Learn Git and GitHub without any code!

Using the Hello World guide, you'll start a branch, write comments, and open a pull request.

Read the guide

amandabgaeta / **King-County-Housing-Modeling**

forked from learn-co-curriculum/dsc-phase-2-project

Code | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

main

King-County-Housing-Modeling / **Notebook.ipynb**

Go to file | ...

amandabgaeta updates with final notebook and images

Latest commit `4b8c474` 36 minutes ago **History**

1 contributor

`26.7 MB`

Download

Final Project Submission

Please fill out:

- Student name: Amanda Gaeta
- Student pace: part time
- Scheduled project review date/time: February 8th, 2020 at 11am CST
- Instructor name: Lindsey Berlin

Introduction

COVID has made working from home a trend that will continue. With a job less dependent on living in a certain area, people are reassessing where they live whether that is within the same area or a completely different state or country. Another side effect of COVID has been the extreme reduction in interest rates, which have driven home purchases to skyrocketed.

This model has been constructed with both of these in mind, for those interested in buying a home in King County. Buyers should be able to progress decisioning on what zipcodes they can target in their home search along with what they can expect in relation to home features and quality within their budget. This should aid in their understanding on what they want to prioritize and/or sacrifice to get a home that fits their needs and is in their price range. Information derived can also point home buyers to research other things about the areas they are interesting like schooling for future family.

Notebook Summary

Data Processing

Below is a summary of the processing of the original dataset as well as functions defined.

Cleansing:

Nulls:

- Filled nulls for waterfront

Converted:

- date from string to datetime
- yr_renovated from float to int

*Calculated:

- sqft_basement from sqft_living and sqft_above

Dropped

- view as it is not a feature of the home; not useful for business case

Editing:

Log Transformations applied to:

- price
- sqft_living
- sqft_lot

Functions:

correlation_view:

- Evaluates correlations between all Dataframe columns/variables and displays correlation heatmap (lower triangle only) with or without correlation value annotation

high_corr:

- Evaluates correlations between all Dataframe columns/variables and displays DataFrame of variable correlations above the threshold defined

model_scores:

- Given inputs defined below, this function returns Training and Test Scores including R2, Mean Absolute Error, and Root Mean Squared Erorr via train_test_split, instantiating LinearRegression(), fitting training data, and calculating target predictions for the train and test data. Note: No scaler applied

model_scores_stanscale:

- Given inputs defined below, this function returns Training and Test Scores including R2, Mean Absolute Error, and Root Mean Squared Erorr via train_test_split, applying a Standard Scaler, instantiating LinearRegression(), fitting training data, and calculating target predictions for the train and test data.

qq_plot:

- Given inputs defined below, this function returns a qq plot to check for normal probability distribution

plot_map:

- Creates and displays Folium map of plotted longitudes and latitude returned in max zoom to make all points visible

Feature Engineering:

Feature 1

- Created: has_basement (Boolean)
- Dropped: sqft_basement

Feature 2

- Created: has_been_renovated (Boolean)
- Dropped: yr_renovated

Feature 3

- Created: age (in years)
- Dropped: yr_built

Models and Results

Models

Note: Models with biggest impact italicized

Phase 1: First Models and Editing Data for Business Case

1. Nothing But Data Cleanse - 69/68

2. Pared down locations - 69/69

Phase 2: Addressing Highly Correlated Xs and Feature Engineering

3. Model without sqft_above 67/68

4. New Features and Booleans 67/68

5. Testing without sqft_lot15 67/68

6. Testing without sqft_living15 65/66

Phase 3: Log Transforms and OHE

7. Post Log-Transform 69/70

8. OHE zipcode 83/83

9. Standard Scaler 83/83

Results

From first to the last iteration of the model, the R2 was increased from 0.68 to 0.83, and RMSE was reduced from 204,432 USD to 90,878 USD -- meaning the final model can predict within ~91,000 USD of the pricing.

Learnings

Data Analysis learnings

In [4]:
```
# Import packages
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import statsmodels.api as sm
import seaborn as sns

# For data cleansing
from datetime import datetime

# For mapping
import folium

# Import statsmodels
import statsmodels.api as sm

# Import scikit learn tools
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

In [5]:
```
# Import data
data = pd.read_csv('data/kc_house_data.csv')
data.head()
```

Out[5]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_reno
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN	0.0	...	7	1180	0.0	1955	0.0
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	...	7	2170	400.0	1951	1991.0

2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	...	6	770	0.0	1933	NaN
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	...	7	1050	910.0	1965	0.0
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	...	8	1680	0.0	1987	0.0

5 rows × 21 columns

In [6]:
```
# Overview of data types and completeness of data
data.info()
```

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

Data Cleansing

In [7]:
```
# Convert date from string to datetime; Reassign Series to be in date time format
```

```
data['date'] = pd.to_datetime(data['date'],format='%m/%d/%Y')

# Waterfront nulls - Fillna with 0 -- assume no waterfront for null values
data['waterfront'] = data['waterfront'].fillna(0.0)

# Convert sqft_basement to numerical, found '?' value, replace with null
data['sqft_basement'] = data['sqft_basement'].str.replace('?','')

# Fill nulls with using sqft_living and sqft_above that have no nulls
for row in data['sqft_basement'].index:
    data['sqft_basement'][row] = data['sqft_living'][row] - data['sqft_above'][row]

# yr_renovated nulls - Fill nulls with 0.0 as others are, assume no renovation if null
data['yr_renovated'] = data['yr_renovated'].fillna(0.0)

# Convert yr_renovated to int for cleansing and to match yr_built
data['yr_renovated'] = data['yr_renovated'].astype(int)

# Drop view, number of times house has been viewed is not a home feature
data = data.drop(labels='view', axis=1)
```

```
<ipython-input-7-e6a731736f3d>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy
  data['sqft_basement'][row] = data['sqft_living'][row] - data['sqft_above'][row]
```

In [8]:
```
# Final overview of data set after cleansing
data.info()
```

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

Function Definition

Correlations

Understanding correlations serves as a check of multicollinearity of the X variables. This notebook starts with checking a limit of 0.8 which is notably severe and is lowered to 0.6 as the model undergoes editing.

Below are the functions utilized for these checks.

Source: Severe multi-collinearity - http://www.sfu.ca/~dsignori/buec333/lecture%2016.pdf (http://www.sfu.ca/~dsignori/buec333/lecture%2016.pdf)

In [9]:
```python
def correlation_view(df, annot):
    """Evaluates correlations between all Dataframe columns/variables and displays correlation heatmap
    (lower triangle only) with or without correlation value annotation
    --
    Inputs:
    - df - Panda DataFrame
    - annot - Options are True or False. If value is True, each cell in correlation heatmap grid will have
    correlation value noted in the cell along with t. If value is False, the cell will be blank and only colored by
    color in relation to gradient scale
    --
    Outputs:
    - Correlation heatmap lower triangle with cool(blue) to hot (red) correlation gradient. Upper triangle is masked.
    """
    # From Seaborn documentation - https://seaborn.pydata.org/examples/many_pairwise_correlations.html
    # Compute the correlation matrix
    corr = df.corr()

    # Generate a mask for the upper triangle
```

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

    # Set up the matplotlib figure
    f, ax = plt.subplots(figsize=(15, 20))

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

    # Draw the heatmap with the mask and correct aspect ratio
    sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
                square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=annot)
```

In [10]:
```
def high_corr(df, thresh):
    """Evaluates correlations between all Dataframe columns/variables and displays DataFrame of variable correlations
    above the threshold defined
    --
    Inputs:
    - df - Pandas DataFrame
    - thresh - correlation threshold or limit willing to be accepted; if thresh=0.8, output will show all variables
    correlations that are higher than 0.8
    Outputs:
    - Pandas DataFrame of variable correlations above the threshold defined
    """
    # Define correlation threshold
    corr_val=thresh
    # Create DataFrame of feature 1, feature 2, and their correlation value; reset the index
    df2 = df.corr().unstack().reset_index()
    # Filter DataFrame to only show rows with correlation values above the defined threshold
    high_corr = df2[(df2[0]>corr_val)& (df2[0]<1)]
    # Show DataFrame of correlated values above threshold
    return high_corr
```

Modeling

The basis of this notebook is in iterative modeling, thus the below functions were created for efficiency to obtain model scores with and without scalers

In [11]:
```
# Define function to obtain R2, MSE, and RMSE
def model_scores(df, remove, target, testsize, rs):
    """Given inputs defined below, this function returns Training and Test Scores including R2, Mean Absolute Error,
    and Root Mean Squared Erorr via train_test_split, instantiating LinearRegression(), fitting training data, and
    calculating target predictions for the train and test data. Note: No scaler applied
    --
```

```
    Inputs:
    - df - Pandas DataFrame
    - remove - column name(s) that should not be considered in model evaluation, put single value in quotes
    (ex: 'price'), if multiple values format as list (ex: ['id', 'price'])
    - target - column name for target variable, put in quotes; ex: 'price'
    - testsize - Test size for train_test_split
    - rs - random state in train_test_split to get reproducable results
    --
    No outputs
    """
    # Define X and y
    X_cols = [c for c in df.columns.to_list() if c not in remove]
    X = df[X_cols]
    y = df[target]
    # Train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=testsize, random_state=rs)
    # Instantiate
    lr = LinearRegression()
    # Fit training data
    lr.fit(X_train, y_train)
    # Grab predictions for train and test set
    y_pred_train = lr.predict(X_train)
    y_pred_test = lr.predict(X_test)
    #Return Results
    print("Training Scores:")
    print(f"R2: {round(r2_score(y_train, y_pred_train),3)}") # can account for X amount of variance
    print(f"Mean Absolute Error: {round(mean_absolute_error(y_train, y_pred_train),3)}") # X amount off in predicting target var
iable
    print(f"Root Mean Squared Error: {round(mean_squared_error(y_train, y_pred_train, squared=False),3)}")
    print("---")
    print("Testing Scores:")
    print(f"R2: {round(r2_score(y_test, y_pred_test),3)}")
    print(f"Mean Absolute Error: {round(mean_absolute_error(y_test, y_pred_test),3)}")
    print(f"Root Mean Squared Error: {round(mean_squared_error(y_test, y_pred_test, squared=False),3)}")
```

In [12]:

```
# Define function to obtain R2, MSE, and RMSE
def model_scores_stanscale(df, remove, target, testsize, rs):
    """Given inputs defined below, this function returns Training and Test Scores including R2, Mean Absolute Error,
    and Root Mean Squared Erorr via train_test_split, applying a Standard Scaler, instantiating LinearRegression(),
    fitting training data, and calculating target predictions for the train and test data.
    --
    Inputs:
    - df - Pandas DataFrame
    - remove - column name(s) that should not be considered in model evaluation, put single value in quotes
    (ex: 'price'), if multiple values format as list (ex: ['id', 'price'])
```

```
    - target - column name for target variable, put in quotes; ex: 'price'
    - testsize - Test size for train_test_split
    - rs - random state in train_test_split to get reproducable results
    --
    No outputs
    """
    # Define X and y
    X_cols = [c for c in df.columns.to_list() if c not in remove]
    X = df[X_cols]
    y = df[target]
    # Train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=testsize, random_state=rs)
    # Instantiate a new scaler to scale our data with Standard Scaler
    scaler = StandardScaler()
    # Train scaler on training data, then fit to testing
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    # Instantiate
    lr = LinearRegression()
    # Fit training data
    lr.fit(X_train, y_train)
    # Grab predictions for train and test set
    y_pred_train = lr.predict(X_train)
    y_pred_test = lr.predict(X_test)
    #Return Results
    print("Training Scores:")
    print(f"R2: {round(r2_score(y_train, y_pred_train),3)}") # can account for X amount of variance
    print(f"Mean Absolute Error: {round(mean_absolute_error(y_train, y_pred_train),3)}") # X amount off in predicting target var
iable
    print(f"Root Mean Squared Error: {round(mean_squared_error(y_train, y_pred_train, squared=False),3)}")
    print("---")
    print("Testing Scores:")
    print(f"R2: {round(r2_score(y_test, y_pred_test),3)}")
    print(f"Mean Absolute Error: {round(mean_absolute_error(y_test, y_pred_test),3)}")
    print(f"Root Mean Squared Error: {round(mean_squared_error(y_test, y_pred_test, squared=False),3)}")
```

Plotting and Mapping

In [13]:
```
def qq_plot(df, remove, target):
    """Given inputs defined below, this function returns a qq plot to check for normal probability distribution
    --
    Inputs:
    - df - Pandas DataFrame
    - remove - column name(s) that should not be considered in model evaluation, put single value in quotes
```

```
    - remove - column name(s) that should not be considered in model evaluation, put single value in quotes
    (ex: 'price'), if multiple values format as list (ex: ['id', 'price'])
    - target - column name for target variable, put in quotes; ex: 'price'
    --
    No outputs
    """
    X_cols = [c for c in df.columns.to_list() if c not in remove]

    X = df[X_cols]
    y = df[target]

    lr = LinearRegression()

    lr.fit(X, y)

    preds = lr.predict(X)

    residuals = y-preds
    fig = sm.qqplot(residuals, line = 'r')
```

In [14]:

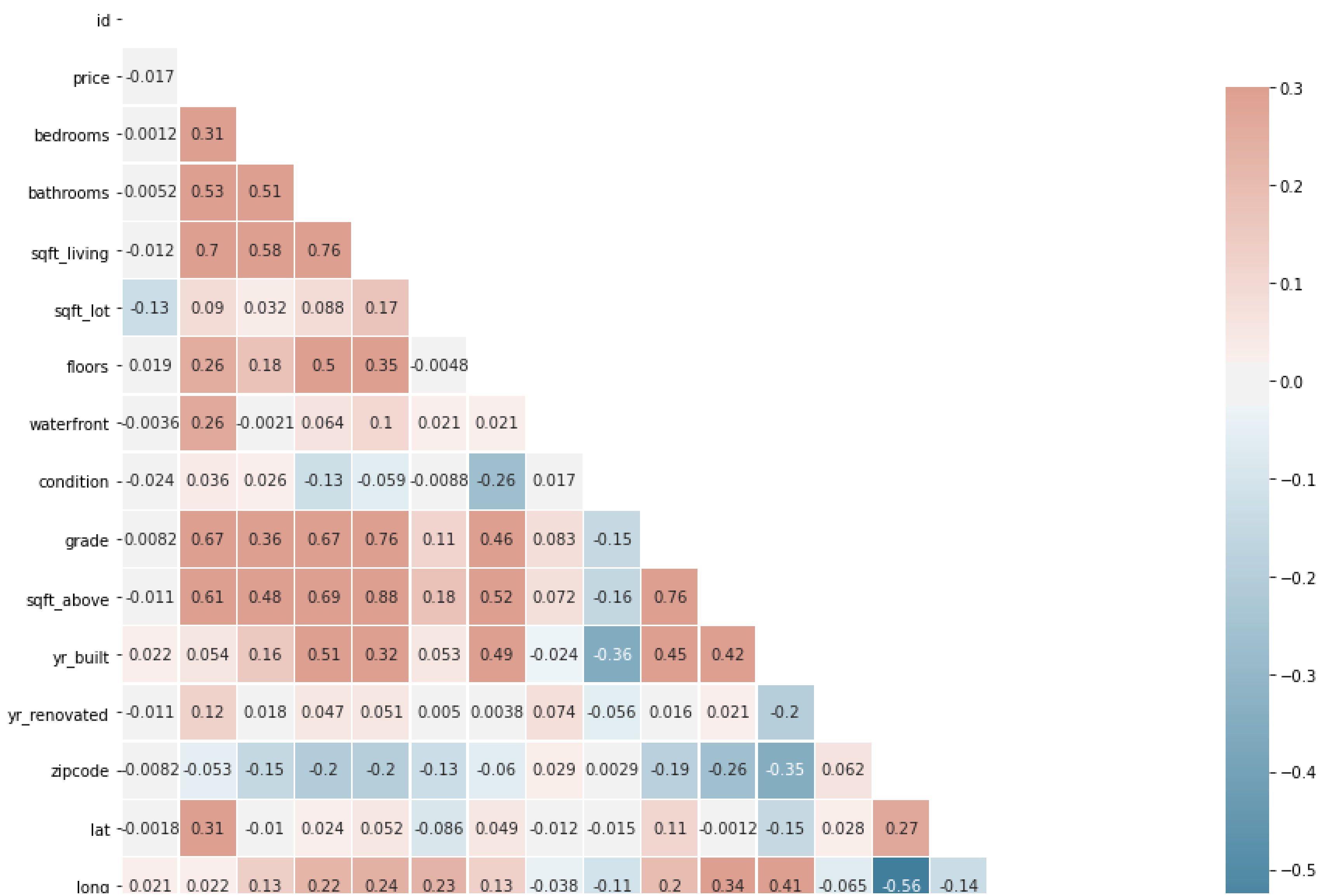
```
# Stack Overflow: https://stackoverflow.com/questions/39401729/plot-latitude-longitude-points-from-dataframe-on-folium-map-ipyth
on

def plot_map(df):
    """Creates and displays Folium map of plotted longitudes and latitude returned in max zoom to
    make all points visible
    --
    Inputs:
    - df = Pandas DataFrame that contains numeric column for latitude labeled 'lat' and numeric column for
    longitude labeled 'long'
     """
    # Create a map
    this_map = folium.Map(prefer_canvas=True)

    def plotDot(point):
        '''input: series that contains a numeric named latitude and a numeric named longitude
        this function creates a CircleMarker and adds it to your this_map'''
        folium.CircleMarker(location=[point.lat, point.long],
                            radius=1,
                            weight=1, popup=point.long).add_to(this_map)

    #use df.apply(,axis=1) to "iterate" through every row in your dataframe
    df.apply(plotDot, axis = 1)


    #Set the zoom to the maximum possible
```

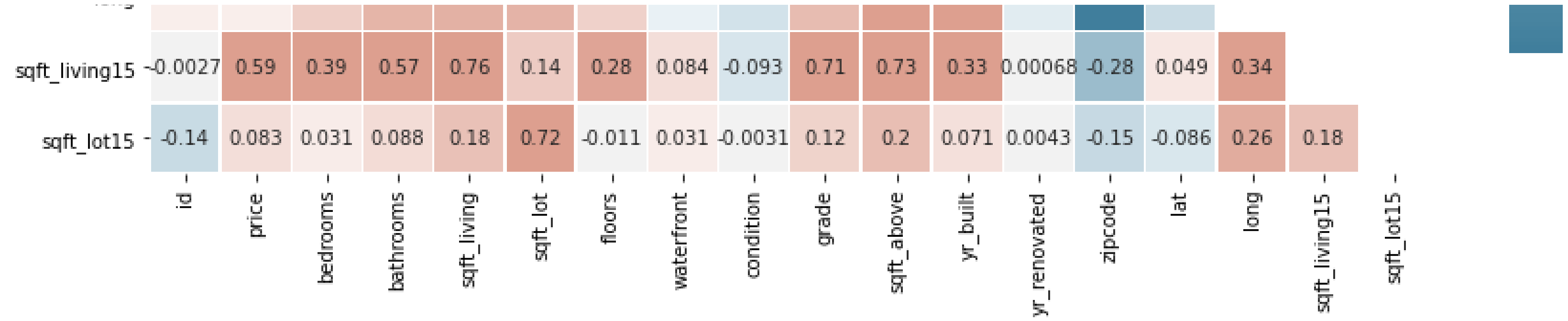
```
    this_map.fit_bounds(this_map.get_bounds())

    return this_map
```

Initial Correlations

In [15]:

```
# Initial correlations in main dataset after cleansing
correlation_view(data, annot=True)
```





In [16]:
```
# Obtain ordered list of variables with high correlation with target variable, price
data.corr().price.sort_values(ascending=False)
# Grade, sqft_living, sqft_above, sqft_living15, bathrooms are most correlated with price
```

Out[16]:
```
price            1.000000
sqft_living      0.701917
grade            0.667951
sqft_above       0.605368
sqft_living15    0.585241
bathrooms        0.525906
bedrooms         0.308787
lat              0.306692
waterfront       0.264306
floors           0.256804
yr_renovated     0.117855
sqft_lot         0.089876
sqft_lot15       0.082845
yr_built         0.053953
condition        0.036056
long             0.022036
id              -0.016772
zipcode         -0.053402
Name: price, dtype: float64
```

Phase 1: First Models and Editing Data for Business Case

Model 1: Nothing But Data Cleanse

In [17]:
```
model_scores(data, remove=['price', 'date', 'id'], target =['price'], testsize=0.33, rs=42)
# Training > Testing so slightly overfit
```

```
Training Scores:
```

```
Training Scores:
R2: 0.696
Mean Absolute Error: 129057.129
Root Mean Squared Error: 204151.006
---
Testing Scores:
R2: 0.68
Mean Absolute Error: 128165.89
Root Mean Squared Error: 204432.439
```

In [18]:

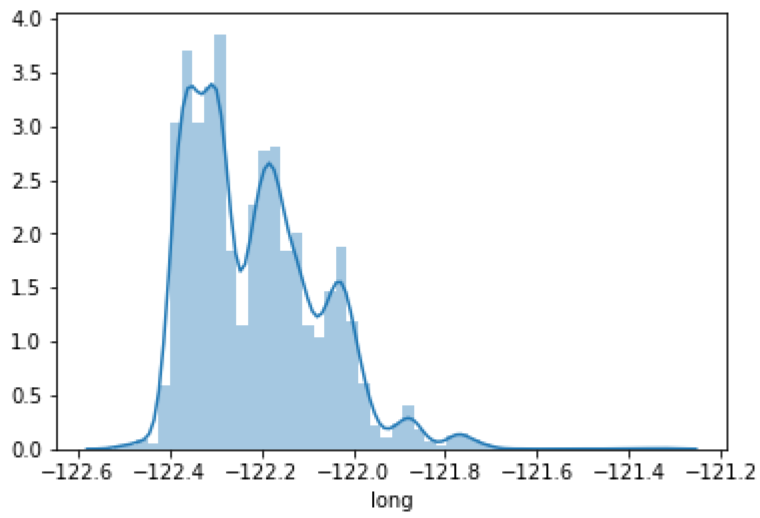
```
# plot_map(data)
```

In [19]:

```
# Map showing outliers mostly in the east, look at distribution of longitude (east/west)
sns.distplot(data['long'])
```

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb9ccb8c550>
```



In [20]:

```
# Different look at distribuion numbers. Can see max outlier
data['long'].describe()
```

Out[20]:

```
count    21597.000000
mean      -122.213982
std          0.140724
min       -122.519000
25%       -122.328000
50%       -122.231000
75%       -122.125000
max       -121.315000
Name: long, dtype: float64
```

In [21]:
```
# Filter down data to 75% and remap
data_longzoom = data[data['long'] <= -122.125]
```

In [22]:
```
# Cut out 25% of records, but should be better for model to have a more condensed space
data_longzoom.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16214 entries, 0 to 21596
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             16214 non-null  int64
 1   date           16214 non-null  datetime64[ns]
 2   price          16214 non-null  float64
 3   bedrooms       16214 non-null  int64
 4   bathrooms      16214 non-null  float64
 5   sqft_living    16214 non-null  int64
 6   sqft_lot       16214 non-null  int64
 7   floors         16214 non-null  float64
 8   waterfront     16214 non-null  float64
 9   condition      16214 non-null  int64
 10  grade          16214 non-null  int64
 11  sqft_above     16214 non-null  int64
 12  sqft_basement  16214 non-null  object
 13  yr_built       16214 non-null  int64
 14  yr_renovated   16214 non-null  int64
 15  zipcode        16214 non-null  int64
 16  lat            16214 non-null  float64
 17  long           16214 non-null  float64
 18  sqft_living15  16214 non-null  int64
 19  sqft_lot15     16214 non-null  int64
dtypes: datetime64[ns](1), float64(6), int64(12), object(1)
memory usage: 2.6+ MB
```

In [23]:
```
# Replot map with zoomed in longitudes
plot_map(data_longzoom)
```

Out[23]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [24]:
```
# How many zipcodes accounted for before?
data['zipcode'].nunique()
```

Out[24]: 70

In [25]:
```
# Zipcodes down from 70 to 57
```

```
data_longzoom['zipcode'].nunique()
```

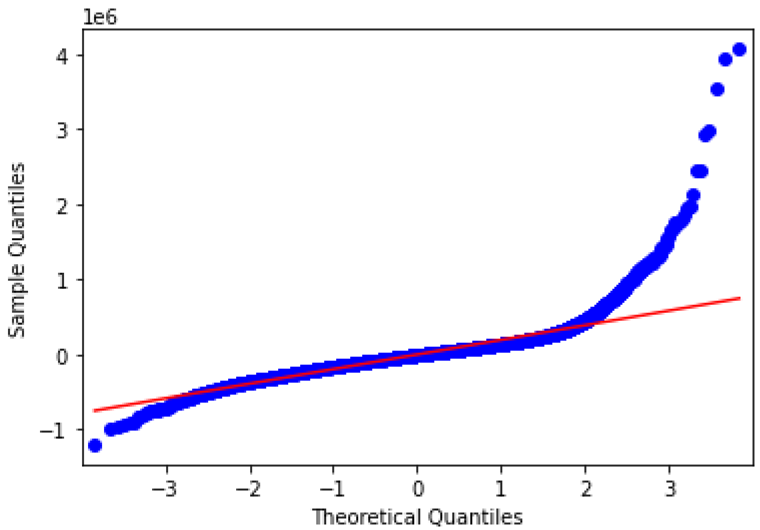
Out[25]: 57

Model 2: Pared down locations

In [26]:
```
model_scores(data_longzoom, remove=['price', 'date', 'id'], target =['price'], testsize=0.33, rs=42)
# No real change in R-squared, slightly higher test vs full previous dataset
# Training > Testing so slightly overfit
```

```
Training Scores:
R2: 0.696
Mean Absolute Error: 135584.93
Root Mean Squared Error: 208699.643
---
Testing Scores:
R2: 0.693
Mean Absolute Error: 138727.36
Root Mean Squared Error: 226383.5
```

In [27]:
```
qq_plot(data_longzoom, remove=['price', 'date', 'id'], target='price')
```



In [28]:
```
high_corr(data_longzoom, 0.7)
# 0.8 correlation is severe multi-collinearity - http://www.sfu.ca/~dsignori/buec333/lecture%2016.pdf
# Plenty of highly correlated features to address - sqft_living and sqft_above is severe and should be first priority
```

Out[28]:

	level_0	level_1	0

22	price	sqft_living	0.703570
58	bathrooms	sqft_living	0.743494
73	sqft_living	price	0.703570
75	sqft_living	bathrooms	0.743494
81	sqft_living	grade	0.740156
82	sqft_living	sqft_above	0.858212
88	sqft_living	sqft_living15	0.717626
166	grade	sqft_living	0.740156
172	grade	sqft_above	0.730115
184	sqft_above	sqft_living	0.858212
189	sqft_above	grade	0.730115
292	sqft_living15	sqft_living	0.717626

Phase 2: Addressing Highly Correlated Xs and Feature Engineering

Investigating and Addressing Outliers

In [29]:

```
# Descibe full data set to see any glaring outliers
data_longzoom.describe()
```

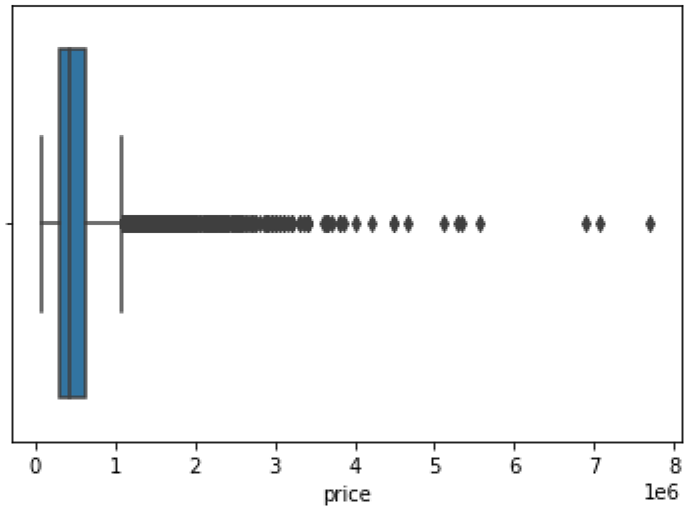
Out[29]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	grade	sqft_a
count	1.621400e+04	1.621400e+04	16214.000000	16214.000000	16214.000000	16214.000000	16214.000000	16214.000000	16214.000000	16214.000000	1621
mean	4.545986e+09	5.335436e+05	3.333169	2.029049	1971.582953	9514.380165	1.446620	0.007278	3.452695	7.523066	1646.
std	2.830427e+09	3.889760e+05	0.962320	0.773342	870.450928	17842.969808	0.547647	0.085001	0.675099	1.108744	746.3
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	520.000000	1.000000	0.000000	1.000000	4.000000	370.0
25%	2.172000e+09	3.100000e+05	3.000000	1.500000	1360.000000	4929.250000	1.000000	0.000000	3.000000	7.000000	1130.
50%	3.885802e+09	4.350000e+05	3.000000	2.000000	1810.000000	7265.500000	1.000000	0.000000	3.000000	7.000000	1440.
75%	7.212651e+09	6.230000e+05	4.000000	2.500000	2400.000000	9647.000000	2.000000	0.000000	4.000000	8.000000	1970.

max	9.900000e+09	7.700000e+06	33.000000	8.000000	12050.000000	843309.000000	3.500000	1.000000	5.000000	13.000000	8860.

In [30]:
```
# Clear outlier in max price
sns.boxplot(data_longzoom['price'])
# 1.25 mil accounts for most
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb9a58047f0>



In [31]:
```
# Current shape
data_longzoom.shape
```

Out[31]: (16214, 20)

In [32]:
```
# How many records left when taking out outliers
data_longzoom[data_longzoom['price'] <= 1250000]['id'].count()
```

Out[32]: 15474

In [33]:
```
(16214-15474)/16214
# 4% of data, but outliers will impact model success
```

