1 King County Real Estate - Housing Analysis

1.1 Business Question:

King County Real Estate has hired us to investigate which features of a home have the greatest effect on price.

- They would like us to make a model to predict housing prices.
- From that model, they would like to know which factors have the largest effect on price.

1.2 Data Importing & Cleaning

The dataset "kc_house_data.csv" was obtained from the link below. King County 2014-2015 House Sales dataset

https://osf.io/twq9p/ (https://osf.io/twq9p/)

https://github.com/bigbenx3/housing_analysis_project/blob/main/column_names%20descriptions.md
(https://github.com/bigbenx3/housing_analysis_project/blob/main/column_names%20descriptions.md)

The descriptions for each feature/column in the dataset

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.formula.api import ols
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import RFE
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import scipy.stats as stats


sns.set_style("whitegrid")
%matplotlib inline

sns.set(rc={'figure.figsize':(11,8)})
```

In [2]:
```
url = "https://raw.githubusercontent.com/learn-co-curriculum/dsc-phase-2-project/main/data/kc_house_data.csv"
df_import = pd.read_csv(url, error_bad_lines=False)
```

In [3]:
```
df_import.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_b
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN	0.0	...	7	1180	0.0	1
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	...	7	2170	400.0	1
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	...	6	770	0.0	1
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	...	7	1050	910.0	1
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	...	8	1680	0.0	1

5 rows × 21 columns

Sneak preview of all the features in dataset.

1.2.0.1 Exploring Datatypes - What are the datatypes present?

Ideally we want to look at what the feature is and make sure certain features that are expected to be numerical, that they are in fact numerical and

rectify accordingly.

In [4]:
```
1 df_import.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
```

In [5]:
```
1 df_import.isnull().sum()
```

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

There are three columns/ features that have missing (NaN) or empty values:

- waterfront
- view
- yr_renovated

Let's see if there are duplicates in the dataset first

In [6]:
```
1 df_import.duplicated().sum()
```

Out[6]:
```
0
```

No duplicates to remove that may diminish the number of null values we have to contend with.

And for "waterfront" and "year renovated", the null values make up a large portion. Just dropping those values may not be the best idea.

Let's go through all the datatypes and make sure they correctly correspond to each feature. This may help our efforts to rectify the missing values, as well.

The link below gives descriptions for each feature of this dataset.
https://github.com/bigbenx3/housing_analysis_project/blob/main/column_names%20descriptions.md
(https://github.com/bigbenx3/housing_analysis_project/blob/main/column_names%20descriptions.md)

Looking at the descriptions from the .MD file in the link above and then later closely at values in some of the columns, we can discern whether the feature is categorical or quantitative in nature for this dataset.

We will ignore "id" and "date" features for now.

1.2.0.2 Switch to new dataframe name df1_mod_ed

This new df will be the copy of the original dataframe which we will modify.

In [7]:
```
df_mod_ed = df_import.copy()
df_mod_ed.head()
```

Out[7]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_b
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN	0.0	...	7	1180	0.0	1
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	...	7	2170	400.0	1
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	...	6	770	0.0	1
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	...	7	1050	910.0	1
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	...	8	1680	0.0	1

5 rows × 21 columns

Why are bathrooms and floors floats and not integers?

In [8]:
```
import pandas as pd
bathrooms_unique_values = df_mod_ed["bathrooms"].unique()
print(sorted(bathrooms_unique_values))
```

```
[0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25, 2.5, 2.75, 3.0, 3.25, 3.5, 3.75, 4.0, 4.25, 4.5, 4.75, 5.0, 5.25, 5.5, 5.7
5, 6.0, 6.25, 6.5, 6.75, 7.5, 7.75, 8.0]
```

According to the links below: https://www.badeloftusa.com/buying-guides/75-bathroom/ (https://www.badeloftusa.com/buying-guides/75-bathroom/)

https://www.badeloftusa.com/buying-guides/bathrooms/#:~:text=A%20half%20bathroom%20is%20a,just%20a%20toilet%20and%20sink (https://www.badeloftusa.com/buying-guides/bathrooms/#:~:text=A%20half%20bathroom%20is%20a,just%20a%20toilet%20and%20sink).

The decimals, (.25, .5, .75), all indicate the type and size of bathrooms within the homes. So, this isn't incorrect.

In [9]:
```
import pandas as pd
floors_unique_values = df_mod_ed["floors"].unique()
print(sorted(floors_unique_values))
```

```
[1.0, 1.5, 2.0, 2.5, 3.0, 3.5]
```

Accordingly, a half a story indicates smaller space on the "half floors" See the link below. https://www.gimme-shelter.com/what-is-a-1-5-storey-house-50104/ (https://www.gimme-shelter.com/what-is-a-1-5-storey-house-50104/)

Now, waterfront. One of the columns with null values.

In [10]:
```
sns.histplot(df_mod_ed["waterfront"])
```

Out[10]:
```
<AxesSubplot:xlabel='waterfront', ylabel='Count'>
```

Overwhelming number of homes with no waterfront view.

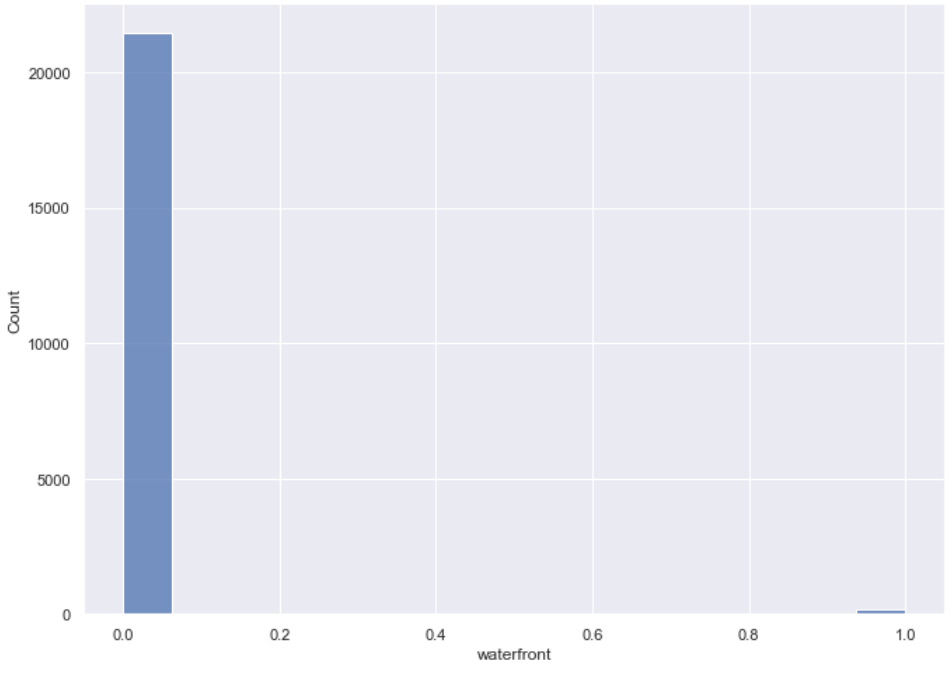
In [11]:
```
df_mod_ed.isnull().sum()
```

Out[11]:
```
id                0
date              0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront     2376
view             63
condition         0
grade             0
sqft_above        0
sqft_basement     0
yr_built          0
yr_renovated   3842
zipcode           0
lat               0
long              0
sqft_living15     0
sqft_lot15        0
dtype: int64
```

In [12]:
```
df_mod_ed["waterfront"] = df_mod_ed["waterfront"].fillna(0)
```

In [13]:
```
sns.histplot(df_mod_ed["waterfront"])
```

Out[13]:
```
<AxesSubplot:xlabel='waterfront', ylabel='Count'>
```



The new distribution and there no longer any nulls.

In [14]:
```
import pandas as pd
waterfront_unique_values = df_mod_ed["waterfront"].unique()
print(sorted(waterfront_unique_values))
```

```
[0.0, 1.0]
```

Base on the description, waterfront measures whether a house has a view to a waterfront. This list of unique values suggest that the "0" and "1" responds to "absence" and "presence" of a waterfront view, respectively. And that there are missing values.

This would conclude waterfront to be a categorical feature. And right now, it is a numerical value as a float.

In [15]:
```
df_mod_ed["waterfront"] = df_mod_ed["waterfront"].astype(str)
```

In [16]:
```
type(df_mod_ed["waterfront"][0])
```

Out[16]:
```
str
```

Now it is a categorical feature.

Double check with .info()

Again, "waterfront" no longer has missing values. The features "view" and "yr_renovated" still have missing values.

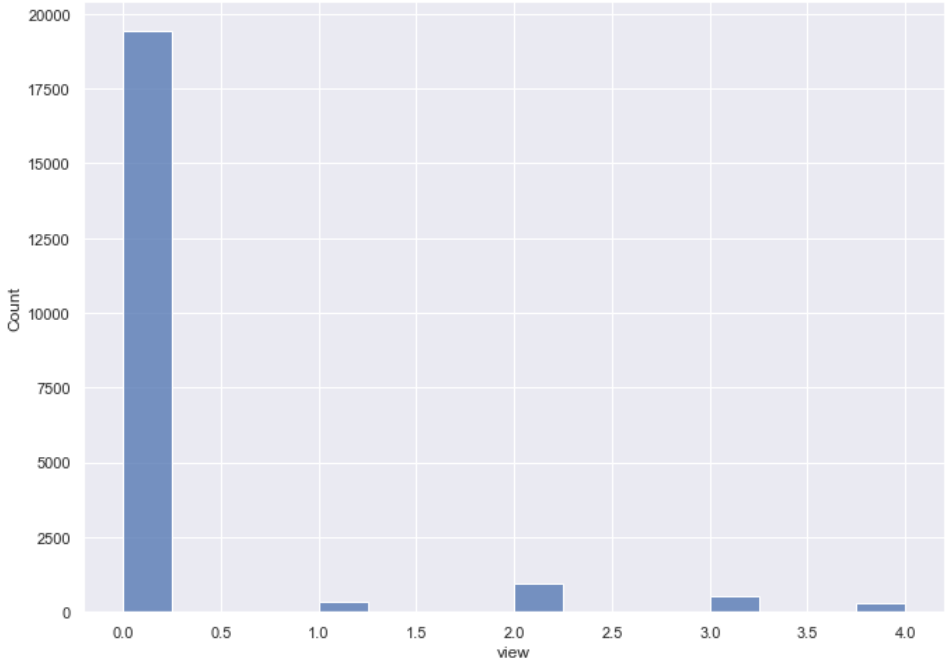
In [17]:
```
import pandas as pd
view_unique_values = df_mod_ed["view"].unique()
sorted(view_unique_values)
```

Out[17]:
```
[0.0, nan, 1.0, 2.0, 3.0, 4.0]
```

view - is a record for how many times the house has been viewed. And there are missing values.

In [18]:
```
sns.histplot(df_mod_ed["view"])
```

Out[18]:
```
<AxesSubplot:xlabel='view', ylabel='Count'>
```



In [19]:
```
df_mod_ed.view.describe()
```

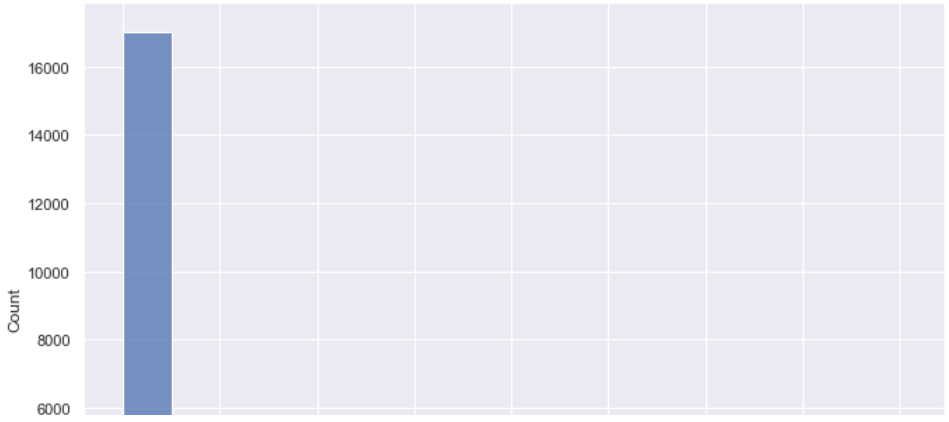
Out[19]:
```
count    21534.000000
mean         0.233863
std          0.765686
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          4.000000
Name: view, dtype: float64
```

It's a bit odd to find that a majority of the homes have not been viewed at any one time, before presumably being sold. I wonder why that is...

Let's try to fill both missing values for "yr_renovated" and "view" with the median

In [20]:
```
1 sns.histplot(df_mod_ed["yr_renovated"])
```

Out[20]: <AxesSubplot:xlabel='yr_renovated', ylabel='Count'>



In [21]:
```
1 view_median = df_mod_ed.view.median()
2 view_median
```

Out[21]: 0.0

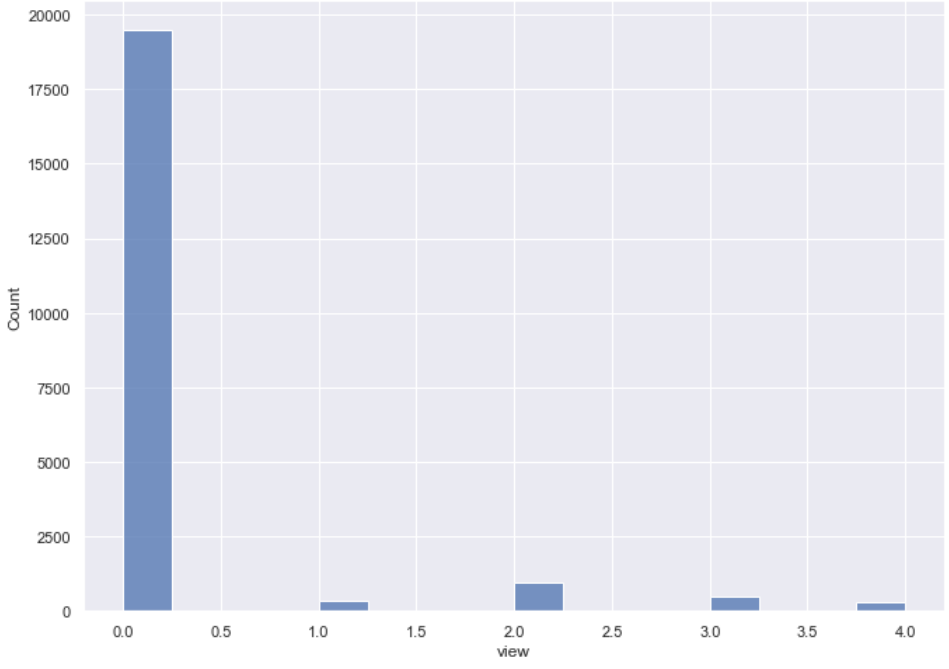
In [22]:
```
1 yr_renovated_median = df_mod_ed.yr_renovated.median()
2 yr_renovated_median
```

Out[22]: 0.0

In [23]:
```
1 df_mod_ed.fillna(value = {"view":view_median,"yr_renovated":yr_renovated_median}
2                                 ,inplace = True)
```

