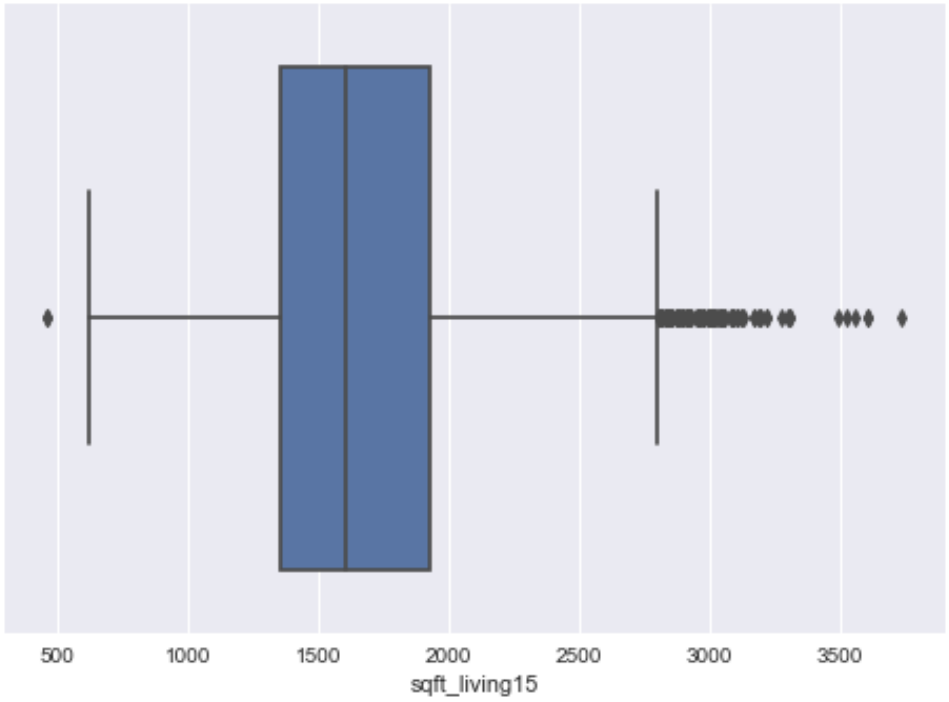


In [98]:

```
sns.boxplot(x=df6_low['sqft_living15'])
```

Out[98]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a3800cc18>
```

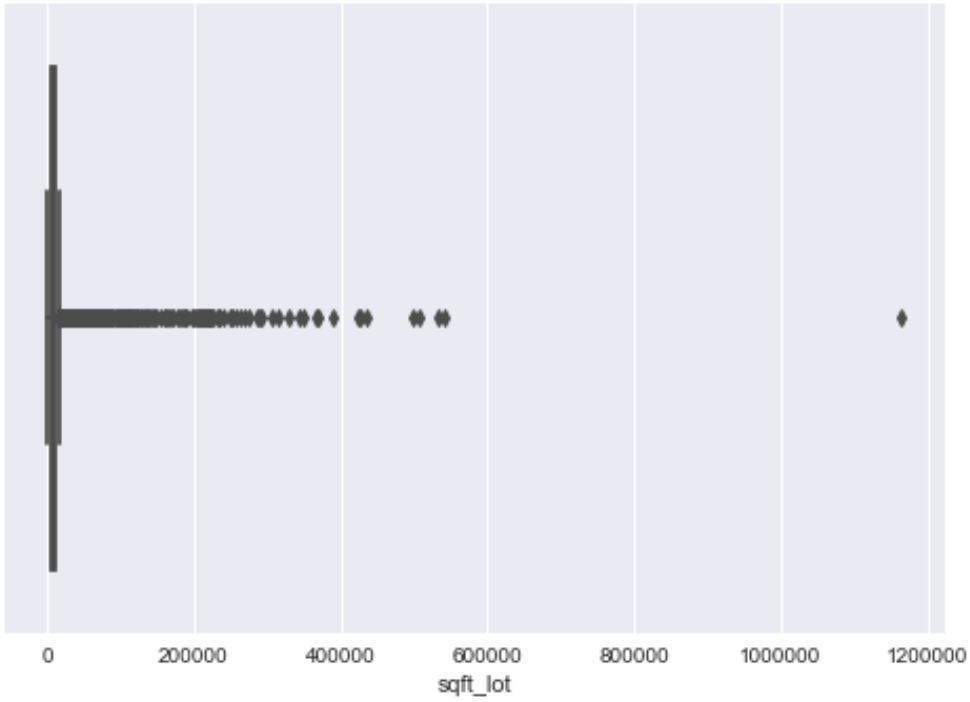


In [99]:

```
sns.boxplot(x=df6_low['sqft_lot'])
```

Out[99]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a3987f080>
```

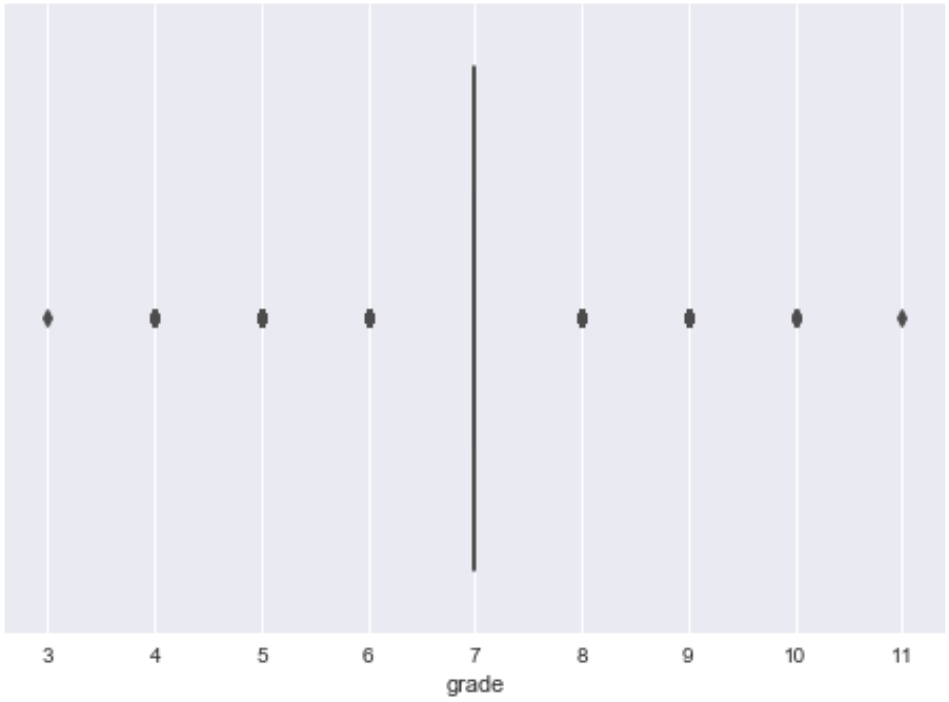


In [100]:

```
sns.boxplot(x=df6_low['grade'])
```

Out[100]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a3996ab38>
```

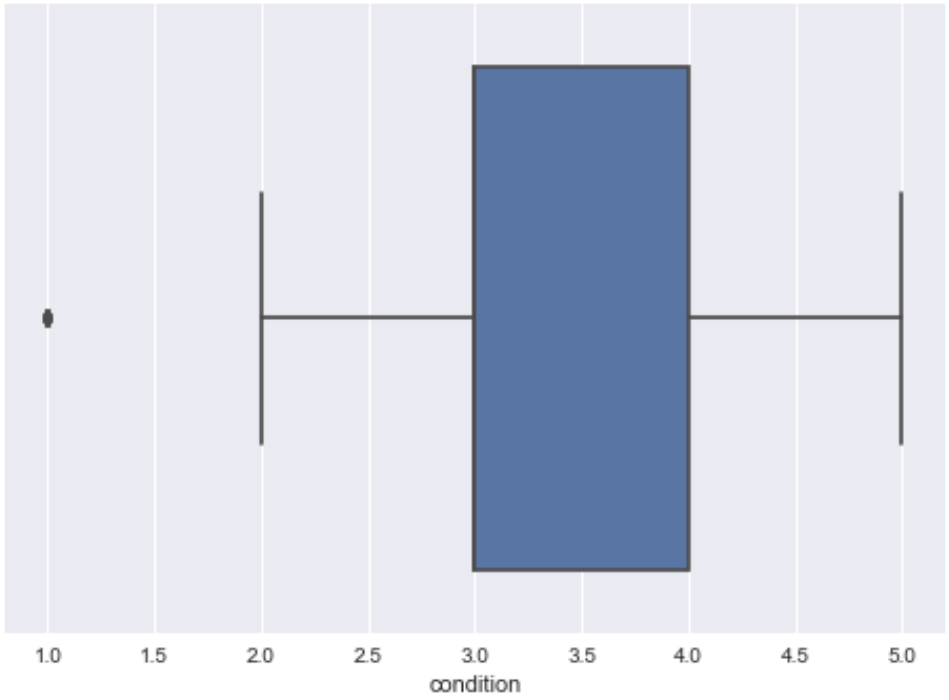


In [101]:

```
sns.boxplot(x=df6_low['condition'])
```

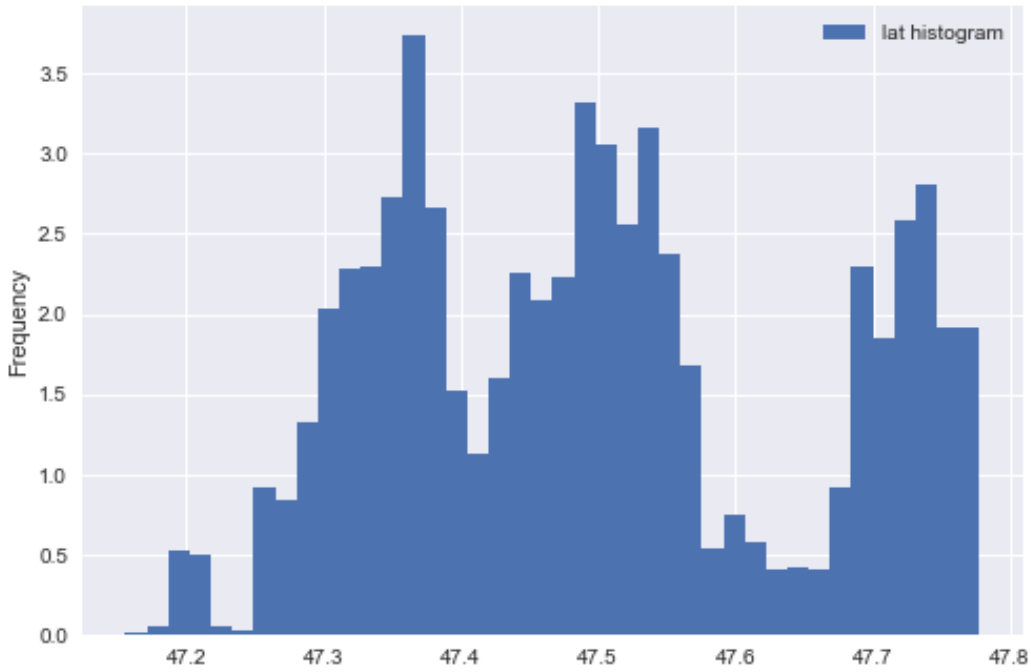
Out[101]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a39a56588>
```



