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/)

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.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)})
```

The following is a function to download via pandas csv, excel, or json files to jupyter notebook:

In [2]:
```
def files_import_pd(file_path):
    if file_path.endswith("csv"):
        return pd.read_csv(file_path)
    if file_path.endswith("tsv"):
        return pd.read_csv(file_path, sep="\t")
    if file_path.endswith("xlsx"):
        return pd.read_excel(file_path)
    if file_path.endswith("json"):
        return pd.read_json(file_path)
    else:
        print("NOT CSV/TSV/EXCEL/JSON FILE")
```

In [3]:
```
df1 = files_import_pd(r"C:\Users\bigbenx3\2021_flatiron\flatiron_projects\housing_analysis_project\kc_house_data.csv")
df1.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basemen
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	(
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	40(
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	(
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	91(
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	(

5 rows × 21 columns

Good. Imported the first dataframe. Let's look at its contents. ***Objective: Checking for nulls.

In [4]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64  
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64  
 6   sqft_lot       21613 non-null  int64  
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64  
 9   view           21613 non-null  int64  
 10  condition      21613 non-null  int64  
 11  grade          21613 non-null  int64  
 12  sqft_above     21613 non-null  int64  
 13  sqft_basement  21613 non-null  int64  
 14  yr_built       21613 non-null  int64  
 15  yr_renovated   21613 non-null  int64  
 16  zipcode        21613 non-null  int64  
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64  
 20  sqft_lot15     21613 non-null  int64  
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

Also, there are datatypes for the values within the columns that we may want to change, for example "price".

In [5]:
```
df1.isnull().sum()
```

Out[5]:
```
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
```

So it appears there are no missing/ empty values.

1.3 Data - Manipulation

In [6]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

So, there are 21 columns, aka our features, and we don't need all of them.

First We're going to remove the columns for features we aren't accounting for. This is in the interest of time and simplicity of the model.

In [7]:
```
df1.head()
```

Out[7]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basemen
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	(
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	40(
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	(
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	91(
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	(

5 rows × 21 columns

Also, in other words, we'll only keep a select number of columns.

In [8]:
```
df = df1[["sqft_lot", "sqft_living",
                    "grade", "condition", "bathrooms", "bedrooms",
                    "waterfront", "price", "floors", "lat", "long"]]
```

We are eliminating the columns below:

yr_built

date

view

sqft_above

sqft_basement

yr_renovated

zipcode

lat

long

sqft_living15

sqft_lot15

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   sqft_lot     21613 non-null  int64  
 1   sqft_living  21613 non-null  int64  
 2   grade        21613 non-null  int64  
 3   condition    21613 non-null  int64  
 4   bathrooms    21613 non-null  float64
 5   bedrooms     21613 non-null  int64  
 6   waterfront   21613 non-null  int64  
 7   price        21613 non-null  float64
 8   floors       21613 non-null  float64
 9   lat          21613 non-null  float64
 10  long         21613 non-null  float64
dtypes: float64(5), int64(6)
memory usage: 1.8 MB
```

From 21 to 11 columns to account for.

1.4 Exploratory Analysis

We want to get a sense of the data, the values, for each feature and remove the outliers in preparation to building a model.

Before, that we want to change the datatypes for some the columns, for example "price".

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   sqft_lot     21613 non-null  int64  
 1   sqft_living  21613 non-null  int64  
 2   grade        21613 non-null  int64  
 3   condition    21613 non-null  int64  
 4   bathrooms    21613 non-null  float64
 5   bedrooms     21613 non-null  int64  
 6   waterfront   21613 non-null  int64  
 7   price        21613 non-null  float64
 8   floors       21613 non-null  float64
 9   lat          21613 non-null  float64
 10  long         21613 non-null  float64
dtypes: float64(5), int64(6)
memory usage: 1.8 MB
```

In [11]:
```
df["price"] = df["price"].astype(int)
```

```
<ipython-input-11-d7d05832fc73>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df["price"] = df["price"].astype(int)
```

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   sqft_lot     21613 non-null  int64  
 1   sqft_living  21613 non-null  int64  
 2   grade        21613 non-null  int64  
 3   condition    21613 non-null  int64  
 4   bathrooms    21613 non-null  float64
 5   bedrooms     21613 non-null  int64  
 6   waterfront   21613 non-null  int64  
 7   price        21613 non-null  int32  
 8   floors       21613 non-null  float64
 9   lat          21613 non-null  float64
 10  long         21613 non-null  float64
dtypes: float64(4), int32(1), int64(6)
memory usage: 1.7 MB
```

We may need to change the other features into another datatype. For now, this will do.

1.4.0.1 Prices Overview

The dependent variable here is price of the homes. Let's get a sense of the prices.

In [13]:
```
df.price.describe()
```

Out[13]:
```
count    2.161300e+04
mean     5.400881e+05
std      3.671272e+05
min      7.500000e+04
25%      3.219500e+05
50%      4.500000e+05
75%      6.450000e+05
max      7.700000e+06
Name: price, dtype: float64
```

count 21,600

mean 540,000

std 367,000

min 75,000

25% 321,900

50% 450,000

75% 645000

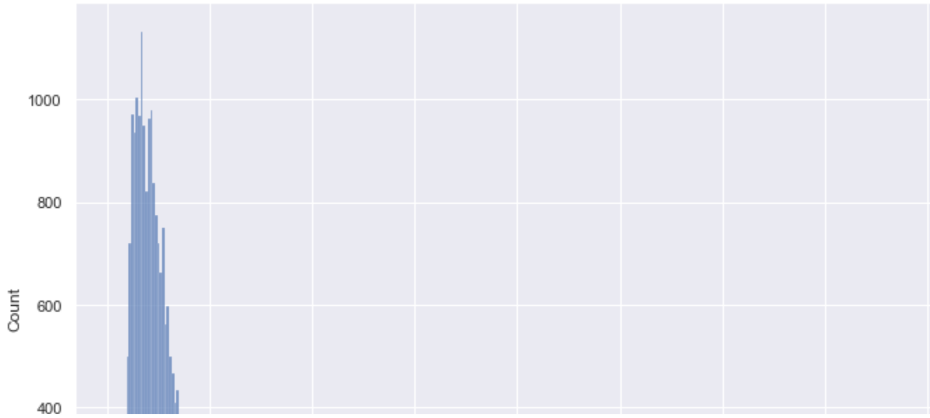
max 7,700,000

(USD) 2014-2015 King County, Washington 98001

It's easier to see now the corresponding numerical values.

In [14]:
```
sns.histplot(df.price)
```

Out[14]: <AxesSubplot:xlabel='price', ylabel='Count'>



https://www.thoughtco.com/what-is-the-interquartile-range-rule-3126244 (https://www.thoughtco.com/what-is-the-interquartile-range-rule-3126244)

In [15]:
```
q3, q1 = np.percentile(df["price"], [75 ,25])
iqr = q3 - q1
iqr
```

Out[15]: 323050.0

323050 is the interquartile range.

In [16]:
```
q3
```

Out[16]: 645000.0

Oh, ok- the 75percentile.

In [17]:
```
q1
```

Out[17]: 321950.0

And the 25percentile.

In [18]:
```
323050*1.5
```

Out[18]: 484575.0

This number will allow us to find the range that are outliers.

Though it's not often affected much by them, the interquartile range can be used to detect outliers. This is done using these steps:

Calculate the interquartile range for the data.

Multiply the interquartile range (IQR) by 1.5 (a constant used to discern outliers).

Add 1.5 x (IQR) to the third quartile. Any number greater than this is a suspected outlier.

Subtract 1.5 x (IQR) from the first quartile. Any number less than this is a suspected outlier.

https://www.thoughtco.com/what-is-the-interquartile-range-rule-3126244 (https://www.thoughtco.com/what-is-the-interquartile-range-rule-3126244)

In [19]:
```
645000+484575.0
```

Out[19]: 1129575.0

In [20]:
```
321950-484575.0
```

Out[20]: -162625.0

So regarding "price" values, any home price < -162625 and > 1129575 are outliers. And since we don't deal with negative numbers with price, we'll ignore the < -162625 part.

In [21]:
```
import pandas as pd
df_price_unique_values = df["price"].unique()
print(sorted(df_price_unique_values))
```

```
[75000, 78000, 80000, 81000, 82000, 82500, 83000, 84000, 85000, 86500, 89000, 89950, 90000, 92000, 95000, 96500, 99000,
100000, 102500, 104950, 105000, 105500, 106000, 107000, 109000, 109500, 110000, 110700, 111300, 112000, 114000, 114975,
115000, 118000, 118125, 119500, 119900, 120000, 120750, 121800, 122000, 123000, 123300, 124000, 124500, 124740, 125000,
126000, 126500, 128000, 128750, 129000, 129888, 130000, 132500, 132825, 133000, 133400, 134000, 135000, 135900, 136500,
137000, 137124, 137900, 139000, 139500, 139950, 140000, 141800, 142000, 142500, 143000, 144000, 144975, 145000, 145600,
146000, 146300, 147000, 147200, 147400, 147500, 148000, 148226, 148900, 149000, 149500, 149900, 150000, 150550, 151000,
151100, 151600, 152000, 152275, 152500, 152900, 153000, 153500, 153503, 154000, 154200, 154500, 154950, 155000, 156000,
156601, 157000, 157340, 157500, 158000, 158550, 158800, 159000, 159075, 159100, 159995, 160000, 160134, 160797, 161000,
161500, 161700, 162000, 162248, 162500, 162950, 163000, 163250, 163500, 163800, 164000, 164808, 164950, 165000, 165050,
166000, 166600, 166950, 167000, 167500, 168000, 168500, 169000, 169100, 169317, 169500, 169575, 169900, 169950, 170000,
170500, 171000, 171500, 171800, 172000, 172040, 172380, 172500, 173000, 173250, 174000, 174500, 174900, 174950, 175000,
175003, 175409, 176000, 176250, 176500, 177000, 177500, 178000, 178500, 179000, 179500, 179900, 179950, 180000, 180250,
180500, 181000, 181100, 182000, 182200, 182500, 182568, 182700, 183000, 183500, 183750, 184000, 184500, 184900, 185000,
185850, 185900, 186000, 186375, 186950, 187000, 187250, 187300, 187500, 188000, 188200, 188500, 189000, 189650, 189900,
189950, 190000, 190500, 190848, 191000, 191950, 192000, 192500, 192950, 193000, 193500, 194000, 194250, 194820, 194900,
194990, 195000, 195500, 195700, 196000, 196440, 196500, 196700, 196900, 197000, 197200, 197400, 197500, 198000, 198400,
198500, 198900, 199000, 199129, 199400, 199500, 199900, 199950, 199988, 199990, 199999, 200000, 200126, 200450, 200500,
201000, 201500, 201700, 202000, 202200, 202500, 202950, 203000, 203700, 204000, 204250, 204555, 204700, 204750, 204900,
204950, 204995, 205000, 205425, 205500, 205950, 206000, 206135, 206325, 206600, 206990, 207000, 207100, 207200, 207500,
207950, 208000, 208400, 208417, 208500, 208633, 208800, 208950, 209000, 209500, 209900, 209950, 209977, 209995, 210000,
210490, 210500, 210750, 211000, 212000, 212500, 212625, 212644, 212700, 213000, 213400, 213500, 213550, 213675, 213800,
213950, 214000, 214100, 214946, 214950, 215000, 215150, 215500, 216000, 216180, 216300, 216500, 216600, 216650, 217000,
217450, 217500, 218000, 218250, 218450, 218500, 219000, 219200, 219500, 219900, 219950, 220000, 220500, 220650, 221000,
221347, 221700, 221900, 222000, 222200, 222400, 222500, 222900, 223000, 223990, 224000, 224097, 224400, 224500, 224950,
224975, 225000, 225205, 225500, 225800, 225900, 226000, 226450, 226500, 226550, 226740, 226750, 226800, 226950, 227000,
227064, 227450, 227490, 227500, 227950, 228000, 228500, 228800, 228900, 228950, 229000, 229050, 229500, 229800, 229900,
229950, 229999, 230000, 230005, 230500, 230950, 231000, 231200, 231500, 231750, 232000, 232500, 232603, 232900, 233000,
233500, 233703, 234000, 234300, 234500, 234550, 234900, 234950, 234975, 234999, 235000, 235245, 235500, 235750, 235867,
236000, 236500, 236775, 237000, 237100, 237200, 237500, 237502, 237600, 237950, 238000, 238950, 239000, 239300, 239800,
239900, 239950, 239975, 239999, 240000, 240415, 240500, 241000, 241250, 241400, 241450, 241500, 242000, 242025, 242050,
242150, 242500, 242550, 243000, 243400, 243500, 243800, 243950, 244000, 244500, 244615, 244900, 244950, 245000, 245100,
245500, 245560, 245700, 245990, 246000, 246500, 246600, 246700, 246900, 246950, 247000, 247200, 247300, 247500, 247800,
248000, 248500, 249000, 249500, 249900, 249950, 250000, 250200, 250250, 250275, 250500, 250600, 250750, 250800, 251000,
251100, 251200, 251700, 251750, 252000, 252350, 252500, 252700, 252750, 253000, 253101, 253200, 253400, 253500, 253750,
253779, 253905, 254000, 254500, 254600, 254922, 254950, 254999, 255000, 255500, 255544, 255900, 255950, 256000, 256400,
256500, 256703, 256750, 256883, 256900, 256950, 257000, 257100, 257200, 257500, 257700, 257950, 258000, 258305, 258500,
258750, 258800, 258900, 258950, 259000, 259250, 259500, 259875, 259900, 259950, 260000, 260250, 260600, 260656, 260750,
261000, 261350, 261490, 261500, 261590, 261950, 262000, 262500, 263000, 263300, 263400, 263500, 263700, 263850, 263900,
263950, 264000, 264250, 264500, 264900, 264950, 265000, 265050, 265500, 265800, 265900, 265950, 265953, 266000, 266200,
266490, 266500, 266750, 266950, 267000, 267100, 267300, 267345, 267500, 267800, 267950, 268000, 268300, 268450, 268500,
268643, 268750, 268950, 269000, 269100, 269500, 269800, 269900, 269950, 270000, 270500, 270950, 271000, 271115, 271310,
271500, 271675, 271900, 271920, 271950, 272000, 272167, 272450, 272500, 272750, 272925, 272950, 273000, 273148, 273500,
273950, 274000, 274250, 274500, 274700, 274800, 274900, 274950, 274975, 275000, 275053, 275250, 275400, 275436, 275500,
275900, 276000, 276200, 276500, 276693, 276750, 276900, 277000, 277140, 277284, 277500, 277554, 277700, 277950, 278000,
278100, 278226, 278500, 278750, 278800, 279000, 279200, 279475, 279500, 279800, 279900, 279950, 280000, 280005, 280017,
280300, 280400, 280500, 280927, 280950, 281000, 281500, 281700, 282000, 282150, 282500, 282510, 282613, 282900, 282950,
283000, 283200, 283450, 283500, 283700, 283748, 284000, 284200, 284700, 284850, 284900, 284950, 285000, 285167, 285500,
285650, 285750, 285900, 285950, 286000, 286285, 286300, 286308, 286500, 286651, 286700, 286800, 286900, 286950, 287000,
287200, 287450, 287500, 287600, 287653, 288000, 288250, 288349, 288350, 288400, 288790, 289000, 289200, 289275, 289500,
289571, 289659, 289900, 289950, 289999, 290000, 290256, 290300, 290500, 290700, 290750, 290900, 291000, 291375, 291500,
291600, 291700, 291750, 291850, 291970, 292000, 292050, 292500, 292600, 293000, 293467, 293500, 293550, 294000, 294010,
294350, 294400, 294450, 294500, 294570, 294700, 294900, 294950, 294999, 295000, 295450, 295500, 295700, 295832, 295950,
296000, 296475, 296500, 297000, 297262, 297300, 297500, 297950, 297975, 298000, 298450, 298500, 298700, 298800, 298900,
298950, 299000, 299250, 299500, 299800, 299880, 299900, 299950, 299980, 299990, 299995, 299999, 300000, 300499, 300500,
300523, 301000, 301350, 301500, 301950, 302000, 302059, 302100, 302200, 302282, 302300, 302495, 302500, 302860, 303000,
303100, 303210, 303500, 303697, 303700, 304000, 304400, 304500, 304700, 304900, 304950, 304999, 305000, 305100, 305240,
305450, 305495, 305500, 305950, 306000, 306500, 306888, 306950, 307000, 307150, 307300, 307450, 307500, 307550, 307635,
307700, 307999, 308000, 308130, 308500, 308550, 308625, 308900, 308950, 309000, 309212, 309500, 309600, 309620, 309780,
309900, 309933, 309950, 310000, 310597, 310650, 310950, 311000, 311100, 311300, 311500, 311600, 311750, 311850, 312000,
312200, 312500, 312620, 312891, 312900, 313000, 313100, 313200, 313300, 313500, 313950, 313999, 314000, 314200, 314500,
314900, 314950, 314963, 315000, 315001, 315275, 315450, 315500, 316000, 316475, 316500, 316750, 317000, 317500, 317625,
317750, 317950, 318000, 318200, 318400, 318500, 318700, 318888, 318989, 319000, 319450, 319500, 319502, 319900, 319950,
319990, 320000, 320600, 320900, 321000, 321027, 321500, 321950, 322000, 322200, 322400, 322500, 322968, 323000, 323400,
323500, 323800, 324000, 324360, 324450, 324500, 324747, 324800, 324888, 324900, 324950, 325000, 325088, 325250, 325500,
325900, 326000, 326100, 326188, 326250, 326500, 326989, 326995, 327000, 327200, 327500, 327555, 328000, 328423, 328500,
328950, 329000, 329350, 329445, 329500, 329780, 329800, 329900, 329922, 329932, 329950, 329990, 329995, 329999, 330000,
330490, 330600, 330675, 330950, 331000, 331210, 331292, 331500, 331950, 332000, 332100, 332220, 332500, 332544, 332888,
332900, 333000, 333490, 333500, 333700, 333760, 333800, 334000, 334009, 334200, 334500, 334550, 334850, 334888, 334900,
334950, 334990, 334998, 334999, 335000, 335500, 335606, 335620, 335750, 335900, 335950, 336000, 336500, 336600, 336750,
336800, 336900, 336950, 337000, 337500, 337900, 338000, 338150, 338500, 338800, 338900, 338950, 338995, 339000, 339100,
339275, 339300, 339500, 339888, 339900, 339950, 339989, 339990, 339995, 339999, 340000, 340500, 340768, 340895, 341000,
341500, 341780, 341950, 342000, 342400, 342450, 342500, 342888, 343000, 343500, 343566, 343888, 344000, 344200, 344500,
344900, 344950, 345000, 345100, 345500, 345600, 345950, 346000, 346100, 346150, 346290, 346300, 346500, 346900, 346950,
347000, 347500, 347950, 348000, 348125, 348140, 348450, 348500, 348580, 349000, 349170, 349500, 349810, 349900, 349950,
349990, 350000, 350500, 350900, 351000, 351358, 351999, 352000, 352450, 352499, 352500, 352750, 352800, 352900, 352950,
353000, 353250, 353500, 353750, 353900, 353950, 354000, 354450, 354500, 354800, 354900, 354901, 354950, 355000, 355200,
355300, 355425, 355500, 355900, 355950, 356000, 356200, 356250, 356500, 356700, 356999, 357000, 357186, 357250, 357500,
```

