

In [1]:

```
pip install pillow
```

Requirement already satisfied: pillow in c:\users\rani\_\anaconda6\lib\site-packages (9.4.0)  
Note: you may need to restart the kernel to use updated packages.

In [2]:

```
from IPython.display import Image
```

In [3]:

```
Image ("Images/Home builder.jpg" , width=1000, height=500)
```

Out[3]:



# KING COUNTY HOUSE SALES ANALYSIS REPORT

**Author:** Sneha Bhaskar

**Date:** 24 September 2023

## Overview

This project analyzes the King County House Sales Dataset to answer the business case question on what type of housing features to focus when building a house and selling in King County Washington, United States. Utilizing the OSEMN framework for data science, it appears that the most valuable feature for a Pro

Home Builder Company would be living area square footage and zipcode. Zip codes provide contextual information about property values, local demand, and building regulations, allowing builders to strategically choose locations and tailor their designs to specific demographics. Square footage, on the other hand, directly influences construction costs, selling price, and design considerations plus building grade improves, the house price rises as well

## Business Problem

The Pro Home Builder Company aims to construct residences in King County Washington, and wants to identify which housing features/areas to focus on for maximum profitability. By understanding which amenities significantly influence home prices, the company can make data-driven choices on the types of houses to build in order to optimize profits

### Import standard packages

In [4]:

```
import pandas as pd #data manipulation and analysis. Can create data frames, filter data, import numpy as np #numerical python library for array manipulation, math functions etc. import seaborn as sns #data visualization library based on Matplotlib, with higher-level import matplotlib.pyplot as plt #2D plotting lib. plt.style.use("ggplot") #changes the default style of Matplotlib plots to ggplot.

from statsmodels.formula.api import ols #ordinary least square (OLS) is a regression mode import statsmodels.api as sm #statsmodel includes various statistical tests, models and f from statsmodels.stats.outliers_influence import variance_inflation_factor #a measure to import scipy.stats as stats #statistical functions from the SciPy lib, which builds on Nu from sklearn.linear_model import LinearRegression #linear model from sklearn.model_selection import train_test_split #function for splitting datasets int from sklearn.metrics import mean_squared_error #function for computing the mean squared from sklearn.preprocessing import OneHotEncoder #preprocessing technique to convert categ from sklearn.model_selection import cross_val_score #function for evaluating a model usin from sklearn.model_selection import KFold #K-Fold cross validator from itertools import combinations #itertools is used for generating all possible combin
```

# Kings County House Dataset

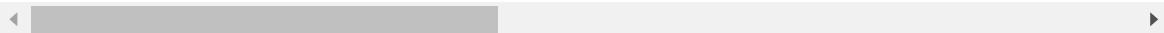
In [5]:

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

Out[5]:

|              | <b>id</b>  | <b>date</b> | <b>price</b> | <b>bedrooms</b> | <b>bathrooms</b> | <b>sqft_living</b> | <b>sqft_lot</b> | <b>floors</b> | <b>w</b> |
|--------------|------------|-------------|--------------|-----------------|------------------|--------------------|-----------------|---------------|----------|
| <b>0</b>     | 7129300520 | 10/13/2014  | 221900       | 3               | 1.00             | 1180               | 5650            | 1.0           |          |
| <b>1</b>     | 6414100192 | 12/09/2014  | 538000       | 3               | 2.25             | 2570               | 7242            | 2.0           |          |
| <b>2</b>     | 5631500400 | 2/25/2015   | 180000       | 2               | 1.00             | 770                | 10000           | 1.0           |          |
| <b>3</b>     | 2487200875 | 12/09/2014  | 604000       | 4               | 3.00             | 1960               | 5000            | 1.0           |          |
| <b>4</b>     | 1954400510 | 2/18/2015   | 510000       | 3               | 2.00             | 1680               | 8080            | 1.0           |          |
| ...          | ...        | ...         | ...          | ...             | ...              | ...                | ...             | ...           | ...      |
| <b>21592</b> | 263000018  | 5/21/2014   | 360000       | 3               | 2.50             | 1530               | 1131            | 3.0           |          |
| <b>21593</b> | 6600060120 | 2/23/2015   | 400000       | 4               | 2.50             | 2310               | 5813            | 2.0           |          |
| <b>21594</b> | 1523300141 | 6/23/2014   | 402101       | 2               | 0.75             | 1020               | 1350            | 2.0           |          |
| <b>21595</b> | 291310100  | 1/16/2015   | 400000       | 3               | 2.50             | 1600               | 2388            | 2.0           |          |
| <b>21596</b> | 1523300157 | 10/15/2014  | 325000       | 2               | 0.75             | 1020               | 1076            | 2.0           |          |

21597 rows × 21 columns



In [6]:

```
#check for duplicates
df.duplicated(keep = False).sum()
```

Out[6]:

0

## OBTAİN

This step involves understanding stakeholder requirements, gathering information on the problem, and finally sourcing data we think will be necessary for solving this problem

In [7]:

```
#Removing columns that is not required from this dataframe for our business problem
df.drop(['id'], axis= 1, inplace = True)
df.head()
```

Out[7]:

|   | date       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | c |
|---|------------|--------|----------|-----------|-------------|----------|--------|------------|------|---|
| 0 | 10/13/2014 | 221900 | 3        | 1.00      | 1180        | 5650     | 1.0    | NaN        | 0.0  |   |
| 1 | 12/09/2014 | 538000 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0.0        | 0.0  |   |
| 2 | 2/25/2015  | 180000 | 2        | 1.00      | 770         | 10000    | 1.0    | 0.0        | 0.0  |   |
| 3 | 12/09/2014 | 604000 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0.0        | 0.0  |   |
| 4 | 2/18/2015  | 510000 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0.0        | 0.0  |   |

In [8]:

```
print("The number of rows is",df.shape[0])
print('The number of columns is',df.shape[1])
```

The number of rows is 21597  
The number of columns is 20

## SCRUB

This step focuses on cleaning the data, which involves handling missing values, removing outliers, and converting data types and many more. Data cleaning is crucial because the quality of data affects the quality of the final model produced

In [9]:

```
df.info()
#all of the columns are in integer/float format except "date" and "sqft_basement" are in

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   date              21597 non-null   object  
 1   price             21597 non-null   int64  
 2   bedrooms          21597 non-null   int64  
 3   bathrooms         21597 non-null   float64 
 4   sqft_living       21597 non-null   int64  
 5   sqft_lot          21597 non-null   int64  
 6   floors            21597 non-null   float64 
 7   waterfront        19221 non-null   float64 
 8   view              21534 non-null   float64 
 9   condition         21597 non-null   int64  
 10  grade             21597 non-null   int64  
 11  sqft_above        21597 non-null   int64  
 12  sqft_basement    21597 non-null   object  
 13  yr_built          21597 non-null   int64  
 14  yr_renovated     17755 non-null   float64 
 15  zipcode           21597 non-null   int64  
 16  lat               21597 non-null   float64 
 17  long              21597 non-null   float64 
 18  sqft_living15     21597 non-null   int64  
 19  sqft_lot15        21597 non-null   int64  
dtypes: float64(7), int64(11), object(2)
memory usage: 3.3+ MB
```

In [10]:

```
df.describe()
#Bedrooms has an outliers of 33 rooms
#waterfront, condition, grade Looks like a categorical data
```

Out[10]:

|       | price        | bedrooms     | bathrooms    | sqft_living  | sqft_lot     | floors       |
|-------|--------------|--------------|--------------|--------------|--------------|--------------|
| count | 2.159700e+04 | 21597.000000 | 21597.000000 | 21597.000000 | 2.159700e+04 | 21597.000000 |
| mean  | 5.402966e+05 | 3.373200     | 2.115826     | 2080.321850  | 1.509941e+04 | 1.494096     |
| std   | 3.673681e+05 | 0.926299     | 0.768984     | 918.106125   | 4.141264e+04 | 0.539683     |
| min   | 7.800000e+04 | 1.000000     | 0.500000     | 370.000000   | 5.200000e+02 | 1.000000     |
| 25%   | 3.220000e+05 | 3.000000     | 1.750000     | 1430.000000  | 5.040000e+03 | 1.000000     |
| 50%   | 4.500000e+05 | 3.000000     | 2.250000     | 1910.000000  | 7.618000e+03 | 1.500000     |
| 75%   | 6.450000e+05 | 4.000000     | 2.500000     | 2550.000000  | 1.068500e+04 | 2.000000     |
| max   | 7.700000e+06 | 33.000000    | 8.000000     | 13540.000000 | 1.651359e+06 | 3.500000     |

In [11]:

```
#check for null values
df.isna().sum()
#waterfront and yr_renovated are the only columns with null values
```

Out[11]:

|               |       |
|---------------|-------|
| 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 |

## Fill null values for waterfront

In [12]:

```
print(df.waterfront.isnull().sum())
print(df['waterfront'].value_counts())
```

|                                |       |
|--------------------------------|-------|
| 2376                           |       |
| 0.0                            | 19075 |
| 1.0                            | 146   |
| Name: waterfront, dtype: int64 |       |

In [13]:

```
print('Mean = ', df['waterfront'].mean())
print('Median = ', df['waterfront'].median())
```

Mean = 0.007595858696217679  
Median = 0.0

In [14]:

```
# Replace all the missing values in the waterfront column with the Median
df['waterfront'].fillna(df['waterfront'].median(), inplace=True)
df['waterfront'].isnull().sum()
```

Out[14]:

0

In [15]:

df

Out[15]:

|       | date       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | v   |
|-------|------------|--------|----------|-----------|-------------|----------|--------|------------|-----|
| 0     | 10/13/2014 | 221900 | 3        | 1.00      | 1180        | 5650     | 1.0    | 0.0        | 0.  |
| 1     | 12/09/2014 | 538000 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0.0        | 0.  |
| 2     | 2/25/2015  | 180000 | 2        | 1.00      | 770         | 10000    | 1.0    | 0.0        | 0.  |
| 3     | 12/09/2014 | 604000 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0.0        | 0.  |
| 4     | 2/18/2015  | 510000 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0.0        | 0.  |
| ...   | ...        | ...    | ...      | ...       | ...         | ...      | ...    | ...        | ... |
| 21592 | 5/21/2014  | 360000 | 3        | 2.50      | 1530        | 1131     | 3.0    | 0.0        | 0.  |
| 21593 | 2/23/2015  | 400000 | 4        | 2.50      | 2310        | 5813     | 2.0    | 0.0        | 0.  |
| 21594 | 6/23/2014  | 402101 | 2        | 0.75      | 1020        | 1350     | 2.0    | 0.0        | 0.  |
| 21595 | 1/16/2015  | 400000 | 3        | 2.50      | 1600        | 2388     | 2.0    | 0.0        | 0.  |
| 21596 | 10/15/2014 | 325000 | 2        | 0.75      | 1020        | 1076     | 2.0    | 0.0        | 0.  |

21597 rows × 20 columns

Create a new column to replace the yr\_renovated where 1 = Renovated and 0 = Not renovated

In [16]:

df.yr\_renovated.mean()

Out[16]:

83.6367783722895

In [17]:

df.yr\_renovated.median()

Out[17]:

0.0

In [18]:

df.yr\_renovated.fillna(df.yr\_renovated.median(), inplace=True)  
df.yr\_renovated.isnull().sum()

Out[18]:

0

In [19]:

df['renovated'] = np.where(df['yr\_renovated'] == 0, 0, 1)

In [20]:

```
#drop the "yr_renovated" column
df.drop('yr_renovated', axis=1, inplace = True)
```

In [21]:

df

Out[21]:

|       | date       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | vintage |
|-------|------------|--------|----------|-----------|-------------|----------|--------|------------|---------|
| 0     | 10/13/2014 | 221900 | 3        | 1.00      | 1180        | 5650     | 1.0    | 0.0        | 0.      |
| 1     | 12/09/2014 | 538000 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0.0        | 0.      |
| 2     | 2/25/2015  | 180000 | 2        | 1.00      | 770         | 10000    | 1.0    | 0.0        | 0.      |
| 3     | 12/09/2014 | 604000 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0.0        | 0.      |
| 4     | 2/18/2015  | 510000 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0.0        | 0.      |
| ...   | ...        | ...    | ...      | ...       | ...         | ...      | ...    | ...        | ...     |
| 21592 | 5/21/2014  | 360000 | 3        | 2.50      | 1530        | 1131     | 3.0    | 0.0        | 0.      |
| 21593 | 2/23/2015  | 400000 | 4        | 2.50      | 2310        | 5813     | 2.0    | 0.0        | 0.      |
| 21594 | 6/23/2014  | 402101 | 2        | 0.75      | 1020        | 1350     | 2.0    | 0.0        | 0.      |
| 21595 | 1/16/2015  | 400000 | 3        | 2.50      | 1600        | 2388     | 2.0    | 0.0        | 0.      |
| 21596 | 10/15/2014 | 325000 | 2        | 0.75      | 1020        | 1076     | 2.0    | 0.0        | 0.      |

21597 rows × 20 columns

In [22]:

```
df.renovated.value_counts()
```

Out[22]:

```
0    20853
1     744
Name: renovated, dtype: int64
```

Clean "Sqft\_Basement" column by replacing '?' with 0 and converting the data type from object to float

In [23]:

```
df.sqft_basement.value_counts()
```

Out[23]:

```
0      12826  
?      454  
600    217  
500    209  
700    208  
...  
1920    1  
3480    1  
2730    1  
2720    1  
248     1  
Name: sqft_basement, Length: 304, dtype: int64
```

In [24]:

```
#replace the '?' in the basement column with 0  
df['sqft_basement'] = df['sqft_basement'].replace(to_replace='?', value='0.0')
```

In [25]:

```
df.sqft_basement.value_counts()
```

Out[25]:

```
0      12826  
0.0    454  
600    217  
500    209  
700    208  
...  
1920    1  
3480    1  
2730    1  
2720    1  
248     1  
Name: sqft_basement, Length: 304, dtype: int64
```

In [26]:

```
#convert the data type of 'sqft_basement' from object to float  
df['sqft_basement'] = df['sqft_basement'].astype(float)
```

In [27]:

```
#confirm we don't have any null values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype  
 --- 
 0   date             21597 non-null   object  
 1   price            21597 non-null   int64  
 2   bedrooms         21597 non-null   int64  
 3   bathrooms        21597 non-null   float64 
 4   sqft_living      21597 non-null   int64  
 5   sqft_lot          21597 non-null   int64  
 6   floors            21597 non-null   float64 
 7   waterfront        21597 non-null   float64 
 8   view              21534 non-null   float64 
 9   condition         21597 non-null   int64  
 10  grade             21597 non-null   int64  
 11  sqft_above        21597 non-null   int64  
 12  sqft_basement     21597 non-null   float64 
 13  yr_built          21597 non-null   int64  
 14  zipcode           21597 non-null   int64  
 15  lat                21597 non-null   float64 
 16  long               21597 non-null   float64 
 17  sqft_living15     21597 non-null   int64  
 18  sqft_lot15         21597 non-null   int64  
 19  renovated          21597 non-null   int32  
dtypes: float64(7), int32(1), int64(11), object(1)
memory usage: 3.2+ MB
```

In [28]:

```
#check the data statistics
df.describe()
```

Out[28]:

|       | price        | bedrooms     | bathrooms    | sqft_living  | sqft_lot     | floors       |
|-------|--------------|--------------|--------------|--------------|--------------|--------------|
| count | 2.159700e+04 | 21597.000000 | 21597.000000 | 21597.000000 | 2.159700e+04 | 21597.000000 |
| mean  | 5.402966e+05 | 3.373200     | 2.115826     | 2080.321850  | 1.509941e+04 | 1.494096     |
| std   | 3.673681e+05 | 0.926299     | 0.768984     | 918.106125   | 4.141264e+04 | 0.539683     |
| min   | 7.800000e+04 | 1.000000     | 0.500000     | 370.000000   | 5.200000e+02 | 1.000000     |
| 25%   | 3.220000e+05 | 3.000000     | 1.750000     | 1430.000000  | 5.040000e+03 | 1.000000     |
| 50%   | 4.500000e+05 | 3.000000     | 2.250000     | 1910.000000  | 7.618000e+03 | 1.500000     |
| 75%   | 6.450000e+05 | 4.000000     | 2.500000     | 2550.000000  | 1.068500e+04 | 2.000000     |
| max   | 7.700000e+06 | 33.000000    | 8.000000     | 13540.000000 | 1.651359e+06 | 3.500000     |



# PRICE

In [29]:

```
# Check price distribution
df['price'].div(1000000).hist(color = "yellow")
plt.title('Price Distribution');
plt.xlabel('Price - in Millions :)');
plt.ylabel('Number of Houses');

#the majority of houses are below the price of 2 million
```

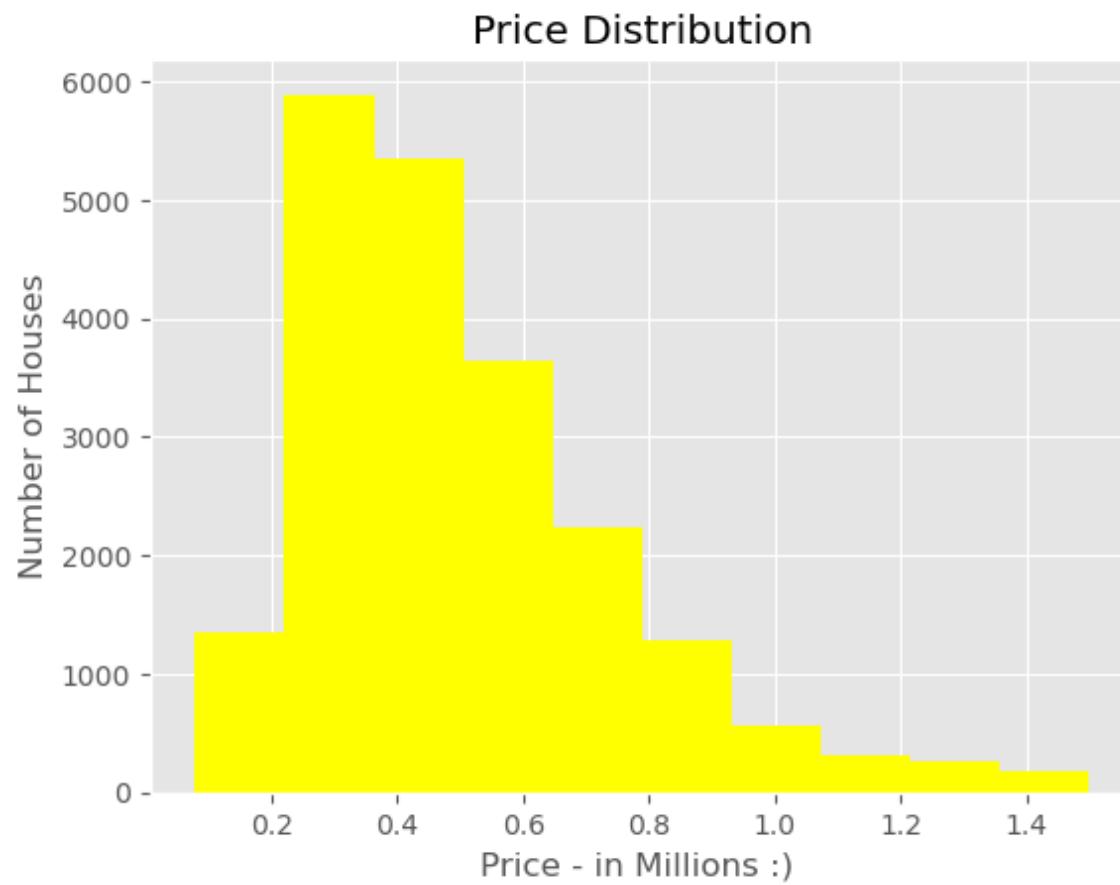


In [30]:

```
#reduce the data to only houses equal to or below 1.5 million
df_og = df.copy()
# remove houses above 1.5 million, boolean mask
df = df[df['price'] <= 1500000]
```

In [31]:

```
# RECHECK price distribution
df['price'].div(1000000).hist(color = "yellow")
plt.title('Price Distribution');
plt.xlabel('Price - in Millions :)');
plt.ylabel('Number of Houses');
```

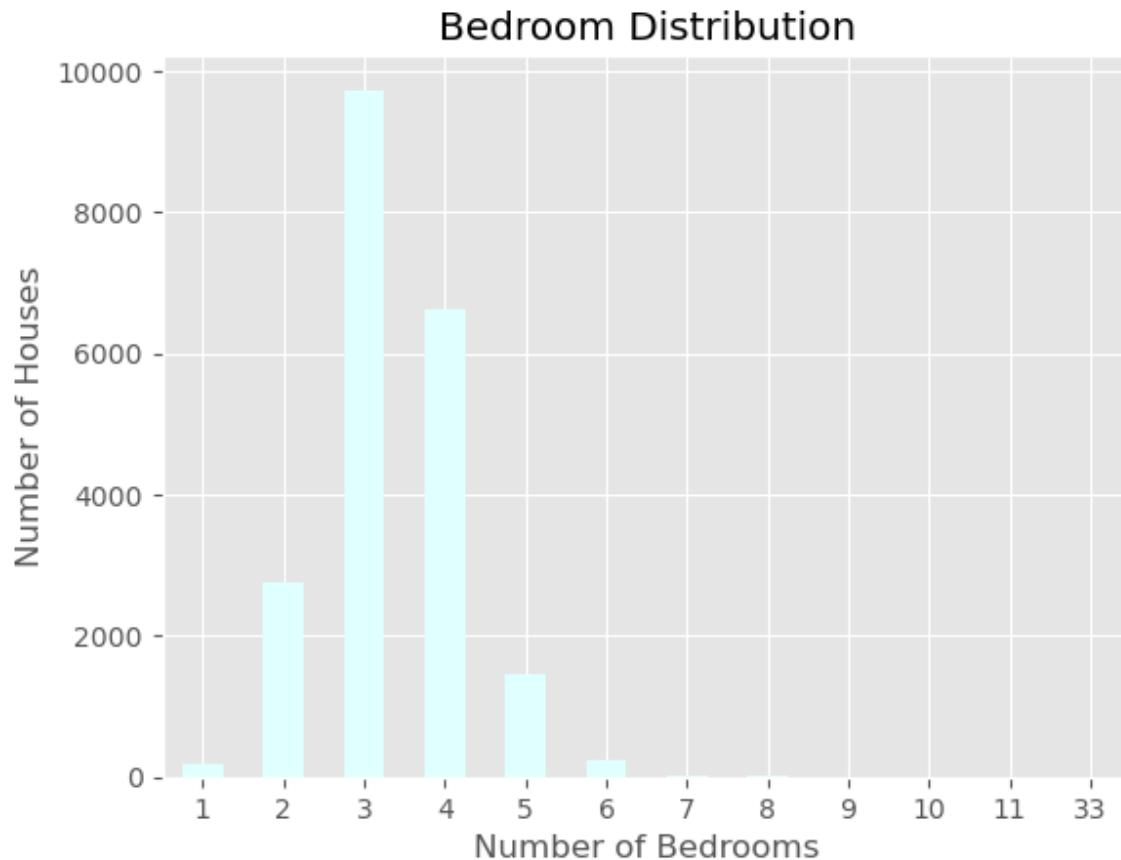


**Majority of the houses are in between 0.3 - 0.5 price range**

## BEDROOMS

In [32]:

```
# Plot bedrooms as categorical - ordinal
df['bedrooms'].value_counts().sort_index().plot(kind = 'bar', color = "lightcyan")
plt.xticks(rotation = 0);
plt.title('Bedroom Distribution');
plt.xlabel('Number of Bedrooms');
plt.ylabel('Number of Houses');
```



