1 Phase 2 Project

- Student: Jonathan Holt
- DS Flex

1.1 The Problem

Trailblazer Renovations is a home contractor that specializes in home additions and renovations. They are based in Northern California, but have been getting requests from homeowners in Washington State. They are interested in expanding their business, but want to gauge the situation before diving in.

They have already decided to focus their efforts on King County Washington as it contains a large population of people (including metropolitan Seattle) and could be a good opportunity for growth for them.

I have been tasked with analyzing home prices in King County to determine where the market for home renovations lies so that Trailblazer can meet that need. Part of this analysis is to determine which renovation types have the biggest increase in home value so that those services can be the focus of their marketing campaign.

While they plan on marketing to all of King County, they want to know if there are any particular areas that they should focus a greater portion of their energy in marketing to.

1.2 The Data

I have the king county dataset which gives information on home values in King County. I have also found information clarifying what the different building grade and condition types are from the King County government website.

source: Data/column_names.md

Here is what is contained in the raw data:

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

1.3 Glossary

- additional information about the Data that I have

1.3.1 Building Condition Explaination

https://info.kingcounty.gov/assessor/esales/Glossary.aspx?type=r#d (https://info.kingcounty.gov/assessor/esales/Glossary.aspx?type=r#d) (accessed 12/6/2021)

Relative to age and grade. Coded 1-5.

1) Poor:

- Worn out. Repair and overhaul needed on painted surfaces, roofing, plumbing, heating and numerous functional inadequacies. Excessive deferred maintenance and abuse, limited value-in-use, approaching abandonment or major reconstruction; reuse or change in occupancy is imminent. Effective age is near the end of the scale regardless of the actual chronological age.

2) Fair:

- Badly worn. Much repair needed. Many items need refinishing or overhauling, deferred maintenance obvious, inadequate building utility and systems all shortening the life expectancy and increasing the effective age.

3) Average:

- Some evidence of deferred maintenance and normal obsolescence with age in that a few minor repairs are needed, along with some refinishing. All major components still functional and contributing toward an extended life expectancy. Effective age and utility is standard for like properties of its class and usage.

4) Good:

- No obvious maintenance required but neither is everything new. Appearance and utility are above the standard and the overall effective age will be lower than the typical property.

5) Very Good:

- All items well maintained, many having been overhauled and repaired as they have shown signs of wear, increasing the life expectancy and lowering the effective age with little deterioration or obsolescence evident with a high degree of utility.

1.3.2 Building Grade Explanation

https://info.kingcounty.gov/assessor/esales/Glossary.aspx?type=r#d (https://info.kingcounty.gov/assessor/esales/Glossary.aspx?type=r#d) (accessed 12/6/2021)

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

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

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

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

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

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

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

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

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

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

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

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

1.4 Questions to Answer

1. What features have the biggest effect on home price?
2. Are these features able to be changed/renovated, or are they fixed?
3. What are the best features that Trailblazer should market to potential clients?
4. Who should Trailblazer target their marketing campaign toward?
5. Are there any features that I recommend against?

2 Data Preparation

2.1 Importing Data & Settings

In [1]:

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant
from statsmodels.formula.api import ols
from statsmodels.regression.linear_model import OLS
from sklearn.model_selection import train_test_split
import scipy.stats as stats

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

pd.set_option('display.max_rows', 1000) #change the amount of rows displayed
plt.style.use('seaborn')

#function for displaying money in millions.
def display_millions(x, pos):
    return '${:1.1f}M'.format(x*1e-6)

#function for displaying money in thousands.
def display_thousands(x, pos):
    return '${:1.1f}K'.format(x*1e-3)

df = pd.read_csv('Data/kc_house_data.csv')
df.head()
```

executed in 1.30s, finished 06:21:58 2021-12-22

Out[1]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_baser
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN	0.0	...	7	1180	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	...	7	2170	4
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	...	6	770	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	...	7	1050	9
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	...	8	1680	

5 rows × 21 columns

2.2 Dropping Uncessary Columns

- Looking at the column descriptions, I have determined that several are unecessary.
- Of course, if I find that I need any of them, I can easily get them back into the dataset.

In [2]:

```
df = df.drop(df[['id','date', 'view', 'lat', 'long', 'yr_renovated']], axis=1)
df.head()
```

executed in 10ms, finished 06:21:58 2021-12-22

Out[2]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	grade	sqft_above	sqft_basement	yr_built	zipcode
0	221900.0	3	1.00	1180	5650	1.0	NaN	3	7	1180	0.0	1955	98178
1	538000.0	3	2.25	2570	7242	2.0	0.0	3	7	2170	400.0	1951	98125
2	180000.0	2	1.00	770	10000	1.0	0.0	3	6	770	0.0	1933	98028
3	604000.0	4	3.00	1960	5000	1.0	0.0	5	7	1050	910.0	1965	98136
4	510000.0	3	2.00	1680	8080	1.0	0.0	3	8	1680	0.0	1987	98074

2.3 Checking Data Types

In [3]:
```
df.info()
```
executed in 11ms, finished 06:21:58 2021-12-22

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          21597 non-null  float64
 1   bedrooms       21597 non-null  int64  
 2   bathrooms      21597 non-null  float64
 3   sqft_living    21597 non-null  int64  
 4   sqft_lot       21597 non-null  int64  
 5   floors         21597 non-null  float64
 6   waterfront     19221 non-null  float64
 7   condition      21597 non-null  int64  
 8   grade          21597 non-null  int64  
 9   sqft_above     21597 non-null  int64  
 10  sqft_basement  21597 non-null  object 
 11  yr_built       21597 non-null  int64  
 12  zipcode        21597 non-null  int64  
 13  sqft_living15  21597 non-null  int64  
 14  sqft_lot15     21597 non-null  int64  
dtypes: float64(4), int64(10), object(1)
memory usage: 2.5+ MB
```

Analysis:

- waterfront seems to have Null Values
- sqft_basement is an object and will need to be converted to float or integer.

2.3.1 Fixing Waterfront

In [4]:
```
waterfront_cleaned = df['waterfront'].fillna(0)
df['waterfront'] = waterfront_cleaned
df.isna().sum()
```
executed in 5ms, finished 06:21:58 2021-12-22

Out[4]:
```
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
zipcode          0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

2.3.2 Fixing sqft_basement

- slicing out all records with a '?' and calculating the correct value using other known fields.

In [5]:
```python
unknown_basements = df[df['sqft_basement'] == '?']
known_basements = df[df['sqft_basement'] != '?']

sqft_basement = unknown_basements.apply(lambda x: x['sqft_living'] - x['sqft_above'], axis=1)
unknown_basements['sqft_basement'] = sqft_basement

cleaned_df = known_basements.append(unknown_basements)

#changing to float so that decminals are in the same format
cleaned_df['sqft_basement'] = cleaned_df['sqft_basement'].astype(float)
cleaned_df['sqft_above'] = cleaned_df['sqft_above'].astype(float)

cleaned_df['sqft_basement'].value_counts().head()

```

executed in 22ms, finished 06:21:58 2021-12-22

```
<ipython-input-5-177a306978bb>:5: 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/indexin
g.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy)
  unknown_basements['sqft_basement'] = sqft_basement
```

Out[5]:
```
0.0      13110
600.0      221
700.0      218
500.0      214
800.0      206
Name: sqft_basement, dtype: int64
```

2.3.3 Changing Zip Code to Category

In [6]:
```python
cleaned_df['zipcode'] = df['zipcode'].astype(str)
cleaned_df['zipcode'].value_counts().head()
```

executed in 21ms, finished 06:21:58 2021-12-22

Out[6]:
```
98103    602
98038    589
98115    583
98052    574
98117    553
Name: zipcode, dtype: int64
```

2.3.4 Dropping Bedroom Outliers

In [7]:
```python
cleaned_df['bedrooms'].value_counts().head(20)
```

executed in 4ms, finished 06:21:58 2021-12-22

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

In [8]:
```
cleaned_df.sort_values('bedrooms', ascending=False).head(10)
```
executed in 12ms, finished 06:21:58 2021-12-22

Out[8]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	grade	sqft_above	sqft_basement	yr_built	zipc
15856	640000.0	33	1.75	1620	6000	1.0	0.0	5	7	1040.0	580.0	1947	98
8748	520000.0	11	3.00	3000	4960	2.0	0.0	3	7	2400.0	600.0	1918	98
15147	650000.0	10	2.00	3610	11914	2.0	0.0	4	7	3010.0	600.0	1958	98
19239	660000.0	10	3.00	2920	3745	2.0	0.0	4	7	1860.0	1060.0	1913	98
13301	1150000.0	10	5.25	4590	10920	1.0	0.0	3	9	2500.0	2090.0	2008	98
8537	450000.0	9	7.50	4050	6504	2.0	0.0	3	7	4050.0	0.0	1996	98
18428	934000.0	9	3.00	2820	4480	2.0	0.0	3	7	1880.0	940.0	1918	98
4231	700000.0	9	3.00	3680	4400	2.0	0.0	3	7	2830.0	850.0	1908	98
16830	1400000.0	9	4.00	4620	5508	2.5	0.0	3	11	3870.0	750.0	1915	98
6073	1280000.0	9	4.50	3650	5000	2.0	0.0	3	8	2530.0	1120.0	1915	98

I doubt that any house has 33 bedrooms. It is likely a typo. I will also drop the records with 10 and 11 bedrooms to remove some of the extreme outliers. This will only be 5 records and have very minimal impact on my data.

In [9]:
```
#dropping outliers
cleaned_df = cleaned_df.sort_values('bedrooms', ascending=False).reset_index()
cleaned_df = cleaned_df.drop([0,1,2,3,4,5])
cleaned_df.head(5)
```
executed in 15ms, finished 06:21:58 2021-12-22

Out[9]:

	index	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	grade	sqft_above	sqft_basement	yr_built
6	18428	934000.0	9	3.0	2820	4480	2.0	0.0	3	7	1880.0	940.0	1918
7	4231	700000.0	9	3.0	3680	4400	2.0	0.0	3	7	2830.0	850.0	1908
8	16830	1400000.0	9	4.0	4620	5508	2.5	0.0	3	11	3870.0	750.0	1915
9	6073	1280000.0	9	4.5	3650	5000	2.0	0.0	3	8	2530.0	1120.0	1915
10	4092	599999.0	9	4.5	3830	6988	2.5	0.0	3	7	2450.0	1380.0	1938

In [10]:
```
#dropping index
cleaned_df = cleaned_df.drop(['index'], axis=1)
```
executed in 4ms, finished 06:21:58 2021-12-22

In [11]:
```
1 cleaned_df.info()
```
executed in 7ms, finished 06:21:58 2021-12-22

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21591 entries, 6 to 21596
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          21591 non-null  float64
 1   bedrooms       21591 non-null  int64  
 2   bathrooms      21591 non-null  float64
 3   sqft_living    21591 non-null  int64  
 4   sqft_lot       21591 non-null  int64  
 5   floors         21591 non-null  float64
 6   waterfront     21591 non-null  float64
 7   condition      21591 non-null  int64  
 8   grade          21591 non-null  int64  
 9   sqft_above     21591 non-null  float64
 10  sqft_basement  21591 non-null  float64
 11  yr_built       21591 non-null  int64  
 12  zipcode        21591 non-null  object 
 13  sqft_living15  21591 non-null  int64  
 14  sqft_lot15     21591 non-null  int64  
dtypes: float64(6), int64(8), object(1)
memory usage: 2.6+ MB
```