```
357562, 357823, 357950, 358000, 358500, 358800, 358803, 358990, 359000, 359500, 359782, 359800, 359900, 359950, 359999,
360000, 360400, 360500, 361000, 361280, 361500, 361550, 361600, 361810, 362000, 362300, 362362, 362500, 362764, 362865,
362950, 363000, 363500, 363750, 363990, 364000, 364250, 364500, 364808, 364900, 364950, 364988, 365000, 365070, 365250,
365500, 365650, 366000, 366350, 366400, 366500, 366750, 367000, 367300, 367400, 367500, 367777, 367899, 367950, 367999,
368000, 368250, 368500, 368750, 368888, 369000, 369160, 369300, 369500, 369900, 369946, 369950, 369990, 370000, 370037,
370228, 370350, 370500, 370900, 370950, 371000, 371025, 371500, 372000, 372220, 372400, 372500, 372977, 373000, 373500,
374000, 374150, 374500, 374900, 374950, 374990, 375000, 375500, 375900, 375950, 376000, 376500, 376950, 377000, 377500,
377691, 378000, 378500, 378510, 378750, 378800, 378950, 379000, 379260, 379400, 379500, 379600, 379750, 379770, 379880,
379900, 379950, 380000, 380500, 380600, 380950, 381000, 381156, 381500, 381800, 382000, 382450, 382495, 382500, 382880,
382888, 383000, 383001, 383150, 383610, 383900, 383962, 384000, 384200, 384205, 384400, 384435, 384500, 384900, 384950,
385000, 385100, 385195, 385200, 385500, 386000, 386100, 386180, 386380, 386500, 386591, 386900, 386950, 387000, 387500,
387846, 387865, 387990, 388000, 388500, 388598, 389000, 389100, 389250, 389500, 389517, 389700, 389800, 389900, 389950,
389990, 389999, 390000, 390500, 391000, 391265, 391500, 392000, 392137, 392400, 392440, 392450, 392500, 392800, 393000,
393500, 393820, 394000, 394250, 394475, 394500, 394900, 394950, 394999, 395000, 395300, 395350, 395825, 395900, 395950,
396000, 396400, 396450, 396480, 396500, 396675, 396800, 396900, 397000, 397380, 397450, 397500, 397900, 397950, 397990,
398000, 398096, 398500, 398651, 398750, 398950, 399000, 399440, 399500, 399700, 399888, 399895, 399900, 399950, 399963,
399990, 399995, 400000, 400200, 400375, 400800, 400950, 401000, 401500, 401750, 402000, 402101, 402200, 402300, 402395,
402500, 402723, 403000, 403250, 403500, 403504, 403900, 403950, 404000, 404500, 404600, 404763, 404950, 405000, 405100,
405300, 405500, 405600, 406000, 406100, 406250, 406430, 406500, 406550, 406650, 407000, 407185, 407193, 407450, 407500,
408000, 408200, 408474, 408500, 408506, 409000, 409124, 409316, 409500, 409900, 409950, 410000, 410500, 410988, 411000,
411100, 411500, 411605, 411715, 411753, 411800, 412000, 412133, 412250, 412450, 412500, 412950, 413000, 413100, 413107,
413252, 413450, 413500, 413565, 413800, 413900, 414000, 414050, 414250, 414500, 414900, 414950, 414999, 415000, 415250,
415500, 415885, 415900, 415950, 416000, 416100, 416286, 416500, 417000, 417200, 417250, 417400, 417500, 417838, 418000,
418200, 418395, 418500, 418800, 418900, 419000, 419190, 419354, 419500, 419600, 419625, 419700, 419900, 419950, 419990,
419995, 420000, 420200, 420250, 420550, 420850, 421000, 421200, 421500, 422000, 422120, 422250, 422500, 422800, 423000,
423500, 423700, 423800, 424000, 424240, 424305, 424500, 424900, 424950, 425000, 425500, 425590, 425996, 426000, 426250,
426500, 426700, 426950, 427000, 427005, 427200, 427500, 427550, 427800, 427874, 428000, 428040, 428400, 428750, 428900,
428950, 429000, 429300, 429500, 429592, 429800, 429900, 429950, 430000, 430100, 430236, 430760, 431000, 431200, 431500,
431650, 431750, 432000, 432100, 432250, 432500, 432900, 433000, 433190, 433200, 433495, 433500, 434000, 434400, 434500,
434900, 434975, 435000, 435010, 435500, 436000, 436110, 436300, 436472, 436500, 436800, 436952, 437000, 437400, 437500,
437718, 437850, 438000, 438200, 438400, 438500, 438600, 438750, 438800, 438900, 438924, 438950, 439000, 439108, 439500,
439800, 439888, 439900, 439950, 439990, 439995, 440000, 440150, 440250, 440500, 441000, 441500, 441750, 442000, 442200,
442250, 442500, 442515, 442573, 442900, 443000, 443500, 443600, 443725, 443750, 443950, 444000, 444500, 444900, 444950,
445000, 445434, 445500, 445700, 445800, 445830, 445838, 445950, 446000, 446250, 446450, 446500, 446800, 446950, 447000,
447055, 447450, 447500, 448000, 448175, 448500, 449000, 449228, 449250, 449400, 449500, 449888, 449900, 449950, 449990,
449999, 450000, 450500, 450600, 450800, 451000, 451101, 451300, 451555, 452000, 452100, 452250, 452500, 452950, 453000,
453246, 453250, 453500, 454000, 454200, 454280, 454450, 454800, 454900, 454950, 455000, 455500, 455600, 455800, 455850,
455950, 456000, 456150, 456200, 456500, 456700, 457000, 457500, 458000, 458400, 458450, 458500, 458950, 459000, 459500,
459800, 459900, 459950, 459990, 459995, 460000, 460458, 460500, 460940, 461000, 461100, 461500, 461550, 462000, 462370,
462500, 462550, 462600, 462608, 463000, 463500, 463800, 463828, 464000, 464050, 464500, 464550, 464600, 464625, 464900,
464950, 465000, 465250, 465425, 465500, 465750, 465950, 466000, 466200, 466500, 466750, 466800, 466950, 467000, 467100,
467500, 468000, 468500, 469000, 469500, 469775, 469900, 469950, 469995, 470000, 470101, 470450, 470500, 470950, 471000,
471001, 471275, 471500, 471750, 471835, 472000, 472217, 472500, 472800, 473000, 473600, 473975, 474000, 474500, 474800,
474900, 474905, 474950, 475000, 475200, 475226, 475300, 475500, 475580, 475999, 476000, 476100, 476500, 476800, 476900,
477000, 477500, 477590, 478000, 478500, 478830, 479000, 479200, 479349, 479500, 479900, 479950, 479990, 480000, 480500,
480680, 481000, 481015, 481203, 481450, 481500, 482000, 482500, 482975, 483000, 483300, 483453, 483500, 483945, 484000,
484259, 484950, 484998, 485000, 485230, 485500, 486000, 486700, 486940, 487000, 487028, 487250, 487275, 487500, 487585,
487600, 488000, 488250, 488500, 488800, 489000, 489200, 489500, 489900, 489950, 489990, 490000, 490500, 490600, 491000,
491150, 491234, 491300, 491500, 491950, 492000, 492450, 492500, 492650, 493000, 493500, 494000, 494400, 494500, 494815,
494900, 494950, 494995, 495000, 495200, 495500, 495800, 496000, 496500, 496600, 496700, 496752, 496800, 497000, 497300,
497500, 497950, 498000, 498445, 498500, 498688, 498800, 499000, 499100, 499160, 499431, 499500, 499900, 499922, 499950,
499990, 500000, 500007, 500012, 501000, 502000, 502500, 502501, 502550, 502700, 502775, 503000, 503045, 503500, 504058,
504200, 504500, 504600, 504750, 504975, 505000, 505400, 505500, 505657, 506000, 506400, 506500, 506950, 507000, 507200,
507250, 507500, 507950, 508000, 508300, 508450, 508500, 508800, 509000, 509007, 509250, 509500, 509900, 509950, 509990,
510000, 510250, 510500, 511000, 511100, 511200, 511500, 511555, 511718, 512000, 512031, 512500, 513000, 514000, 514500,
514700, 514950, 515000, 515055, 515100, 515500, 515700, 515805, 516000, 516130, 516200, 516250, 516500, 517000, 517100,
517500, 517534, 517850, 517950, 518000, 518380, 518500, 519000, 519500, 519900, 519950, 519990, 519995, 520000, 520500,
521000, 521450, 521500, 521900, 522000, 522250, 522500, 523000, 523460, 523500, 523950, 524000, 524225, 524250, 524400,
524500, 524950, 525000, 525126, 525300, 525888, 526000, 526500, 526750, 527000, 527200, 527500, 527550, 527700, 527900,
527950, 528000, 529000, 529100, 529219, 529500, 529888, 529900, 529941, 529950, 529999, 530000, 530100, 530200, 531000,
531155, 531500, 531800, 532000, 532170, 532500, 533000, 533050, 533112, 533250, 533300, 533380, 533500, 533600, 534000,
534500, 534640, 534950, 535000, 535100, 535365, 535500, 535610, 535800, 535900, 535950, 536000, 536500, 536650, 536751,
537000, 537100, 537250, 537500, 538000, 538200, 538250, 538500, 538888, 538900, 539000, 539500, 539900, 539950, 540000,
540400, 540500, 541000, 541100, 541125, 541338, 541500, 541800, 541900, 542000, 542126, 542300, 542500, 542525, 542950,
543000, 543115, 543200, 543500, 544000, 544300, 544500, 544800, 544900, 544950, 544999, 545000, 545400, 545500, 545800,
546000, 546200, 546500, 546800, 546940, 547000, 547500, 548000, 548050, 548500, 548800, 549000, 549010, 549500, 549800,
549900, 549950, 549995, 550000, 550120, 550285, 550388, 550500, 550700, 551000, 551100, 551500, 551870, 552000, 552100,
552250, 552321, 552500, 552625, 552700, 552775, 552900, 553000, 553500, 553650, 554000, 554500, 554600, 554663, 554729,
554820, 554950, 554990, 555000, 555500, 555565, 555700, 555750, 555950, 556000, 556300, 557000, 557500, 557510, 557800,
557865, 558000, 559000, 559500, 559630, 559900, 559950, 560000, 560200, 561000, 561500, 561600, 561750, 562000, 562100,
562200, 562500, 563000, 563225, 563250, 563500, 563750, 563950, 564000, 564450, 564500, 564800, 564950, 565000, 565500,
565997, 566000, 566950, 567000, 567035, 567500, 568000, 568450, 568500, 569000, 569500, 569888, 569900, 569950, 569995,
569999, 570000, 570500, 571000, 571500, 571900, 572000, 572115, 572500, 572650, 572800, 573000, 573300, 573500, 574000,
574500, 574800, 574950, 575000, 575550, 575575, 575700, 575950, 576000, 576250, 576750, 576925, 577000, 577288, 577450,
577500, 578000, 578500, 578550, 578888, 579000, 579100, 579500, 579950, 580000, 580050, 580135, 580379, 580500, 581000,
582000, 582500, 582800, 583000, 583500, 583800, 584000, 584950, 584999, 585000, 585083, 585188, 585444, 585888, 586000,
586500, 587000, 587100, 587206, 587450, 587500, 587750, 588000, 588500, 589000, 589410, 589450, 589500, 589900, 589950,
589999, 590000, 590300, 591000, 591500, 591975, 592000, 592100, 592350, 592500, 593000, 593450, 593500, 593567, 593700,
593777, 594000, 594491, 594866, 594950, 595000, 595500, 595888, 596000, 596500, 597000, 597157, 597326, 597400, 597500,
597750, 598000, 598200, 598500, 598555, 598600, 598780, 598800, 598850, 598950, 598992, 599000, 599380, 599500, 599832,
599900, 599950, 599990, 599995, 599999, 600000, 600600, 601000, 601002, 601150, 601450, 601500, 602000, 602500, 603000,
603500, 604000, 604700, 604950, 605000, 605004, 605125, 605500, 606000, 606150, 606400, 606500, 607000, 607010, 607500,
```