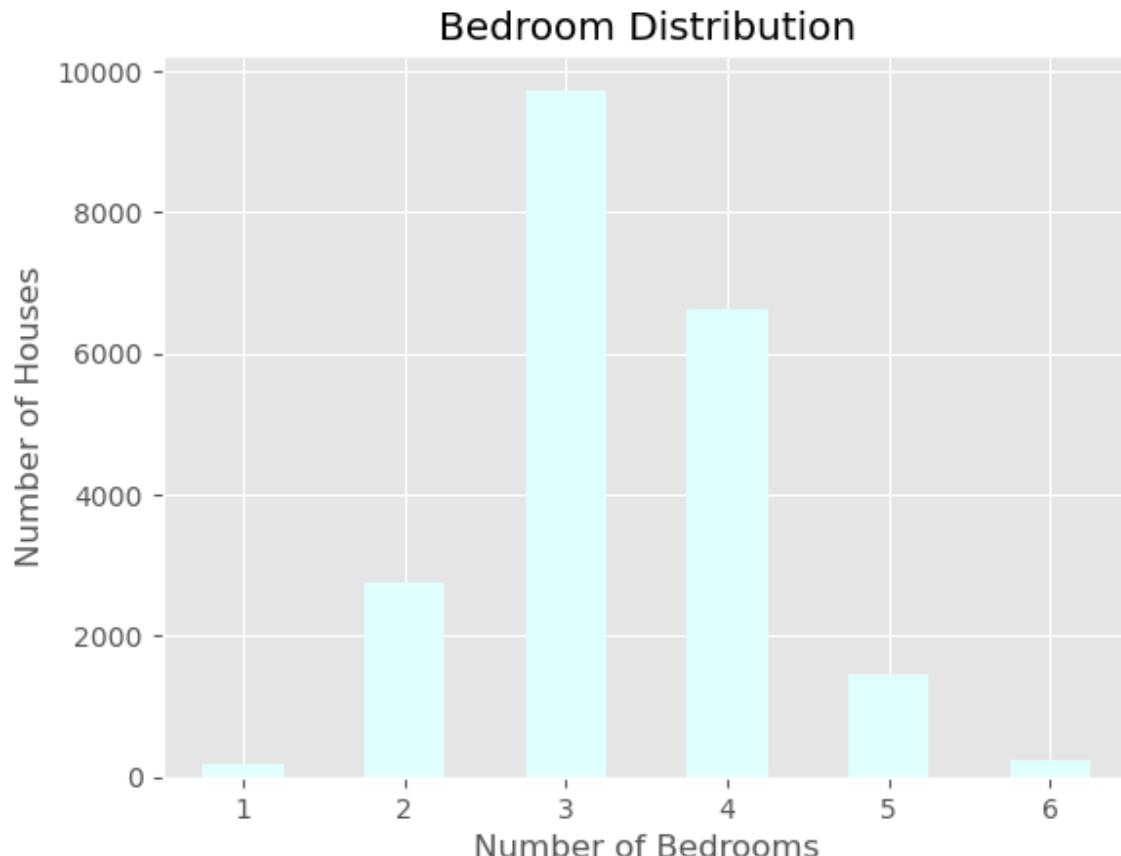
In [33]:

```
#reduce the data to only bedrooms equal to or below 6
df_og = df.copy()
# remove bedrooms above 6, boolean mask
df = df[df['bedrooms'] <=6]
```

In [34]:

```
# Plot bedrooms as categorical - ordinal
df['bedrooms'].value_counts().sort_index().plot(kind = 'bar', color = "lightcyan")
plt.xticks(rotation = 0);
plt.title('Bedroom Distribution');
plt.xlabel('Number of Bedrooms');
plt.ylabel('Number of Houses');

#by removing the 33 outliers for bedrooms , its summarises that houses with 3 or 4 bedro
```

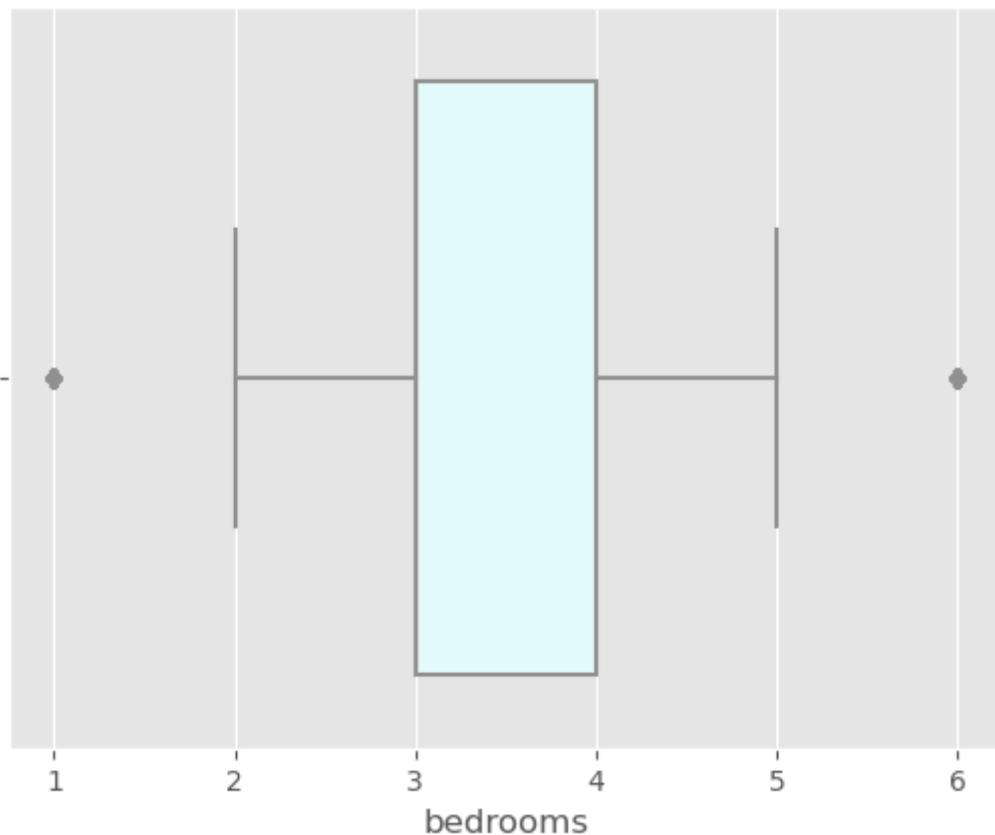


Houses with 3 or 4 bedrooms are the most common in the dataset

In [35]:

```
sns.boxplot(x = df['bedrooms'], color = "lightcyan");
```

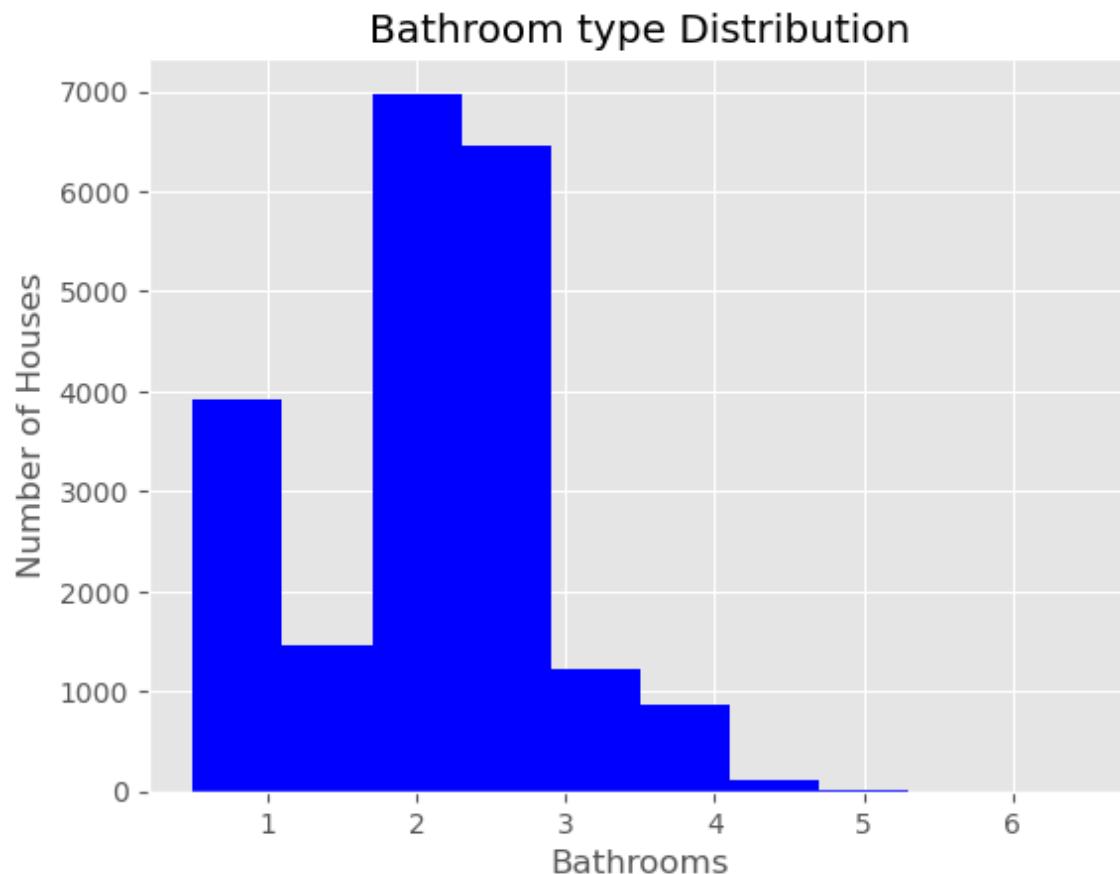
#Boxplot shows are normal distribution



## BATHROOMS

In [36]:

```
# Check bathroom types distribution
df['bathrooms'].hist(color = "BLUE")
plt.title('Bathroom type Distribution');
plt.xlabel('Bathrooms');
plt.ylabel('Number of Houses');
```

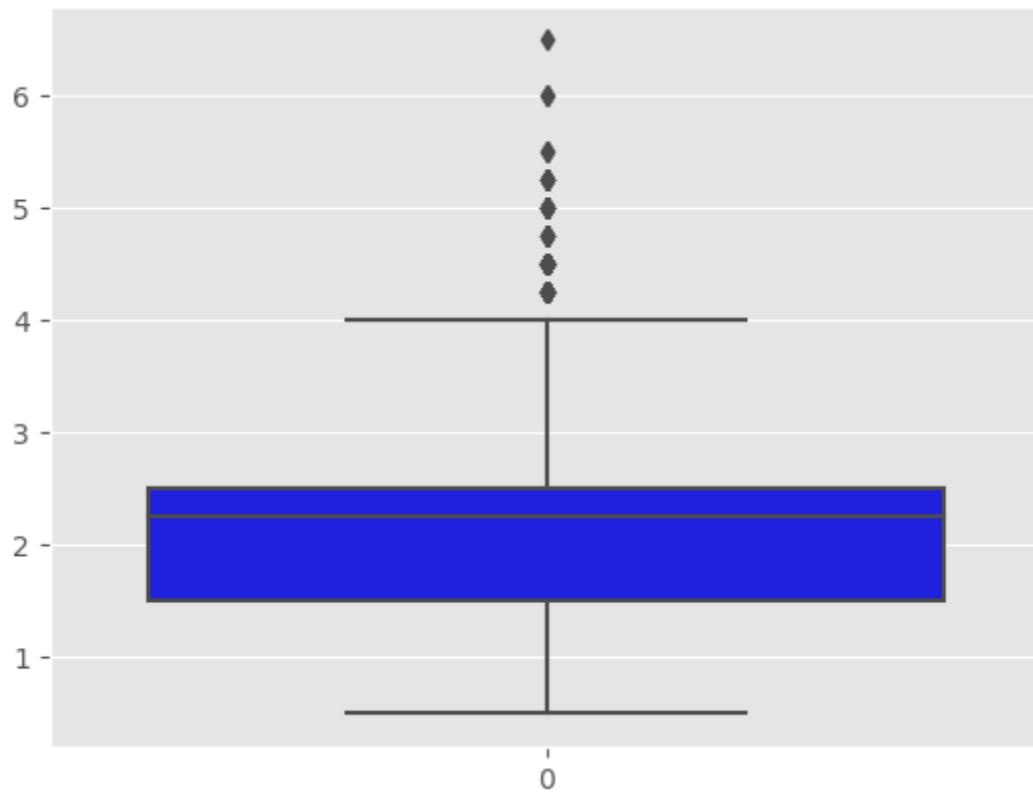


There appears to be an outlier and most of the houses have 2 or 3 bathrooms

In [37]:

```
# Boxplot to detect outliers
sns.boxplot(df['bathrooms'], color = "BLUE");

#there are handful of outliers above 4
```



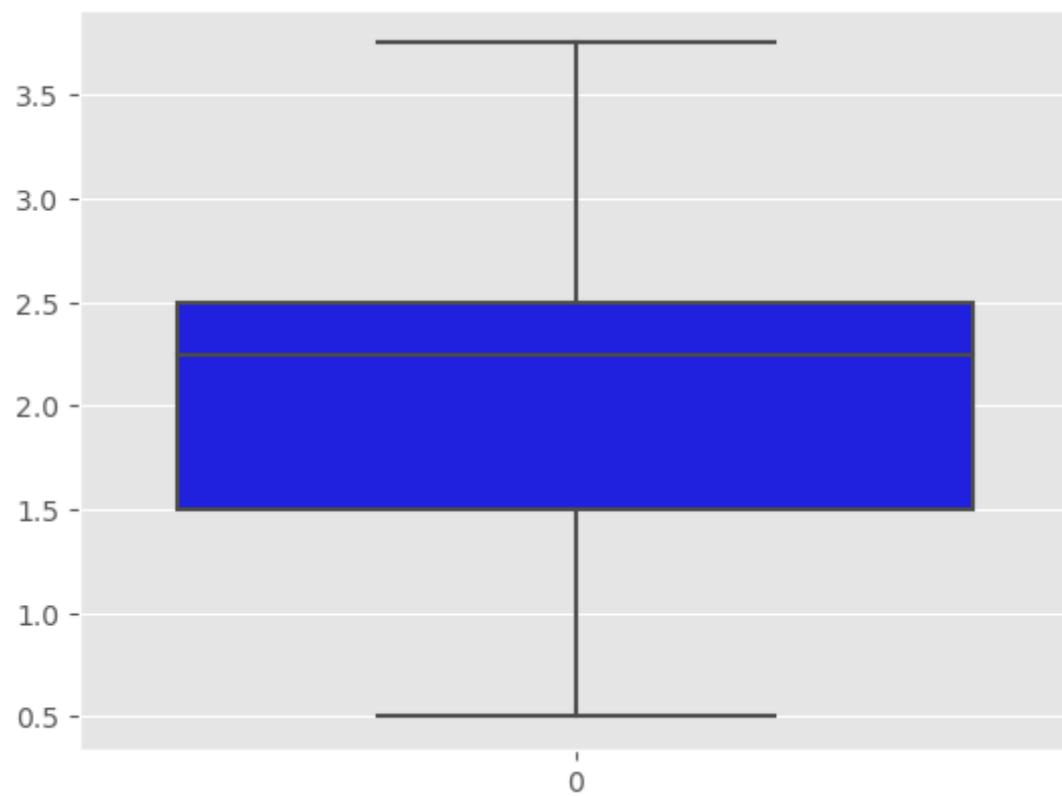
In [38]:

```
# Drop houses that have more than 4 bathrooms
df = df[df['bathrooms'] < 4]
```

In [39]:

```
# Boxplot to detect outliers
sns.boxplot(df['bathrooms'], color = "BLUE");

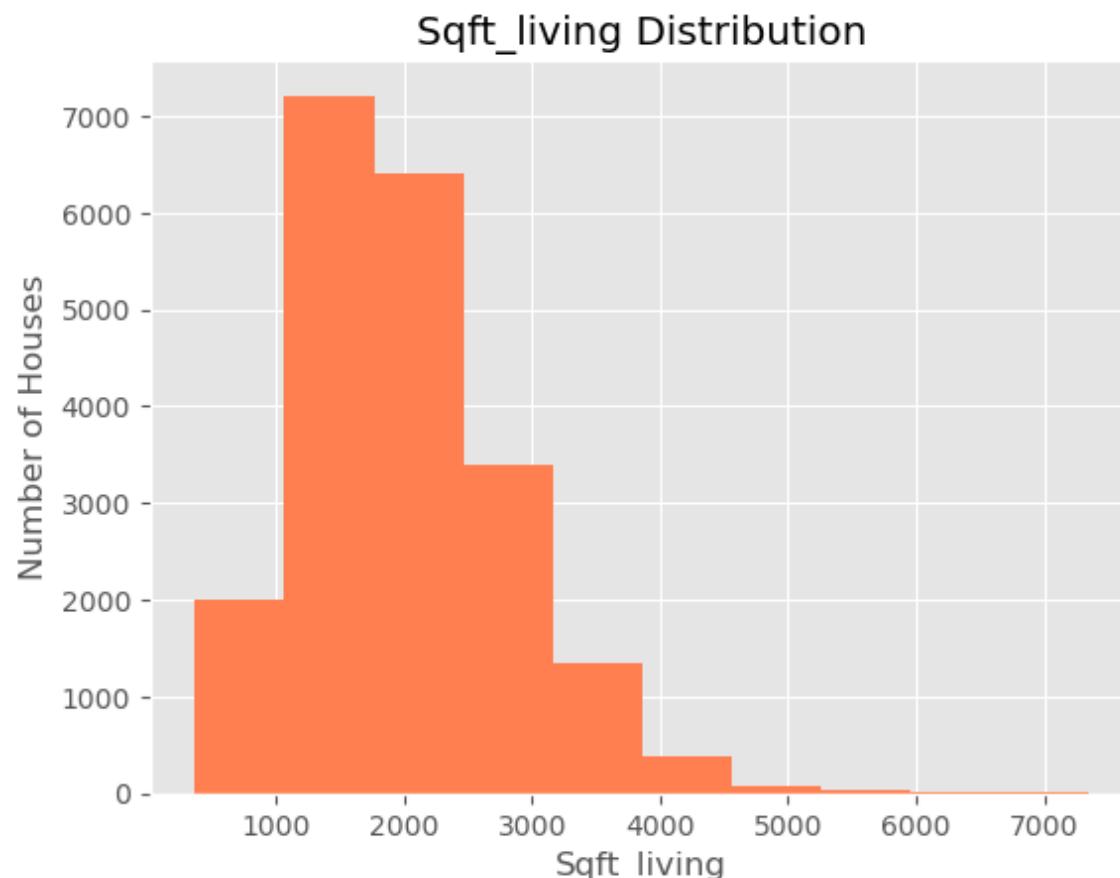
#Boxplot shows are normal distribution
```



## SQFT\_LIVING

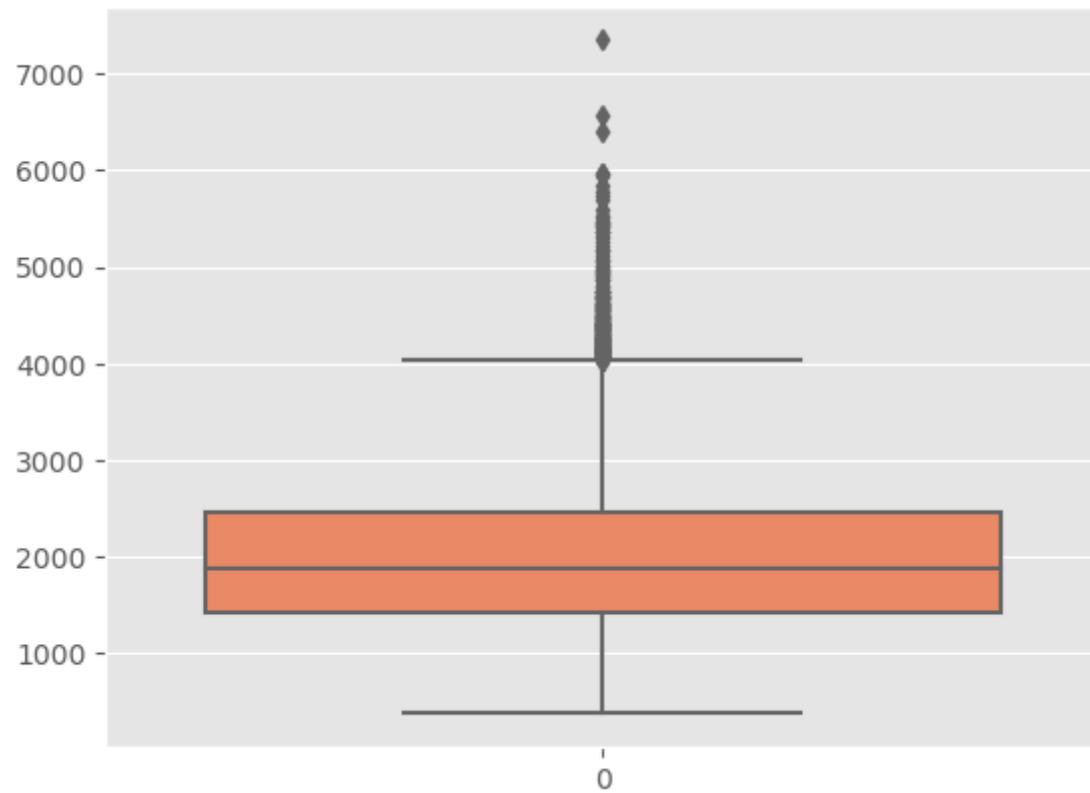
In [40]:

```
# Check sqft_living distribution
df['sqft_living'].hist(color = "coral")
plt.title('Sqft_living Distribution');
plt.xlabel('Sqft_living');
plt.ylabel('Number of Houses');
```



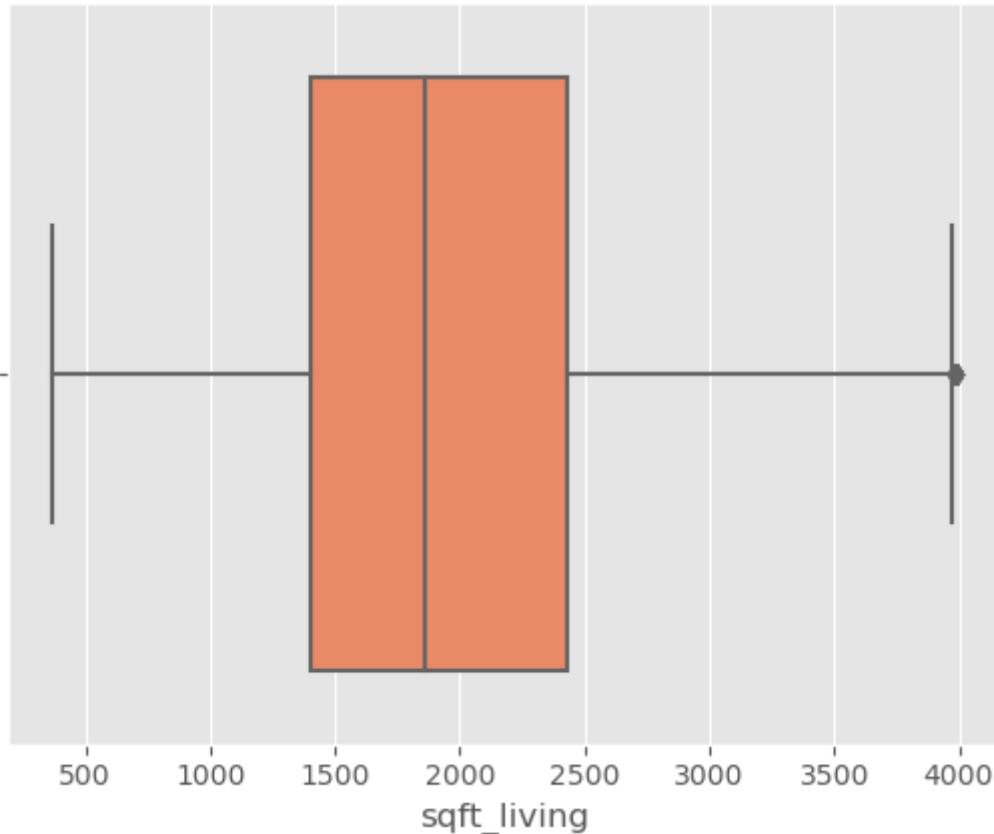
In [41]:

```
# Box plot for outliers  
#There are outliers above 4000 sqft_living.  
sns.boxplot(df['sqft_living'], color = "coral");
```



In [42]:

```
# Drop outliers in sqft_living
df = df[df['sqft_living'] < 4000]
sns.boxplot(x = df['sqft_living'], color = "coral");
```

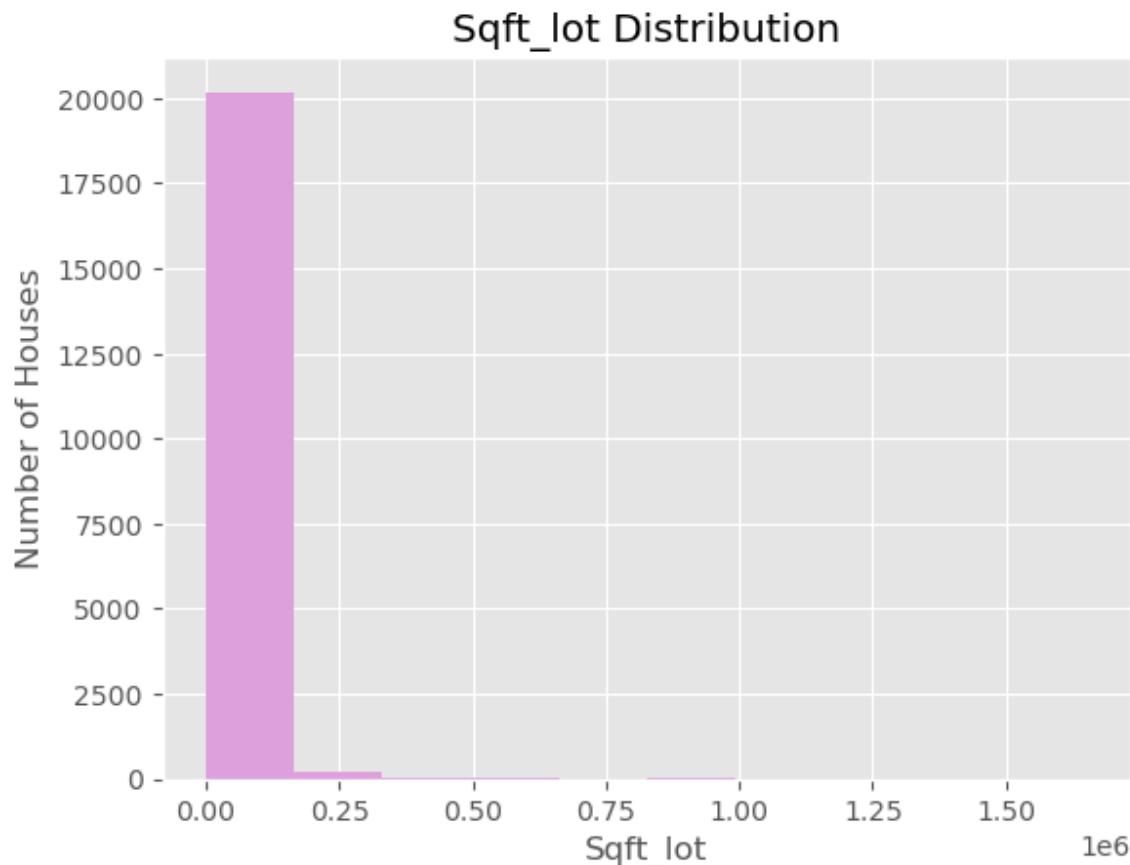


**1500 - 2500 sqft\_living is most commonly observed footage of the house in this dataset**

## SQFT\_LOT

In [43]:

```
# Check sqft_lot distribution
df['sqft_lot'].hist(color = "plum")
plt.title('Sqft_lot Distribution');
plt.xlabel('Sqft_lot');
plt.ylabel('Number of Houses');
```