Out[33]: 0.04563957074133465

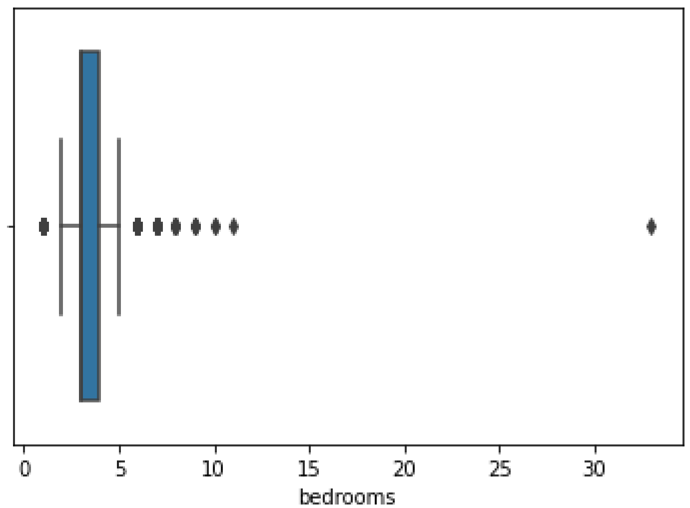
In [34]:
```
# Create "master variable" data_focused and filter out price outliers
data_focused = data_longzoom[data_longzoom['price'] <= 1250000]
```

In [35]:
```
# Bedrooms also had outliers
data_focused['bedrooms'].describe()
```

```
Out[35]: count    15474.000000
         mean         3.294494
         std          0.946613
         min          1.000000
         25%          3.000000
         50%          3.000000
         75%          4.000000
         max         33.000000
         Name: bedrooms, dtype: float64
```

```
In [36]: sns.boxplot(data_focused['bedrooms'])
         # 1-5 bedrooms looks like where data is concentrated
```

```
Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb9a565f1f0>
```




```
In [37]: # How many records in 1-5 bedroom range?
         data_focused[(data_focused['bedrooms'] >=1) & (data_focused['bedrooms'] <=5)]['id'].count()
```

```
Out[37]: 15231
```

```
In [38]: (15474-15231)/15474
         # Another 1% of data loss, but for betterment of model and concentration
```

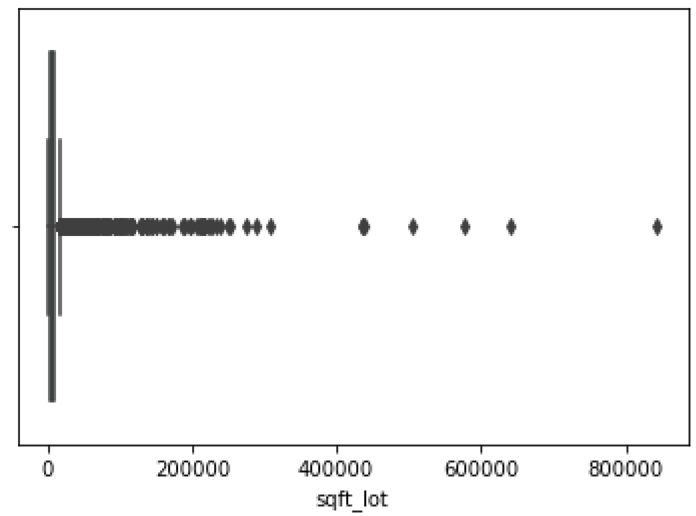
```
Out[38]: 0.015703761147731678
```

```
In [39]: # Update master variable and dataframe
         data_focused = data_focused[(data_focused['bedrooms'] >=1) & (data_focused['bedrooms'] <=5)]
```

```
In [40]: #Sqft lot also had outliers
```

In [40]:
```
#sqft_lot also had outliers
sns.boxplot(data_focused['sqft_lot'])
```

Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb99fc3cdc0>



In [41]:
```
data_focused['sqft_lot'].describe()
```

Out[41]:
```
count     15231.000000
mean       9236.394196
std       17938.694219
min         520.000000
25%        4800.000000
50%        7203.000000
75%        9506.500000
max      843309.000000
Name: sqft_lot, dtype: float64
```

In [42]:
```
# Many over the 75% mark, but few above 30000. Remove these
(data_focused['sqft_lot']>=30000).sum()
```

Out[42]: 390

In [43]:
```
# Update master variable and dataframe
data_focused = data_focused[data_focused['sqft_lot']<30000]
```

In [44]:
```
# Check bathrooms since they have more detailed measurement and large range (in context of 5 bedroom houses)
data_focused.groupby(by='bathrooms')['id'].count()
```

Out[44]:
```
bathrooms
0.50       4
```

```
0.75      47
1.00    3389
1.25       7
1.50    1230
1.75    2371
2.00    1465
2.25    1386
2.50    3056
2.75     721
3.00     470
3.25     257
3.50     327
3.75      48
4.00      33
4.25      15
4.50      13
4.75       1
5.00       1
Name: id, dtype: int64
```

In [45]:
```python
# Create new column for full baths to consolidate data values
data_focused['full_bath'] = 0

# For each value in data['bathrooms']
for row in data_focused['bathrooms'].index:
    if (data_focused['bathrooms'][row] <1):
        data_focused['full_bath'][row] = 0
    if (data_focused['bathrooms'][row] >=1) and (data_focused['bathrooms'][row] <2):
        data_focused['full_bath'][row] = 1
    if (data_focused['bathrooms'][row] >=2) and (data_focused['bathrooms'][row] <3):
        data_focused['full_bath'][row] = 2
    if (data_focused['bathrooms'][row] >=3) and (data_focused['bathrooms'][row] <4):
        data_focused['full_bath'][row] = 3
    if (data_focused['bathrooms'][row] >=4) and (data_focused['bathrooms'][row] <5):
        data_focused['full_bath'][row] = 4
    if (data_focused['bathrooms'][row] >=5) and (data_focused['bathrooms'][row] <6):
        data_focused['full_bath'][row] = 5
```

```
<ipython-input-45-8bc225fe4a62>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy
  data_focused['full_bath'][row] = 1
<ipython-input-45-8bc225fe4a62>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy
  data_focused['full_bath'][row] = 2
<ipython-input-45-8bc225fe4a62>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy
  data_focused['full_bath'][row] = 3
<ipython-input-45-8bc225fe4a62>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy
  data_focused['full_bath'][row] = 0
<ipython-input-45-8bc225fe4a62>:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy
  data_focused['full_bath'][row] = 4
<ipython-input-45-8bc225fe4a62>:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy
  data_focused['full_bath'][row] = 5
```

In [46]:
```
# Overview into new distribution of bathroom counts
data_focused.groupby(by='full_bath')['id'].count()
```

Out[46]:
```
full_bath
0      51
1    6997
2    6628
3    1102
4      62
5       1
Name: id, dtype: int64
```

In [47]:
```
# Majority of data is between 1 and 3 baths
data_focused[(data_focused['full_bath']>0) & (data_focused['full_bath']<4)]['id'].count()
```

Out[47]:
```
14727
```

In [48]:
```
(15223-15096)/15223
# Less than 1% loss in removing outliers
```

Out[48]:
```
0.008342639427182552
```

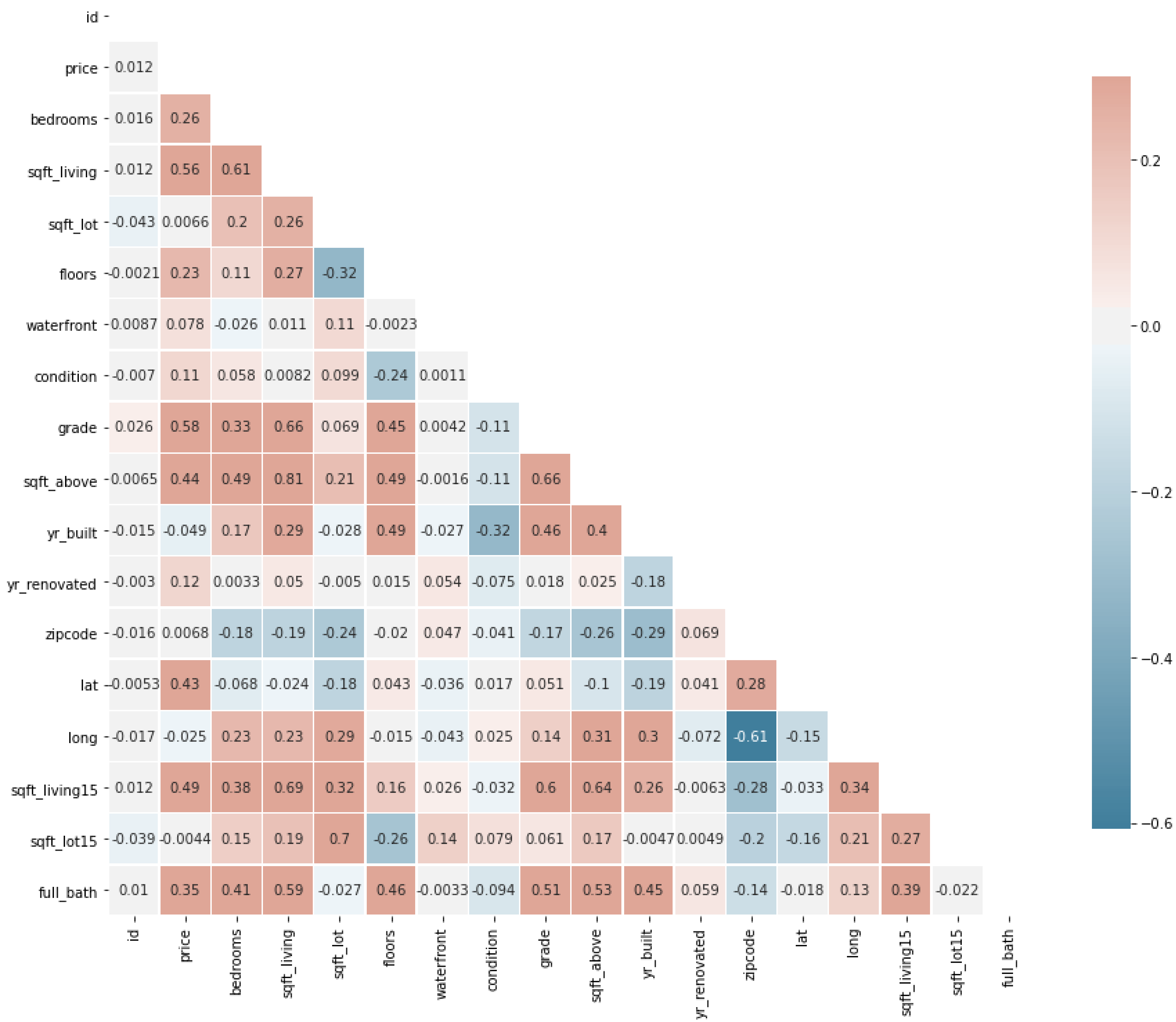
In [49]:
```
# Update master variable
data_focused = data_focused[(data_focused['full_bath']>0) & (data_focused['full_bath']<4)]
```

In [50]:
```
# Update master variable and drop bathroom since it was source of full_bath and only need one in model
data_focused = data_focused.drop(labels='bathrooms', axis=1)
```

In [51]:
```
data_focused.info()
# Homes with price <= $1.25 million, 1-5 bedrooms, 1-3 bathrooms (full_bath), smaller geographical area
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14727 entries, 0 to 21593
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             14727 non-null  int64
 1   date           14727 non-null  datetime64[ns]
 2   price          14727 non-null  float64
 3   bedrooms       14727 non-null  int64
 4   sqft_living    14727 non-null  int64
 5   sqft_lot       14727 non-null  int64
 6   floors         14727 non-null  float64
 7   waterfront     14727 non-null  float64
 8   condition      14727 non-null  int64
 9   grade          14727 non-null  int64
 10  sqft_above     14727 non-null  int64
 11  sqft_basement  14727 non-null  object
 12  yr_built       14727 non-null  int64
 13  yr_renovated   14727 non-null  int64
 14  zipcode        14727 non-null  int64
 15  lat            14727 non-null  float64
 16  long           14727 non-null  float64
 17  sqft_living15  14727 non-null  int64
 18  sqft_lot15     14727 non-null  int64
 19  full_bath      14727 non-null  int64
dtypes: datetime64[ns](1), float64(5), int64(13), object(1)
memory usage: 2.4+ MB
```

In [52]:
```
# Review correlations with edited dataset
correlation_view(data_focused, annot=True)
```


id
price 0.012
bedrooms 0.016 0.26
sqft_living 0.012 0.56 0.61
sqft_lot -0.043 0.0066 0.2 0.26
floors -0.0021 0.23 0.11 0.27 -0.32
waterfront -0.0087 0.078 -0.026 0.011 0.11 -0.0023
condition -0.007 0.11 0.058 0.0082 0.099 -0.24 0.0011
grade 0.026 0.58 0.33 0.66 0.069 0.45 0.0042 -0.11
sqft_above -0.0065 0.44 0.49 0.81 0.21 0.49 -0.0016 -0.11 0.66
yr_built -0.015 -0.049 0.17 0.29 -0.028 0.49 -0.027 -0.32 0.46 0.4
yr_renovated -0.003 0.12 0.0033 0.05 -0.005 0.015 0.054 -0.075 0.018 0.025 -0.18
zipcode -0.016 0.0068 -0.18 -0.19 -0.24 -0.02 0.047 -0.041 -0.17 -0.26 -0.29 0.069
lat -0.0053 0.43 -0.068 -0.024 -0.18 0.043 -0.036 0.017 0.051 -0.1 -0.19 0.041 0.28
long -0.017 -0.025 0.23 0.23 0.29 -0.015 -0.043 0.025 0.14 0.31 0.3 -0.072 -0.61 -0.15
sqft_living15 0.012 0.49 0.38 0.69 0.32 0.16 0.026 -0.032 0.6 0.64 0.26 -0.0063 -0.28 -0.033 0.34
sqft_lot15 -0.039 -0.0044 0.15 0.19 0.7 -0.26 0.14 0.079 0.061 0.17 -0.0047 0.0049 -0.2 -0.16 0.21 0.27
full_bath 0.01 0.35 0.41 0.59 -0.027 0.46 -0.0033 -0.094 0.51 0.53 0.45 0.059 -0.14 -0.018 0.13 0.39 -0.022
id price bedrooms sqft_living sqft_lot floors waterfront condition grade sqft_above yr_built yr_renovated zipcode lat long sqft_living15 sqft_lot15 full_bath
0.2
0.0
−0.2
−0.4
−0.6

In [53]:
```
high_corr(data_focused,0.7)
# Removing outliers addressed many of previously high correlation features
# sqft_above and sqft_living is still high at .8.
# Test without sqft_above as sqft_living has higher correlation with price
```

Out[53]:

	level_0	level_1	0
63	sqft_living	sqft_above	0.80977
165	sqft_above	sqft_living	0.80977