```
608000, 608095, 608250, 608500, 608700, 609000, 609500, 609850, 609900, 609950, 610000, 610360, 610685, 610750, 610950,
611000, 611206, 611900, 612000, 612125, 612500, 612995, 613000, 613200, 613500, 614000, 614285, 614306, 614905, 614950,
615000, 615500, 615750, 616000, 616200, 616300, 616500, 616750, 616950, 617000, 617450, 617950, 618000, 618080, 618250,
618500, 619000, 619100, 619400, 619420, 619500, 619790, 619850, 619950, 619990, 620000, 620047, 621000, 621138, 621500,
622000, 622100, 622200, 622500, 622950, 623000, 623300, 623500, 624000, 624500, 624800, 624900, 624950, 625000, 625250,
625500, 625504, 625700, 626000, 626100, 626500, 626700, 627000, 627250, 627500, 627800, 628000, 628990, 629000, 629500,
629800, 629950, 630000, 630100, 630500, 631000, 631500, 631625, 631750, 632000, 632500, 632750, 632925, 633000, 633100,
633634, 634000, 634800, 634950, 635000, 635200, 635250, 635500, 635700, 636000, 636100, 636230, 637000, 637250, 637500,
637800, 637850, 638000, 638150, 638250, 638500, 638700, 639000, 639500, 639888, 639900, 639950, 639983, 640000, 640500,
641000, 641200, 641250, 641500, 642000, 642450, 642860, 643000, 643002, 643403, 643500, 643950, 644000, 644500, 645000,
645500, 646000, 646800, 647000, 647500, 648000, 648360, 648475, 648752, 649000, 649500, 649800, 649950, 649990, 650000,
650100, 650500, 650880, 651000, 651100, 651500, 652000, 652100, 652427, 652450, 652500, 652600, 653000, 653450, 653500,
653675, 653750, 654000, 654300, 654500, 654950, 655000, 655100, 655275, 655500, 656000, 656500, 657000, 657044, 657100,
657500, 658000, 658100, 658500, 658588, 658600, 659000, 659500, 659950, 660000, 660500, 661000, 661254, 661500, 662000,
662500, 662700, 662990, 663000, 663500, 664000, 664500, 664950, 665000, 665900, 666000, 666500, 666570, 667000, 667400,
667500, 667750, 668000, 668500, 668750, 669000, 669500, 669888, 669950, 670000, 670500, 670950, 671000, 671300, 671500,
672000, 672324, 672500, 672600, 672800, 673000, 673200, 674000, 674250, 674600, 674725, 674750, 674950, 675000, 675500,
675750, 675900, 676000, 676101, 676500, 677000, 677100, 677500, 677790, 677900, 677915, 678000, 678100, 678500, 678700,
678940, 679000, 679900, 679950, 679975, 679990, 680000, 680200, 681000, 681500, 681716, 682000, 682500, 683000, 683500,
684000, 684680, 685000, 685100, 685530, 685650, 685900, 686000, 686500, 687000, 687015, 687500, 688000, 688100, 688500,
688888, 689000, 689500, 689800, 689888, 689900, 689950, 690000, 690500, 690700, 691000, 691100, 691500, 692000, 692500,
693000, 694000, 695000, 695500, 696000, 696500, 696950, 697000, 698000, 699000, 699188, 699800, 699850, 699900, 699950,
699999, 700000, 700180, 700500, 701000, 702000, 702500, 703000, 703011, 703300, 703770, 704000, 704111, 704300, 705000,
705380, 705640, 706000, 707000, 707500, 707900, 708000, 709000, 709050, 709950, 710000, 710200, 710500, 710800, 711000,
711600, 711777, 711800, 712000, 712198, 712500, 713000, 713250, 713400, 713414, 713500, 713900, 714000, 715000, 715500,
716000, 716100, 716125, 716500, 716528, 717000, 717500, 717550, 718000, 718500, 719000, 719521, 719950, 720000, 720001,
720168, 720500, 721000, 721500, 722000, 722080, 722500, 722800, 723000, 724000, 724500, 724800, 724950, 725000, 725126,
725500, 725786, 725995, 726000, 726500, 726888, 727000, 727160, 727500, 728000, 728050, 728725, 728935, 729000, 729032,
729500, 729950, 729953, 729999, 730000, 730001, 730100, 731000, 731100, 731500, 731688, 731781, 732000, 732350, 732500,
732600, 733000, 733500, 734000, 734200, 734500, 734950, 734990, 735000, 736000, 736500, 737000, 737500, 738000, 738500,
738515, 738950, 739000, 739375, 739500, 739888, 739900, 739999, 740000, 740500, 741000, 741500, 742000, 742500, 743000,
743700, 744000, 744500, 745000, 745641, 746000, 746300, 746500, 747000, 747450, 747500, 748000, 749000, 749400, 749500,
749700, 749950, 749995, 749999, 750000, 750500, 751000, 751305, 751750, 752000, 752500, 752875, 752888, 753000, 753888,
754000, 754300, 754800, 754842, 754950, 754999, 755000, 756000, 756100, 756450, 757000, 757500, 758000, 758800, 759000,
759600, 759900, 759950, 759990, 760000, 760005, 760250, 760369, 760500, 760750, 761000, 762000, 762300, 762400, 762450,
762500, 763000, 763101, 763776, 764000, 765000, 766000, 766500, 766950, 767250, 767450, 767500, 768000, 768500, 769000,
769900, 769950, 769995, 770000, 770126, 771000, 771005, 771150, 772000, 772500, 772650, 773000, 774000, 774888, 774900,
774950, 775000, 775900, 775950, 776000, 776500, 777000, 777700, 778000, 778100, 778983, 779000, 779380, 779950, 780000,
780500, 781000, 781500, 782000, 782500, 782900, 783000, 783200, 783350, 783500, 784000, 784500, 784950, 785000, 785200,
785500, 785950, 786000, 787000, 787500, 787888, 788000, 788500, 788600, 789000, 789500, 789800, 789888, 789900, 790000,
790100, 790500, 791000, 791500, 792000, 792500, 793000, 794154, 794500, 795000, 795127, 796000, 796500, 797000, 797500,
798000, 798500, 798750, 798800, 799000, 799200, 799500, 799900, 799950, 799990, 800000, 800500, 800866, 801000, 801501,
802000, 802500, 802541, 802945, 803000, 803100, 804000, 804100, 804995, 805000, 805500, 806000, 807000, 807100, 807500,
808000, 808100, 808250, 808900, 809000, 809950, 810000, 811000, 811500, 812000, 812500, 813000, 813500, 814000, 814842,
814950, 815000, 815241, 816000, 817000, 817250, 817500, 818000, 818500, 818900, 819000, 819900, 819995, 820000, 820875,
821000, 822000, 822500, 822600, 823000, 824000, 824500, 825000, 825050, 825500, 825750, 826000, 826600, 827000, 827235,
827500, 828000, 828200, 828500, 828950, 829000, 829900, 829950, 829995, 830000, 830005, 830200, 831000, 831500, 831548,
832000, 832500, 832600, 833000, 833450, 834000, 834500, 834538, 834800, 834950, 834995, 835000, 835100, 836000, 836500,
837000, 837219, 837500, 837700, 838000, 838300, 838400, 839000, 839704, 839900, 839950, 839990, 840000, 840500, 841000,
842000, 842500, 843000, 843500, 844000, 845000, 845800, 845950, 846000, 846450, 847000, 847093, 847700, 848000, 848750,
849000, 849900, 849950, 849990, 850000, 850830, 851000, 851500, 852000, 852500, 852600, 852880, 853000, 853505, 853800,
854000, 855000, 855169, 856000, 856500, 856600, 857000, 857326, 857500, 858000, 858450, 859000, 859900, 859950, 859990,
860000, 861000, 861111, 861990, 862000, 862500, 863000, 863500, 864000, 864327, 864500, 865000, 865950, 866000, 866059,
866500, 866800, 868000, 868500, 868700, 869000, 869900, 869950, 870000, 870300, 870515, 871000, 872000, 872500, 872750,
873000, 874000, 874150, 874950, 875000, 875909, 876650, 877500, 878000, 879000, 879900, 879950, 880000, 881000, 882566,
882990, 883000, 884250, 884744, 884900, 885000, 885250, 886000, 887000, 887200, 887250, 887500, 888000, 888550, 888990,
889000, 889950, 890000, 890776, 890900, 891000, 891500, 892500, 893880, 894000, 894400, 895000, 895900, 895950, 895990,
896000, 897000, 897500, 898000, 898500, 898888, 899000, 899100, 899900, 899950, 900000, 901000, 902000, 902500, 903000,
905000, 906000, 907000, 907500, 907687, 908800, 908950, 908990, 909000, 909500, 909950, 910000, 911000, 911100, 912000,
913000, 913888, 914154, 914500, 914600, 915000, 915557, 917000, 917500, 918000, 919000, 919204, 919950, 919990, 920000,
921000, 921500, 921800, 922000, 922755, 923990, 924000, 925000, 925500, 925850, 925900, 926250, 926300, 926500, 927000,
928950, 928990, 929000, 929950, 930000, 930800, 931000, 931088, 932800, 932808, 932990, 933000, 933399, 934000, 934550,
935000, 935100, 936000, 937000, 937500, 937750, 938000, 939000, 940000, 941000, 941500, 942000, 942500, 942990, 943500,
945000, 945800, 946000, 947500, 948000, 949000, 949880, 949950, 949990, 950000, 950968, 951000, 951250, 952000, 952500,
952990, 953007, 954500, 955000, 955500, 955990, 957000, 957500, 958000, 959000, 959750, 959900, 960000, 961000, 961500,
962000, 962800, 963000, 963990, 964000, 965000, 965800, 966000, 967000, 967500, 968000, 968060, 968933, 969000, 969500,
969950, 969990, 970000, 970500, 971000, 971971, 972000, 972800, 974350, 975000, 976000, 978000, 978500, 979000, 979500,
979700, 980000, 981000, 982000, 982218, 984000, 985000, 986000, 987000, 987500, 988000, 988500, 988830, 988990, 989000,
989900, 989990, 990000, 990400, 991500, 991700, 992000, 993000, 993500, 994000, 994900, 995000, 995500, 996000, 997000,
997950, 998000, 998160, 998500, 998800, 999000, 999950, 999999, 1000000, 1000750, 1001000, 1003000, 1005000, 1007500, 10
08000, 1010000, 1010800, 1011000, 1012000, 1013050, 1014250, 1015000, 1017000, 1017100, 1020000, 1025000, 1027000, 10275
00, 1028000, 1028950, 1029000, 1029280, 1029900, 1030000, 1031000, 1033890, 1034500, 1035000, 1035290, 1035480, 1037000,
1038000, 1039000, 1040000, 1040890, 1042000, 1042030, 1042500, 1045000, 1046250, 1047000, 1047500, 1049000, 1049990, 105
0000, 1051000, 1052000, 1052500, 1054690, 1054710, 1055000, 1057000, 1058000, 1058800, 1059000, 1059500, 1060000, 106160
0, 1062000, 1062500, 1063000, 1065000, 1065500, 1067000, 1068000, 1070000, 1072000, 1072500, 1075000, 1078000, 1078500,
1079000, 1080000, 1081000, 1084500, 1085000, 1085500, 1086000, 1087500, 1088000, 1088890, 1089000, 1090000, 1093000, 109
5000, 1096500, 1098000, 1099500, 1099880, 1100000, 1101000, 1102030, 1103990, 1104500, 1105000, 1107460, 1108000, 111000
0, 1112000, 1112500, 1112750, 1115000, 1115500, 1118000, 1120000, 1120280, 1122500, 1125000, 1126000, 1127000, 1127500,
1130000, 1131000, 1135000, 1135250, 1137500, 1138990, 1139990, 1140000, 1142000, 1145000, 1146800, 1148000, 1149000, 115
0000, 1151250, 1153000, 1155000, 1156000, 1157200, 1157400, 1160000, 1161000, 1164000, 1165000, 1168000, 1169000, 117000
0, 1174660, 1175000, 1180000, 1180500, 1184000, 1185000, 1186040, 1187500, 1190000, 1191000, 1195000, 1197000, 1197350,
1198000, 1199000, 1199500, 1200000, 1200690, 1202500, 1205000, 1206500, 1206690, 1208000, 1209000, 1210000, 1211000, 121
```