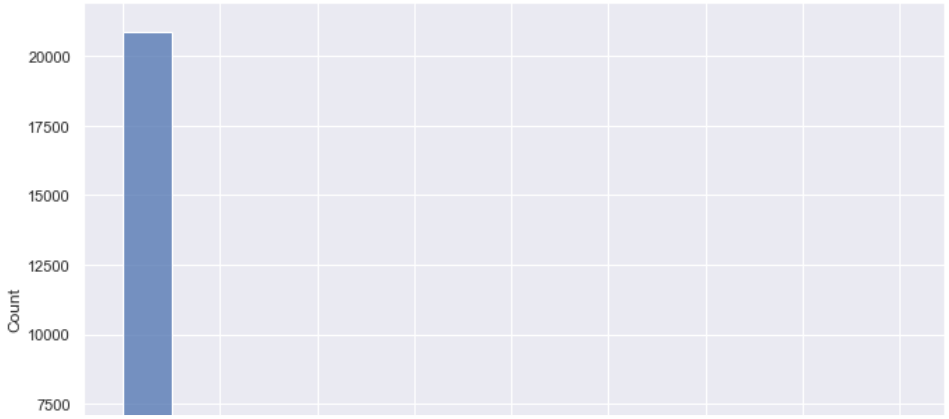
In [24]:
```
1 sns.histplot(df_mod_ed["view"])
```

Out[24]: <AxesSubplot:xlabel='view', ylabel='Count'>



In [25]:
```
1 sns.histplot(df_mod_ed["yr_renovated"])
```

Out[25]: <AxesSubplot:xlabel='yr_renovated', ylabel='Count'>



1.2.0.3 IMPROVE?

Turns out that for both columns, the median was 0. For now, that will have to do to fill in the missing values for "yr_renovated" and "view"

In [26]:
```
1 df_mod_ed.isnull().sum()
```

Out[26]:
```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

1.2.0.4 No nulls for Now...

Continuing to check thru the features and whether they have appropriate corresponding datatypes.

In [27]:
```
1 import pandas as pd
2 condition_unique_values = df_mod_ed["condition"].unique()
3 print(sorted(condition_unique_values))
```

```
[1, 2, 3, 4, 5]
```

Condition may be presumed to contain qualitative data. However, the numerical scale for condition, so quantitative feature.

In [28]:
```
1 import pandas as pd
2 grade_unique_values = df_mod_ed["grade"].unique()
3 print(sorted(grade_unique_values))
```