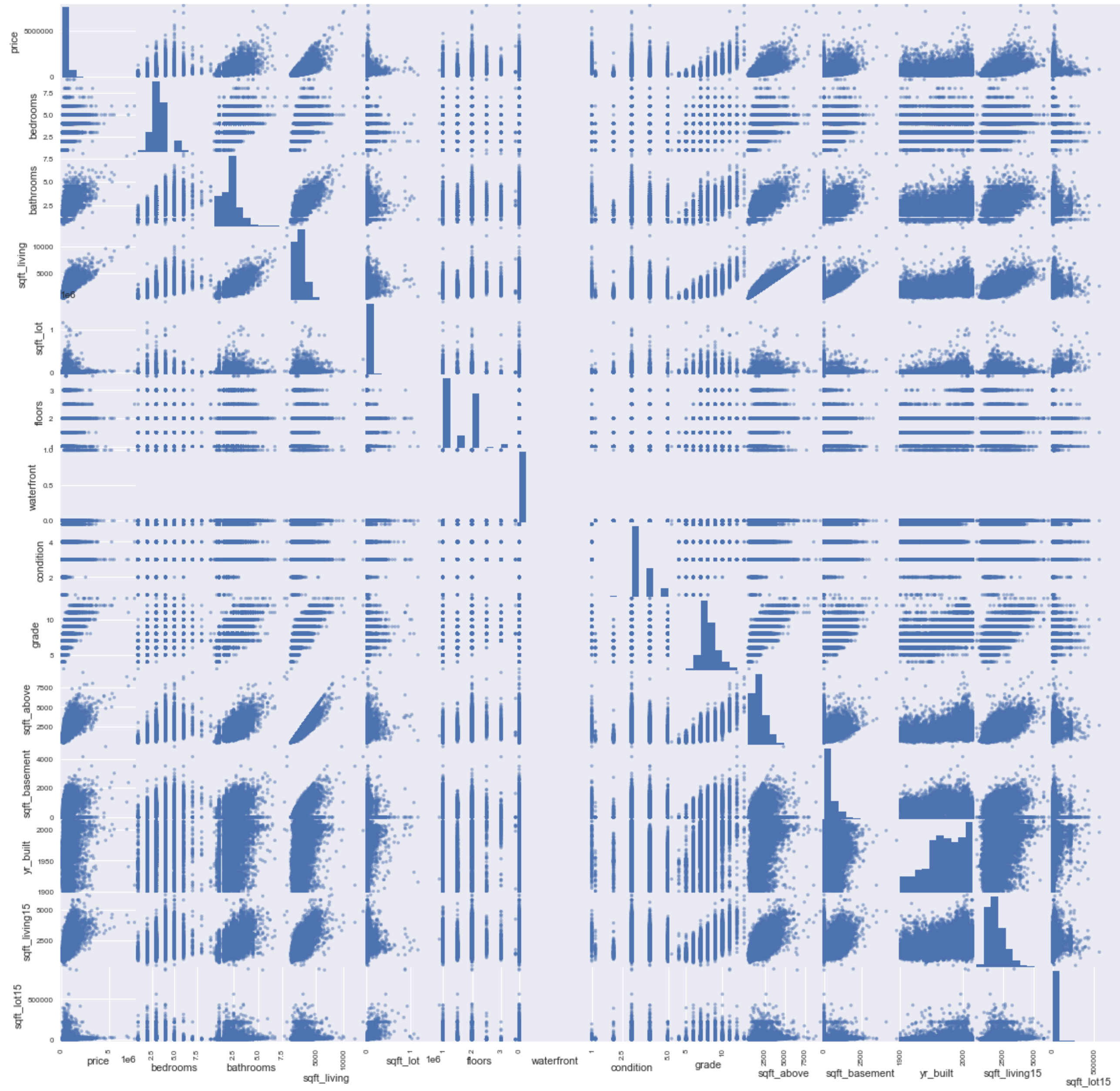
- The only object that is left is zipcode, which is correct. I don't want this to be considered a continuous variable. I will one-hot-encode it when I get to that step.

3 Exploratory Data Analysis & Feature Selection

3.1 Exploring Data with Scatter Plot

In [12]:
```
#using scatter plot to look for linear relationships
pd.plotting.scatter_matrix(cleaned_df, figsize = [20,20]);
plt.show()
```

executed in 14.8s, finished 06:22:13 2021-12-22



Analysis of Scatter Plots

<u>The following variables seem to have linear relationships:</u>

- **Price:** linear relationship with: bedrooms, sqft_above, & sqft_basement.
  - price also seems to have a linear relationship with categorical variable 'grade'.
- **Bedrooms** linear relationship with: bathrooms, sqft_living, sqft_above, & sqft_basement
- **Sqft_Living and Sqft_Above have the closest linear relationship**
  - They are very similar data points. I may need to eliminate one to prevent multicolinearity.

<u>The Following Variables seem to be categorical:</u>

- **Floors**
- **Waterfront**
- **Condition**
- **Zip code** (not shown in this scatter plot because it is an object)

<u>Ordinal Variables:</u>

- **Bedrooms**
- **Bathrooms**
- **Grade** (I am going to treat grade as a continuous variable as it has very linear relationships with many features. Including price.)

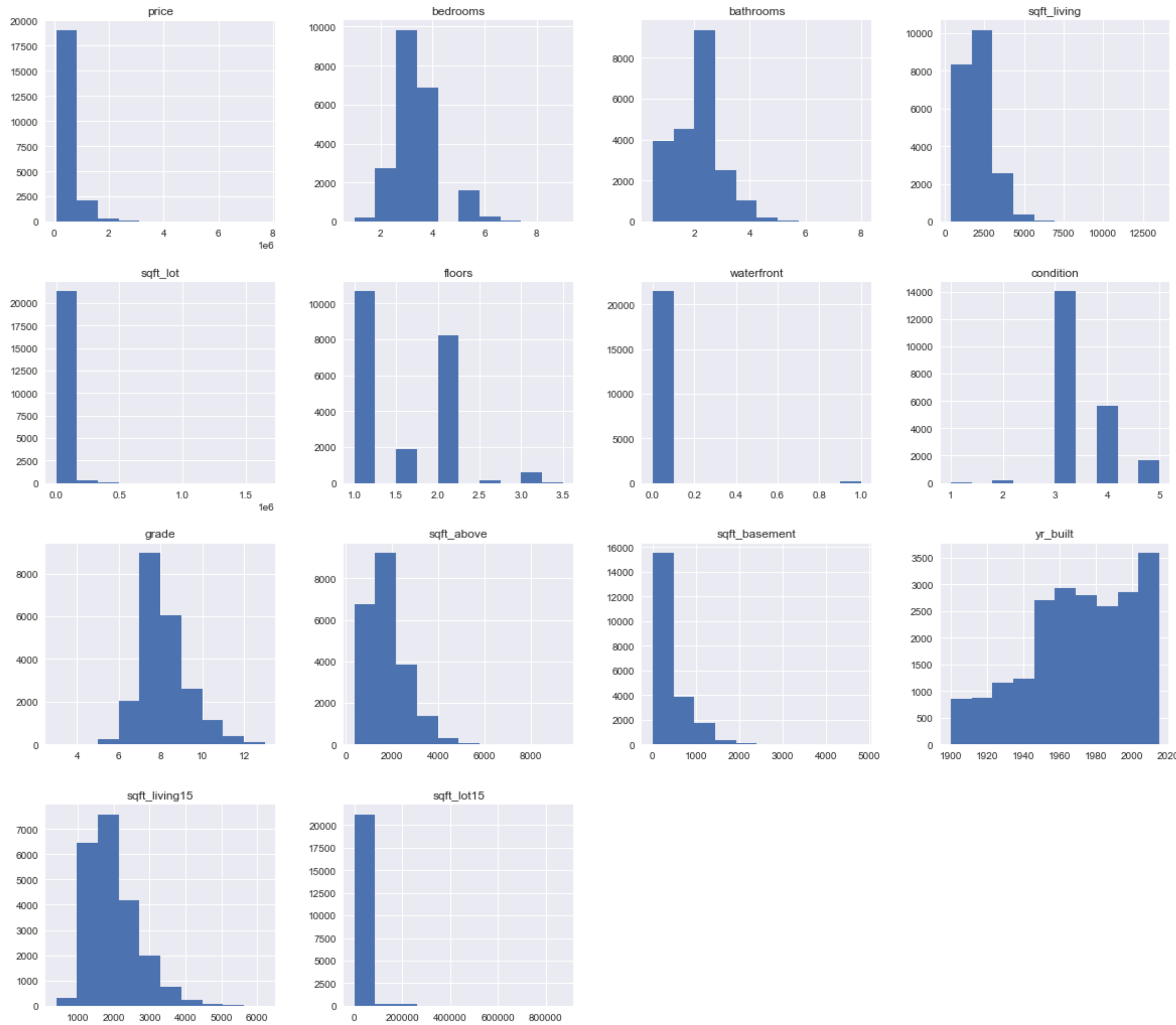
3.2 Histograms Comparing Price and Features

- each feature is compared against price. (Price is y axis)

In [13]:

```
cleaned_df.hist(figsize = (20,18));
```

executed in 821ms, finished 06:22:14 2021-12-22

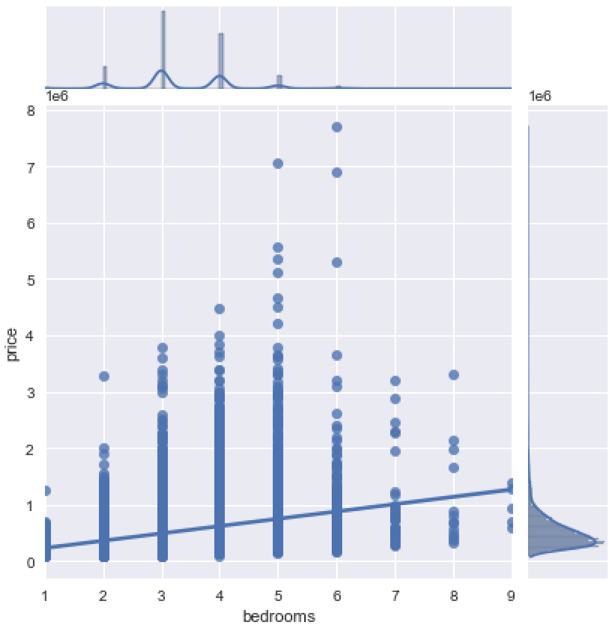


Analysis of Histograms:

- Price is very skewed. I will need to fix this as it is my target variable.
- Lot size (sqft_lot and sqft_lot15) seem to be consistant across the board. Especially when compared to the variation in house size (sqft_living, sqft_above, sqft_living15).
- The vast majority of homes have a condition of 3, which is the middle value of "average". This column isn't giving me much information so it should probably be dropped.

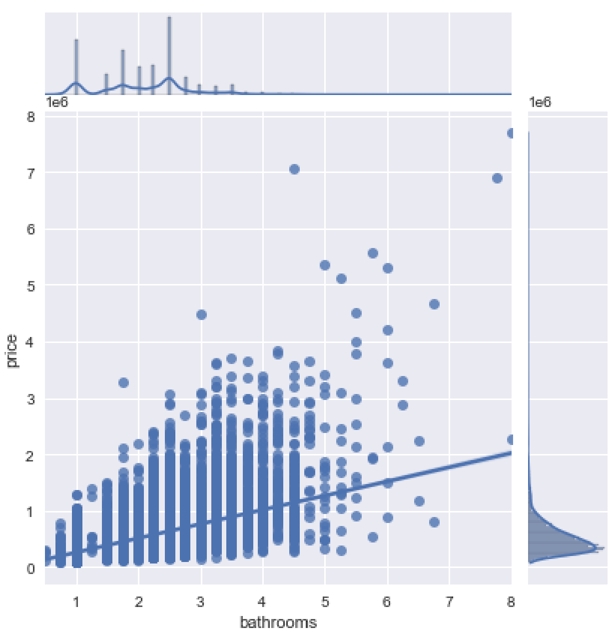
3.3 Analysis of key variables against the Target (price) using jointplots

In [14]:
```
sns.jointplot('bedrooms','price', data=cleaned_df, kind='reg');
```
executed in 3.95s, finished 06:22:17 2021-12-22



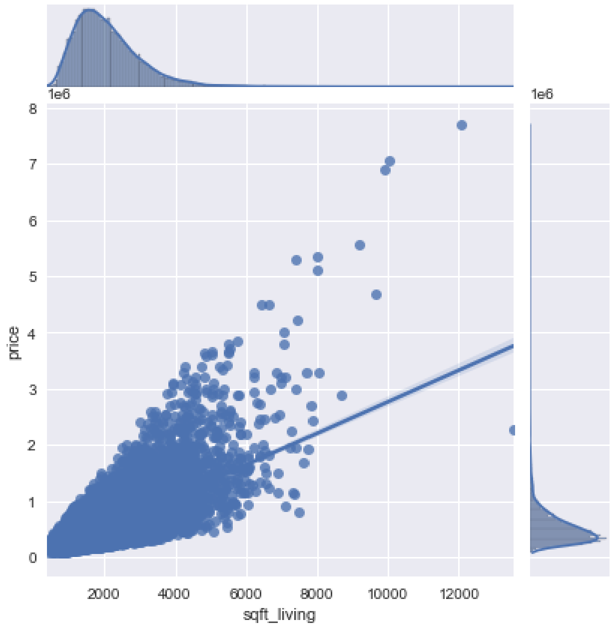
Bedrooms: While this is an ordinal variable, it behaves more like a categorical than a continuous variable. 7 bedrooms isn't necessarily better than 2 bedrooms, it all depends on the house itself. That said, there is a slight linear relationship. I may try both methods to see which way this feature will best fit my model.