In [44]:

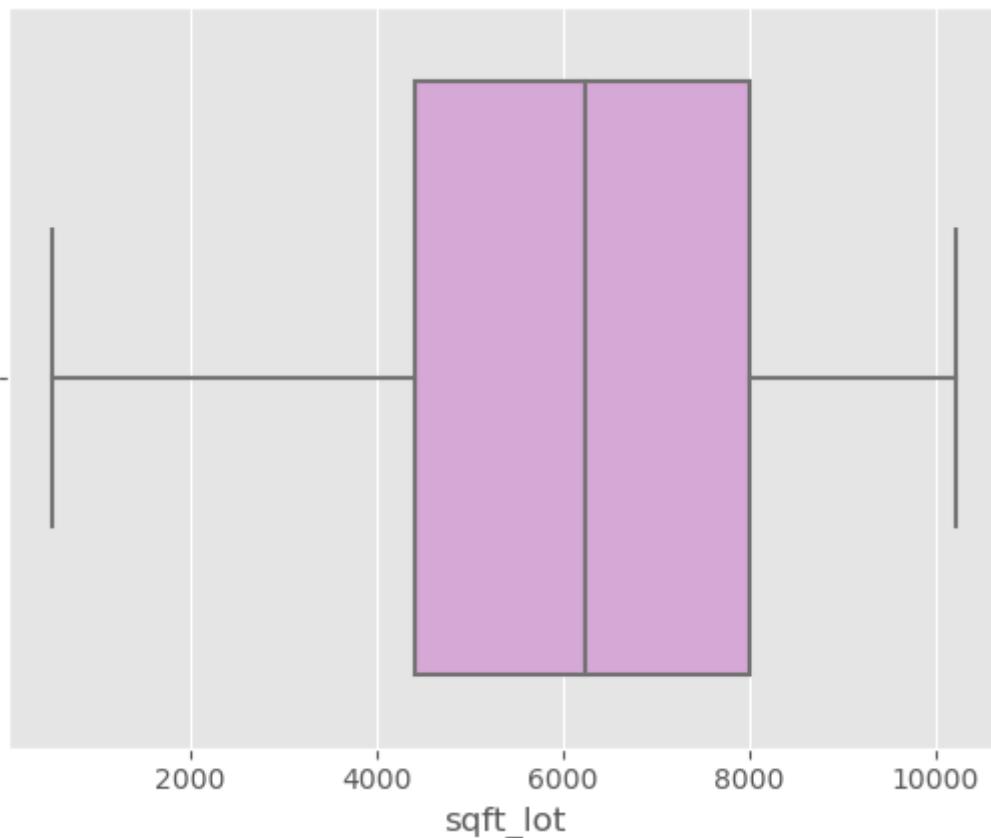
```
df.describe()
```

Out[44]:

|       | price        | bedrooms     | bathrooms    | sqft_living  | sqft_lot     | floors       |
|-------|--------------|--------------|--------------|--------------|--------------|--------------|
| count | 2.044600e+04 | 20446.000000 | 20446.000000 | 20446.000000 | 2.044600e+04 | 20446.000000 |
| mean  | 4.872377e+05 | 3.310036     | 2.033735     | 1954.156705  | 1.399462e+04 | 1.472684     |
| std   | 2.327866e+05 | 0.850293     | 0.676472     | 724.038899   | 3.766160e+04 | 0.536808     |
| min   | 7.800000e+04 | 1.000000     | 0.500000     | 370.000000   | 5.200000e+02 | 1.000000     |
| 25%   | 3.150000e+05 | 3.000000     | 1.500000     | 1400.000000  | 5.000000e+03 | 1.000000     |
| 50%   | 4.380000e+05 | 3.000000     | 2.000000     | 1860.000000  | 7.500000e+03 | 1.000000     |
| 75%   | 6.050000e+05 | 4.000000     | 2.500000     | 2430.000000  | 1.023275e+04 | 2.000000     |
| max   | 1.500000e+06 | 6.000000     | 3.750000     | 3990.000000  | 1.651359e+06 | 3.500000     |

In [45]:

```
# Drop houses that are above the 75 percentile
df = df[df['sqft_lot'] < 1.023275e+04]
sns.boxplot(x = df['sqft_lot'], color = "plum");
```

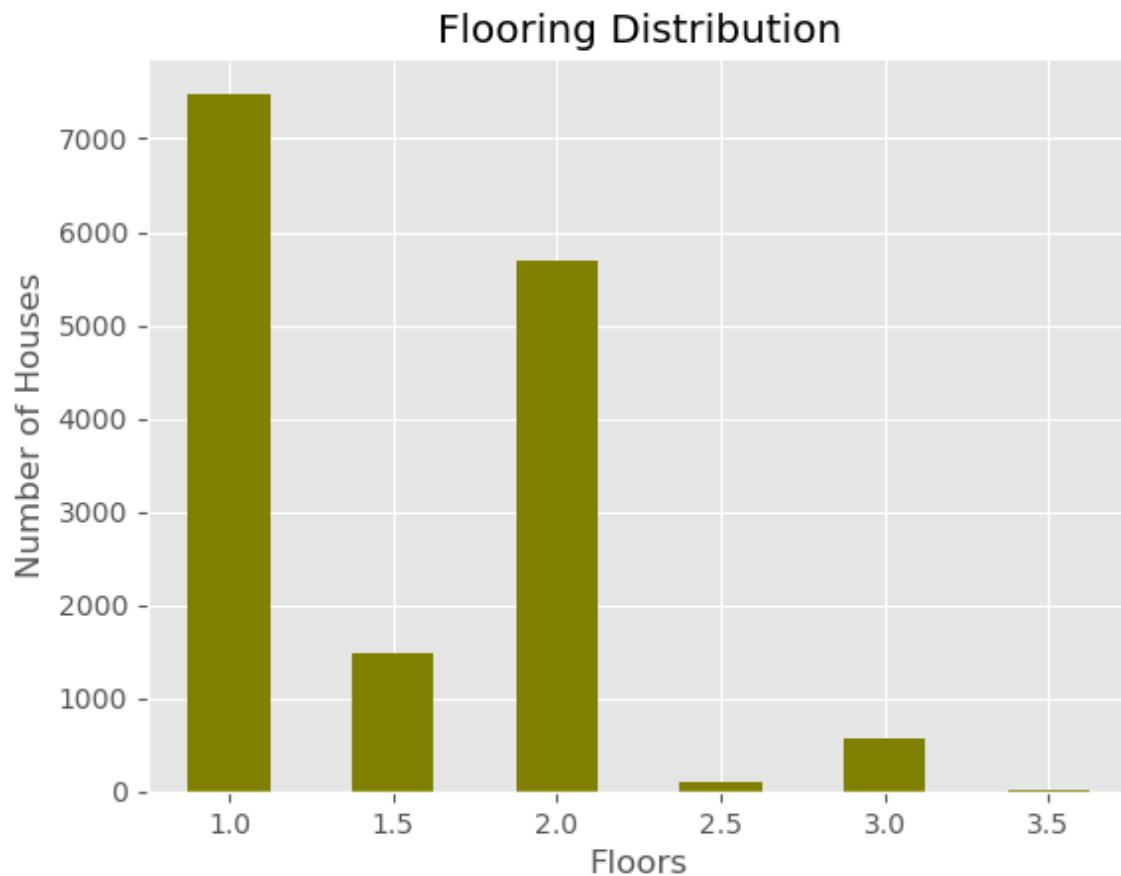


The majority of houses have between 4000 and 8000 sqft\_lot.

## FLOORS

In [46]:

```
# Plot a bar chart to check distribution of flooring
df['floors'].value_counts().sort_index().plot(kind = 'bar', color = "OLIVE")
plt.xticks(rotation = 0)
plt.title('Flooring Distribution');
plt.xlabel('Floors');
plt.ylabel('Number of Houses');
```

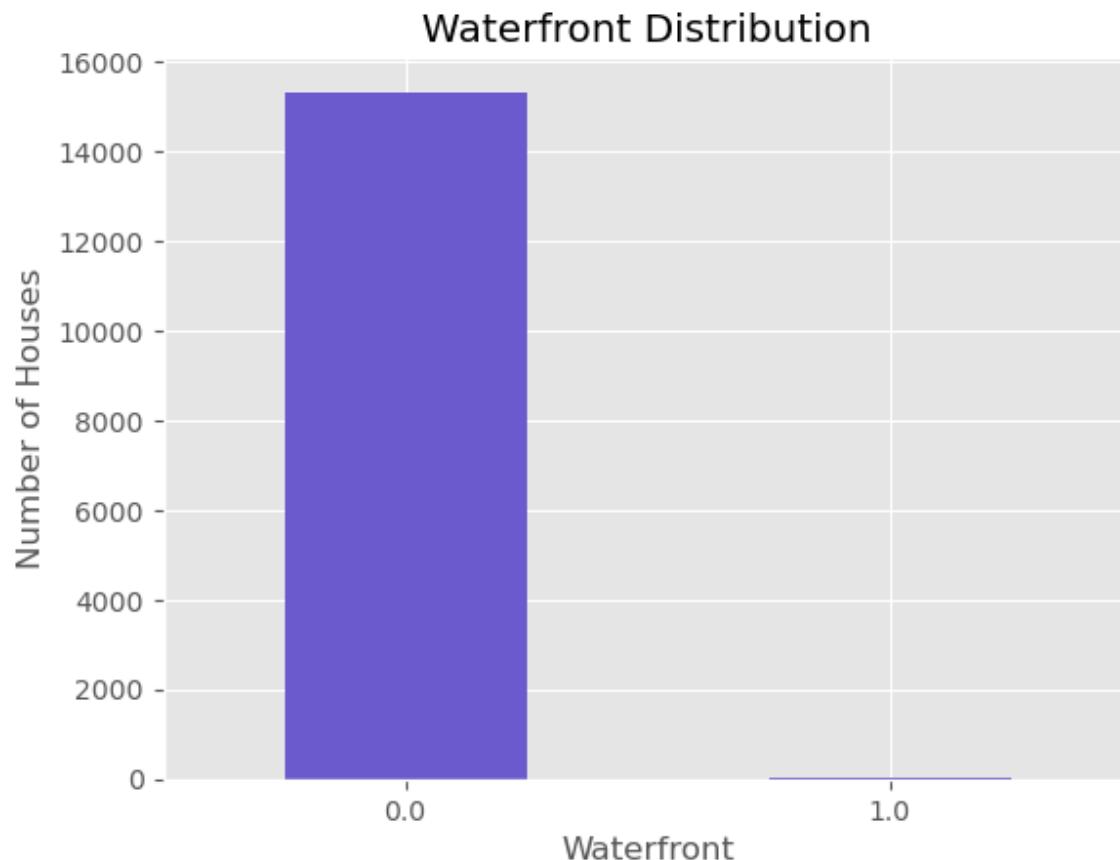


**Most houses have either 1 or 2 floors.**

# WATERFRONT

In [47]:

```
# Plot a bar chart for waterfront
df['waterfront'].value_counts().sort_index().plot(kind = 'bar', color = "slateblue")
plt.xticks(rotation = 0)
plt.title('Waterfront Distribution');
plt.xlabel('Waterfront');
plt.ylabel('Number of Houses');
```



Waterfront is categorical as it has two categories. Also, the majority of the homes are not waterfront. 99% of houses do not have waterfront.

In [48]:

```
#checking data types  
df.dtypes
```

Out[48]:

```
date          object  
price         int64  
bedrooms      int64  
bathrooms     float64  
sqft_living   int64  
sqft_lot      int64  
floors        float64  
waterfront    float64  
view          float64  
condition     int64  
grade         int64  
sqft_above    int64  
sqft_basement float64  
yr_built      int64  
zipcode       int64  
lat           float64  
long          float64  
sqft_living15 int64  
sqft_lot15    int64  
renovated     int32  
dtype: object
```

## EXPLORE

After cleaning the data, I then explore it to understand its structure, trends, and patterns. This typically involves generating descriptive statistics, and visualizing the data using various kinds of plots. The goal is to gain insights that will help you when modeling.

## Identify categorical variables

In [49]:

```
df
```

Out[49]:

|       | date       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view |
|-------|------------|--------|----------|-----------|-------------|----------|--------|------------|------|
| 0     | 10/13/2014 | 221900 | 3        | 1.00      | 1180        | 5650     | 1.0    | 0.0        | 0.   |
| 1     | 12/09/2014 | 538000 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0.0        | 0.   |
| 2     | 2/25/2015  | 180000 | 2        | 1.00      | 770         | 10000    | 1.0    | 0.0        | 0.   |
| 3     | 12/09/2014 | 604000 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0.0        | 0.   |
| 4     | 2/18/2015  | 510000 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0.0        | 0.   |
| ...   | ...        | ...    | ...      | ...       | ...         | ...      | ...    | ...        | ...  |
| 21592 | 5/21/2014  | 360000 | 3        | 2.50      | 1530        | 1131     | 3.0    | 0.0        | 0.   |
| 21593 | 2/23/2015  | 400000 | 4        | 2.50      | 2310        | 5813     | 2.0    | 0.0        | 0.   |
| 21594 | 6/23/2014  | 402101 | 2        | 0.75      | 1020        | 1350     | 2.0    | 0.0        | 0.   |
| 21595 | 1/16/2015  | 400000 | 3        | 2.50      | 1600        | 2388     | 2.0    | 0.0        | 0.   |
| 21596 | 10/15/2014 | 325000 | 2        | 0.75      | 1020        | 1076     | 2.0    | 0.0        | 0.   |

15334 rows × 20 columns

In [50]:

```
#fill out NaN values for Zipcode
df.zipcode.mean()
df.zipcode.median()
df.zipcode.fillna(df.zipcode.median(),inplace=True)
df.zipcode.isnull().sum()
```

Out[50]:

```
0
```

In [51]:

```
df.tail()
```

Out[51]:

|       | date       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view |
|-------|------------|--------|----------|-----------|-------------|----------|--------|------------|------|
| 21592 | 5/21/2014  | 360000 | 3        | 2.50      | 1530        | 1131     | 3.0    | 0.0        | 0.   |
| 21593 | 2/23/2015  | 400000 | 4        | 2.50      | 2310        | 5813     | 2.0    | 0.0        | 0.   |
| 21594 | 6/23/2014  | 402101 | 2        | 0.75      | 1020        | 1350     | 2.0    | 0.0        | 0.   |
| 21595 | 1/16/2015  | 400000 | 3        | 2.50      | 1600        | 2388     | 2.0    | 0.0        | 0.   |
| 21596 | 10/15/2014 | 325000 | 2        | 0.75      | 1020        | 1076     | 2.0    | 0.0        | 0.   |

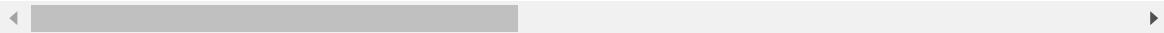
In [52]:

```
df.reset_index(drop = True)
```

Out[52]:

|       | date       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | vie |
|-------|------------|--------|----------|-----------|-------------|----------|--------|------------|-----|
| 0     | 10/13/2014 | 221900 | 3        | 1.00      | 1180        | 5650     | 1.0    | 0.0        | 0.  |
| 1     | 12/09/2014 | 538000 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0.0        | 0.  |
| 2     | 2/25/2015  | 180000 | 2        | 1.00      | 770         | 10000    | 1.0    | 0.0        | 0.  |
| 3     | 12/09/2014 | 604000 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0.0        | 0.  |
| 4     | 2/18/2015  | 510000 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0.0        | 0.  |
| ...   | ...        | ...    | ...      | ...       | ...         | ...      | ...    | ...        | ... |
| 15329 | 5/21/2014  | 360000 | 3        | 2.50      | 1530        | 1131     | 3.0    | 0.0        | 0.  |
| 15330 | 2/23/2015  | 400000 | 4        | 2.50      | 2310        | 5813     | 2.0    | 0.0        | 0.  |
| 15331 | 6/23/2014  | 402101 | 2        | 0.75      | 1020        | 1350     | 2.0    | 0.0        | 0.  |
| 15332 | 1/16/2015  | 400000 | 3        | 2.50      | 1600        | 2388     | 2.0    | 0.0        | 0.  |
| 15333 | 10/15/2014 | 325000 | 2        | 0.75      | 1020        | 1076     | 2.0    | 0.0        | 0.  |

15334 rows × 20 columns



In [53]:

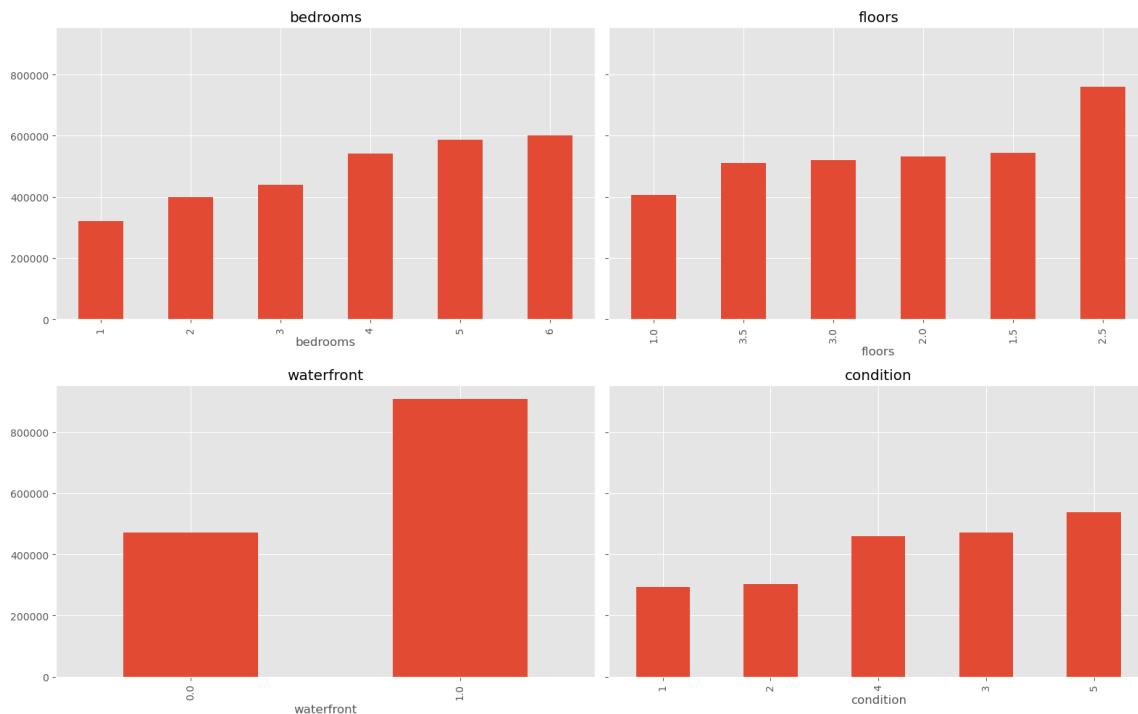
```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(16,10), sharey=True)

cat = ['bedrooms', 'floors', 'waterfront', 'condition']

for col, ax in zip(cat, axes.flatten()):
    (df.groupby(col) # group values together by column of interest
     .mean(numeric_only=True)['price'] # take the mean of the saleprice for each group
     .sort_values() # sort the groups in ascending order
     .plot
     .bar(ax=ax)) # create a bar graph on the axis

    ax.set_title(col) # Make the title the name of the column

fig.tight_layout()
```



In [54]:

```
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

# creating instance of one-hot-encoder
encoder = OneHotEncoder(drop='first')

# Assume df[cat] contains the categorical columns of your original dataframe
# perform one-hot encoding on cat list
encoder_df = pd.DataFrame(encoder.fit_transform(df[cat]).toarray())

# update column names
cols = []
for x in encoder.get_feature_names_out():
    if 'x0' in x:
        cols.append(x.replace('x0', 'bedrooms'))
    elif 'x1' in x:
        cols.append(x.replace('x1', 'floors'))
    elif 'x2' in x:
        cols.append(x.replace('x2', 'waterfront'))
    elif 'x3' in x:
        cols.append(x.replace('x3', 'condition'))
    else:
        cols.append(x)

# set encoder_df columns equal cols
encoder_df.columns = cols

# merge one-hot encoded columns back with original DataFrame
df_ohe = df.join(encoder_df)

# view final df
print(df_ohe.head())
```

```

      date   price  bedrooms  bathrooms  sqft_living  sqft_lot  floors
\\
0  10/13/2014  221900          3       1.00      1180     5650    1.0
1  12/09/2014  538000          3       2.25      2570     7242    2.0
2   2/25/2015  180000          2       1.00       770    10000    1.0
3  12/09/2014  604000          4       3.00      1960     5000    1.0
4   2/18/2015  510000          3       2.00      1680     8080    1.0

  waterfront  view  condition  ...  floors_1.5  floors_2.0  floors_2.5  \\
0        0.0    0.0          3  ...        0.0        0.0        0.0
1        0.0    0.0          3  ...        0.0        1.0        0.0
2        0.0    0.0          3  ...        0.0        0.0        0.0
3        0.0    0.0          5  ...        0.0        0.0        0.0
4        0.0    0.0          3  ...        0.0        0.0        0.0

  floors_3.0  floors_3.5  waterfront_1.0  condition_2  condition_3  \\
0        0.0        0.0        0.0        0.0        1.0
1        0.0        0.0        0.0        0.0        1.0
2        0.0        0.0        0.0        0.0        1.0
3        0.0        0.0        0.0        0.0        0.0
4        0.0        0.0        0.0        0.0        1.0

  condition_4  condition_5
0        0.0        0.0
1        0.0        0.0
2        0.0        0.0
3        0.0        1.0
4        0.0        0.0

```

[5 rows x 35 columns]

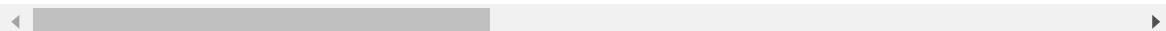
In [55]:

```
df_ohe.head()
```

Out[55]:

|   | date       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | c |
|---|------------|--------|----------|-----------|-------------|----------|--------|------------|------|---|
| 0 | 10/13/2014 | 221900 | 3        | 1.00      | 1180        | 5650     | 1.0    | 0.0        | 0.0  |   |
| 1 | 12/09/2014 | 538000 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0.0        | 0.0  |   |
| 2 | 2/25/2015  | 180000 | 2        | 1.00      | 770         | 10000    | 1.0    | 0.0        | 0.0  |   |
| 3 | 12/09/2014 | 604000 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0.0        | 0.0  |   |
| 4 | 2/18/2015  | 510000 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0.0        | 0.0  |   |

5 rows x 35 columns



In [56]:

```
new_cols = []
for x in df_ohe:
    new_cols.append(x.replace(".", "_"))
df_ohe.columns = new_cols
df_ohe.head()
```

Out[56]:

|   | date       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | c |
|---|------------|--------|----------|-----------|-------------|----------|--------|------------|------|---|
| 0 | 10/13/2014 | 221900 | 3        | 1.00      | 1180        | 5650     | 1.0    | 0.0        | 0.0  |   |
| 1 | 12/09/2014 | 538000 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0.0        | 0.0  |   |
| 2 | 2/25/2015  | 180000 | 2        | 1.00      | 770         | 10000    | 1.0    | 0.0        | 0.0  |   |
| 3 | 12/09/2014 | 604000 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0.0        | 0.0  |   |
| 4 | 2/18/2015  | 510000 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0.0        | 0.0  |   |

5 rows × 35 columns

In [57]:

```
# Drop categorical features to avoid redundancy
df_ohe.drop((cat), axis=1, inplace=True)
```

**Converting date to age of the house sold**

In [58]:

```
# Change date dtype to datetime
df['date'] = pd.to_datetime(df['date'])

df['date'].min()
```

Out[58]:

Timestamp('2014-05-02 00:00:00')

In [59]:

```
# check dtypes
df.dtypes
```

Out[59]:

| date          | datetime64[ns] |
|---------------|----------------|
| price         | int64          |
| bedrooms      | int64          |
| bathrooms     | float64        |
| sqft_living   | int64          |
| sqft_lot      | int64          |
| floors        | float64        |
| waterfront    | float64        |
| view          | float64        |
| condition     | int64          |
| grade         | int64          |
| sqft_above    | int64          |
| sqft_basement | float64        |
| yr_built      | int64          |
| zipcode       | int64          |
| lat           | float64        |
| long          | float64        |
| sqft_living15 | int64          |
| sqft_lot15    | int64          |
| renovated     | int32          |
|               | dtype: object  |

In [60]:

```
# Create new column 'age at sold' by subtracting 'date' and 'yr_built'
age_at_sold = df['date'].dt.year - df['yr_built']
df['age_at_sold'] = age_at_sold
df.head()
```

Out[60]:

|   | date       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condit |
|---|------------|--------|----------|-----------|-------------|----------|--------|------------|------|--------|
| 0 | 2014-10-13 | 221900 | 3        | 1.00      | 1180        | 5650     | 1.0    | 0.0        | 0.0  |        |
| 1 | 2014-12-09 | 538000 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0.0        | 0.0  |        |
| 2 | 2015-02-25 | 180000 | 2        | 1.00      | 770         | 10000    | 1.0    | 0.0        | 0.0  |        |
| 3 | 2014-12-09 | 604000 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0.0        | 0.0  |        |
| 4 | 2015-02-18 | 510000 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0.0        | 0.0  |        |

5 rows × 21 columns

age\_at\_sold = New column is now a continuous feature.