```
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
```

Grading scales may be either categorical or quantitative. Grading scale from 3 to 13, quantitative feature.

In [29]:
```
import pandas as pd
sqft_basement_unique_values = df_mod_ed["sqft_basement"].unique()
sorted(sqft_basement_unique_values)
```

Out[29]:
```
['0.0',
 '10.0',
 '100.0',
 '1000.0',
 '1008.0',
 '1010.0',
 '1020.0',
 '1024.0',
 '1030.0',
 '1040.0',
 '1050.0',
 '1060.0',
 '1070.0',
 '1080.0',
 '1090.0',
 '110.0',
 '1100.0',
 '1110.0',
 '1120.0',
```

So there is a "?" as the last unique value for this column.

In [30]:
```
type(df_mod_ed["sqft_basement"][0])
```

Out[30]:
```
str
```

And sqft_basement column should hold numerical values corresponding to square footage of the basement. Right now, they're all string values.

We need to replace the "?" first before converting the "sqft_basement" values to an integer type.

In [31]:
```
df_mod_ed.sqft_basement.value_counts()
```

Out[31]:
```
0.0       12826
?           454
600.0       217
500.0       209
700.0       208
           ...
1930.0        1
2720.0        1
4820.0        1
1525.0        1
3480.0        1
Name: sqft_basement, Length: 304, dtype: int64
```

There are 454 entries corresponding to the "?". And yet here the data type states integer...

In [32]:
```
sns.histplot(df_mod_ed["sqft_basement"])
```

Out[32]:
```
<AxesSubplot:xlabel='sqft_basement', ylabel='Count'>
```

So with this sort of huge disparity in the distribution of values for sqft_basement (square footage of the basement) and inputting any value might

introduce bias...

In [33]:
```
df_mod_ed.head()
```

Out[33]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_b
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	...	7	1180	0.0	1
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	...	7	2170	400.0	1
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	...	6	770	0.0	1
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	...	7	1050	910.0	1
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	...	8	1680	0.0	1

5 rows × 21 columns

It would seem that... sqft_living, sqft_livingsquare (footage of the home), minus the sqft_above (square footage of house apart from basement) equals sqft_basement (square footage of the basement).

sqft_living - sqft_above = sqft_basement

Therefore, sqft_above and sqft_basement is a bit redundant info regarding square footage of the home. It seems the most appropriate action seeing the huge disparity again in sqft_basement.

We will decide to drop "sqft_above" and "sqft_basement"

In [34]:
```
df_mod_ed = df_mod_ed.drop(columns=["sqft_above", "sqft_basement"])
df_mod_ed.head()
```

Out[34]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renovated	zi
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	3	7	1955	0.0	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	3	7	1951	1991.0	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	3	6	1933	0.0	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	5	7	1965	0.0	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	3	8	1987	0.0	

The columns "sqft_above", "sqft_basement" have been dropped.

Now, to look at "zipcode"

Zipcode should be categorical variable and not an "int64" integer. So we have to change the values to string.

In [35]:
```
df_mod_ed["zipcode"] = df_mod_ed["zipcode"].astype(str)
```

In [36]:
```
type(df_mod_ed["zipcode"][0])
```

Out[36]:
```
str
```

Note: categorical data need to be encoded to be used in a model that accepts only numerical values.

1.2.0.5 NEW DF - df1

Before we start removing outliers, we'll use a copy of df_mod_ed to continue cleaning.

In [37]:
```
df1 = df_mod_ed.copy()
df1.head()
```

Out[37]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renovated	zi
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	3	7	1955	0.0	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	3	7	1951	1991.0	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	3	6	1933	0.0	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	5	7	1965	0.0	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	3	8	1987	0.0	

1.3 Data - Cleaning & Manipulation

1.3.0.1 Outlier Detection

Let's again look back at the sort of data that we have, using the .describe() function. Are there outliers we should remove first?

In [38]:
```
df1.describe()
```

Out[38]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	view	condition	grade
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	21597.000000	21597.000000	21597.000000	21597.000000
mean	4.580474e+09	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	1.494096	0.233181	3.409825	7.657915
std	2.876736e+09	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	0.539683	0.764673	0.650546	1.173200
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.000000	0.000000	1.000000	3.000000
25%	2.123049e+09	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1.000000	0.000000	3.000000	7.000000
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	3.000000	7.000000
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	2.000000	0.000000	4.000000	8.000000
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	4.000000	5.000000	13.000000

- "id" but we will ignore this feature because the id is unique identifier for each house
- "date" when house was sold, sort of irrelevant at face value to house price
- "view" is how many times the house was viewed. This isn't really relevant but will be confirmed with the .corr(). And they are just counts of the times viewed, not really applicable to eliminate outliers for.
- "waterfront" is based on presence or absence of a waterfront. The "1" and "0" are indicative of presence of waterfront. No outliers there.

So we see a couple of max values that are way outside of the 75 percentile:

- "bedrooms" : 33 bedrooms max when the 75% is 4 bedrooms
- "bathrooms": 8 bathrooms max when 75% is 2.5 bathrooms
- "sqft_living": 13540 sqft max when 75% is 2550 sqft
- "sqft_lot": 1.65 million sqft when 75% is 10688 sqft
- "sqft_above": 9410 sqft max when 75% is 2210 sqft
- "sqft_basement": 4820 sqft max when 75% is 560 sqft
- "sqft_living15": 6210 sqft max when 75% is 2360 sqft
- "sqft_lot15": 871200 sqft max, 75% is 10083 sqft

You may have noticed that such features as "price", "grade", "condition", "yr_built", "yr_renovated", "zipcode", "lat", "long" were not included as features needing to remove outliers. These will be reviewed later down the line.

So to prevent skewing of our model, let's try and remove the outliers from all the columns.

1.3.0.2 Bedrooms

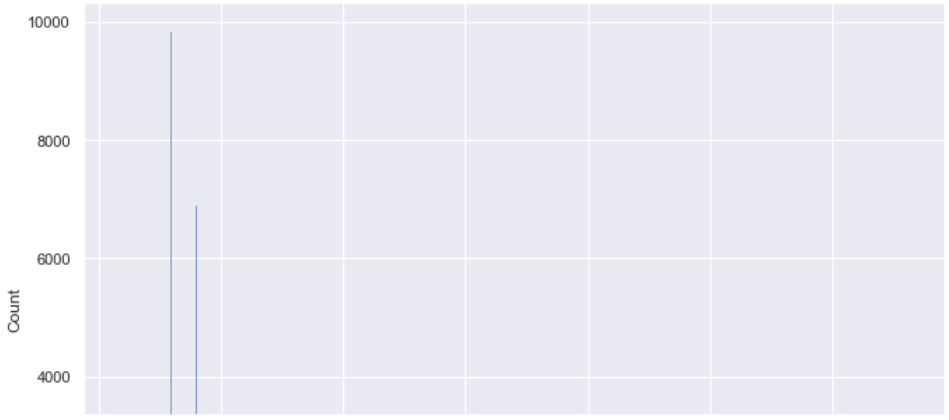
First, let's set up an initial test run that will iterate through all the columns in our data frame. We'll use "bedrooms" to test.

In [39]:
```
df1.bedrooms.describe()
```

Out[39]:
```
count    21597.000000
mean         3.373200
std          0.926299
min          1.000000
25%          3.000000
50%          3.000000
75%          4.000000
max         33.000000
Name: bedrooms, dtype: float64
```

In [40]:
```
sns.histplot(df1["bedrooms"])
```

Out[40]:
```
<AxesSubplot:xlabel='bedrooms', ylabel='Count'>
```



We see the histogram display extend to include the 33 bedrooms, however, most of the data is within 7 bedrooms.

Let's set up the upper and lower limits.

In [41]:
```
upper_limit = df1.bedrooms.mean() + 3*df1.bedrooms.std()
upper_limit
```

Out[41]:
```
6.152096665105352
```

In [42]:
```
lower_limit = df1.bedrooms.mean() - 3*df1.bedrooms.std()
lower_limit
```

Out[42]:
```
0.5943032978524658
```

So, these are the two boundaries that will help discern what is and what isn't an outlier.

In [43]:
```
df1[(df1.bedrooms>upper_limit)|(df1.bedrooms<lower_limit)]
```

Out[43]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renc
556	5486800070	6/20/2014	1950000.0	7	3.50	4640	15235	2.0	0.0	1.0	3	11	1965	
1134	4024100951	1/5/2015	420000.0	7	3.00	2940	8624	1.0	0.0	0.0	3	8	1977	
1239	7227802030	6/23/2014	350000.0	7	3.00	2800	9569	1.0	0.0	2.0	3	7	1963	
1658	9126101740	12/4/2014	490000.0	8	5.00	2800	2580	2.0	0.0	0.0	3	8	1997	
3717	5451100490	1/15/2015	884900.0	7	4.75	5370	10800	1.5	0.0	0.0	3	8	1967	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
18808	4040500100	10/20/2014	539000.0	7	2.25	2620	6890	2.0	0.0	0.0	4	7	1961	
18960	1778360150	6/20/2014	1240000.0	7	5.50	6630	13782	2.0	0.0	0.0	3	10	2004	
19239	8812401450	12/29/2014	660000.0	10	3.00	2920	3745	2.0	0.0	0.0	4	7	1913	
19287	3756900027	11/25/2014	575000.0	8	3.00	3840	15990	1.0	0.0	0.0	3	7	1961	

So, these would be our outliers, 75 of them for the "bedrooms" column. Those entries below the "lower_limit" and above the "upper_limit".

In [44]:
```
df_out_rmv1 = df1[(df1.bedrooms<upper_limit) & (df1.bedrooms>lower_limit)]
df_out_rmv1
```

Out[44]:

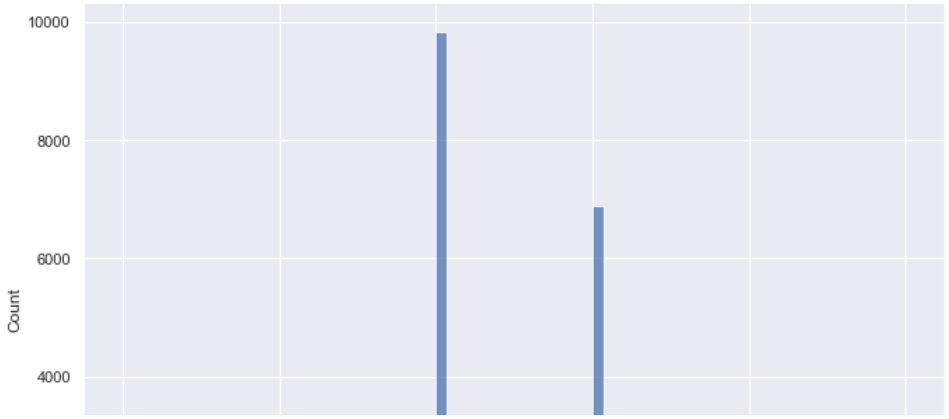
	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renov
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	3	7	1955	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	3	7	1951	1
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	3	6	1933	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	5	7	1965	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	3	8	1987	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	0.0	0.0	3	8	2009	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	0.0	0.0	3	8	2014	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	0.0	0.0	3	7	2009	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	0.0	0.0	3	8	2004	

In [45]:
```
df_out_rmv1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21535 entries, 0 to 21596
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21535 non-null  int64
 1   date           21535 non-null  object
 2   price          21535 non-null  float64
 3   bedrooms       21535 non-null  int64
 4   bathrooms      21535 non-null  float64
 5   sqft_living    21535 non-null  int64
 6   sqft_lot       21535 non-null  int64
 7   floors         21535 non-null  float64
 8   waterfront     21535 non-null  object
 9   view           21535 non-null  float64
 10  condition      21535 non-null  int64
 11  grade          21535 non-null  int64
 12  yr_built       21535 non-null  int64
 13  yr_renovated   21535 non-null  float64
 14  zipcode        21535 non-null  object
```

In [46]:
```
1 sns.histplot(df_out_rmv1["bedrooms"])
```

Out[46]:
```
<AxesSubplot:xlabel='bedrooms', ylabel='Count'>
```



So we started out with 21,613 non-null entries and now end up with 21538 remaining entries (still non-null entries) as displayed by .info(). That is congruent to 75 outliers we saw before removed from the original dataframe.

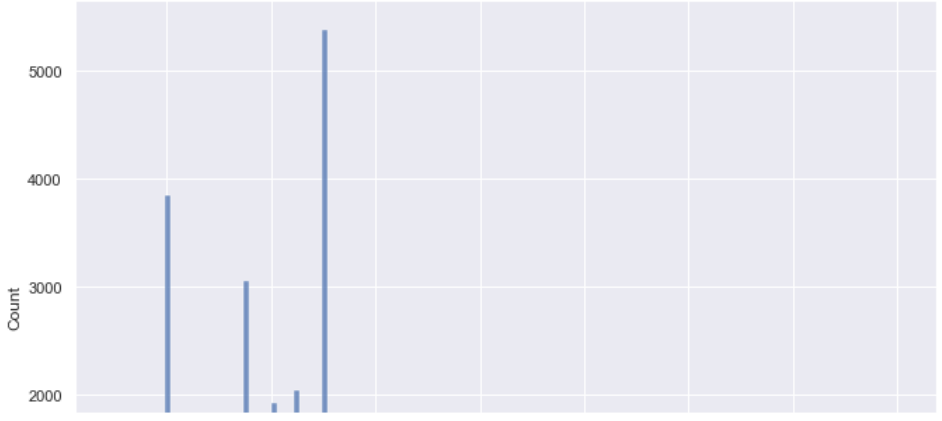
1.3.0.3 Bathrooms

In [47]:
```
1 df_out_rmv1.bathrooms.describe()
```

Out[47]:
```
count    21535.000000
mean         2.111400
std          0.762291
min          0.500000
25%          1.750000
50%          2.250000
75%          2.500000
max          8.000000
Name: bathrooms, dtype: float64
```

In [48]:
```
1 sns.histplot(df_out_rmv1["bathrooms"])
```

Out[48]:
```
<AxesSubplot:xlabel='bathrooms', ylabel='Count'>
```



Let's set up the upper and lower limits.

In [49]:
```
1 upper_limit = df_out_rmv1.bathrooms.mean() + 3*df_out_rmv1.bathrooms.std()
2 upper_limit
```

Out[49]:
```
4.398271820553814
```

In [50]:
```
1 lower_limit = df_out_rmv1.bathrooms.mean() - 3*df_out_rmv1.bathrooms.std()
2 lower_limit
```

Out[50]:
```
-0.17547172768174502
```

So, these are the two boundaries that will help discern what is and what isn't an outlier.

In [51]:
```
df_out_rmv1[(df_out_rmv1.bathrooms>upper_limit)|(df_out_rmv1.bathrooms<lower_limit)]
```

Out[51]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renc
5	7237550310	5/12/2014	1230000.0	4	4.50	5420	101930	1.0	0.0	0.0	3	11	2001	
270	4054500390	10/7/2014	1370000.0	4	4.75	5310	57346	2.0	0.0	0.0	4	11	1989	
300	3225069065	6/24/2014	3080000.0	4	5.00	4550	18641	1.0	1.0	4.0	3	10	2002	
450	4055700030	5/2/2015	1450000.0	3	4.50	3970	24920	2.0	0.0	2.0	3	10	1977	
527	3225079035	6/18/2014	1600000.0	6	5.00	6050	230652	2.0	0.0	3.0	3	11	2001	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21328	8835770170	8/22/2014	1490000.0	5	6.00	6880	279968	2.0	0.0	3.0	3	12	2007	
21478	2413910120	7/2/2014	915000.0	3	4.50	3850	62726	2.0	0.0	0.0	3	10	2013	
21490	2524069097	5/9/2014	2240000.0	5	6.50	7270	130017	2.0	0.0	0.0	3	12	2010	
21535	1561750040	12/24/2014	1380000.0	5	4.50	4350	13405	2.0	0.0	0.0	3	11	2014	

Those entries below the "lower_limit" and above the "upper_limit".

In [52]:
```
df_out_rmv2 = df_out_rmv1[(df_out_rmv1.bathrooms<upper_limit) & (df_out_rmv1.bathrooms>lower_limit)]
df_out_rmv2
```

Out[52]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renov
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	3	7	1955	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	3	7	1951	1
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	3	6	1933	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	5	7	1965	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	3	8	1987	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	0.0	0.0	3	8	2009	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	0.0	0.0	3	8	2014	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	0.0	0.0	3	7	2009	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	0.0	0.0	3	8	2004	

In [53]:
```
sns.histplot(df_out_rmv2["bathrooms"])
```

Out[53]:
```
<AxesSubplot:xlabel='bathrooms', ylabel='Count'>
```

We see the new histogram display after removal of outliers.

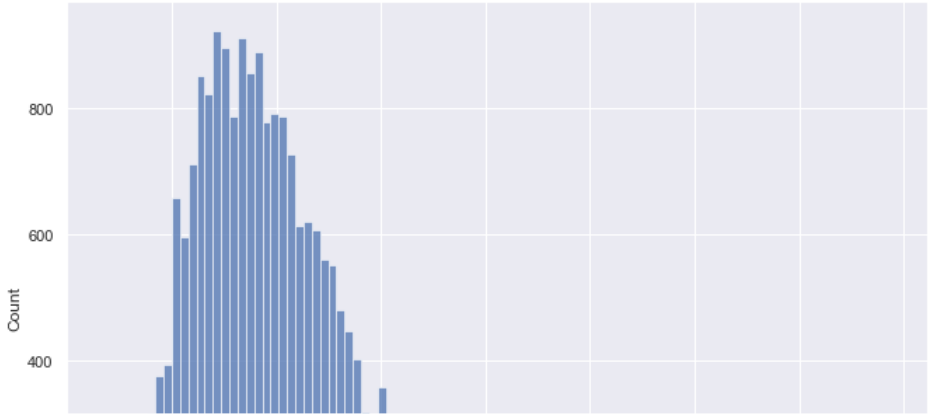
1.3.0.4 Sqft_living

In [54]:
```
1 df_out_rmv2.sqft_living.describe()
```

Out[54]:
```
count    21363.000000
mean      2051.320414
std        861.820882
min        370.000000
25%       1420.000000
50%       1900.000000
75%       2520.000000
max       7850.000000
Name: sqft_living, dtype: float64
```

In [55]:
```
1 sns.histplot(df_out_rmv2["sqft_living"])
```

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



Let's set up the upper and lower limits.

In [56]:
```
1 upper_limit = df_out_rmv2.sqft_living.mean() + 3*df_out_rmv2.sqft_living.std()
2 upper_limit
```

Out[56]:
```
4636.783060440892
```

In [57]:
```
1 lower_limit = df_out_rmv2.sqft_living.mean() - 3*df_out_rmv2.sqft_living.std()
2 lower_limit
```

Out[57]:
```
-534.142232841772
```

So, these are the two boundaries that will help discern what is and what isn't an outlier.

In [58]:
```
1 df_out_rmv2[(df_out_rmv2.sqft_living>upper_limit)|(df_out_rmv2.sqft_living<lower_limit)]
```

Out[58]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renov
70	1525059190	9/12/2014	1040000.0	5	3.25	4770	50094	1.0	0.0	0.0	4	11	1973	
153	7855801670	4/1/2015	2250000.0	4	3.25	5180	19850	2.0	0.0	3.0	3	12	2006	
269	7960900060	5/4/2015	2900000.0	4	3.25	5050	20100	1.5	0.0	2.0	3	11	1982	
384	713500030	7/28/2014	1350000.0	5	3.50	4800	14984	2.0	0.0	2.0	3	11	1998	
419	8678500060	7/10/2014	1550000.0	5	4.25	6070	171626	2.0	0.0	0.0	3	12	1999	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21451	2311400056	12/1/2014	1990000.0	5	3.50	5230	8960	2.0	0.0	0.0	3	11	2014	
21504	7237550100	8/25/2014	1410000.0	4	4.00	4920	50621	2.0	0.0	0.0	3	10	2012	
21505	7430500110	12/9/2014	1380000.0	5	3.50	5150	12230	2.0	0.0	2.0	3	10	2007	
21514	8964800330	4/7/2015	3000000.0	4	3.75	5090	14823	1.0	0.0	0.0	3	11	2013	

Those entries below the "lower_limit" and above the "upper_limit".

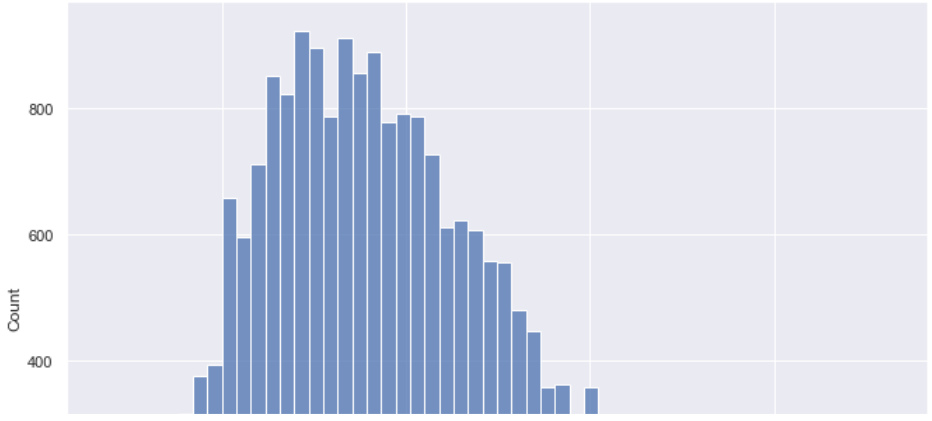
In [59]:
```
df_out_rmv3 = df_out_rmv2[(df_out_rmv2.sqft_living<upper_limit) & (df_out_rmv2.sqft_living>lower_limit)]
df_out_rmv3
```

Out[59]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renov
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	3	7	1955	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	3	7	1951	1
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	3	6	1933	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	5	7	1965	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	3	8	1987	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	0.0	0.0	3	8	2009	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	0.0	0.0	3	8	2014	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	0.0	0.0	3	7	2009	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	0.0	0.0	3	8	2004	

In [60]:
```
sns.histplot(df_out_rmv3["sqft_living"])
```

Out[60]: <AxesSubplot:xlabel='sqft_living', ylabel='Count'>



We see the new histogram display after removal of outliers.

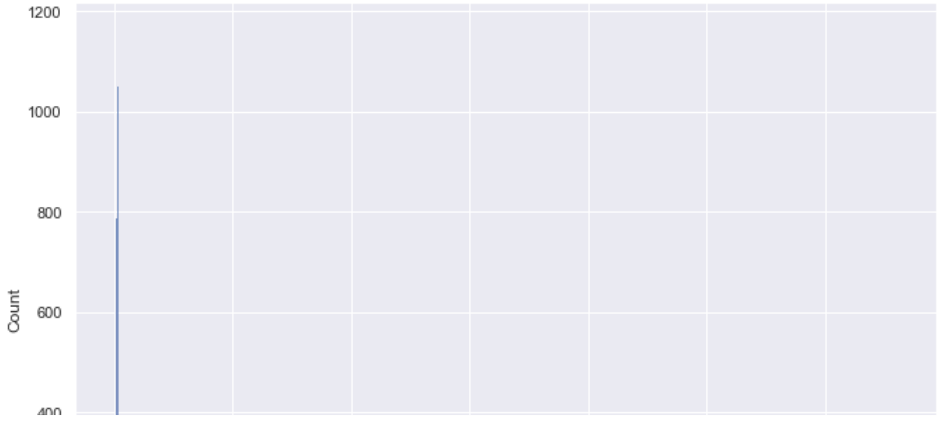
1.3.0.5 Sqft_lot

In [61]:
```
df_out_rmv3.sqft_lot.describe()
```

Out[61]:
```
count    2.115000e+04
mean     1.457195e+04
std      3.989216e+04
min      5.200000e+02
25%      5.005000e+03
50%      7.560000e+03
75%      1.046400e+04
max      1.651359e+06
Name: sqft_lot, dtype: float64
```

In [62]:
```
1 sns.histplot(df_out_rmv3["sqft_lot"])
```

Out[62]: <AxesSubplot:xlabel='sqft_lot', ylabel='Count'>



Let's set up the upper and lower limits.

In [63]:
```
1 upper_limit = df_out_rmv3.sqft_lot.mean() + 3*df_out_rmv3.sqft_lot.std()
2 upper_limit
```

Out[63]: 134248.41899222514

In [64]:
```
1 lower_limit = df_out_rmv3.sqft_lot.mean() - 3*df_out_rmv3.sqft_lot.std()
2 lower_limit
```

Out[64]: -105104.51695912822

So, these are the two boundaries that will help discern what is and what isn't an outlier.

In [65]:
```
1 df_out_rmv3[(df_out_rmv3.sqft_lot>upper_limit)|(df_out_rmv3.sqft_lot<lower_limit)]
```

Out[65]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renc
145	1526069017	12/3/2014	921500.0	4	2.50	3670	315374	2.0	0.0	0.0	4	9	1994	
199	1222069094	10/14/2014	385000.0	3	1.75	1350	155073	1.0	0.0	0.0	4	7	1969	
238	326069104	7/1/2014	800000.0	3	3.50	3830	221284	2.0	0.0	0.0	3	10	1993	
380	1726059053	9/16/2014	270000.0	2	1.50	1380	209959	1.0	0.0	0.0	1	6	1954	
411	2422029094	7/16/2014	517534.0	2	1.00	833	143947	1.0	0.0	0.0	3	5	2006	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21074	1624079024	5/15/2014	720000.0	3	2.50	3150	151588	2.0	0.0	0.0	3	9	2007	
21309	2826079027	11/12/2014	659000.0	3	2.50	3090	384634	2.0	0.0	0.0	3	8	2007	
21335	3421069049	10/21/2014	565000.0	2	1.75	1130	276170	1.0	0.0	0.0	3	8	2006	
21415	2725079018	5/9/2014	800000.0	4	3.25	3540	159430	2.0	0.0	0.0	3	9	2007	

Those entries below the "lower_limit" and above the "upper_limit".

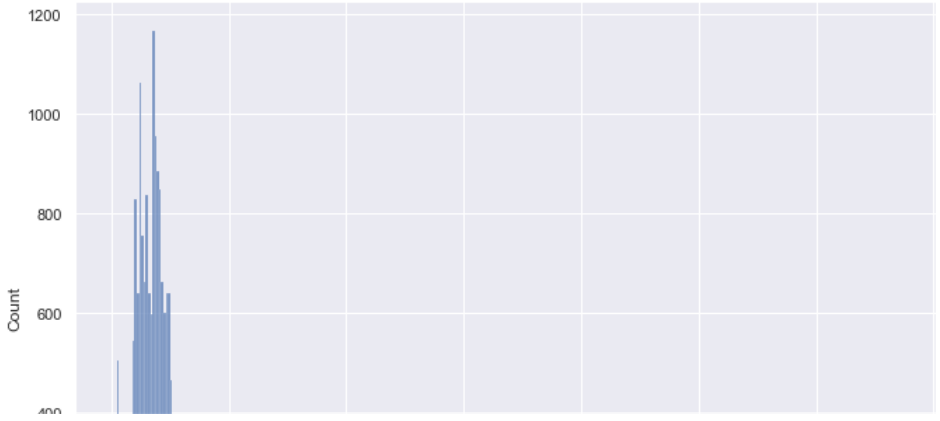
In [66]:
```
df_out_rmv4 = df_out_rmv3[(df_out_rmv3.sqft_lot<upper_limit) & (df_out_rmv3.sqft_lot>lower_limit)]
df_out_rmv4
```

Out[66]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renov
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	3	7	1955	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	3	7	1951	1
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	3	6	1933	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	5	7	1965	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	3	8	1987	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	0.0	0.0	3	8	2009	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	0.0	0.0	3	8	2014	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	0.0	0.0	3	7	2009	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	0.0	0.0	3	8	2004	

In [67]:
```
sns.histplot(df_out_rmv4["sqft_lot"])
```

Out[67]: <AxesSubplot:xlabel='sqft_lot', ylabel='Count'>



We see the new histogram display after removal of outliers.

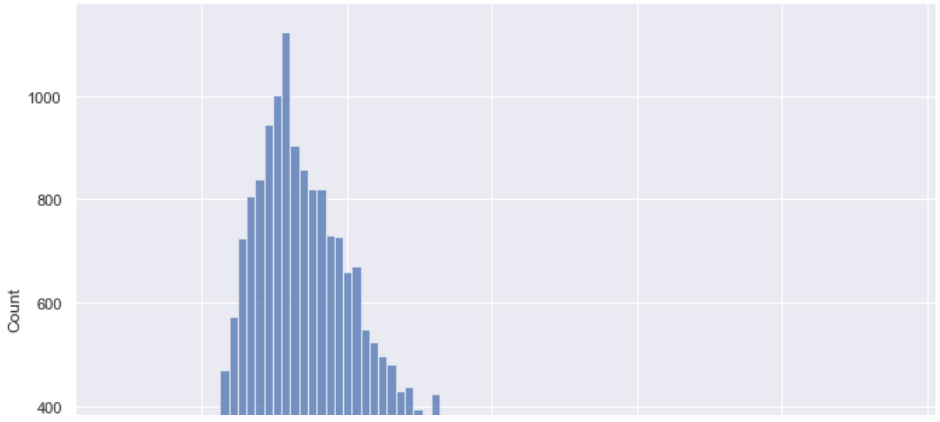
1.3.0.6 Sqft_living15

In [68]:
```
df_out_rmv4.sqft_living15.describe()
```

Out[68]:
```
count    20827.000000
mean      1953.841120
std        647.662431
min        399.000000
25%       1480.000000
50%       1820.000000
75%       2320.000000
max       5790.000000
Name: sqft_living15, dtype: float64
```

In [69]:
```
1 sns.histplot(df_out_rmv4["sqft_living15"])
```

Out[69]:
```
<AxesSubplot:xlabel='sqft_living15', ylabel='Count'>
```



So, these are the two boundaries that will help discern what is and what isn't an outlier.

In [70]:
```
1 df_out_rmv4[(df_out_rmv4.sqft_living15>upper_limit)|(df_out_rmv4.sqft_living15<lower_limit)]
```

Out[70]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renovated	zipcode	lat	long	sq

No outliers displayed.

1.3.0.7 Sqft_lot15

In [71]:
```
1 df_out_rmv4.sqft_lot15.describe()
```

Out[71]:
```
count     20827.000000
mean      10299.062563
std       15587.828662
min         651.000000
25%        5040.000000
50%        7522.000000
75%        9828.000000
max      434728.000000
Name: sqft_lot15, dtype: float64
```

In [72]:
```
1 sns.histplot(df_out_rmv4["sqft_lot15"])
```

Out[72]:
```
<AxesSubplot:xlabel='sqft_lot15', ylabel='Count'>
```

Let's set up the upper and lower limits.

In [73]:
```
upper_limit = df_out_rmv4.sqft_lot15.mean() + 3*df_out_rmv4.sqft_lot15.std()
upper_limit
```

Out[73]: 57062.54854896887

In [74]:
```
lower_limit = df_out_rmv4.sqft_lot15.mean() - 3*df_out_rmv4.sqft_lot15.std()
lower_limit
```

Out[74]: -36464.42342293056

So, these are the two boundaries that will help discern what is and what isn't an outlier.

In [75]:
```
df_out_rmv4[(df_out_rmv4.sqft_lot15>upper_limit)|(df_out_rmv4.sqft_lot15<lower_limit)]
```

Out[75]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renc
49	822039084	3/11/2015	1350000.0	3	2.50	2753	65005	1.0	1.0	2.0	5	9	1953	
98	722079104	7/11/2014	314000.0	3	1.75	1810	41800	1.0	0.0	0.0	5	7	1980	
132	1243100136	6/12/2014	784000.0	3	3.50	3950	111078	1.5	0.0	0.0	3	9	1989	