In [15]:
```
sns.jointplot('bathrooms','price', data=cleaned_df, kind='reg');
```
executed in 3.82s, finished 06:22:21 2021-12-22



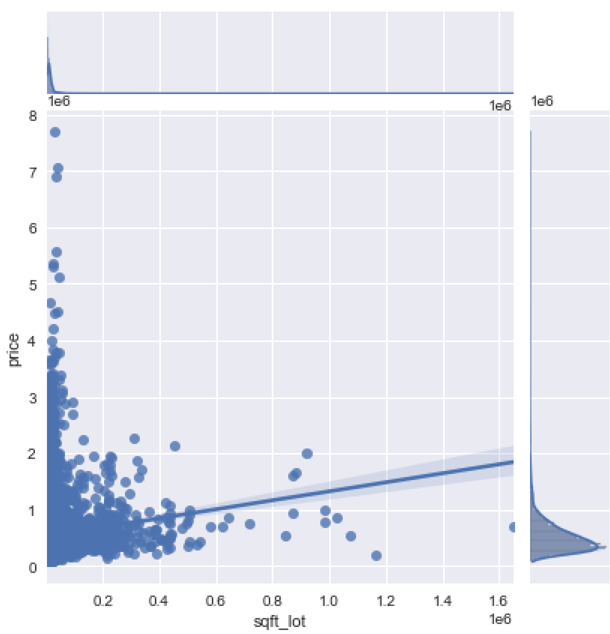
Bathroooms: Unlike bedrooms, bathrooms behave more like a continous variable than a categorical one, so I will treat it as such. This also means that it will be a better predictor than bedrooms. If I have to chose between one or the other, I should choose bathrooms. This is likely because the scatterplot showed them to have a very linear relationship with each other.

In [16]:
```
sns.jointplot('sqft_living','price', data=cleaned_df, kind='reg');
```
executed in 4.02s, finished 06:22:25 2021-12-22



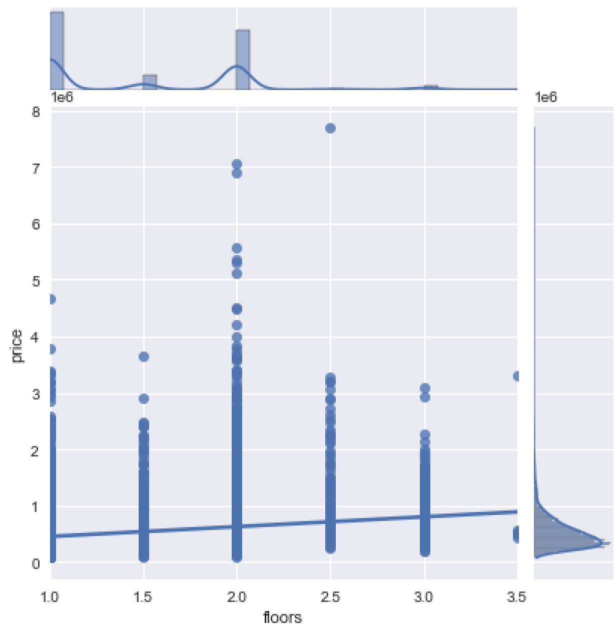
Sqft_living: This seems to be a very linear relationship. This makes sense as the bigger the house it, the more likely that it costs more.

In [17]:
```
sns.jointplot('sqft_lot','price', data=cleaned_df, kind='reg');
```
executed in 6.77s, finished 06:22:32 2021-12-22



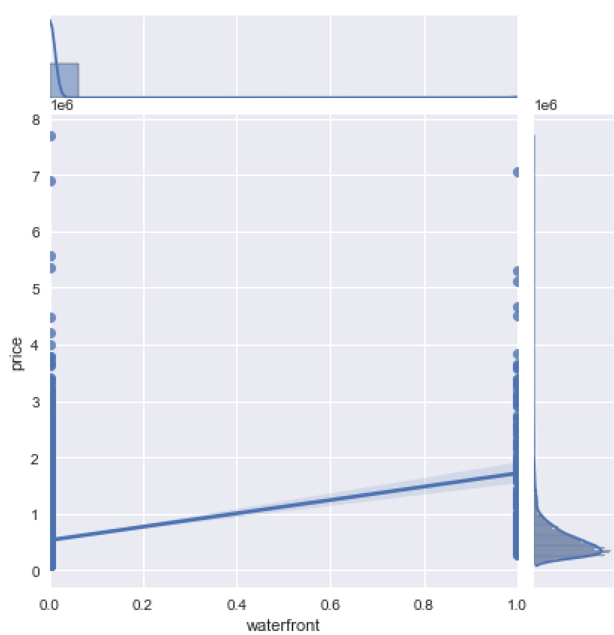
sqft_lot: Lot size has a slight correlation with the price of a house, but there are a lot of outliers, especially with little to no lot size. It will be hard to use this as a predictor.

In [18]:
```
sns.jointplot('floors','price', data=cleaned_df, kind='reg');

```
executed in 3.76s, finished 06:22:36 2021-12-22



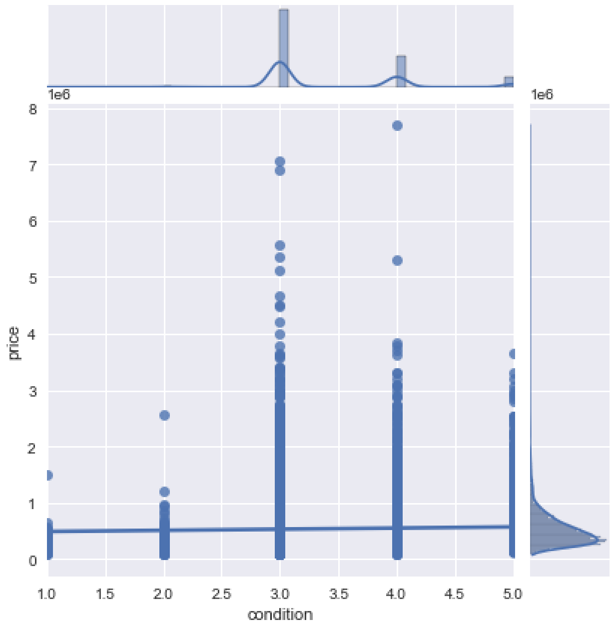
Floors: There is almost no correlation with price, so this feature can likely be excluded from my model.

In [19]:
```
sns.jointplot('waterfront','price', data=cleaned_df, kind='reg');
```
executed in 3.79s, finished 06:22:40 2021-12-22



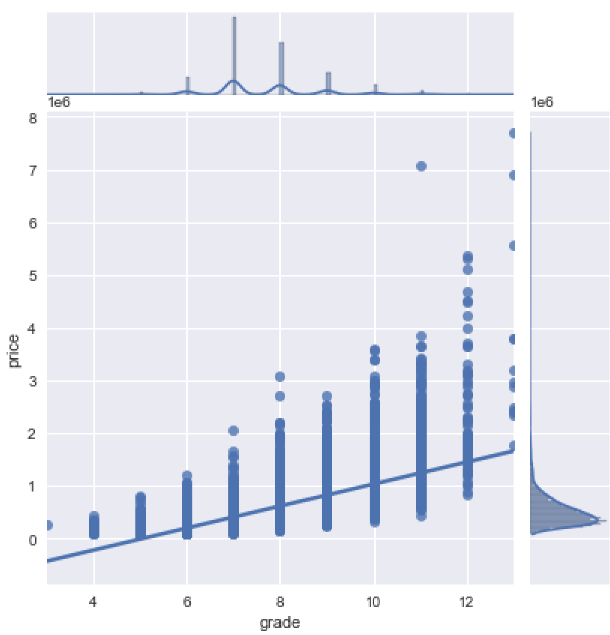
Waterfront: There appears to be a slight linear relationship between price and being on the waterfront. There are very few houses in this dataset that are on the waterfront, but this feature may still affect the price and should be left in unless it causes issues.

In [20]:
```
sns.jointplot('condition','price', data=cleaned_df, kind='reg');
```
executed in 3.71s, finished 06:22:43 2021-12-22



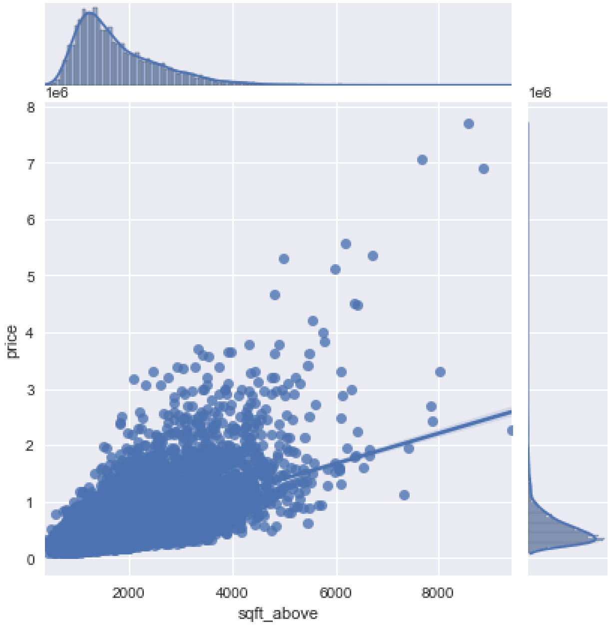
<u>Condition</u>: As noted in the analysis of the scatter plot, condition doesn't have much of an affect on price. I can drop this feature.

In [21]:
```
sns.jointplot('grade','price', data=cleaned_df, kind='reg');
```
executed in 3.86s, finished 06:22:47 2021-12-22



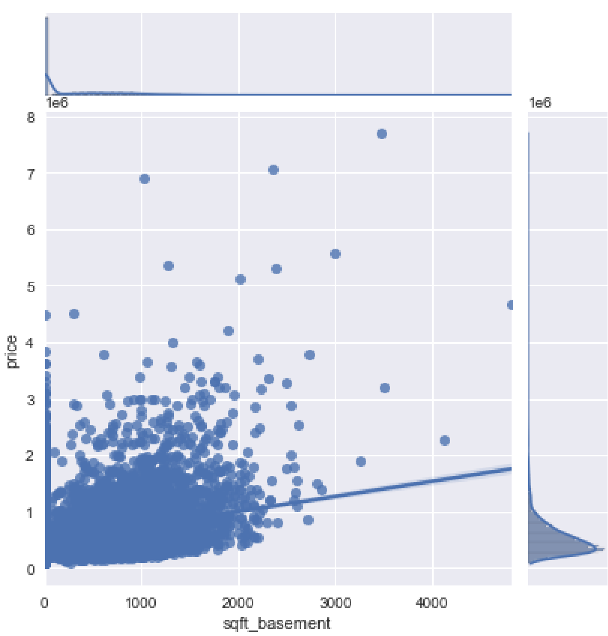
<u>Grade</u>: Grade has a slightly linear relationship with a little noise. I should keep it as a continous variable. The relationship looks like it could be improved with some cleaning, removing outliers, etc.