In [61]:

```
# Drop columns 'date'  
df.drop('date', axis=1, inplace=True)
```

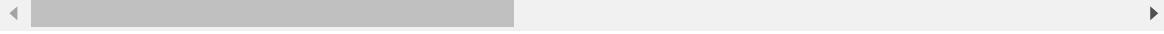
In [62]:

df

Out[62]:

|       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition |
|-------|--------|----------|-----------|-------------|----------|--------|------------|------|-----------|
| 0     | 221900 | 3        | 1.00      | 1180        | 5650     | 1.0    | 0.0        | 0.0  | 3         |
| 1     | 538000 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0.0        | 0.0  | 3         |
| 2     | 180000 | 2        | 1.00      | 770         | 10000    | 1.0    | 0.0        | 0.0  | 3         |
| 3     | 604000 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0.0        | 0.0  | 5         |
| 4     | 510000 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0.0        | 0.0  | 3         |
| ...   | ...    | ...      | ...       | ...         | ...      | ...    | ...        | ...  | ..        |
| 21592 | 360000 | 3        | 2.50      | 1530        | 1131     | 3.0    | 0.0        | 0.0  | 3         |
| 21593 | 400000 | 4        | 2.50      | 2310        | 5813     | 2.0    | 0.0        | 0.0  | 3         |
| 21594 | 402101 | 2        | 0.75      | 1020        | 1350     | 2.0    | 0.0        | 0.0  | 3         |
| 21595 | 400000 | 3        | 2.50      | 1600        | 2388     | 2.0    | 0.0        | 0.0  | 3         |
| 21596 | 325000 | 2        | 0.75      | 1020        | 1076     | 2.0    | 0.0        | 0.0  | 3         |

15334 rows × 20 columns



# Check for multicollinearity

## Scatter matrix

In [63]:

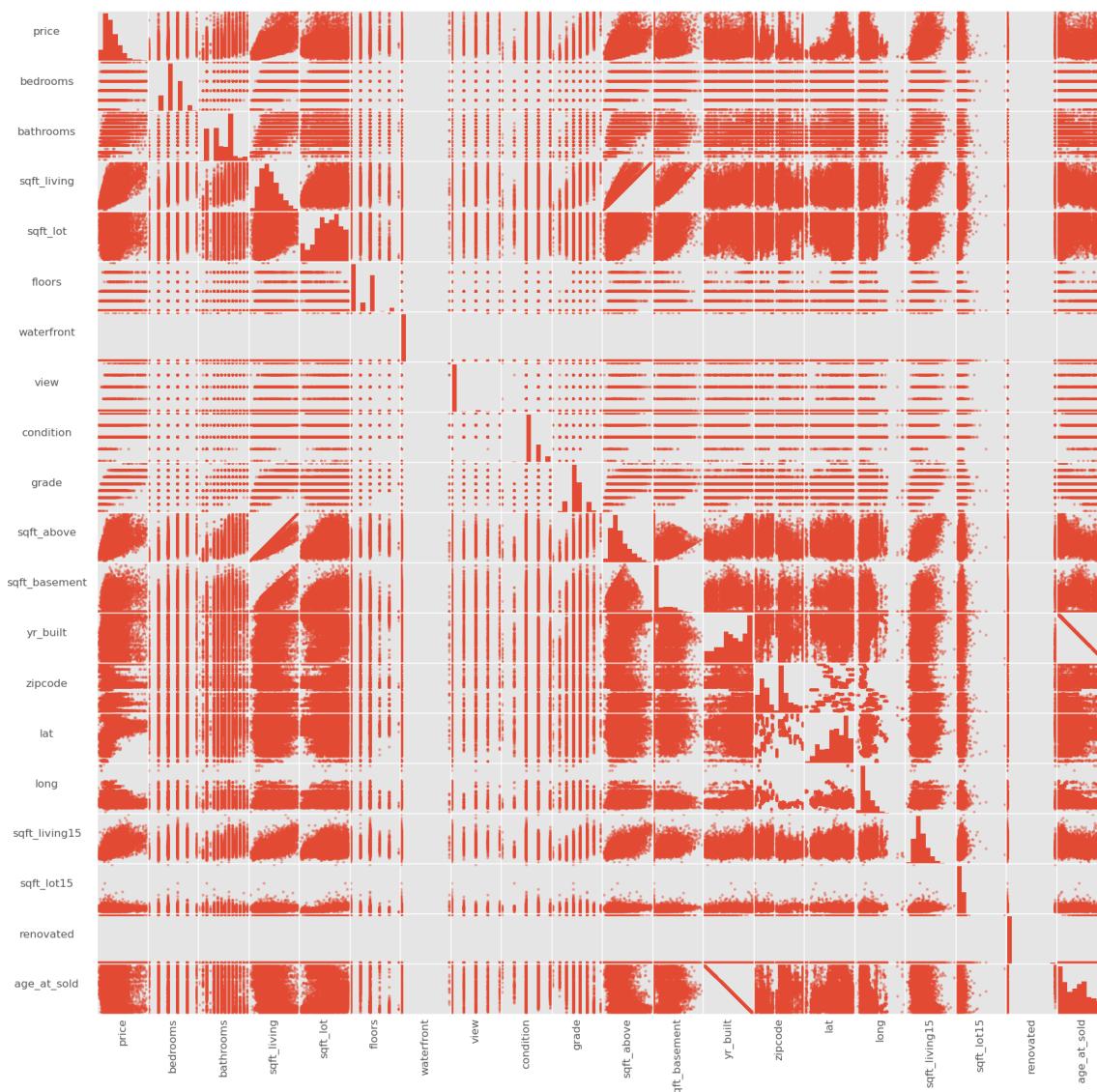
```
sm = pd.plotting.scatter_matrix(df, figsize=[20, 20]);

# Rotates the text
[s.xaxis.label.set_rotation(90) for s in sm.reshape(-1)]
[s.yaxis.label.set_rotation(0) for s in sm.reshape(-1)]

#May need to offset Label when rotating to prevent overlap of figure
[s.get_yaxis().set_label_coords(-1,0.5) for s in sm.reshape(-1)]

#Hide all ticks
[s.set_xticks([]) for s in sm.reshape(-1)]
[s.set_yticks([]) for s in sm.reshape(-1)]

plt.show()
```



## Correlation matrix

In [64]:

```
#Correlation matrix
df.corr(numeric_only=True)
```

Out[64]:

|               | price     | bedrooms  | bathrooms | sqft_living | sqft_lot  | floors    | waterfront |
|---------------|-----------|-----------|-----------|-------------|-----------|-----------|------------|
| price         | 1.000000  | 0.270020  | 0.401291  | 0.582435    | -0.091096 | 0.258474  | 0.062988   |
| bedrooms      | 0.270020  | 1.000000  | 0.484538  | 0.628025    | 0.230280  | 0.159391  | -0.026551  |
| bathrooms     | 0.401291  | 0.484538  | 1.000000  | 0.701893    | -0.092625 | 0.533343  | -0.009154  |
| sqft_living   | 0.582435  | 0.628025  | 0.701893  | 1.000000    | 0.171210  | 0.343853  | -0.006088  |
| sqft_lot      | -0.091096 | 0.230280  | -0.092625 | 0.171210    | 1.000000  | -0.449486 | 0.007880   |
| floors        | 0.258474  | 0.159391  | 0.533343  | 0.343853    | -0.449486 | 1.000000  | 0.003375   |
| waterfront    | 0.062988  | -0.026551 | -0.009154 | -0.006088   | 0.007880  | 0.003375  | 1.000000   |
| view          | 0.288403  | 0.031689  | 0.082679  | 0.160663    | 0.024788  | -0.004820 | 0.200719   |
| condition     | 0.062857  | 0.015076  | -0.152621 | -0.069797   | 0.143082  | -0.284153 | 0.008267   |
| grade         | 0.600733  | 0.325391  | 0.608525  | 0.670470    | -0.047194 | 0.487685  | -0.006307  |
| sqft_above    | 0.456314  | 0.507400  | 0.626796  | 0.840791    | 0.136252  | 0.537249  | -0.007722  |
| sqft_basement | 0.269508  | 0.262739  | 0.194036  | 0.365580    | 0.077200  | -0.294297 | 0.002646   |
| yr_built      | -0.033115 | 0.165549  | 0.555123  | 0.328073    | -0.089224 | 0.523722  | -0.016596  |
| zipcode       | 0.058303  | -0.169524 | -0.225719 | -0.192169   | -0.245899 | -0.083009 | 0.017435   |
| lat           | 0.443162  | -0.075943 | -0.041972 | -0.005617   | -0.191920 | 0.022036  | -0.005560  |
| long          | -0.030962 | 0.168526  | 0.263014  | 0.264305    | 0.219901  | 0.150705  | -0.005365  |
| sqft_living15 | 0.483158  | 0.403860  | 0.519489  | 0.728039    | 0.199125  | 0.275359  | 0.010680   |
| sqft_lot15    | -0.096889 | 0.180471  | -0.089923 | 0.132528    | 0.832437  | -0.394623 | 0.051694   |
| renovated     | 0.107844  | 0.003951  | 0.026586  | 0.031777    | -0.014106 | -0.006285 | 0.029324   |
| age_at_sold   | 0.033206  | -0.165735 | -0.555538 | -0.328501   | 0.089285  | -0.523947 | 0.016707   |

◀ ▶

In [65]:

```
#Return True for positive or negative correlations that are bigger than 0.75 in the corre
abs(df.corr(numeric_only=True)) > 0.75
```

Out[65]:

|               | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view  | condition | grade | sqft_above | sqft_basement | yr_built | zipcode | lat   | long  | sqft_living15 | sqft_lot15 | renovated | age_at_sold |
|---------------|-------|----------|-----------|-------------|----------|--------|------------|-------|-----------|-------|------------|---------------|----------|---------|-------|-------|---------------|------------|-----------|-------------|
| price         | True  | False    | False     | False       | False    | False  | False      | False | False     | False | False      | False         | False    | False   | False | False | False         | False      | False     |             |
| bedrooms      | False | True     | False     | False       | False    | False  | False      | False | False     | False | False      | False         | False    | False   | False | False | False         | False      | False     |             |
| bathrooms     | False | False    | True      | False       | False    | False  | False      | False | False     | False | False      | False         | False    | False   | False | False | False         | False      | False     |             |
| sqft_living   | False | False    | False     | True        | False    | False  | False      | False | False     | False | False      | False         | False    | False   | False | False | False         | False      | False     |             |
| sqft_lot      | False | False    | False     | False       | True     | False  | False      | False | False     | False | False      | False         | False    | False   | False | False | False         | False      | False     |             |
| floors        | False | False    | False     | False       | False    | True   | False      | False | False     | False | False      | False         | False    | False   | False | False | False         | False      | False     |             |
| waterfront    | False | False    | False     | False       | False    | False  | True       | False | False     | False | False      | False         | False    | False   | False | False | False         | False      | False     |             |
| view          | False | False    | False     | False       | False    | False  | False      | True  | False     | False | False      | False         | False    | False   | False | False | False         | False      | False     |             |
| condition     | False | False    | False     | False       | False    | False  | False      | False | True      | False | False      | False         | False    | False   | False | False | False         | False      | False     |             |
| grade         | False | False    | False     | False       | False    | False  | False      | False | False     | True  | False      | False         | False    | False   | False | False | False         | False      | False     |             |
| sqft_above    | False | False    | False     | True        | False    | False  | False      | False | False     | False | True       | False         | False    | False   | False | False | False         | False      | False     |             |
| sqft_basement | False | False    | False     | False       | False    | False  | False      | False | False     | False | False      | True          | False    | False   | False | False | False         | False      | False     |             |
| yr_built      | False | False    | False     | False       | False    | False  | False      | False | False     | False | False      | False         | True     | False   | False | False | False         | False      | False     |             |
| zipcode       | False | False    | False     | False       | False    | False  | False      | False | False     | False | False      | False         | False    | True    | False | False | False         | False      | False     |             |
| lat           | False | False    | False     | False       | False    | False  | False      | False | False     | False | False      | False         | False    | False   | True  | False | False         | False      | False     |             |
| long          | False | False    | False     | False       | False    | False  | False      | False | False     | False | False      | False         | False    | False   | False | True  | False         | False      | False     |             |
| sqft_living15 | False | False    | False     | False       | False    | False  | False      | False | False     | False | False      | False         | False    | False   | False | False | True          | False      | False     |             |
| sqft_lot15    | False | False    | False     | False       | True     | False  | False      | False | False     | False | False      | False         | False    | False   | False | False | False         | True       | False     |             |
| renovated     | False | False    | False     | False       | False    | False  | False      | False | False     | False | False      | False         | False    | False   | False | False | False         | False      | True      |             |
| age_at_sold   | False | False    | False     | False       | False    | False  | False      | False | False     | False | False      | False         | False    | False   | False | False | False         | False      | True      |             |

Now, include stack and zip to create a more robust solution that will return the variable pairs from the correlation matrix that have correlations over .75, but less than 1.

In [66]:

```
df1 = df.corr(numeric_only=True).abs().stack().reset_index().sort_values(0, ascending=False)

df1['pairs'] = list(zip(df1.level_0, df1.level_1))

df1.set_index(['pairs'], inplace = True)

df1.drop(columns=['level_1', 'level_0'], inplace = True)

# cc for correlation coefficient
df1.columns = ['cc']

df1.drop_duplicates(inplace=True)

df1[(df1.cc > .75) & (df1.cc < 1)]
```

Out[66]:

|                           | cc       |
|---------------------------|----------|
| pairs                     |          |
| (age_at_sold, yr_built)   | 0.999890 |
| (sqft_above, sqft_living) | 0.840791 |
| (sqft_lot, sqft_lot15)    | 0.832437 |

**There are three sets of variables that are highly correlated.**

- age\_at\_sold with yr\_built
- sqft\_above with sqft\_living
- sqft\_lot15 with sqft\_lot).

Did not removed any variables yet

## Heatmap

In [67]:

```
# View columns
df.columns
```

Out[67]:

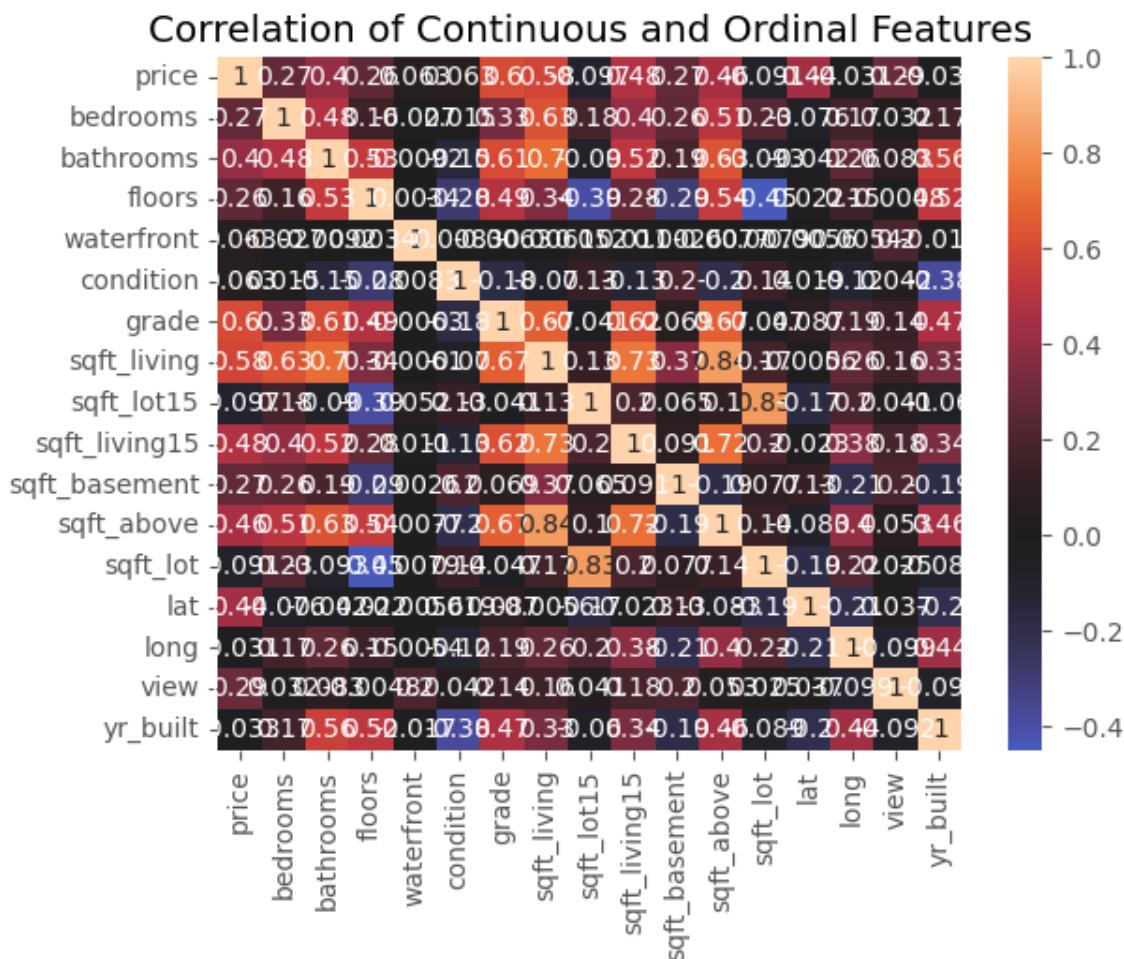
```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'zipcode', 'lat', 'long', 'sqft_living15',
       'sqft_lot15', 'renovated', 'age_at_sold'],
      dtype='object')
```

In [68]:

```
# Create a list of continuous and ordinal categorical features
cont_ordinal = ['price', 'bedrooms', 'bathrooms', 'floors', 'waterfront', 'condition', 'grade', 'sqft_living', 'sqft_lot15', 'sqft_living15', 'sqft_basement', 'sqft_above', 'sqft_lot', 'lat', 'long', 'view', 'yr_built']
```

In [69]:

```
# Correlation heat map with continuous and ordinal features
corr = df[cont_ordinal].corr()
sns.heatmap(corr, center=0, annot=True).set_title('Correlation of Continuous and Ordinal Features')
```



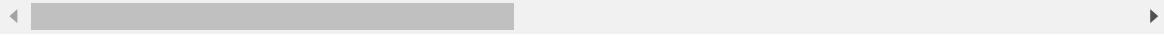
In [70]:

df

Out[70]:

|       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition |
|-------|--------|----------|-----------|-------------|----------|--------|------------|------|-----------|
| 0     | 221900 | 3        | 1.00      | 1180        | 5650     | 1.0    | 0.0        | 0.0  | 3         |
| 1     | 538000 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0.0        | 0.0  | 3         |
| 2     | 180000 | 2        | 1.00      | 770         | 10000    | 1.0    | 0.0        | 0.0  | 3         |
| 3     | 604000 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0.0        | 0.0  | 5         |
| 4     | 510000 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0.0        | 0.0  | 3         |
| ...   | ...    | ...      | ...       | ...         | ...      | ...    | ...        | ...  | ..        |
| 21592 | 360000 | 3        | 2.50      | 1530        | 1131     | 3.0    | 0.0        | 0.0  | 3         |
| 21593 | 400000 | 4        | 2.50      | 2310        | 5813     | 2.0    | 0.0        | 0.0  | 3         |
| 21594 | 402101 | 2        | 0.75      | 1020        | 1350     | 2.0    | 0.0        | 0.0  | 3         |
| 21595 | 400000 | 3        | 2.50      | 1600        | 2388     | 2.0    | 0.0        | 0.0  | 3         |
| 21596 | 325000 | 2        | 0.75      | 1020        | 1076     | 2.0    | 0.0        | 0.0  | 3         |

15334 rows × 20 columns



Can see a correlation between (sqft\_above, sqft\_living) is 0.840791 and (sqft\_lot15, sqft\_lot) is 0.83243

## Double checking for null values

In [71]:

df

Out[71]:

|       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition |
|-------|--------|----------|-----------|-------------|----------|--------|------------|------|-----------|
| 0     | 221900 | 3        | 1.00      | 1180        | 5650     | 1.0    | 0.0        | 0.0  | 3         |
| 1     | 538000 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0.0        | 0.0  | 3         |
| 2     | 180000 | 2        | 1.00      | 770         | 10000    | 1.0    | 0.0        | 0.0  | 3         |
| 3     | 604000 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0.0        | 0.0  | 5         |
| 4     | 510000 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0.0        | 0.0  | 3         |
| ...   | ...    | ...      | ...       | ...         | ...      | ...    | ...        | ...  | ..        |
| 21592 | 360000 | 3        | 2.50      | 1530        | 1131     | 3.0    | 0.0        | 0.0  | 3         |
| 21593 | 400000 | 4        | 2.50      | 2310        | 5813     | 2.0    | 0.0        | 0.0  | 3         |
| 21594 | 402101 | 2        | 0.75      | 1020        | 1350     | 2.0    | 0.0        | 0.0  | 3         |
| 21595 | 400000 | 3        | 2.50      | 1600        | 2388     | 2.0    | 0.0        | 0.0  | 3         |
| 21596 | 325000 | 2        | 0.75      | 1020        | 1076     | 2.0    | 0.0        | 0.0  | 3         |

15334 rows × 20 columns



In [72]:

```
#check for null values
df.isnull().sum()
```

Out[72]:

```
price          0
bedrooms       0
bathrooms      0
sqft_living    0
sqft_lot       0
floors         0
waterfront     0
view           43
condition      0
grade          0
sqft_above     0
sqft_basement  0
yr_built       0
zipcode        0
lat            0
long           0
sqft_living15  0
sqft_lot15    0
renovated      0
age_at_sold   0
dtype: int64
```

In [73]:

```
#view has 43 null values
df['view'].value_counts()
```

Out[73]:

```
0.0    14249
2.0     550
3.0     221
1.0     190
4.0      81
Name: view, dtype: int64
```

In [74]:

```
df['view'].mean()
```

Out[74]:

```
0.1489111241907004
```

In [75]:

```
df['view'].median()
```

Out[75]:

```
0.0
```

In [76]:

```
df['view'].fillna(df['view'].median(), inplace=True)
```

In [77]:

```
df.view.isnull().sum()
```

Out[77]:

0

## MODEL

The modeling phase involves selecting appropriate algorithms to analyze the data. This phase usually involves training and testing a model.