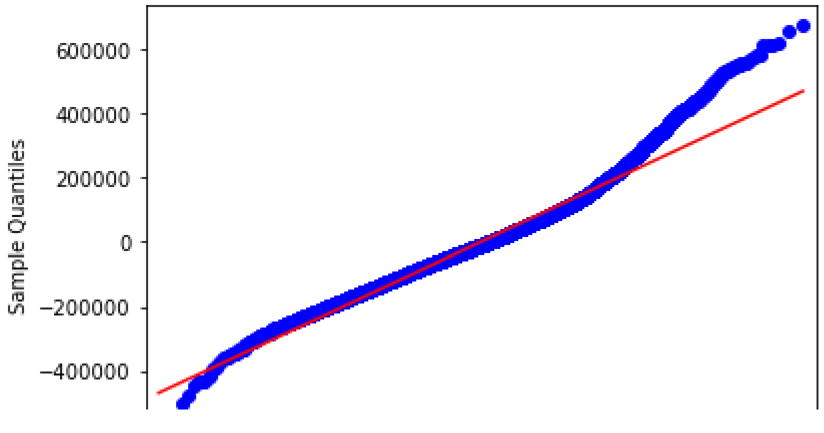
Model 3: Model without sqft_above

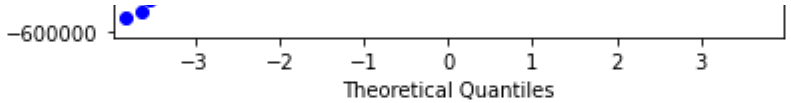
In [54]:
```
# Create new variable for test table without sqft_above
data_focused_nosfa = data_focused.drop(labels='sqft_above', axis=1)
```

In [55]:
```
model_scores(data_focused_nosfa, remove=['price', 'date', 'id'], target =['price'], testsize=0.33, rs=42)
# R-squared down, but expected because of high correlation between the two
# Testing > Training so slightly underfit
```

```
Training Scores:
R2: 0.67
Mean Absolute Error: 93379.26
Root Mean Squared Error: 125055.975
---
Testing Scores:
R2: 0.681
Mean Absolute Error: 93135.831
Root Mean Squared Error: 124364.04
```

In [56]:
```
qq_plot(data_focused_nosfa, remove=['price', 'date', 'id'], target='price')
```





In [57]:
```
high_corr(data_focused_nosfa, 0.7)
# Check back on correlations. None over 0.7 now.
```

Out[57]:

	level_0	level_1	0

In [58]:
```
# Reassign master variable
data_focused = data_focused_nosfa
```

Additional Feature Engineering and Replacement

In [59]:
```
data_focused[data_focused['sqft_basement']==0]['id'].count()
# Nearly half of dataset doesn't have basement
```

Out[59]: 8331

In [60]:
```
# Create column for has_basement, input Boolean values (1 if has a basement) and replace sqft_basement
data_focused['has_basement'] = 0

for row in data_focused['sqft_basement'].index:
    if (data_focused['sqft_basement'][row] > 0.0):
        data_focused['has_basement'][row] = 1
    else:
        data_focused['has_basement'][row] = 0
```

```
<ipython-input-60-08e2cdac7c5a>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy
  data_focused['has_basement'][row] = 0
<ipython-input-60-08e2cdac7c5a>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy
  data_focused['has_basement'][row] = 1
```

In [61]:
```
data_focused[data_focused['yr_renovated']==0]['id'].count()
# Most homes haven't been renovated
```

```
Out[61]: 14231
```

```
In [62]: # Create new column for Boolean has_been_renovated
data_focused['has_been_renovated'] = 0

# For each value in data['yr_renovated'], if 0 put 0 in has_been_renovated; 1 if it has other values
for row in data_focused['yr_renovated'].index:
    if data_focused['yr_renovated'][row] == 0:
        data_focused['has_been_renovated'][row] = 0
    elif data_focused['yr_renovated'][row] > 0:
        data_focused['has_been_renovated'][row] = 1
```

```
<ipython-input-62-e1927eecb0d8>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy
  data_focused['has_been_renovated'][row] = 0
<ipython-input-62-e1927eecb0d8>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy
  data_focused['has_been_renovated'][row] = 1
```

```
In [63]: # Update master variable and drop replaced columns ()
data_focused = data_focused.drop(labels=['sqft_basement', 'yr_renovated'], axis=1)
```

```
In [64]: data_focused.columns
```

```
Out[64]: Index(['id', 'date', 'price', 'bedrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'condition', 'grade', 'yr_built', 'zipcode', 'lat',
       'long', 'sqft_living15', 'sqft_lot15', 'full_bath', 'has_basement',
       'has_been_renovated'],
      dtype='object')
```

```
In [65]: # Create age from yr_built
data_focused['age'] = 0
```

```
In [66]: # Also need year of sale for age
data_focused['yr_of_sale'] = pd.DatetimeIndex(data_focused['date']).year
```

```
In [67]: # Check work
```

```
data_focused['yr_of_sale'].head()
```

Out[67]:
```
0    2014
1    2014
2    2015
3    2014
6    2014
Name: yr_of_sale, dtype: int64
```

In [68]:
```
for row in data_focused['age'].index:
    data_focused['age'][row] = data_focused['yr_of_sale'][row] - data_focused['yr_built'][row]
```

```
<ipython-input-68-59e13b09e762>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy
  data_focused['age'][row] = data_focused['yr_of_sale'][row] - data_focused['yr_built'][row]
```

In [69]:
```
data_focused['age'].describe()
# min is -1 need to take out negatives, data errors means sold before built
```

Out[69]:
```
count    14727.000000
mean        49.278740
std         29.820287
min         -1.000000
25%         25.000000
50%         50.000000
75%         68.000000
max        115.000000
Name: age, dtype: float64
```

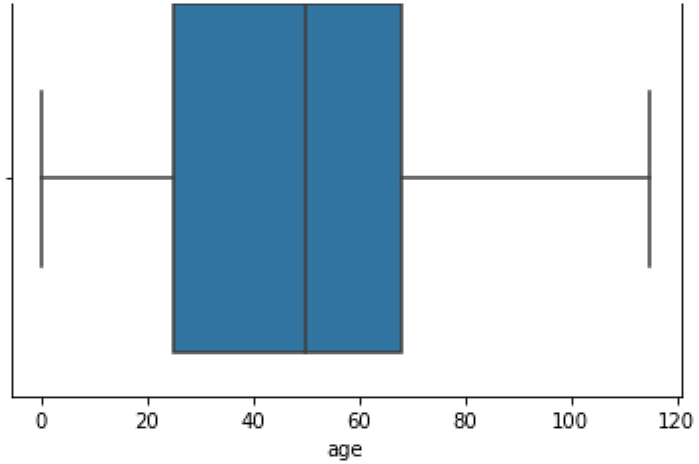
In [70]:
```
# How many records have negative age value
data_focused[data_focused['age'] <0]['id'].count()
```

Out[70]: 11

In [71]:
```
# Update master variable to filter out negative age values
data_focused = data_focused[data_focused['age'] >=0]
```

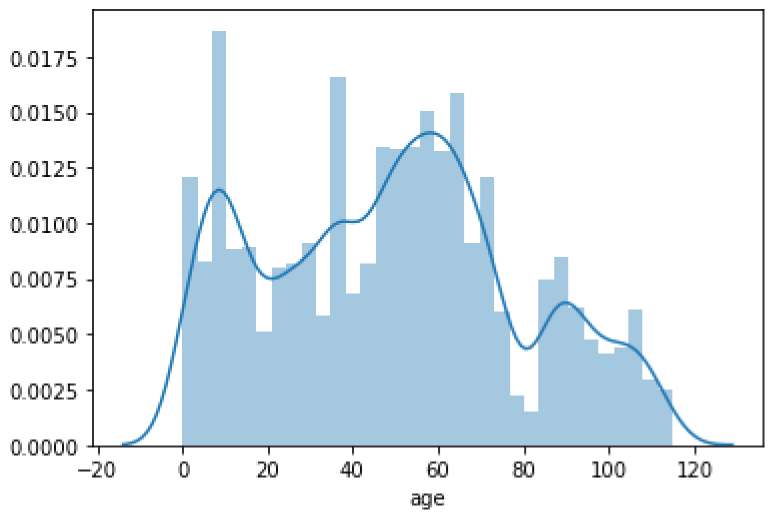
In [72]:
```
# Boxplot of age to see general distribution and extremes
sns.boxplot(data_focused['age'])
```

Out[72]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb99ffd7790>



In [73]:
```
# Different view of distribution to explore age > 70 occurance
sns.distplot(data_focused['age'])
```

Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb9a1e889d0>



In [74]:
```
# Take out columns used to assist in building this feature - would contribute to multicollinearity
data_focused = data_focused.drop(labels=['yr_of_sale', 'yr_built'], axis=1)
```

In [75]:
```
data_focused = data_focused[data_focused['age'] >=0]
```

In [76]:
```
# Check work
data_focused.columns
```

Out[76]: Index(['id', 'date', 'price', 'bedrooms', 'sqft_living', 'sqft_lot', 'floors',

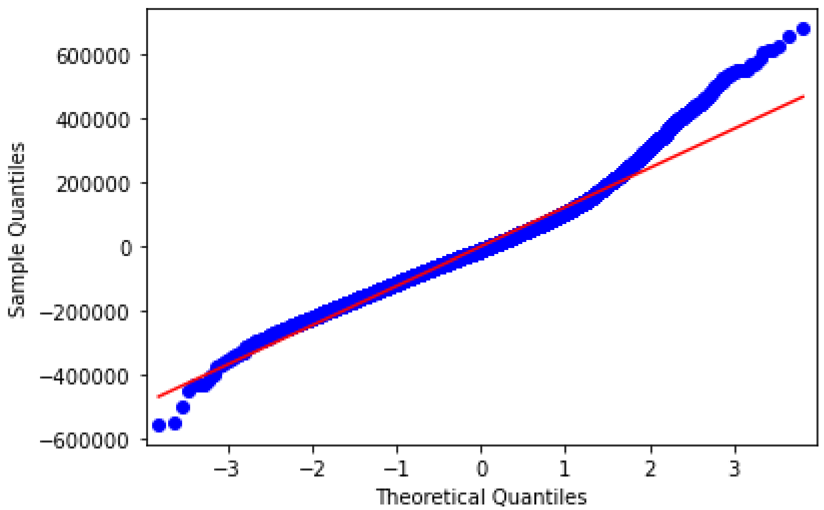
```
       'waterfront', 'condition', 'grade', 'zipcode', 'lat', 'long',
       'sqft_living15', 'sqft_lot15', 'full_bath', 'has_basement',
       'has_been_renovated', 'age'],
      dtype='object')
```

Model 4: New Features and Booleans

In [77]:
```
model_scores(data_focused, remove=['price', 'date', 'id'], target =['price'], testsize=0.33, rs=42)
# R-squared slightly up and data simplified
# Testing > Training so slightly underfit
```

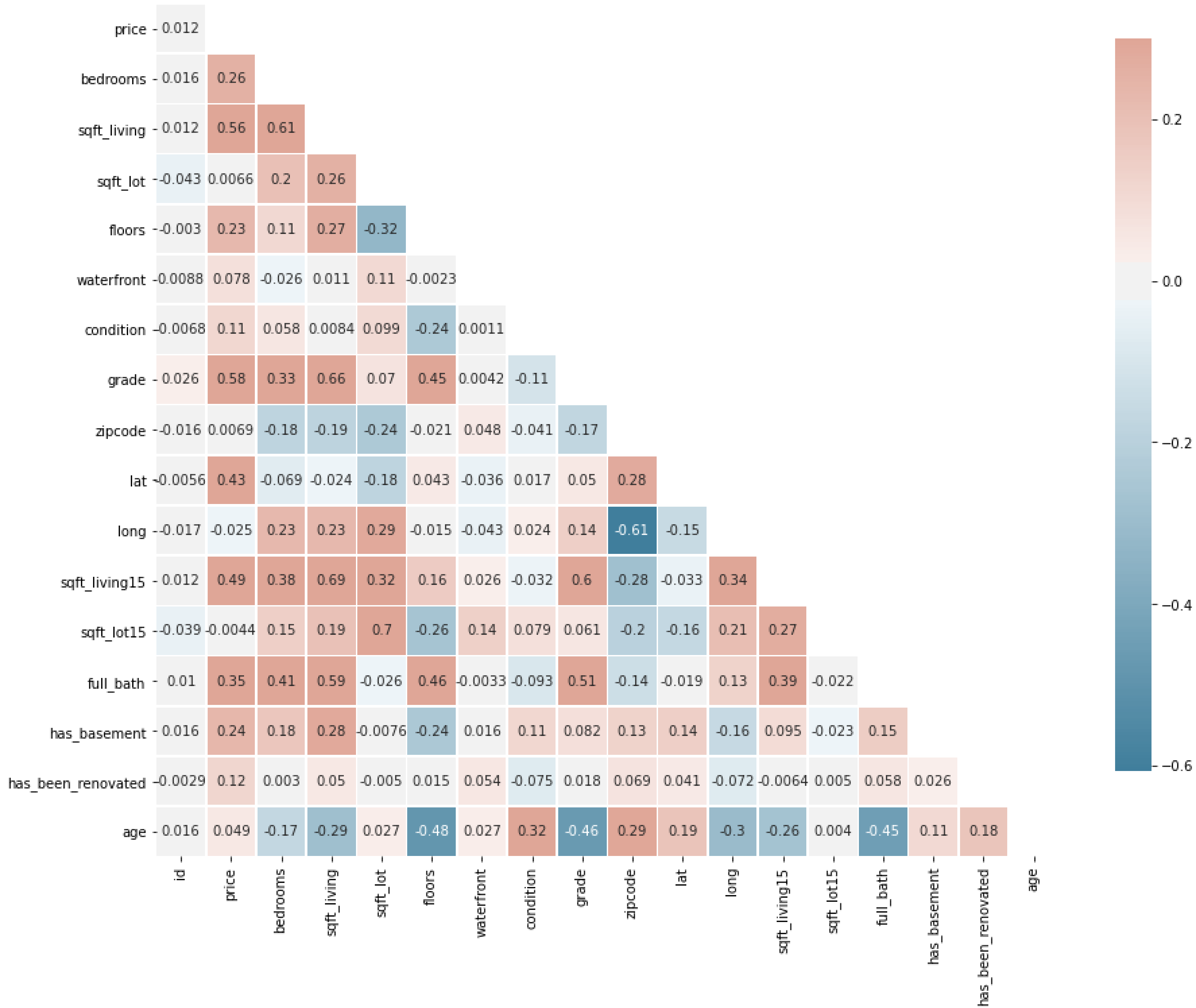
```
Training Scores:
R2: 0.672
Mean Absolute Error: 92948.693
Root Mean Squared Error: 124292.556
---
Testing Scores:
R2: 0.681
Mean Absolute Error: 93117.546
Root Mean Squared Error: 125178.436
```

In [78]:
```
qq_plot(data_focused, remove=['price', 'date', 'id'], target='price')
# similar qq plot, straying at 2 deviations from mean
```