147	3224079105	8/6/2014	430000.0	2	2.50	2420	60984	2.0	0.0	0.0	3	7	2007	
198	2824079053	1/13/2015	440000.0	3	2.50	1910	66211	2.0	0.0	0.0	3	7	1997	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
21271	123059127	5/2/2014	625000.0	4	3.25	2730	54014	1.0	0.0	0.0	3	9	2007	
21345	3123089027	7/21/2014	472000.0	3	2.50	3800	104979	2.0	0.0	0.0	3	8	2005	
21370	774101755	4/17/2015	320000.0	3	1.75	1790	66250	1.5	0.0	0.0	3	7	2003	
21470	98300230	4/28/2015	1460000.0	4	4.00	4620	130208	2.0	0.0	0.0	3	10	2014	

Those entries below the "lower_limit" and above the "upper_limit".

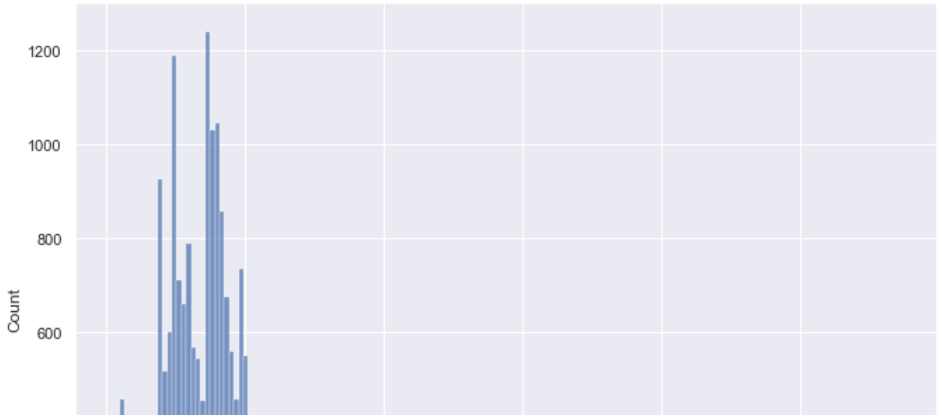
In [76]:
```
df_out_rmv_all = df_out_rmv4[(df_out_rmv4.sqft_lot15<upper_limit) & (df_out_rmv4.sqft_lot15>lower_limit)]
df_out_rmv_all
```

Out[76]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renov
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	3	7	1955	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	3	7	1951	1
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	3	6	1933	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	5	7	1965	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	3	8	1987	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	0.0	0.0	3	8	2009	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	0.0	0.0	3	8	2014	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	0.0	0.0	3	7	2009	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	0.0	0.0	3	8	2004	

In [77]:
```
1 sns.histplot(df_out_rmv_all["sqft_lot15"])
```

Out[77]: <AxesSubplot:xlabel='sqft_lot15', ylabel='Count'>



We see the new histogram display after removal of outliers.

In [78]:
```
1 df_out_rmv_all.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20552 entries, 0 to 21596
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             20552 non-null  int64
 1   date           20552 non-null  object
 2   price          20552 non-null  float64
 3   bedrooms       20552 non-null  int64
 4   bathrooms      20552 non-null  float64
 5   sqft_living    20552 non-null  int64
 6   sqft_lot       20552 non-null  int64
 7   floors         20552 non-null  float64
 8   waterfront     20552 non-null  object
 9   view           20552 non-null  float64
 10  condition      20552 non-null  int64
 11  grade          20552 non-null  int64
 12  yr_built       20552 non-null  int64
 13  yr_renovated   20552 non-null  float64
```

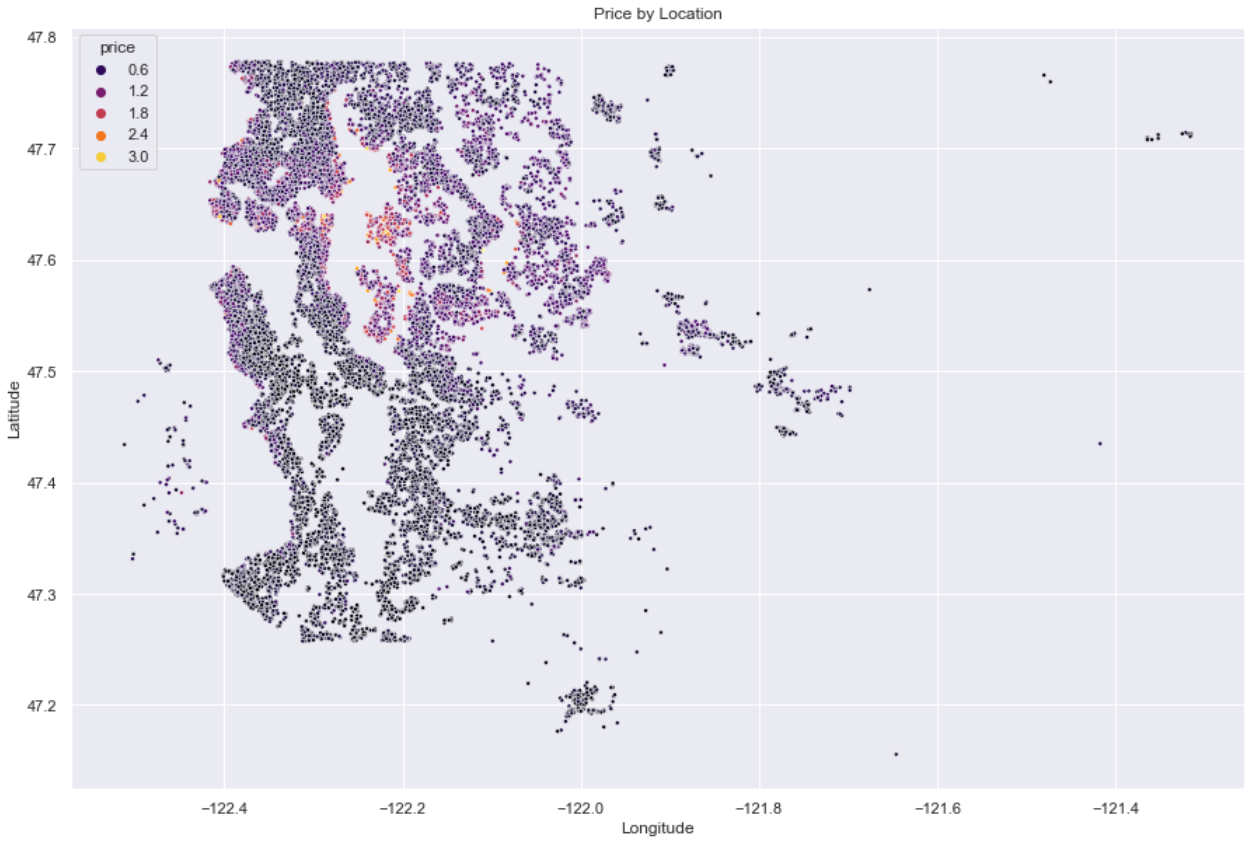
So after all the outliers are removed, we are left with 20012 non-null entries in our dataframe, now renamed as "df_out_rmv_all", indicated by our .info().

1.3.0.8 Location

We can plot the data from "df_out_rmv_all" to see the location of each house in respective of its "price"

In [79]:
```
fig = plt.figure(figsize=(15,10))
ax = sns.scatterplot(x=df_out_rmv_all["long"], y=df_out_rmv_all["lat"], hue=df_out_rmv_all["price"], palette="inferno"
                     marker=".")
ax.set( xlabel="Longitude",
        ylabel="Latitude",
        title="Price by Location")
```

Out[79]:
```
[Text(0.5, 0, 'Longitude'),
 Text(0, 0.5, 'Latitude'),
 Text(0.5, 1.0, 'Price by Location')]
```



To be honest, the graphical plot isn't that indicative of location being a huge contributor to price. There seems to be just pockets of above 1.2 million USD homes and the majority of homes priced at, less than or equal to 1.2 million USD.

In [80]:
```
1 df_out_rmv_all["price"].describe()
```

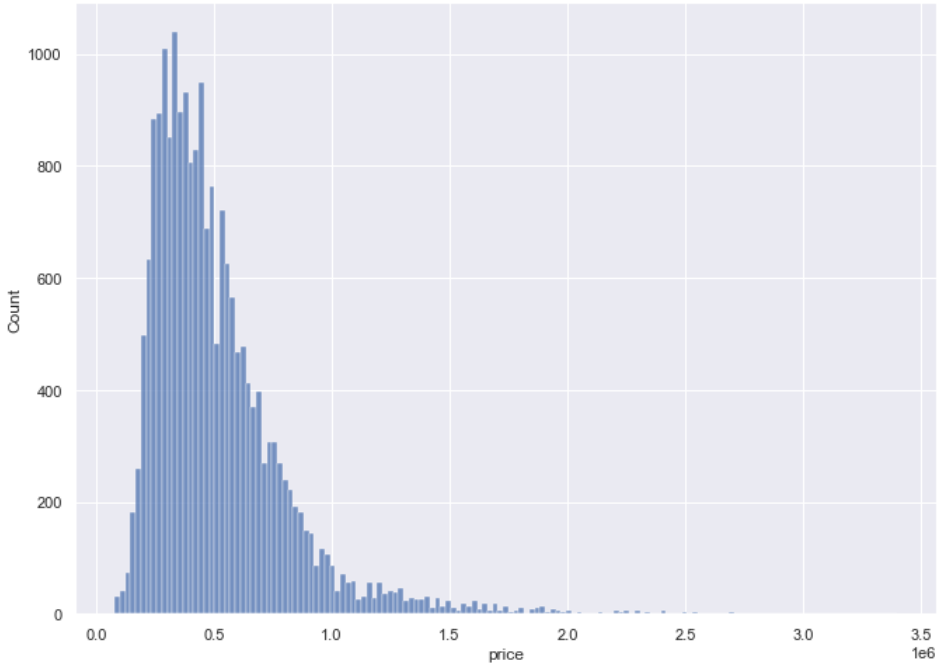
Out[80]:
```
count    2.055200e+04
mean     5.158910e+05
std      3.014929e+05
min      7.800000e+04
25%      3.180000e+05
50%      4.450000e+05
75%      6.253125e+05
max      3.400000e+06
Name: price, dtype: float64
```

And all these prices shown on the scatter plot are in millions of $ (2014 - 2015 USD). Our max is at 3.1 million dollars here displayed by our .describe() function.

From our scatter plot of location and the .describe() function, it would appear that we may have skewed distribution of prices. Let's see a histogram plot of price.

In [81]:
```
1 sns.histplot(df_out_rmv_all["price"])
```

Out[81]:
```
<AxesSubplot:xlabel='price', ylabel='Count'>
```



After the other features with max well over the 75 percentile removed of their outliers. We will revisit the features we had put aside before.

These features include "price", "grade", "condition", "yr_built", "yr_renovated", "zipcode", "lat", "long" that were not included as features needing to remove outliers, before. Let's see their distributions now.

1.3.0.9 Price

Let's set up the upper and lower limits.

In [82]:
```
1 upper_limit = df_out_rmv_all.price.mean() + 3*df_out_rmv_all.price.std()
2 upper_limit
```

Out[82]:
```
1420369.5904382865
```

In [83]:
```
1 lower_limit = df_out_rmv_all.price.mean() - 3*df_out_rmv_all.price.std()
2 lower_limit
```

Out[83]:
```
-388587.6642997113
```

So, these are the two boundaries that will help discern what is and what isn't an outlier.

In [84]:
```
df_out_rmv_all[(df_out_rmv_all.price>upper_limit)|(df_out_rmv_all.price<lower_limit)]
```

Out[84]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renc
21	2524049179	8/26/2014	2000000.0	3	2.75	3050	44867	1.0	0.0	4.0	3	9	1968	
125	4389200955	3/2/2015	1450000.0	4	2.75	2750	17789	1.5	0.0	0.0	3	8	1914	
216	46100204	2/21/2015	1510000.0	5	3.00	3300	33474	1.0	0.0	3.0	3	9	1957	
246	2025069065	9/29/2014	2400000.0	4	2.50	3650	8354	1.0	1.0	4.0	3	9	2000	
282	7424700045	5/13/2015	2050000.0	5	3.00	3830	8480	2.0	0.0	1.0	5	9	1905	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
21498	3262300818	2/27/2015	1870000.0	4	3.75	3790	8797	2.0	0.0	0.0	3	11	2006	
21524	715010530	1/13/2015	1880000.0	5	3.50	4410	13000	2.0	0.0	3.0	3	10	2014	
21552	524059330	1/30/2015	1700000.0	4	3.50	3830	8963	2.0	0.0	0.0	3	10	2014	
21581	191100405	4/21/2015	1580000.0	4	3.25	3410	10125	2.0	0.0	0.0	3	10	2007	

Those entries below the "lower_limit" and above the "upper_limit".

In [85]:
```
df_new_1 = df_out_rmv_all[(df_out_rmv_all.price<upper_limit) & (df_out_rmv_all.price>lower_limit)]
df_new_1
```

Out[85]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renov
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	3	7	1955	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	3	7	1951	1
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	3	6	1933	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	5	7	1965	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	3	8	1987	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	0.0	0.0	3	8	2009	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	0.0	0.0	3	8	2014	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	0.0	0.0	3	7	2009	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	0.0	0.0	3	8	2004	

In [86]:
```
sns.histplot(df_new_1["price"])
```

Out[86]:
```
<AxesSubplot:xlabel='price', ylabel='Count'>
```

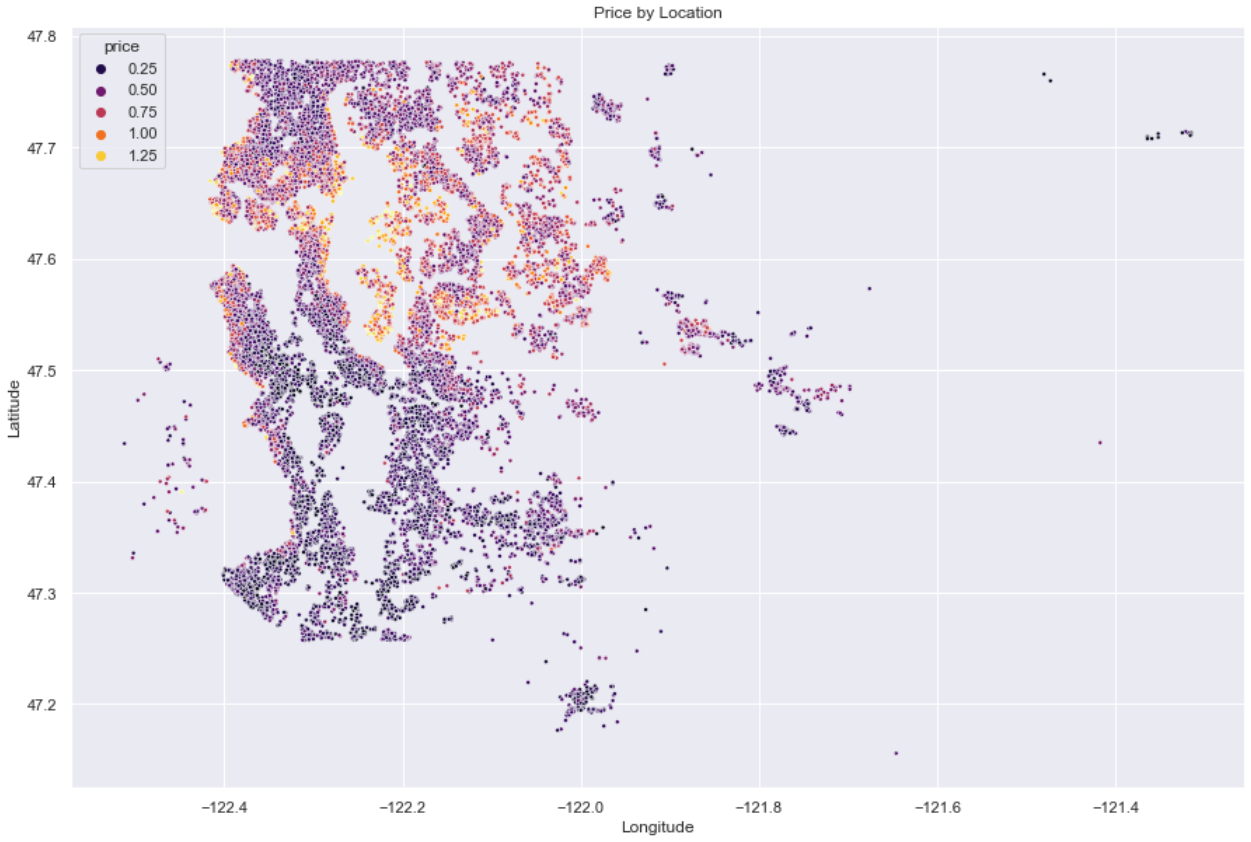
Much tighter distribution compared to before removing outliers.

1.3.0.10 New Price and Location

Even though, this won't be our final plot for price and location distribution, let's see the difference with outliers removed.

In [87]:
```
fig = plt.figure(figsize=(15,10))
ax = sns.scatterplot(x=df_new_1["long"], y=df_new_1["lat"], hue=df_new_1["price"], palette="inferno",
                     marker=".")
ax.set( xlabel="Longitude",
        ylabel="Latitude",
        title="Price by Location")
```

Out[87]:
```
[Text(0.5, 0, 'Longitude'),
 Text(0, 0.5, 'Latitude'),
 Text(0.5, 1.0, 'Price by Location')]
```



In [88]:
```
df_new_1["price"].describe()
```

Out[88]:
```
count    2.016900e+04
mean     4.907284e+05
std      2.358591e+05
min      7.800000e+04
25%      3.150000e+05
50%      4.399950e+05
75%      6.142850e+05
max      1.420000e+06
Name: price, dtype: float64
```

Much better distribution of prices and the plot gives us a better perspective regarding location relation to home price.

Let's move on to the outliers of those other features we left out before.

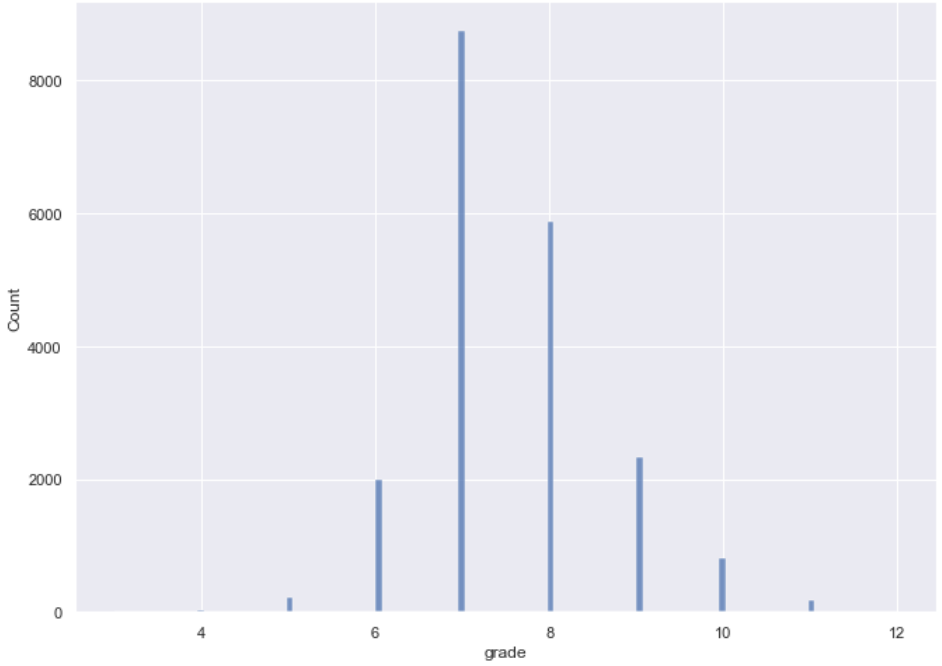
1.3.0.11 Grade

In [89]:
```
1 df_new_1["grade"].describe()
```

Out[89]:
```
count    20169.000000
mean         7.552184
std          1.052738
min          3.000000
25%          7.000000
50%          7.000000
75%          8.000000
max         12.000000
Name: grade, dtype: float64
```

In [90]:
```
1 sns.histplot(df_new_1["grade"])
```

Out[90]: <AxesSubplot:xlabel='grade', ylabel='Count'>



Let's set up the upper and lower limits.

In [91]:
```
1 upper_limit = df_new_1.grade.mean() + 3*df_new_1.grade.std()
2 upper_limit
```

Out[91]: 10.710397559750565

In [92]:
```
1 lower_limit = df_new_1.grade.mean() - 3*df_new_1.grade.std()
2 lower_limit
```

Out[92]: 4.393970529891955

So, these are the two boundaries that will help discern what is and what isn't an outlier.

In [93]:
```
df_new_1[(df_new_1.grade>upper_limit)|(df_new_1.grade<lower_limit)]
```

Out[93]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renov
313	4139480200	6/18/2014	1380000.0	4	3.25	4290	12103	1.0	0.0	3.0	3	11	1997	
314	4139480200	12/9/2014	1400000.0	4	3.25	4290	12103	1.0	0.0	3.0	3	11	1997	
350	7325600160	6/4/2014	299000.0	1	0.75	560	12120	1.0	0.0	0.0	3	4	1967	
431	2944010240	9/8/2014	988000.0	4	3.00	4040	19700	2.0	0.0	0.0	3	11	1987	
465	8658300340	5/23/2014	80000.0	1	0.75	430	5050	1.0	0.0	0.0	2	4	1912	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
20905	3864000120	4/8/2015	1180000.0	4	3.25	3780	10099	1.0	0.0	1.0	3	11	2006	
21018	8121100155	2/25/2015	810000.0	4	3.50	2700	2868	2.0	0.0	0.0	3	11	2006	
21134	1692900095	6/18/2014	1400000.0	4	2.75	3870	10046	2.0	0.0	0.0	3	11	2005	
21192	323059327	7/3/2014	1030000.0	4	3.50	4370	10860	2.0	0.0	0.0	3	11	2008	

Those entries below the "lower_limit" and above the "upper_limit".

In [94]:
```
df_new_2 = df_new_1[(df_new_1.grade<upper_limit) & (df_new_1.grade>lower_limit)]
df_new_2
```

Out[94]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renov
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	3	7	1955	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	3	7	1951	1
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	3	6	1933	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	5	7	1965	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	3	8	1987	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	0.0	0.0	3	8	2009	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	0.0	0.0	3	8	2014	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	0.0	0.0	3	7	2009	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	0.0	0.0	3	8	2004	

In [95]:
```
sns.histplot(df_new_2["grade"])
```

Out[95]:
```
<AxesSubplot:xlabel='grade', ylabel='Count'>
```

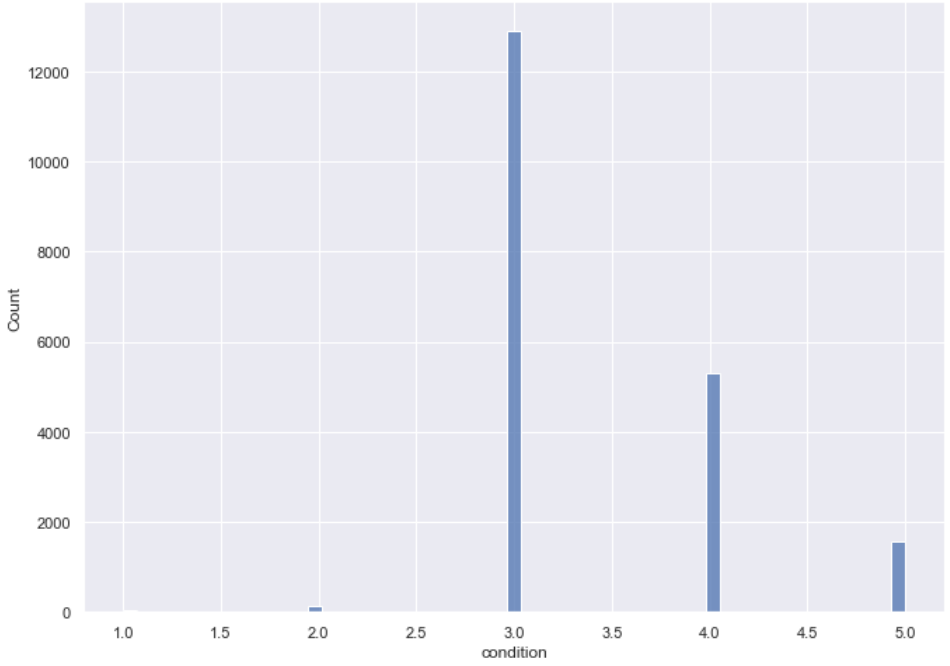
1.3.0.12 Condition

In [96]:
```
1 df_new_2["condition"].describe()
```

Out[96]:
```
count    19964.000000
mean         3.414496
std          0.651027
min          1.000000
25%          3.000000
50%          3.000000
75%          4.000000
max          5.000000
Name: condition, dtype: float64
```

In [97]:
```
1 sns.histplot(df_new_2["condition"])
```

Out[97]:
```
<AxesSubplot:xlabel='condition', ylabel='Count'>
```



Let's set up the upper and lower limits.

In [98]:
```
1 upper_limit = df_new_2.condition.mean() + 3*df_new_2.condition.std()
2 upper_limit
```

Out[98]:
```
5.367577516987661
```

In [99]:
```
1 lower_limit = df_new_2.condition.mean() - 3*df_new_2.condition.std()
2 lower_limit
```

Out[99]:
```
1.4614146689470209
```

So, these are the two boundaries that will help discern what is and what isn't an outlier.

In [100]:
```
df_new_2[(df_new_2.condition>upper_limit)|(df_new_2.condition<lower_limit)]
```

Out[100]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_reno
36	9435300030	5/28/2014	550000.0	4	1.00	1660	34848	1.0	0.0	0.0	1	5	1933	
397	5175800060	6/23/2014	365000.0	4	2.00	1940	25600	1.0	0.0	0.0	1	8	1962	
1440	5694000710	11/7/2014	352950.0	3	1.00	1760	3000	1.5	0.0	0.0	1	6	1900	
1732	913000340	1/2/2015	252000.0	1	1.00	680	1638	1.0	0.0	4.0	1	6	1910	1
2221	3886902445	3/16/2015	535000.0	2	1.00	920	9000	1.0	0.0	0.0	1	6	1954	
3002	9187200245	12/31/2014	441000.0	4	1.50	1100	3300	1.0	0.0	0.0	1	7	1919	
3199	723049596	5/9/2014	255000.0	2	1.00	810	7980	1.0	0.0	0.0	1	6	1928	
3971	6324000090	5/11/2015	210000.0	2	1.00	990	8140	1.0	0.0	0.0	1	6	1910	
4647	859000110	10/2/2014	125000.0	1	1.00	500	7440	1.0	0.0	0.0	1	5	1928	
7369	9275200080	11/7/2014	295000.0	3	1.50	720	7450	1.0	0.0	1.0	1	5	1924	

Those entries below the "lower_limit" and above the "upper_limit".

In [101]:
```
df_new_3 = df_new_2[(df_new_2.condition<upper_limit) & (df_new_2.condition>lower_limit)]
df_new_3
```

Out[101]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_reno
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	3	7	1955	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	3	7	1951	1
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	3	6	1933	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	5	7	1965	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	3	8	1987	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	0.0	0.0	3	8	2009	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	0.0	0.0	3	8	2014	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	0.0	0.0	3	7	2009	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	0.0	0.0	3	8	2004	

In [102]:
```
sns.histplot(df_new_3["condition"])
```

Out[102]:
```
<AxesSubplot:xlabel='condition', ylabel='Count'>
```

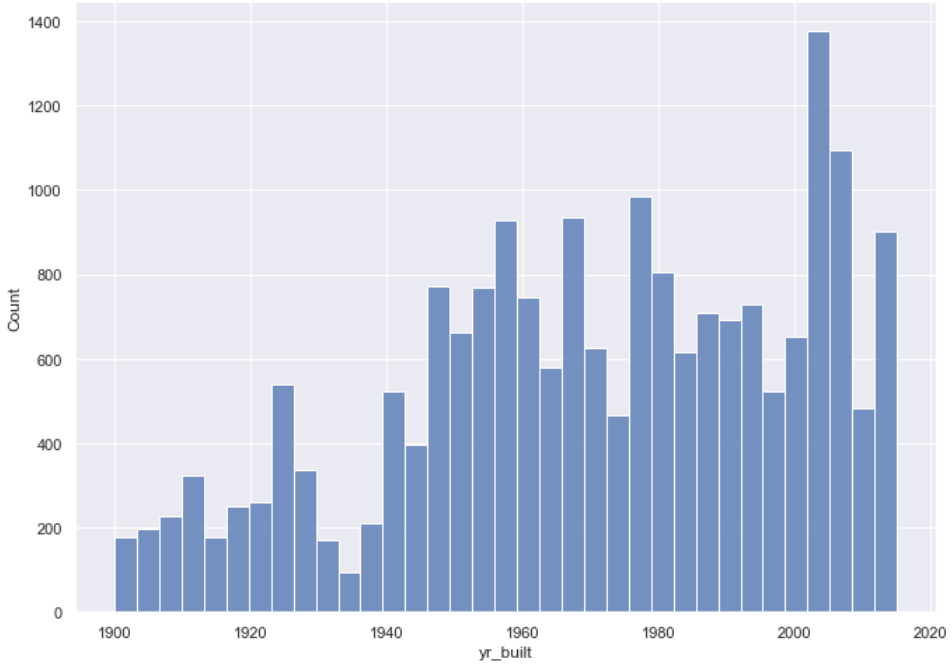
1.3.0.13 Yr_Built

In [103]:
```
1 df_new_3["yr_built"].describe()
```

Out[103]:
```
count    19938.000000
mean      1970.347878
std         29.373279
min       1900.000000
25%       1951.000000
50%       1973.000000
75%       1996.000000
max       2015.000000
Name: yr_built, dtype: float64
```

In [104]:
```
1 sns.histplot(df_new_3["yr_built"])
```

Out[104]: <AxesSubplot:xlabel='yr_built', ylabel='Count'>



Let's set up the upper and lower limits.

In [105]:
```
1 upper_limit = df_new_3.yr_built.mean() + 3*df_new_3.yr_built.std()
2 upper_limit
```

Out[105]: 2058.467716628846

In [106]:
```
1 lower_limit = df_new_3.yr_built.mean() - 3*df_new_3.yr_built.std()
2 lower_limit
```

Out[106]: 1882.2280402173774

So, these are the two boundaries that will help discern what is and what isn't an outlier.

In [107]:
```
1 df_new_3[(df_new_3.yr_built>upper_limit)|(df_new_3.yr_built<lower_limit)]
```

Out[107]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renovated	zipcode	lat	long	s

No outliers to remove for yr_built.

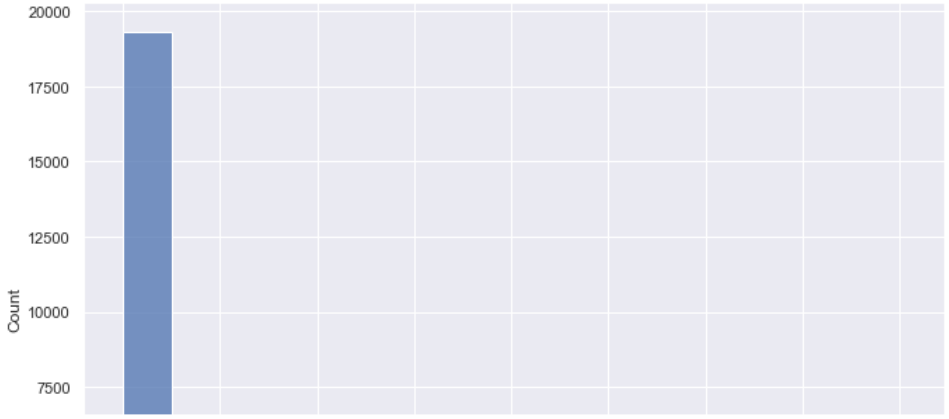
1.3.0.14 Yr_renovated

In [108]:
```
1 df_new_3["yr_renovated"].describe()
```

Out[108]:
```
count    19938.000000
mean        63.964590
std        351.546113
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max       2015.000000
Name: yr_renovated, dtype: float64
```

In [109]:
```
1 sns.histplot(df_new_3["yr_renovated"])
```

Out[109]:
```
<AxesSubplot:xlabel='yr_renovated', ylabel='Count'>
```



This feature "yr_renovated" doesn't seem to have much of a distribution. We will see if it has a correlation later on with price.

1.3.0.15 Floors

In [110]:
```
1 df_new_3["floors"].describe()
```

Out[110]:
```
count    19938.000000
mean         1.476377
std          0.538849
min          1.000000
25%          1.000000
50%          1.000000
75%          2.000000
max          3.500000
Name: floors, dtype: float64
```

In [111]:
```
1 sns.histplot(df_new_3["floors"])
```

Out[111]:
```
<AxesSubplot:xlabel='floors', ylabel='Count'>
```

Let's set up the upper and lower limits.

In [112]:
```
upper_limit = df_new_3.floors.mean() + 3*df_new_3.floors.std()
upper_limit
```

Out[112]: 3.092923925913987

In [113]:
```
lower_limit = df_new_3.floors.mean() - 3*df_new_3.floors.std()
lower_limit
```

Out[113]: -0.14017038995250664

So, these are the two boundaries that will help discern what is and what isn't an outlier.

In [114]:
```
df_new_3[(df_new_3.floors>upper_limit)|(df_new_3.floors<lower_limit)]
```

Out[114]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renovated
10066	1972202010	8/1/2014	435000.0	3	3.00	1440	1350	3.5	0.0	2.0	3	8	2005	0.0
11582	3180100023	1/30/2015	544000.0	3	2.50	1760	1755	3.5	0.0	0.0	3	8	1998	0.0
14871	8673400177	4/2/2015	525000.0	3	3.00	1730	1074	3.5	0.0	0.0	3	8	2006	0.0
15410	1702900664	4/16/2015	479000.0	2	2.50	1730	1037	3.5	0.0	0.0	3	8	2008	0.0
20292	1972200426	9/18/2014	525000.0	2	2.75	1310	1268	3.5	0.0	0.0	3	8	2007	0.0
20756	1972200428	6/25/2014	563500.0	3	2.50	1400	1312	3.5	0.0	0.0	3	8	2007	0.0

Those entries below the "lower_limit" and above the "upper_limit".

In [115]:
```
df_new_4 = df_new_3[(df_new_3.floors<upper_limit) & (df_new_3.floors>lower_limit)]
df_new_4
```

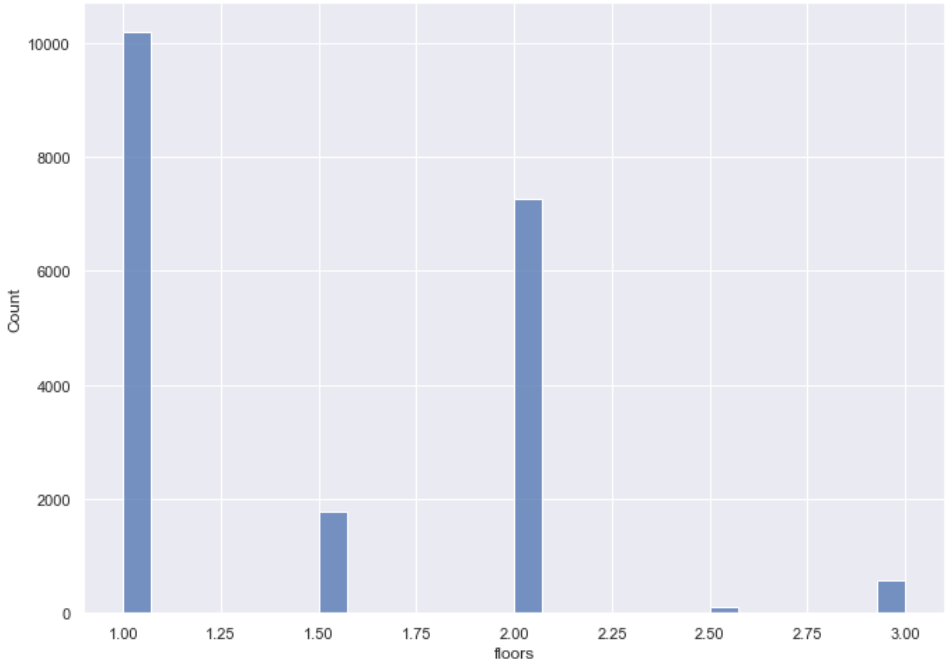
Out[115]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renovated
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	3	7	1955	0.0
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	3	7	1951	1991.0
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	3	6	1933	0.0
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	5	7	1965	0.0
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	3	8	1987	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	0.0	0.0	3	8	2009	0.0
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	0.0	0.0	3	8	2014	0.0
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	0.0	0.0	3	7	2009	0.0
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	0.0	0.0	3	8	2004	0.0
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	2.0	0.0	0.0	3	7	2008	0.0

19932 rows × 19 columns

In [116]:
```
1 sns.histplot(df_new_4["floors"])
```

Out[116]: <AxesSubplot:xlabel='floors', ylabel='Count'>



Doesn't help the distribution all that much but the real question is if there is a correlation between the number of floors and the resulting price of the home.

Now outliers have been removed for all the features, we need need to test the features for their correlation to price.

1.4 Exploratory Analysis - Feature selection

1.4.0.1 Correlation with Target Method

Simplifying the dataset means removing columns that might not be relevant for our current analysis.

If we are trying to investigate factors that affect the price and value of a home:

1. So, "id" and "date" are irrelevant because they aren't really features pertaining to the house, not components of the house. REMOVED "id" and "date"

So that leaves us with 18 other features (18 other columns excluding "price") to account for in our model.

To start, we can look at correlation between pairs of features to try to get an idea from there, what features may the most helpful in our model.

In [117]:
```
1 df_new_4.corr()
```

Out[117]:

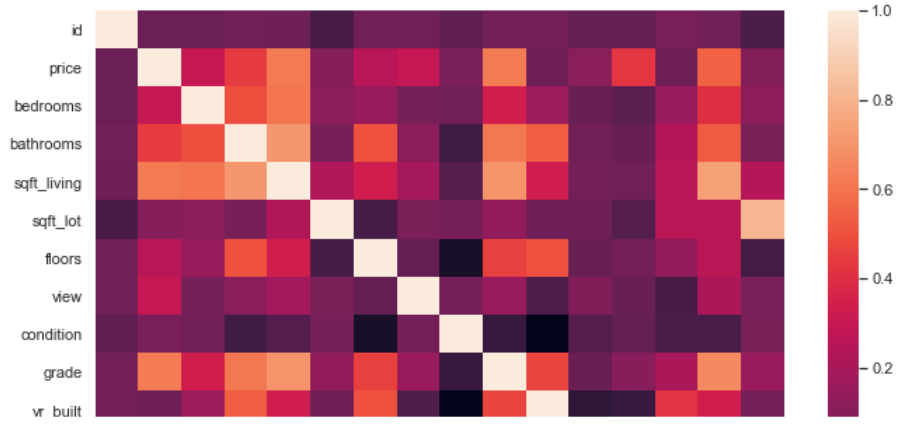
	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	view	condition	grade	yr_built	yr_renovated
id	1.000000	0.008546	0.007668	0.032771	0.022789	-0.108946	0.029209	0.025686	-0.028197	0.038874	0.036163	-0.011278
price	0.008546	1.000000	0.299355	0.442695	0.617115	0.092687	0.255178	0.294662	0.057837	0.622512	0.015045	0.107696
bedrooms	0.007668	0.299355	1.000000	0.493181	0.609691	0.113049	0.153688	0.045674	0.027243	0.334615	0.165173	0.002778
bathrooms	0.032771	0.442695	0.493181	1.000000	0.708522	0.053853	0.505027	0.104555	-0.135713	0.614984	0.538506	0.030256
sqft_living	0.022789	0.617115	0.609691	0.708522	1.000000	0.224953	0.334687	0.187479	-0.058389	0.698566	0.331293	0.034945
sqft_lot	-0.108946	0.092687	0.113049	0.053853	0.224953	1.000000	-0.115064	0.061530	0.048017	0.131796	0.022525	0.016371
floors	0.029209	0.255178	0.153688	0.505027	0.334687	-0.115064	1.000000	-0.010215	-0.277547	0.456204	0.501542	-0.000482
view	0.025686	0.294662	0.045674	0.104555	0.187479	0.061530	-0.010215	1.000000	0.041906	0.159609	-0.081364	0.080798
condition	-0.028197	0.057837	0.027243	-0.135713	-0.058389	0.048017	-0.277547	0.041906	1.000000	-0.166376	-0.366383	-0.057734
grade	0.038874	0.622512	0.334615	0.614984	0.698566	0.131796	0.456204	0.159609	-0.166376	1.000000	0.470043	0.002258
yr_built	0.036163	0.015045	0.165173	0.538506	0.331293	0.022525	0.501542	-0.081364	-0.366383	0.470043	1.000000	-0.195863
yr_renovated	-0.011278	0.107696	0.002778	0.030256	0.034945	0.016371	-0.000482	0.080798	-0.057734	0.002258	-0.195863	1.000000
lat	-0.009411	0.426256	-0.036311	-0.002378	0.029485	-0.066650	0.035458	0.004233	-0.009502	0.101315	-0.160237	0.027000
long	0.057251	0.018787	0.153553	0.239947	0.256045	0.258950	0.133510	-0.107983	-0.098410	0.206086	0.414804	-0.065530
sqft_living15	0.024220	0.551444	0.398843	0.527815	0.738169	0.262659	0.258855	0.210118	-0.101670	0.663991	0.336076	-0.023637
sqft_lot15	-0.097399	0.091546	0.123828	0.061024	0.240145	0.809516	-0.124425	0.055403	0.061176	0.161588	0.047564	0.012266

So even with correlation values with each pair of variables, it's still a lot of information to digest. What are we looking for? : High absolute values in the "price" row/column

To make it easier for us to discern, let's use a heatmap.

In [118]:
```
1 sns.heatmap(df_new_4.corr())
```

Out[118]: <AxesSubplot:>

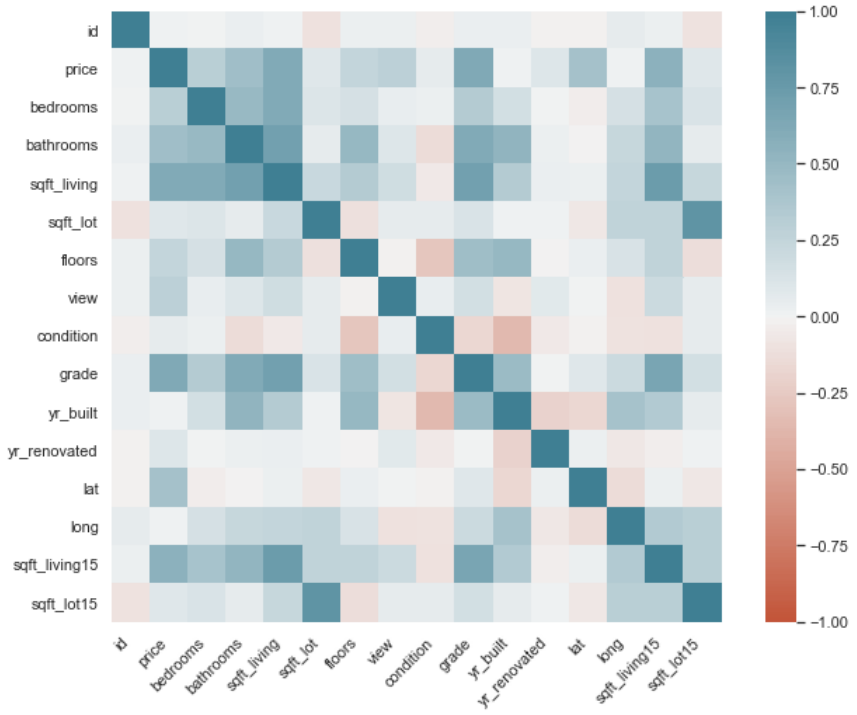


Still a bit cluttered. (The lighter colors along the "price" row/column, represent the positive correlations between price and another variable).

Isolating the "price" column can give us a better visual to draw conclusions from.

In [119]:

```
corr = df_new_4.corr()
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);
```



In [120]:
```
price_corrs = df_new_4.corr()["price"].map(abs).sort_values(ascending=False)
price_corrs
```

Out[120]:
```
price            1.000000
grade            0.622512
sqft_living      0.617115
sqft_living15    0.551444
bathrooms        0.442695
lat              0.426256
bedrooms         0.299355
view             0.294662
floors           0.255178
yr_renovated     0.107696
sqft_lot         0.092687
sqft_lot15       0.091546
condition        0.057837
long             0.018787
yr_built         0.015045
id               0.008546
Name: price, dtype: float64
```

The highest correlation was "sqft_living" (sqft footage of living space of home) to "price"; and "grade" (quality of the house) to "price".

1.4.0.2 Final_Location__price_map

Also, let's look at the distribution of prices based on location, now that we've cleaned all our features.

In [121]:
```
fig = plt.figure(figsize=(15,10))
ax = sns.scatterplot(x=df_new_4["long"], y=df_new_4["lat"], hue=df_new_4["price"], palette="inferno",
                     marker=".")
ax.set( xlabel="Longitude",
        ylabel="Latitude",
        title="Price by Location")
```

Out[121]:
```
[Text(0.5, 0, 'Longitude'),
 Text(0, 0.5, 'Latitude'),
 Text(0.5, 1.0, 'Price by Location')]
```

So this scatter plot reflects the results of our heatmap for correlation to "price". Latitude is more relevant to price as higher priced homes are above 47.5° N. There is strict contrast of home prices above and below 47.5° N. Though as shown in the heatmap for correlation, there is a very weak correlation between longitude and price.

1.5 Preprocessing

In [122]:
```
df_clean = df_new_4.copy()
df_clean.head()
```

Out[122]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renovated	zi
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	3	7	1955	0.0	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	3	7	1951	1991.0	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	3	6	1933	0.0	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	5	7	1965	0.0	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	3	8	1987	0.0	

1.5.0.1 Encoding the categorical data

This includes "zipcode" and "waterfront".

In [123]:
```
df_clean["waterfront"] = df_clean["waterfront"].str.replace("0.0", "Absent").str.replace("1.0", "Present")
df_clean.head()
```