## MODEL 1

In [78]:

```
df
```

Out[78]:

|       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition |
|-------|--------|----------|-----------|-------------|----------|--------|------------|------|-----------|
| 0     | 221900 | 3        | 1.00      | 1180        | 5650     | 1.0    | 0.0        | 0.0  | 3         |
| 1     | 538000 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0.0        | 0.0  | 3         |
| 2     | 180000 | 2        | 1.00      | 770         | 10000    | 1.0    | 0.0        | 0.0  | 3         |
| 3     | 604000 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0.0        | 0.0  | 5         |
| 4     | 510000 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0.0        | 0.0  | 3         |
| ...   | ...    | ...      | ...       | ...         | ...      | ...    | ...        | ...  | ..        |
| 21592 | 360000 | 3        | 2.50      | 1530        | 1131     | 3.0    | 0.0        | 0.0  | 3         |
| 21593 | 400000 | 4        | 2.50      | 2310        | 5813     | 2.0    | 0.0        | 0.0  | 3         |
| 21594 | 402101 | 2        | 0.75      | 1020        | 1350     | 2.0    | 0.0        | 0.0  | 3         |
| 21595 | 400000 | 3        | 2.50      | 1600        | 2388     | 2.0    | 0.0        | 0.0  | 3         |
| 21596 | 325000 | 2        | 0.75      | 1020        | 1076     | 2.0    | 0.0        | 0.0  | 3         |

15334 rows × 20 columns

In [79]:

```
train, test = train_test_split(df, random_state = 150)
train.head()
```

Out[79]:

|       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition |
|-------|--------|----------|-----------|-------------|----------|--------|------------|------|-----------|
| 16187 | 447000 | 2        | 1.00      | 720         | 7500     | 1.0    | 0.0        | 2.0  | 3         |
| 6767  | 809000 | 4        | 1.75      | 1790        | 8372     | 1.0    | 0.0        | 0.0  | 4         |
| 17872 | 217000 | 2        | 1.00      | 720         | 4760     | 1.0    | 0.0        | 0.0  | 5         |
| 20242 | 778983 | 4        | 2.50      | 2490        | 5647     | 2.0    | 0.0        | 0.0  | 3         |
| 5748  | 387000 | 2        | 2.25      | 1230        | 1280     | 2.0    | 0.0        | 0.0  | 3         |

In [80]:

```
# Create model function to fit a Linear regression that includes rmse
def model1(train, test):

    target = 'price'
    x_cols = list(train.columns)
    x_cols.remove('price')

    predictors = '+' .join(x_cols)
    formula = target + '~' + predictors
    model = ols(formula=formula, data=train).fit()

    # RMSE
    train_err = (mean_squared_error(train['price'], model.predict(train)))**0.5
    test_err = (mean_squared_error(test['price'], model.predict(test)))**0.5

    print("Train RMSE: ", train_err)
    print("Test RMSE: ", test_err, '\n')

    return model
```

In [81]:

```
model1(train, test).summary()
```

Train RMSE: 122255.66801937918

Test RMSE: 121587.4592701153

Out[81]:

OLS Regression Results

|                       |                  |                            |             |
|-----------------------|------------------|----------------------------|-------------|
| <b>Dep. Variable:</b> | price            | <b>R-squared:</b>          | 0.706       |
| <b>Model:</b>         | OLS              | <b>Adj. R-squared:</b>     | 0.706       |
| <b>Method:</b>        | Least Squares    | <b>F-statistic:</b>        | 1452.       |
| <b>Date:</b>          | Wed, 20 Sep 2023 | <b>Prob (F-statistic):</b> | 0.00        |
| <b>Time:</b>          | 19:54:17         | <b>Log-Likelihood:</b>     | -1.5103e+05 |

In [81]: Observations: 1500706 R-squared: 0.706  
 AIC: 3.021e+05  
 DF Residuals: 14800 BIC: 3.022e+05

However this can be improved !  
**Df Model:** 19

**Covariance Type:** nonrobust  
**A skewness value of 0.995** is a moderate positive skew, suggesting that there are somewhat more low values and fewer high values, and the "tail" of the distribution stretches more towards the right (higher values).  
**coef std err t P>|t|** [0.025 0.975]

|                  |            |          |        |       |           |          |
|------------------|------------|----------|--------|-------|-----------|----------|
| <b>Intercept</b> | -4.992e+07 | 5.56e+06 | -8.978 | 0.000 | -6.08e+07 | -3.9e+07 |
|------------------|------------|----------|--------|-------|-----------|----------|

**bedrooms** 1.414e+04 1810.270 -7.813 0.000 1.77e+04 1.06e+04  
**A kurtosis value of 6.506** is greater than 3, indicating a leptokurtic distribution. This suggests that the dataset has more extreme values (outliers) than we would expect from a normal distribution. It implies that the tails of the distribution are relatively heavy, and that data points are more likely to be found far from the mean compared to a normally distributed dataset.  
**sqft\_living** 18.0345 18.503 6.379 0.000 81.766 154.303

|                 |         |       |        |       |        |        |
|-----------------|---------|-------|--------|-------|--------|--------|
| <b>sqft_lot</b> | -6.1074 | 0.923 | -6.617 | 0.000 | -7.917 | -4.298 |
|-----------------|---------|-------|--------|-------|--------|--------|

|               |           |          |       |       |         |          |
|---------------|-----------|----------|-------|-------|---------|----------|
| <b>floors</b> | 7886.2621 | 3564.781 | 2.212 | 0.027 | 898.684 | 1.49e+04 |
|---------------|-----------|----------|-------|-------|---------|----------|

**MODEL 2**  
**waterfront** 2.259e+05 3.77e+04 8.649 0.000 2.52e+05 4e+05

|             |           |          |        |       |          |         |
|-------------|-----------|----------|--------|-------|----------|---------|
| <b>view</b> | 4.287e+04 | 2094.430 | 20.470 | 0.000 | 3.88e+04 | 4.7e+04 |
|-------------|-----------|----------|--------|-------|----------|---------|

In [82]: **condition** 2.709e+04 1978.907 13.688 0.000 2.32e+04 3.1e+04

```
print(train.columns)
print(test.columns)
```

|                   |         |        |        |       |         |        |
|-------------------|---------|--------|--------|-------|---------|--------|
| <b>sqft_above</b> | -3.3708 | 18.502 | -0.182 | 0.855 | -39.638 | 32.897 |
|-------------------|---------|--------|--------|-------|---------|--------|

|                      |          |        |        |       |         |        |
|----------------------|----------|--------|--------|-------|---------|--------|
| <b>sqft_basement</b> | -13.4485 | 18.286 | -0.735 | 0.462 | -49.292 | 22.395 |
|----------------------|----------|--------|--------|-------|---------|--------|

```
yr_built 2.081e+04 view 4.861 condition 0.000 grade 0.04 sqft_above ,  

'sqft_basement', 'yr_built', 'zipcode', 'lat', 'long', 'sqft_living  

15', zipcode -263.1826 28.728 -9.161 0.000 -319.495 -206.871
```

|              |            |           |          |             |      |         |
|--------------|------------|-----------|----------|-------------|------|---------|
| 'sqft_lot15' | -9.610e+05 | renovated | 1.27e+00 | age_at_sold | 1.05 | 5.8e+05 |
|--------------|------------|-----------|----------|-------------|------|---------|

```
dtype='object')
```

|                                                                                                     |         |       |        |       |        |        |
|-----------------------------------------------------------------------------------------------------|---------|-------|--------|-------|--------|--------|
| Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floor<br>s', 'sqft_living15']) | 40.8444 | 3.442 | 11.866 | 0.000 | 34.097 | 47.592 |
|-----------------------------------------------------------------------------------------------------|---------|-------|--------|-------|--------|--------|

|                                                                                                                                                                  |        |       |       |       |       |       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|-------|-------|-------|-------|-------|
| 'waterfront', 'view', 'condition', 'grade', 'sqft_above',<br>'sqft_lot15', 'sqft_basement', 'yr_built', 'zipcode', 'lat', 'long', 'sqft_living<br>15', renovated | 4.3054 | 0.761 | 5.661 | 0.000 | 5.796 | 2.815 |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|-------|-------|-------|-------|-------|

|                                                           |           |          |       |       |          |          |
|-----------------------------------------------------------|-----------|----------|-------|-------|----------|----------|
| 'sqft_lot15', 'renovated', 'age_at_sold'],<br>age_at_sold | 2.917e+04 | 6905.832 | 4.224 | 0.000 | 1.56e+04 | 4.27e+04 |
|-----------------------------------------------------------|-----------|----------|-------|-------|----------|----------|

```
dtype='object')
```

|                 |          |                       |       |
|-----------------|----------|-----------------------|-------|
| <b>Omnibus:</b> | 2292.811 | <b>Durbin-Watson:</b> | 1.979 |
|-----------------|----------|-----------------------|-------|

|                       |       |                          |          |
|-----------------------|-------|--------------------------|----------|
| <b>Prob(Omnibus):</b> | 0.000 | <b>Jarque-Bera (JB):</b> | 7785.939 |
|-----------------------|-------|--------------------------|----------|

|              |       |                  |      |
|--------------|-------|------------------|------|
| <b>Skew:</b> | 0.995 | <b>Prob(JB):</b> | 0.00 |
|--------------|-------|------------------|------|

|                  |       |                  |          |
|------------------|-------|------------------|----------|
| <b>Kurtosis:</b> | 6.506 | <b>Cond. No.</b> | 4.80e+08 |
|------------------|-------|------------------|----------|

Notes:

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

In [83]:

[2] The condition number is large, 4.8e+08. This might indicate that there are

~~strongly linear relationships in your data~~

```
# remove features with p_value less than 0.05
model2 = model1(train, test).summary()
p_table = model2.tables[1]
p_table = pd.DataFrame(p_table.data)
p_table.columns = p_table.iloc[0]
p_table = p_table.drop(0)
p_table = p_table.set_index(p_table.columns[0])
p_table['P>|t|'] = p_table['P>|t|'].astype(float)
sig = list(p_table[p_table['P>|t|'] < 0.05].index)
sig.remove('Intercept')
print("Total # features: ", len(p_table))
print("Total significant features", len(sig))
p_table.head()
```

Train RMSE: 122255.66801937918

Test RMSE: 121587.4592701153

Total # features: 20

Total significant features 17

Out[83]:

|             | coef       | std err  | t      | P> t | [0.025    | 0.975]    |
|-------------|------------|----------|--------|------|-----------|-----------|
| Intercept   | -4.992e+07 | 5.56e+06 | -8.978 | 0.0  | -6.08e+07 | -3.9e+07  |
| bedrooms    | -1.414e+04 | 1810.270 | -7.813 | 0.0  | -1.77e+04 | -1.06e+04 |
| bathrooms   | 2.156e+04  | 2917.233 | 7.389  | 0.0  | 1.58e+04  | 2.73e+04  |
| sqft_living | 118.0345   | 18.503   | 6.379  | 0.0  | 81.766    | 154.303   |
| sqft_lot    | -6.1074    | 0.923    | -6.617 | 0.0  | -7.917    | -4.298    |

In [84]:

```
# Gathering together significant features
train, test = train[sig+['price']], test[sig+['price']]
```

In [85]:

```
# Applying only features with a p_value less than 0.05
model2 = model1(train, test)
model2.summary()
```

Train RMSE: 122284.7267523019

Test RMSE: 121596.23044719924

**Out[85]:**

OLS Regression Results

| <b>Dep. Variable:</b>    | price            | <b>R-squared:</b>          | 0.706       |       |           |           |
|--------------------------|------------------|----------------------------|-------------|-------|-----------|-----------|
| <b>Model:</b>            | OLS              | <b>Adj. R-squared:</b>     | 0.706       |       |           |           |
| <b>Method:</b>           | Least Squares    | <b>F-statistic:</b>        | 1623.       |       |           |           |
| <b>Date:</b>             | Wed, 20 Sep 2023 | <b>Prob (F-statistic):</b> | 0.00        |       |           |           |
| <b>Time:</b>             | 19:54:17         | <b>Log-Likelihood:</b>     | -1.5103e+05 |       |           |           |
| <b>No. Observations:</b> | 11500            | <b>AIC:</b>                | 3.021e+05   |       |           |           |
| <b>Df Residuals:</b>     | 11482            | <b>BIC:</b>                | 3.022e+05   |       |           |           |
| <b>Df Model:</b>         | 17               |                            |             |       |           |           |
| <b>Covariance Type:</b>  | nonrobust        |                            |             |       |           |           |
|                          | coef             | std err                    | t           | P> t  | [0.025    | 0.975]    |
| <b>Intercept</b>         | -4.914e+07       | 5.55e+06                   | -8.854      | 0.000 | -6e+07    | -3.83e+07 |
| <b>bedrooms</b>          | -1.415e+04       | 1810.415                   | -7.815      | 0.000 | -1.77e+04 | -1.06e+04 |
| <b>bathrooms</b>         | 2.036e+04        | 2867.004                   | 7.100       | 0.000 | 1.47e+04  | 2.6e+04   |
| <b>sqft_living</b>       | 110.7889         | 3.496                      | 31.688      | 0.000 | 103.936   | 117.642   |
| <b>sqft_lot</b>          | -5.7416          | 0.908                      | -6.322      | 0.000 | -7.522    | -3.961    |
| <b>floors</b>            | 1.189e+04        | 3066.633                   | 3.876       | 0.000 | 5875.209  | 1.79e+04  |
| <b>waterfront</b>        | 3.261e+05        | 3.77e+04                   | 8.653       | 0.000 | 2.52e+05  | 4e+05     |
| <b>view</b>              | 4.227e+04        | 2075.824                   | 20.361      | 0.000 | 3.82e+04  | 4.63e+04  |
| <b>condition</b>         | 2.664e+04        | 1968.122                   | 13.537      | 0.000 | 2.28e+04  | 3.05e+04  |
| <b>grade</b>             | 8.989e+04        | 1919.873                   | 46.823      | 0.000 | 8.61e+04  | 9.37e+04  |
| <b>yr_built</b>          | 2.031e+04        | 2445.193                   | 8.306       | 0.000 | 1.55e+04  | 2.51e+04  |
| <b>zipcode</b>           | -263.7018        | 28.731                     | -9.178      | 0.000 | -320.020  | -207.384  |
| <b>lat</b>               | 5.595e+05        | 9266.046                   | 60.377      | 0.000 | 5.41e+05  | 5.78e+05  |
| <b>long</b>              | -5.612e+04       | 1.19e+04                   | -4.699      | 0.000 | -7.95e+04 | -3.27e+04 |
| <b>sqft_living15</b>     | 42.2596          | 3.382                      | 12.495      | 0.000 | 35.630    | 48.889    |
| <b>sqft_lot15</b>        | -4.3441          | 0.760                      | -5.712      | 0.000 | -5.835    | -2.853    |
| <b>renovated</b>         | 2.914e+04        | 6906.844                   | 4.220       | 0.000 | 1.56e+04  | 4.27e+04  |
| <b>age_at_sold</b>       | 2.246e+04        | 2444.354                   | 9.190       | 0.000 | 1.77e+04  | 2.73e+04  |
| <b>Omnibus:</b>          | 2296.870         | <b>Durbin-Watson:</b>      | 1.979       |       |           |           |
| <b>Prob(Omnibus):</b>    | 0.000            | <b>Jarque-Bera (JB):</b>   | 7827.477    |       |           |           |
| <b>Skew:</b>             | 0.995            | <b>Prob(JB):</b>           | 0.00        |       |           |           |
| <b>Kurtosis:</b>         | 6.518            | <b>Cond. No.</b>           | 4.79e+08    |       |           |           |

Notes:

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

[2] The condition number is large, 4.79e+08. This might indicate that there are strong multicollinearity or other numerical problems.

## Check assumptions

### Prepare Continuous Data for Modeling: Checking Linearity

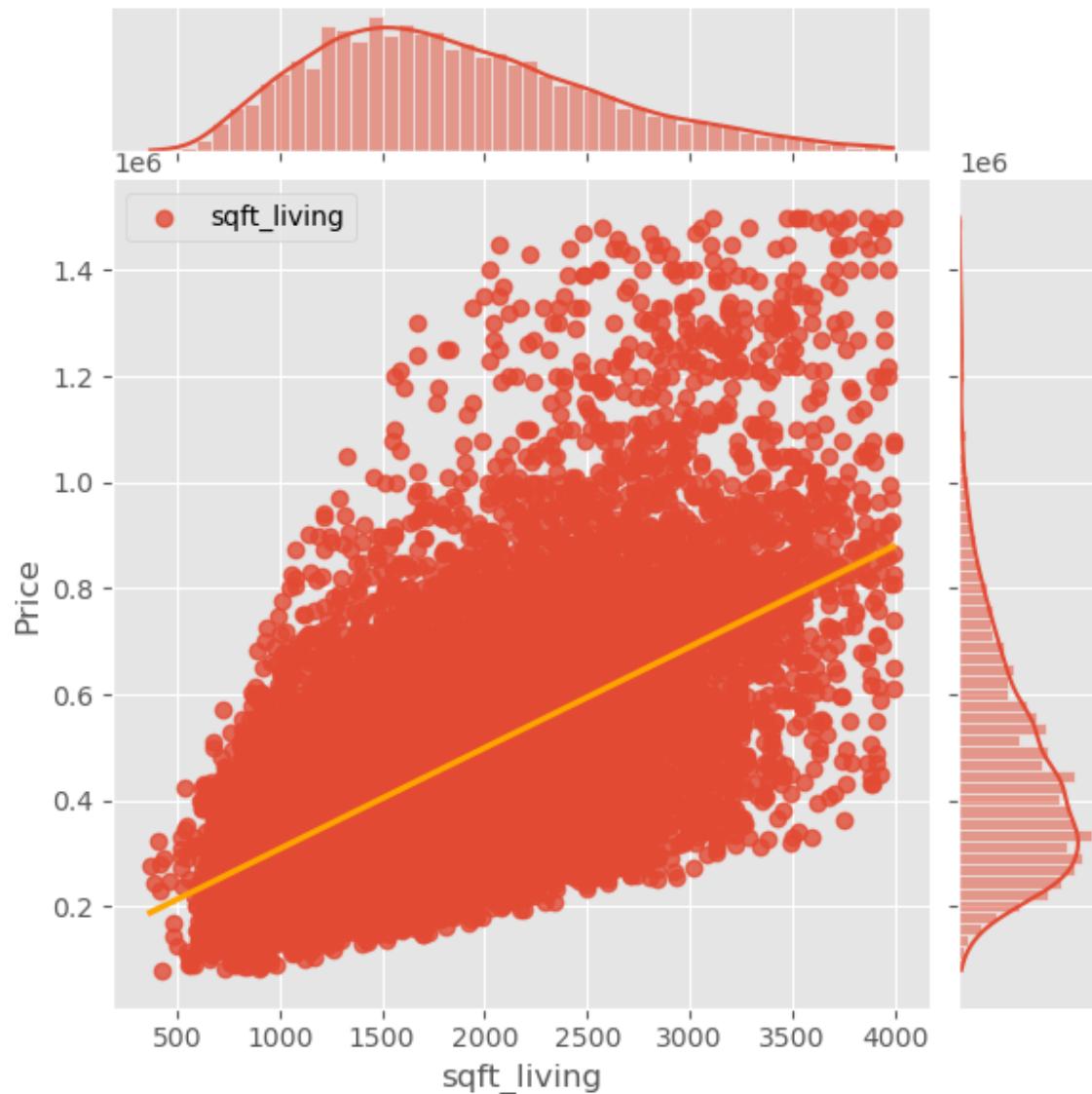
In [86]:

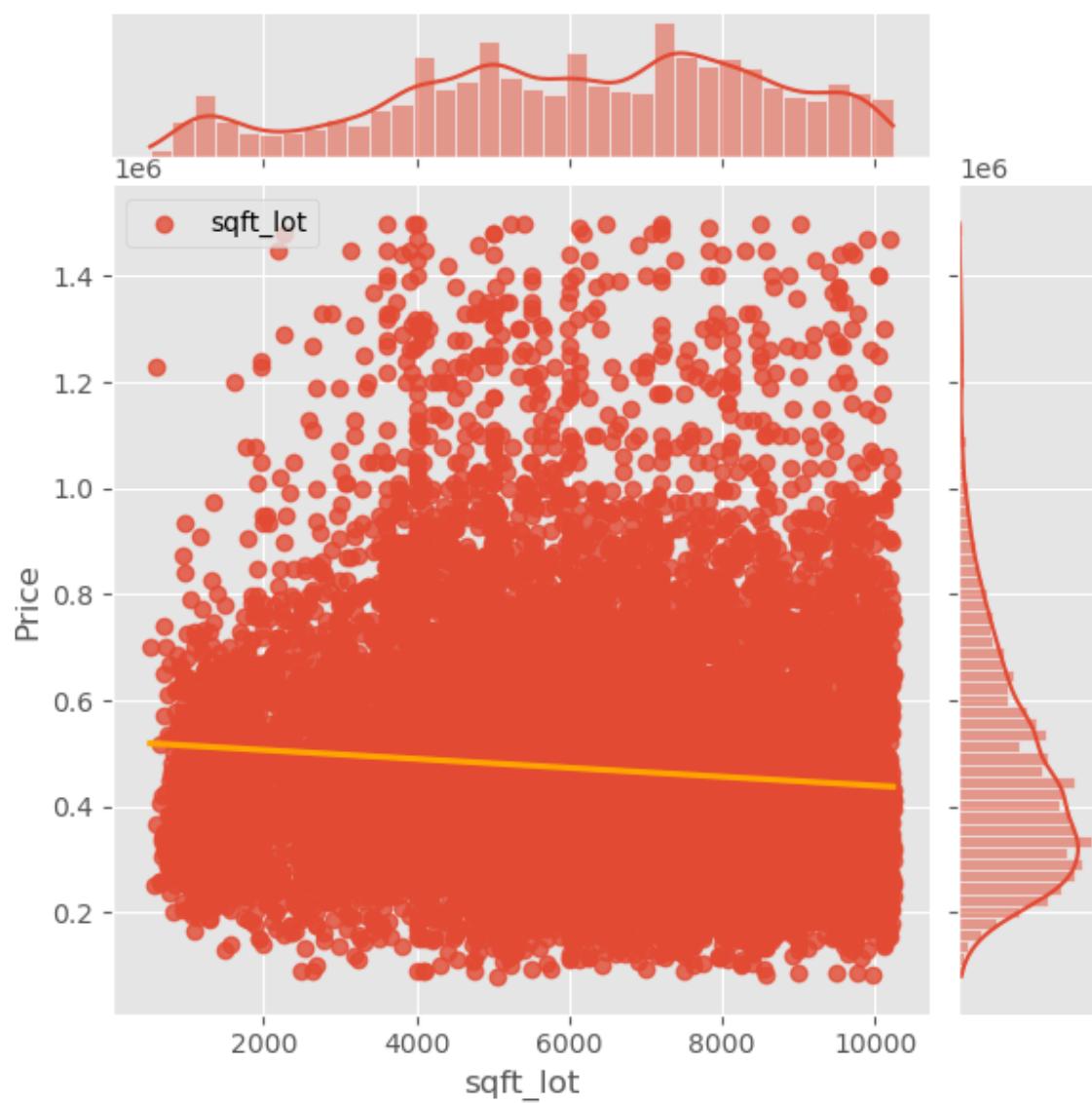
```
# Select and visualize continuous data (data that can take any value)
continuous = ['sqft_living','sqft_lot','sqft_above','sqft_living15','sqft_lot15']

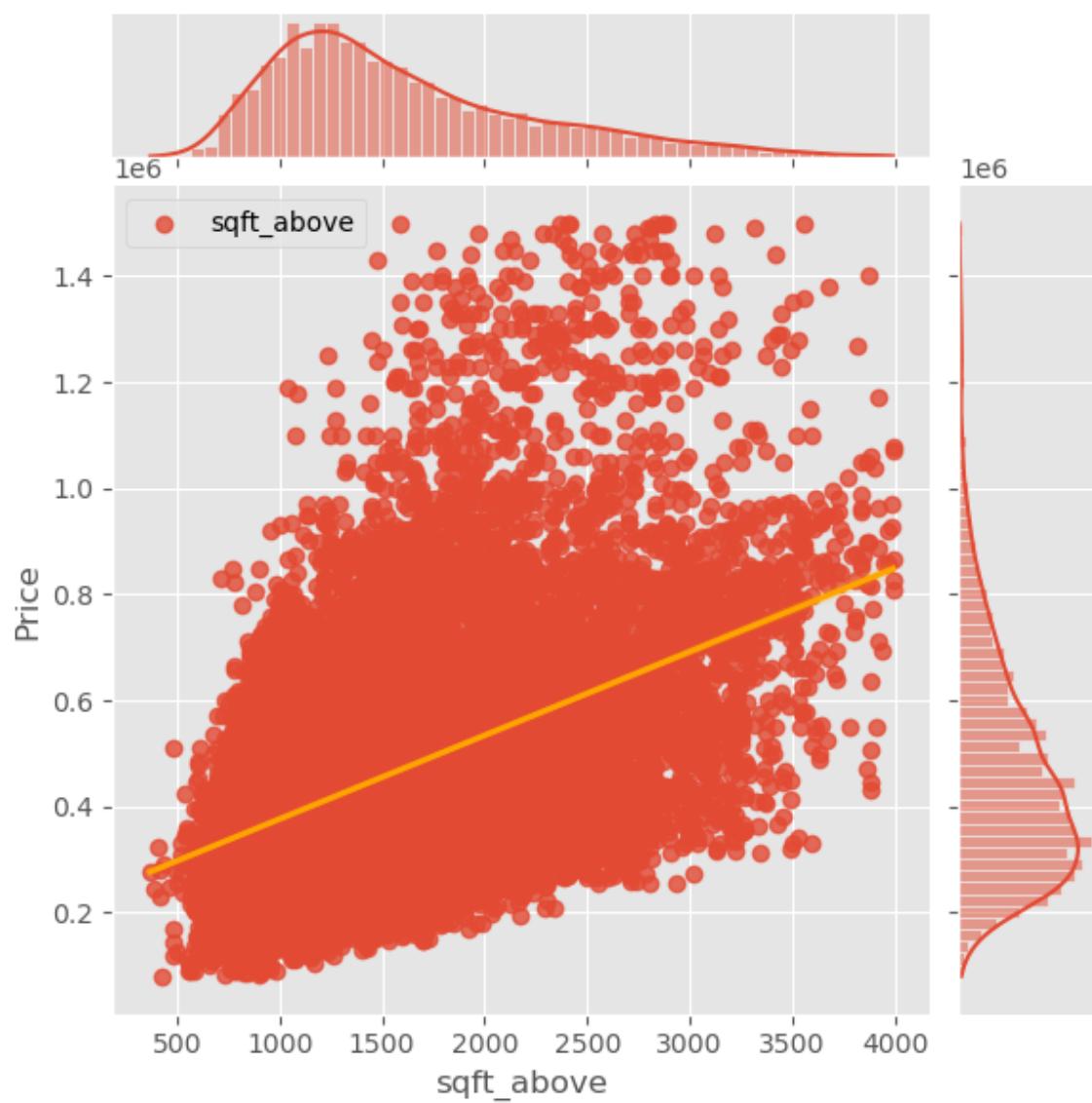
for column in continuous:
    sns.jointplot(x=column, y="price", data=df, kind='reg', label=column,joint_kws={'line_color': 'yellow' })

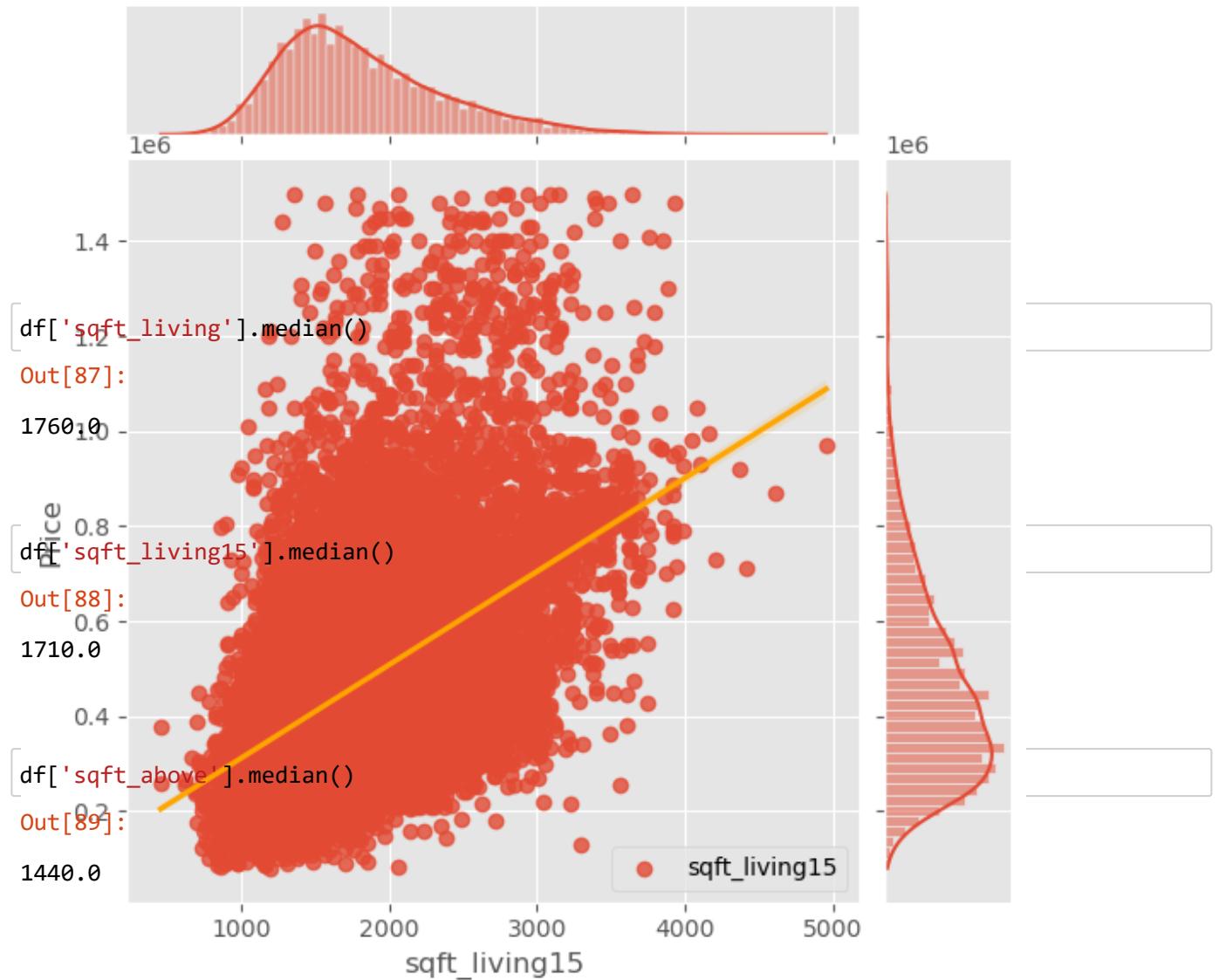
# Set y-axis label
plt.ylabel('Price')

plt.legend()
plt.show()
```