In [22]:
```
sns.jointplot('sqft_above','price', data=cleaned_df, kind='reg');
```
executed in 3.96s, finished 06:22:51 2021-12-22



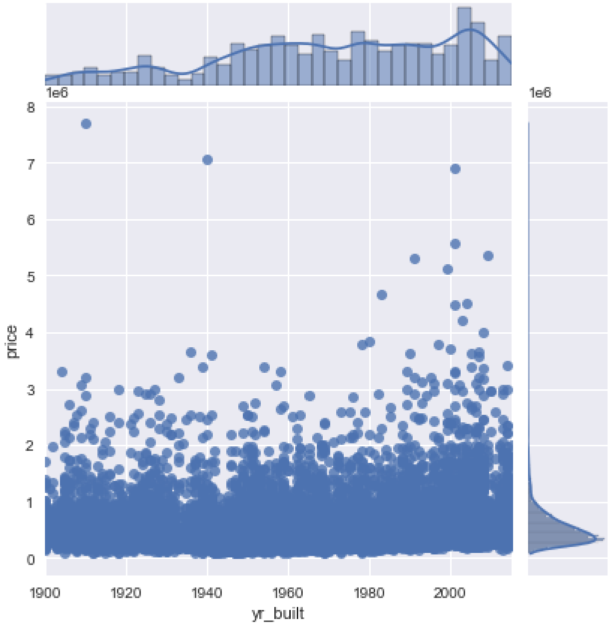
Sqft_Above: This feature is almost exactly the same thing as sqft_living. (Scatter plot supports this, as well as their glossary descriptions being very similar). I will almost definitely need to remove one of the two of these variables and use the other due to multicolinearity. I will determine which to use when I check for that.

In [23]:
```
sns.jointplot('sqft_basement','price', data=cleaned_df, kind='reg');
```
executed in 3.80s, finished 06:22:55 2021-12-22



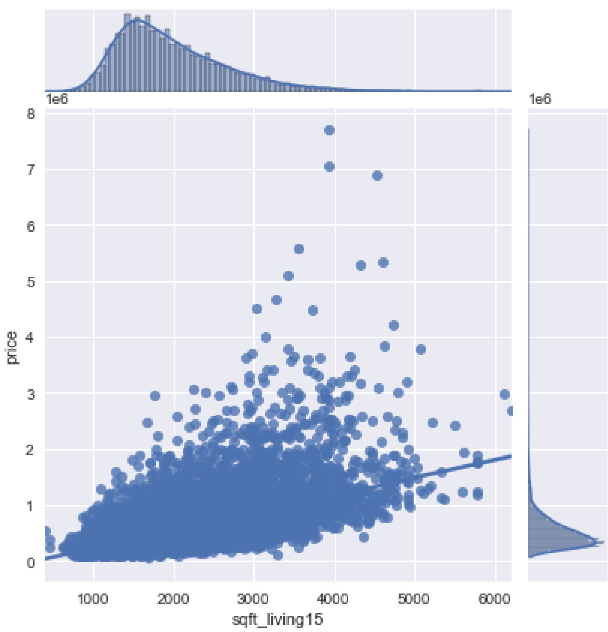
Basement: Basement size has a slight linear relationship with price. But I also see that there are many outliers that have very little size that are skewing the results. If I can deal with them. it could improve the prediction power of this feature.

In [24]:
```
sns.jointplot('yr_built','price', data=cleaned_df, kind='reg');
```
executed in 3.83s, finished 06:22:59 2021-12-22



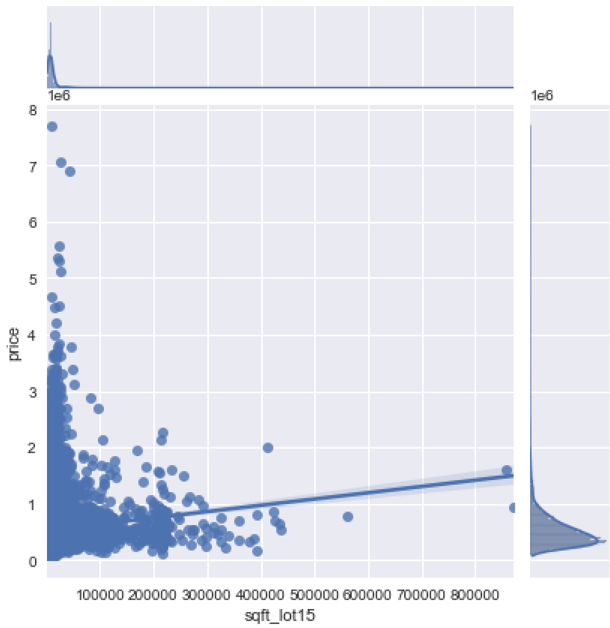
Year Built: The year each house was built seems to have no relationship with Price and can likely be excluded from my model.

In [25]:
```
sns.jointplot('sqft_living15','price', data=cleaned_df, kind='reg');
```
executed in 3.90s, finished 06:23:03 2021-12-22



sqft_living15: The size of houses nearby does have a linear relationship with price. Looks fairly close to sqft_living and sqft_above so there's a strong chance of multicolinearity here as well.

In [26]:
```
sns.jointplot('sqft_lot15','price', data=cleaned_df, kind='reg');
```
executed in 5.68s, finished 06:23:08 2021-12-22



sqft_lot15: Looks identical to sqft_lot, which I likely won't end up using. This will likely be dropped as well. If I use either, it would be just that one as they are very likely to be multicolinear.

3.4 Feature Selection

- sqft_living15 and sqft_lot15 give the same information as sqft_living and sqft_lot, so I will drop them to avoid multicolinearity.
- yr_built and condition didn't have a linear relationships with price, so I am not including them in my model.

Dropping Columns

In [27]:
```
cleaned_df = cleaned_df.drop(['sqft_living15', 'yr_built', 'sqft_lot15', 'condition'], axis=1)
cleaned_df.head(1)
```
executed in 8ms, finished 06:23:08 2021-12-22

Out[27]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	grade	sqft_above	sqft_basement	zipcode
6	934000.0	9	3.0	2820	4480	2.0	0.0	7	1880.0	940.0	98105

4 Initial Modeling

In [28]:
```
list(cleaned_df.columns)
```
executed in 3ms, finished 06:23:08 2021-12-22

Out[28]:
```
['price',
 'bedrooms',
 'bathrooms',
 'sqft_living',
 'sqft_lot',
 'floors',
 'waterfront',
 'grade',
 'sqft_above',
 'sqft_basement',
 'zipcode']
```

In [29]:
```
# Defining the problem
outcome = 'price'
x_cols = list(cleaned_df.columns)
x_cols.remove(outcome)

train, test = train_test_split(cleaned_df, random_state=23)

print(len(train), len(test))
```
executed in 5ms, finished 06:23:08 2021-12-22

```
16193 5398
```

In [30]:
```
# Fitting the actual model
predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model = ols(formula=formula, data=train).fit()
model.summary()
```
executed in 298ms, finished 06:23:09 2021-12-22

Out[30]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.788
Model:	OLS	**Adj. R-squared:**	0.787
Method:	Least Squares	**F-statistic:**	776.1
Date:	Wed, 22 Dec 2021	**Prob (F-statistic):**	0.00
Time:	06:23:09	**Log-Likelihood:**	-2.1793e+05
No. Observations:	16193	**AIC:**	4.360e+05
Df Residuals:	16115	**BIC:**	4.366e+05
Df Model:	77		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]

4.1 Analysis of First Model:

- The R-squared is 78.8%, which is really good. My data cleaning and feature selection must have paid off.
- That said, I'm sure further refinement will be necessary.
- There are high p-values on several of the zipcodes.
- Skew is higher than I would like it to be.
- Kurtosis is also very high.

5 Refining My Model

5.1 Cleaning & Encoding

In [31]:
```
encoded_df= cleaned_df
```
executed in 2ms, finished 06:23:09 2021-12-22

In [32]:
```
#cleaning columns so that I can One-Hot Encode them
subs = [(' ', '_'),('.',''),("'",""),
        ('+','plus'), ('-','_')]

def col_formatting(col):
    for old, new in subs:
        col = col.replace(old,new)
    return col
```
executed in 2ms, finished 06:23:09 2021-12-22

In [33]:
```
encoded_df.columns = [col_formatting(col) for col in encoded_df.columns]
```
executed in 2ms, finished 06:23:09 2021-12-22

In [34]:
```
list(encoded_df.columns)
```
executed in 2ms, finished 06:23:09 2021-12-22

Out[34]:
```
['price',
 'bedrooms',
 'bathrooms',
 'sqft_living',
 'sqft_lot',
 'floors',
 'waterfront',
 'grade',
 'sqft_above',
 'sqft_basement',
 'zipcode']
```

In [35]:
```
#one-hot encoding
feats = ['floors', 'waterfront','zipcode'] #treating bedrooms as a continous variable helps the mode
encoded_df[feats] = encoded_df[feats].astype(str)
encoded_df = pd.get_dummies(encoded_df, drop_first=True)
```
executed in 39ms, finished 06:23:09 2021-12-22

In [36]:
```
encoded_df.head(1)
```
executed in 11ms, finished 06:23:09 2021-12-22

Out[36]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	grade	sqft_above	sqft_basement	floors_1.5	floors_2.0	...	zipcode_98146	zipcod
6	934000.0	9	3.0	2820	4480	7	1880.0	940.0	0	1	...	0	

1 rows × 83 columns

In [37]:
```
encoded_df.columns = [col_formatting(col) for col in encoded_df.columns]
```
executed in 2ms, finished 06:23:09 2021-12-22

5.1.1 Normalizing Data

In [38]:
```
def norm_feat(series):
    return (series - series.mean())/series.std()
```
executed in 2ms, finished 06:23:09 2021-12-22

In [39]:

```
df_norm = norm_feat(encoded_df)
df_norm.head()
```

executed in 39ms, finished 06:23:09 2021-12-22

Out[39]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	grade	sqft_above	sqft_basement	floors_15	floors_20	...	zipcode_98146
6	1.071718	6.26552	1.152054	0.806230	-0.256447	-0.560824	0.110668	1.465326	-0.311518	1.273992	...	-0.116269
7	0.434798	6.26552	1.152054	1.743178	-0.258379	-0.560824	1.258497	1.261941	-0.311518	1.273992	...	-0.116269
8	2.340115	6.26552	2.454359	2.767284	-0.231627	2.848409	2.515068	1.035958	-0.311518	-0.784898	...	-0.116269
9	2.013489	6.26552	3.105511	1.710494	-0.243892	0.291485	0.896025	1.872096	-0.311518	1.273992	...	-0.116269
10	0.162607	6.26552	3.105511	1.906599	-0.195894	-0.560824	0.799365	2.459653	-0.311518	-0.784898	...	-0.116269

5 rows × 83 columns

5.1.2 Checking Multicolinearity with VIF scores

In [40]:
```
x_cols = list(df_norm.columns)
x_cols.remove(outcome)
X = df_norm[x_cols]
vif = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
list(zip(x_cols, vif))
```
executed in 45.3s, finished 06:23:54 2021-12-22

```
/Users/jonathanholt/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/statsmodels/stats/outlier
s_influence.py:193: RuntimeWarning: divide by zero encountered in double_scalars
  vif = 1. / (1. - r_squared_i)
```