In [79]:
```
correlation_view(data_focused, annot=True)
```

id -




```
In [80]: # Nothing over 0.7, checking 0.6 as thresh
```

```
In [80]: # Nothing over 0.7, checking 0.6 as thresh
high_corr(data_focused, 0.6)
# Sqft_lot15 has high correlation with sqft_lot. Sqft_lot has slightly higher correlation with price and
# is related to the specific property versus the area. Test without sqft_lot15
```

Out[80]:

	level_0	level_1	0
39	bedrooms	sqft_living	0.612915
56	sqft_living	bedrooms	0.612915
62	sqft_living	grade	0.660963
66	sqft_living	sqft_living15	0.685718
85	sqft_lot	sqft_lot15	0.699094
147	grade	sqft_living	0.660963
156	grade	sqft_living15	0.603016
219	sqft_living15	sqft_living	0.685718
224	sqft_living15	grade	0.603016
238	sqft_lot15	sqft_lot	0.699094

Model 5: Testing without sqft_lot15

```
In [81]: # Create new variable for test table without sqft_lot15
data_focused_nolot15_test = data_focused.drop(labels='sqft_lot15', axis=1)
```

```
In [82]: model_scores(data_focused_nolot15_test, remove=['price', 'date', 'id'], target =['price'], testsize=0.33, rs=42)
# R-squared same - Keep sqft_lot15 off
# Testing > Training so slightly underfit
```

```
Training Scores:
R2: 0.672
Mean Absolute Error: 92979.976
Root Mean Squared Error: 124340.25
---
Testing Scores:
R2: 0.681
Mean Absolute Error: 93162.869
Root Mean Squared Error: 125270.939
```

```
In [83]: # Reassign master variable
```

```
In [83]: # Reassign master variable
data_focused = data_focused_nolot15_test
```

```
In [84]: # Review columns, check work
data_focused.columns
```

```
Out[84]: Index(['id', 'date', 'price', 'bedrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'condition', 'grade', 'zipcode', 'lat', 'long',
       'sqft_living15', 'full_bath', 'has_basement', 'has_been_renovated',
       'age'],
      dtype='object')
```

```
In [85]: # Nothing over 0.7, checking 0.6 as thresh
high_corr(data_focused, 0.6)
# Sqft_living15 has high correlation with sqft_living. Sqft_living has slightly higher correlation with price and
# is related to the specific property versus the area. Test without sqft_living15
```

Out[85]:

	level_0	level_1	0
37	bedrooms	sqft_living	0.612915
53	sqft_living	bedrooms	0.612915
59	sqft_living	grade	0.660963
63	sqft_living	sqft_living15	0.685718
139	grade	sqft_living	0.660963
148	grade	sqft_living15	0.603016
207	sqft_living15	sqft_living	0.685718
212	sqft_living15	grade	0.603016

Model 6: Testing without sqft_living15

```
In [86]: # Sqft_living15 also had high correlaton with sqft_living. Create new variable to test without
data_focused_noliv15_test = data_focused.drop(labels='sqft_living15', axis=1)
```

```
In [87]: model_scores(data_focused_noliv15_test, remove=['price', 'date', 'id'], target =['price'], testsize=0.33, rs=42)
# R-squared down, but expected - need to rid of highly correlated variables that measure similar things
# Testing > Training so slightly underfit
```

```
Training Scores:
R2: 0.658
```

```
Mean Absolute Error: 94555.599
Root Mean Squared Error: 126852.587
---
Testing Scores:
R2: 0.669
Mean Absolute Error: 94771.598
Root Mean Squared Error: 127640.594
```

In [88]:
```
# Reassign master variable
data_focused = data_focused_noliv15_test
```

In [89]:
```
# Check work, view columns
data_focused.columns
```

Out[89]:
```
Index(['id', 'date', 'price', 'bedrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'condition', 'grade', 'zipcode', 'lat', 'long',
       'full_bath', 'has_basement', 'has_been_renovated', 'age'],
      dtype='object')
```

In [90]:
```
# Nothing over 0.7, checking 0.6 as thresh
high_corr(data_focused,0.6)
# Down to only 2 correlations - bedrooms and sqft_living / sqft_living and grade
# Will move on to additional levers and come back if needed.
```

Out[90]:

	level_0	level_1	0
35	bedrooms	sqft_living	0.612915
50	sqft_living	bedrooms	0.612915
56	sqft_living	grade	0.660963
131	grade	sqft_living	0.660963

Phase 3: Log Transforms and OHE

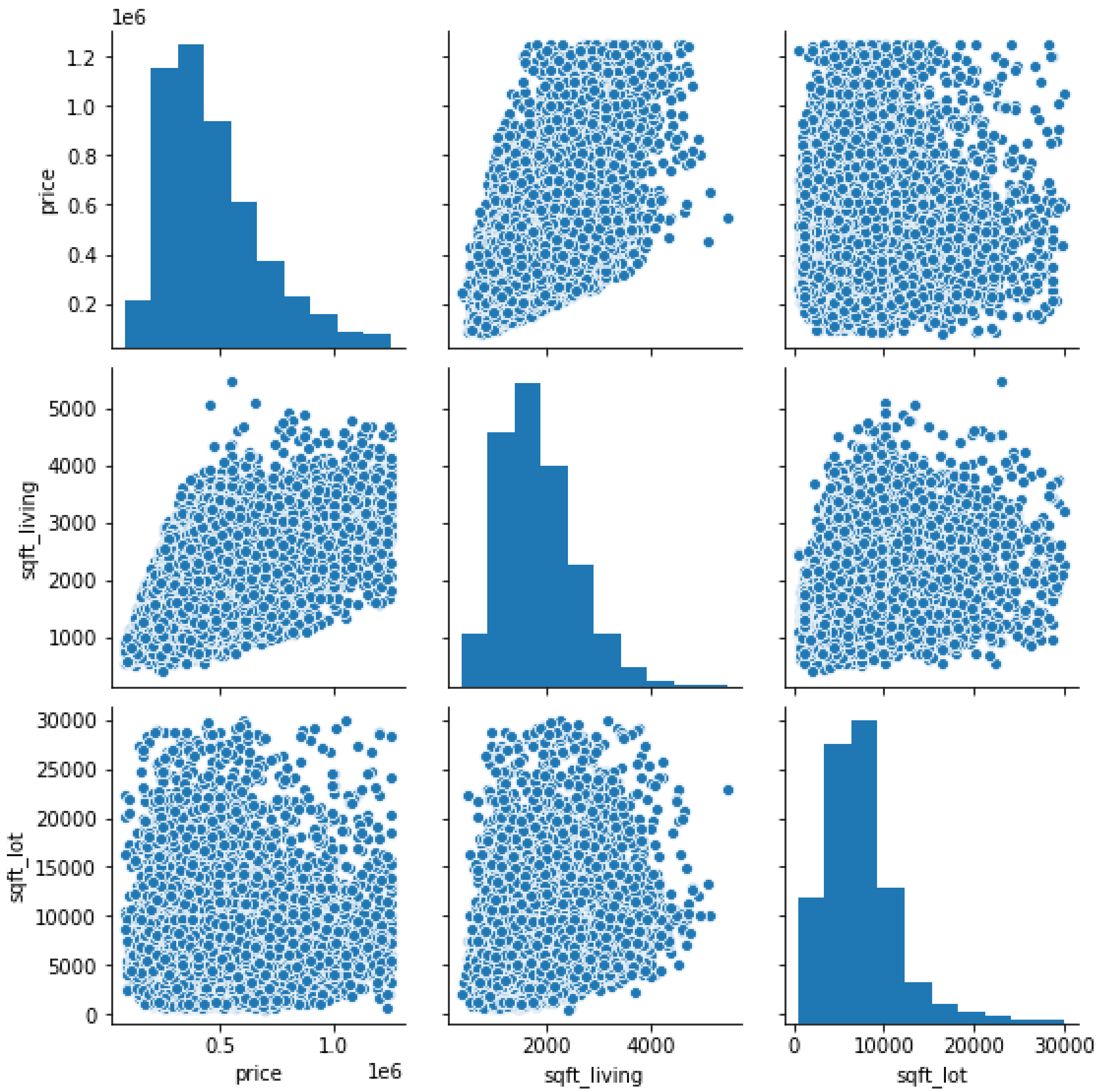
Log transform continuous variables

In [91]:
```
continuous = ['price', 'sqft_living', 'sqft_lot']
```

In [92]:
```
# View distributions
sns.pairplot(data_focused[continuous])
```

```
# continuous are skewed left
```

Out[92]: <seaborn.axisgrid.PairGrid at 0x7fb9a21a9b20>

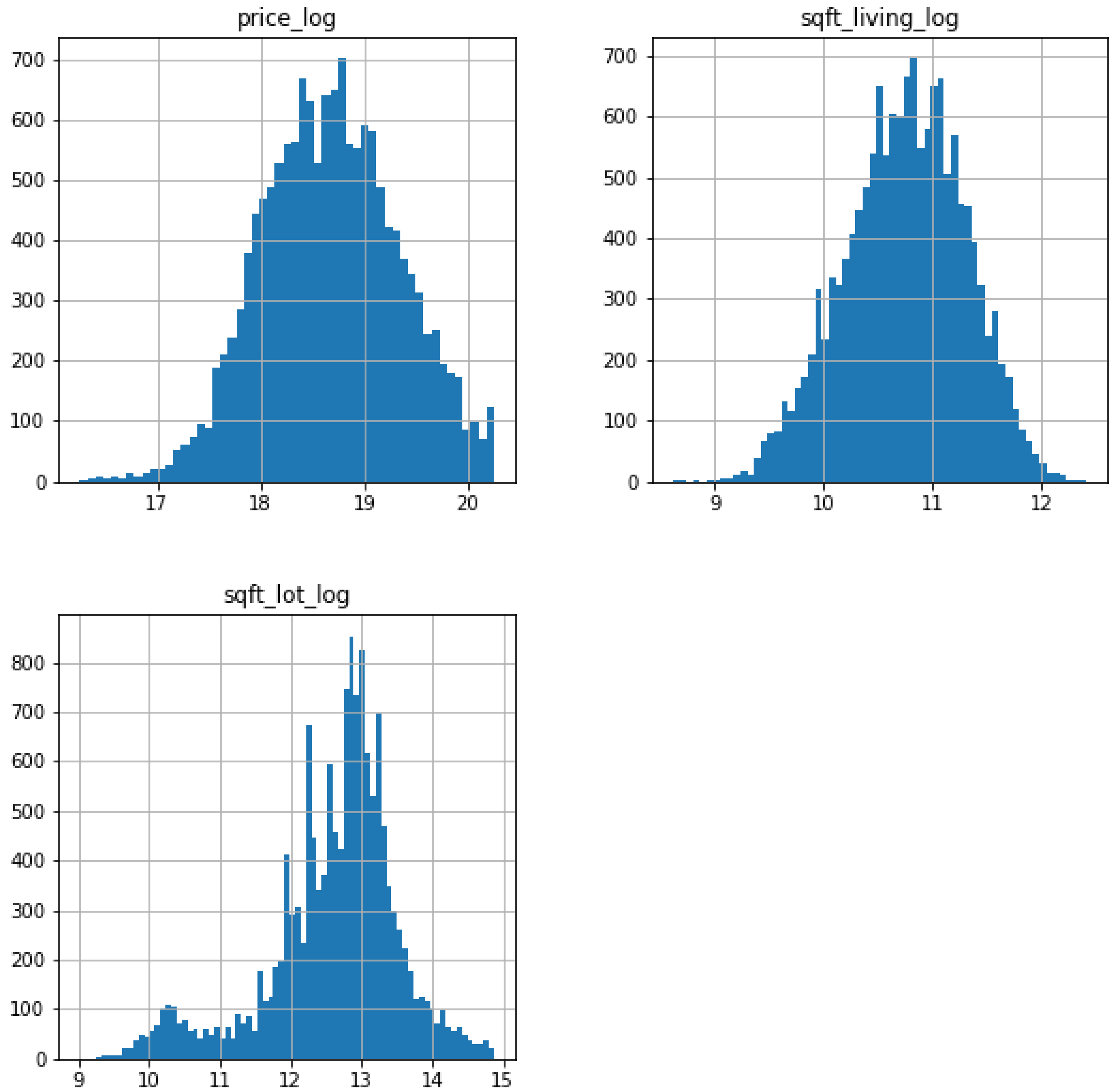


In [93]:
```
# continuous are skewed left, log transform
data_focused_cont = data_focused[continuous]
```

In [94]:
```
#  Create new columns with "_log" for log transformed values
log_names = [f'{column}_log' for column in data_focused_cont.columns]

# Log transform, create a new table with log_names as column names, plot new values
data_focused_log = np.log2(data_focused_cont) #Changed to log 1p insttead of log2
```

```
data_focused_log = np.log2(data_focused_cont) #changed to log 1p instead of log2
data_focused_log.columns = log_names
data_focused_log.hist(figsize=(10, 10), bins='auto')
```

Out[94]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fb9cc85b460>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb9a1f28910>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fb9a44b4b20>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb9a44e7670>]],
      dtype=object)
```



In [95]:
```
# Preview table
data_focused_log
```

```
data_focused_log
```

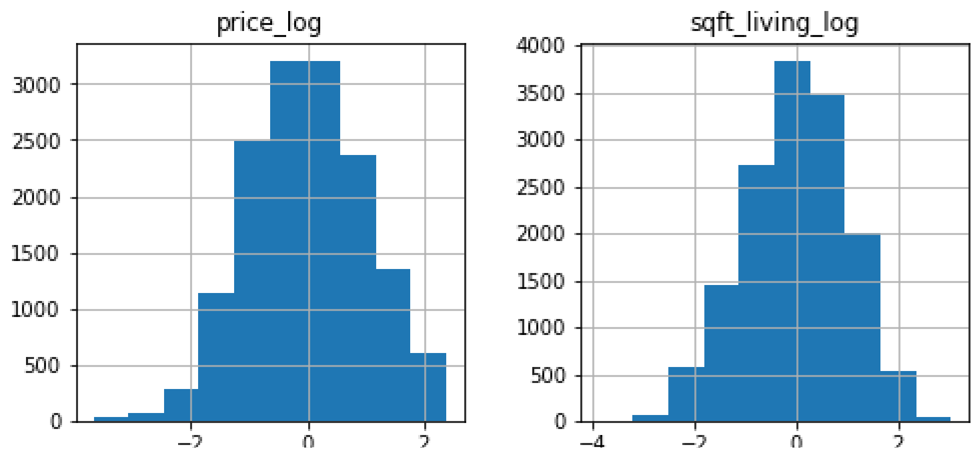
Out[95]:

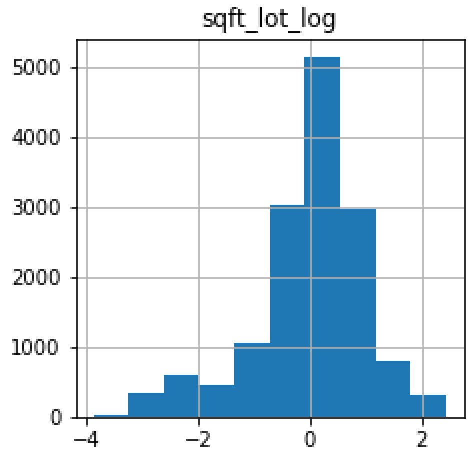
	price_log	sqft_living_log	sqft_lot_log
0	17.759550	10.204571	12.464035
1	19.037247	11.327553	12.822172
2	17.457637	9.588715	13.287712
3	19.204189	10.936638	12.287712
6	17.974213	10.743993	12.735344
...	...	...	...
21589	19.220069	11.299208	12.556267
21590	19.945924	11.777255	12.813781
21591	18.857568	10.355351	10.337622
21592	18.457637	10.579316	10.143383
21593	18.609640	11.173677	12.505067

14716 rows × 3 columns

In [96]:

```python
# Standardize the continuous variables
def normalize(feature):
    return (feature - feature.mean()) / feature.std()

features_final = data_focused_log.apply(normalize)

features_final.hist(figsize  = [8, 8]);
```





In [97]:
```
# Create new table with logs replacing original columns and values
data_focused_with_logs = pd.concat([data_focused, features_final],axis=1)
data_focused_with_logs = data_focused_with_logs.drop(labels=['sqft_lot'], axis=1)
#removed dropping 'price'
```

In [98]:
```
data_focused_with_logs.columns
```

Out[98]:
```
Index(['id', 'date', 'price', 'bedrooms', 'sqft_living', 'floors',
       'waterfront', 'condition', 'grade', 'zipcode', 'lat', 'long',
       'full_bath', 'has_basement', 'has_been_renovated', 'age', 'price_log',
       'sqft_living_log', 'sqft_lot_log'],
      dtype='object')
```

Model 7: Post Log-Transform

In [99]:
```
model_scores(data_focused_with_logs, remove=['price_log', 'price', 'sqft_living', 'date', 'id'], target='price_log', testsize=0.
33,
             rs=42)
# R2 +0.04. Testing > Training, so slightly underfit
# RMSEs Logged. More manual code below to unlog these values to $ amount
```

```
Training Scores:
R2: 0.7
Mean Absolute Error: 0.425
Root Mean Squared Error: 0.546
---
Testing Scores:
R2: 0.705
```

```
Mean Absolute Error: 0.426
Root Mean Squared Error: 0.548
```

Manual code to unlog and get RMSE in dollars

In [100]:
```
def evaluate_model(y_train, y_test, y_pred_train, y_pred_test):
    """Evaluates model on y_train, y_test, y_pred_train, y_pred_test and calculates R2, Mean Absolute Error, and
    Root Mean Absolute Error. This is separated out in order to give access to these datasets for unlogging.
    --
    Inputs:
    - y_train - true target for training set (derived from train_test_split)
    - y_test - true target for test set (derived from train_test_split)
    - y_pred_train - predicted target for training set
    - y_pred_test - predicted target for test set
    --
    No Outputs
    """
    print("Training Scores:")
    print(f"R2: {round(r2_score(y_train, y_pred_train),3)}") #can account for X amount of variance
    print(f"Mean Absolute Error: {round(mean_absolute_error(y_train, y_pred_train),3)}") #about X amount off in predicting pric
e
    print(f"Root Mean Squared Error: {round(mean_squared_error(y_train, y_pred_train, squared=False),3)}")
    print("---")
    print("Testing Scores:")
    print(f"R2: {round(r2_score(y_test, y_pred_test),3)}")
    print(f"Mean Absolute Error: {round(mean_absolute_error(y_test, y_pred_test),3)}")
    print(f"Root Mean Squared Error: {round(mean_squared_error(y_test, y_pred_test, squared=False),3)}")
```

In [101]:
```
# Define X and y
X_cols = [c for c in data_focused_with_logs.columns.to_list() if c not in ['price_log', 'price', 'sqft_living', 'date', 'id']]
X = data_focused_with_logs[X_cols]
y = data_focused_with_logs['price']
# Train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

In [102]:
```
# Log y
y_train_log = np.log1p(y_train)
y_test_log = np.log1p(y_test)
```

In [103]:
```
# Instantiate
lr = LinearRegression()
# Fit training data
lr.fit(X_train, y_train_log)
```

```
# Grab predictions for train and test set, these will now be in log terms
y_pred_train_log = lr.predict(X_train)
y_pred_test_log = lr.predict(X_test)
```

In [104]:
```
evaluate_model(y_train_log, y_test_log, y_pred_train_log, y_pred_test_log)
```

```
Training Scores:
R2: 0.7
Mean Absolute Error: 0.195
Root Mean Squared Error: 0.25
---
Testing Scores:
R2: 0.705
Mean Absolute Error: 0.195
Root Mean Squared Error: 0.251
```

In [105]:
```
# Unlog y_pred_test
y_pred_train = np.exp(y_pred_train_log)
y_pred_test = np.exp(y_pred_test_log)
```

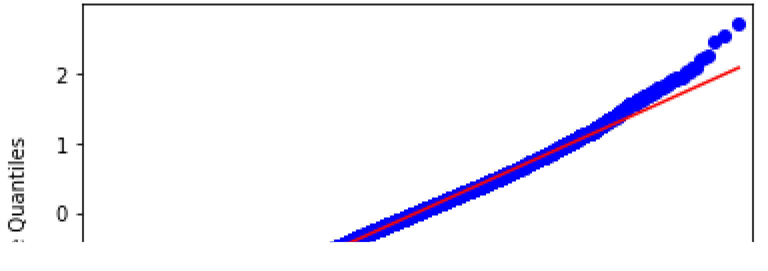
In [106]:
```
# Recalculate Train and Test RMSE
print(f"Train RMSE: {mean_squared_error(y_train, y_pred_train, squared=False):.3f}")
print(f"Test RMSE: {mean_squared_error(y_test, y_pred_test, squared=False):.3f}")
# Model about $129k amount off in predicting price
```

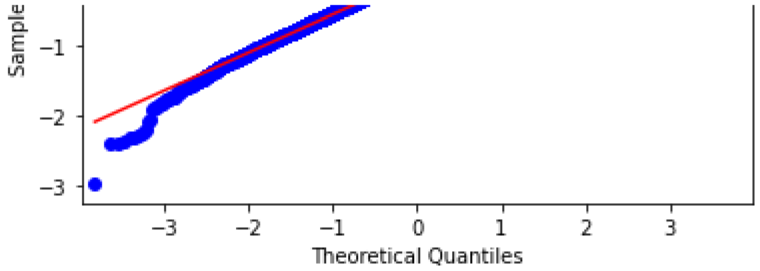
```
Train RMSE: 127936.893
Test RMSE: 128559.926
```

In [107]:
```
#Sanity check, matches logged RMSE from evaluate_model
print(f"Train RMSE Logged: {round(mean_squared_error(y_train_log, y_pred_train_log, squared=False),3)}")
print(f"Test RMSE Logged: {round(mean_squared_error(y_test_log, y_pred_test_log, squared=False),3)}")
```

```
Train RMSE Logged: 0.25
Test RMSE Logged: 0.251
```

In [108]:
```
qq_plot(data_focused_with_logs, remove=['price_log', 'price', 'sqft_living', 'date', 'id'], target='price_log')
# qqplot line MASSIVELY improved, outliers ~3 deviations from mean impacted
```





In [109]:
```
# Update master variable
data_focused = data_focused_with_logs
```

Model 8: OHE zipcode

In [110]:
```
# Look into zipcodes
set(data_focused['zipcode'])
```

Out[110]:
```
{98001,
 98002,
 98003,
 98004,
 98005,
 98006,
 98007,
 98008,
 98011,
 98023,
 98027,
 98028,
 98030,
 98031,
 98032,
 98033,
 98034,
 98039,
 98040,
 98042,
 98052,
 98055,
 98056,
 98058,
 98059,
 98070,
 98072,
```

```
 98092,
 98102,
 98103,
 98105,
 98106,
 98107,
 98108,
 98109,
 98112,
 98115,
 98116,
 98117,
 98118,
 98119,
 98122,
 98125,
 98126,
 98133,
 98136,
 98144,
 98146,
 98148,
 98155,
 98166,
 98168,
 98177,
 98178,
 98188,
 98198,
 98199}
```

In [111]:
```
cat_cols = ['zipcode']
```

In [112]:
```
data_focused_zipohe = pd.get_dummies(data_focused, columns=cat_cols)
```

In [113]:
```
data_focused_zipohe.head()
```

Out[113]:

	id	date	price	bedrooms	sqft_living	floors	waterfront	condition	grade	lat	...	zipcode_98146	zipcode_98148	zipcode_98155	zipcode
0	7129300520	2014-10-13	221900.0	3	1180	1.0	0.0	3	7	47.5112	...	0	0	0	0
1	6414100192	2014-12-09	538000.0	3	2570	2.0	0.0	3	7	47.7210	...	0	0	0	0

2	5631500400	2015-02-25	180000.0	2	770	1.0	0.0	3	6	47.7379	...	0	0	0	0
3	2487200875	2014-12-09	604000.0	4	1960	1.0	0.0	5	7	47.5208	...	0	0	0	0
6	1321400060	2014-06-27	257500.0	3	1715	2.0	0.0	3	7	47.3097	...	0	0	0	0

5 rows × 75 columns

In [114]:
```
len(list(data_focused_zipohe.columns))
```

Out[114]: 75

In [115]:
```
model_scores(data_focused_zipohe, remove=['price_log', 'price', 'sqft_living', 'date', 'id'], target =['price_log'], testsize=0.33, rs=42)
# R-squared from 0.70 to .83
# Training and Test score are extremely close - if anything very slightly overfit with Training .002 > Testing
# MSE and RMSE trending down from .425 and .652
```

```
Training Scores:
R2: 0.836
Mean Absolute Error: 0.3
Root Mean Squared Error: 0.403
---
Testing Scores:
R2: 0.838
Mean Absolute Error: 0.301
Root Mean Squared Error: 0.406
```

In [116]:
```
high_corr(data_focused_zipohe, 0.6)
# No additional high correlation variables of those not removed from Dataframe
```

Out[116]:

	level_0	level_1	0
88	price	price_log	0.960251
151	bedrooms	sqft_living	0.612915
163	bedrooms	sqft_living_log	0.645531
224	sqft_living	bedrooms	0.612915
229	sqft_living	grade	0.660963
237	sqft_living	sqft_living_log	0.971761

521	grade	sqft_living	0.660963
533	grade	sqft_living_log	0.650787
1037	price_log	price	0.960251
1112	sqft_living_log	bedrooms	0.645531
1113	sqft_living_log	sqft_living	0.971761
1117	sqft_living_log	grade	0.650787

Manual code to unlog and get RMSE in dollars

In [117]:
```
# Define X and y
X_cols = [c for c in data_focused_zipohe.columns.to_list() if c not in ['price_log', 'price',  'sqft_living', 'date', 'id']]
X = data_focused_zipohe[X_cols]
y = data_focused_zipohe['price']
# Train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

In [118]:
```
# Log y
y_train_log = np.log1p(y_train)
y_test_log = np.log1p(y_test)
```

In [119]:
```
# Instantiate
lr = LinearRegression()
# Fit training data
lr.fit(X_train, y_train_log)
# Grab predictions for train and test set, these will now be in log terms
y_pred_train_log = lr.predict(X_train)
y_pred_test_log = lr.predict(X_test)
```

In [120]:
```
evaluate_model(y_train_log, y_test_log, y_pred_train_log, y_pred_test_log)
```

```
Training Scores:
R2: 0.836
Mean Absolute Error: 0.137
Root Mean Squared Error: 0.185
---
Testing Scores:
R2: 0.838
Mean Absolute Error: 0.138
Root Mean Squared Error: 0.186
```

Root Mean Squared Error: 0.186

In [121]:
```
# Unlog y_pred_test
y_pred_train = np.exp(y_pred_train_log)
y_pred_test = np.exp(y_pred_test_log)
```

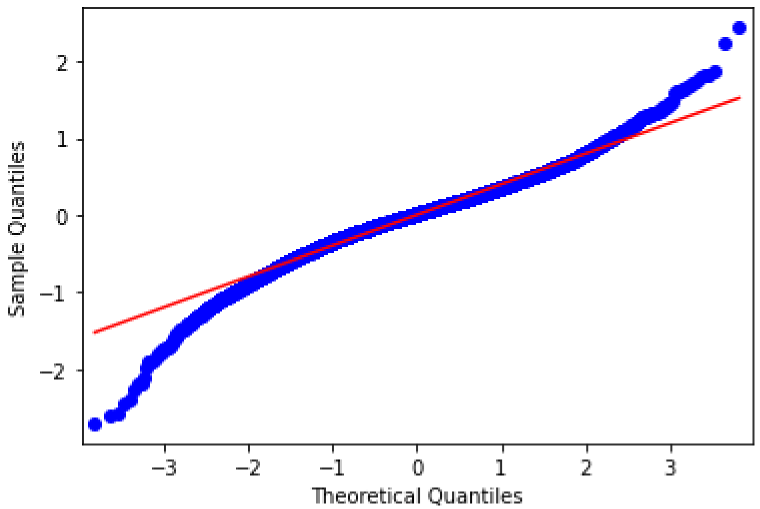
In [122]:
```
# Recalculate Train and Test RMSE
print(f"Train RMSE: {mean_squared_error(y_train, y_pred_train, squared=False):.3f}")
print(f"Test RMSE: {mean_squared_error(y_test, y_pred_test, squared=False):.3f}")
# Model about $129k amount off in predicting price
```

```
Train RMSE: 90505.332
Test RMSE: 90878.305
```

In [123]:
```
#Sanity check, matches logged RMSE from evaluate_model
print(f"Train RMSE Logged: {round(mean_squared_error(y_train_log, y_pred_train_log, squared=False),3)}")
print(f"Test RMSE Logged: {round(mean_squared_error(y_test_log, y_pred_test_log, squared=False),3)}")
```

```
Train RMSE Logged: 0.185
Test RMSE Logged: 0.186
```

In [124]:
```
qq_plot(data_focused_zipohe, remove=['price_log', 'price', 'sqft_living', 'date', 'id'], target='price_log')
# qqplot line drops off at extremes
```



In [125]:
```
# Update master variable
data_focused = data_focused_zipohe
```

Increased R-squared Test Score from 0.680 to 0.838 (23%), and Decreased RMSE from 204432 to 90878 (55%)

Model 9: Apply Standard Scaler to data

In [126]:
```
model_scores_stanscale(data_focused, remove=['price_log', 'price', 'sqft_living', 'date', 'id'], target =['price_log'], testsiz
e=0.33,
                       rs=42)
# Same R2 scores (as expected)
```

```
Training Scores:
R2: 0.836
Mean Absolute Error: 0.3
Root Mean Squared Error: 0.403
---
Testing Scores:
R2: 0.838
Mean Absolute Error: 0.301
Root Mean Squared Error: 0.406
```

Manual code to unlog and get RMSE in dollars

In [127]:
```
# Define X and y
X_cols = [c for c in data_focused.columns.to_list() if c not in ['price_log', 'price', 'sqft_living', 'date', 'id']]
X = data_focused[X_cols]
y = data_focused['price']
# Train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