```
2500, 1215000, 1216000, 1218000, 1220000, 1222500, 1225000, 1227500, 1228000, 1229000, 1230000, 1234000, 1234570, 123458
0, 1236000, 1236300, 1237500, 1238000, 1240000, 1240420, 1242000, 1242500, 1245000, 1247000, 1248000, 1249000, 1250000,
1255000, 1255780, 1256500, 1258000, 1260000, 1260500, 1262000, 1264000, 1265000, 1266520, 1267500, 1268890, 1270000, 127
2000, 1272500, 1274950, 1275000, 1278000, 1280000, 1280600, 1284000, 1285000, 1288000, 1289000, 1289990, 1290000, 129500
0, 1295650, 1297000, 1298000, 1298890, 1299890, 1300000, 1302000, 1305000, 1306000, 1307000, 1308000, 1309500, 1310000,
1311000, 1312000, 1313000, 1315000, 1320000, 1321500, 1321620, 1324050, 1325000, 1326000, 1328000, 1330000, 1333000, 133
5000, 1337500, 1338750, 1339000, 1340000, 1345000, 1346400, 1348000, 1349000, 1350000, 1355000, 1356920, 1360000, 136250
0, 1364000, 1365000, 1370000, 1375000, 1378000, 1378600, 1379900, 1380000, 1381000, 1384000, 1385000, 1387000, 1387800,
1388000, 1389000, 1393000, 1395000, 1395710, 1398000, 1399000, 1399950, 1400000, 1405000, 1406890, 1408760, 1410000, 141
1600, 1415000, 1419000, 1420000, 1425000, 1430000, 1430800, 1436000, 1437500, 1438890, 1440000, 1442500, 1443920, 144400
0, 1445000, 1450000, 1452000, 1454000, 1457000, 1459000, 1460000, 1462500, 1465000, 1468000, 1470000, 1475000, 1476000,
1480000, 1481000, 1482500, 1484900, 1485000, 1488000, 1490000, 1495000, 1500000, 1505000, 1506000, 1510000, 1511250, 151
5000, 1517000, 1518630, 1520000, 1525000, 1530000, 1532500, 1535000, 1537000, 1538000, 1540000, 1544500, 1545000, 155000
0, 1555000, 1557600, 1562000, 1563100, 1564350, 1565000, 1568000, 1569500, 1570000, 1575000, 1578000, 1580000, 1582500,
1583000, 1590000, 1595000, 1598000, 1598890, 1599950, 1600000, 1605000, 1610000, 1612500, 1615000, 1620000, 1620500, 162
5000, 1629000, 1635000, 1636000, 1637500, 1640000, 1646000, 1648000, 1650000, 1651000, 1655000, 1660000, 1662000, 166500
0, 1670000, 1675000, 1679000, 1680000, 1681000, 1688000, 1690000, 1691000, 1695000, 1697000, 1698000, 1698890, 1699000,
1699990, 1700000, 1702500, 1705000, 1710000, 1712500, 1712750, 1715000, 1720000, 1727000, 1728000, 1730000, 1735000, 173
8000, 1740000, 1749000, 1750000, 1755000, 1760000, 1762000, 1765000, 1769000, 1770000, 1775000, 1776000, 1780000, 178500
0, 1789950, 1795000, 1799000, 1800000, 1802750, 1810000, 1815000, 1820000, 1822500, 1824100, 1825000, 1830000, 1835000,
1839900, 1850000, 1851000, 1855000, 1862000, 1865000, 1870000, 1875000, 1880000, 1881580, 1886700, 1890000, 1895000, 189
8000, 1899000, 1900000, 1901000, 1905000, 1910000, 1920000, 1925000, 1928000, 1940000, 1945000, 1950000, 1955000, 195900
0, 1960000, 1965000, 1965220, 1970000, 1975000, 1980000, 1987500, 1989000, 1990000, 1998000, 1999000, 1999950, 2000000,
2005000, 2027000, 2048000, 2050000, 2065000, 2075000, 2095000, 2100000, 2110000, 2125000, 2135000, 2140000, 2147500, 215
0000, 2152500, 2160000, 2175000, 2180000, 2187730, 2193000, 2195000, 2196000, 2200000, 2205000, 2225000, 2230000, 223889
0, 2250000, 2260000, 2271150, 2280000, 2288000, 2298000, 2300000, 2320000, 2321000, 2328000, 2340000, 2350000, 2351960,
2367000, 2375000, 2384000, 2385000, 2395000, 2400000, 2408000, 2415000, 2450000, 2453500, 2458000, 2466350, 2475000, 247
9000, 2480000, 2485000, 2500000, 2510000, 2525000, 2532000, 2535000, 2537000, 2538000, 2544750, 2546000, 2555000, 257400
0, 2575000, 2600000, 2630000, 2641100, 2650000, 2680000, 2700000, 2720000, 2725000, 2750000, 2795000, 2850000, 2880500,
2885000, 2888000, 2890000, 2900000, 2903000, 2920000, 2945000, 2950000, 2983000, 2998000, 3000000, 3065000, 3070000, 307
5000, 3100000, 3120000, 3168750, 3200000, 3204000, 3278000, 3300000, 3345000, 3395000, 3400000, 3418800, 3567000, 360000
0, 3635000, 3640900, 3650000, 3710000, 3800000, 3850000, 4000000, 4208000, 4489000, 4500000, 4668000, 5110800, 5300000,
5350000, 5570000, 6885000, 7062500, 7700000]
```

Again, ignoring the negative range because our prices start at 75,000. so let's drop values greater than 1129575.

However, let's double check on how many entries we will be discarding before we do so.

In [22]:
```python
price_counts = df.groupby("price")["price"].agg("count").sort_values(ascending=True)
price_counts
```

Out[22]:
```
price
75000       1
607010      1
608095      1
608250      1
608500      1
         ... 
425000    150
500000    152
550000    159
350000    172
450000    172
Name: price, Length: 4028, dtype: int64
```

In [23]:
```python
pd.set_option("display.max_rows", 5000)
```

In [24]:
```
price_counts = df.groupby("price")["price"].agg("count").sort_values(ascending=False)
price_counts
```

```
950000     47
610000     47
665000     47
690000     44
685000     44
505000     44
825000     43
595000     42
205000     42
710000     41
740000     41
900000     40
645000     40
715000     39
190000     39
875000     39
720000     38
765000     37
175000     36
1050000    36
```

In [25]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   sqft_lot     21613 non-null  int64  
 1   sqft_living  21613 non-null  int64  
 2   grade        21613 non-null  int64  
 3   condition    21613 non-null  int64  
 4   bathrooms    21613 non-null  float64
 5   bedrooms     21613 non-null  int64  
 6   waterfront   21613 non-null  int64  
 7   price        21613 non-null  int32  
 8   floors       21613 non-null  float64
 9   lat          21613 non-null  float64
 10  long         21613 non-null  float64
dtypes: float64(4), int32(1), int64(6)
memory usage: 1.7 MB
```

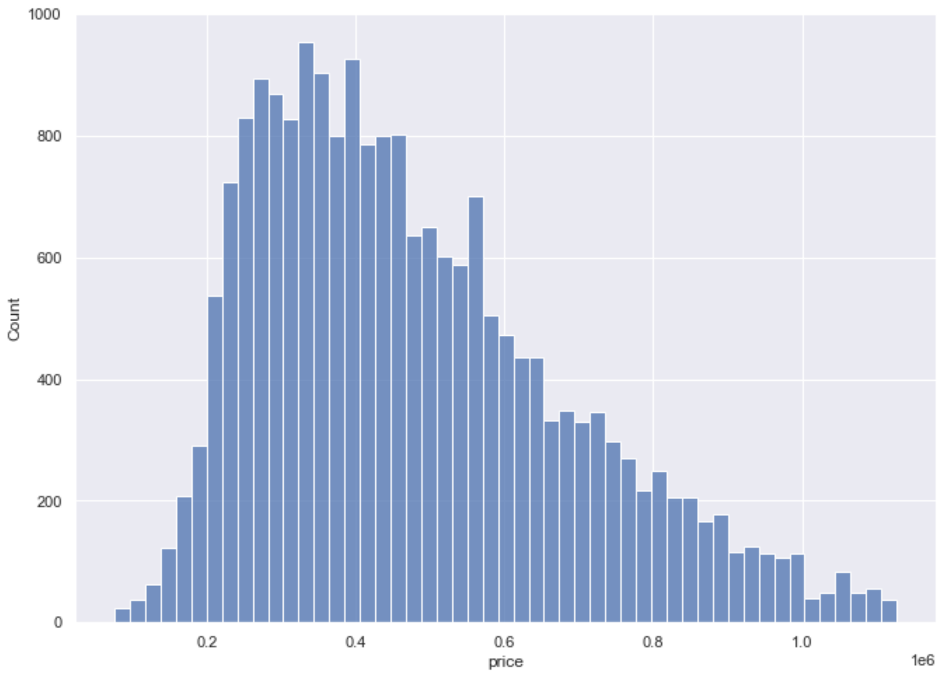
Since price values greater than 1129575 are outliers, we have to keep values less than or equal to 1129575.

In [26]:
```
df_outliers_rmvd = df[df["price"] <= 1129575]
df_outliers_rmvd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20467 entries, 0 to 21612
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   sqft_lot     20467 non-null  int64  
 1   sqft_living  20467 non-null  int64  
 2   grade        20467 non-null  int64  
 3   condition    20467 non-null  int64  
 4   bathrooms    20467 non-null  float64
 5   bedrooms     20467 non-null  int64  
 6   waterfront   20467 non-null  int64  
 7   price        20467 non-null  int32  
 8   floors       20467 non-null  float64
 9   lat          20467 non-null  float64
 10  long         20467 non-null  float64
dtypes: float64(4), int32(1), int64(6)
memory usage: 1.8 MB
```

In [27]:
```
sns.histplot(df_outliers_rmvd.price)
```

Out[27]: <AxesSubplot:xlabel='price', ylabel='Count'>



Our new visual plot. Not the best, but with the outliers removed, it'll work for now.

In [28]:
```
df_outliers_rmvd.price.describe()
```

Out[28]:
```
count    2.046700e+04
mean     4.769846e+05
std      2.083713e+05
min      7.500000e+04
25%      3.150000e+05
50%      4.375000e+05
75%      6.000000e+05
max      1.127500e+06
Name: price, dtype: float64
```

Now, trying to simplify the code: This will be out reuseable template for the other features.

In [29]:
```
q3, q1 = np.percentile(df_outliers_rmvd["price"], [75 ,25])
iqr = q3 - q1
print("iqr=", iqr)
print("q3=", q3)
print("q1=", q1)
print("constant=", iqr*1.5)
```

```
iqr= 285000.0
q3= 600000.0
q1= 315000.0
constant= 427500.0
```

In [30]:
```
print("suspected outliers are greater than this number:", q3+(iqr*1.5))
print("suspected outliers are less than this number", q1-(iqr*1.5))
```

```
suspected outliers are greater than this number: 1027500.0
suspected outliers are less than this number -112500.0
```

So regarding "price", any price value < -162625 and > 1129575 are outliers.

Trying to create a reuseable template. We'll try it with Living Space Square Footage.

1.4.0.2 Living Space Square Footage

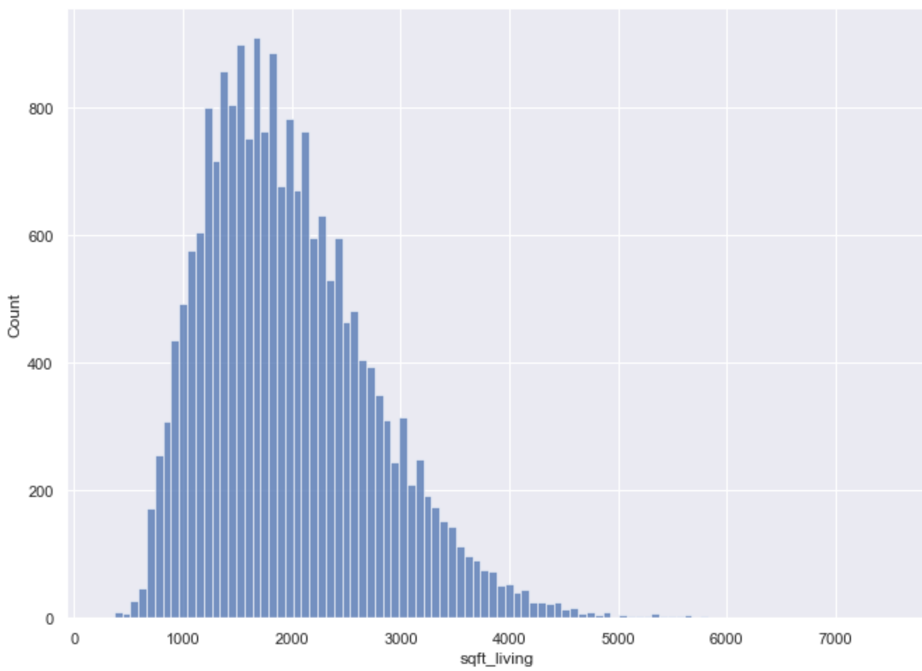
In [31]:
```
df_outliers_rmvd.sqft_living.describe()
```

Out[31]:
```
count    20467.000000
mean      1975.558167
std        774.833460
min        290.000000
25%       1400.000000
50%       1860.000000
75%       2431.000000
max       7480.000000
Name: sqft_living, dtype: float64
```

Visual Plot: Initial Look

In [32]:
```
sns.histplot(df_outliers_rmvd["sqft_living"])
```

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



Now, trying to take out the outliers to hopefully normalize the distribution.

In [33]:
```
q3, q1 = np.percentile(df_outliers_rmvd["sqft_living"], [75 ,25])
iqr = q3 - q1
print("iqr=", iqr)
print("q3=", q3)
print("q1=", q1)
print("constant=", iqr*1.5)
```

```
iqr= 1031.0
q3= 2431.0
q1= 1400.0
constant= 1546.5
```

In [34]:
```
print("suspected outliers are greater than this number:", q3+(iqr*1.5))
print("suspected outliers are less than this number", q1-(iqr*1.5))
```

```
suspected outliers are greater than this number: 3977.5
suspected outliers are less than this number -146.5
```

So regarding "sqft_living", any sqft_living value < -146.5 and > 3977.5 are outliers. Again, any negative numbers, we can sort of ignore, unless negative values start appearing on our histogram plot.

Let's remove the outliers.

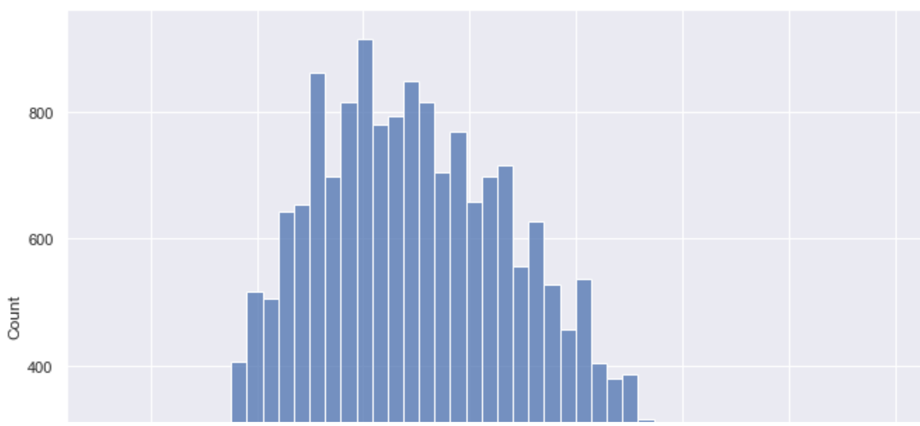
In [35]:
```
df_outliers_rmvd = df_outliers_rmvd[df_outliers_rmvd["sqft_living"] <= 3977.5]
df_outliers_rmvd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20147 entries, 0 to 21612
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   sqft_lot     20147 non-null  int64  
 1   sqft_living  20147 non-null  int64  
 2   grade        20147 non-null  int64  
 3   condition    20147 non-null  int64  
 4   bathrooms    20147 non-null  float64
 5   bedrooms     20147 non-null  int64  
 6   waterfront   20147 non-null  int64  
 7   price        20147 non-null  int32  
 8   floors       20147 non-null  float64
 9   lat          20147 non-null  float64
 10  long         20147 non-null  float64
dtypes: float64(4), int32(1), int64(6)
memory usage: 1.8 MB
```

Let's see the new histogram plot.