Out[40]:
```
[('bedrooms', 1.762884316470203),
 ('bathrooms', 3.1354539328330007),
 ('sqft_living', inf),
 ('sqft_lot', 1.2180483564649132),
 ('grade', 3.2803657450221477),
 ('sqft_above', inf),
 ('sqft_basement', inf),
 ('floors_15', 1.2702746143368044),
 ('floors_20', 2.429082325023472),
 ('floors_25', 1.1033907900598452),
 ('floors_30', 1.3520102111574994),
 ('floors_35', 1.007850756007571),
 ('waterfront_10', 1.0615885226056663),
 ('zipcode_98002', 1.5411528708335134),
 ('zipcode_98003', 1.756711400703128),
 ('zipcode_98004', 1.8879939883720351),
 ('zipcode_98005', 1.4733139631279435),
 ('zipcode_98006', 2.390836236858363),
 ('zipcode_98007', 1.3874215323839338),
 ('zipcode_98008', 1.775740846271686),
 ('zipcode_98010', 1.2777745114206935),
 ('zipcode_98011', 1.5291035510375375),
 ('zipcode_98014', 1.365439019203317),
 ('zipcode_98019', 1.5221944675217256),
 ('zipcode_98022', 1.656617683589679),
 ('zipcode_98023', 2.3341104209902643),
 ('zipcode_98024', 1.2382682579358484),
 ('zipcode_98027', 2.123905085326323),
 ('zipcode_98028', 1.7637962963901423),
 ('zipcode_98029', 1.8947134545796582),
 ('zipcode_98030', 1.6898662405658949),
 ('zipcode_98031', 1.7354026588113676),
 ('zipcode_98032', 1.3418989568181634),
 ('zipcode_98033', 2.1670357583160063),
 ('zipcode_98034', 2.4555616642348204),
 ('zipcode_98038', 2.5799109882438693),
 ('zipcode_98039', 1.153977089163956),
 ('zipcode_98040', 1.806299177310307),
 ('zipcode_98042', 2.4536023091120307),
 ('zipcode_98045', 1.6035209193673245),
 ('zipcode_98052', 2.5428184452519886),
 ('zipcode_98053', 2.117196678505801),
 ('zipcode_98055', 1.722002136832341),
 ('zipcode_98056', 2.086190470251917),
 ('zipcode_98058', 2.2159946065480565),
 ('zipcode_98059', 2.260267509308709),
 ('zipcode_98065', 1.8505229070139937),
 ('zipcode_98070', 1.3697626007211792),
 ('zipcode_98072', 1.7443085167597563),
 ('zipcode_98074', 2.217074749131239),
 ('zipcode_98075', 2.0134826319385226),
 ('zipcode_98077', 1.564886832374753),
 ('zipcode_98092', 1.94888213139931),
 ('zipcode_98102', 1.2933507139482716),
 ('zipcode_98103', 2.719989079323814),
 ('zipcode_98105', 1.6379975797466884),
 ('zipcode_98106', 1.9104842984676202),
 ('zipcode_98107', 1.7714787537535734),
 ('zipcode_98108', 1.5110478942316459),
 ('zipcode_98109', 1.3133031552438066),
 ('zipcode_98112', 1.7723122801161033),
 ('zipcode_98115', 2.5892387733080127),
 ('zipcode_98116', 1.917910118105171),
 ('zipcode_98117', 2.5172699303151385),
 ('zipcode_98118', 2.3690849478261544),
```

```
 ('zipcode_98119', 1.5289574239353305),
 ('zipcode_98122', 1.8182833317978475),
 ('zipcode_98125', 2.1130928097839696),
 ('zipcode_98126', 1.967683689673378),
 ('zipcode_98133', 2.3322428262064765),
 ('zipcode_98136', 1.7266605277265163),
 ('zipcode_98144', 1.9549362719369046),
 ('zipcode_98146', 1.7815583252192286),
 ('zipcode_98148', 1.1558032700281131),
 ('zipcode_98155', 2.1983048850687457),
 ('zipcode_98166', 1.694214941658717),
 ('zipcode_98168', 1.7373804625805962),
 ('zipcode_98177', 1.7034559299903589),
 ('zipcode_98178', 1.7167342133235788),
 ('zipcode_98188', 1.3703820570078786),
 ('zipcode_98198', 1.758952358532574),
 ('zipcode_98199', 1.8874016438612018)]
```

5.1.3 Analysis:

- sqft_living, sqft_above, and sqft_basement all have infinite VIF scores.
- sqft_living or sqft_above will definitely need to be dropped. I will check for colinear pairs to see if one is causing more problems than the other.
- The other VIF scores are acceptable.

5.1.4 Finding Colinear Pairs

In [41]:

```
cc_df = df_norm.corr().abs().stack().reset_index().sort_values(0, ascending=False)

cc_df['pairs'] = list(zip(cc_df.level_0, cc_df.level_1))

cc_df.set_index(['pairs'], inplace = True)

cc_df.drop(columns=['level_1', 'level_0'], inplace = True)

# cc for correlation coefficient
cc_df.columns = ['cc']

cc_df.drop_duplicates(inplace=True)

cc_df[(cc_df.cc>.70) & (cc_df.cc<1)]
```

executed in 163ms, finished 06:23:54 2021-12-22

Out[41]:

pairs	cc
(sqft_above, sqft_living)	0.876477
(sqft_living, grade)	0.763120
(sqft_above, grade)	0.756391
(bathrooms, sqft_living)	0.755941
(sqft_living, price)	0.702029

Analysis:

- sqft_living is causing more problems than sqft_above. My data analysis thus far has shown them to be incredibly similar so I will be fine keeping just one of them.
- Grade is highly correlated to sqft_above, but I will initially leave it in my model and see what it looks like. I don't want to remove too many features if I can keep them.

In [42]:

```
df_norm = df_norm.drop(['sqft_living'], axis=1)
df_norm.head(1)
```

executed in 11ms, finished 06:23:54 2021-12-22

Out[42]:

	price	bedrooms	bathrooms	sqft_lot	grade	sqft_above	sqft_basement	floors_15	floors_20	floors_25	...	zipcode_98146
6	1.071718	6.26552	1.152054	-0.256447	-0.560824	0.110668	1.465326	-0.311518	1.273992	-0.086675	...	-0.116269

1 rows × 82 columns

In [43]:

```
cc_df = df_norm.corr().abs().stack().reset_index().sort_values(0, ascending=False)

cc_df['pairs'] = list(zip(cc_df.level_0, cc_df.level_1))

cc_df.set_index(['pairs'], inplace = True)

cc_df.drop(columns=['level_1', 'level_0'], inplace = True)

# cc for correlation coefficient
cc_df.columns = ['cc']

cc_df.drop_duplicates(inplace=True)

cc_df[(cc_df.cc>.70) & (cc_df.cc<1)]
```

executed in 158ms, finished 06:23:55 2021-12-22

Out[43]:

pairs	cc
(grade, sqft_above)	0.756391

That took care of most of the correlated pairs. Now let's check the vif scores again and see if it resolved the infinite correlations.

In [44]:
```
x_cols = list(df_norm.columns)
x_cols.remove(outcome)
X = df_norm[x_cols]
vif = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
list(zip(x_cols, vif))
```
executed in 44.0s, finished 06:24:38 2021-12-22

Out[44]:
```
[('bedrooms', 1.7681469134950174),
 ('bathrooms', 3.136291868974238),
 ('sqft_lot', 1.2180499054745253),
 ('grade', 3.28053119354771),
 ('sqft_above', 4.4797511573497015),
 ('sqft_basement', 1.9155359304596866),
 ('floors_15', 1.270277097802257),
 ('floors_20', 2.429535579553982),
 ('floors_25', 1.1039585166717556),
 ('floors_30', 1.352020550898107),
 ('floors_35', 1.007879305695572),
 ('waterfront_10', 1.0615894361516676),
 ('zipcode_98002', 1.5411536516400872),
 ('zipcode_98003', 1.7567114745952686),
 ('zipcode_98004', 1.887994996695563),
 ('zipcode_98005', 1.4733143253648087),
 ('zipcode_98006', 2.3908372609955353),
 ('zipcode_98007', 1.387421811892771),
 ('zipcode_98008', 1.775742402426752),
 ('zipcode_98010', 1.2777752468620431),
 ('zipcode_98011', 1.5291037780357966),
 ('zipcode_98014', 1.365439728227529),
 ('zipcode_98019', 1.5221948549989457),
 ('zipcode_98022', 1.6566188371131427),
 ('zipcode_98023', 2.3341107187647023),
 ('zipcode_98024', 1.2382698301740895),
 ('zipcode_98027', 2.123911497010405),
 ('zipcode_98028', 1.7637978155616285),
 ('zipcode_98029', 1.8947151756171914),
 ('zipcode_98030', 1.6898664028999562),
 ('zipcode_98031', 1.7354041105021736),
 ('zipcode_98032', 1.34189905328653),
 ('zipcode_98033', 2.167040019902266),
 ('zipcode_98034', 2.455583077401331),
 ('zipcode_98038', 2.5799131530296693),
 ('zipcode_98039', 1.1539781160515958),
 ('zipcode_98040', 1.8063003929647425),
 ('zipcode_98042', 2.4536194976852967),
 ('zipcode_98045', 1.6035225842542653),
 ('zipcode_98052', 2.5428194141503275),
 ('zipcode_98053', 2.117207732225706),
 ('zipcode_98055', 1.7220022111581863),
 ('zipcode_98056', 2.086190687561142),
 ('zipcode_98058', 2.21600063617012),
 ('zipcode_98059', 2.260276099006659),
 ('zipcode_98065', 1.8505240528525273),
 ('zipcode_98070', 1.3697643598303582),
 ('zipcode_98072', 1.7443089963801373),
 ('zipcode_98074', 2.217082059289114),
 ('zipcode_98075', 2.013487774411539),
 ('zipcode_98077', 1.5648875882690803),
 ('zipcode_98092', 1.948883526776969),
 ('zipcode_98102', 1.3061308531451485),
 ('zipcode_98103', 2.7199949182648635),
 ('zipcode_98105', 1.6410052983127326),
 ('zipcode_98106', 1.9104844558578689),
 ('zipcode_98107', 1.7714809534375389),
 ('zipcode_98108', 1.5110483805803245),
 ('zipcode_98109', 1.313306408188517),
 ('zipcode_98112', 1.7723238403664696),
 ('zipcode_98115', 2.5892390381079338),
 ('zipcode_98116', 1.917910825952164),
 ('zipcode_98117', 2.51727458855732),
 ('zipcode_98118', 2.3690878056312434),
 ('zipcode_98119', 1.528959135731437),
 ('zipcode_98122', 1.818296553650024),
 ('zipcode_98125', 2.11309408004312),
 ('zipcode_98126', 1.9676862977698655),
 ('zipcode_98133', 2.3322438051852536),
```

```
('zipcode_98136', 1.7266611749979022),
('zipcode_98144', 1.9549393357270866),
('zipcode_98146', 1.7815592573401628),
('zipcode_98148', 1.155803321939537),
('zipcode_98155', 2.198305869179016),
('zipcode_98166', 1.694215788012742),
('zipcode_98168', 1.7373810945305277),
('zipcode_98177', 1.7034589249407943),
('zipcode_98178', 1.7167357126367289),
('zipcode_98188', 1.3703831214137947),
('zipcode_98198', 1.7589543978635906),
('zipcode_98199', 1.8874050065276484)]
```

I am happy with these VIF scores. My target variable (price) still has a decent correlation with sqft_above, grade and bathrooms, but they are within the limits that I have (VIF < 7), and they are predictors that I want to keep if at all possible.

