1 Final Project Submission: Real Estate Linear Regression Model Analysis

(Phase 2)

- Student Name: Tenicka Norwood
- Program Pace: self paced
- Scheduled Project Review Time:
- Instructor name: Joe Comeaux
- Blog post Url:https://medium.com/@tenicka.norwood/working-with-data-while-trying-to-stay-dry-fa4ebf5e5f64 (https://medium.com/@tenicka.norwood/working-with-data-while-trying-to-stay-dry-fa4ebf5e5f64)

Photo by: anyaberkut on Canva (https://www.canva.com)

2 Overview:

LandingPad Realtors is a real estate business that helps families with school-aged children relocate to King County and find the perfect home to meet their families needs. LandingPad provides potential homeowners with home purchase options within their ideal budget.

For this project, I will start by identifying the characteristics of homes that increase housing costs. The effect of each relevant feature will then be identified and communicated to the team at LandingPad. This project will be grounded in performing a statistical analysis of the price of houses in the King County House dataset and creating a multiple linear regression model that accurately predicts the sale price of a house in King County.

Linear Regression is often described as a predictive model based on the sum of weighted independent variables. This mathematical linear relationship between an dependent variable and one or more dependent varibles and is often shown as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots \beta_n x_n$$

where β_0 is a constant that shows the y-intercept and β_1 to β_n are coefficients that show how the dependent variable y changes with the independent variables x_1 to x_n.

3 Business Understanding

- **Stakeholder**: LandingPad Realtors
- **Busines Case**: I have been hired by LandingPad to accurately predict the housing prices within the King County Housing Market. Executives at LandingPad want to launch a multimedia campaign to reach their target audience of young families moving to the Kings County Area and want a reliable model that can be refined over time as more information becomes available.

4 Objectives

We will use the CRISP DM model to:

- Understand the Data
  - Examine and document surface properties of the data
  - Dig deeper into the data to visualize and identify relationships among the data
- Prepare the Data

    - Select, clean, construct, integrate and format data
- Model the Data
    - Determine which algorithm to try
- Evaluate the Model
    - Determine if the model meets the business success criteria and determine next steps

Then, I will use this to build and refine a **linear regression model** that I can use to answer the following guiding questions:

1. **Which neighborhoods have the highest average home price?**
   Understanding the effect of neighborhood location on home price is key information for potential home owners and realtors
2. **How does the number of bedrooms affect the sale price of a home?**
   Insights on the affect of attributes on the sale price can help new home owners budget appropriately
3. **How does proximity to a highly rated school affect the sale price of a home?**
   Knowing which homes are connected to highly rated schools is vital information for families with school aged children.

The recommendations garnered by answering these questions will be valuable to LandingPad Realtors because they will help prospective home buyers confidently determine which homes yield the best options within their price range.

5 Data Understanding

In this project I will use the CRISP DM method. The dataset selected in this project are from the :

- King County House Sales Dataset found in `kc_house_data.csv`

The dataset can be found in the data folder of this repository along with a file called `column_names.md` which provides description of the features within the dataset. More information about the features on the site of [the King County Assessor. (https://info.kingcounty.gov/assessor/esales/Glossary.aspx?type=r)](https://info.kingcounty.gov/assessor/esales/Glossary.aspx?type=r)

The King County House Sales Dataset includes sales data for 21,597 homes with 20 features:

Name	Description	Final Datatype	Numeric or Categorical	Target or Feature
`id`	Unique identifier for a house	`int`	Numeric	Feature
`date`	Date house was sold	`datetime`	Numeric	Feature
`price`	Sale price (**prediction target**)	`int`	Numeric	Target
`bedrooms`	Number of bedrooms	`int`	Numeric	Feature
`bathrooms`	Number of bathrooms	`float`	Numeric	Feature
`sqft_living`	Square footage of living space in the home	`int`	Numeric	Feature
`sqft_lot`	Square footage of the lot	`int`	Numeric	Feature
`floors`	Number of floors(levels) in house	`float`	Numeric	Feature
`waterfront`	Whether the house is on a waterfront	`float`	Categorical	Feature
`view`	Quality of view from house	`float`	Categorical	Feature
`condition`	How good the overall condition of the house is. Related to the maintenance of house	`int`	Numeric	Feature
`grade`	Overall grade of the house. Related to the construction and design of the house	`int`	Categorical	Feature
`sqft_above`	Square footage of house apart from basement	`int`	Numeric	Feature
`sqft_basement`	Square footage of the basement	`float`	Numeric	Feature
`yr_built`	Year when house was built	`int`	Numeric	Feature
`yr_renovated`	Year when house was renovated	`int`	Numeric	Feature
`zipcode`	ZIP Code used by the United States Postal Service	`int`	Categorical	Feature
`lat`	Latitude coordinate	`float`	Numeric	Feature
`long`	Longitude coordinate	`float`	Numeric	Feature
`sqft_living15`	The square footage of interior housing living space for the nearest 15 neighbors	`int`	Numeric	Feature
`sqft_lot15`	The square footage of the land lots of the nearest 15 neighbors	`int`	Numeric	Feature

5.1 Data Preparation

5.2 Import libraries and Visualization Packages

Importing libraries at the beginning allows access to modules and other tools throughout this project that help to make the tasks within this project manageable to implement. The main libraries that will be used within this project include:

- `pandas` : a data analysis and manipulation library which allows for flexible reading, writing, and reshaping of data
- `numpy` : a key library that brings the computationaly power of languages like C to Python
- `matplotlib` : a comprehensive visualization library
- `seaborn` : a data visualization library based on matplotlib

In [1]:

```
# Import libraries and visualization packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import statsmodels.api as sm
from selenium import webdriver
from selenium.common.exceptions import WebDriverException
import warnings
warnings.simplefilter(action ='ignore', category = DeprecationWarning)

# Allow plots to display and be stored inline within a notebook
%matplotlib inline

# Used for working with the z-score
from scipy import stats

# Used for working with long url
from urllib.parse import urlencode

# Set display option to readable format
pd.set_option('display.float_format', lambda x: '%.2f' % x)
```

5.3 Load Data Using Pandas

Read in data from `kc_house_data.csv` using `.read_csv()` from the pandas library.

In [2]:

```
# Read Data from Kings County House Data
df = pd.read_csv('data/kc_house_data.csv', index_col = 0)
```

Let's look at the first five rows of the Kings County Housing Data.

In [3]:

```
df.head()
```

Out[3]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	y
7129300520	10/13/2014	221900.00	3	1.00	1180	5650	1.00	NaN	NONE	Average	7 Average	1180	0.0	
6414100192	12/9/2014	538000.00	3	2.25	2570	7242	2.00	NO	NONE	Average	7 Average	2170	400.0	
5631500400	2/25/2015	180000.00	2	1.00	770	10000	1.00	NO	NONE	Average	6 Low Average	770	0.0	
2487200875	12/9/2014	604000.00	4	3.00	1960	5000	1.00	NO	NONE	Very Good	7 Average	1050	910.0	
1954400510	2/18/2015	510000.00	3	2.00	1680	8080	1.00	NO	NONE	Average	8 Good	1680	0.0	

5.3.1 Clean the Data

In order to clean the data, I typically address missing data, place holders and datatypes. This is the most important step of this project because if data is not appropriate for the model, the results will be inherently inaccuarate and my model will result in lackluster predictions.

To dig deeper into the data, I will:

- Review the datatypes found within the entire dataframe
- Address duplicates, missing and placeholder data
- Address incorrect or incongruous datatypes for the model