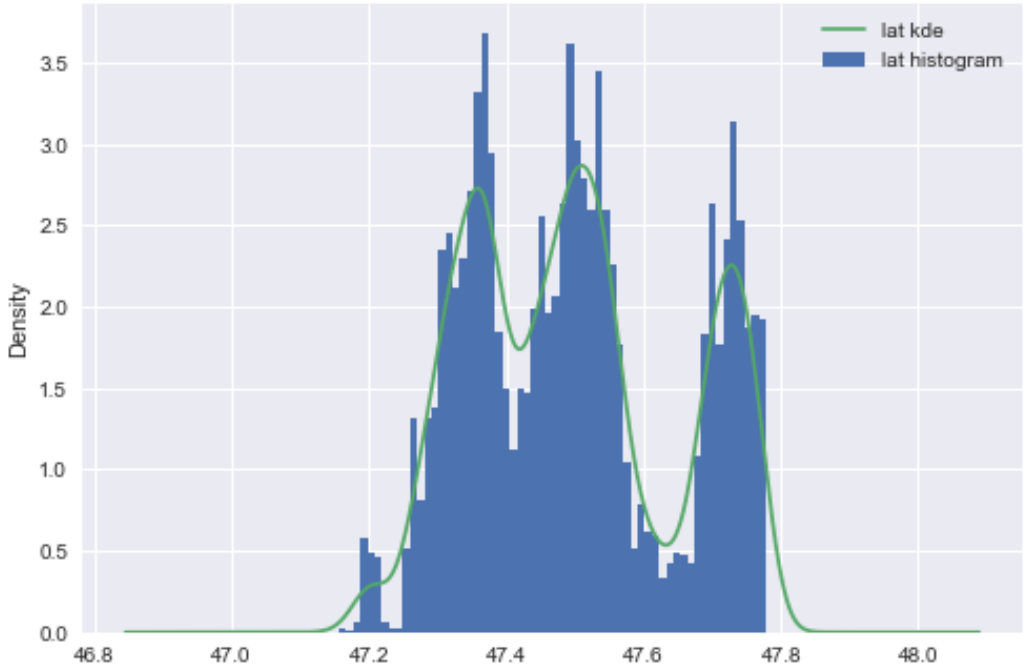
In [102]:

```
df6_low.lat.plot.hist(density=True, bins=40, label = 'lat histogr
plt.legend()
plt.show()
```



In [103]:

```
df6_low.lat.plot.hist(density=True, bins=60, label = 'lat histogr
df6_low.lat.plot.kde(label ='lat kde')
plt.legend()
plt.show()
```



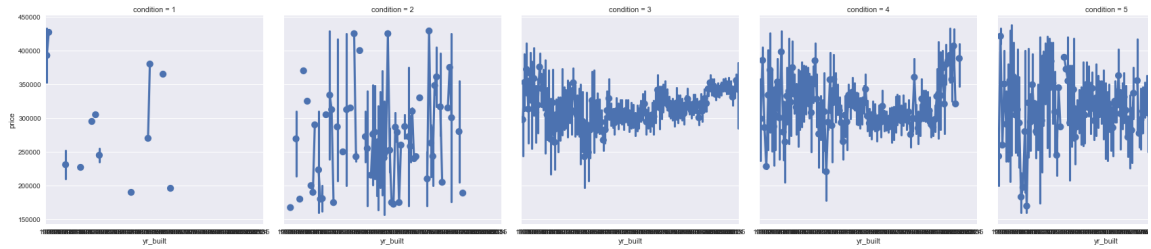
In [104]:

```
#Exploring different plots
sns.factorplot(data=df6_low, x= "yr_built", y="price", col="condi
```

```
/Users/MZhang/opt/anaconda3/envs/learn-env/lib/python3.6/site-pac
s/seaborn/categorical.py:3704: UserWarning: The `factorplot` func
has been renamed to `catplot`. The original name will be removed
future release. Please update your code. Note that the default `k
in `factorplot` (`'point'`) has changed `'strip'` in `catplot`.
  warnings.warn(msg)
```

Out[104]:

```
<seaborn.axisgrid.FacetGrid at 0x1a39a80ba8>
```

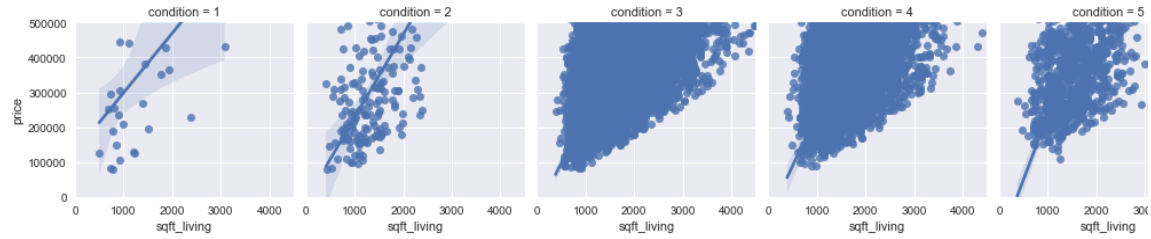



In [105]:

```
g = sns.FacetGrid(df, col="condition")
g.map(sns.regplot, "sqft_living", "price")
plt.xlim(0, 4500)
plt.ylim(0, 500000)
```

Out[105]:

```
(0, 500000)
```

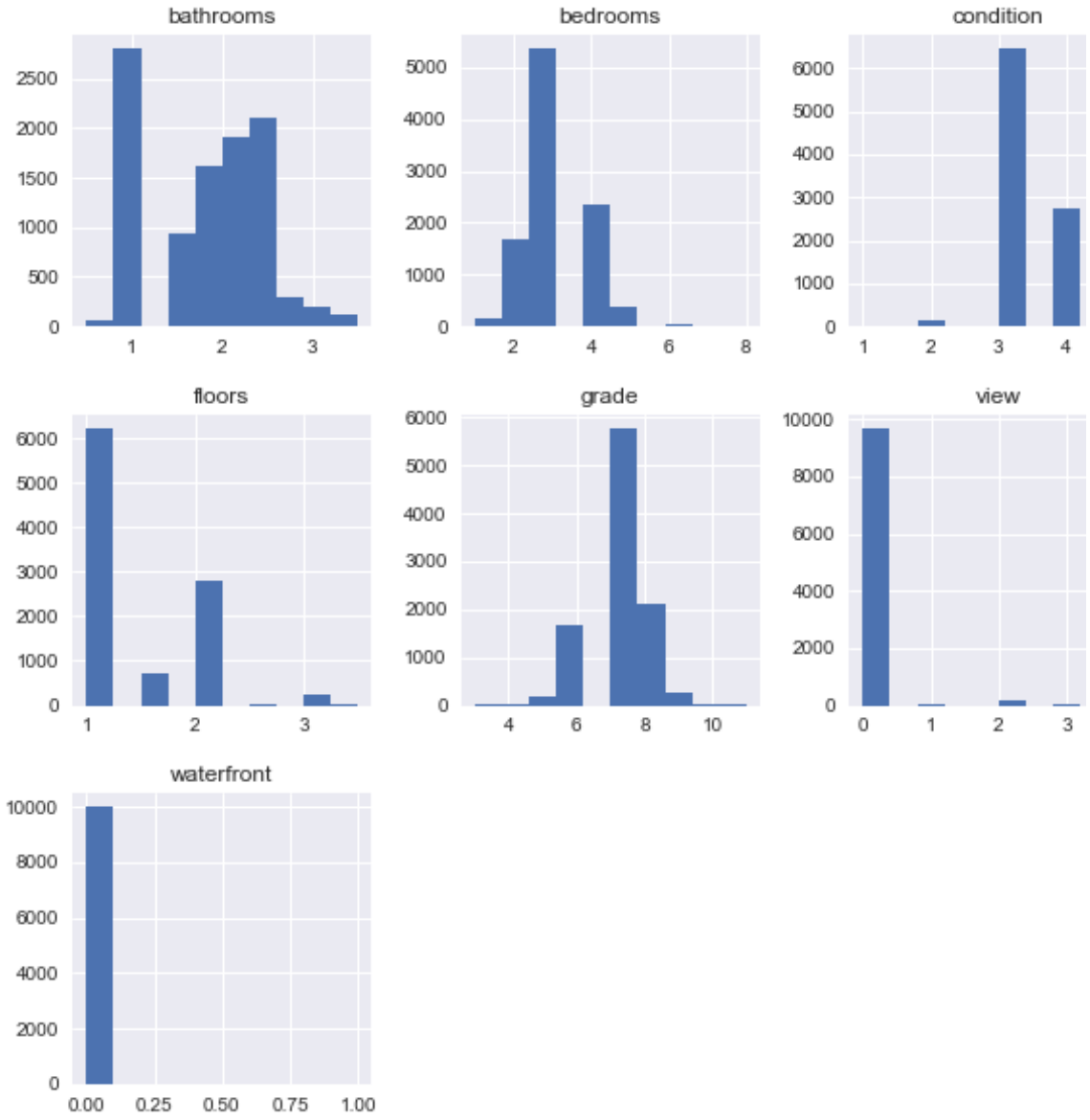


In [106]:

```
df6_low[['bedrooms', 'bathrooms', 'floors', 'waterfront', 'view',
```

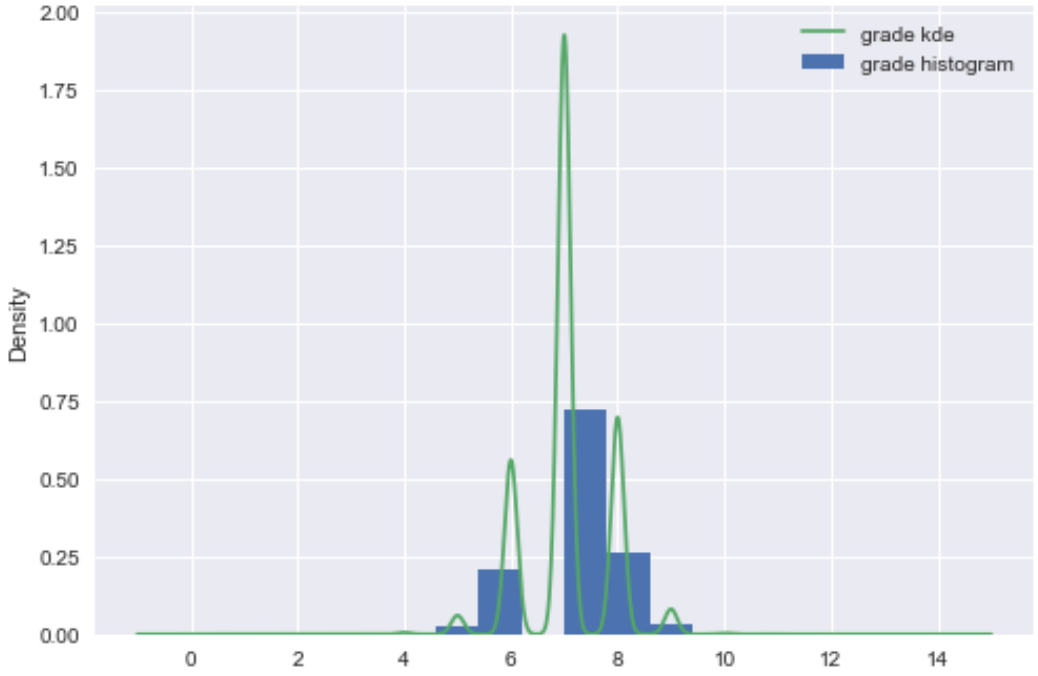
Out[106]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1a3a66
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a3bb0
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a3bb1
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1a3a7b
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a3a8b
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a3a9f
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1a3aa2
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a3aac
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a3aac
>]],
      dtype=object)
```


bathrooms
bedrooms
condition
floors
grade
view
waterfront

In [108]:

```
df6_low.grade.plot.hist(density=True, label = 'grade histogram')
df6_low.grade.plot.kde(label ='grade kde')
plt.legend()
plt.show()
```

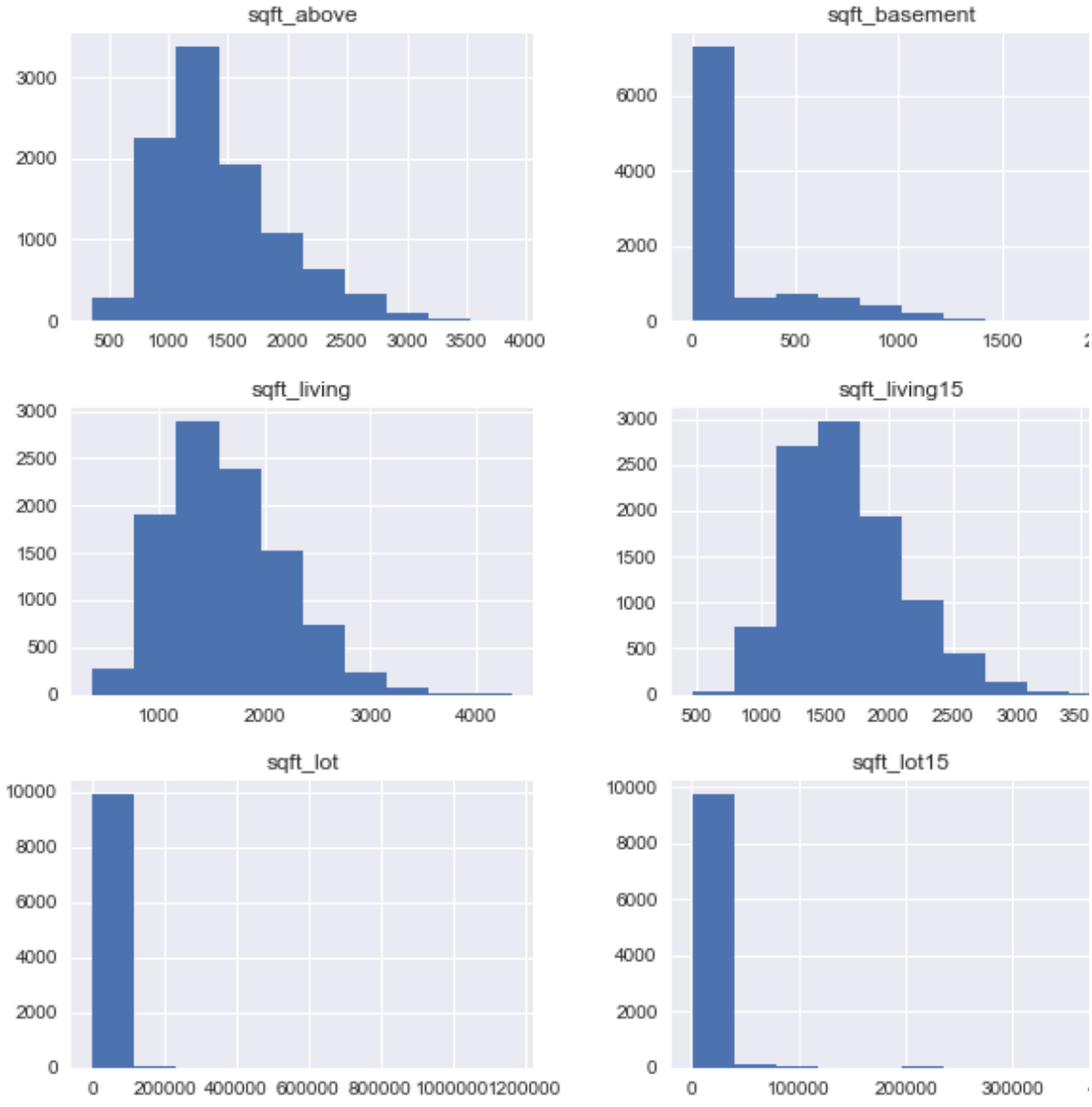


In [109]:

```
df6_low[['sqft_living', 'sqft_lot', 'sqft_above', 'sqft_basement'
```

Out[109]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1a3af7
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a3b1c
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1a3b1f
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a3b23
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1a3bb4
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a3bf4
>]],
      dtype=object)
```



In [111]:

```
df6_low['sqft_lot'].max()
```

Out[111]:

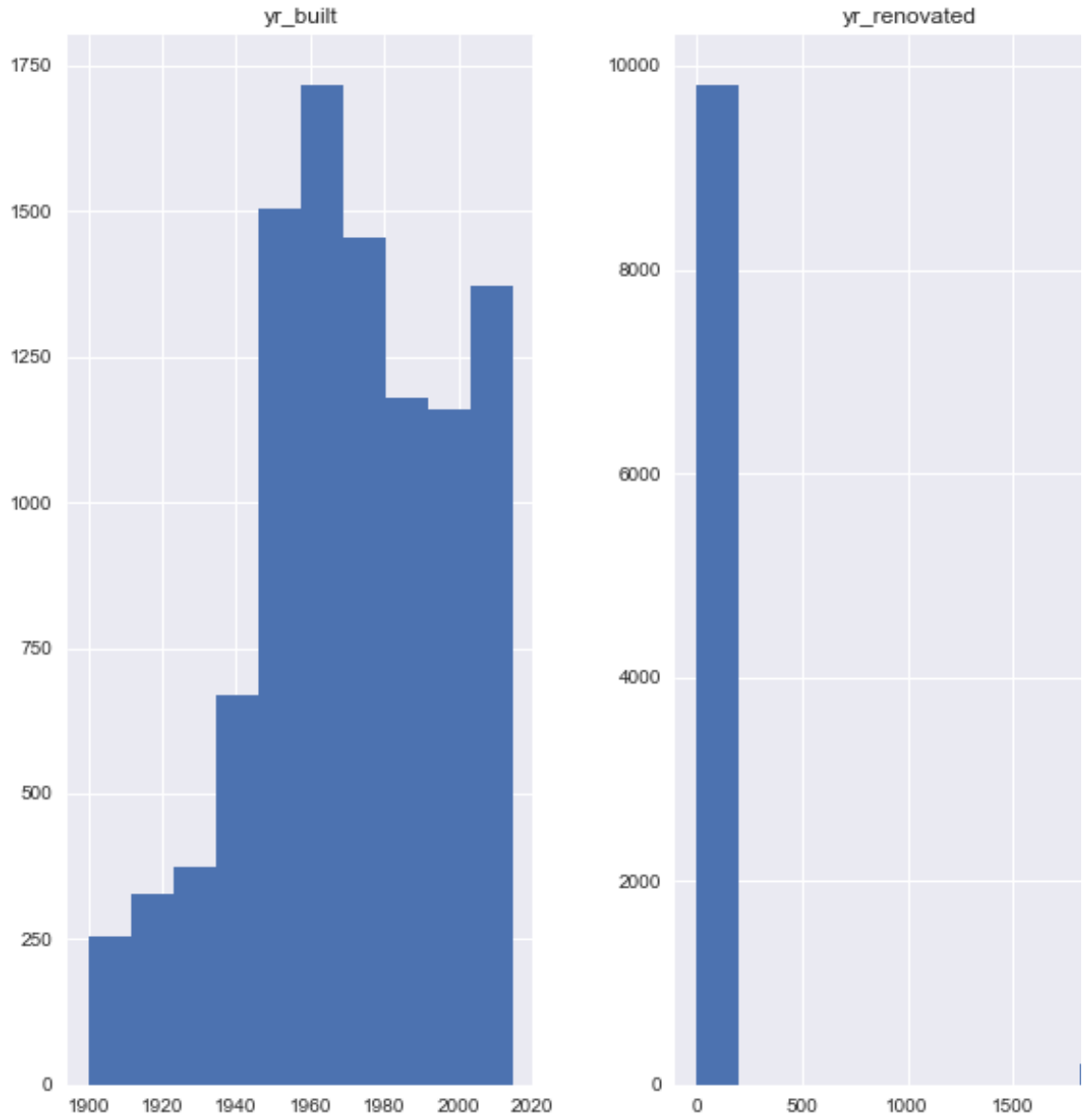
```
1164794
```

In [113]:

```
df6_low[['yr_built', 'yr_renovated']].hist(figsize  = [10, 10])
```

Out[113]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1a3b02
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a3c0e
>]],
      dtype=object)
```

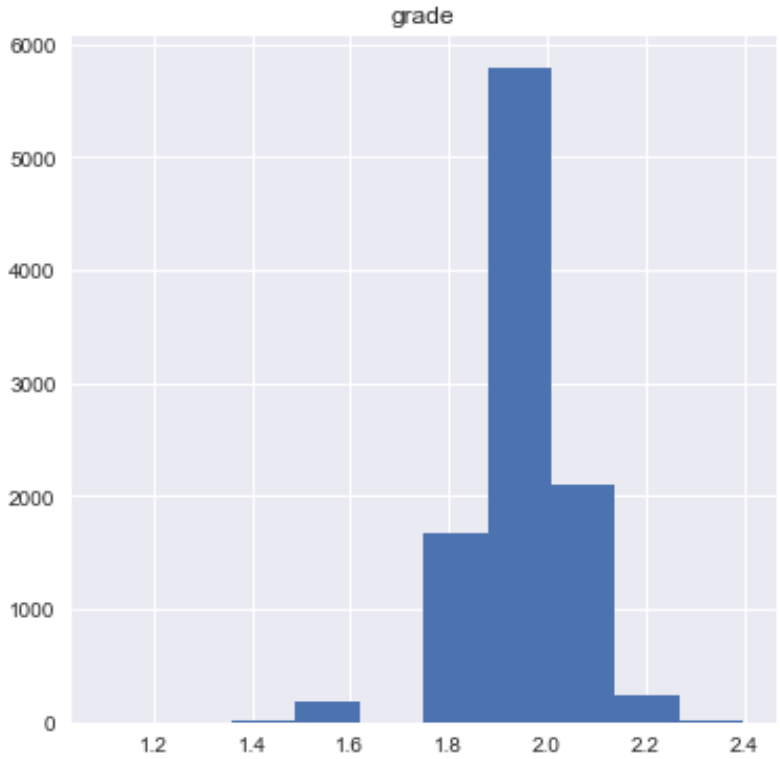


In [213]:

```
#Testing how log transmations would change feature distributions
#Yr_built seems to become more normal
data_log = pd.DataFrame([])
data_log['grade'] = np.log(df6_low['grade'])
data_log.hist(figsize  = [6, 6])
```

Out[213]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1a41e7
>]],
      dtype=object)
```



In [214]:

```
#Transforming(log transformation) and min max scaling grade
df0 = pd.DataFrame()
log_grade = data_log['grade']
scaled_grade = (log_grade - min(log_grade)) / (max(log_grade) - m
df0['sc_grade'] = scaled_grade

#Scaling the rest features so they are all on the same scale.
scaled_bath = (df6_low['bathrooms'] - min(df6_low['bathrooms']))
df0['sc_bath'] = scaled_bath

scaled_bed = (df6_low['bedrooms'] - min(df6_low['bedrooms'])) / (
df0['sc_bed'] = scaled_bed

scaled_cond = (df6_low['condition'] - min(df6_low['condition']))
df0['sc_cond'] = scaled_cond

scaled_floor = (df6_low['floors'] - min(df6_low['floors'])) / (ma
df0['sc_floor'] = scaled_floor

scaled_view = (df6_low['view'] - min(df6_low['view'])) / (max(df6
df0['sc_view'] = scaled_view

scaled_waterfront = (df6_low['waterfront'] - min(df6_low['waterfr
df0['sc_waterfront'] = scaled_waterfront

scaled_living = (df6_low['sqft_living'] - min(df6_low['sqft_livin
df0['sc_living'] = scaled_living

scaled_living15 = (df6_low['sqft_living15'] - min(df6_low['sqft_l
df0['sc_living15'] = scaled_living15

scaled_lot15 = (df6_low['sqft_lot15'] - min(df6_low['sqft_lot15']
df0['sc_lot15'] = scaled_lot15

scaled_lot = (df6_low['sqft_lot'] - min(df6_low['sqft_lot'])) / (
df0['sc_lot'] = scaled_lot

scaled_above = (df6_low['sqft_above'] - min(df6_low['sqft_above']
df0['sc_above'] = scaled_above

scaled_basement = (df6_low['sqft_basement'] - min(df6_low['sqft_b
df0['sc_basement'] = scaled_basement
```

In [216]:

```
#Adding original price, year built and year renovated without sca
df0['price'] = df6_low['price']
df0['yr_built'] = df6_low['yr_built']
df0['lat'] = df6_low['lat']
df0['zipcode'] = df6_low['zipcode']
```

In [217]:

```
#Inspect the new data frame (df0)
df0.head()
```

Out[217]:

| | sc_grade | sc_bath | sc_bed | sc_cond | sc_floor | sc_view | sc_waterfront | sc_living |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.652127 | 0.166667 | 0.285714 | 0.5 | 0.0 | 0.0 | 0.0 | 0.204030 |
| **2** | 0.533484 | 0.166667 | 0.142857 | 0.5 | 0.0 | 0.0 | 0.0 | 0.100756 |
| **6** | 0.652127 | 0.583333 | 0.285714 | 0.5 | 0.4 | 0.0 | 0.0 | 0.338791 |
| **7** | 0.652127 | 0.333333 | 0.285714 | 0.5 | 0.0 | 0.0 | 0.0 | 0.173804 |
| **8** | 0.652127 | 0.166667 | 0.285714 | 0.5 | 0.0 | 0.0 | 0.0 | 0.355164 |