```
<ipython-input-123-229aa746fac1>:1: FutureWarning: The default value of regex will change from True to False in a future
version.
  df_clean["waterfront"] = df_clean["waterfront"].str.replace("0.0", "Absent").str.replace("1.0", "Present")
```

Out[123]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	yr_built	yr_renovated	zi
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	Absent	0.0	3	7	1955	0.0	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	Absent	0.0	3	7	1951	1991.0	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	Absent	0.0	3	6	1933	0.0	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	Absent	0.0	5	7	1965	0.0	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	Absent	0.0	3	8	1987	0.0	

Get dummies should be able to extend out every feature out into it's values for different columns.

In [124]:
```
df_prep = pd.get_dummies(df_clean, columns=["zipcode", "waterfront"])
df_prep.head()
```

Out[124]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	view	condition	...	zipcode_98155	zipcode_98166	zipcode_9
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0.0	3	...	0	0	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	3	...	0	0	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	3	...	0	0	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	5	...	0	0	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	3	...	0	0	

5 rows × 89 columns

So now, that has extended the "zipcode" and "waterfront" values to become their own features so to speak.

In [125]:
```
df_prep.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 19932 entries, 0 to 21596
Data columns (total 89 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  19932 non-null  int64
 1   date                19932 non-null  object
 2   price               19932 non-null  float64
 3   bedrooms            19932 non-null  int64
 4   bathrooms           19932 non-null  float64
 5   sqft_living         19932 non-null  int64
 6   sqft_lot            19932 non-null  int64
 7   floors              19932 non-null  float64
 8   view                19932 non-null  float64
 9   condition           19932 non-null  int64
 10  grade               19932 non-null  int64
 11  yr_built            19932 non-null  int64
 12  yr_renovated        19932 non-null  float64
 13  lat                 19932 non-null  float64
```

Now to drop "id" and "date" so, we may use this data in our model.

Data used by the models must be numerical in nature, floats or integers. However, "date" values are objects and since we "date" and "id" features don't correlate to "price", we can definitely drop them.

In [126]:
```
df_prep = df_prep.drop(columns=["id", "date"])
df_prep.head()
```

Out[126]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	view	condition	grade	yr_built	...	zipcode_98155	zipcode_98166	zipcode_98168	zi
0	221900.0	3	1.00	1180	5650	1.0	0.0	3	7	1955	...	0	0	0	
1	538000.0	3	2.25	2570	7242	2.0	0.0	3	7	1951	...	0	0	0	
2	180000.0	2	1.00	770	10000	1.0	0.0	3	6	1933	...	0	0	0	
3	604000.0	4	3.00	1960	5000	1.0	0.0	5	7	1965	...	0	0	0	
4	510000.0	3	2.00	1680	8080	1.0	0.0	3	8	1987	...	0	0	0	

5 rows × 87 columns

1.6 Model Building

1.6.0.1 Model_1

This model will borrow the dataset from model_3 and will drop "sqft_above" and "sqft_basement"

In [127]:
```
df_model_1 = df_prep.copy()
df_model_1.head()
```

Out[127]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	view	condition	grade	yr_built	...	zipcode_98155	zipcode_98166	zipcode_98168	zi
0	221900.0	3	1.00	1180	5650	1.0	0.0	3	7	1955	...	0	0	0	
1	538000.0	3	2.25	2570	7242	2.0	0.0	3	7	1951	...	0	0	0	
2	180000.0	2	1.00	770	10000	1.0	0.0	3	6	1933	...	0	0	0	
3	604000.0	4	3.00	1960	5000	1.0	0.0	5	7	1965	...	0	0	0	
4	510000.0	3	2.00	1680	8080	1.0	0.0	3	8	1987	...	0	0	0	

5 rows × 87 columns

In [128]:
```
X1 = df_model_1.drop("price", axis=1)
Y1 = df_model_1["price"]