In [128]:
```
# Log y
y_train_log = np.log1p(y_train)
y_test_log = np.log1p(y_test)
```

In [129]:
```
# Instantiate a new scaler to scale our data with Standard Scaler
scaler = StandardScaler()
# Train scaler on training data, then fit to testing
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [130]:
```
# Instantiate
lr = LinearRegression()
# Fit training data
lr.fit(X_train, y_train_log)
```

```
# Grab predictions for train and test set, these will now be in log terms
y_pred_train_log = lr.predict(X_train)
y_pred_test_log = lr.predict(X_test)
```

In [131]:

```
evaluate_model(y_train_log, y_test_log, y_pred_train_log, y_pred_test_log)
```

```
Training Scores:
R2: 0.836
Mean Absolute Error: 0.137
Root Mean Squared Error: 0.185
---
Testing Scores:
R2: 0.838
Mean Absolute Error: 0.138
Root Mean Squared Error: 0.186
```

In [132]:

```
# Unlog y_pred_test
y_pred_train = np.exp(y_pred_train_log)
y_pred_test = np.exp(y_pred_test_log)
```

In [133]:

```
# Recalculate Train and Test RMSE
print(f"Train RMSE: {mean_squared_error(y_train, y_pred_train, squared=False):.3f}")
print(f"Test RMSE: {mean_squared_error(y_test, y_pred_test, squared=False):.3f}")
# Model about $129k amount off in predicting price
```

```
Train RMSE: 90505.332
Test RMSE: 90878.305
```

In [134]:

```
# Sanity check, matches logged RMSE from evaluate_model
print(f"Train RMSE Logged: {round(mean_squared_error(y_train_log, y_pred_train_log, squared=False),3)}")
print(f"Test RMSE Logged: {round(mean_squared_error(y_test_log, y_pred_test_log, squared=False),3)}")
```

```
Train RMSE Logged: 0.185
Test RMSE Logged: 0.186
```

In [135]:

```
# Check X columns
X.columns
```

Out[135]:

```
Index(['bedrooms', 'floors', 'waterfront', 'condition', 'grade', 'lat', 'long',
       'full_bath', 'has_basement', 'has_been_renovated', 'age',
       'sqft_living_log', 'sqft_lot_log', 'zipcode_98001', 'zipcode_98002',
       'zipcode_98003', 'zipcode_98004', 'zipcode_98005', 'zipcode_98006',
       'zipcode_98007', 'zipcode_98008', 'zipcode_98011', 'zipcode_98023',
       'zipcode_98027', 'zipcode_98028', 'zipcode_98030', 'zipcode_98031',
       'zipcode_98032', 'zipcode_98033', 'zipcode_98034', 'zipcode_98039',
       'zipcode_98040', 'zipcode_98042', 'zipcode_98052', 'zipcode_98055',
```

```
       'zipcode_98056', 'zipcode_98058', 'zipcode_98059', 'zipcode_98070',
       'zipcode_98072', 'zipcode_98092', 'zipcode_98102', 'zipcode_98103',
       'zipcode_98105', 'zipcode_98106', 'zipcode_98107', 'zipcode_98108',
       'zipcode_98109', 'zipcode_98112', 'zipcode_98115', 'zipcode_98116',
       'zipcode_98117', 'zipcode_98118', 'zipcode_98119', 'zipcode_98122',
       'zipcode_98125', 'zipcode_98126', 'zipcode_98133', 'zipcode_98136',
       'zipcode_98144', 'zipcode_98146', 'zipcode_98148', 'zipcode_98155',
       'zipcode_98166', 'zipcode_98168', 'zipcode_98177', 'zipcode_98178',
       'zipcode_98188', 'zipcode_98198', 'zipcode_98199'],
      dtype='object')
```

In [136]:

```
# Round coefficients 3 decimals to make more legible
lr.coef_ = np.around(lr.coef_,3)
```

In [137]:

```
# Look at the coefficients with the names of each col
var_coeff_dict = dict(zip(X.columns, lr.coef_))
# Create DataFrame for organization
var_coeff_df = pd.DataFrame.from_dict(dict(zip(X.columns, lr.coef_)), orient='index')
var_coeff_df
```

Out[137]:

	0
bedrooms	-0.014
floors	0.007
waterfront	0.604
condition	0.051
grade	0.121
...	...
zipcode_98177	0.042
zipcode_98178	-0.300
zipcode_98188	-0.394
zipcode_98198	-0.404
zipcode_98199	0.311

70 rows × 1 columns

In [138]:

```
# Look for highest coefficients. What zipcodes drive price up?
# zipcode source: https://www.unitedstateszipcodes.org/98011/
```

```
var_coeff_df.sort_values(by=0, ascending=False).head(6)
# 98039: Medina
# 98004: Bellevue
# 98112: (Seattle: Mann|Central Area)
# 98102: (Seattle: Eastlake|Cascade)
# 98109: (Seattle: Westlake|Cascade)
# Medina, Bellevue and parts of Seattle are highest positive influencers of price
```

Out[138]:

	0
zipcode_98039	0.753
waterfront	0.604
zipcode_98004	0.581
zipcode_98112	0.477
zipcode_98102	0.471
zipcode_98109	0.451

In [139]:

```
# Look at lowest coefficients. What zipcodes drive price down?
var_coeff_df.sort_values(by=0).head(5)
# 98023/98003/98001/98002: Auburn
# 98032: Kent
# Auburn, Kent are highest negative influencers of price
```

Out[139]:

	0
zipcode_98023	-0.518
zipcode_98032	-0.505
zipcode_98003	-0.470
zipcode_98001	-0.468
zipcode_98002	-0.444

In [140]:

```
# What areas are around mean influence?
print(f'Mean Coefficient: {round(var_coeff_df[var_coeff_df.columns[0]].mean(),3)}')
print(f'Median Coefficient: {round(var_coeff_df[var_coeff_df.columns[0]].median(),3)}')
var_coeff_df[(var_coeff_df[0]>=-0.01) & (var_coeff_df[0]<=0.02)].sort_values(by=0).head(60)
# 98072: Woodinville
# 98027: Issaquah (far East, boonies)
# 98118: Seattle (Seward Park|Ranier Valley)
```

```
Mean Coefficient: 0.011
Median Coefficient: 0.016
```

Out[140]:

	0
zipcode_98072	-0.006
zipcode_98027	-0.000
zipcode_98118	-0.000
age	0.001
floors	0.007

In [141]:

```
lr.intercept_
```

Out[141]: -45.154014479598125

In [142]:

```
var_coeff_df.sort_values(by=0, ascending=False).head(60)
# Beyond zipcode waterfront is highest driver of price,
# then latitude (makes sense with southern Seattle area being mean)
# grade is next highest, has_been_renovated, and condition
```

Out[142]:

	0
zipcode_98039	0.753
waterfront	0.604
zipcode_98004	0.581
zipcode_98112	0.477
zipcode_98102	0.471
zipcode_98109	0.451
zipcode_98119	0.424
zipcode_98105	0.408
zipcode_98040	0.397
zipcode_98107	0.320
zipcode_98122	0.312
zipcode_98199	0.311
zipcode_98115	0.291

zipcode_98103	0.284
zipcode_98033	0.276
zipcode_98117	0.270
zipcode_98005	0.270
zipcode_98116	0.255
zipcode_98006	0.219
zipcode_98008	0.209
zipcode_98136	0.205
zipcode_98052	0.191
zipcode_98007	0.190
zipcode_98144	0.184
sqft_living_log	0.172
lat	0.160
grade	0.121
has_been_renovated	0.065
condition	0.051
sqft_lot_log	0.049
zipcode_98034	0.048
zipcode_98126	0.047
zipcode_98177	0.042
zipcode_98125	0.040
full_bath	0.025
floors	0.007
age	0.001
zipcode_98118	-0.000
zipcode_98027	-0.000

zipcode_98072	-0.006
bedrooms	-0.014
has_basement	-0.031
zipcode_98011	-0.042
zipcode_98059	-0.066
zipcode_98028	-0.094
zipcode_98133	-0.094
zipcode_98056	-0.108
zipcode_98155	-0.117
zipcode_98108	-0.127
zipcode_98106	-0.170
zipcode_98166	-0.197
zipcode_98146	-0.236
zipcode_98070	-0.264
zipcode_98058	-0.288
zipcode_98055	-0.298
zipcode_98178	-0.300
zipcode_98042	-0.355
zipcode_98148	-0.363
zipcode_98031	-0.364
zipcode_98030	-0.380

Data Analysis

Looking at the results from the last model, it was clear that zipcodes were main drivers of price especially with that being a large portion the variables considered. From here, analysis was approached by analyzed the top 5 positive price driving zipcodes (df: positive_zips) and the top 5 negative price driving zipcodes (df: negative_zips). This is where it was found that there were around twice as many home records available in the negative zipcodes than the positive ones.

Top 5 Positive Price Driving Zipcodes:

- 98039: Medina
- 98004: Bellevue
- 98112: (Seattle: Mann|Central Area)
- 98109: (Seattle: Westlake|Cascade)
- 98102: (Seattle: Eastlake|Cascade)

Top 5 Negative Price Driving Zipcodes:

- 98023/98001/98003/98002: Auburn
- 98032: Kent

Additionally top home features that drove price beyond zipcode location were plotted and analyzed.

Top Home Feature Positive Price Drivers:

- waterfront
- sqft_living
- grade
- has_been_renovated
- condition

In [143]:
```
# Create table for zips that drive price up
# 98039: Medina
# 98004: Bellevue
# 98112: (Seattle: Mann|Central Area)
# 98102: (Seattle: Eastlake|Cascade)
# 98109: (Seattle: Westlake|Cascade)
positive_zips = data_focused[(data_focused['zipcode_98039'] == 1) | (data_focused['zipcode_98004'] == 1) |
             (data_focused['zipcode_98112'] == 1) | (data_focused['zipcode_98109'] == 1) |
             (data_focused['zipcode_98102'] == 1)]

# Create column label, so when merge data for analysis, will know which rows are Positive zips
positive_zips['label']='Positive'
```

```
<ipython-input-143-d544d941bf81>:12: 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: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy
  positive_zips['label']='Positive'
```

In [144]:
```
len(positive_zips)
```

Out[144]: 539

In [145]:
```python
# Create table for zips that drive price down
# 98032: Kent
# 98023/98001/98003/98002: Auburn
negative_zips = data_focused[(data_focused['zipcode_98032'] == 1) | (data_focused['zipcode_98023'] == 1) |
              (data_focused['zipcode_98001'] == 1) | (data_focused['zipcode_98003'] == 1) |
              (data_focused['zipcode_98002'] == 1)]

# Create column label, so when merge data for analysis, will know which rows are Negative zips
negative_zips['label']='Negative'
```

```
<ipython-input-145-8b2beaa77979>:9: 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: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy
  negative_zips['label']='Negative'
```

In [146]:
```python
len(negative_zips)
# Over twice as many negative zip records than positive. Explore balancing this more in future.
```

Out[146]: 1388

In [147]:
```python
# Merge positive_zips and negative_zips tables
extreme_zips = pd.concat([positive_zips, negative_zips])
extreme_zips = extreme_zips.reset_index(drop=True)
extreme_zips
```

Out[147]:

	id	date	price	bedrooms	sqft_living	floors	waterfront	condition	grade	lat	...	zipcode_98148	zipcode_98155	zipcode_98166	zip
0	3303700376	2014-12-01	667000.0	3	1400	1.5	0.0	5	8	47.6221	...	0	0	0	0
1	3394100030	2014-09-09	975000.0	4	2720	2.0	0.0	3	10	47.5815	...	0	0	0	0
2	1952200240	2014-06-11	850830.0	3	2070	1.5	0.0	5	9	47.6415	...	0	0	0	0
3	2450000295	2014-10-07	1090000.0	3	2920	2.0	0.0	3	8	47.5814	...	0	0	0	0
4	809001525	2014-06-25	890000.0	4	2550	2.0	0.0	3	8	47.6354	...	0	0	0	0

...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1922	3304030220	2015-04-21	480000.0	4	2940	2.0	0.0	3	9	47.3444	...	0	0	0	0
1923	2909310100	2014-10-15	332000.0	4	2380	2.0	0.0	3	7	47.2815	...	0	0	0	0
1924	5007500120	2015-02-26	341780.0	4	2260	2.0	0.0	3	7	47.3507	...	0	0	0	0
1925	9578500790	2014-11-11	399950.0	3	3087	2.0	0.0	3	8	47.2974	...	0	0	0	0
1926	8956200760	2014-10-13	541800.0	4	3118	2.0	0.0	3	9	47.2931	...	0	0	0	0

1927 rows × 76 columns

In [148]:
```
# Stat overview
extreme_zips.groupby(by='label').agg({'price': ['mean', 'median'], 'age': ['mean'], 'grade': ['mean', 'median'],
                                      'condition':['mean', 'median'], 'bedrooms':['mean'], 'full_bath':['mean']})
# Median home price is 67% more in positive zips.
# Homes older in positive neighborhoods
# Avg grade is 7-8 for both positive and negative. This is on a scale of 13 and is average grade.
# 7 = "Average grade of construction and design. Commonly seen in plats and older sub-divisions."
# Condition is 3 out of 5, also noting average home
# Both averaging 3 beds, 1-2 full baths
```

Out[148]:

	price		**age**	**grade**		**condition**		**bedrooms**	**full_bath**
	mean	**median**	**mean**	**mean**	**median**	**mean**	**median**	**mean**	**mean**
label									
Negative	269631.689481	256641.5	37.695245	7.324928	7	3.443084	3	3.366715	1.600144
Positive	791265.068646	775000.0	62.243043	7.842301	8	3.443414	3	3.204082	1.651206

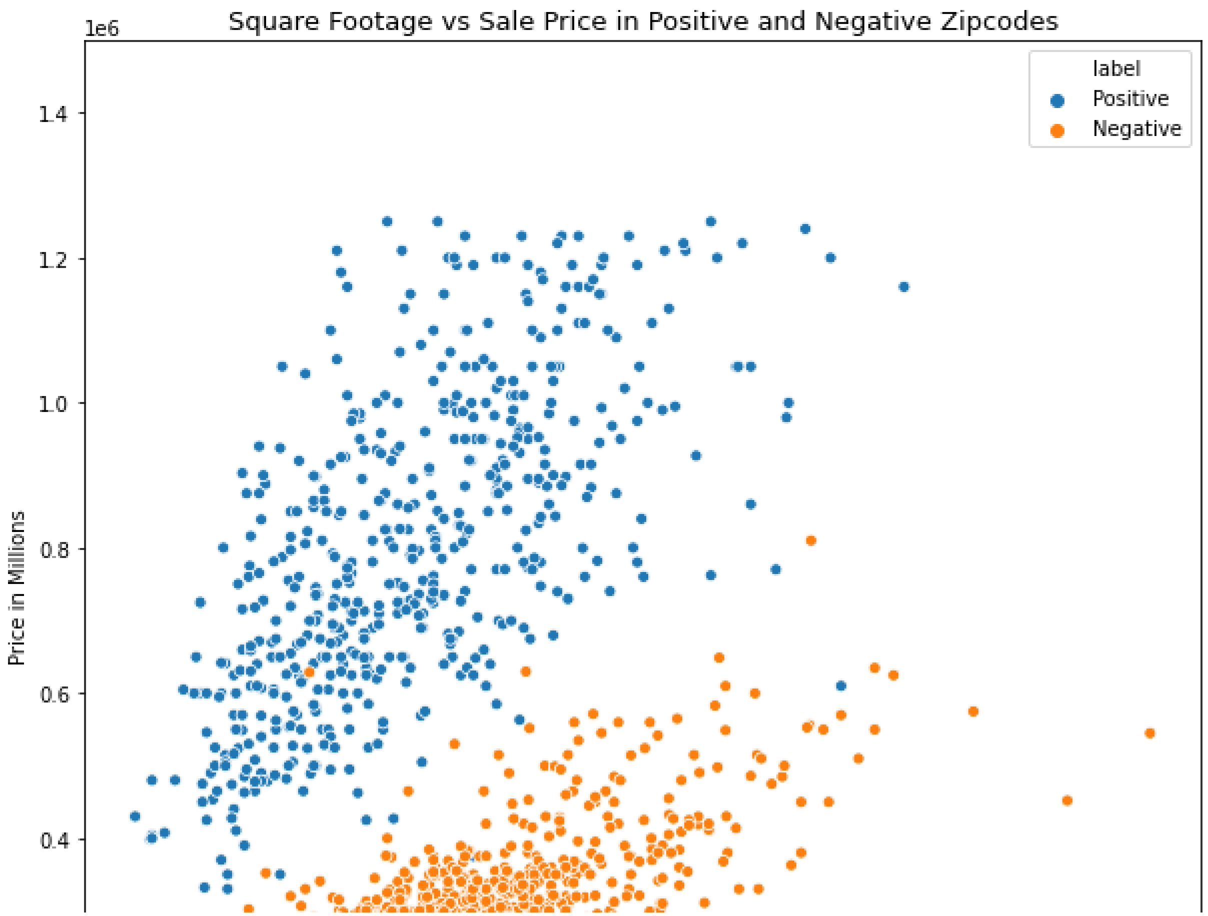
In [149]:
```
# Look into distribution and stat overviews of home features driving price and standard qualities
# Waterfront is model's top driver
extreme_zips.groupby(by='waterfront')['id'].count()
# Model has limited information about homes with waterfronts. Note to look into this data point and
# expansion of data available in modeling
```

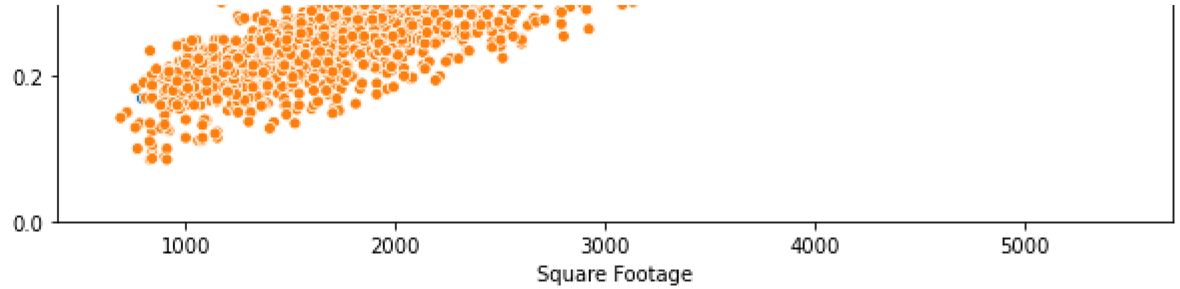
Out[149]: waterfront