In [219]:

```
train, test = train_test_split(df0, test_size=0.25, random_state=
```

In [220]:

```
train.shape
```

Out[220]:

```
(7509, 17)
```

In [224]:

```
outcome = 'price'
predictors = train.drop(['price', 'sc_waterfront', 'sc_basement',
                         'sc_bath', 'sc_bed', 'lat', 'zipcode'], a
pred_sum = "+".join(predictors)
formula = outcome + "~" + pred_sum
model = ols(formula= formula, data=train).fit()
```

In [225]:

```
model.summary()
```

Out[225]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | price | **R-squared:** | 0.186 |
| **Model:** | OLS | **Adj. R-squared:** | 0.185 |
| **Method:** | Least Squares | **F-statistic:** | 214.7 |
| **Date:** | Sun, 29 Nov 2020 | **Prob (F-statistic):** | 0.00 |
| **Time:** | 14:50:00 | **Log-Likelihood:** | -93863. |
| **No. Observations:** | 7509 | **AIC:** | 1.877e+05 |
| **Df Residuals:** | 7500 | **BIC:** | 1.878e+05 |
| **Df Model:** | 8 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **Intercept** | 1.166e+06 | 7.49e+04 | 15.574 | 0.000 | 1.02e+06 | 1.31e+06 |
| **sc_grade** | 2.341e+05 | 1.2e+04 | 19.442 | 0.000 | 2.1e+05 | 2.58e+05 |
| **sc_cond** | 7482.9065 | 4963.447 | 1.508 | 0.132 | -2246.841 | 1.72e+04 |
| **sc_floor** | 6.211e+04 | 4538.574 | 13.685 | 0.000 | 5.32e+04 | 7.1e+04 |
| **sc_view** | 3.178e+04 | 7784.313 | 4.083 | 0.000 | 1.65e+04 | 4.7e+04 |
| **sc_living** | 6.554e+04 | 7815.771 | 8.386 | 0.000 | 5.02e+04 | 8.09e+04 |
| **sc_living15** | 5.761e+04 | 7877.165 | 7.313 | 0.000 | 4.22e+04 | 7.3e+04 |
| **sc_lot** | 1.706e+05 | 3.58e+04 | 4.771 | 0.000 | 1e+05 | 2.41e+05 |
| **yr_built** | -538.6158 | 39.340 | -13.691 | 0.000 | -615.733 | -461.499 |

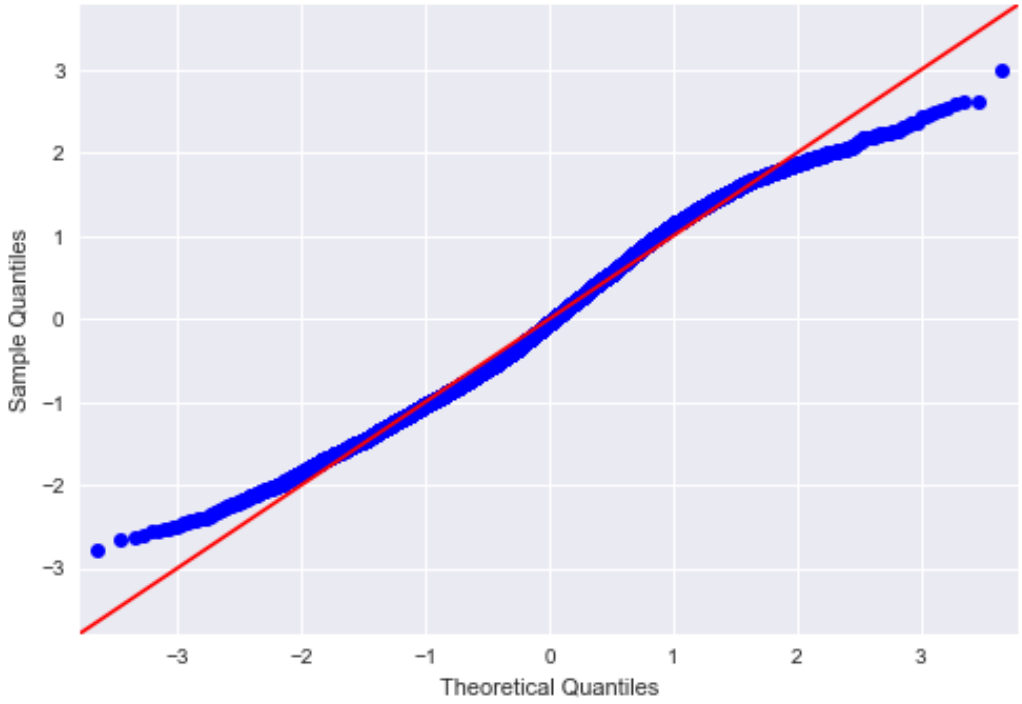
| | | | |
|---|---|---|---|
| **Omnibus:** | 419.509 | **Durbin-Watson:** | 1.972 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 164.000 |
| **Skew:** | 0.072 | **Prob(JB):** | 2.44e-36 |
| **Kurtosis:** | 2.291 | **Cond. No.** | 1.97e+05 |

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifiec
[2] The condition number is large, 1.97e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [226]:

```
# QQ Plot
residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', f
fig.show()
```

```
/Users/MZhang/opt/anaconda3/envs/learn-env/lib/python3.6/site-pac
s/ipykernel_launcher.py:4: UserWarning: Matplotlib is currently u
module://ipykernel.pylab.backend_inline, which is a non-GUI backe
so cannot show the figure.
  after removing the cwd from sys.path.
```

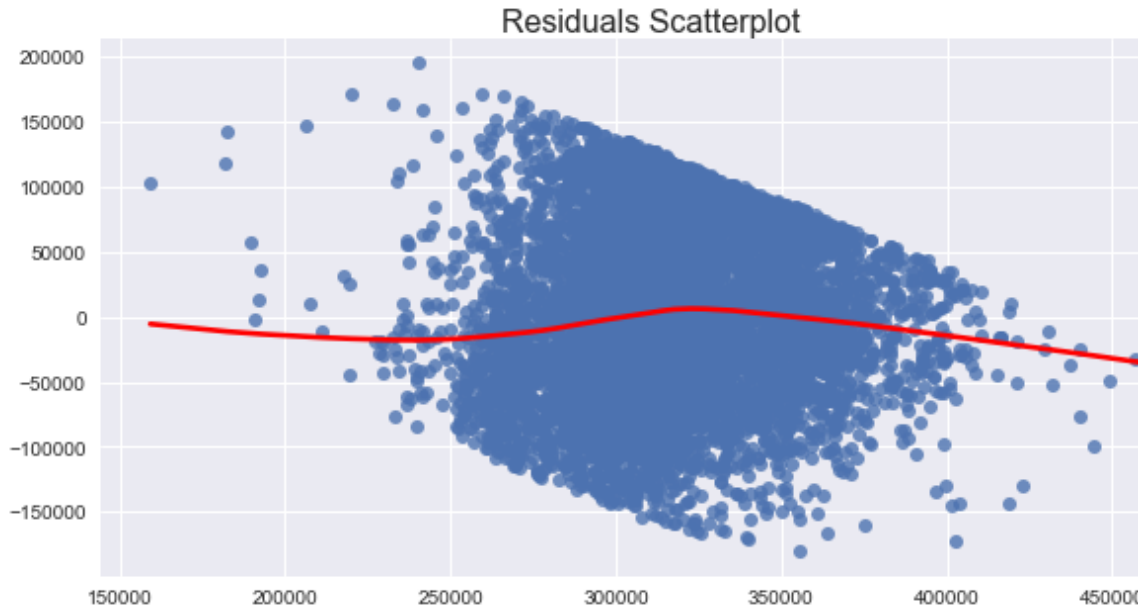


In [227]:

```
plt.figure(figsize=(10,5))
sns.regplot(x=model.predict(), y=model.resid, lowess=True, line_k
plt.title('Residuals Scatterplot', fontsize=16, y=.99)
```

Out[227]:

```
Text(0.5, 0.99, 'Residuals Scatterplot')
```




## ### Approach D

```
At this point (Saturday afternoon - df7), we thought more about a
variables we had been leaving out - location - lat, long, zipcode
looking at maps of where the sales were located, we could see a p
relationship between home price and latitude. So we added this va
predictor and boom, our R2 shot upto 0.492, and the t-score for
(the highest of the 10 predictor variables). So we seemed to be o
path.

Decided to create bands for latitude. Based on the map and knowle
County, decided on 6 equal bands and used the .cut method to crea
created dummies for lat using the cuts. And after creating them r
needed to re-name the columns. We did a similar procedure for yr_
6 bands of 20 year intervals and named the columns to represent h
old the homes are.
```

In [228]:

```
# Create an interaction term from 2 variables: to avoid multicoll
df6_low['bed_bath'] = (df6_low['bedrooms'] * df6_low['bathrooms']
df6_low.head()
```

```
/Users/MZhang/opt/anaconda3/envs/learn-env/lib/python3.6/site-pac
s/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pa
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
 (http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing
l#returning-a-view-versus-a-copy)
```

Out[228]:

| id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors | wate |
|---|---|---|---|---|---|---|---|---|
| 129300520 | 10/13/2014 | 221900.0 | 3 | 1.00 | 1180 | 5650 | 1.0 | |
| 631500400 | 2/25/2015 | 180000.0 | 2 | 1.00 | 770 | 10000 | 1.0 | |
| 321400060 | 6/27/2014 | 257500.0 | 3 | 2.25 | 1715 | 6819 | 2.0 | |
| 008000270 | 1/15/2015 | 291850.0 | 3 | 1.50 | 1060 | 9711 | 1.0 | |
| 414600126 | 4/15/2015 | 229500.0 | 3 | 1.00 | 1780 | 7470 | 1.0 | |

In [ ]:

```
# Try and create bands of "lat" with .cut method? lat vaules rang
# example: df_ages['age_by_decade'] = pd.cut(x=df_ages['age'], bi
```

In [229]:

```
pd.cut(df6_low['lat'], bins=6)
```

Out[229]:

```
0          (47.467, 47.57]
2         (47.674, 47.778]
6          (47.26, 47.363]
7         (47.363, 47.467]
8          (47.467, 47.57]
                ...
21592     (47.674, 47.778]
21593      (47.467, 47.57]
21594      (47.57, 47.674]
21595      (47.467, 47.57]
21596      (47.57, 47.674]
Name: lat, Length: 10013, dtype: category
Categories (6, interval[float64]): [(47.155, 47.26] < (47.26, 47.
< (47.363, 47.467] < (47.467, 47.57] < (47.57, 47.674] < (47.674,
778]]
```

In [230]:

```
pd.cut(df6_low['lat'], bins=6).value_counts()
```

Out[230]:

```
(47.467, 47.57]     2782
(47.674, 47.778]    2193
(47.363, 47.467]    2133
(47.26, 47.363]     2081
(47.57, 47.674]      591
(47.155, 47.26]      233
Name: lat, dtype: int64
```

In [231]:

```
df6_low['lat_bands'] = pd.cut(df6_low['lat'], bins=6)
df6_low.head(2)
```

```
/Users/MZhang/opt/anaconda3/envs/learn-env/lib/python3.6/site-pac
s/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pa
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
 (http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing
l#returning-a-view-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

Out[231]:

| | id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors |
|---|---|---|---|---|---|---|---|---|
| **0** | 7129300520 | 10/13/2014 | 221900.0 | 3 | 1.0 | 1180 | 5650 | 1.0 |
| **2** | 5631500400 | 2/25/2015 | 180000.0 | 2 | 1.0 | 770 | 10000 | 1.0 |

In [232]:

```
yrbuilt_labels = ['115', '95', '75', '55', '35', '15']
cut_bins = [1900, 1920, 1940, 1960, 1980, 2000, 2015]
df6_low['yrbuilt_bands'] = pd.cut(df6_low['yr_built'], bins=cut_b
df6_low.head(2)
```