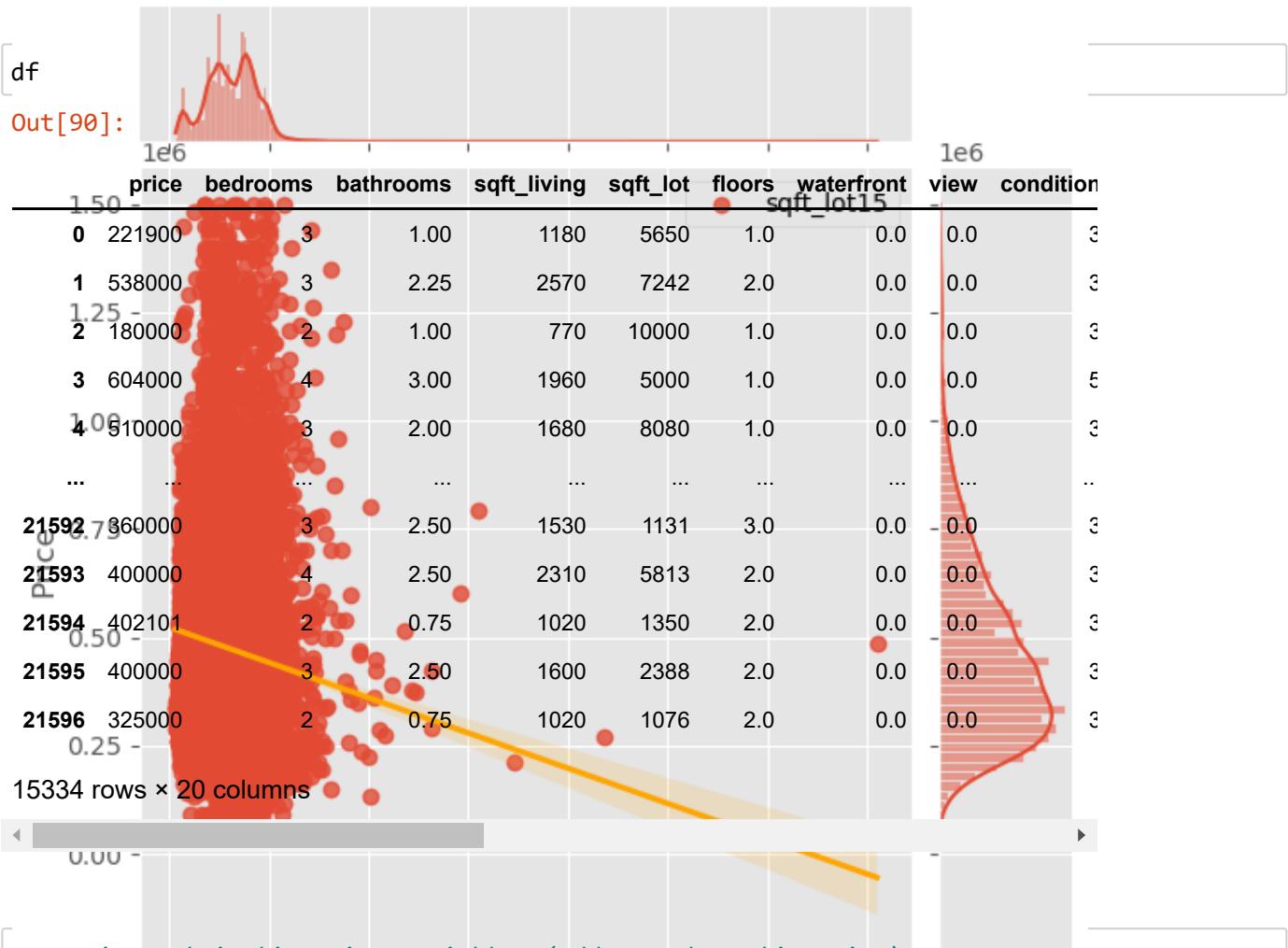






sqft\_living, sqft\_above, sqft\_lot15 all have a positive linear relationship with our target variable (price). Since continuous predictors have high multicollinearity (as addressed previously) I will use only sqft\_living

## MODEL 3 - addressed multicollinearity



```
#Removing multicollinearity variables (Address the colinearity)
df.drop(['sqft_above', 'sqft_lots', 'age_at_sold'], axis=1, inplace=True)
df.head()
```

Out[91]:

|   | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition | gr |
|---|--------|----------|-----------|-------------|----------|--------|------------|------|-----------|----|
| 0 | 221900 | 3        | 1.00      | 1180        | 5650     | 1.0    | 0.0        | 0.0  | 3         |    |
| 1 | 538000 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0.0        | 0.0  | 3         |    |
| 2 | 180000 | 2        | 1.00      | 770         | 10000    | 1.0    | 0.0        | 0.0  | 3         |    |
| 3 | 604000 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0.0        | 0.0  | 5         |    |
| 4 | 510000 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0.0        | 0.0  | 3         |    |

In [92]:

```
train, test = train_test_split(df, random_state = 150)
train.head()
```

Out[92]:

|       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition |
|-------|--------|----------|-----------|-------------|----------|--------|------------|------|-----------|
| 16187 | 447000 | 2        | 1.00      | 720         | 7500     | 1.0    | 0.0        | 2.0  | 3         |
| 6767  | 809000 | 4        | 1.75      | 1790        | 8372     | 1.0    | 0.0        | 0.0  | 4         |
| 17872 | 217000 | 2        | 1.00      | 720         | 4760     | 1.0    | 0.0        | 0.0  | 5         |
| 20242 | 778983 | 4        | 2.50      | 2490        | 5647     | 2.0    | 0.0        | 0.0  | 3         |
| 5748  | 387000 | 2        | 2.25      | 1230        | 1280     | 2.0    | 0.0        | 0.0  | 3         |

◀ ▶

In [93]:

```
# Create model function to fit a Linear regression that includes rmse
def model(train, test):

    target = 'price'
    x_cols = list(train.columns)
    x_cols.remove('price')

    predictors = '+' .join(x_cols)
    formula = target + '~' + predictors
    model = ols(formula=formula, data=train).fit()

    # RMSE
    train_err = (mean_squared_error(train['price'], model.predict(train)))**0.5
    test_err = (mean_squared_error(test['price'], model.predict(test)))**0.5

    print("Train RMSE: ", train_err)
    print("Test RMSE: ", test_err, '\n')

    return model
```

In [94]:

```
model(train, test).summary()
```

Train RMSE: 122874.48068419255

Test RMSE: 122673.49484315024

**Out[94]:**

OLS Regression Results

|                       |       |                        |       |
|-----------------------|-------|------------------------|-------|
| <b>Dep. Variable:</b> | price | <b>R-squared:</b>      | 0.703 |
| <b>Model:</b>         | OLS   | <b>Adj. R-squared:</b> | 0.703 |

After addressing the collinearity and dropping 'sqft\_living15' column there was'nt any improvement in terms of R2 square instead there was a reduction in 0.003 0.00

A skewness Time of 0.989 suggests a moderately right-skewed distribution. This means:

- No. Observations:** 11500 **AIC:** 3.022e+05  
 • Most of the data points are concentrated on the left side of the distribution.  
**Df Residuals:** 11483 **BIC:** 3.023e+05  
 • The tail of the distribution extends more towards the right, not the left.  
 • The **Df Model:** typically greater than the median in a positively skewed distribution.

**Covariance Type:** nonrobust  
 Kurtosis of 6.416 is leptokurtic, meaning it has fatter tails and a sharper peak compared to a normal distribution. This would suggest:

|                       | coef       | std err   | t                        | P> t     | [0.025    | 0.975]    |
|-----------------------|------------|-----------|--------------------------|----------|-----------|-----------|
| <b>Intercept</b>      | 4.424e+04  | 3.536e+04 | 1.240                    | 0.215    | 1.424e+04 | 7.424e+04 |
| <b>bedrooms</b>       | -1.404e+04 | 1818.999  | -7.719                   | 0.000    | -1.76e+04 | -1.05e+04 |
| <b>bathrooms</b>      | 2.194e+04  | 2926.184  | 7.498                    | 0.000    | 1.62e+04  | 2.77e+04  |
| <b>sqft_living</b>    | 115.4947   | 3.902     | 29.602                   | 0.000    | 107.847   | 123.142   |
| <b>sqft_lot</b>       | -9.8514    | 0.641     | -15.367                  | 0.000    | -11.108   | -8.595    |
| <b>floors</b>         | 8499.5620  | 3547.739  | 2.396                    | 0.017    | 1545.388  | 1.55e+04  |
| <b>waterfront</b>     | 3.196e+05  | 3.79e+04  | 8.444                    | 0.000    | 2.45e+05  | 3.94e+05  |
| <b>view</b>           | 4.301e+04  | 2102.288  | 20.457                   | 0.000    | 3.89e+04  | 4.71e+04  |
| <b>condition</b>      | 2.568e+04  | 1982.959  | 12.950                   | 0.000    | 2.18e+04  | 2.96e+04  |
| <b>grade</b>          | 8.942e+04  | 1934.990  | 46.210                   | 0.000    | 8.56e+04  | 9.32e+04  |
| <b>sqft_basement</b>  | -10.7853   | 4.406     | -2.448                   | 0.014    | -19.422   | -2.149    |
| <b>yr_built</b>       | -2178.2131 | 57.930    | -37.600                  | 0.000    | -2291.767 | -2064.659 |
| <b>zipcode</b>        | -262.4633  | 28.869    | -9.091                   | 0.000    | -319.052  | -205.874  |
| <b>lat</b>            | 5.596e+05  | 9353.707  | 59.826                   | 0.000    | 5.41e+05  | 5.78e+05  |
| <b>long</b>           | -5.99e+04  | 1.22e+04  | -4.902                   | 0.000    | -8.39e+04 | -3.59e+04 |
| <b>sqft_living15</b>  | 38.4202    | 3.437     | 11.180                   | 0.000    | 31.684    | 45.156    |
| <b>renovated</b>      | 2.545e+04  | 6923.065  | 3.676                    | 0.000    | 1.19e+04  | 3.9e+04   |
| <b>Omnibus:</b>       | 2260.095   |           | <b>Durbin-Watson:</b>    | 1.979    |           |           |
| <b>Prob(Omnibus):</b> | 0.000      |           | <b>Jarque-Bera (JB):</b> | 7467.245 |           |           |
| <b>Skew:</b>          | 0.989      |           | <b>Prob(JB):</b>         | 0.00     |           |           |
| <b>Kurtosis:</b>      | 6.416      |           | <b>Cond. No.</b>         | 2.17e+08 |           |           |

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 2.17e+08. This might indicate that there are strong multicollinearity or other numerical problems.

## NO SIGNIFICANT DIFFERENCES WERE FOUND IN MODEL 3 FROM MODEL 1 AFTER ADDRESSING THE MULTICOLLINEARITY

### Log transformation and normalisation only in continuous data

In [95]:

```
df
```

Out[95]:

|       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition |
|-------|--------|----------|-----------|-------------|----------|--------|------------|------|-----------|
| 0     | 221900 | 3        | 1.00      | 1180        | 5650     | 1.0    | 0.0        | 0.0  | 3         |
| 1     | 538000 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0.0        | 0.0  | 3         |
| 2     | 180000 | 2        | 1.00      | 770         | 10000    | 1.0    | 0.0        | 0.0  | 3         |
| 3     | 604000 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0.0        | 0.0  | 5         |
| 4     | 510000 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0.0        | 0.0  | 3         |
| ...   | ...    | ...      | ...       | ...         | ...      | ...    | ...        | ...  | ..        |
| 21592 | 360000 | 3        | 2.50      | 1530        | 1131     | 3.0    | 0.0        | 0.0  | 3         |
| 21593 | 400000 | 4        | 2.50      | 2310        | 5813     | 2.0    | 0.0        | 0.0  | 3         |
| 21594 | 402101 | 2        | 0.75      | 1020        | 1350     | 2.0    | 0.0        | 0.0  | 3         |
| 21595 | 400000 | 3        | 2.50      | 1600        | 2388     | 2.0    | 0.0        | 0.0  | 3         |
| 21596 | 325000 | 2        | 0.75      | 1020        | 1076     | 2.0    | 0.0        | 0.0  | 3         |

15334 rows × 17 columns

◀ ▶

In [96]:

```
df.describe()
```

Out[96]:

|       | price        | bedrooms     | bathrooms    | sqft_living  | sqft_lot     | floors       |
|-------|--------------|--------------|--------------|--------------|--------------|--------------|
| count | 1.533400e+04 | 15334.000000 | 15334.000000 | 15334.000000 | 15334.000000 | 15334.000000 |
| mean  | 4.721766e+05 | 3.262880     | 2.007125     | 1857.591626  | 6097.929829  | 1.504695     |
| std   | 2.240077e+05 | 0.852576     | 0.687194     | 684.015787   | 2427.995819  | 0.553630     |
| min   | 8.000000e+04 | 1.000000     | 0.500000     | 370.000000   | 520.000000   | 1.000000     |
| 25%   | 3.090000e+05 | 3.000000     | 1.500000     | 1340.000000  | 4400.000000  | 1.000000     |
| 50%   | 4.250000e+05 | 3.000000     | 2.000000     | 1760.000000  | 6250.000000  | 1.500000     |
| 75%   | 5.847125e+05 | 4.000000     | 2.500000     | 2290.000000  | 8007.500000  | 2.000000     |
| max   | 1.500000e+06 | 6.000000     | 3.750000     | 3990.000000  | 10232.000000 | 3.500000     |

◀ ▶

# MODEL 4 - USING LOG TRANSFORMATION FOR CONTINUOUS DATA

In [97]:

```
outcome = 'price'
x_cols = ['sqft_living', 'sqft_lot', 'sqft_living15', 'sqft_basement']
predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model = ols(formula=formula, data=df).fit()
model.summary()
```

Out[97]:

OLS Regression Results

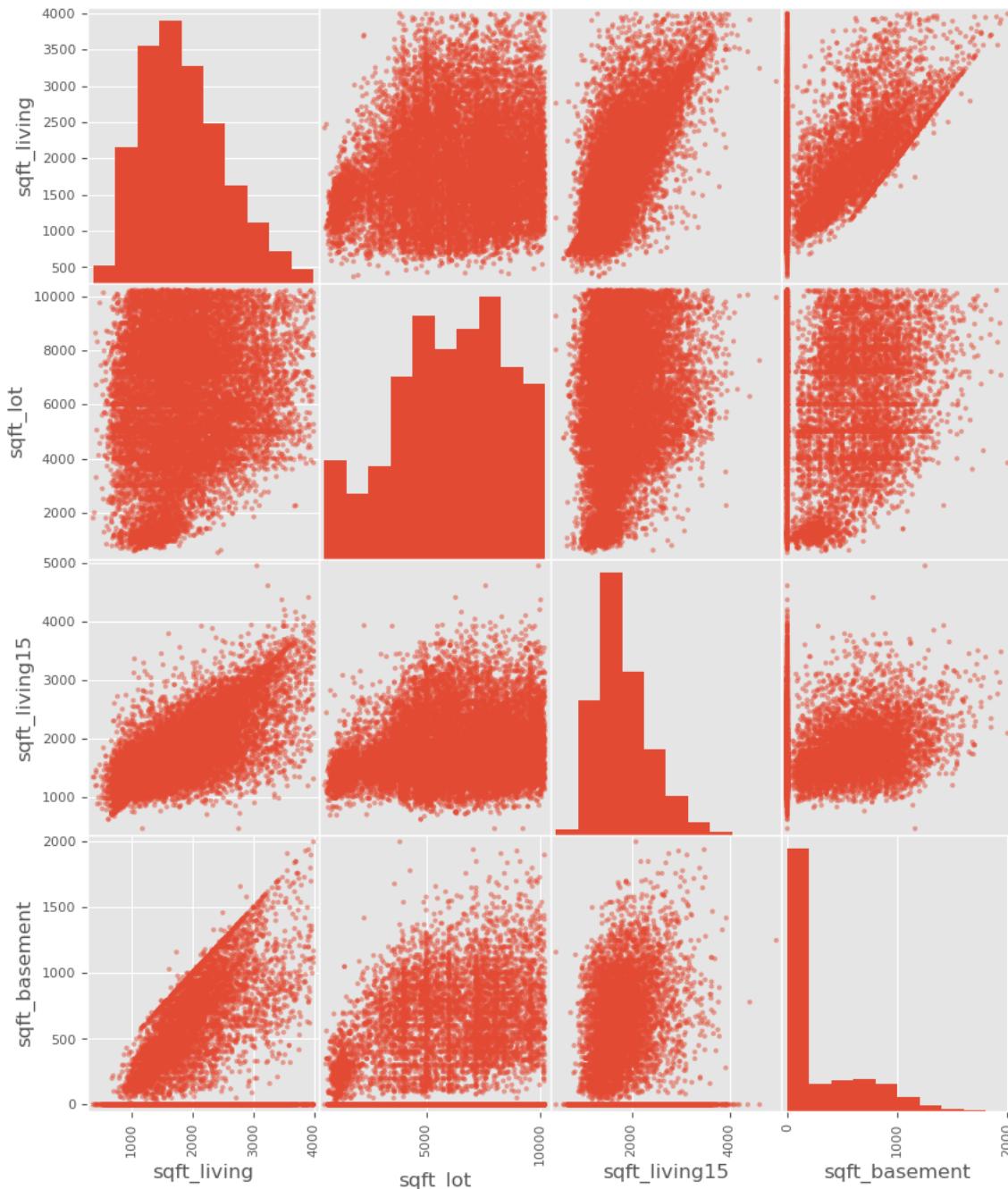
| <b>Dep. Variable:</b>    | price            | <b>R-squared:</b>          | 0.398       |       |          |          |
|--------------------------|------------------|----------------------------|-------------|-------|----------|----------|
| <b>Model:</b>            | OLS              | <b>Adj. R-squared:</b>     | 0.398       |       |          |          |
| <b>Method:</b>           | Least Squares    | <b>F-statistic:</b>        | 2534.       |       |          |          |
| <b>Date:</b>             | Wed, 20 Sep 2023 | <b>Prob (F-statistic):</b> | 0.00        |       |          |          |
| <b>Time:</b>             | 19:54:25         | <b>Log-Likelihood:</b>     | -2.0677e+05 |       |          |          |
| <b>No. Observations:</b> | 15334            | <b>AIC:</b>                | 4.136e+05   |       |          |          |
| <b>Df Residuals:</b>     | 15329            | <b>BIC:</b>                | 4.136e+05   |       |          |          |
| <b>Df Model:</b>         | 4                |                            |             |       |          |          |
| <b>Covariance Type:</b>  | nonrobust        |                            |             |       |          |          |
|                          | coef             | std err                    | t           | P> t  | [0.025   | 0.975]   |
| <b>Intercept</b>         | 1.644e+05        | 5554.947                   | 29.595      | 0.000 | 1.54e+05 | 1.75e+05 |
| <b>sqft_living</b>       | 141.8809         | 3.330                      | 42.602      | 0.000 | 135.353  | 148.409  |
| <b>sqft_lot</b>          | -19.7057         | 0.591                      | -33.336     | 0.000 | -20.864  | -18.547  |
| <b>sqft_living15</b>     | 81.6155          | 3.901                      | 20.922      | 0.000 | 73.969   | 89.262   |
| <b>sqft_basement</b>     | 65.2759          | 4.186                      | 15.595      | 0.000 | 57.071   | 73.480   |
| <b>Omnibus:</b>          | 2414.066         | <b>Durbin-Watson:</b>      | 1.980       |       |          |          |
| <b>Prob(Omnibus):</b>    | 0.000            | <b>Jarque-Bera (JB):</b>   | 5066.425    |       |          |          |
| <b>Skew:</b>             | 0.946            | <b>Prob(JB):</b>           | 0.00        |       |          |          |
| <b>Kurtosis:</b>         | 5.087            | <b>Cond. No.</b>           | 2.78e+04    |       |          |          |

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 2.78e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In [98]:

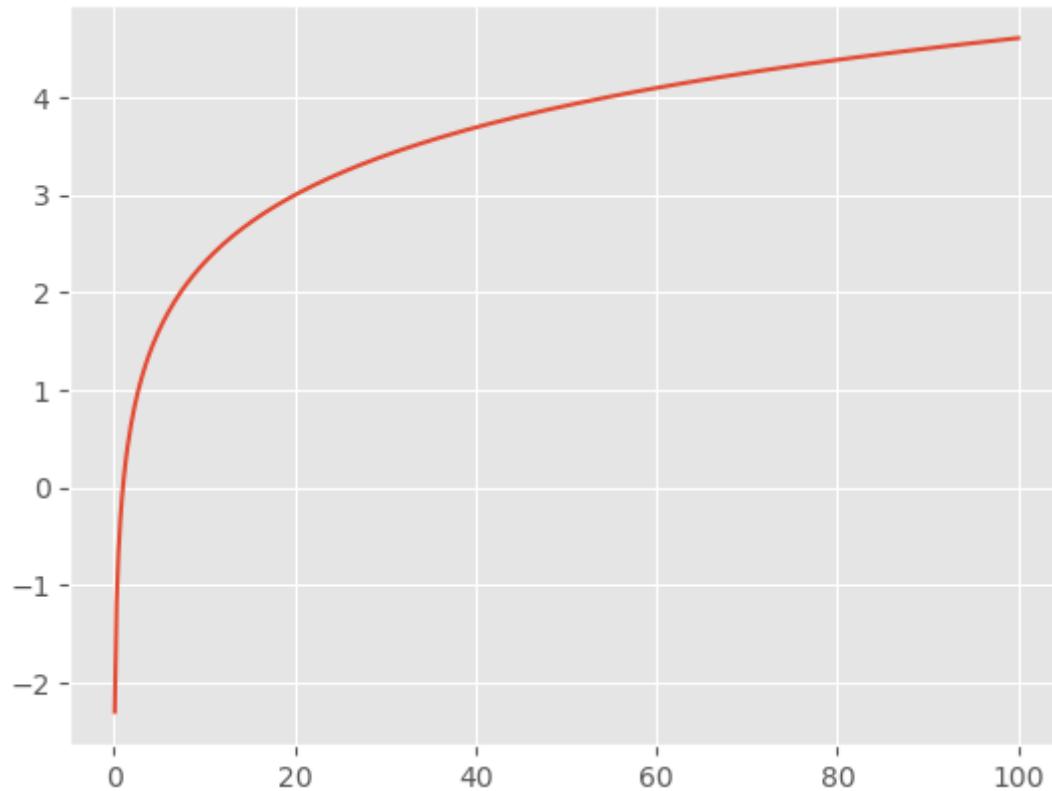
```
pd.plotting.scatter_matrix(df[x_cols], figsize=(10,12));
```



In [99]:

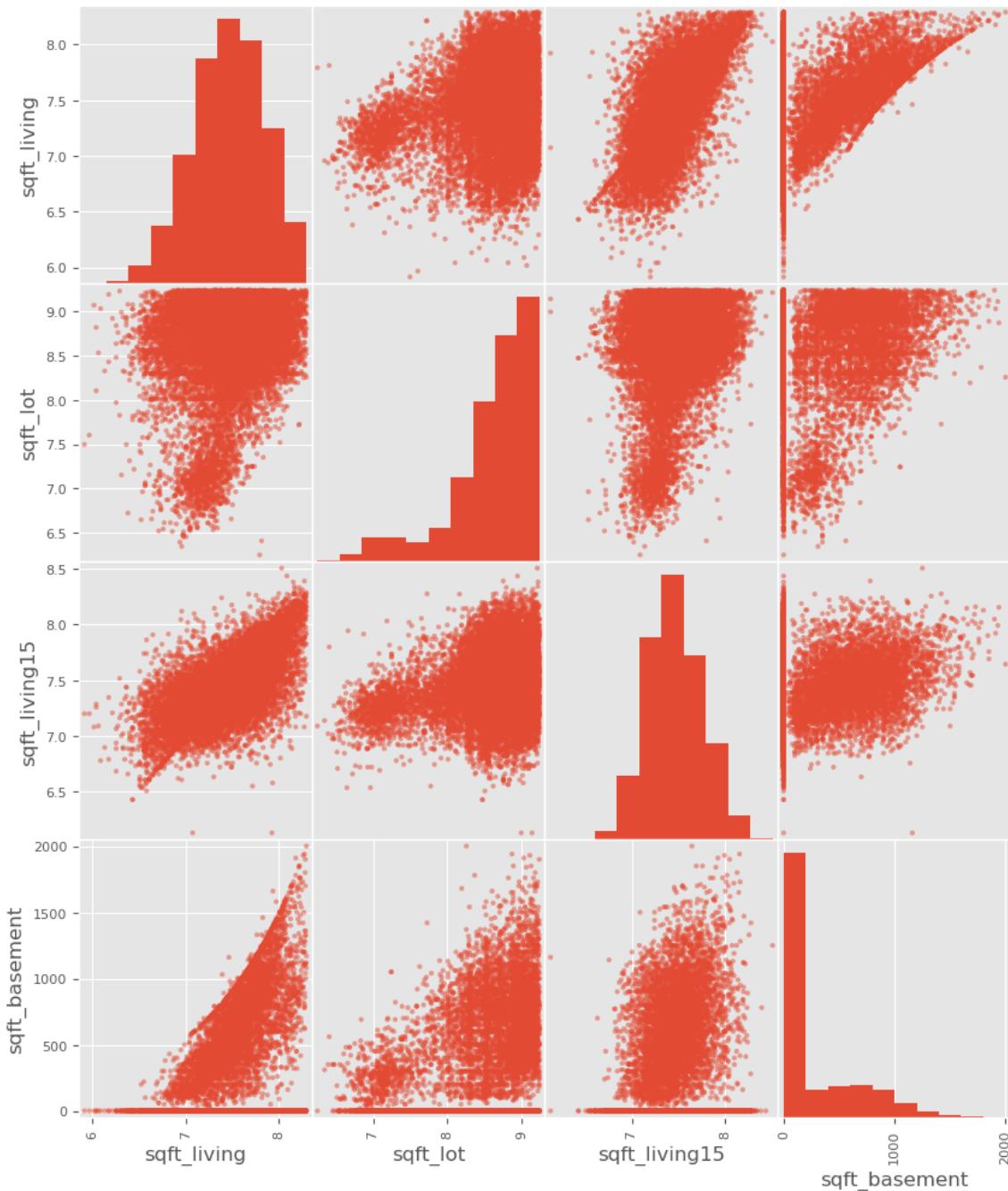
```
x = np.linspace(start=-100, stop=100, num=10**3)
y = np.log(x)
plt.plot(x, y);
```

C:\Users\rani\_\AppData\Local\Temp\ipykernel\_14464\762412792.py:2: RuntimeWarning: invalid value encountered in log  
y = np.log(x)



In [100]:

```
non_normal = ['sqft_living', 'sqft_lot', 'sqft_living15']
for feat in non_normal:
    df[feat] = df[feat].map(lambda x: np.log(x))
pd.plotting.scatter_matrix(df[x_cols], figsize=(10,12));
```



In [101]:

```
outcome = 'price'
x_cols = ['sqft_living', 'sqft_lot', 'sqft_living15', 'sqft_basement']
predictors = '+' .join(x_cols)
formula = outcome + '~' + predictors
model = ols(formula=formula, data=df).fit()
model.summary()
```

Out[101]:

OLS Regression Results

| <b>Dep. Variable:</b>    | price            | <b>R-squared:</b>          | 0.362       |       |           |          |
|--------------------------|------------------|----------------------------|-------------|-------|-----------|----------|
| <b>Model:</b>            | OLS              | <b>Adj. R-squared:</b>     | 0.361       |       |           |          |
| <b>Method:</b>           | Least Squares    | <b>F-statistic:</b>        | 2170.       |       |           |          |
| <b>Date:</b>             | Wed, 20 Sep 2023 | <b>Prob (F-statistic):</b> | 0.00        |       |           |          |
| <b>Time:</b>             | 19:54:30         | <b>Log-Likelihood:</b>     | -2.0722e+05 |       |           |          |
| <b>No. Observations:</b> | 15334            | <b>AIC:</b>                | 4.145e+05   |       |           |          |
| <b>Df Residuals:</b>     | 15329            | <b>BIC:</b>                | 4.145e+05   |       |           |          |
| <b>Df Model:</b>         | 4                |                            |             |       |           |          |
| <b>Covariance Type:</b>  | nonrobust        |                            |             |       |           |          |
|                          | coef             | std err                    | t           | P> t  | [0.025    | 0.975]   |
| <b>Intercept</b>         | -1.884e+06       | 4.02e+04                   | -46.863     | 0.000 | -1.96e+06 | -1.8e+06 |
| <b>sqft_living</b>       | 2.24e+05         | 5952.604                   | 37.623      | 0.000 | 2.12e+05  | 2.36e+05 |
| <b>sqft_lot</b>          | -7.83e+04        | 2710.864                   | -28.885     | 0.000 | -8.36e+04 | -7.3e+04 |
| <b>sqft_living15</b>     | 1.798e+05        | 7254.859                   | 24.789      | 0.000 | 1.66e+05  | 1.94e+05 |
| <b>sqft_basement</b>     | 68.8601          | 4.277                      | 16.099      | 0.000 | 60.476    | 77.244   |
| <b>Omnibus:</b>          | 2590.857         | <b>Durbin-Watson:</b>      | 1.986       |       |           |          |
| <b>Prob(Omnibus):</b>    | 0.000            | <b>Jarque-Bera (JB):</b>   | 5385.142    |       |           |          |
| <b>Skew:</b>             | 1.011            | <b>Prob(JB):</b>           | 0.00        |       |           |          |
| <b>Kurtosis:</b>         | 5.083            | <b>Cond. No.</b>           | 1.25e+04    |       |           |          |

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 1.25e+04. This might indicate that there are strong multicollinearity or other numerical problems.

### Comparing OLS regression results before and after log transformation for continuous data

- The R-squared is 0.398 for the first model and 0.362 for the second, which means the model explains around 47-45% of the variability in price.
- The variable sqft\_basement seems to be not statistically significant in both models (high p-value).
- Both models suffer from non-normality in the residuals, as evidenced by the Omnibus and JB tests.
- The condition number (Cond. No.) is lower in the second model, which might imply less multicollinearity.

- The increase in skewness from 0.946 to 1.011 indicates that the distribution has become more positively skewed, suggesting more frequent higher values in the dataset.
- The increase in kurtosis from 5.467 to 5.083 suggests that the distribution has become more "tailed,"

## INTERPRET

The final step is interpreting the results of the models.

## Summary of Models

- Model 1 -> R- SQUARE = 0.706 , Number of Features = 19 , Best Model =  No change in data except dropped one column (ID)
- Model 2 -> R- SQUARE = 0.706 , Number of Features = 17 , Best Model =  Removed P-value greater than 0.05 for 3 column (sqft\_above and sqft\_basement)
- Model 3 -> R- SQUARE = 0.703 , Number of Features = 17 , Best Model =  Addressed multicollinearity and compared to Model 1
- Model 4 -> R- SQUARE = 0.362 , Number of Features = 5 , Best Model =  Applied Log transformation only for continuous data

## ANOTHER APPROACH !

## Let's first Standardize the variables

### MODEL 5 Using selected variables

In [102]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
col_df = df['sqft_living15'].values.reshape(-1,1)
scale_df = scaler.fit_transform(col_df)
df['sqft_living15_sc'] = scale_df.flatten()

x = df['sqft_living15_sc']
ax = sns.distplot(x)
```

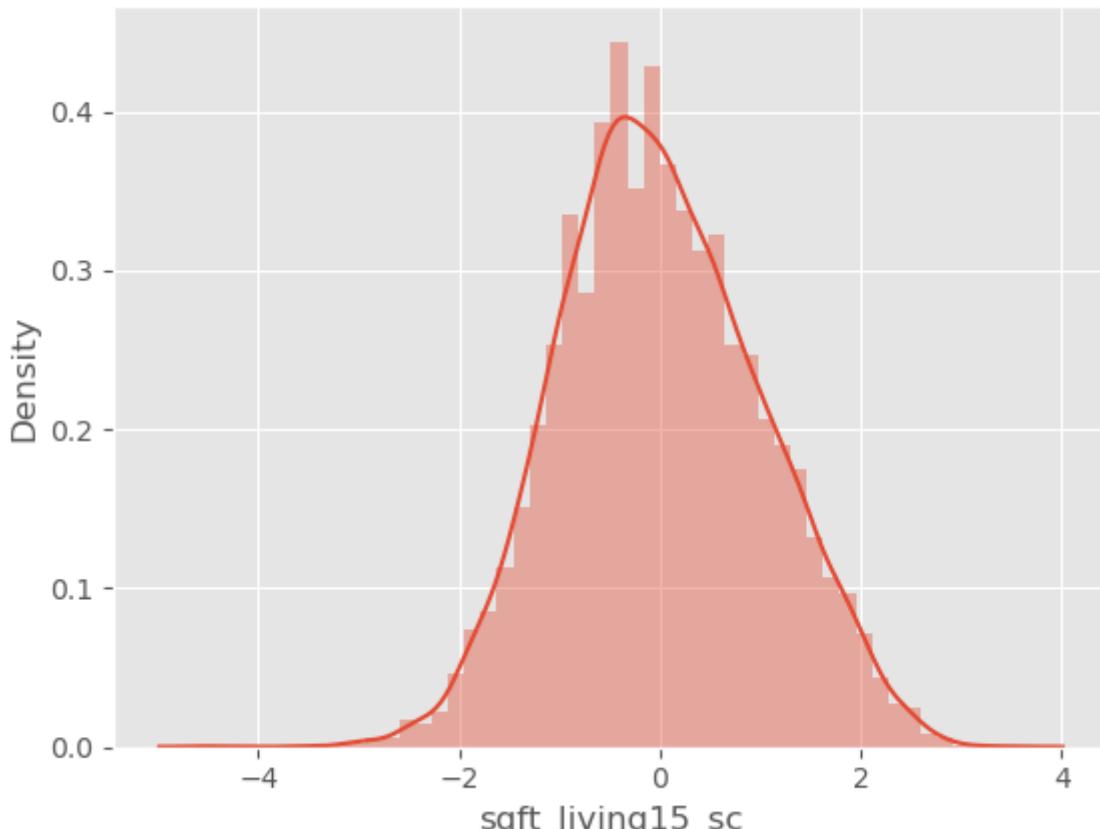
C:\Users\rani\_\AppData\Local\Temp\ipykernel\_14464\1073268049.py:8: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see <https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751> (<https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>)

```
ax = sns.distplot(x)
```



In [103]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
col_df = df['sqft_living'].values.reshape(-1,1)
scale_df = scaler.fit_transform(col_df)
df['sqft_living_sc'] = scale_df.flatten()

x = df['sqft_living_sc']
ax = sns.distplot(x)
```

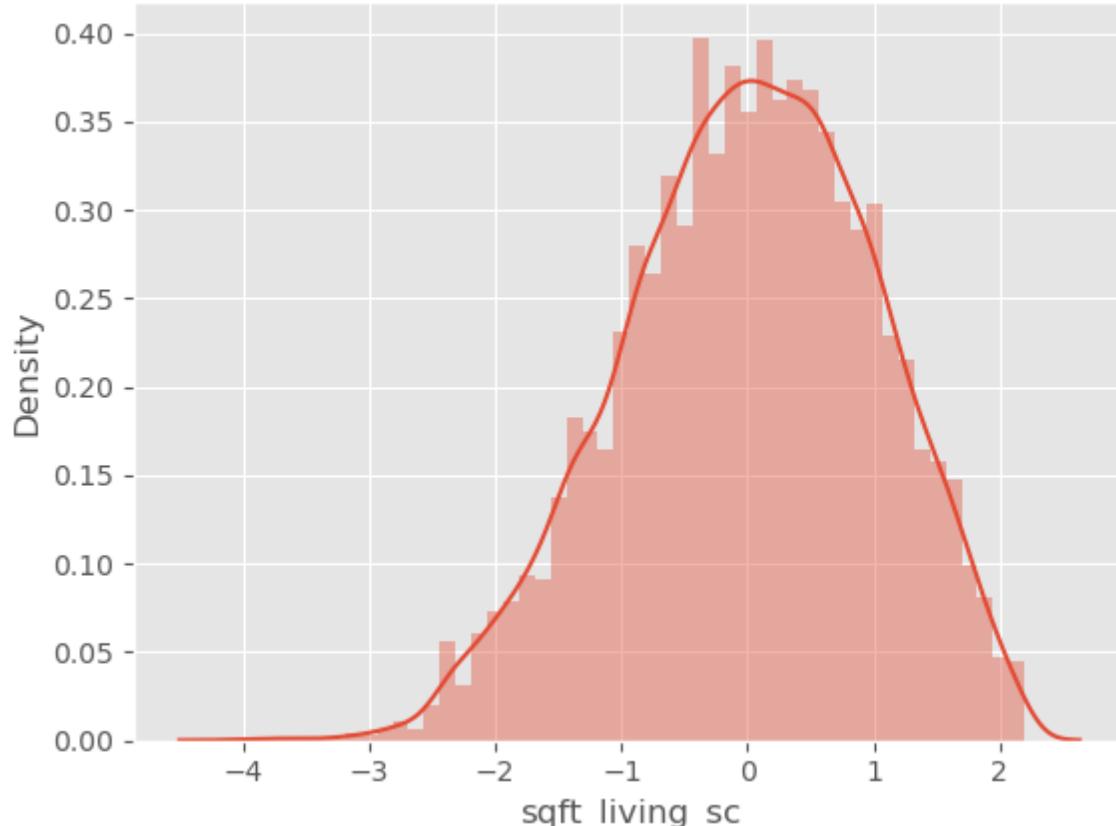
C:\Users\rani\_\AppData\Local\Temp\ipykernel\_14464\896630809.py:8: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

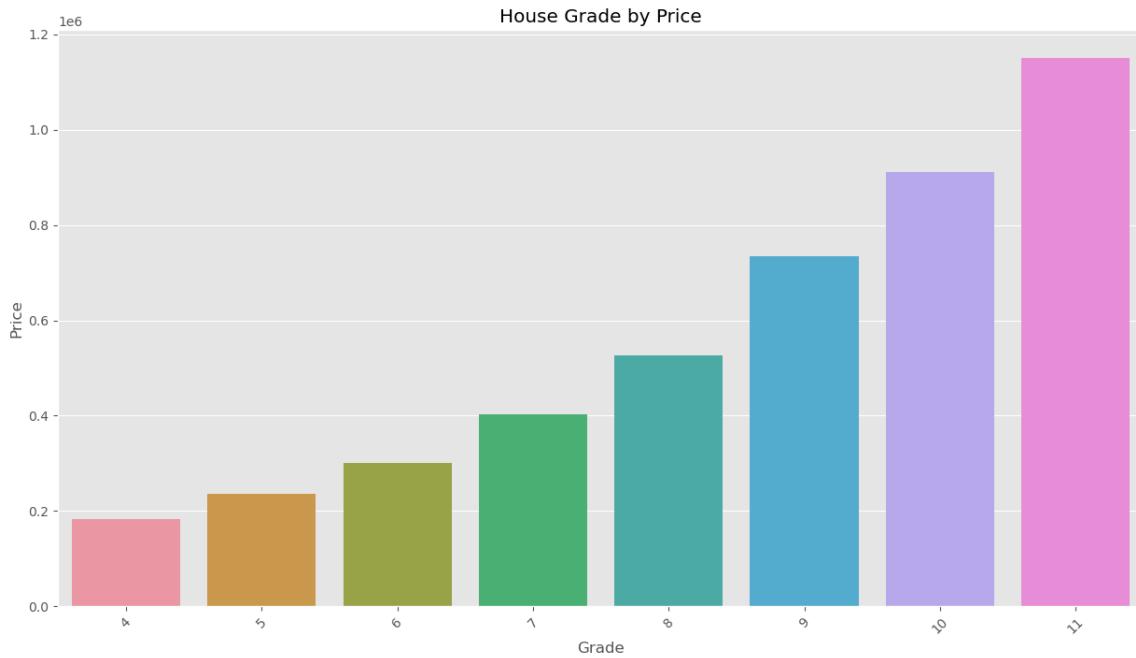
For a guide to updating your code to use the new functions, please see <https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751> (<https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>)

```
ax = sns.distplot(x)
```



In [124]:

```
plt.figure(figsize=(15, 8))
sns.barplot(x='grade', y='price', data=df, errorbar=None)
plt.xticks(rotation=45)
plt.xlabel('Grade')
plt.ylabel('Price')
plt.title('House Grade by Price')
plt.show()
```



#### Building grade:

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

- 1-3 Falls short of minimum building standards. Normally cabin or inferior structure.
- 4 Generally older, low quality construction. Does not meet code.
- 5 Low construction costs and workmanship. Small, simple design.
- 6 Lowest grade currently meeting building code. Low quality materials and simple designs.
- 7 Average grade of construction and design. Commonly seen in plats and older sub-divisions.
- 8 Just above average in construction and design. Usually better materials in both the exterior and interior finish work.
- 9 Better architectural design with extra interior and exterior design and quality.
- 10 Homes of this quality generally have high quality features. Finish work is better and more design quality is seen in the floor plans. Generally have a larger square footage.
- 11 Custom design and higher quality finish work with added amenities of solid woods, bathroom fixtures and more luxurious options.
- 12 Custom design and excellent builders. All materials are of the highest quality and all conveniences are present.

13 Generally custom designed and built. Mansion level. Large amount of highest quality cabinet work, wood

The visualization above shows that when the building grade improves, the house price rises as well. We can see in the bar plot that the mean house price for a home with a grade of 11 is far above other grades. While the building grade of 4 falls short of minimum building standards based on the King County grading system.

In [105]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
col_df = df['grade'].values.reshape(-1,1)
scale_df = scaler.fit_transform(col_df)
df['grade_sc'] = scale_df.flatten()

x = df['grade_sc']
ax = sns.distplot(x)
```

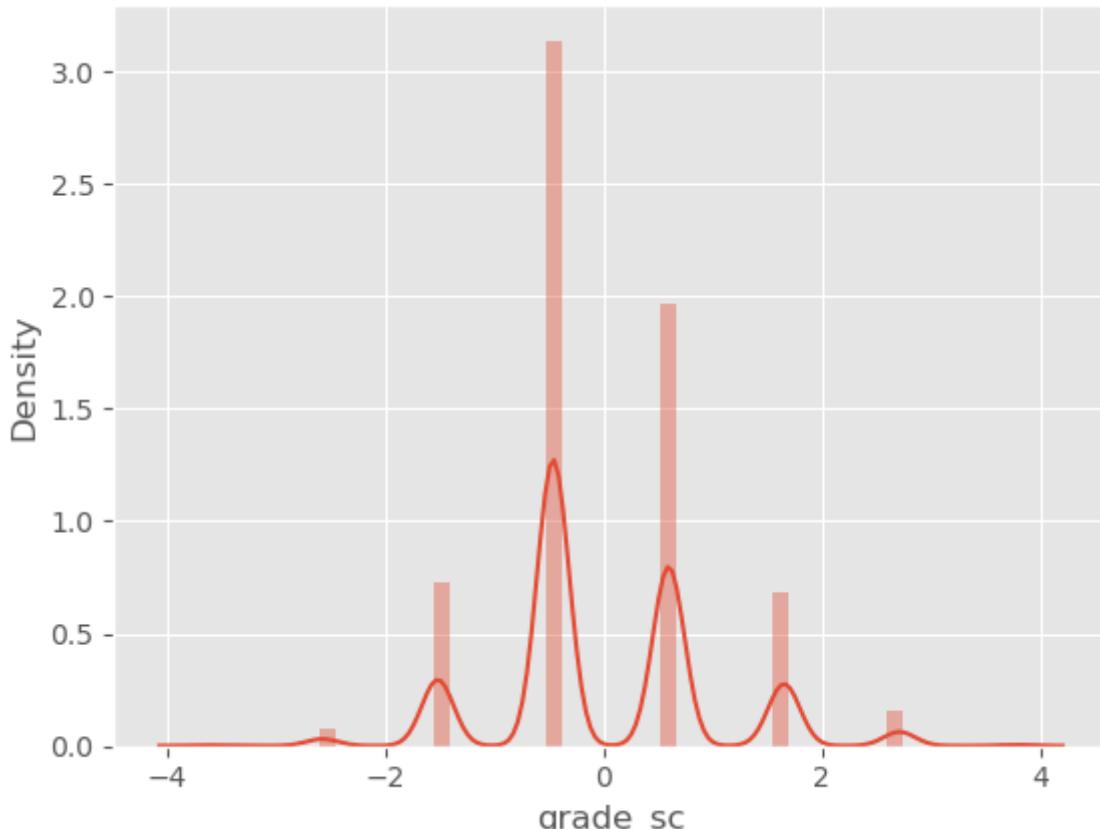
C:\Users\rani\_\AppData\Local\Temp\ipykernel\_14464\3865672094.py:8: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see <https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751> (<https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>)

```
ax = sns.distplot(x)
```



In [106]:

```
f = 'price~grade_sc+sqft_living_sc+sqft_living15_sc'
model = ols(formula=f, data=df).fit()
model.summary()
```

Out[106]:

OLS Regression Results

| <b>Dep. Variable:</b>    | price            | <b>R-squared:</b>          | 0.404       |       |          |          |
|--------------------------|------------------|----------------------------|-------------|-------|----------|----------|
| <b>Model:</b>            | OLS              | <b>Adj. R-squared:</b>     | 0.404       |       |          |          |
| <b>Method:</b>           | Least Squares    | <b>F-statistic:</b>        | 3460.       |       |          |          |
| <b>Date:</b>             | Wed, 20 Sep 2023 | <b>Prob (F-statistic):</b> | 0.00        |       |          |          |
| <b>Time:</b>             | 19:54:32         | <b>Log-Likelihood:</b>     | -2.0670e+05 |       |          |          |
| <b>No. Observations:</b> | 15334            | <b>AIC:</b>                | 4.134e+05   |       |          |          |
| <b>Df Residuals:</b>     | 15330            | <b>BIC:</b>                | 4.134e+05   |       |          |          |
| <b>Df Model:</b>         | 3                |                            |             |       |          |          |
| <b>Covariance Type:</b>  | nonrobust        |                            |             |       |          |          |
|                          | coef             | std err                    | t           | P> t  | [0.025   | 0.975]   |
| <b>Intercept</b>         | 4.722e+05        | 1397.045                   | 337.982     | 0.000 | 4.69e+05 | 4.75e+05 |
| <b>grade_sc</b>          | 9.106e+04        | 1934.308                   | 47.075      | 0.000 | 8.73e+04 | 9.48e+04 |
| <b>sqft_living_sc</b>    | 5.414e+04        | 2178.768                   | 24.848      | 0.000 | 4.99e+04 | 5.84e+04 |
| <b>sqft_living15_sc</b>  | 1.245e+04        | 2050.301                   | 6.073       | 0.000 | 8432.070 | 1.65e+04 |
| <b>Omnibus:</b>          | 2142.294         | <b>Durbin-Watson:</b>      | 1.961       |       |          |          |
| <b>Prob(Omnibus):</b>    | 0.000            | <b>Jarque-Bera (JB):</b>   | 3825.446    |       |          |          |
| <b>Skew:</b>             | 0.914            | <b>Prob(JB):</b>           | 0.00        |       |          |          |
| <b>Kurtosis:</b>         | 4.628            | <b>Cond. No.</b>           | 2.88        |       |          |          |

Notes:

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

We can see that after standardization of our selected features our model performed poorly than before COMPARING TO MODEL 1 AND 2.

In [107]:

```
#include the zipcode column to the model
#zipcodes have no numerical significance in our data, this variable is treated as categorical
f = 'price~grade_sc+sqft_living_sc+sqft_living15_sc+C(zipcode)'
model = ols(formula=f, data=df).fit()
model.summary()
```

**Out[107]:**

OLS Regression Results

| <b>Dep. Variable:</b>      | price            | <b>R-squared:</b>          | 0.775       |                 |               |               |
|----------------------------|------------------|----------------------------|-------------|-----------------|---------------|---------------|
| <b>Model:</b>              | OLS              | <b>Adj. R-squared:</b>     | 0.774       |                 |               |               |
| <b>Method:</b>             | Least Squares    | <b>F-statistic:</b>        | 730.7       |                 |               |               |
| <b>Date:</b>               | Wed, 20 Sep 2023 | <b>Prob (F-statistic):</b> | 0.00        |                 |               |               |
| <b>Time:</b>               | 19:54:32         | <b>Log-Likelihood:</b>     | -1.9922e+05 |                 |               |               |
| <b>No. Observations:</b>   | 15334            | <b>AIC:</b>                | 3.986e+05   |                 |               |               |
| <b>Df Residuals:</b>       | 15261            | <b>BIC:</b>                | 3.991e+05   |                 |               |               |
| <b>Df Model:</b>           | 72               |                            |             |                 |               |               |
| <b>Covariance Type:</b>    | nonrobust        |                            |             |                 |               |               |
|                            | <b>coef</b>      | <b>std err</b>             | <b>t</b>    | <b>P&gt; t </b> | <b>[0.025</b> | <b>0.975]</b> |
| <b>Intercept</b>           | 2.625e+05        | 7103.962                   | 36.951      | 0.000           | 2.49e+05      | 2.76e+05      |
| <b>C(zipcode)[T.98002]</b> | 5.112e+04        | 1.09e+04                   | 4.684       | 0.000           | 2.97e+04      | 7.25e+04      |
| <b>C(zipcode)[T.98003]</b> | 1.568e+04        | 1.05e+04                   | 1.496       | 0.135           | -4859.404     | 3.62e+04      |
| <b>C(zipcode)[T.98004]</b> | 5.824e+05        | 1.3e+04                    | 44.957      | 0.000           | 5.57e+05      | 6.08e+05      |
| <b>C(zipcode)[T.98005]</b> | 3.531e+05        | 1.68e+04                   | 21.022      | 0.000           | 3.2e+05       | 3.86e+05      |
| <b>C(zipcode)[T.98006]</b> | 3.047e+05        | 1.01e+04                   | 30.143      | 0.000           | 2.85e+05      | 3.25e+05      |
| <b>C(zipcode)[T.98007]</b> | 2.585e+05        | 1.31e+04                   | 19.750      | 0.000           | 2.33e+05      | 2.84e+05      |
| <b>C(zipcode)[T.98008]</b> | 2.603e+05        | 1.03e+04                   | 25.320      | 0.000           | 2.4e+05       | 2.8e+05       |
| <b>C(zipcode)[T.98010]</b> | 1.201e+05        | 2.1e+04                    | 5.713       | 0.000           | 7.89e+04      | 1.61e+05      |
| <b>C(zipcode)[T.98011]</b> | 1.485e+05        | 1.16e+04                   | 12.824      | 0.000           | 1.26e+05      | 1.71e+05      |
| <b>C(zipcode)[T.98014]</b> | 1.754e+05        | 2.61e+04                   | 6.716       | 0.000           | 1.24e+05      | 2.27e+05      |
| <b>C(zipcode)[T.98019]</b> | 8.871e+04        | 1.32e+04                   | 6.704       | 0.000           | 6.28e+04      | 1.15e+05      |
| <b>C(zipcode)[T.98022]</b> | 4.537e+04        | 1.21e+04                   | 3.756       | 0.000           | 2.17e+04      | 6.9e+04       |
| <b>C(zipcode)[T.98023]</b> | -6358.3963       | 8958.869                   | -0.710      | 0.478           | -2.39e+04     | 1.12e+04      |
| <b>C(zipcode)[T.98024]</b> | 2.03e+05         | 3.44e+04                   | 5.894       | 0.000           | 1.35e+05      | 2.7e+05       |
| <b>C(zipcode)[T.98027]</b> | 2.324e+05        | 1.04e+04                   | 22.337      | 0.000           | 2.12e+05      | 2.53e+05      |
| <b>C(zipcode)[T.98028]</b> | 1.328e+05        | 1.1e+04                    | 12.094      | 0.000           | 1.11e+05      | 1.54e+05      |
| <b>C(zipcode)[T.98029]</b> | 2.22e+05         | 9600.360                   | 23.121      | 0.000           | 2.03e+05      | 2.41e+05      |
| <b>C(zipcode)[T.98030]</b> | 6114.1759        | 1.03e+04                   | 0.591       | 0.555           | -1.42e+04     | 2.64e+04      |
| <b>C(zipcode)[T.98031]</b> | 1.473e+04        | 1.02e+04                   | 1.444       | 0.149           | -5267.858     | 3.47e+04      |
| <b>C(zipcode)[T.98032]</b> | 2.38e+04         | 1.31e+04                   | 1.812       | 0.070           | -1942.272     | 4.95e+04      |
| <b>C(zipcode)[T.98033]</b> | 3.812e+05        | 9588.995                   | 39.753      | 0.000           | 3.62e+05      | 4e+05         |
| <b>C(zipcode)[T.98034]</b> | 1.966e+05        | 8827.053                   | 22.277      | 0.000           | 1.79e+05      | 2.14e+05      |
| <b>C(zipcode)[T.98038]</b> | 3.113e+04        | 8686.068                   | 3.584       | 0.000           | 1.41e+04      | 4.82e+04      |
| <b>C(zipcode)[T.98039]</b> | 7.727e+05        | 4.09e+04                   | 18.900      | 0.000           | 6.93e+05      | 8.53e+05      |
| <b>C(zipcode)[T.98040]</b> | 4.959e+05        | 1.37e+04                   | 36.144      | 0.000           | 4.69e+05      | 5.23e+05      |
| <b>C(zipcode)[T.98042]</b> | 2.313e+04        | 9058.715                   | 2.553       | 0.011           | 5372.175      | 4.09e+04      |

|                     |            |          |        |       |           |           |
|---------------------|------------|----------|--------|-------|-----------|-----------|
| C(zipcode)[T.98045] | 1.039e+05  | 1.47e+04 | 7.090  | 0.000 | 7.52e+04  | 1.33e+05  |
| C(zipcode)[T.98052] | 2.617e+05  | 8974.764 | 29.158 | 0.000 | 2.44e+05  | 2.79e+05  |
| C(zipcode)[T.98053] | 2.593e+05  | 1.02e+04 | 25.526 | 0.000 | 2.39e+05  | 2.79e+05  |
| C(zipcode)[T.98055] | 6.24e+04   | 1.01e+04 | 6.187  | 0.000 | 4.26e+04  | 8.22e+04  |
| C(zipcode)[T.98056] | 1.284e+05  | 9293.634 | 13.814 | 0.000 | 1.1e+05   | 1.47e+05  |
| C(zipcode)[T.98058] | 4.18e+04   | 9278.424 | 4.506  | 0.000 | 2.36e+04  | 6e+04     |
| C(zipcode)[T.98059] | 9.91e+04   | 9680.914 | 10.237 | 0.000 | 8.01e+04  | 1.18e+05  |
| C(zipcode)[T.98065] | 1.366e+05  | 9873.457 | 13.839 | 0.000 | 1.17e+05  | 1.56e+05  |
| C(zipcode)[T.98070] | 2.178e+05  | 3.83e+04 | 5.684  | 0.000 | 1.43e+05  | 2.93e+05  |
| C(zipcode)[T.98072] | 1.553e+05  | 1.32e+04 | 11.779 | 0.000 | 1.29e+05  | 1.81e+05  |
| C(zipcode)[T.98074] | 2.289e+05  | 1.03e+04 | 22.321 | 0.000 | 2.09e+05  | 2.49e+05  |
| C(zipcode)[T.98075] | 2.562e+05  | 1.14e+04 | 22.428 | 0.000 | 2.34e+05  | 2.79e+05  |
| C(zipcode)[T.98077] | 1.958e+05  | 2.94e+04 | 6.670  | 0.000 | 1.38e+05  | 2.53e+05  |
| C(zipcode)[T.98092] | -2.162e+04 | 1.01e+04 | -2.141 | 0.032 | -4.14e+04 | -1823.879 |
| C(zipcode)[T.98102] | 4.568e+05  | 1.32e+04 | 34.676 | 0.000 | 4.31e+05  | 4.83e+05  |
| C(zipcode)[T.98103] | 3.563e+05  | 8381.081 | 42.511 | 0.000 | 3.4e+05   | 3.73e+05  |
| C(zipcode)[T.98105] | 4.529e+05  | 1.04e+04 | 43.502 | 0.000 | 4.33e+05  | 4.73e+05  |
| C(zipcode)[T.98106] | 1.529e+05  | 9394.093 | 16.273 | 0.000 | 1.34e+05  | 1.71e+05  |
| C(zipcode)[T.98107] | 3.46e+05   | 9727.831 | 35.569 | 0.000 | 3.27e+05  | 3.65e+05  |
| C(zipcode)[T.98108] | 1.432e+05  | 1.06e+04 | 13.448 | 0.000 | 1.22e+05  | 1.64e+05  |
| C(zipcode)[T.98109] | 4.861e+05  | 1.29e+04 | 37.791 | 0.000 | 4.61e+05  | 5.11e+05  |
| C(zipcode)[T.98112] | 5.221e+05  | 1.02e+04 | 51.053 | 0.000 | 5.02e+05  | 5.42e+05  |
| C(zipcode)[T.98115] | 3.574e+05  | 8433.391 | 42.374 | 0.000 | 3.41e+05  | 3.74e+05  |
| C(zipcode)[T.98116] | 3.401e+05  | 9326.051 | 36.469 | 0.000 | 3.22e+05  | 3.58e+05  |
| C(zipcode)[T.98117] | 3.505e+05  | 8492.311 | 41.278 | 0.000 | 3.34e+05  | 3.67e+05  |
| C(zipcode)[T.98118] | 2.031e+05  | 8627.299 | 23.542 | 0.000 | 1.86e+05  | 2.2e+05   |
| C(zipcode)[T.98119] | 4.721e+05  | 1.09e+04 | 43.368 | 0.000 | 4.51e+05  | 4.93e+05  |
| C(zipcode)[T.98122] | 3.463e+05  | 9578.762 | 36.153 | 0.000 | 3.28e+05  | 3.65e+05  |
| C(zipcode)[T.98125] | 2.214e+05  | 9086.817 | 24.363 | 0.000 | 2.04e+05  | 2.39e+05  |
| C(zipcode)[T.98126] | 2.429e+05  | 9151.685 | 26.537 | 0.000 | 2.25e+05  | 2.61e+05  |
| C(zipcode)[T.98133] | 1.795e+05  | 8766.169 | 20.479 | 0.000 | 1.62e+05  | 1.97e+05  |
| C(zipcode)[T.98136] | 2.958e+05  | 9832.800 | 30.085 | 0.000 | 2.77e+05  | 3.15e+05  |
| C(zipcode)[T.98144] | 2.885e+05  | 9244.885 | 31.202 | 0.000 | 2.7e+05   | 3.07e+05  |
| C(zipcode)[T.98146] | 1.523e+05  | 1.01e+04 | 15.115 | 0.000 | 1.33e+05  | 1.72e+05  |
| C(zipcode)[T.98148] | 8.49e+04   | 1.71e+04 | 4.969  | 0.000 | 5.14e+04  | 1.18e+05  |
| C(zipcode)[T.98155] | 1.762e+05  | 9384.587 | 18.779 | 0.000 | 1.58e+05  | 1.95e+05  |
| C(zipcode)[T.98166] | 1.338e+05  | 1.19e+04 | 11.222 | 0.000 | 1.1e+05   | 1.57e+05  |
| C(zipcode)[T.98168] | 9.239e+04  | 1.07e+04 | 8.611  | 0.000 | 7.14e+04  | 1.13e+05  |
| C(zipcode)[T.98177] | 2.499e+05  | 1.07e+04 | 23.277 | 0.000 | 2.29e+05  | 2.71e+05  |
| C(zipcode)[T.98178] | 9.114e+04  | 1.02e+04 | 8.949  | 0.000 | 7.12e+04  | 1.11e+05  |

```
C(zipcode)[T.98188] 6.384e+04 1.29e+04 4.951 0.000 3.86e+04 8.91e+04
C(zipcode)[T.98198] 6.585e+04 1.03e+04 6.382 0.000 4.56e+04 8.61e+04
C(zipcode)[T.98199] 4.074e+05 9532.378 42.741 0.000 3.89e+05 4.26e+05
    grade_sc 4.668e+04 1267.898 36.818 0.000 4.42e+04 4.92e+04
    sqft_living_sc 7.548e+04 1357.667 55.597 0.000 7.28e+04 7.81e+04
    sqft_living15_sc 2.532e+04 1362.295 18.584 0.000 2.26e+04 2.8e+04
```

|                       |          |                          |           |
|-----------------------|----------|--------------------------|-----------|
| <b>Omnibus:</b>       | 4323.067 | <b>Durbin-Watson:</b>    | 2.000     |
| <b>Prob(Omnibus):</b> | 0.000    | <b>Jarque-Bera (JB):</b> | 22074.428 |
| <b>Skew:</b>          | 1.270    | <b>Prob(JB):</b>         | 0.00      |
| <b>Kurtosis:</b>      | 8.301    | <b>Cond. No.</b>         | 108.      |

#### Notes:

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

## Validate Model using the Train-Test-Split

In [108]:

```
from sklearn.model_selection import train_test_split
X = df.drop('price', axis=1)
y = df['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

In [109]:

```
df_train = pd.DataFrame(X_train, columns = X.columns)
df_train['price'] = y_train
df_test = pd.DataFrame(X_test, columns = X.columns)
df_test['price'] = y_test
```

In [110]:

```
from sklearn.metrics import r2_score
y_hat_test = model.predict(df_test)
r2_score(y_test, y_hat_test)
```

Out[110]:

0.768252174841723

Additional clean up of the data to figure out the dataset timeline in years and average price per zipcodes

In [111]:

```
unique_years = df['yr_built'].unique()
print(sorted(unique_years))
```

```
[1900, 1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1909, 1910, 1911, 1
912, 1913, 1914, 1915, 1916, 1917, 1918, 1919, 1920, 1921, 1922, 1923, 192
4, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933, 1934, 1935, 1936,
1937, 1938, 1939, 1940, 1941, 1942, 1943, 1944, 1945, 1946, 1947, 1948, 19
49, 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 196
1, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973,
1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 19
86, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 199
8, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,
2011, 2012, 2013, 2014, 2015]
```

In [112]:

```
def yr_builtinbin(year):
    if year < 1950:
        return 'Before 1900'
    elif year < 1980:
        return '1950-1979'
    elif year < 2000:
        return '1980-1999'
    else:
        return '2000+'
```

```
df['year_bins'] = df['yr_built'].apply(yr_builtinbin)
```

In [113]:

df

Out[113]:

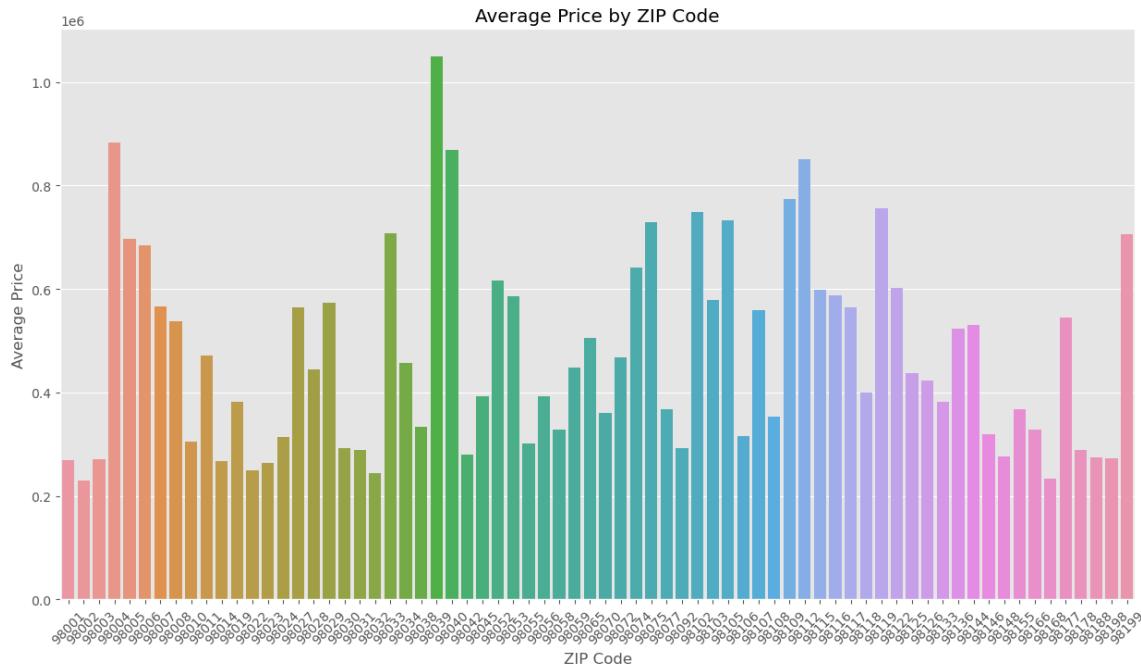
|       | price  | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition |
|-------|--------|----------|-----------|-------------|----------|--------|------------|------|-----------|
| 0     | 221900 | 3        | 1.00      | 7.073270    | 8.639411 | 1.0    | 0.0        | 0.0  |           |
| 1     | 538000 | 3        | 2.25      | 7.851661    | 8.887653 | 2.0    | 0.0        | 0.0  |           |
| 2     | 180000 | 2        | 1.00      | 6.646391    | 9.210340 | 1.0    | 0.0        | 0.0  |           |
| 3     | 604000 | 4        | 3.00      | 7.580700    | 8.517193 | 1.0    | 0.0        | 0.0  |           |
| 4     | 510000 | 3        | 2.00      | 7.426549    | 8.997147 | 1.0    | 0.0        | 0.0  |           |
| ...   | ...    | ...      | ...       | ...         | ...      | ...    | ...        | ...  | ...       |
| 21592 | 360000 | 3        | 2.50      | 7.333023    | 7.030857 | 3.0    | 0.0        | 0.0  |           |
| 21593 | 400000 | 4        | 2.50      | 7.745003    | 8.667852 | 2.0    | 0.0        | 0.0  |           |
| 21594 | 402101 | 2        | 0.75      | 6.927558    | 7.207860 | 2.0    | 0.0        | 0.0  |           |
| 21595 | 400000 | 3        | 2.50      | 7.377759    | 7.778211 | 2.0    | 0.0        | 0.0  |           |
| 21596 | 325000 | 2        | 0.75      | 6.927558    | 6.981006 | 2.0    | 0.0        | 0.0  |           |

15334 rows × 21 columns



In [114]:

```
plt.figure(figsize=(15, 8))
sns.barplot(x='zipcode', y='price', data=df, errorbar=None)
plt.xticks(rotation=45)
plt.xlabel('ZIP Code')
plt.ylabel('Average Price')
plt.title('Average Price by ZIP Code')
plt.show()
```

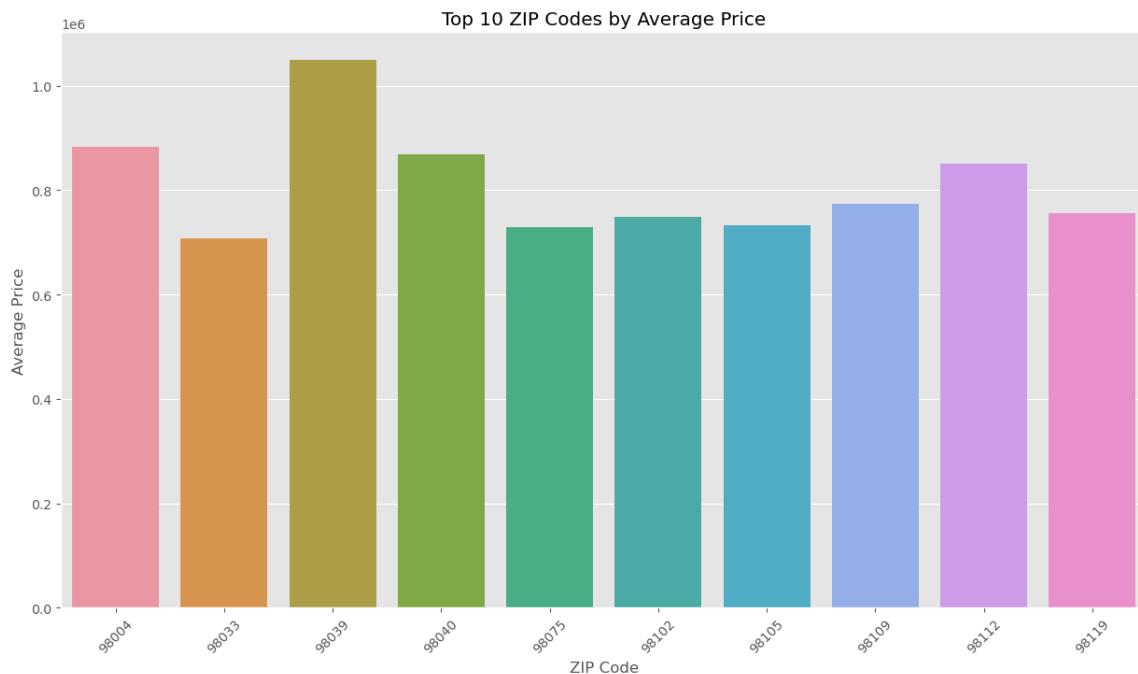


In [115]:

```
# Calculate the average price for each ZIP code
avg_price_per_zip = df.groupby('zipcode')['price'].mean().reset_index()

# Sort ZIP codes by average price in descending order and take top 10
top_10_zip = avg_price_per_zip.sort_values('price', ascending=False).head(10)

# Plotting
plt.figure(figsize=(15, 8))
sns.barplot(x='zipcode', y='price', data=top_10_zip, errorbar=None)
plt.xticks(rotation=45)
plt.xlabel('ZIP Code')
plt.ylabel('Average Price')
plt.title('Top 10 ZIP Codes by Average Price')
plt.show()
```



## Analysis Summary:

Upon conducting multiple regression analyses, the fifth model demonstrated a robust R-squared value of 0.775, signifying that approximately 78% of the variability in home prices within King County can be explained by the predictors in the model. Notably, the postal code emerged as one of the most significant determinants influencing property valuation. Additional variables that were positively correlated with property prices include:

- Living Space Square Footage
- Building Grade, as per the King County Grading System
- Average Interior Square Footage of the Nearest 15 Residences

## Strategic Recommendations for a Pro Home Builder Company:

To optimize the profitability and marketability of residential properties in King County, the following strategies are recommended:

- Strategic Location Selection: Prioritize property acquisition in high-value zip codes to benefit from the area's intrinsically higher property valuations.
- Optimize Living Space: Expand the square footage of the living area to align with the positive correlation observed between living space and property value.
- Quality of Construction: Employ premium-quality building materials and construction techniques to secure the highest possible building grade according to the King County Grading System. This will not only enhance the property's intrinsic value but also its appeal to potential buyers.
- Neighborhood Composition: Opt for land parcels where the surrounding 15 residences have a higher average interior square footage. This is indicative of a neighborhood's overall property value and can positively influence the value of the new development.

**By implementing these targeted strategies, housing development firms can substantially increase**