```
0.0    1926
1.0       1
Name: id, dtype: int64
```

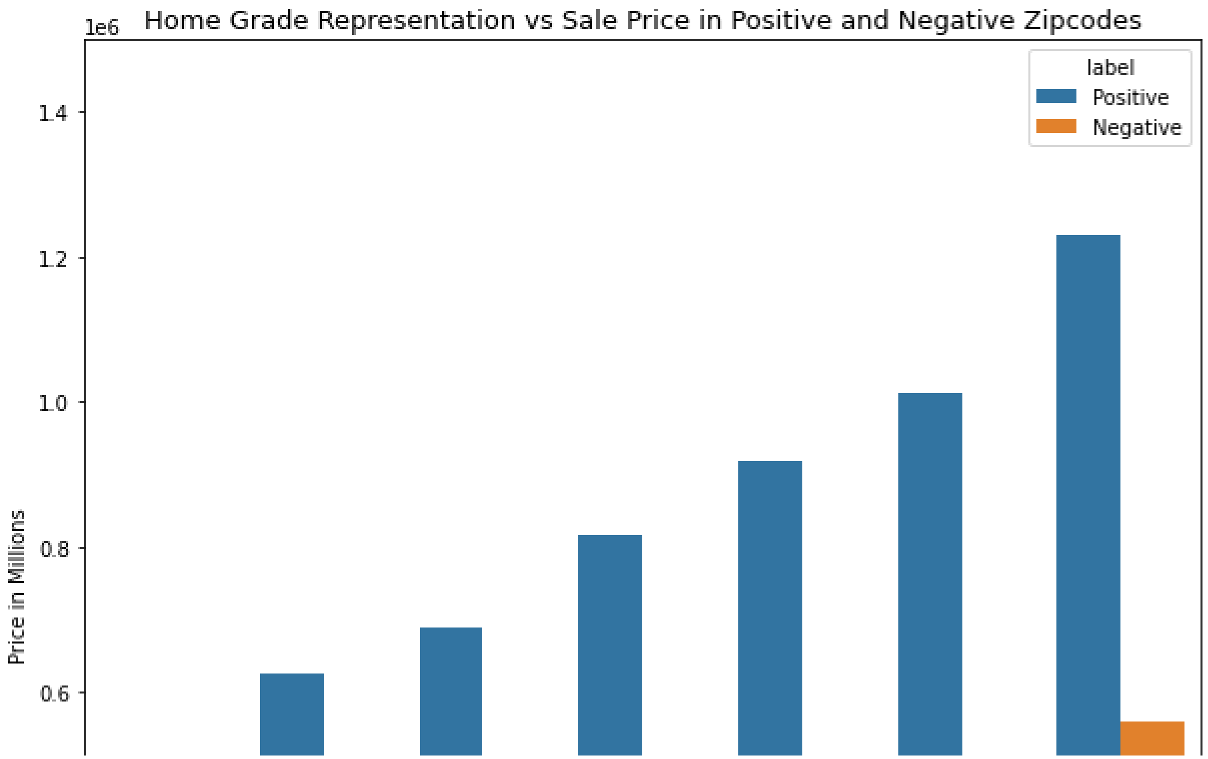
In [153]:

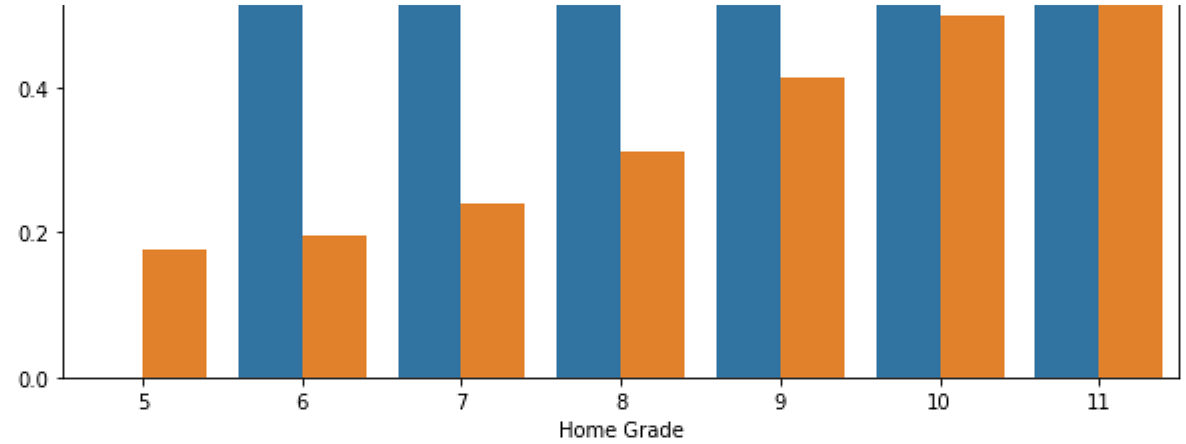
```python
# Sqft_living was next highest driver of price. Look at scatter plot in positive versus negative zipcodes
# and relation of total square footage to price
plt.figure(figsize=(10,10))
sns.scatterplot(x=extreme_zips['sqft_living'], y=extreme_zips['price'], hue=extreme_zips['label'], ci=None)
plt.xlabel(xlabel='Square Footage')
plt.ylabel(ylabel='Price in Millions')
plt.ylim(0, 1500000)
plt.title('Square Footage vs Sale Price in Positive and Negative Zipcodes', size=13 )
plt.savefig("images/Pos_Neg_Sq_Ft_Comp.png")
plt.show()
# These are almost parallel paths with negative zipcodes sitting consistently below the prices for the positive ones
# For homes in negative zipcodes, there is concentration especially for home from 1000-3000 square feet.
# Homes in positive zipcodes also have more variation in price range within a given zipcode. This also could be
# showing more prevalently as there is less records available on positive zipcodes
```





In [154]:
```
# Grade was next highest driver of price. Look at availability of grades in positive versus negative zipcodes
# and relation of grade to price
plt.figure(figsize=(10,10))
sns.barplot(x=extreme_zips['grade'], y=extreme_zips['price'], hue=extreme_zips['label'], ci=None)
plt.xlabel(xlabel='Home Grade')
plt.ylabel(ylabel='Price in Millions')
plt.ylim(0, 1500000)
plt.title('Home Grade Representation vs Sale Price in Positive and Negative Zipcodes', size=13 )
plt.savefig("images/Pos_Neg_Home_Grade_Comp")
plt.show()
# Comparisons show that there is a consistent gap of at least $400k even in the lower range of grades between
# homes in positive versus homes in negative zipcodes. Also positive zip homes are ~3x the price at similar grades
# versus negative
```





In [155]:

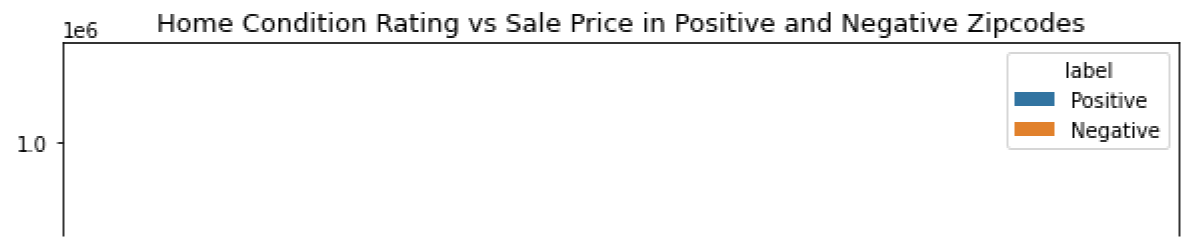
```
# Renovation is model's top driver
extreme_zips.groupby(by='has_been_renovated')['id'].count()
# Model has limited information about homes with renovations. Note to look into this data point and
# expansion of data available in modeling
```

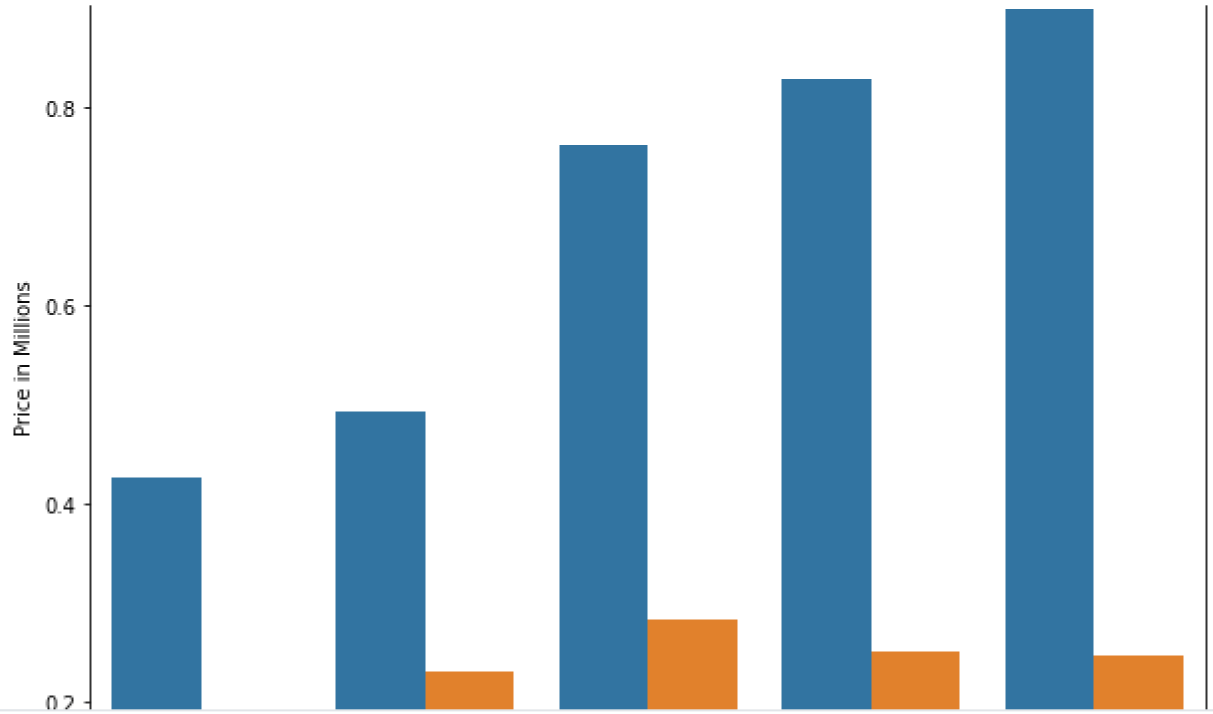
Out[155]:

```
has_been_renovated
0    1877
1      50
Name: id, dtype: int64
```

In [156]:

```
# Condition was next highest driver of price. Look at availability of grades in positive versus negative zipcodes
# and relation of grade to price
plt.figure(figsize=(10,10))
sns.barplot(x=extreme_zips['condition'], y=extreme_zips['price'], hue=extreme_zips['label'], ci=None)
plt.xlabel(xlabel='Home Condition')
plt.ylabel(ylabel='Price in Millions')
plt.ylim(0, 1100000)
plt.title('Home Condition Rating vs Sale Price in Positive and Negative Zipcodes', size=13 )
plt.savefig("images/Pos_Neg_Home_Condition_Comp")
plt.show()
# Comparisons show that there is a consistent gap of at least $300k between homes in positive
# versus homes in negative zipcodes in the same condititon. Can also see that homes in negative zips actually
# peak in price in condition rating of 3 and taper off in ratings 4 and 5, while positive zip home prices trend
# upward consistently
```




0.8
0.6
0.4
0.2
Price in Millions

/