```
/Users/MZhang/opt/anaconda3/envs/learn-env/lib/python3.6/site-pac
s/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pa
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
 (http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing
l#returning-a-view-versus-a-copy)
  This is separate from the ipykernel package so we can avoid doi
mports until
```

Out[232]:

| | id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors |
|---|---|---|---|---|---|---|---|---|
| **0** | 7129300520 | 10/13/2014 | 221900.0 | 3 | 1.0 | 1180 | 5650 | 1.0 |
| **2** | 5631500400 | 2/25/2015 | 180000.0 | 2 | 1.0 | 770 | 10000 | 1.0 |

In [233]:

```
df6_low.head()
```

Out[233]:

| | id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors |
|---|---|---|---|---|---|---|---|---|
| **0** | 7129300520 | 10/13/2014 | 221900.0 | 3 | 1.00 | 1180 | 5650 | 1.0 |
| **2** | 5631500400 | 2/25/2015 | 180000.0 | 2 | 1.00 | 770 | 10000 | 1.0 |
| **6** | 1321400060 | 6/27/2014 | 257500.0 | 3 | 2.25 | 1715 | 6819 | 2.0 |
| **7** | 2008000270 | 1/15/2015 | 291850.0 | 3 | 1.50 | 1060 | 9711 | 1.0 |
| **8** | 2414600126 | 4/15/2015 | 229500.0 | 3 | 1.00 | 1780 | 7470 | 1.0 |

In [ ]:

```
# consider turning lat_bands and yr_built_bands into dummies?
# bedrooms_dummies = pd.get_dummies(df1['bedrooms'], prefix='bed'
# df2 = pd.concat([df1, bedrooms_dummies], axis=1)
# df2.head()
```

In [234]:

```
yrbuilt_bands_dummies = pd.get_dummies(df6_low['yrbuilt_bands'],
df7 = pd.concat([df6_low, yrbuilt_bands_dummies], axis=1)
df7.head()
```

Out[234]:

| | id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors |
|---|---|---|---|---|---|---|---|---|
| **0** | 7129300520 | 10/13/2014 | 221900.0 | 3 | 1.00 | 1180 | 5650 | 1.0 |
| **2** | 5631500400 | 2/25/2015 | 180000.0 | 2 | 1.00 | 770 | 10000 | 1.0 |
| **6** | 1321400060 | 6/27/2014 | 257500.0 | 3 | 2.25 | 1715 | 6819 | 2.0 |
| **7** | 2008000270 | 1/15/2015 | 291850.0 | 3 | 1.50 | 1060 | 9711 | 1.0 |
| **8** | 2414600126 | 4/15/2015 | 229500.0 | 3 | 1.00 | 1780 | 7470 | 1.0 |

In [235]:

```
lat_bands_dummies = pd.get_dummies(df7['lat_bands'], prefix='lat'
df8 = pd.concat([df7, lat_bands_dummies], axis=1)
df8.head()
```

Out[235]:

| | id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors |
|---|---|---|---|---|---|---|---|---|
| **0** | 7129300520 | 10/13/2014 | 221900.0 | 3 | 1.00 | 1180 | 5650 | 1.0 |
| **2** | 5631500400 | 2/25/2015 | 180000.0 | 2 | 1.00 | 770 | 10000 | 1.0 |
| **6** | 1321400060 | 6/27/2014 | 257500.0 | 3 | 2.25 | 1715 | 6819 | 2.0 |
| **7** | 2008000270 | 1/15/2015 | 291850.0 | 3 | 1.50 | 1060 | 9711 | 1.0 |
| **8** | 2414600126 | 4/15/2015 | 229500.0 | 3 | 1.00 | 1780 | 7470 | 1.0 |

In [236]:

```
# Need to rename the lat dummies
df9 = df8.rename(columns={'lat_(47.26, 47.363]':'lat_2', 'lat_(47.
df9.head()
```

Out[236]:

| | id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors |
|---|---|---|---|---|---|---|---|---|
| **0** | 7129300520 | 10/13/2014 | 221900.0 | 3 | 1.00 | 1180 | 5650 | 1.0 |
| **2** | 5631500400 | 2/25/2015 | 180000.0 | 2 | 1.00 | 770 | 10000 | 1.0 |
| **6** | 1321400060 | 6/27/2014 | 257500.0 | 3 | 2.25 | 1715 | 6819 | 2.0 |
| **7** | 2008000270 | 1/15/2015 | 291850.0 | 3 | 1.50 | 1060 | 9711 | 1.0 |
| **8** | 2414600126 | 4/15/2015 | 229500.0 | 3 | 1.00 | 1780 | 7470 | 1.0 |

In [123]:

```
df9.describe()
```

Out[123]:

| | id | price | bedrooms | bathrooms | sqft_living | s |
|---|---|---|---|---|---|---|
| **count** | 1.001300e+04 | 10013.000000 | 10013.000000 | 10013.000000 | 10013.000000 | 1.0013( |
| **mean** | 4.532335e+09 | 315458.913512 | 3.138021 | 1.810846 | 1620.552382 | 1.1499( |
| **std** | 2.835075e+09 | 71806.431996 | 0.821046 | 0.634038 | 546.113648 | 2.7640( |
| **min** | 1.000102e+06 | 154000.000000 | 1.000000 | 0.500000 | 370.000000 | 5.7200( |
| **25%** | 2.120069e+09 | 259900.000000 | 3.000000 | 1.000000 | 1220.000000 | 5.2640( |
| **50%** | 3.834001e+09 | 319000.000000 | 3.000000 | 1.750000 | 1560.000000 | 7.5750( |
| **75%** | 7.262200e+09 | 375000.000000 | 4.000000 | 2.500000 | 1960.000000 | 9.7500( |
| **max** | 9.900000e+09 | 438000.000000 | 8.000000 | 3.500000 | 4340.000000 | 1.1647( |

In [237]:

```
#Doing some more outlier removal
# Start with removal of sqft_living
df11 = df9.loc[df9['sqft_living'] < 3100]
df11.shape
```

Out[237]:

```
(9900, 34)
```

In [238]:

```
df11 = df11.loc[df11['grade'] < 10]
df11.shape
```

Out[238]:

```
(9892, 34)
```

In [239]:

```
df11 = df11.loc[df11['grade'] > 4]
df11.shape
```

Out[239]:

```
(9875, 34)
```

In [240]:

```
df11 = df11.loc[df11['sqft_lot'] < 50000]
df11.shape
```

Out[240]:

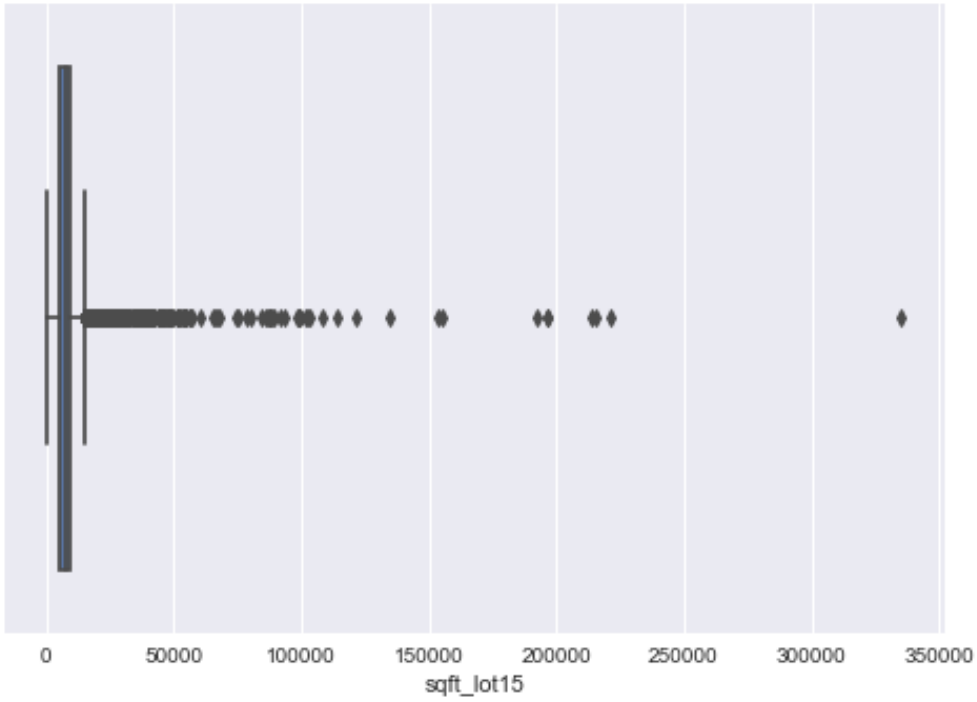
```
(9653, 34)
```

In [128]:

```
sns.boxplot(x=df11['sqft_lot15'])
```

Out[128]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a3c53be80>
```

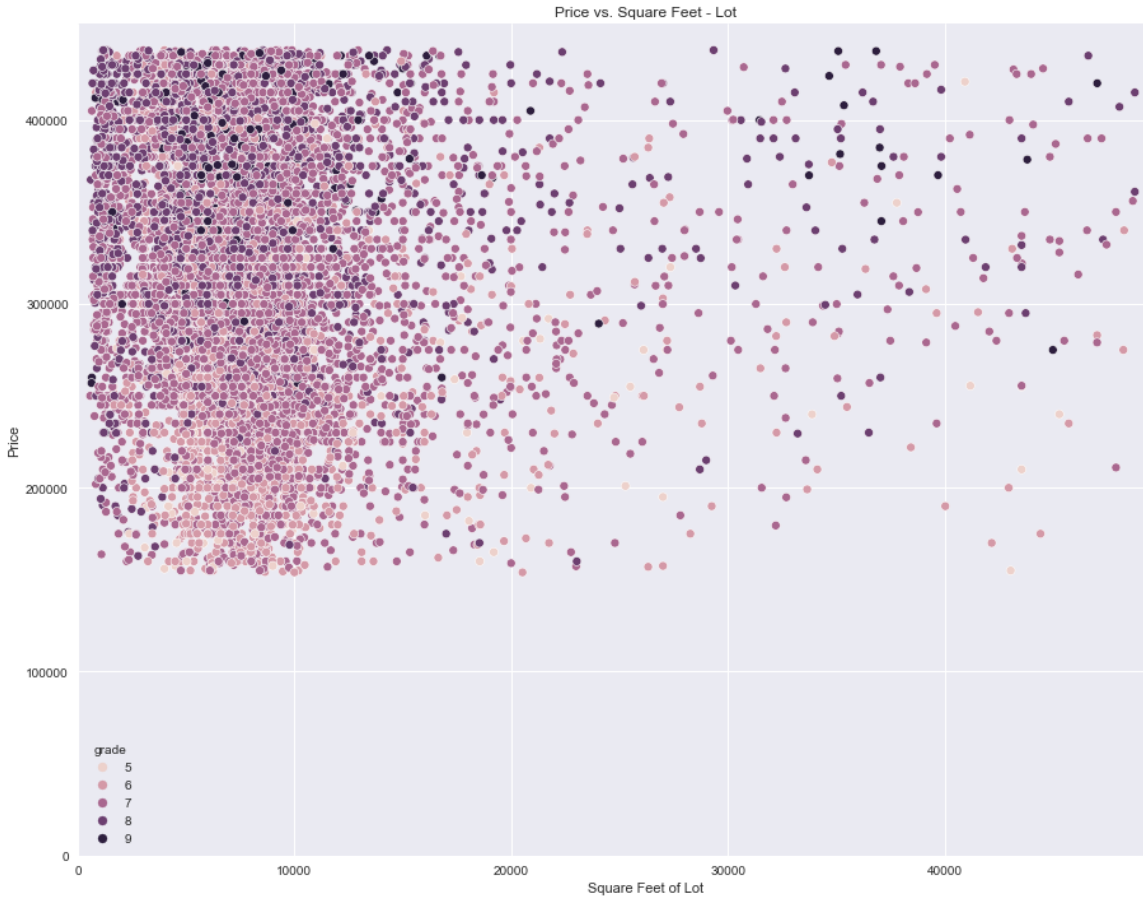


In [129]:

```
plt.figure(figsize=(16,12))
ax = sns.scatterplot(data=df11, x="sqft_lot", y="price", hue="gra
plt.xlim(0,None)
plt.ylim(0,None)
plt.title("Price vs. Square Feet - Lot")
plt.xlabel("Square Feet of Lot")
plt.ylabel("Price")
```