5.1.5 Running the model again

In [45]:

```python
# Defining the problem
outcome = 'price'
x_cols = list(df_norm.columns)
x_cols.remove(outcome)
train, test = train_test_split(df_norm, random_state=23)
print(len(train), len(test))
```

executed in 7ms, finished 06:24:39 2021-12-22

```
16193 5398
```

In [46]:

```python
# Fitting the actual model
predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model = ols(formula=formula, data=train).fit()
model.summary()
```

executed in 289ms, finished 06:24:39 2021-12-22

Out[46]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.789
Model:	OLS	**Adj. R-squared:**	0.788
Method:	Least Squares	**F-statistic:**	745.4
Date:	Wed, 22 Dec 2021	**Prob (F-statistic):**	0.00
Time:	06:24:39	**Log-Likelihood:**	-10365.
No. Observations:	16193	**AIC:**	2.089e+04
Df Residuals:	16111	**BIC:**	2.152e+04
Df Model:	81		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]

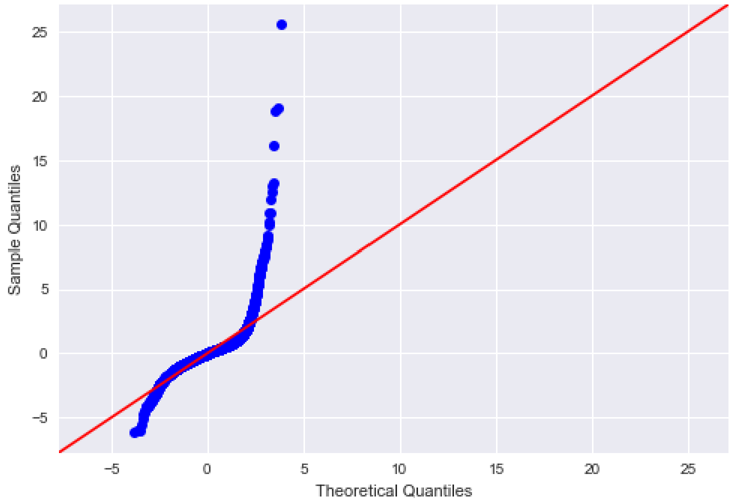
Model Analysis:

- R-squared is 78.9%. I would ideally like to see it at 80% or above, but this is incredibly close.
- Prob(F-statistic) is 0, which means that there is good model integrity.
- Kurtosis is still really high. I will need to refine it so that it is closer to normal (3)
- Model is skewed. I need to address this as well.

6 Further Model Refinement

6.1 Q-Q Plot (Checking Normality Assumption)

In [47]:
```
fig = sm.graphics.qqplot(model.resid, dist=stats.norm, line='45', fit=True)
```
executed in 75ms, finished 06:24:39 2021-12-22

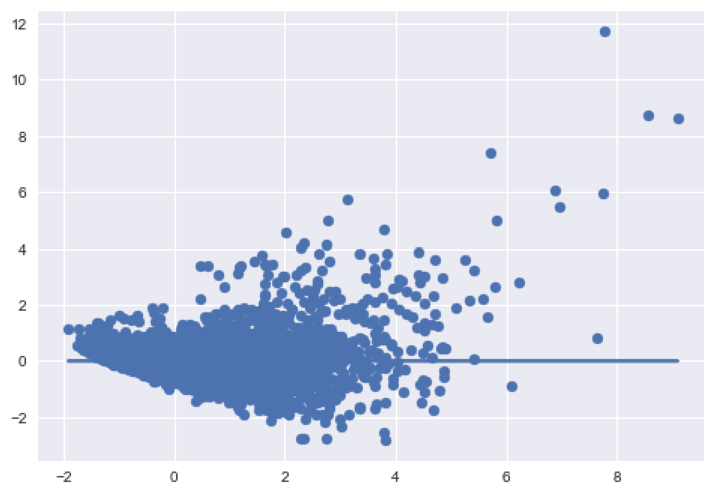


There are more errors as price increases. This needs to be refined so that the model is accurate. This model cannot be used without further refinement.

6.2 Scatter Plot (Checking For Homoscedasticity)

In [48]:
```
plt.scatter(model.predict(train[x_cols]), model.resid)
plt.plot(model.predict(train[x_cols]), [0 for i in range(len(train))])
```
executed in 184ms, finished 06:24:39 2021-12-22

Out[48]: [<matplotlib.lines.Line2D at 0x7f822f690d60>]



This plot is funnel-shaped. The errors show some correlation with price, so I need to refine this.

6.3 Dealing with Outliers

I will now switch back to encoded_df so that I can see what the acutal price is, instead of the normalized price. I will drop the same columns that I dropped from df_norm so that both datasets contain the same base data.

In [49]:
```python
encoded_df = encoded_df.drop(['sqft_living'], axis=1)
encoded_df.head(1)
```
executed in 10ms, finished 06:24:39 2021-12-22

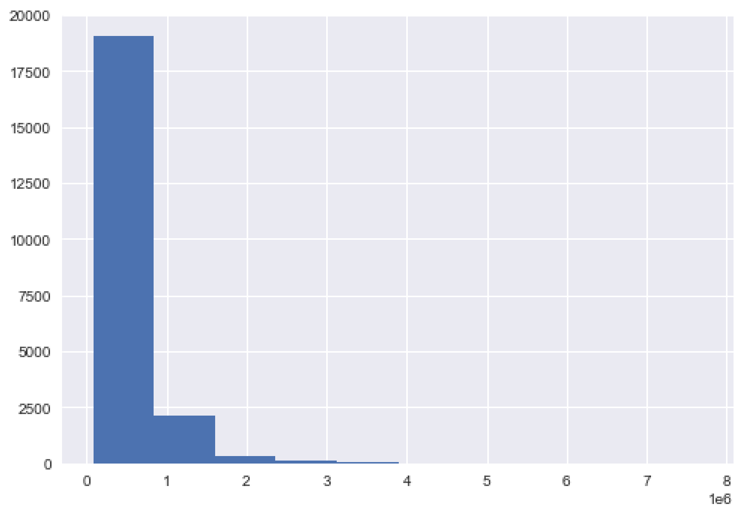
Out[49]:

	price	bedrooms	bathrooms	sqft_lot	grade	sqft_above	sqft_basement	floors_15	floors_20	floors_25	...	zipcode_98146	zipcode_
6	934000.0	9	3.0	4480	7	1880.0	940.0	0	1	0	...	0	

1 rows × 82 columns

In [50]:
```python
encoded_df.price.hist()
```
executed in 80ms, finished 06:24:39 2021-12-22

Out[50]: <AxesSubplot:>



This is the same histogram that I checked before. Price is very skewed. I need to drop some of the outliers to see if I improve this distribution and make it more normalized.

6.3.1 Checking price percentiles to find the outliers

In [51]:
```python
for i in range(80,100):
    q = i/100
    print("{} percentile: {}".format(q, encoded_df.price.quantile(q=q)))
```
executed in 18ms, finished 06:24:39 2021-12-22

```
0.8 percentile: 700500.0
0.81 percentile: 718000.0
0.82 percentile: 730000.8
0.83 percentile: 749950.0
0.84 percentile: 760003.0
0.85 percentile: 779665.0
0.86 percentile: 799000.0
0.87 percentile: 815000.0
0.88 percentile: 836600.0000000003
0.89 percentile: 859953.9999999999
0.9 percentile: 887000.0
0.91 percentile: 919986.0000000001
0.92 percentile: 950000.0
0.93 percentile: 997665.0000000007
0.94 percentile: 1060000.0
0.95 percentile: 1160000.0
0.96 percentile: 1260000.0
0.97 percentile: 1390000.0
0.98 percentile: 1600000.0
0.99 percentile: 1970000.0
```

In [52]:
```
for i in range(0,20):
    q = i/100
    print("{} percentile: {}".format(q, encoded_df.price.quantile(q=q)))
```
executed in 16ms, finished 06:24:39 2021-12-22

```
0.0 percentile: 78000.0
0.01 percentile: 154000.0
0.02 percentile: 175000.0
0.03 percentile: 192000.0
0.04 percentile: 202500.0
0.05 percentile: 210000.0
0.06 percentile: 219950.0
0.07 percentile: 226500.0
0.08 percentile: 234000.0
0.09 percentile: 240000.0
0.1 percentile: 245000.0
0.11 percentile: 250000.0
0.12 percentile: 255000.0
0.13 percentile: 260000.0
0.14 percentile: 266000.0
0.15 percentile: 270000.0
0.16 percentile: 275021.2
0.17 percentile: 280000.0
0.18 percentile: 286000.0
0.19 percentile: 291000.0
```

In [53]:
```
df = encoded_df

orig_tot = len(df)
df = df[df.price < 1500000]# Subsetting to remove extreme outliers
df = df[df.price > 125000]
print('Percent removed:', (orig_tot -len(df))/orig_tot)
train, test = train_test_split(df, random_state=23)

# Refit model with subset features
predictors = '+'.join(x_cols)
formula = outcome + "~" + predictors
final_model = ols(formula=formula, data=train).fit()
final_model.summary()

```
executed in 407ms, finished 06:24:40 2021-12-22

```
Percent removed: 0.028669352971145385
```

Model Analysis: Removing some of the price outliers on each end improved the model. (R-sqaured is now at 81%)

- This only removed 2.9% of the data, which is acceptable.

In [54]:
```python
#making sure the changes are saved as final_df
final_df = df
final_df.head(1)
```
executed in 9ms, finished 06:24:40 2021-12-22

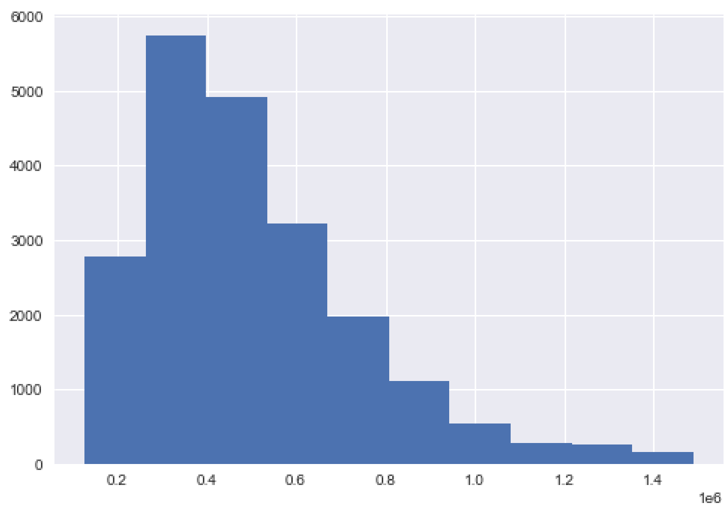
Out[54]:

	price	bedrooms	bathrooms	sqft_lot	grade	sqft_above	sqft_basement	floors_15	floors_20	floors_25	...	zipcode_98146	zipcode_
6	934000.0	9	3.0	4480	7	1880.0	940.0	0	1	0	...	0	

1 rows × 82 columns

In [55]:
```python
final_df.price.hist()
```
executed in 74ms, finished 06:24:40 2021-12-22

Out[55]: <AxesSubplot:>



While this is not a normal distribution, it now looks more normalized than before. This is a good sign. I will see if this has done enough to help the model or if I need to refine further.

6.4 Normalizing final_df and running model on it

In [56]:
```python
final_df_norm = norm_feat(final_df)
final_df_norm.head(1)
```
executed in 30ms, finished 06:24:40 2021-12-22

Out[56]:

	price	bedrooms	bathrooms	sqft_lot	grade	sqft_above	sqft_basement	floors_15	floors_20	floors_25	...	zipcode_98146	zi
6	1.756998	6.379373	1.259494	-0.256734	-0.550632	0.176477	1.562765	-0.313701	1.289979	-0.079584	...	-0.11569	

1 rows × 82 columns

In [57]:
```python
# Defining the problem
outcome = 'price'
x_cols = list(final_df_norm.columns)
x_cols.remove(outcome)
train, test = train_test_split(final_df_norm, random_state=23)
print(len(train), len(test))
```
executed in 16ms, finished 06:24:40 2021-12-22

```
15729 5243
```

In [58]:

```python
# Fitting the actual model
predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model = ols(formula=formula, data=train).fit()
model.summary()
```

executed in 276ms, finished 06:24:40 2021-12-22

Out[58]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.813
Model:	OLS	**Adj. R-squared:**	0.812
Method:	Least Squares	**F-statistic:**	837.9
Date:	Wed, 22 Dec 2021	**Prob (F-statistic):**	0.00
Time:	06:24:40	**Log-Likelihood:**	-9154.7
No. Observations:	15729	**AIC:**	1.847e+04
Df Residuals:	15647	**BIC:**	1.910e+04
Df Model:	81		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]

Analysis:

- R-squared is above my target of 80%.
- Prob(F-statistic) is still 0, so the model continues to have integrity on the test group.
- Skew is significantly better at 0.971. It is still not 0, but this is only a 1/4 of the skew that the previous model was showing (3.961).
- Kurtosis still exsists (7.45), but is also greatly improved over the previous model (63.337).