In [36]:
```
sns.histplot(df_outliers_rmvd["sqft_living"])
```

Out[36]:
```
<AxesSubplot:xlabel='sqft_living', ylabel='Count'>
```



Still a bit crude but we can work with that for now.

1.4.0.3 Lot Square Footage

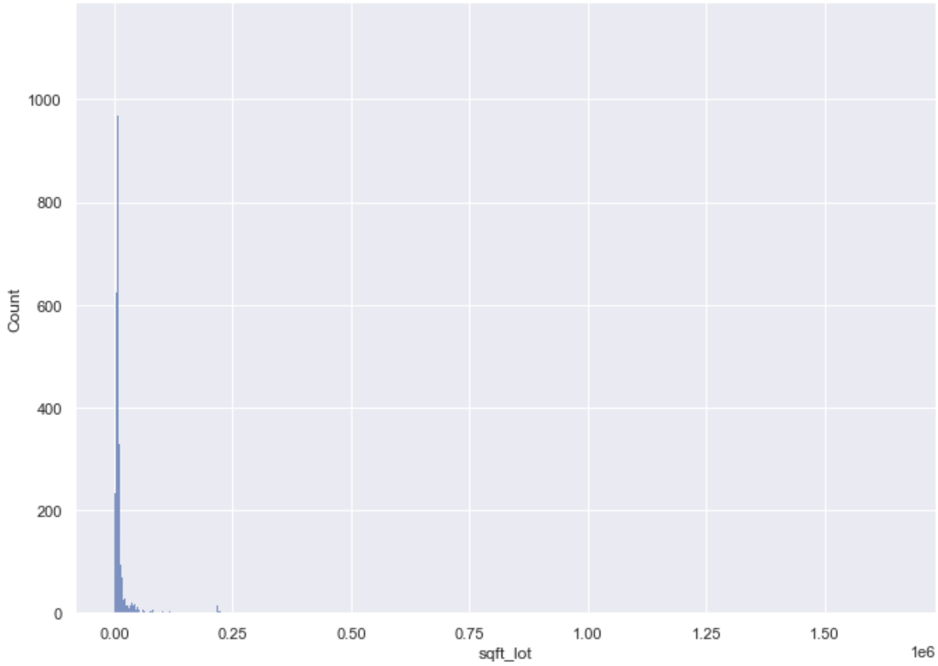
In [37]:
```
df_outliers_rmvd.sqft_lot.describe()
```

Out[37]:
```
count    2.014700e+04
mean     1.399957e+04
std      3.787604e+04
min      5.200000e+02
25%      5.000000e+03
50%      7.482000e+03
75%      1.020000e+04
max      1.651359e+06
Name: sqft_lot, dtype: float64
```

Visual Plot: Initial Look

In [38]:
```
sns.histplot(df_outliers_rmvd["sqft_lot"])
```

Out[38]: <AxesSubplot:xlabel='sqft_lot', ylabel='Count'>



Now, trying to take out the outliers to hopefully normalize the distribution.

In [39]:
```
q3, q1 = np.percentile(df_outliers_rmvd["sqft_lot"], [75 ,25])
iqr = q3 - q1
print("iqr=", iqr)
print("q3=", q3)
print("q1=", q1)
print("constant=", iqr*1.5)
```

```
iqr= 5200.0
q3= 10200.0
q1= 5000.0
constant= 7800.0
```

In [40]:
```
print("suspected outliers are greater than this number:", q3+(iqr*1.5))
print("suspected outliers are less than this number", q1-(iqr*1.5))
```

```
suspected outliers are greater than this number: 18000.0
suspected outliers are less than this number -2800.0
```

So regarding "sqft_living", any sqft_living value < -2800.0 and > 18000.0 are outliers. Again, any negative numbers, we can sort of ignore, unless negative values start appearing on our histogram plot.

Let's remove the outliers.

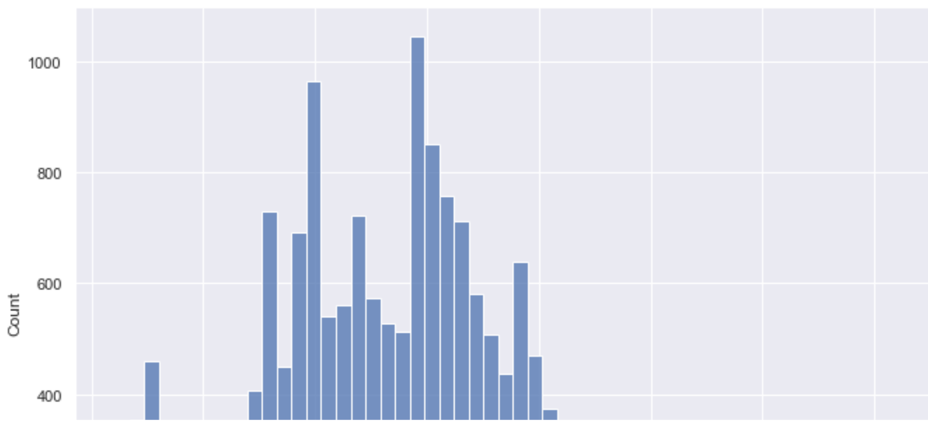
In [41]:
```
df_outliers_rmvd = df_outliers_rmvd[df_outliers_rmvd["sqft_lot"] <= 18000]
df_outliers_rmvd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18032 entries, 0 to 21612
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   sqft_lot     18032 non-null  int64  
 1   sqft_living  18032 non-null  int64  
 2   grade        18032 non-null  int64  
 3   condition    18032 non-null  int64  
 4   bathrooms    18032 non-null  float64
 5   bedrooms     18032 non-null  int64  
 6   waterfront   18032 non-null  int64  
 7   price        18032 non-null  int32  
 8   floors       18032 non-null  float64
 9   lat          18032 non-null  float64
 10  long         18032 non-null  float64
dtypes: float64(4), int32(1), int64(6)
memory usage: 1.6 MB
```

Let's see the new histogram plot.

In [42]:
```
sns.histplot(df_outliers_rmvd["sqft_lot"])
```

Out[42]: <AxesSubplot:xlabel='sqft_lot', ylabel='Count'>



Still a bit crude but we can work with that for now.

And since lot_sqftspace is a bit difficult to discern for a general correlation, we might just scratch the feature altogether towards the end.

1.4.0.4 Bedrooms

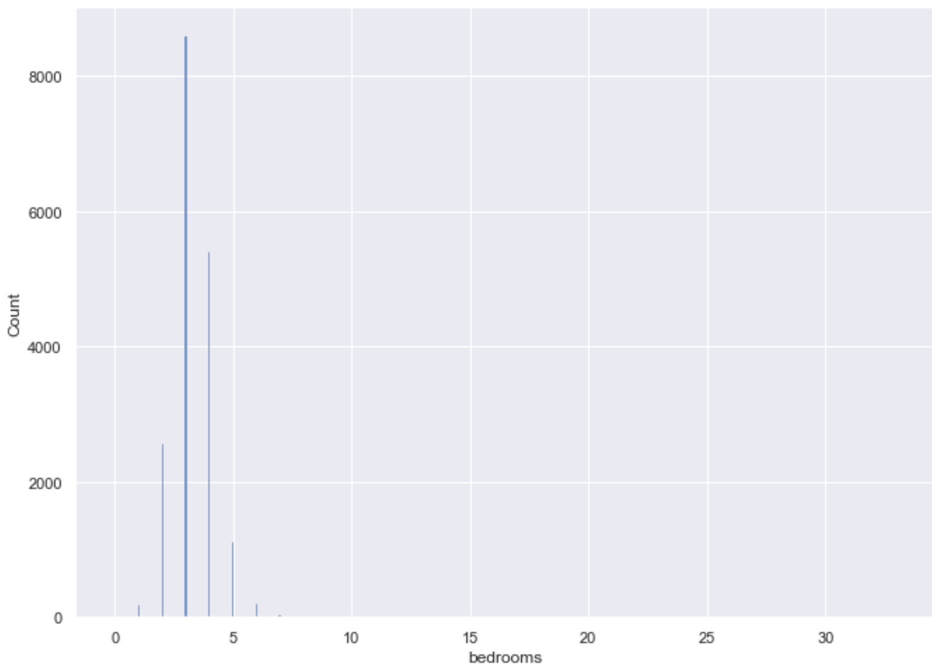
In [43]:
```
df_outliers_rmvd.bedrooms.describe()
```

Out[43]:
```
count    18032.000000
mean         3.300909
std          0.907553
min          0.000000
25%          3.000000
50%          3.000000
75%          4.000000
max         33.000000
Name: bedrooms, dtype: float64
```

Visual Plot: Initial Look

In [44]:
```
sns.histplot(df_outliers_rmvd["bedrooms"])
```

Out[44]:
```
<AxesSubplot:xlabel='bedrooms', ylabel='Count'>
```



Now, trying to take out the outliers to hopefully normalize the distribution.

In [45]:
```
q3, q1 = np.percentile(df_outliers_rmvd["bedrooms"], [75 ,25])
iqr = q3 - q1
print("iqr=", iqr)
print("q3=", q3)
print("q1=", q1)
print("constant=", iqr*1.5)
```

```
iqr= 1.0
q3= 4.0
q1= 3.0
constant= 1.5
```

In [46]:
```
print("suspected outliers are greater than this number:", q3+(iqr*1.5))
print("suspected outliers are less than this number", q1-(iqr*1.5))
```

```
suspected outliers are greater than this number: 5.5
suspected outliers are less than this number 1.5
```

So regarding "sqft_living", any sqft_living value < 1.5 and > 5.5 are outliers. Again, any negative numbers, we can sort of ignore, unless negative values start appearing on our histogram plot.

Let's remove the outliers.

In [47]:
```
df_outliers_rmvd = df_outliers_rmvd[df_outliers_rmvd["bedrooms"]<= 5.5]
df_outliers_rmvd = df_outliers_rmvd[df_outliers_rmvd["bedrooms"]>= 1.5]
df_outliers_rmvd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17627 entries, 0 to 21612
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   sqft_lot     17627 non-null  int64  
 1   sqft_living  17627 non-null  int64  
 2   grade        17627 non-null  int64  
 3   condition    17627 non-null  int64  
 4   bathrooms    17627 non-null  float64
 5   bedrooms     17627 non-null  int64  
 6   waterfront   17627 non-null  int64  
 7   price        17627 non-null  int32  
 8   floors       17627 non-null  float64
 9   lat          17627 non-null  float64
 10  long         17627 non-null  float64
dtypes: float64(4), int32(1), int64(6)
memory usage: 1.5 MB
```

We have to double check that both portions of the range were kept and not discarded.

In [48]:
```
df_outliers_rmvd.loc[df_outliers_rmvd["bedrooms"] <= 5.5]
```

	sqft_lot	sqft_living	grade	condition	bathrooms	bedrooms	waterfront	price	floors	lat	long
0	5650	1180	7	3	1.00	3	0	221900	1.0	47.5112	-122.257
1	7242	2570	7	3	2.25	3	0	538000	2.0	47.7210	-122.319
2	10000	770	6	3	1.00	2	0	180000	1.0	47.7379	-122.233
3	5000	1960	7	5	3.00	4	0	604000	1.0	47.5208	-122.393
4	8080	1680	8	3	2.00	3	0	510000	1.0	47.6168	-122.045
...	...	...	...	...	...	...	...	...	...	...	...
21608	1131	1530	8	3	2.50	3	0	360000	3.0	47.6993	-122.346
21609	5813	2310	8	3	2.50	4	0	400000	2.0	47.5107	-122.362
21610	1350	1020	7	3	0.75	2	0	402101	2.0	47.5944	-122.299
21611	2388	1600	8	3	2.50	3	0	400000	2.0	47.5345	-122.069
21612	1076	1020	7	3	0.75	2	0	325000	2.0	47.5941	-122.299

17627 rows × 11 columns

In [49]:
```
df_outliers_rmvd.loc[df_outliers_rmvd["bedrooms"] >= 1.5]
```

Out[49]:

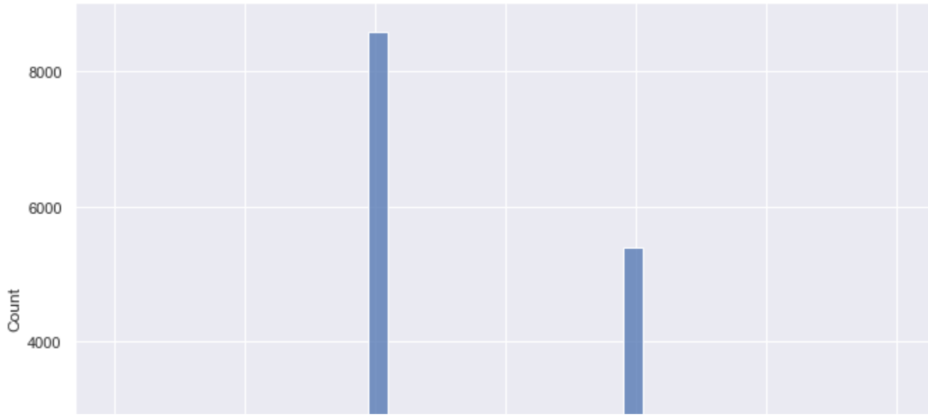
	sqft_lot	sqft_living	grade	condition	bathrooms	bedrooms	waterfront	price	floors	lat	long
0	5650	1180	7	3	1.00	3	0	221900	1.0	47.5112	-122.257
1	7242	2570	7	3	2.25	3	0	538000	2.0	47.7210	-122.319
2	10000	770	6	3	1.00	2	0	180000	1.0	47.7379	-122.233
3	5000	1960	7	5	3.00	4	0	604000	1.0	47.5208	-122.393
4	8080	1680	8	3	2.00	3	0	510000	1.0	47.6168	-122.045
...	...	...	...	...	...	...	...	...	...	...	...
21608	1131	1530	8	3	2.50	3	0	360000	3.0	47.6993	-122.346
21609	5813	2310	8	3	2.50	4	0	400000	2.0	47.5107	-122.362
21610	1350	1020	7	3	0.75	2	0	402101	2.0	47.5944	-122.299
21611	2388	1600	8	3	2.50	3	0	400000	2.0	47.5345	-122.069
21612	1076	1020	7	3	0.75	2	0	325000	2.0	47.5941	-122.299

17627 rows × 11 columns

Let's see the new histogram plot.

In [50]:
```
sns.histplot(df_outliers_rmvd["bedrooms"])
```

Out[50]: <AxesSubplot:xlabel='bedrooms', ylabel='Count'>



Still a bit crude but we can work with that for now.

Very crude correlation and normal distribution curve.