Out[129]:

```
Text(0, 0.5, 'Price')
```



In [241]:

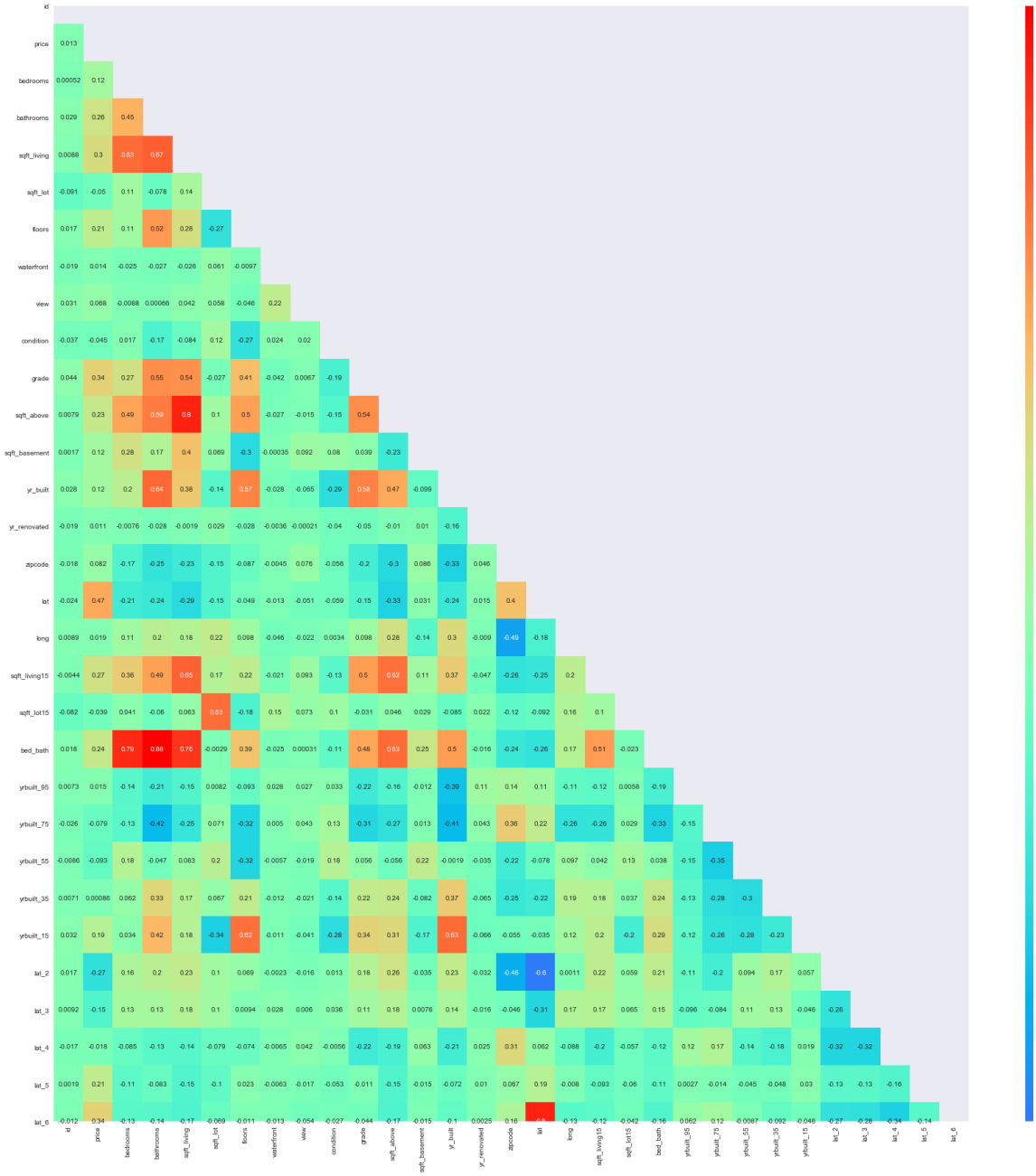
```
corr = df11.corr()
display(corr)
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
plt.figure(figsize=(30,30))
sns.heatmap(corr, cmap='rainbow', mask = mask, annot=True, center
```

| | id | price | bedrooms | bathrooms | sqft_living | sqft_lot | f |
|---|---|---|---|---|---|---|---|
| **id** | 1.000000 | 0.012701 | 0.000521 | 0.028628 | 0.008800 | -0.090754 | 0.01 |
| **price** | 0.012701 | 1.000000 | 0.119963 | 0.263198 | 0.298315 | -0.050016 | 0.20 |
| **bedrooms** | 0.000521 | 0.119963 | 1.000000 | 0.446608 | 0.633950 | 0.108687 | 0.11 |
| **bathrooms** | 0.028628 | 0.263198 | 0.446608 | 1.000000 | 0.666431 | -0.077631 | 0.51 |
| **sqft_living** | 0.008800 | 0.298315 | 0.633950 | 0.666431 | 1.000000 | 0.141563 | 0.28 |
| **sqft_lot** | -0.090754 | -0.050016 | 0.108687 | -0.077631 | 0.141563 | 1.000000 | -0.27 |
| **floors** | 0.016573 | 0.209542 | 0.110903 | 0.519666 | 0.281426 | -0.272702 | 1.00 |
| **waterfront** | -0.019366 | 0.013981 | -0.024631 | -0.026897 | -0.025957 | 0.060897 | -0.00 |
| **view** | 0.031159 | 0.068262 | -0.008831 | 0.000663 | 0.041883 | 0.058470 | -0.04 |
| **condition** | -0.037290 | -0.044942 | 0.017182 | -0.172850 | -0.083736 | 0.124621 | -0.27 |
| **grade** | 0.044445 | 0.342969 | 0.270637 | 0.549476 | 0.536107 | -0.026897 | 0.41 |
| **sqft_above** | 0.007930 | 0.234829 | 0.485041 | 0.594834 | 0.796582 | 0.103424 | 0.49 |
| **sqft_basement** | 0.001676 | 0.120946 | 0.278597 | 0.169247 | 0.396032 | 0.068711 | -0.29 |
| **yr_built** | 0.028258 | 0.119461 | 0.200054 | 0.639398 | 0.376713 | -0.141014 | 0.56 |
| **yr_renovated** | -0.019023 | 0.010762 | -0.007646 | -0.027862 | -0.001921 | 0.028790 | -0.02 |
| **zipcode** | -0.017512 | 0.081948 | -0.166810 | -0.254153 | -0.233638 | -0.154178 | -0.08 |
| **lat** | -0.023926 | 0.467245 | -0.214175 | -0.238216 | -0.293728 | -0.151957 | -0.04 |
| **long** | 0.008892 | 0.019036 | 0.109096 | 0.198068 | 0.176384 | 0.223821 | 0.09 |
| **sqft_living15** | -0.004420 | 0.271706 | 0.359403 | 0.488400 | 0.652313 | 0.173631 | 0.22 |
| **sqft_lot15** | -0.081886 | -0.038577 | 0.041243 | -0.060343 | 0.063176 | 0.629619 | -0.17 |
| **bed_bath** | 0.018485 | 0.237342 | 0.786370 | 0.880520 | 0.756676 | -0.002921 | 0.38 |
| **yrbuilt_95** | 0.007328 | 0.014695 | -0.140582 | -0.207742 | -0.151041 | 0.008153 | -0.09 |
| **yrbuilt_75** | -0.025831 | -0.079179 | -0.126828 | -0.415139 | -0.248746 | 0.070584 | -0.32 |
| **yrbuilt_55** | -0.008599 | -0.093328 | 0.176774 | -0.047210 | 0.083160 | 0.197154 | -0.32 |
| **yrbuilt_35** | 0.007115 | 0.000858 | 0.062389 | 0.330134 | 0.173558 | 0.066814 | 0.21 |
| **yrbuilt_15** | 0.031997 | 0.185903 | 0.033980 | 0.419781 | 0.184797 | -0.337161 | 0.61 |

| | id | price | bedrooms | bathrooms | sqft_living | sqft_lot | f |
|---|---|---|---|---|---|---|---|
| **lat_2** | 0.016825 | -0.274995 | 0.157747 | 0.195817 | 0.225169 | 0.102014 | 0.06 |
| **lat_3** | 0.009233 | -0.150299 | 0.134342 | 0.134336 | 0.176768 | 0.100800 | 0.00 |
| **lat_4** | -0.016785 | -0.018490 | -0.084713 | -0.134790 | -0.137059 | -0.078835 | -0.07 |
| **lat_5** | 0.001941 | 0.212865 | -0.114023 | -0.083259 | -0.149577 | -0.102574 | 0.02 |
| **lat_6** | -0.011860 | 0.342140 | -0.130993 | -0.137605 | -0.166517 | -0.069326 | -0.01 |

Out[241]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a381f9828>
```



In [134]:

```
df11.head(2)
```

Out[134]:

| id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors | water |
|---|---|---|---|---|---|---|---|---|
| 29300520 | 10/13/2014 | 221900.0 | 3 | 1.0 | 1180 | 5650 | 1.0 | |
| 31500400 | 2/25/2015 | 180000.0 | 2 | 1.0 | 770 | 10000 | 1.0 | |

In [258]:

```
#Drop unnecessary columns, looking at correlations and what we do
df12 = df11.drop(['id', 'zipcode', 'bedrooms', 'bathrooms', 'lat_
```

In [136]:

```
df12.head(2)
```

Out[136]:

| ilt | yr_renovated | lat | long | sqft_living15 | sqft_lot15 | yrbuilt_bands | yrbuilt_95 | yrl |
|---|---|---|---|---|---|---|---|---|
| 55 | 0.0 | 47.5112 | -122.257 | 1340 | 5650 | 75 | 0 | |
| 33 | 0.0 | 47.7379 | -122.233 | 2720 | 8062 | 95 | 1 | |

In [259]:

```
df12 = df12.drop(['date', 'sqft_lot', 'floors', 'waterfront', 'sq
                  'yr_renovated', 'lat', 'long', 'sqft_living15',
                  'yrbuilt_95', 'yrbuilt_15', 'lat_2', 'yrbuilt_ba
```

In [260]:

```
# Create train, test split
train, test = train_test_split(df12, test_size=0.25, random_state
print(train.shape, test.shape)
```

```
(7239, 12) (2414, 12)
```

In [261]:

```
train.head(2)
```

Out[261]:

| | price | sqft_living | view | condition | grade | yrbuilt_75 | yrbuilt_55 | yrbuilt_35 | la |
|---|---|---|---|---|---|---|---|---|---|
| **5740** | 225000.0 | 1250 | 0.0 | 3 | 7 | 0 | 1 | 0 | |
| **17283** | 364000.0 | 1020 | 0.0 | 5 | 6 | 1 | 0 | 0 | |

In [262]:

```
#Running the final model after various models and experimentation
outcome = 'price'
predictors = train.drop(['price'], axis=1)
pred_sum = "+".join(predictors)
formula = outcome + "~" + pred_sum
model = ols(formula= formula, data=train).fit()
```

In [263]:

```
model.summary()
```

Out[263]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | price | **R-squared:** | 0.514 |
| **Model:** | OLS | **Adj. R-squared:** | 0.513 |
| **Method:** | Least Squares | **F-statistic:** | 695.3 |
| **Date:** | Sun, 29 Nov 2020 | **Prob (F-statistic):** | 0.00 |
| **Time:** | 15:16:04 | **Log-Likelihood:** | -88591. |
| **No. Observations:** | 7239 | **AIC:** | 1.772e+05 |
| **Df Residuals:** | 7227 | **BIC:** | 1.773e+05 |
| **Df Model:** | 11 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **Intercept** | -3.23e+04 | 7900.461 | -4.088 | 0.000 | -4.78e+04 | -1.68e+04 |
| **sqft_living** | 50.9758 | 1.416 | 35.997 | 0.000 | 48.200 | 53.752 |
| **view** | 1.329e+04 | 1612.491 | 8.241 | 0.000 | 1.01e+04 | 1.64e+04 |
| **condition** | 1.222e+04 | 983.845 | 12.418 | 0.000 | 1.03e+04 | 1.41e+04 |
| **grade** | 2.49e+04 | 1011.555 | 24.611 | 0.000 | 2.29e+04 | 2.69e+04 |
| **yrbuilt_75** | -1.994e+04 | 1708.018 | -11.674 | 0.000 | -2.33e+04 | -1.66e+04 |
| **yrbuilt_55** | -2.094e+04 | 1654.246 | -12.660 | 0.000 | -2.42e+04 | -1.77e+04 |
| **yrbuilt_35** | -7050.8781 | 1810.686 | -3.894 | 0.000 | -1.06e+04 | -3501.404 |
| **lat_3** | 2.623e+04 | 1800.379 | 14.570 | 0.000 | 2.27e+04 | 2.98e+04 |
| **lat_4** | 6.812e+04 | 1794.752 | 37.953 | 0.000 | 6.46e+04 | 7.16e+04 |
| **lat_5** | 1.354e+05 | 2806.260 | 48.242 | 0.000 | 1.3e+05 | 1.41e+05 |
| **lat_6** | 1.168e+05 | 1857.675 | 62.873 | 0.000 | 1.13e+05 | 1.2e+05 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 67.183 | **Durbin-Watson:** | 1.978 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 94.820 |
| **Skew:** | -0.113 | **Prob(JB):** | 2.57e-21 |
| **Kurtosis:** | 3.513 | **Cond. No.** | 2.29e+04 |

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specifiec
[2] The condition number is large, 2.29e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

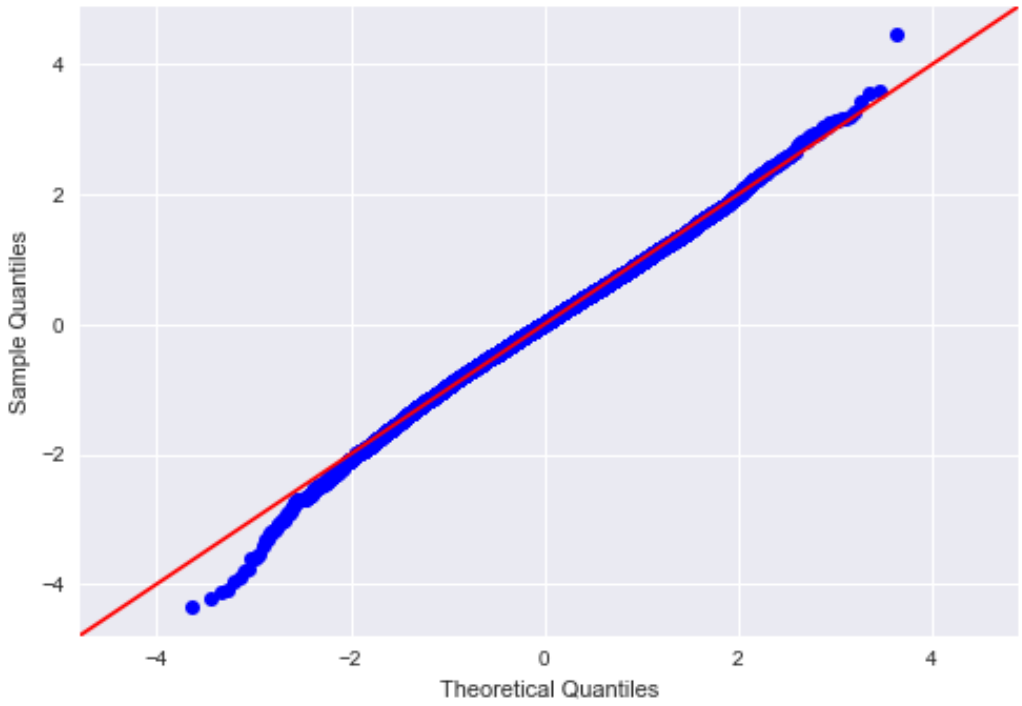
#### Interpretation of our FINAL Model:

```
We ran our training data set with 7,212 records, and used 11 pred
to try and predict home prices (target variable). We got an r2 va
which means that our model (predictor variables) can account for
variation in home prices. We found that all of the predictor vari
values and were therefore significant. By looking at the t-values
relative importance of each predictor - with the latitude bands a
being the most important. We can use the coefficients for each pr
illustrate our model as well.  For example:
Size: Each additional sq. ft of living area will add $52 to the h
all other features are held constant.
Location: A home in latitude 5 (Seattle) will add ~$135K to the h
a home in latitude 1 (south).
```

In [264]:

```
# QQ-Plot
import scipy.stats as stats
residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', f
fig.show()
```

```
/Users/MZhang/opt/anaconda3/envs/learn-env/lib/python3.6/site-pac
s/ipykernel_launcher.py:5: UserWarning: Matplotlib is currently u
module://ipykernel.pylab.backend_inline, which is a non-GUI backe
so cannot show the figure.
  """
```

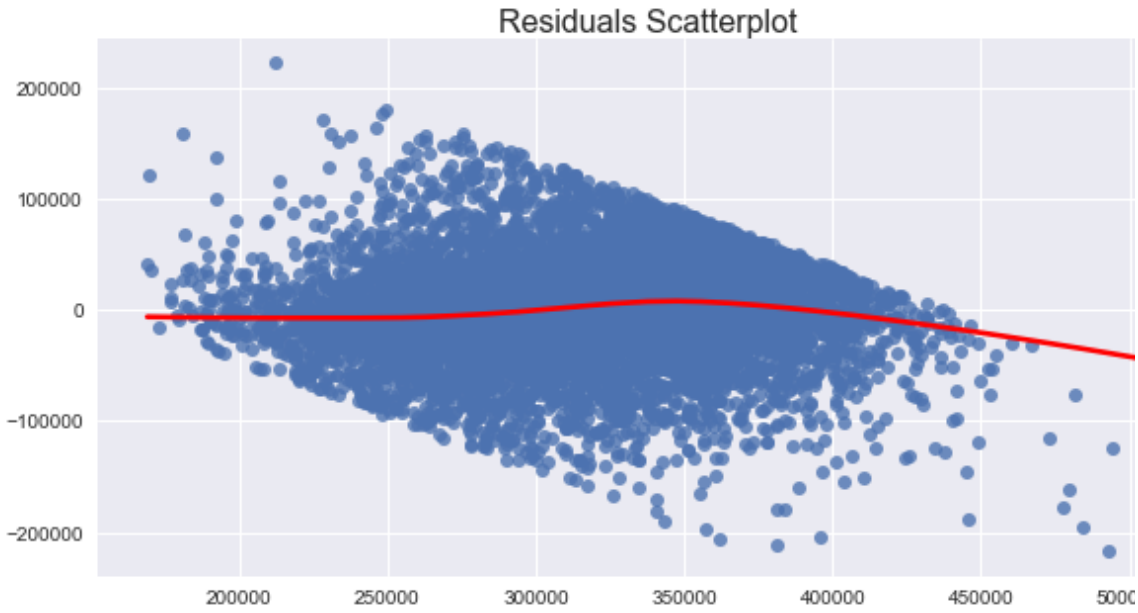


In [265]:

```
plt.figure(figsize=(10,5))
sns.regplot(x=model.predict(), y=model.resid, lowess=True, line_k
plt.title('Residuals Scatterplot', fontsize=16, y=.99)
```

Out[265]:

```
Text(0.5, 0.99, 'Residuals Scatterplot')
```




In [266]:

```
# Conduct a final check of RMSE (after 3 models with train)
# Split data into x and y versions of train and test
y_test = test[['price']]
X_test = test.drop(['price'],axis=1)
y_train = train[['price']]
X_train = train.drop(['price'],axis=1)
```

In [267]:

```
# Try and baseline this to the test data set (before changes)
from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
linreg.fit(X_train, y_train)
y_hat_train = linreg.predict(X_train)
y_hat_test = linreg.predict(X_test)
```

In [268]:

```
from sklearn.metrics import mean_squared_error
train_mse = mean_squared_error(y_train, y_hat_train)
test_mse = mean_squared_error(y_test, y_hat_test)
print('Train Root Mean Squarred Error:', train_mse**0.5)
print('Test Root Mean Squarred Error:', test_mse**0.5)
```

```
Train Root Mean Squarred Error: 49966.8160225452
Test Root Mean Squarred Error: 49307.57685468789
```

In [274]:

```
#Prediction plug and play
df12.drop(['price'], axis=1)
linreg.predict(np.array([2000,0,4,7,0,0,1,0,0,0,1]).reshape(1,-1)
```

Out[274]:

```
array([[402540.19042378]])
```

In [275]:

```
df14 = df12.drop(['price'], axis=1)
```

In [276]:

```
Y=df12[['price']]
```

In [277]:

```
X=df14
```

In [278]:

```
X.head(2)
```

Out[278]:

| | sqft_living | view | condition | grade | yrbuilt_75 | yrbuilt_55 | yrbuilt_35 | lat_3 | lat_4 | lat_ |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1180 | 0.0 | 3 | 7 | 1 | 0 | 0 | 0 | 1 | |
| **2** | 770 | 0.0 | 3 | 6 | 0 | 0 | 0 | 0 | 0 | |

In [271]:

```
#Running cross validation
from sklearn.model_selection import train_test_split, cross_val_s
from sklearn.metrics import mean_squared_error, r2_score
```

In [279]:

```
# Uses original X and Y data - not train or test
model = LinearRegression()
scores = cross_val_score(model, X, Y, cv=8, scoring="r2")
print (scores)
```

```
[0.50012948 0.531374   0.50388437 0.48114011 0.51559394 0.5295758
 0.53672391 0.46941402]
```

In [280]:

```
scores.mean()
```

Out[280]:

```
0.5084794602944305
```

In [ ]:

```
# Observations for r2:
# range 0.47 to 0.54
# average of 0.51
# This is inline with wht we have seen in our model summary with
# It tells us that our model can explain 51% of the variation in
```

In [282]:

```
model = LinearRegression()
scores = cross_val_score(model, X, Y, cv=8, scoring="neg_mean_squ
print (scores)
```

```
[-2.53752915e+09 -2.71652570e+09 -2.37940023e+09 -2.60151639e+09
 -2.50538115e+09 -2.38078604e+09 -2.50621657e+09 -2.31204144e+09]
```

In [284]:

```
rmse_scores = np.sqrt(-scores)
print (rmse_scores)
```

```
[50373.89356451 52120.30030025 48779.0962473  51005.06241948
 50053.78258873 48793.29914888 50062.12707274 48083.69206607]
```

In [285]:

```
rmse_scores.mean()
```

Out[285]:

```
49908.90667599447
```

In [ ]:

```
# Observations for RMSE:
# RMSE ranges from 48,083 to 52,120
# Mean RMSE is 49,932 for our model
```

## ### Additional Exploration Approach

```
For this model, log transform: grade, sqft_living, bedrooms, cond
And min/max scale necessary sqft features.
```

In [299]:

```
df20 = pd.DataFrame()
```

In [300]:

```
#Log transformating features to make the distributions more norma
df20['price'] = df12['price']
df20['grade'] = np.log(df11['grade'])
df20['condition'] = np.log(df11['condition'])
df20['floors'] = np.log(df11['floors'])
```

In [301]:

```
#Min Max Scaling SQFT features because they are on a larger scale
df20['sqft_living'] = (df11['sqft_living'] - min(df11['sqft_livin
df20['sqft_lot'] = (df11['sqft_lot'] - min(df11['sqft_lot'])) / (
df20['sqft_living15'] = (df11['sqft_living15'] - min(df11['sqft_l
```

In [324]:

```
#Adding the same (untouched) features we were using for the final
df20['yrbuilt_75'] = df12['yrbuilt_75']
df20['yrbuilt_55'] = df12['yrbuilt_55']
df20['yrbuilt_35'] = df12['yrbuilt_35']
df20['lat_3'] = df12['lat_3']
df20['lat_4'] = df12['lat_4']
df20['lat_5'] = df12['lat_5']
df20['lat_6'] = df12['lat_6']
df20['view'] = df12['view']
```

In [325]:

```
df20.head(2)
```

Out[325]:

| | price | grade | condition | floors | sqft_living | sqft_lot | sqft_living15 | yrbuilt_75 |
|---|---|---|---|---|---|---|---|---|
| **0** | 221900.0 | 1.945910 | 1.098612 | 0.0 | 0.297794 | 0.102885 | 0.268293 | 1 |
| **2** | 180000.0 | 1.791759 | 1.098612 | 0.0 | 0.147059 | 0.191020 | 0.689024 | 0 |

In [326]:

```
# Create train, test split
train, test = train_test_split(df20, test_size=0.25, random_state
print(train.shape, test.shape)
```

```
(7239, 17) (2414, 17)
```

In [328]:

```
#Model
from statsmodels.formula.api import ols
outcome = 'price'
predictors = train.drop(['price', 'sqft_lot', 'floors'], axis=1)
pred_sum = "+".join(predictors)
formula = outcome + "~" + pred_sum
model = ols(formula= formula, data=train).fit()
```

In [329]:

```
model.summary()
```

Out[329]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | price | **R-squared:** | 0.525 |
| **Model:** | OLS | **Adj. R-squared:** | 0.524 |
| **Method:** | Least Squares | **F-statistic:** | 569.5 |
| **Date:** | Sun, 29 Nov 2020 | **Prob (F-statistic):** | 0.00 |
| **Time:** | 16:01:43 | **Log-Likelihood:** | -88512. |
| **No. Observations:** | 7239 | **AIC:** | 1.771e+05 |
| **Df Residuals:** | 7224 | **BIC:** | 1.772e+05 |
| **Df Model:** | 14 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **Intercept** | -1.804e+05 | 1.5e+04 | -12.050 | 0.000 | -2.1e+05 | -1.51e+05 |
| **grade** | 1.56e+05 | 7552.021 | 20.659 | 0.000 | 1.41e+05 | 1.71e+05 |
| **condition** | 4.668e+04 | 3507.927 | 13.306 | 0.000 | 3.98e+04 | 5.36e+04 |
| **sqft_living** | 1.123e+05 | 4373.218 | 25.690 | 0.000 | 1.04e+05 | 1.21e+05 |
| **sqft_living15** | 7.829e+04 | 6180.574 | 12.666 | 0.000 | 6.62e+04 | 9.04e+04 |
| **yrbuilt_75** | -2.06e+04 | 2156.168 | -9.556 | 0.000 | -2.48e+04 | -1.64e+04 |
| **yrbuilt_55** | -2.276e+04 | 2295.254 | -9.917 | 0.000 | -2.73e+04 | -1.83e+04 |
| **yrbuilt_35** | -9209.4444 | 2541.128 | -3.624 | 0.000 | -1.42e+04 | -4228.091 |
| **lat_3** | 3.405e+04 | 4405.944 | 7.728 | 0.000 | 2.54e+04 | 4.27e+04 |
| **lat_4** | 7.835e+04 | 4374.360 | 17.911 | 0.000 | 6.98e+04 | 8.69e+04 |
| **lat_5** | 1.451e+05 | 4871.354 | 29.786 | 0.000 | 1.36e+05 | 1.55e+05 |
| **lat_6** | 1.258e+05 | 4417.867 | 28.475 | 0.000 | 1.17e+05 | 1.34e+05 |
| **view** | 1.161e+04 | 1615.373 | 7.190 | 0.000 | 8447.897 | 1.48e+04 |
| **yrbuilt_15** | -2756.8619 | 2601.781 | -1.060 | 0.289 | -7857.113 | 2343.389 |
| **lat_2** | 8929.6360 | 4405.332 | 2.027 | 0.043 | 293.897 | 1.76e+04 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 79.248 | **Durbin-Watson:** | 1.974 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 113.954 |
| **Skew:** | -0.129 | **Prob(JB):** | 1.80e-25 |

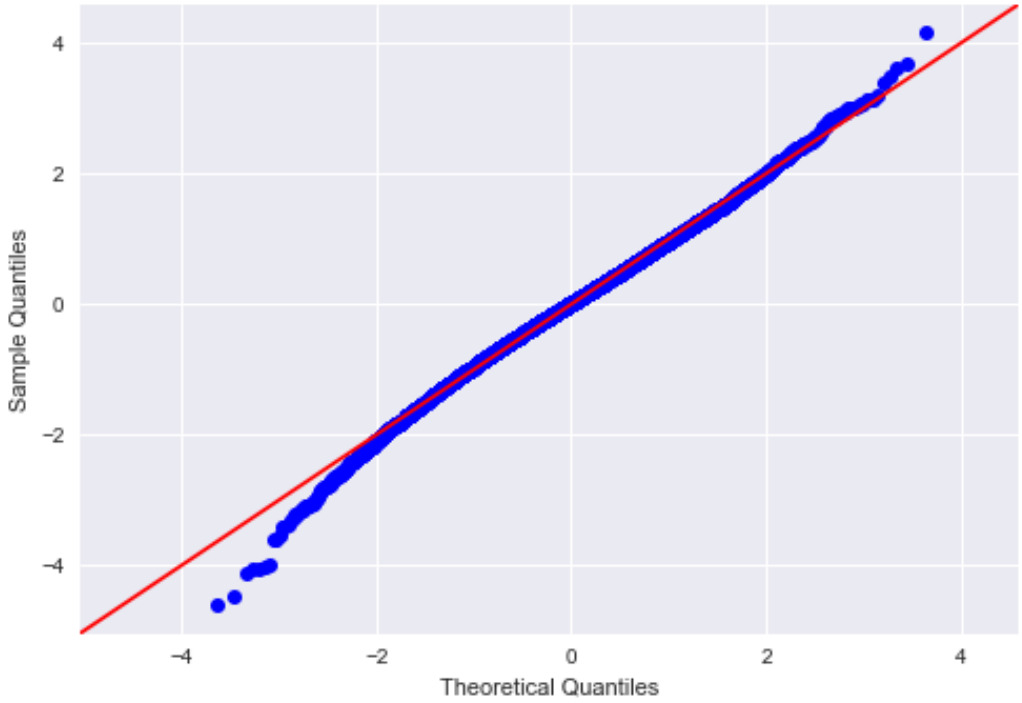
| | | | |
|---|---|---|---|
| **Kurtosis:** | 3.558 | **Cond. No.** | 76.6 |

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifiec

In [330]:

```
# QQ-Plot
import scipy.stats as stats
residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', f
fig.show()
```

```
/Users/MZhang/opt/anaconda3/envs/learn-env/lib/python3.6/site-pac
s/ipykernel_launcher.py:5: UserWarning: Matplotlib is currently u
module://ipykernel.pylab.backend_inline, which is a non-GUI backe
so cannot show the figure.
  """
```

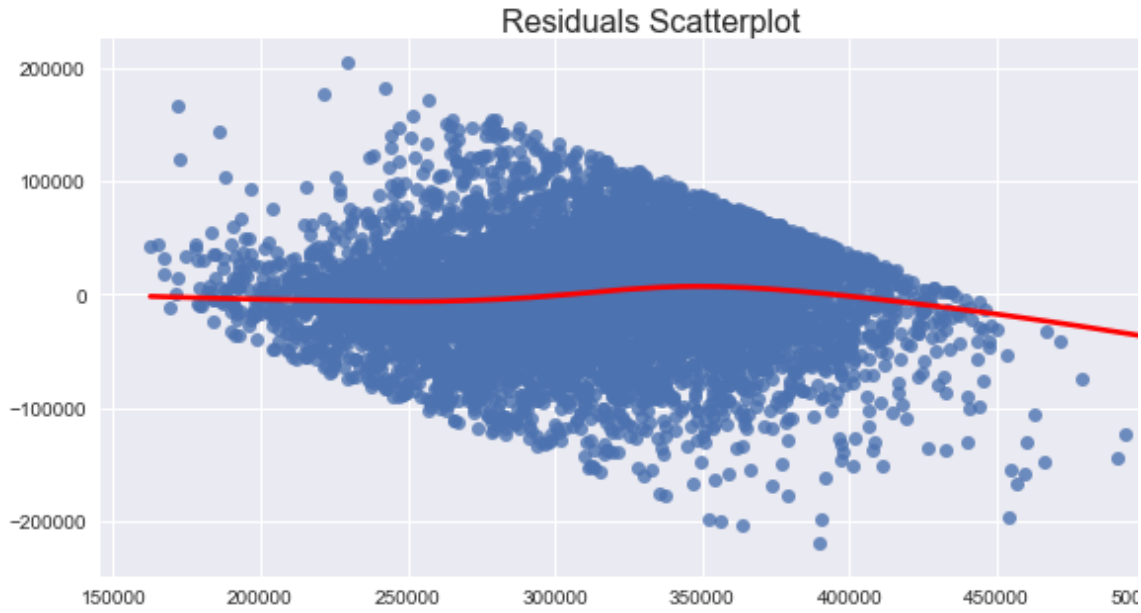


In [331]:

```
plt.figure(figsize=(10,5))
sns.regplot(x=model.predict(), y=model.resid, lowess=True, line_k
plt.title('Residuals Scatterplot', fontsize=16, y=.99)
```

Out[331]:

```
Text(0.5, 0.99, 'Residuals Scatterplot')
```




In [332]:

```
# Split data into x and y versions of train and test
y_test = test[['price']]
X_test = test.drop(['price'],axis=1)
y_train = train[['price']]
X_train = train.drop(['price'],axis=1)
```

In [333]:

```
from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
linreg.fit(X_train, y_train)
y_hat_train = linreg.predict(X_train)
y_hat_test = linreg.predict(X_test)
```

In [334]:

```
from sklearn.metrics import mean_squared_error
train_mse = mean_squared_error(y_train, y_hat_train)
test_mse = mean_squared_error(y_test, y_hat_test)
print('Train Root Mean Squarred Error:', train_mse**0.5)
print('Test Root Mean Squarred Error:', test_mse**0.5)
```

```
Train Root Mean Squarred Error: 49378.859501134444
Test Root Mean Squarred Error: 48823.981126282066
```

In [335]:

```
# Try cross validation
from sklearn.model_selection import train_test_split, cross_val_s
from sklearn.metrics import mean_squared_error, r2_score
```

In [337]:

```
Y2 = df20['price']
```

In [339]:

```
df21 = df20.drop(['price'], axis=1)
```

In [340]:

```
X2 = df21
```

In [341]:

```
model = LinearRegression()
scores = cross_val_score(model, X2, Y2, cv=8, scoring="r2")
print (scores)
```

```
[0.50646653 0.5381383  0.51829932 0.49811755 0.52074977 0.5375320
 0.54757934 0.48985976]
```

In [342]:

```
scores.mean()
```

Out[342]:

```
0.5195928218425727
```

```
-Observations for r2:
-range 0.49 to 0.55
-average of 0.52
-This is inline with wht we have seen in our model summary with t
-It tells us that our model can explain 52% of the variation in h
on our predictors (in the model)
```

In [346]:

```
model = LinearRegression()
scores = cross_val_score(model, X2, Y2, cv=8, scoring="neg_mean_s
print (scores)
```

```
[-2.50535993e+09 -2.67731448e+09 -2.31026524e+09 -2.51639302e+09
 -2.47871489e+09 -2.34052048e+09 -2.44749121e+09 -2.22294863e+09]
```

In [347]:

```
rmse_scores = np.sqrt(-scores)
print (rmse_scores)
```

```
[50053.57062343 51742.77223787 48065.21862463 50163.66232286
 49786.69391852 48378.9259681  49472.12561647 47148.15615257]
```

In [348]:

```
rmse_scores.mean()
```

Out[348]:

```
49351.39068305553
```

```
-Observations for RMSE:
-RMSE ranges from 47,148 to 51742
-Mean RMSE is 49351 for our model
-This means that our model, although it can predict house prices,
-we need to keep in mind that there is about $49K error rate (is
$49K?).
```

```
Interpreting log coefficients of additional model:
Just divide by 100, this represents the derivative or change in p
variable
So, Condition and Grade: So for 1 percent increase in condition,
$466 increase in price. ETC.
        *With all other variables constant.
```