6.5 Checking VIF and Assumptions Again

- I will check VIF scores and assumptions one more time. If I don't see any red flags, I will keep this as my final model.

In [59]:
```
x_cols = list(final_df_norm.columns)
x_cols.remove(outcome)

X = final_df_norm[x_cols]
vif = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
list(zip(x_cols, vif))
```
executed in 42.9s, finished 06:25:23 2021-12-22

Out[59]:
```
[('bedrooms', 1.7805132210413512),
 ('bathrooms', 2.90136926384536),
 ('sqft_lot', 1.2097440760169322),
 ('grade', 2.9362013916818963),
 ('sqft_above', 4.301815724624278),
 ('sqft_basement', 1.9071459760680587),
 ('floors_15', 1.281411310660798),
 ('floors_20', 2.487438131039225),
 ('floors_25', 1.0763573159929398),
 ('floors_30', 1.3639429100387757),
 ('floors_35', 1.0080770846340052),
 ('waterfront_10', 1.0665059472000828),
 ('zipcode_98002', 1.5367511066230723),
 ('zipcode_98003', 1.7691852321122818),
 ('zipcode_98004', 1.572515634881473),
 ('zipcode_98005', 1.4639703539655757),
 ('zipcode_98006', 2.307216204431571),
 ('zipcode_98007', 1.3938247627934675),
 ('zipcode_98008', 1.7618537995514445),
 ('zipcode_98010', 1.2831097810279246),
 ('zipcode_98011', 1.5380375619748208),
 ('zipcode_98014', 1.3579632145038398),
 ('zipcode_98019', 1.5286961745345073),
 ('zipcode_98022', 1.6672551082027045),
 ('zipcode_98023', 2.34007155951752),
 ('zipcode_98024', 1.2180257047769965),
 ('zipcode_98027', 2.1206598332602273),
 ('zipcode_98028', 1.7735756630994133),
 ('zipcode_98029', 1.9012100479461242),
 ('zipcode_98030', 1.6983792850890813),
 ('zipcode_98031', 1.7473505231742046),
 ('zipcode_98032', 1.3368024275612096),
 ('zipcode_98033', 2.1066072622662957),
 ('zipcode_98034', 2.449251261291999),
 ('zipcode_98038', 2.6004108421918786),
 ('zipcode_98039', 1.042182255799099),
 ('zipcode_98040', 1.633542874001812),
 ('zipcode_98042', 2.469034836346855),
 ('zipcode_98045', 1.6112777143904227),
 ('zipcode_98052', 2.5635734426297168),
 ('zipcode_98053', 2.1179191393014456),
 ('zipcode_98055', 1.7286564505914035),
 ('zipcode_98056', 2.09565374197718),
 ('zipcode_98058', 2.2274547074604887),
 ('zipcode_98059', 2.2728351884577584),
 ('zipcode_98065', 1.8640260327152551),
 ('zipcode_98070', 1.4030029588467097),
 ('zipcode_98072', 1.7549170201303026),
 ('zipcode_98074', 2.2177890016256003),
 ('zipcode_98075', 2.0174517798044844),
 ('zipcode_98077', 1.5680050421232048),
 ('zipcode_98092', 1.9622440101457577),
 ('zipcode_98102', 1.2896737429214675),
 ('zipcode_98103', 2.7448099753004795),
 ('zipcode_98105', 1.5982487286563303),
 ('zipcode_98106', 1.9143432243354874),
 ('zipcode_98107', 1.7807867366602639),
 ('zipcode_98108', 1.517801385145602),
 ('zipcode_98109', 1.2923239920645773),
 ('zipcode_98112', 1.6322989327538027),
 ('zipcode_98115', 2.6015050373054764),
 ('zipcode_98116', 1.9234005594634),
 ('zipcode_98117', 2.5420635280167088),
 ('zipcode_98118', 2.3857540175865495),
 ('zipcode_98119', 1.5046864573765653),
 ('zipcode_98122', 1.807582314136095),
 ('zipcode_98125', 2.12454494466598),
 ('zipcode_98126', 1.9802860299977285),
```

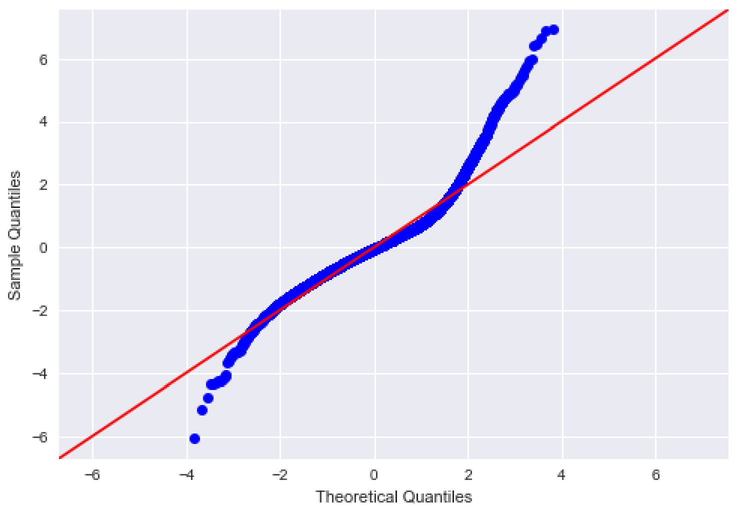
```
('zipcode_98133', 2.3555012751132627),
('zipcode_98136', 1.7318825512553138),
('zipcode_98144', 1.937386461465961),
('zipcode_98146', 1.765532972158311),
('zipcode_98148', 1.1528785433892719),
('zipcode_98155', 2.2047312365574334),
('zipcode_98166', 1.6864993656993774),
('zipcode_98168', 1.70855688239269),
('zipcode_98177', 1.6804130473202337),
('zipcode_98178', 1.72002800180001),
('zipcode_98188', 1.3711635510037357),
('zipcode_98198', 1.765686079283194),
('zipcode_98199', 1.862827512504223)]
```

Analysis:

- Sqft_Above has a VIF of 4.3. While I would like to see a lower score, it is less than my threshold of 7.
- All other VIFs are less than 3.

6.6 Q-Q Plot (Normality Check)

In [60]:
```
fig = sm.graphics.qqplot(model.resid, dist=stats.norm, line='45', fit=True)
```
executed in 76ms, finished 06:25:23 2021-12-22



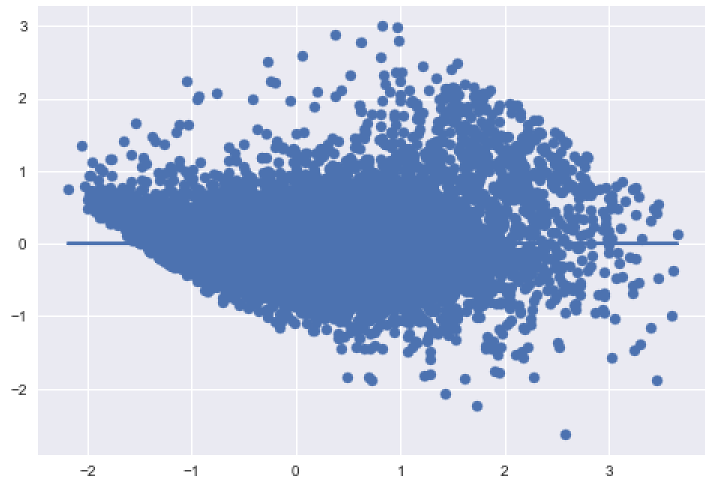
Analysis:

- Normality is definitely improved, but it still isn't ideal.
- However, for the scope of this business problem, my model will be accurate in determining the key factors in home value, so I am good to continue on and make my recommendations.
- **Note:** This model is still going to be unreliable on the extreme tails, so I would not trust it to predict an exact price for each feature. Fortunatley, that is beyond the scale of what I have been asked to do, so I am good to proceed.

6.7 Scatter Plot (Homoscedasticity Check)

In [61]:
```
plt.scatter(model.predict(train[x_cols]), model.resid)
plt.plot(model.predict(train[x_cols]), [0 for i in range(len(train))])
```
executed in 185ms, finished 06:25:23 2021-12-22

Out[61]: [<matplotlib.lines.Line2D at 0x7f81d80bdd00>]



Analysis:

- The scatter plot has significantly improved, even though it is definitely not perfect.
- The extreme left tail is stll not showing a random distribution of errors.
- Like with the Q-Q Test, these results are not perfect, but they show the model is reliable enough for me to make reccomendations within the scope that was outlined in the business problem.

7 Final Analysis and Recommendations

In [62]:

```
# Bringing up my model to analyze
results = ols(formula=formula, data=train).fit()
model.summary()

```

executed in 274ms, finished 06:25:23 2021-12-22

Out[62]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.813
Model:	OLS	**Adj. R-squared:**	0.812
Method:	Least Squares	**F-statistic:**	837.9
Date:	Wed, 22 Dec 2021	**Prob (F-statistic):**	0.00
Time:	06:25:23	**Log-Likelihood:**	-9154.7
No. Observations:	15729	**AIC:**	1.847e+04
Df Residuals:	15647	**BIC:**	1.910e+04
Df Model:	81		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.0014	0.003	0.407	0.684	0.008	0.005

7.1 Ranking Features using Coefficients

In [63]:
```
#Checking coefficients
coeff = results.params
ranked_features = coeff.sort_values(ascending=False)
ranked_features
```
executed in 4ms, finished 06:25:23 2021-12-22

Out[63]:
```
sqft_above       0.465615
grade            0.245212
zipcode_98004    0.239604
zipcode_98103    0.238497
zipcode_98115    0.230675
zipcode_98117    0.221982
zipcode_98112    0.207422
zipcode_98033    0.205835
zipcode_98040    0.203155
zipcode_98199    0.196098
sqft_basement    0.188620
zipcode_98105    0.187643
zipcode_98006    0.182300
zipcode_98116    0.170142
zipcode_98052    0.169215
zipcode_98119    0.168100
zipcode_98122    0.157014
zipcode_98107    0.156044
zipcode_98144    0.141874
zipcode_98109    0.134831
zipcode_98136    0.127628
zipcode_98008    0.124526
zipcode_98034    0.124321
zipcode_98102    0.123045
zipcode_98053    0.122202
zipcode_98005    0.121180
zipcode_98075    0.120752
zipcode_98125    0.120102
zipcode_98074    0.119938
zipcode_98118    0.112282
zipcode_98126    0.111259
zipcode_98029    0.110944
zipcode_98177    0.110417
zipcode_98027    0.107638
zipcode_98133    0.101887
waterfront_10    0.091645
zipcode_98007    0.088104
zipcode_98155    0.087036
zipcode_98039    0.084515
zipcode_98072    0.080688
zipcode_98106    0.068374
zipcode_98056    0.065656
zipcode_98077    0.064862
zipcode_98059    0.064692
zipcode_98146    0.064208
zipcode_98028    0.062496
zipcode_98065    0.061429
zipcode_98011    0.055968
zipcode_98166    0.055933
zipcode_98108    0.049572
sqft_lot         0.048605
zipcode_98045    0.040482
zipcode_98024    0.038580
zipcode_98019    0.035351
zipcode_98070    0.034703
zipcode_98178    0.031751
zipcode_98014    0.028617
bathrooms        0.025779
zipcode_98168    0.025565
zipcode_98038    0.024596
zipcode_98010    0.023792
zipcode_98058    0.022732
zipcode_98055    0.022235
zipcode_98198    0.017318
floors_15        0.013571
zipcode_98148    0.013455
zipcode_98188    0.011924
zipcode_98022    0.010400
zipcode_98002    0.008422
zipcode_98042    0.006281
```

```
zipcode_98031     0.006090
zipcode_98032     0.000992
zipcode_98030     0.000895
zipcode_98003    -0.000723
Intercept        -0.001408
floors_25        -0.002244
zipcode_98092    -0.009494
floors_35        -0.010117
zipcode_98023    -0.014752
bedrooms         -0.036919
floors_30        -0.069169
floors_20        -0.073321
dtype: float64
```