1.4.0.5 Bathrooms

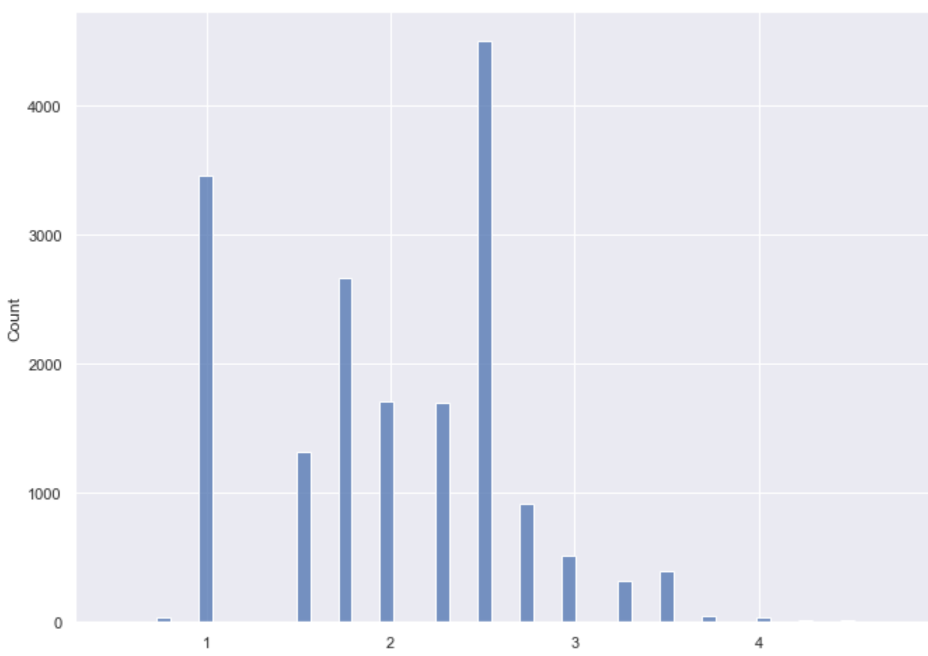
In [51]:
```
df_outliers_rmvd.bathrooms.describe()
```

Out[51]:
```
count    17627.000000
mean         2.012779
std          0.675859
min          0.500000
25%          1.500000
50%          2.000000
75%          2.500000
max          4.750000
Name: bathrooms, dtype: float64
```

Visual Plot: Initial Look

In [52]:
```
sns.histplot(df_outliers_rmvd["bathrooms"])
```

Out[52]: <AxesSubplot:xlabel='bathrooms', ylabel='Count'>



Now, trying to take out the outliers to hopefully normalize the distribution.

In [53]:
```
q3, q1 = np.percentile(df_outliers_rmvd["bathrooms"], [75 ,25])
iqr = q3 - q1
print("iqr=", iqr)
print("q3=", q3)
print("q1=", q1)
print("constant=", iqr*1.5)
```

```
iqr= 1.0
q3= 2.5
q1= 1.5
constant= 1.5
```

In [54]:
```
print("suspected outliers are greater than this number:", q3+(iqr*1.5))
print("suspected outliers are less than this number", q1-(iqr*1.5))
```

```
suspected outliers are greater than this number: 4.0
suspected outliers are less than this number 0.0
```

So regarding "sqft_living", any sqft_living value < 0 and > 4 are outliers. Again, any negative numbers, we can sort of ignore, unless negative values start appearing on our histogram plot.

Let's remove the outliers.

In [55]:
```
df_outliers_rmvd = df_outliers_rmvd[df_outliers_rmvd["bathrooms"] <= 4]
df_outliers_rmvd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17604 entries, 0 to 21612
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   sqft_lot     17604 non-null  int64  
 1   sqft_living  17604 non-null  int64  
 2   grade        17604 non-null  int64  
 3   condition    17604 non-null  int64  
 4   bathrooms    17604 non-null  float64
 5   bedrooms     17604 non-null  int64  
 6   waterfront   17604 non-null  int64  
 7   price        17604 non-null  int32  
 8   floors       17604 non-null  float64
 9   lat          17604 non-null  float64
 10  long         17604 non-null  float64
dtypes: float64(4), int32(1), int64(6)
memory usage: 1.5 MB
```

Let's see the new histogram plot.

In [56]:
```
sns.histplot(df_outliers_rmvd["bathrooms"])
```

Out[56]:
```
<AxesSubplot:xlabel='bathrooms', ylabel='Count'>
```

Still a bit crude but we can work with that for now.

Very crude correlation as well.

1.4.0.6 Grade

Now grade is one of those that need not remove outliers because we just need to understand what grade homes is considered more expensive. So just a correlation will do.

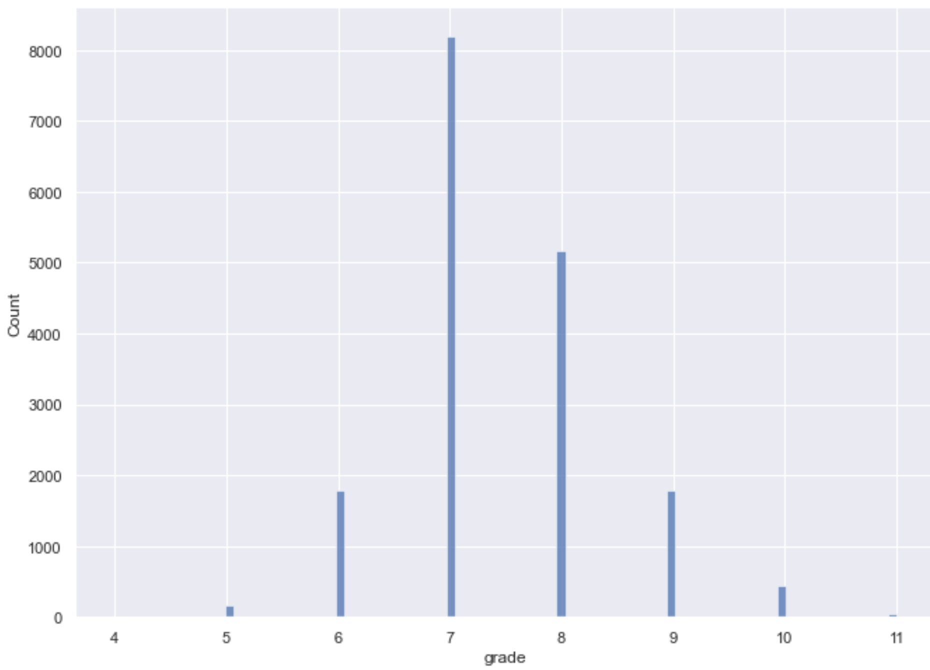
In [57]:
```
df_outliers_rmvd.grade.describe()
```

Out[57]:
```
count    17604.000000
mean         7.460691
std          0.948117
min          4.000000
25%          7.000000
50%          7.000000
75%          8.000000
max         11.000000
Name: grade, dtype: float64
```

Visual Plot: Initial Look

In [58]:
```
sns.histplot(df_outliers_rmvd["grade"])
```

Out[58]:
```
<AxesSubplot:xlabel='grade', ylabel='Count'>
```



A bit crude, but we will work that in with price later.

1.4.0.7 Condition

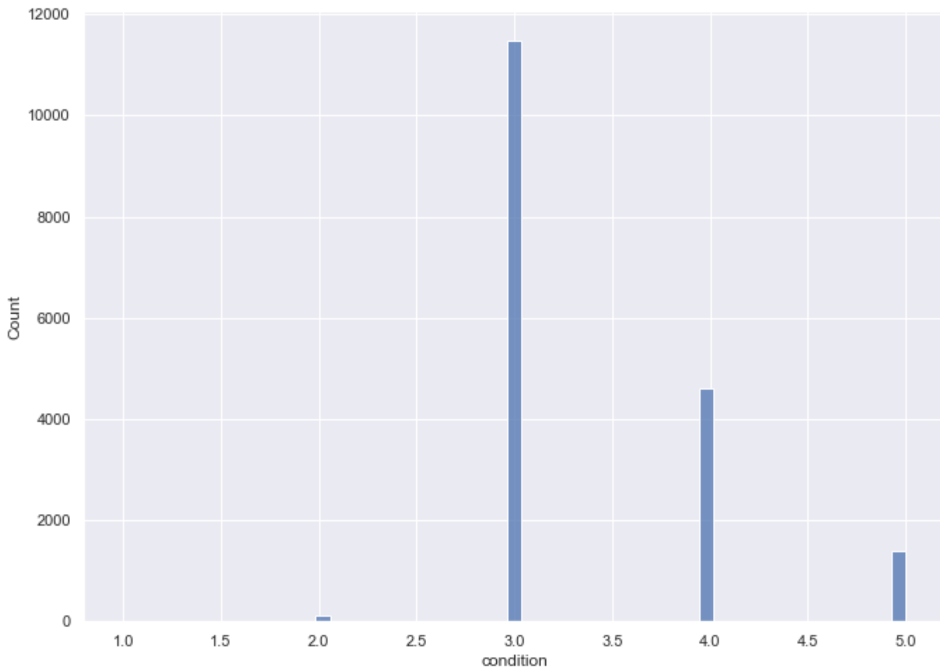
In [59]:
```
df_outliers_rmvd.condition.describe()
```

Out[59]:
```
count    17604.000000
mean         3.411611
std          0.648609
min          1.000000
25%          3.000000
50%          3.000000
75%          4.000000
max          5.000000
Name: condition, dtype: float64
```

Visual Plot: Initial Look

In [60]:
```
sns.histplot(df_outliers_rmvd["condition"])
```

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



Still a bit crude but we can work with that for now.

1.4.0.8 Floors

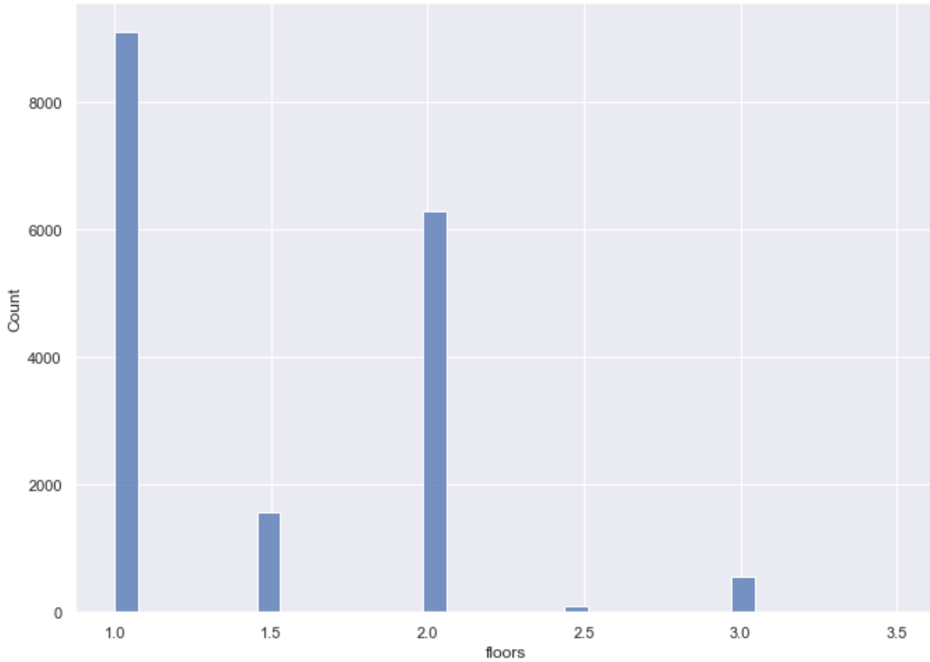
In [61]:
```
df_outliers_rmvd.floors.describe()
```

Out[61]:
```
count    17604.000000
mean         1.473671
std          0.543801
min          1.000000
25%          1.000000
50%          1.000000
75%          2.000000
max          3.500000
Name: floors, dtype: float64
```

Visual Plot: Initial Look

In [62]:
```
sns.histplot(df_outliers_rmvd["floors"])
```

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

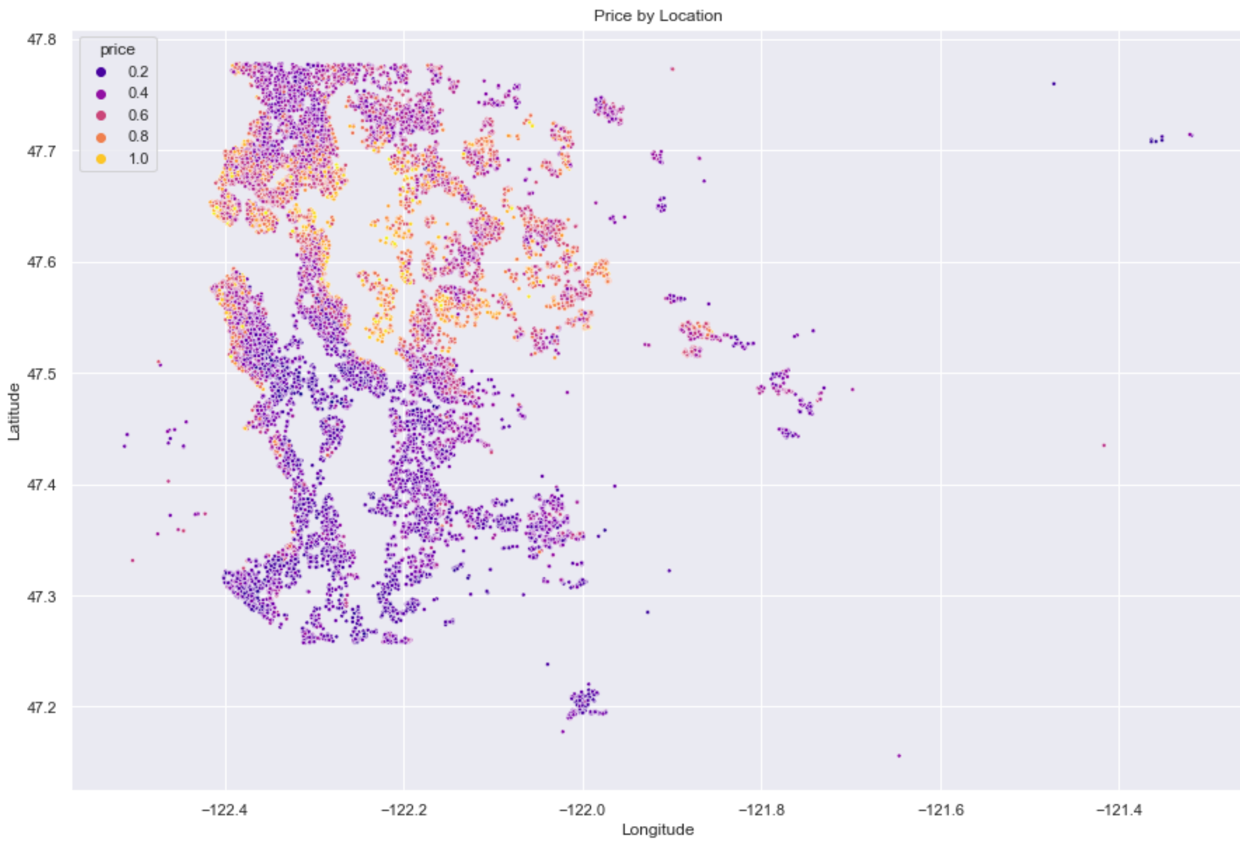


No real correlation yet til we match with price.