5.3.2 Review Datatypes within Data Frame

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21597 entries, 7129300520 to 1523300157
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           21597 non-null  object
 1   price          21597 non-null  float64
 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  object
 8   view           21534 non-null  object
 9   condition      21597 non-null  object
 10  grade          21597 non-null  object
 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(6), int64(8), object(6)
memory usage: 3.5+ MB
```

5.3.3 Address duplicates, missing and placeholder data

In [5]:
```
# Check for placeholders throughout the entire dataframe
df.isin(['?', '#', 'NaN', 'null', 'N/A', '-']).any()
```

Out[5]:
```
date             False
price            False
bedrooms         False
bathrooms        False
sqft_living      False
sqft_lot         False
floors           False
waterfront       False
view             False
condition        False
grade            False
sqft_above       False
sqft_basement     True
yr_built         False
yr_renovated     False
zipcode          False
lat              False
long             False
sqft_living15    False
sqft_lot15       False
dtype: bool
```

In [6]:
```
# Convert sqft_basement to float
# Replace placeholder ? to NAN values
df['sqft_basement'] = pd.to_numeric(df['sqft_basement'], errors="coerce")
```

In [7]:

```
# Check for placeholders throughout the entire dataframe
df.isin(['?', '#', 'NaN', 'null', 'N/A', '-']).any()
```

Out[7]:

```
date             False
price            False
bedrooms         False
bathrooms        False
sqft_living      False
sqft_lot         False
floors           False
waterfront       False
view             False
condition        False
grade            False
sqft_above       False
sqft_basement    False
yr_built         False
yr_renovated     False
zipcode          False
lat              False
long             False
sqft_living15    False
sqft_lot15       False
dtype: bool
```

Let's look at the columns within the `pandas` dataframe.

In [8]:

```
df.columns
```

Out[8]:

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

This dataset has 20 columns with 21597 rows of data. The waterfront, view, condition, grade, sqft_basement and date are object datatypes.

Let's check which columns contain null values.

In [9]:

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

Out[9]:

```
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     454
yr_built            0
yr_renovated     3842
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
dtype: int64
```

5.3.4 Address incorrect or incongruous datatypes for the model

Let's convert the date from a string to a datetime object.

In [10]:

```
# Convert data column in the dataframe to datetime
df['date'] =  pd.to_datetime(df['date'], format='%m/%d/%Y')
```

In [11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21597 entries, 7129300520 to 1523300157
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           21597 non-null  datetime64[ns]
 1   price          21597 non-null  float64
 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  object
 8   view           21534 non-null  object
 9   condition      21597 non-null  object
 10  grade          21597 non-null  object
 11  sqft_above     21597 non-null  int64
 12  sqft_basement  21143 non-null  float64
 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: datetime64[ns](1), float64(7), int64(8), object(4)
memory usage: 3.5+ MB
```

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

Out[12]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_bui
id														
7129300520	2014-10-13	221900.00	3	1.00	1180	5650	1.00	NaN	NONE	Average	7 Average	1180	0.00	195
6414100192	2014-12-09	538000.00	3	2.25	2570	7242	2.00	NO	NONE	Average	7 Average	2170	400.00	195
5631500400	2015-02-25	180000.00	2	1.00	770	10000	1.00	NO	NONE	Average	6 Low Average	770	0.00	193
2487200875	2014-12-09	604000.00	4	3.00	1960	5000	1.00	NO	NONE	Very Good	7 Average	1050	910.00	196
1954400510	2015-02-18	510000.00	3	2.00	1680	8080	1.00	NO	NONE	Average	8 Good	1680	0.00	198

From our check, the waterfront column has 2376 null values. The view column has 63 null values and the yr_renovated column has 3842 null values.

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

Out[13]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	lo
count	21597.00	21597.00	21597.00	21597.00	21597.00	21597.00	21597.00	21143.00	21597.00	17755.00	21597.00	21597.00	21597.
mean	540296.57	3.37	2.12	2080.32	15099.41	1.49	1788.60	291.85	1971.00	83.64	98077.95	47.56	-122.
std	367368.14	0.93	0.77	918.11	41412.64	0.54	827.76	442.50	29.38	399.95	53.51	0.14	0.
min	78000.00	1.00	0.50	370.00	520.00	1.00	370.00	0.00	1900.00	0.00	98001.00	47.16	-122.
25%	322000.00	3.00	1.75	1430.00	5040.00	1.00	1190.00	0.00	1951.00	0.00	98033.00	47.47	-122.
50%	450000.00	3.00	2.25	1910.00	7618.00	1.50	1560.00	0.00	1975.00	0.00	98065.00	47.57	-122.
75%	645000.00	4.00	2.50	2550.00	10685.00	2.00	2210.00	560.00	1997.00	0.00	98118.00	47.68	-122.
max	7700000.00	33.00	8.00	13540.00	1651359.00	3.50	9410.00	4820.00	2015.00	2015.00	98199.00	47.78	-121.