X1_scaled = (X1 - np.mean(X1))/ np.std(X1)
X1_train, X1_test, Y1_train, Y1_test = train_test_split(X1, Y1, test_size=0.2, random_state=11)
```

In [129]:
```
preds_1 = sm.add_constant(X1_scaled)
model_1 = sm.OLS(Y1, preds_1).fit()
display(model_1.summary())
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.830
Model:	OLS	**Adj. R-squared:**	0.829
Method:	Least Squares	**F-statistic:**	1153.
Date:	Wed, 12 May 2021	**Prob (F-statistic):**	0.00
Time:	17:24:19	**Log-Likelihood:**	-2.5671e+05
No. Observations:	19932	**AIC:**	5.136e+05
Df Residuals:	19847	**BIC:**	5.143e+05
Df Model:	84		
Covariance Type:	nonrobust		

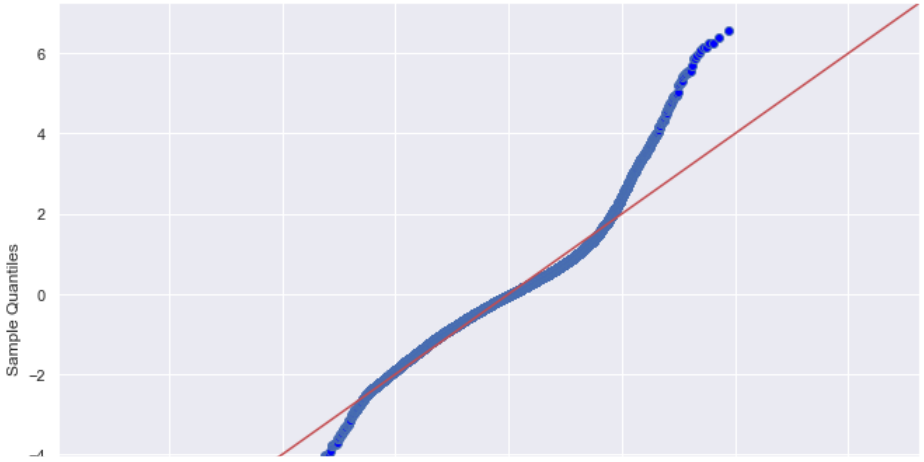
	coef	std err	t	P>|t|	[0.025	0.975]
const	4.864e+05	673.501	722.188	0.000	4.85e+05	4.88e+05
bedrooms	-3804.1149	897.554	-4.238	0.000	-5563.395	-2044.835
bathrooms	7725.6972	1162.739	6.644	0.000	5446.632	1e+04
sqft_living	7.837e+04	1425.698	54.969	0.000	7.56e+04	8.12e+04
sqft_lot	1.121e+04	1159.443	9.666	0.000	8934.388	1.35e+04
floors	1278.7071	931.065	1.373	0.170	-546.259	3103.673
view	2.559e+04	769.895	33.233	0.000	2.41e+04	2.71e+04
condition	1.481e+04	773.077	19.160	0.000	1.33e+04	1.63e+04
grade	5.143e+04	1178.121	43.653	0.000	4.91e+04	5.37e+04
yr_built	-1.827e+04	1179.291	-15.495	0.000	-2.06e+04	-1.6e+04
yr_renovated	6975.3677	715.175	9.753	0.000	5573.565	8377.171
lat	8990.9580	5697.745	1.578	0.115	-2177.098	2.02e+04
long	-6495.5171	4153.814	-1.564	0.118	-1.46e+04	1646.305
sqft_living15	2.223e+04	1179.209	18.848	0.000	1.99e+04	2.45e+04
sqft_lot15	-4727.5577	1244.262	-3.799	0.000	-7166.414	-2288.701
zipcode_98001	-2.121e+04	1541.193	-13.764	0.000	-2.42e+04	-1.82e+04
zipcode_98002	-1.359e+04	1214.714	-11.188	0.000	-1.6e+04	-1.12e+04
zipcode_98003	-1.962e+04	1399.406	-14.023	0.000	-2.24e+04	-1.69e+04
zipcode_98004	3.752e+04	709.568	52.872	0.000	3.61e+04	3.89e+04
zipcode_98005	1.3e+04	713.784	18.218	0.000	1.16e+04	1.44e+04
zipcode_98006	1.315e+04	740.671	17.757	0.000	1.17e+04	1.46e+04
zipcode_98007	6412.3499	722.447	8.876	0.000	4996.293	7828.406
zipcode_98008	7678.2840	811.527	9.462	0.000	6087.623	9268.944
zipcode_98010	-3703.8774	950.263	-3.898	0.000	-5566.473	-1841.282
zipcode_98011	-4814.6893	1032.418	-4.664	0.000	-6838.315	-2791.064
zipcode_98014	-2365.0360	1042.590	-2.268	0.023	-4408.599	-321.473
zipcode_98019	-6933.8509	1200.257	-5.777	0.000	-9286.455	-4581.247
zipcode_98022	-1.335e+04	1617.136	-8.253	0.000	-1.65e+04	-1.02e+04
zipcode_98023	-2.909e+04	1890.729	-15.387	0.000	-3.28e+04	-2.54e+04
zipcode_98024	-506.3409	801.423	-0.632	0.528	-2077.197	1064.515
zipcode_98027	3427.6615	941.012	3.643	0.000	1583.199	5272.124
zipcode_98028	-7059.2525	1150.211	-6.137	0.000	-9313.762	-4804.743
zipcode_98029	7691.3027	1050.010	7.325	0.000	5633.195	9749.411
zipcode_98030	-1.704e+04	1097.186	-15.528	0.000	-1.92e+04	-1.49e+04
zipcode_98031	-1.735e+04	994.964	-17.434	0.000	-1.93e+04	-1.54e+04
zipcode_98032	-1.341e+04	915.036	-14.655	0.000	-1.52e+04	-1.16e+04
zipcode_98033	2.182e+04	968.347	22.537	0.000	1.99e+04	2.37e+04
zipcode_98034	263.6771	1245.466	0.212	0.832	-2177.540	2704.895

zipcode_98038	-1.822e+04	1657.493	-10.992	0.000	-2.15e+04	-1.5e+04
zipcode_98039	1.451e+04	678.034	21.402	0.000	1.32e+04	1.58e+04
zipcode_98040	2.736e+04	680.392	40.212	0.000	2.6e+04	2.87e+04
zipcode_98042	-2.34e+04	1483.945	-15.770	0.000	-2.63e+04	-2.05e+04
zipcode_98045	-3459.3931	1498.780	-2.308	0.021	-6397.128	-521.659
zipcode_98052	1.19e+04	1175.185	10.123	0.000	9593.332	1.42e+04
zipcode_98053	9458.9374	1233.521	7.668	0.000	7041.134	1.19e+04
zipcode_98055	-1.439e+04	818.254	-17.591	0.000	-1.6e+04	-1.28e+04
zipcode_98056	-9214.5214	738.779	-12.473	0.000	-1.07e+04	-7766.453
zipcode_98058	-1.898e+04	981.493	-19.334	0.000	-2.09e+04	-1.71e+04
zipcode_98059	-9923.7348	845.165	-11.742	0.000	-1.16e+04	-8267.140
zipcode_98065	-3482.9387	1453.491	-2.396	0.017	-6331.903	-633.975
zipcode_98070	-6763.4737	891.348	-7.588	0.000	-8510.590	-5016.357
zipcode_98072	-1964.0575	1156.332	-1.699	0.089	-4230.564	302.449
zipcode_98074	5444.5940	1086.255	5.012	0.000	3315.444	7573.744
zipcode_98075	7784.5103	1028.290	7.570	0.000	5768.976	9800.045
zipcode_98077	-1580.8451	1054.454	-1.499	0.134	-3647.663	485.973
zipcode_98092	-2.14e+04	1435.057	-14.909	0.000	-2.42e+04	-1.86e+04
zipcode_98102	1.489e+04	734.137	20.280	0.000	1.34e+04	1.63e+04
zipcode_98103	2.122e+04	1202.384	17.652	0.000	1.89e+04	2.36e+04
zipcode_98105	2.206e+04	825.113	26.732	0.000	2.04e+04	2.37e+04
zipcode_98106	-7380.7226	865.478	-8.528	0.000	-9077.132	-5684.314
zipcode_98107	1.364e+04	982.547	13.881	0.000	1.17e+04	1.56e+04
zipcode_98108	-6469.3806	717.542	-9.016	0.000	-7875.824	-5062.938
zipcode_98109	1.601e+04	752.775	21.272	0.000	1.45e+04	1.75e+04
zipcode_98112	2.868e+04	770.994	37.196	0.000	2.72e+04	3.02e+04
zipcode_98115	2.214e+04	1142.162	19.380	0.000	1.99e+04	2.44e+04
zipcode_98116	1.328e+04	937.531	14.160	0.000	1.14e+04	1.51e+04
zipcode_98117	1.917e+04	1297.556	14.774	0.000	1.66e+04	2.17e+04
zipcode_98118	-2722.7624	727.020	-3.745	0.000	-4147.783	-1297.742
zipcode_98119	2.089e+04	835.125	25.020	0.000	1.93e+04	2.25e+04
zipcode_98122	1.422e+04	767.994	18.512	0.000	1.27e+04	1.57e+04
zipcode_98125	912.3803	1154.840	0.790	0.430	-1351.203	3175.964
zipcode_98126	682.4675	918.901	0.743	0.458	-1118.656	2483.591
zipcode_98133	-6383.4996	1387.220	-4.602	0.000	-9102.567	-3664.432
zipcode_98136	6996.8636	890.319	7.859	0.000	5251.764	8741.963
zipcode_98144	7286.0045	743.413	9.801	0.000	5828.852	8743.157
zipcode_98146	-7299.5175	890.945	-8.193	0.000	-9045.844	-5553.191
zipcode_98148	-5903.0440	747.827	-7.894	0.000	-7368.847	-4437.241
zipcode_98155	-7524.5875	1370.237	-5.491	0.000	-1.02e+04	-4838.809
zipcode_98166	-8729.6307	941.268	-9.274	0.000	-1.06e+04	-6884.668
zipcode_98168	-1.366e+04	818.975	-16.683	0.000	-1.53e+04	-1.21e+04
zipcode_98177	1972.5111	1140.512	1.729	0.084	-262.987	4208.009
zipcode_98178	-1.454e+04	737.289	-19.724	0.000	-1.6e+04	-1.31e+04
zipcode_98188	-1.111e+04	788.623	-14.089	0.000	-1.27e+04	-9564.989
zipcode_98198	-1.759e+04	1113.808	-15.797	0.000	-1.98e+04	-1.54e+04
zipcode_98199	2.115e+04	1007.996	20.980	0.000	1.92e+04	2.31e+04
waterfront_Absent	-6980.2564	362.344	-19.264	0.000	-7690.480	-6270.032
waterfront_Present	6980.2564	362.344	19.264	0.000	6270.032	7690.480

Omnibus:	3304.593	**Durbin-Watson:**	1.987
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	15178.642
Skew:	0.742	**Prob(JB):**	0.00
Kurtosis:	7.009	**Cond. No.**	6.57e+15

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.13e-27. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [130]:
```
fig = sm.graphics.qqplot(model_1.resid, dist=stats.norm, line="45", fit=True)
```



We see very abnormal residues as especially towards the end, the points are far away from the red line.