1.4.0.9 Location

In [63]:
```
fig = plt.figure(figsize=(15,10))
ax = sns.scatterplot(x=df_outliers_rmvd["long"], y=df_outliers_rmvd["lat"], hue=df_outliers_rmvd["price"], palette="plasma
                     marker=".")
ax.set( xlabel="Longitude",
        ylabel="Latitude",
        title="Price by Location")
```

Out[63]:
```
[Text(0.5, 0, 'Longitude'),
 Text(0, 0.5, 'Latitude'),
 Text(0.5, 1.0, 'Price by Location')]
```



Seems there is a general area from 47.55 North latitude to 47.7 North latitude, where most of the most expensive properties are located.

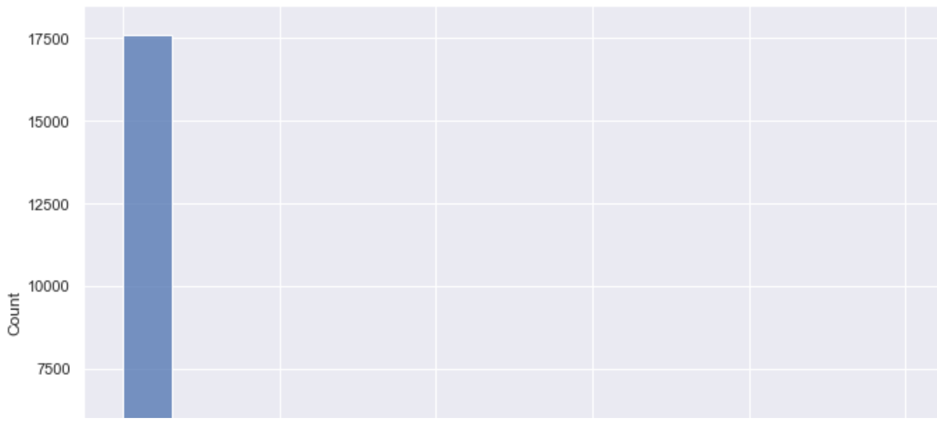
1.4.0.10 Waterfront

In [64]:
```
df_outliers_rmvd.waterfront.describe()
```

Out[64]:
```
count    17604.000000
mean         0.001591
std          0.039851
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: waterfront, dtype: float64
```

In [65]:
```
sns.histplot(df_outliers_rmvd["waterfront"])
```

Out[65]:
```
<AxesSubplot:xlabel='waterfront', ylabel='Count'>
```



For our analysis, we will exclude waterfront as a feature because it doesn't show discernibility, that it would impact price. Perhaps, with the removal of outliers, has skewed the model towards homes without waterfronts and it would be interesting to see the effect a waterfront has on the price. My prior limited background knowledge agrees with the fact that a waterfront property would be more expensive than a similar property without one.

But right now that is my speculation.

1.4.1 Looking at Multicolinearity

In [66]:
```
corr_matrix = df_outliers_rmvd.corr()
print(corr_matrix["price"].sort_values(ascending=False))
```

```
price          1.000000
grade          0.592966
sqft_living    0.579716
lat            0.459020
bathrooms      0.403435
bedrooms       0.269527
floors         0.248990
condition      0.055789
waterfront     0.049695
long           0.019040
sqft_lot      -0.026175
Name: price, dtype: float64
```

Living area and grade have the highest correlations with price. Latitude visually showed more promise as a feature with a high correlation to price of the home.

1.5 Data Modeling

Let's prepare a model and see where our features are at.

In [67]:
```
df_outliers_rmvd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17604 entries, 0 to 21612
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   sqft_lot     17604 non-null  int64
 1   sqft_living  17604 non-null  int64
 2   grade        17604 non-null  int64
 3   condition    17604 non-null  int64
 4   bathrooms    17604 non-null  float64
 5   bedrooms     17604 non-null  int64
 6   waterfront   17604 non-null  int64
 7   price        17604 non-null  int32
 8   floors       17604 non-null  float64
 9   lat          17604 non-null  float64
 10  long         17604 non-null  float64
dtypes: float64(4), int32(1), int64(6)
memory usage: 1.5 MB
```

1.5.0.1 Model 0

In [68]:
```
X = df_outliers_rmvd.drop("price", 1)
y = df_outliers_rmvd["price"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=11)
```

In [69]:
```
predictors = sm.add_constant(X_train)
model_0 = sm.OLS(y_train , predictors).fit()
model_0.summary()
```

Df Residuals:	14072	**BIC:**	3.699e+05
Df Model:	10		
Covariance Type:	nonrobust		

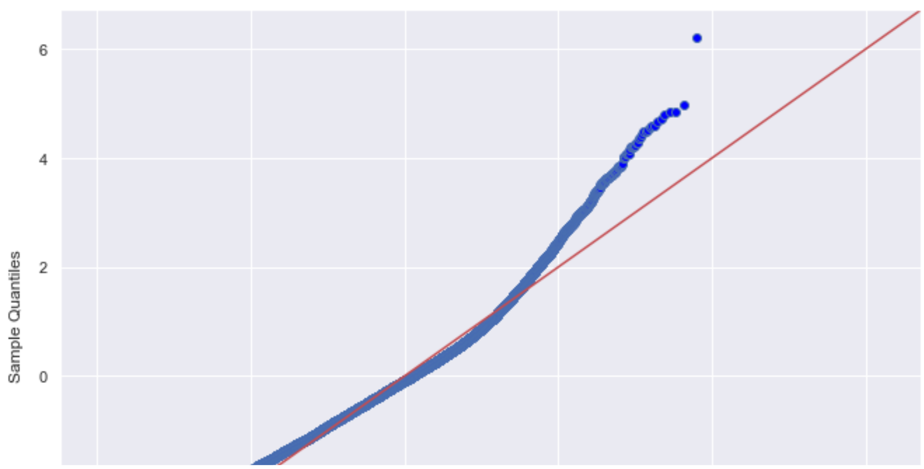
	coef	std err	t	P>|t|	[0.025	0.975]
const	-3.688e+07	1.05e+06	-35.250	0.000	-3.89e+07	-3.48e+07
sqft_lot	-5.5680	0.368	-15.118	0.000	-6.290	-4.846
sqft_living	132.9932	2.753	48.301	0.000	127.596	138.390
grade	7.466e+04	1608.671	46.414	0.000	7.15e+04	7.78e+04
condition	4.359e+04	1680.211	25.943	0.000	4.03e+04	4.69e+04
bathrooms	-1.262e+04	2407.930	-5.241	0.000	-1.73e+04	-7900.940
bedrooms	-1.124e+04	1707.328	-6.583	0.000	-1.46e+04	-7892.602
waterfront	3.334e+05	2.67e+04	12.480	0.000	2.81e+05	3.86e+05
floors	-1.127e+04	2598.797	-4.337	0.000	-1.64e+04	-6177.968

R-Squred value is decent - An R^2 of 1 indicates that the regression predictions perfectly fit the data. Near zero p-values indicated strong evidence that the null hypothesis be rejected. **High Condition number**... something to watch out for too.

In [70]:
```
lr= LinearRegression()
lr.fit(X_train, y_train)

# Use Linear Regression to make predictions for train and test data
y_hat_train = lr.predict(X_train)
y_hat_test = lr.predict(X_test)


# Calculate Root Mean Square Error
train_rmse = np.sqrt(mean_squared_error(y_train, y_hat_train))
test_rmse = np.sqrt(mean_squared_error(y_test, y_hat_test))

# Calculate Mean Absolute Error
test_mae = mean_absolute_error(y_test, y_hat_test)
train_mae = mean_absolute_error(y_train, y_hat_train)

print(f"Train Root Mean Square Error: {train_rmse}")
print(f"Test Root Mean Square Error: {test_rmse}")

print(f"Train Mean Absolute Error: {train_mae}")
print(f"Test Mean Absolute Error: {test_mae}")
```

```
Train Root Mean Square Error: 121883.44785013789
Test Root Mean Square Error: 118963.45174516343
Train Mean Absolute Error: 91959.1544447933
Test Mean Absolute Error: 89819.42589698173
```

In [71]:
```
fig = sm.graphics.qqplot(model_0.resid, dist=stats.norm, line='45', fit=True)
```



This residual plot is not all that good, room for improvement.

1.5.0.2 Model 1.0

The main goal of this model is to see if scaling helps in any way.

In [72]:
```
price_log = np.log(df_outliers_rmvd.price)
price_log = pd.DataFrame(price_log)
```

In [73]:
```
X1 = df_outliers_rmvd.drop('price', 1)
y1 =price_log
```

In [74]:
```
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size=0.2, random_state=11)
```

In [75]:
```
scaler = StandardScaler()
scalerp = StandardScaler()

X_train1[["sqft_lot", "sqft_living", "bathrooms", "bedrooms", "floors", "grade", "lat", "long"]]  =scaler.fit_transform(X_


X_test1[["sqft_lot", "sqft_living", "bathrooms", "bedrooms", "floors", "grade", "lat", "long"]] = scaler.transform(X_test1


y_train1 = scalerp.fit_transform(pd.DataFrame(y_train1))
y_test1 = scalerp.transform(pd.DataFrame(y_test1))
```

```
<ipython-input-75-5824b7b0065a>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  X_train1[["sqft_lot", "sqft_living", "bathrooms", "bedrooms", "floors", "grade", "lat", "long"]]  =scaler.fit_transfor
m(X_train1[["sqft_lot", "sqft_living", "bathrooms", "bedrooms", "floors", "grade", "lat", "long"]])
C:\Users\bigbenx3\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexing.py:1738: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  self._setitem_single_column(loc, value[:, i].tolist(), pi)
<ipython-input-75-5824b7b0065a>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  X_test1[["sqft_lot", "sqft_living", "bathrooms", "bedrooms", "floors", "grade", "lat", "long"]] = scaler.transform(X_t
est1[["sqft_lot", "sqft_living", "bathrooms", "bedrooms", "floors", "grade", "lat", "long"]])
C:\Users\bigbenx3\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexing.py:1738: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  self._setitem_single_column(loc, value[:, i].tolist(), pi)
```

In [76]:
```
predictors = sm.add_constant(X_train1)
model_1 = sm.OLS(y_train1 , predictors).fit()
model_1.summary()
```

Out[76]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.672
Model:	OLS	**Adj. R-squared:**	0.672
Method:	Least Squares	**F-statistic:**	2889.
Date:	Wed, 05 May 2021	**Prob (F-statistic):**	0.00
Time:	14:06:05	**Log-Likelihood:**	-12123.
No. Observations:	14083	**AIC:**	2.427e+04
Df Residuals:	14072	**BIC:**	2.435e+04
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.7137	0.027	-26.085	0.000	-0.767	-0.660
sqft_lot	-0.1061	0.006	-17.893	0.000	-0.118	-0.094
sqft_living	0.4098	0.009	46.696	0.000	0.393	0.427
grade	0.3318	0.007	46.115	0.000	0.318	0.346
condition	0.2084	0.008	26.415	0.000	0.193	0.224
bathrooms	0.0009	0.008	0.112	0.911	-0.014	0.016
bedrooms	-0.0312	0.006	-4.952	0.000	-0.044	-0.019
waterfront	1.5593	0.125	12.432	0.000	1.313	1.805
floors	-0.0254	0.007	-3.825	0.000	-0.038	-0.012
lat	0.4744	0.005	95.280	0.000	0.465	0.484
long	-0.0233	0.005	-4.411	0.000	-0.034	-0.013

Omnibus:	192.624	**Durbin-Watson:**	1.993
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	335.541
Skew:	0.084	**Prob(JB):**	1.37e-73
Kurtosis:	3.737	**Cond. No.**	93.9

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

The issue with the condition number is gone. And r-squared has jumped from 63% to 67%. So, that's sort of the good news.

The bad news: the r-squared is still too low.

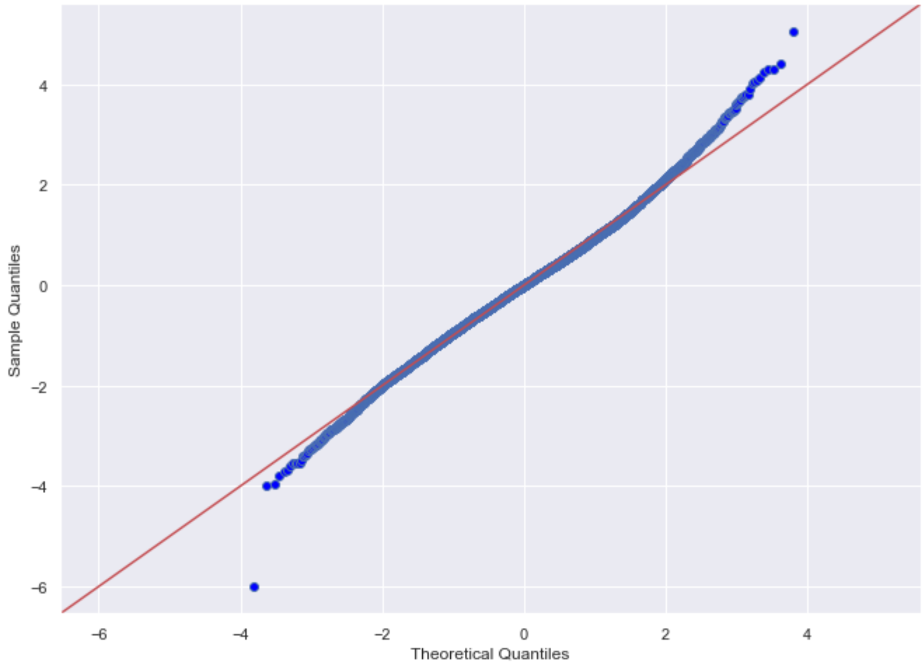
In [77]:
```
lr1= LinearRegression()
lr1.fit(X_train1, y_train1)


# Use Linear Regression to make predictions for train and test data
y_hat_train = lr1.predict(X_train1)
y_hat_test = lr1.predict(X_test1)



# Undo scale
y_train1 = scalerp.inverse_transform(y_train1)
y_test1 = scalerp.inverse_transform(y_test1)
y_hat_train = scalerp.inverse_transform(y_hat_train)
y_hat_test = scalerp.inverse_transform(y_hat_test)

# Undo log
y_train1 = np.exp(y_train1)
y_test1 = np.exp(y_test1)
y_hat_train = np.exp(y_hat_train)
y_hat_test = np.exp(y_hat_test)


# Calculate Root Mean Square Error
train_rmse1 = np.sqrt(mean_squared_error(y_train1, y_hat_train))
test_rmse1 = np.sqrt(mean_squared_error(y_test1, y_hat_test))

# Calculate Mean Absolute Error
test_mae1 = mean_absolute_error(y_test1, y_hat_test)
train_mae1 = mean_absolute_error(y_train1, y_hat_train)

print(f'Train Root Mean Square Error: {train_rmse1}')
print(f'Test Root Mean Square Error: {test_rmse1}')

print(f'Train Mean Absolute Error: {train_mae1}')
print(f'Test Mean Absolute Error: {test_mae1}')
```

```
Train Root Mean Square Error: 124488.97003196516
Test Root Mean Square Error: 120574.03660898627
Train Mean Absolute Error: 89731.55925557266
Test Mean Absolute Error: 86592.15718283084
```

In [78]:
```
y_hat_test
```

Out[78]:
```
array([[266679.09022698],
       [692917.92043294],
       [229329.63677108],
       ...,
       [442731.15968569],
       [459876.7656439 ],
       [609332.96175538]])
```

In [79]:
```
fig = sm.graphics.qqplot(model_1.resid, dist=stats.norm, line='45', fit=True)
```



So here's the dilemma: we don't want a model to be too fitted, overfitted, because then it really isn't any use as a model to predict. It's nothing more than a glorified calculator that spit out calculations and numbers for existing data.