In [14]:
```
df['view'].value_counts()
```

Out[14]:
```
NONE         19422
AVERAGE        957
GOOD           508
FAIR           330
EXCELLENT      317
Name: view, dtype: int64
```

In [15]:
```
#for col in categoricals:
#    print(df[col].value_counts(), "\n")
```

In [16]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21597 entries, 7129300520 to 1523300157
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           21597 non-null  datetime64[ns]
 1   price          21597 non-null  float64
 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  object
 8   view           21534 non-null  object
 9   condition      21597 non-null  object
 10  grade          21597 non-null  object
 11  sqft_above     21597 non-null  int64
 12  sqft_basement  21143 non-null  float64
 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: datetime64[ns](1), float64(7), int64(8), object(4)
memory usage: 3.5+ MB
```

Next I will impute missing values in the dataframe. I replaced `waterfront` , `sqft_basement` and `view` null values with 0 and replace `yr_renovated` null values with the `yr_built` .

In [17]:
```
df["waterfront"].value_counts()
```

Out[17]:
```
NO     19075
YES      146
Name: waterfront, dtype: int64
```

In [18]:
```
df["view"].value_counts()
```

Out[18]:
```
NONE         19422
AVERAGE        957
GOOD           508
FAIR           330
EXCELLENT      317
Name: view, dtype: int64
```

In [19]:
```
df["zipcode"].value_counts()
```

Out[19]:
```
98103    602
98038    589
98115    583
98052    574
98117    553
        ...
98102    104
98010    100
98024     80
98148     57
98039     50
Name: zipcode, Length: 70, dtype: int64
```

In [20]:
```
df["waterfront"].fillna(0, inplace = True)
df["sqft_basement"].fillna(0, inplace = True)
df["view"].fillna(0, inplace = True)
df['yr_renovated'].fillna(df['yr_built'], inplace=True)
```

In [21]:
```
for col in df.columns:
    print('{} : {}'.format(col, df[col].nunique()))
```