In [64]:

```
ranked_features = ranked_features.reset_index()
ranked_features = ranked_features.rename(columns={"index": "zipcode"})
ranked_features = ranked_features.reset_index()
ranked_features = ranked_features.rename(columns={"index": "rank"})
ranked_zip = ranked_features.drop(labels=[0,1,10,35,50,57,64,74,75,77,79,80,81,], axis=0)
ranked_zip = ranked_zip.drop('rank', axis=1)
ranked_zip = ranked_zip.reset_index()
ranked_zip = ranked_zip.drop('index', axis=1)
ranked_zip = ranked_zip.reset_index()
franked_zip = ranked_zip.rename(columns={"index": "rank"})
ranked_zip
```

executed in 13ms, finished 06:25:23 2021-12-22

Out[64]:

	index	zipcode	0
0	0	zipcode_98004	0.239604
1	1	zipcode_98103	0.238497
2	2	zipcode_98115	0.230675
3	3	zipcode_98117	0.221982
4	4	zipcode_98112	0.207422
5	5	zipcode_98033	0.205835
6	6	zipcode_98040	0.203155
7	7	zipcode_98199	0.196098
8	8	zipcode_98105	0.187643
9	9	zipcode_98006	0.182300
10	10	zipcode_98116	0.170142

Analysis:

- Sqft_above is the most impactful feature in regards to the price of a home in King County, with a coefficient of .4656 and the highest t statistic (64.77)
- grade is the second most impactful feature with coefficient of .2452, and t statistic near 42 (however, the majority of the data has a grade of 3 = 'average'.
- basement is #3 with coefficient of .1886 and t statistic of aproximately 40.

In terms of Location:

- being on the waterfront increases the value of your house, but not as much as actual features of the house.
- Zip Code plays a huge rule in value. Different Zip Codes have different values, but the highest one have coefficents and t statistics that would place them in betwen grade and sqft_basement in importance.
- the lowest value Zip Codes don't affect price very much (a couple even have a negative coefficient), but the majority make a statistically relevant difference.

Other Features:

- sqft_lot. coefficient = .048, t-statistic = 12.54
- bathrooms = coefficent = .025, t-statistic = 4.33
- bedrooms & floors have NEGATIVE coefficients

7.2 Questions to Answer - Revisited

7.2.1 Question 1: What features have the biggest effect on home price?

1. Size of House (excluding Basement).
2. Grade
3. Location (Zip Code)
4. Size of Basement
5. Waterfront Location
6. Size of Lot
7. Number of Bathrooms

7.2.2 Question 2: Are these features able to be changed/renovated, or are they fixed?

- Features that CAN be changed (even if difficult):
    1. Size of House
    2. Grade
    3. Basement Size -- I assume that this would be difficult, but not impossible.
    4. Additional Bathrooms

7.2.3 Question 3: What are the best features that Trailblazer Renovations should market to potential clients?

1. Home Size
2. Grade (?)
3. Basement Size
4. Bathrooms

7.2.4 Question 4: Who should Trailblazer target their marketing campaign toward?

- Customers in Zip Codes that had high coefficients in my model.

7.2.5 Question 5: Are there any features that Trailblazer recommend against?

- Extra Bedrooms
- Extra Floors

These features have a negative coefficient in regards to home price. If the client **needs** these features, then that is fine. But the scope of my analysis was to find the features that maximized home value and these two features do not meet that metric.

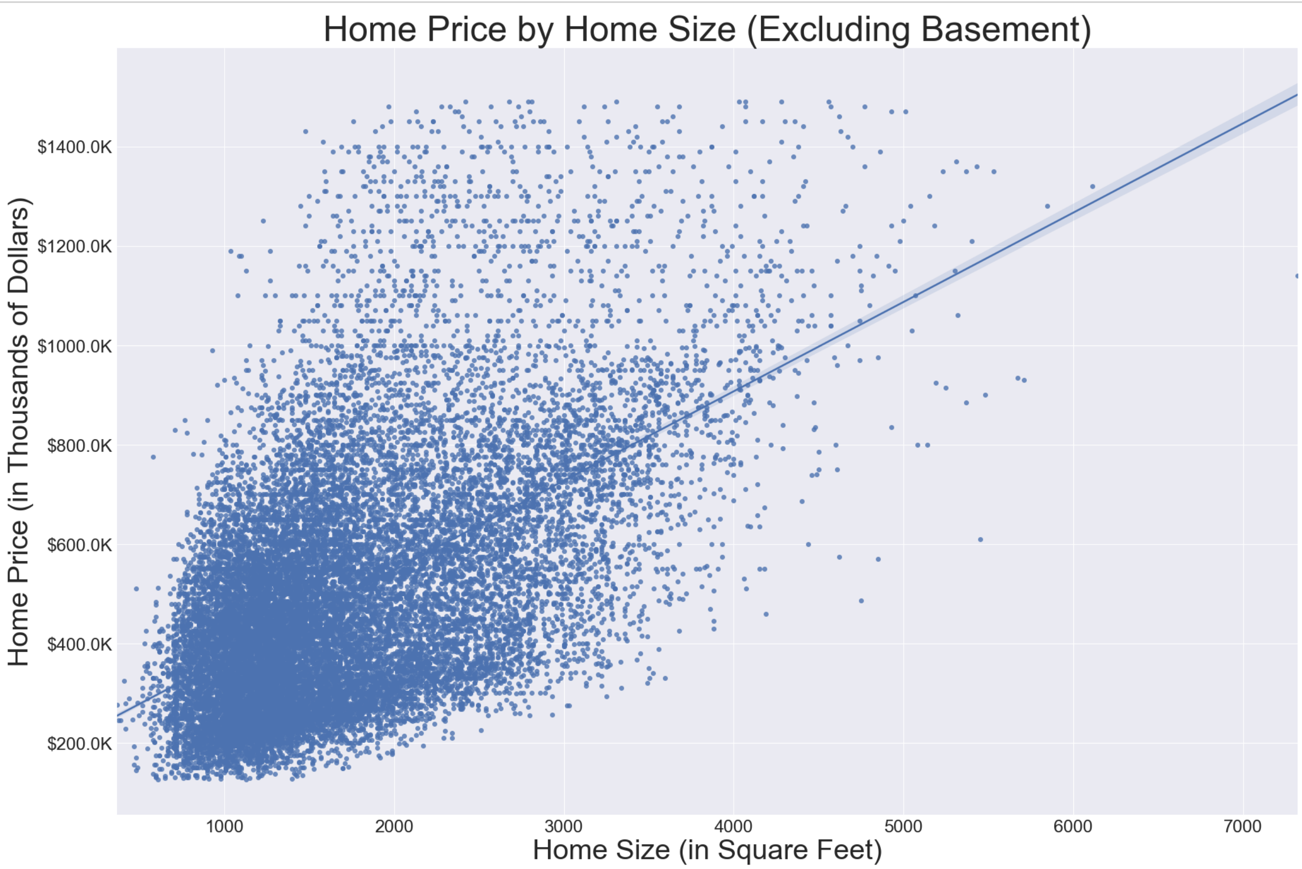
8 Recommendations

8.0.1 Visualization: Regplot of Home Price vs. Square Footage

In [65]:

```
fig, ax = plt.subplots(figsize=(30,20))

p =sns.regplot('sqft_above','price', data=final_df);

plt.title('Home Price by Home Size (Excluding Basement)', fontsize=50)# Set Title


plt.xlabel('Home Size (in Square Feet)', fontsize=40)# Set x-axis label
plt.xticks(fontsize=25)

plt.ylabel('Home Price (in Thousands of Dollars)', fontsize=40)# Set y-axis label
p.yaxis.set_major_formatter(display_thousands)
plt.yticks(fontsize=25)

plt.savefig('images/home_price_by_home_size')

plt.show();
```

executed in 3.74s, finished 06:25:27 2021-12-22



8.1 Recommendation #1

Trailblazer Renovations should focus on home additions. **Increasing the square footage of the home has the greatest effect on the value of the house.** This is advice that can be given to any homeowner whereever they live in King County.

- The increased square footage should be used for common areas as additional bedrooms have a negative corellation with price.
- Extra floors should also be avoided, as floors have a negative corellation with price.
- The exception to this is in the basement. Theoretically, if you are able to increase the space in your basement (perhaps converting mechanical closets/storage into common space), there is value to be added there.
  - I know that it is possible to even add a basement to a house that didn't initially have one, but I also know that it's signficantly more expensive to do that.

8.2 Recommendation #2

Trailblazer Renovations Inc. should use targeted marketing to homeowners in certain zip codes within King County. **There is a high coefficient on many zip codes, especially in the Seattle municipal area.**

Assuming that the cost for labor and materials is the same no matter what zip code the house is located in, the same work with the same materials brings greater value in certain zip codes.

Use the following list as a ranking of where to prioritize marketing efforts.

In [66]:
```
ranked_zip
```
executed in 8ms, finished 06:25:27 2021-12-22

Out[66]:

	index	zipcode	0
0	0	zipcode_98004	0.239604
1	1	zipcode_98103	0.238497
2	2	zipcode_98115	0.230675
3	3	zipcode_98117	0.221982
4	4	zipcode_98112	0.207422
5	5	zipcode_98033	0.205835
6	6	zipcode_98040	0.203155
7	7	zipcode_98199	0.196098
8	8	zipcode_98105	0.187643
9	9	zipcode_98006	0.182300
10	10	zipcode_98116	0.170142

8.2.1 Visualization: Regplot of Home Price vs. Grade

In [67]:

```
#plotting grade vs price

fig, ax = plt.subplots(figsize=(30,20))

p = sns.regplot(x="grade", y="price", data=final_df, label="grade", color='purple')

p.set_xlabel("Grade", fontsize = 40)
plt.xticks(fontsize=25)

p.set_ylabel("Price (in Thousands)", fontsize = 40)
p.yaxis.set_major_formatter(display_thousands)
plt.yticks(fontsize=25)

p.set_title("Home Price by Grade", fontsize = 50)
plt.figsize=(15,10)
plt.savefig('images/home_price_by_grade')

plt.show();
```

executed in 3.52s, finished 06:25:31 2021-12-22

8.3 Recommendation #3

The second most signicant corellation for Home Price is Building Grade. I recommend that all renovations and additions done by TrailBlazer be done to the highest level of quality in terms of both materials and workmanship. If the renovations can cause the house to increase in grade, it will improve the value of the house.

Most houses fall in the 7-8 grade. (71% of the records in the dataset). This is considered "Average" to "Just Above Average". Level 9 cites "better architectural design with extra interior and exterior design and quality."

The more unique and custom the features of the addtion are, the more that it will add value to the home.

9 Summary

I was tasked with analyzing home prices in King County to determine where the market for home renovations exists. Spefically, I was to determine which renovation types have the biggest increase in home value so that those services can be the focus of their marketing campaign.

I used the data provided and after cleaning and pre-processing, was able to develop a model which showed the effect that various features had on the price of homes.

My recommendation is that Trailblazer Renovations Inc. focus on features that add to the size of existing homes (ie Additions). These additions should be additional shared areas such as bathrooms, as extra bedrooms begin to have a negative correlation with home value. I also recommend that the additions be on existing floors of the house, as additional floors also begin to have a negative correlation.

I also recommend that these additions be done with the highest quality materials and to the highest standard as the Grade of the home is also a strong contributor to home value. The more unique and custom the features of the addtion are, the more that it will add value to the home.

As for marketing, my model determined that the location of houses played a major role in their value. Therefore, Trailblazer Inc. should focus on the zip codes with the highest coefficients from the final model. I provided a ranked list for them to use for this purpose.