- High R-squared 0.83: meaning we can explain about 83% of the variance pertaining to "price" That's really good.

However:

- Extremely high condition number, meaning this model is ill-conditioned.
- We also have several features "floors", "lat", "long", and quite a few zipcodes where the p-value > than 0.05 Meaning not statistically significant and indicates strong evidence for the null hypothesis.

Metrics for Model_1

In [131]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
mae_1 = mean_absolute_error(Y1, model_1.predict(preds_1))
mae_1
```

Out[131]: 67616.70589827692

Accuracy of model, prediction off the mean by +/- 67617 USD

In [132]:
```
mae_1/df_clean.price.mean()
```

Out[132]: 0.13901629954261674

Margin of Error: 13.9%

In [133]:
```
msqu_1 = mean_squared_error(Y1, model_1.predict(preds_1))
msqu_1
```

Out[133]: 9002671426.651663

That's the mean squared error for model 1. We square root that to get the root mean squared error (RMSE).

In [134]:
```
rmse_1 = np.sqrt(msqu_1)
rmse_1
```

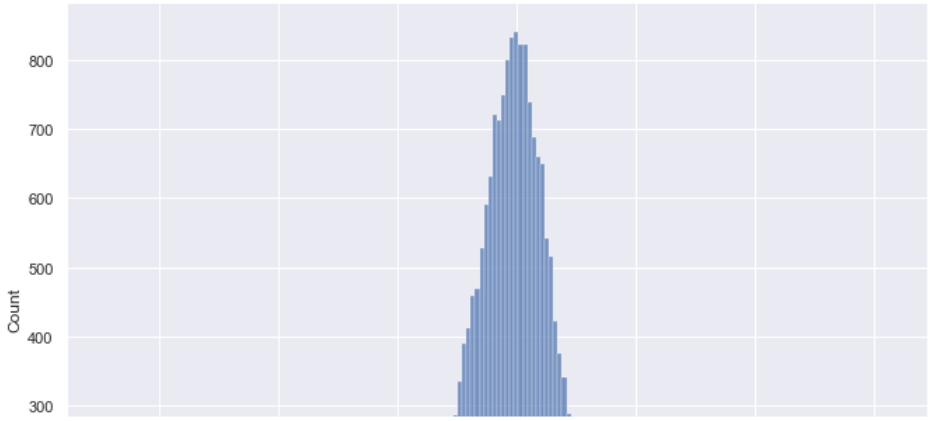
Out[134]: 94882.4084151096

In [135]:
```
print("Root Mean Square Error:", rmse_1)
print("Mean Absolute Error:", mae_1)
```

```
Root Mean Square Error: 94882.4084151096
Mean Absolute Error: 67616.70589827692
```

In [136]:
```
sns.histplot(model_1.resid)
```

Out[136]: <AxesSubplot:ylabel='Count'>



Histogram depicting model_1 distribution.

So, we've seen the p-values of model_1. There were quite a few that were over 0.05, not statistically significant.

Namely:

- floors : 0.170
- lat : 0.115
- long : 0.118
- zipcode_98024: 0.528
- zipcode_98034: 0.832
- zipcode_98072: 0.089
- zipcode_98077: 0.134
- zipcode_98125: 0.430
- zipcode_98126: 0.458
- zipcode_98177: 0.084

Since the model_1 displayed strong multicollinearity, we will try looking at VIF values to improve upon model_1 in model_2.

1.6.0.2 VIF

Let's try using vif (variance inflation factor) values of the features to help reduce multicollinearity in model_2.

In [137]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [138]:
```
vif = [variance_inflation_factor(df_model_1.values, i) for i in range(df_model_1.shape[1])]
X1_cols = list(df_model_1.columns)

list(zip(X1_cols, vif))
```

```
C:\Users\bigbenx3\anaconda3\envs\learn-env\lib\site-packages\statsmodels\stats\outliers_influence.py:193: RuntimeWarn
ing: divide by zero encountered in double_scalars
  vif = 1. / (1. - r_squared_i)
```

After looking at the VIF values, removing "lat": 71.58 and "long": 38.04, as well as the waterfront and zipcode features from the dataframe used for model_2 might help reduce multicollinearity. Other than the zipcode features and the waterfront features (inf), these two features have the largest VIF above 5.

1.6.0.3 Model_2

This model will borrow the dataset from model_1 and will drop those features with VIF values above 5.

In [139]:
```
df_model_2 = df_model_1[["sqft_lot", "sqft_living",
                    "grade", "condition", "bathrooms", "bedrooms",
                    "price", "floors", "view", "yr_built",
                         "yr_renovated", "sqft_living15", "sqft_lot15"]]
df_model_2.head()
```

Out[139]:

	sqft_lot	sqft_living	grade	condition	bathrooms	bedrooms	price	floors	view	yr_built	yr_renovated	sqft_living15	sqft_lot15
0	5650	1180	7	3	1.00	3	221900.0	1.0	0.0	1955	0.0	1340	5650
1	7242	2570	7	3	2.25	3	538000.0	2.0	0.0	1951	1991.0	1690	7639
2	10000	770	6	3	1.00	2	180000.0	1.0	0.0	1933	0.0	2720	8062
3	5000	1960	7	5	3.00	4	604000.0	1.0	0.0	1965	0.0	1360	5000
4	8080	1680	8	3	2.00	3	510000.0	1.0	0.0	1987	0.0	1800	7503

In [140]:
```
X2 = df_model_2.drop("price", axis=1)
Y2 = df_model_2["price"]

X2_scaled = (X2 - np.mean(X2))/ np.std(X2)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(X2, Y2, test_size=0.2, random_state=11)
```

In [141]:
```
preds_2 = sm.add_constant(X2_scaled)
model_2 = sm.OLS(Y2, preds_2).fit()
display(model_2.summary())
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.592
Model:	OLS	**Adj. R-squared:**	0.591
Method:	Least Squares	**F-statistic:**	2404.
Date:	Wed, 12 May 2021	**Prob (F-statistic):**	0.00
Time:	17:24:35	**Log-Likelihood:**	-2.6544e+05
No. Observations:	19932	**AIC:**	5.309e+05
Df Residuals:	19919	**BIC:**	5.310e+05
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.864e+05	1041.928	466.821	0.000	4.84e+05	4.88e+05
sqft_lot	1285.5573	1782.028	0.721	0.471	-2207.366	4778.481
sqft_living	6.15e+04	2170.382	28.334	0.000	5.72e+04	6.58e+04
grade	1.025e+05	1688.177	60.728	0.000	9.92e+04	1.06e+05
condition	1.476e+04	1154.145	12.789	0.000	1.25e+04	1.7e+04
bathrooms	2.228e+04	1775.205	12.549	0.000	1.88e+04	2.58e+04
bedrooms	-1.545e+04	1361.831	-11.347	0.000	-1.81e+04	-1.28e+04
floors	1.971e+04	1343.962	14.667	0.000	1.71e+04	2.23e+04
view	2.434e+04	1092.634	22.281	0.000	2.22e+04	2.65e+04
yr_built	-8.647e+04	1486.295	-58.181	0.000	-8.94e+04	-8.36e+04
yr_renovated	4690.7224	1096.502	4.278	0.000	2541.488	6839.957
sqft_living15	3.308e+04	1684.558	19.637	0.000	2.98e+04	3.64e+04
sqft_lot15	-1.669e+04	1820.107	-9.172	0.000	-2.03e+04	-1.31e+04

Omnibus:	2037.259	**Durbin-Watson:**	1.972
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	4320.966
Skew:	0.649	**Prob(JB):**	0.00
Kurtosis:	4.875	**Cond. No.**	4.98

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

In [142]:
```
fig = sm.graphics.qqplot(model_2.resid, dist=stats.norm, line="45", fit=True)
```

We see very abnormal residues as especially towards the end, the points are far away from the red line.

- Low R-squared 0.59: meaning we can explain about 59% of the variance pertaining to "price" Very poor
- Very low condition number, meaning this model is well-conditioned. It shows on the visual. Other than the front end, the plotted data points are close to the red line, indicating the overall normal residues.

Metrics for Model_2

In [143]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
mae_2 = mean_absolute_error(Y2, model_2.predict(preds_2))
mae_2
```

Out[143]: 111017.76682258763

Accuracy of model, prediction off the mean by +/- 111018 USD

In [144]:
```
mae_2/df_clean.price.mean()
```

Out[144]: 0.22824653940372605

Margin of Error: 22.8% Accuracy has suffered greatly.

In [145]:
```
msqu_2 = mean_squared_error(Y2, model_2.predict(preds_2))
msqu_2
```

Out[145]: 21624344003.605213

That's the mean squared error for model 1. We square root that to get the root mean squared error (RMSE).

In [146]:
```
rmse_2 = np.sqrt(msqu_2)
rmse_2
```

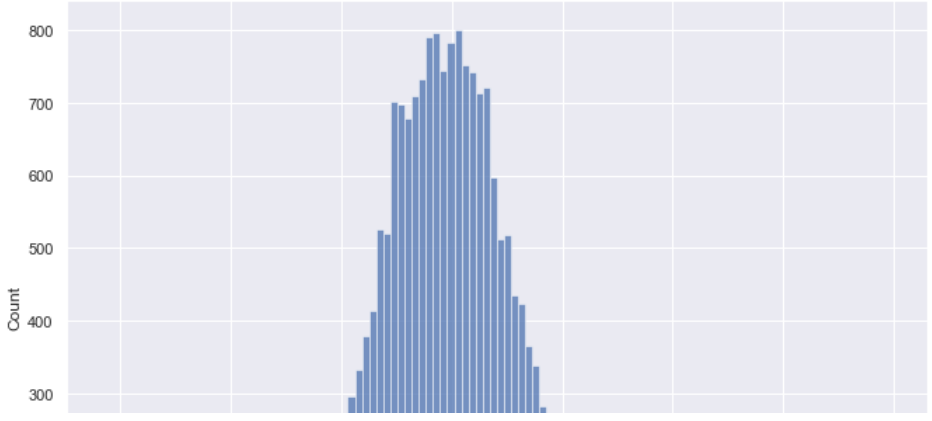
Out[146]: 147052.181226955

In [147]:
```
print("Root Mean Square Error:", rmse_2)
print("Mean Absolute Error:", mae_2)
```

```
Root Mean Square Error: 147052.181226955
Mean Absolute Error: 111017.76682258763
```

In [148]:
```
sns.histplot(model_2.resid)
```

Out[148]: <AxesSubplot:ylabel='Count'>



Histogram depicting model_2 distribution.

So, even though the qqplot depicts greater normal residues plotted, the accuracy of the model has suffered and this current model can only explain 59.2% of the variance surrounding price.

1.7 Which Model?

I think I'll go with model_1 because of the greater accuracy model_1 shows over model_2. Although, it's like picking the lesser of two evils since, the model's accuracy may or may not reflect predictability of future data.

1.8 Data Question - Answers

1. The factors most affecting the price of a house are:

- Location(lat)
- Quality of the house(grade)
- Living area(sqft_living)

1.9 Results

- We have a model that has an Coefficient of Determination(R-squared) value of 0.83 which indicates that our model can explain about 83% of all variation in the data around the mean.
- With a Mean Absolute Error of around 67616 USD, that means our predicted price is, on average, 67616 USD off from our mean. While that number doesn't look too bad our Root Mean Squared Error is around 94882 USD which means that our model is being heavily penalized for predictions that are very far off the actual price.
- Average home price: 486,394 USD. The price prediction was +/- 67616 off the real price (13.9% margin of error)

In [149]:
```
1 df_clean.price.mean()
```

Out[149]:
```
486394.0855910094
```

1.10 Conclusions

Descriptive analysis and modeling reveal which factors contribute most to housing prices:

● Increase Living Area(in square feet)

● Buy homes in regions specified (North of 47.55°N latitude are the more expensive homes) maybe homes outside of this region will likely be more affordable)

● Upgrade the quality of your home: This includes a large part curb appeal, fixing up the walkway, shrubbery surrounding the house, outdoor patio - making the house more aesthetic from the outside. Making your house more energy efficient. Fresh coats of paint could do a great deal like in the kitchen can alter the overall feel. (link below)

https://www.bankrate.com/loans/home-improvement/cheap-fixes-to-boost-the-value-of-your-home/ (https://www.bankrate.com/loans/home-improvement/cheap-fixes-to-boost-the-value-of-your-home/)

1.11 Future Research

- The data we were provided was from 2014 to 2015. And such outdated data may not give us the optimal insights relevant to today's housing situation
- We should be able to get a lot more out of the location data, with further analysis, incorporating data relevant to the zipcode so there is a better determination for prices that can be expected in a more defined area.
- Also, streamlining the methods of getting a more fitted model without going too far into "overfitted" territory. Like I've mentioned before, there is a happy medium in there.
- The most obvious next step is to try out new modeling techniques. While linear regression is a good start, there are many other techniques that I believe could help make better predictions. Of particular interest to me in this context are Polynomial Regression and Weighted Least Squares, that might be promising.

1.12 Presentation Prep

In [150]:
```
fig = plt.figure(figsize=(11,8))
ax = sns.regplot(data=df_clean, x="sqft_living", y="price", marker=".",
     scatter_kws={"color": "grey"}, line_kws={"color": "blue"})

ax.set(  xlabel="Living Area(square feet)",
         ylabel="Price(in Millions of $)",
         title="Price by Living Area",
 )


plt.xlim([0,5000])
plt.ylim([0, 1600000])
plt.show()
```

In [151]:
```
df_clean.grade.describe()
```

Out[151]:
```
count    19932.000000
mean         7.528096
std          0.994624
min          5.000000
25%          7.000000
50%          7.000000
75%          8.000000
max         10.000000
Name: grade, dtype: float64
```

In [152]:
```
fig = plt.figure(figsize=(15,20))
ax = sns.barplot(data=df_clean, x="grade", y="price", ci=None)
ax.set( xticklabels=(["5", "6", "7", "8", "9", "10"]),
        xlabel="Grade",
        ylabel="Price (in Thousands of $)",
        title="Price by Grade"  )
```

In [ ]:
```
1
```