However, we want it to have some degree of fit to the line so that it CAN be used as a model.

A happy medium somewhere in there...

In [80]:
```
results = [ ['Model 0', train_rmse, test_rmse, train_mae, test_mae],
            ['Model 1',train_rmse1, test_rmse1, train_mae1, test_mae1]]

df_results = pd.DataFrame(results, columns=['Model', 'Train RMSE', 'Test RMSE', 'Train MAE', 'Test MAE'])
df_results
```

Out[80]:

	Model	Train RMSE	Test RMSE	Train MAE	Test MAE
0	Model 0	121883.447850	118963.451745	91959.154445	89819.425897
1	Model 1	124488.970032	120574.036609	89731.559256	86592.157183

1.6 CRITICAL-Model Decision

I think I'll go with model 2 because the scaling brought down the condition number, visually it was more aesthetically pleasing.

1.6.0.1 Choosing the Model

Typically lower RSME shows better fit to the line.

In [81]:
```
Xf = df_outliers_rmvd.drop('price', 1)

scalerf= StandardScaler()

Xf[["sqft_lot", "sqft_living", "bathrooms", "bedrooms", "floors", "grade", "condition", "lat", "long"]]  =scalerf.fit_trar

scalerfp = StandardScaler()

price_sc = scalerp.transform(pd.DataFrame(df_outliers_rmvd.price))

y_hat = lr1.predict(Xf)

y_hat = np.exp(scalerp.inverse_transform(y_hat))

y_hat

rmse_f = np.sqrt(mean_squared_error(df_outliers_rmvd.price , y_hat))
mae_f = mean_absolute_error(df_outliers_rmvd.price, y_hat)
```

In [82]:
```
print(f'Root Mean Square Error: {rmse_f}')
print(f'Mean Absolute Error: {mae_f}')
```

```
Root Mean Square Error: 183833.88236696075
Mean Absolute Error: 140227.95035117553
```

In [83]:
```
mae_f
```

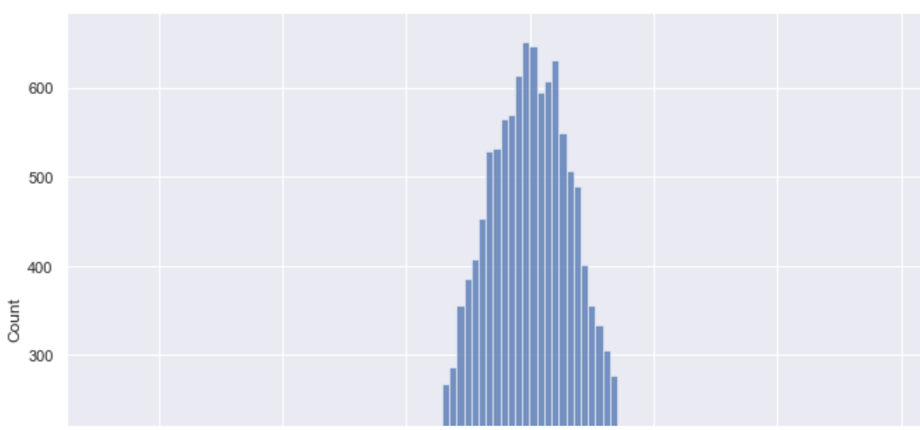
Out[83]:
```
140227.95035117553
```

In [84]:
```
mae_f/df_outliers_rmvd.price.mean()
```

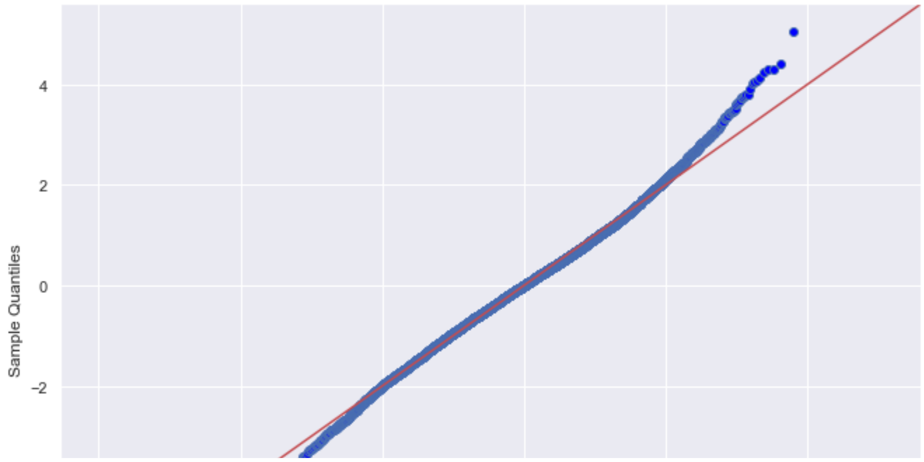
Out[84]:
```
0.3028492292660915
```

In [85]:
```
sns.histplot(model_1.resid)
```

Out[85]:
```
<AxesSubplot:ylabel='Count'>
```

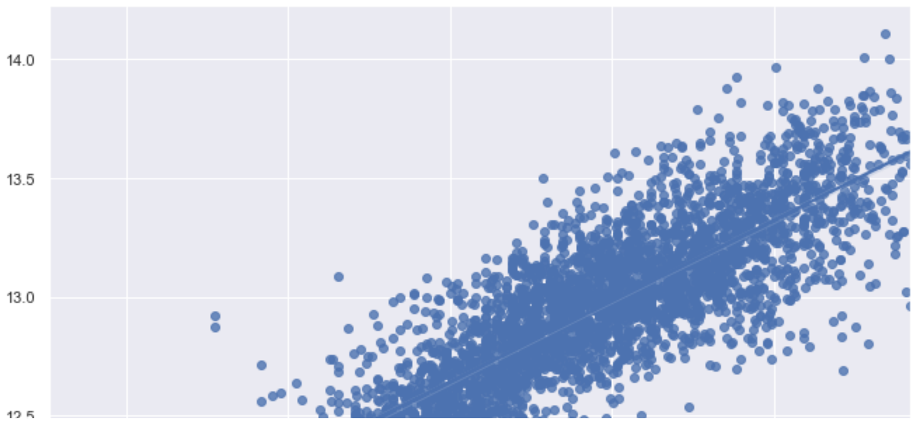


In [86]:
```
fig = sm.graphics.qqplot(model_1.resid, dist=stats.norm, line='45', fit=True)
```



In [87]:
```
sns.regplot(x=np.log(y_test1), y=np.log(y_hat_test))
```

Out[87]: <AxesSubplot:>



The majority of the plot conforms to the best fit line.

1.7 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)
- With a Mean Squared Error of around 140227 USD, that means our predicted price is, on average, 140227 USD off from our mean. While that number doesn't look too bad our Root Mean Squared Error is around 183833 USD which means that our model is being heavily penalized for predictions that are very far off the actual price.

1.8 Conclusions:

- We have a model that has an Coefficient of Determination(R-squared) value of 0.672 which indicates that our model can explain 67.2% of all variation in the data around the mean.
- With a Mean Squared Error of around 140227 USD, that means our predicted price is, on average, 140227 USD off from our mean. While that number doesn't look too bad our Root Mean Squared Error is around 183833 USD which means that our model is being heavily penalized for predictions that are very far off the actual price.

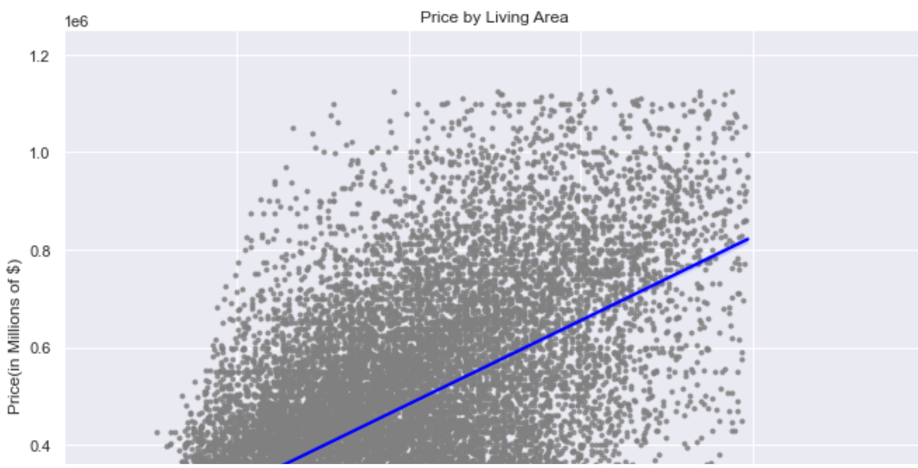
1.9 Future Research

- 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.

1.10 Presentation Prep

In [88]:
```
fig = plt.figure(figsize=(11,8))
ax = sns.regplot(data=df_outliers_rmvd, 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, 1250000])
plt.show()
```

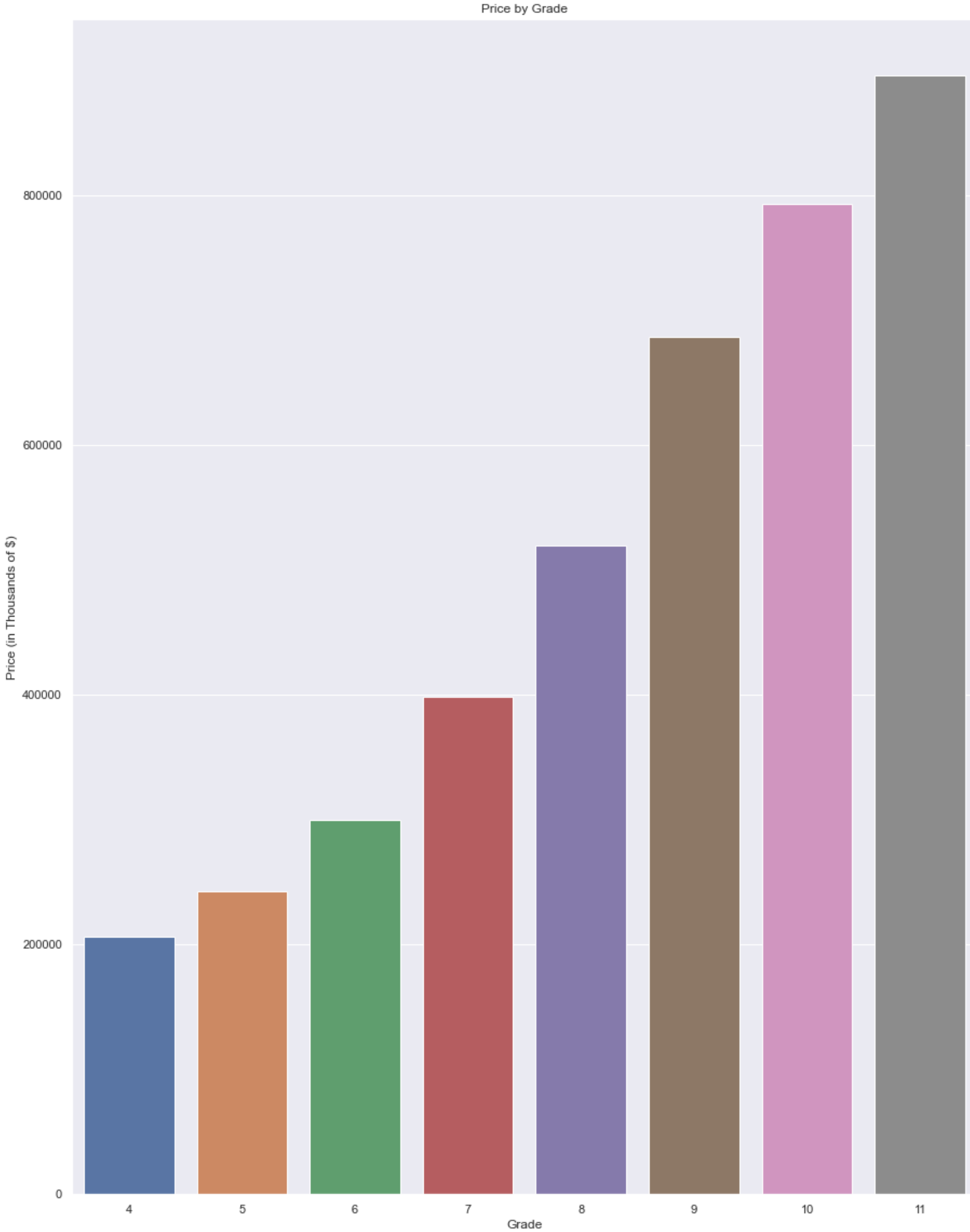


In [89]:
```
df_outliers_rmvd.grade.describe()
```

Out[89]:
```
count    17604.000000
mean         7.460691
std          0.948117
min          4.000000
25%          7.000000
50%          7.000000
75%          8.000000
max         11.000000
Name: grade, dtype: float64
```

In [90]:
```
fig = plt.figure(figsize=(15,20))
ax = sns.barplot(data=df_outliers_rmvd, x="grade", y="price", ci=None)
ax.set( xticklabels=(["4","5", "6", "7", "8", "9", "10", "11"]),
        xlabel="Grade",
        ylabel="Price (in Thousands of $)",
        title="Price by Grade"  )
```

Out[90]:
```
[[Text(0, 0, '4'),
  Text(1, 0, '5'),
  Text(2, 0, '6'),
  Text(3, 0, '7'),
  Text(4, 0, '8'),
  Text(5, 0, '9'),
  Text(6, 0, '10'),
  Text(7, 0, '11')],
 Text(0.5, 0, 'Grade'),
 Text(0, 0.5, 'Price (in Thousands of $)'),
 Text(0.5, 1.0, 'Price by Grade')]
```


Price by Grade
Price (in Thousands of $)
800000
600000
400000
200000
0
4
5
6
7
8
9
10
11
Grade

```
In [ ]:
```