```
date : 372
price : 3622
bedrooms : 12
bathrooms : 29
sqft_living : 1034
sqft_lot : 9776
floors : 6
waterfront : 3
view : 6
condition : 5
grade : 11
sqft_above : 942
sqft_basement : 303
yr_built : 116
yr_renovated : 117
zipcode : 70
lat : 5033
long : 751
sqft_living15 : 777
sqft_lot15 : 8682
```

Now let's check to see if there are any null values in any of the columns of our dataframe.

In [22]:
```
df.isnull().sum()
```

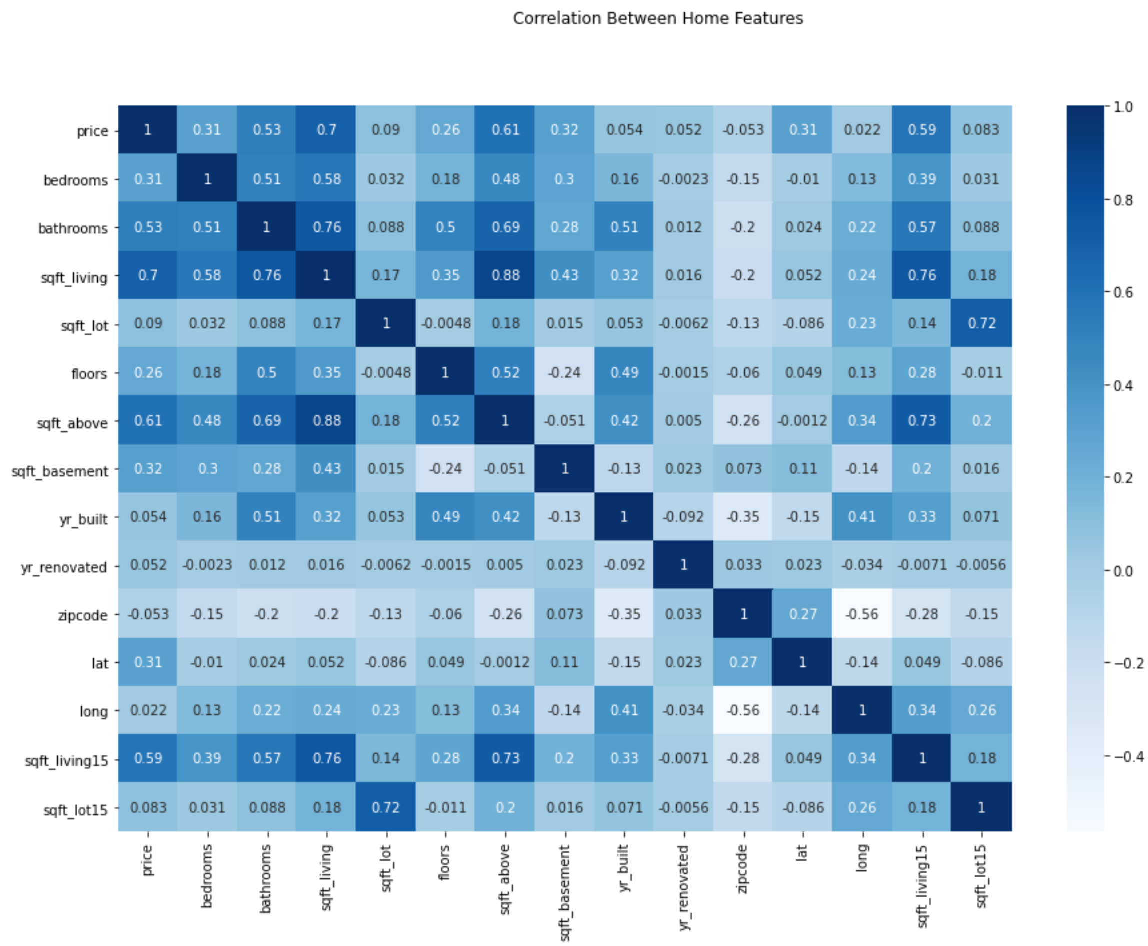
Out[22]:
```
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

In [23]:
```
df['yr_renovated'] = df['yr_renovated'].astype('int64')
```

Great, now let's check out the correlation of features within my dataframe.

In [24]:

```
# Review correlations on filtered dataset
plt.figure(figsize = (15,10)).suptitle("Correlation Between Home Features")
sns.heatmap(df.corr(), annot = True, cmap = "Blues");
```



The correlation shown is called a Pearson correlation and is given by the ratio below:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}}$$

A commonly held rule for interpreting the Pearson correlation is that:

- Very Strong Correlation → 0.8 to 1
- Strong Correlation → 0.6 to 0.799
- Moderate Correlation → 0.4 to 0.599
- Weak Correlation → 0.2 to 0.399
- Very Weak Correlation → 0 to 0.199

From the heatmap, it appears that house `price` (the target) has the **strongest correlation** with `sqft_living` (0.7), a weak correlation with floors `floors` (0.26),a **strong correlation** with `sqft_above` (0.61), a weak correlation with `lat` (0.31) and a moderate correlation with `sqft_living_15` (0.59). The `sqft_living` , `bathrooms` , `sqft_above` and `sqft_living_15` features all seem to be highly correlated (> 0.7) to one another as well. This will be helpful when building our baseline model and refining it after reviewing our metrics.

Looking at a heatmap this complex may be a stretch. Let's create a more elegant way to determine which pairs of features have strong correlation values.

In [25]:

```python
_df = df.corr().abs().stack().reset_index().sort_values(0, ascending = False)

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

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

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

_df.columns = ['cc']

_df.drop_duplicates(inplace = True)

_df[(_df.cc > 0.6) & (_df.cc < 1)]
```

Out[25]:

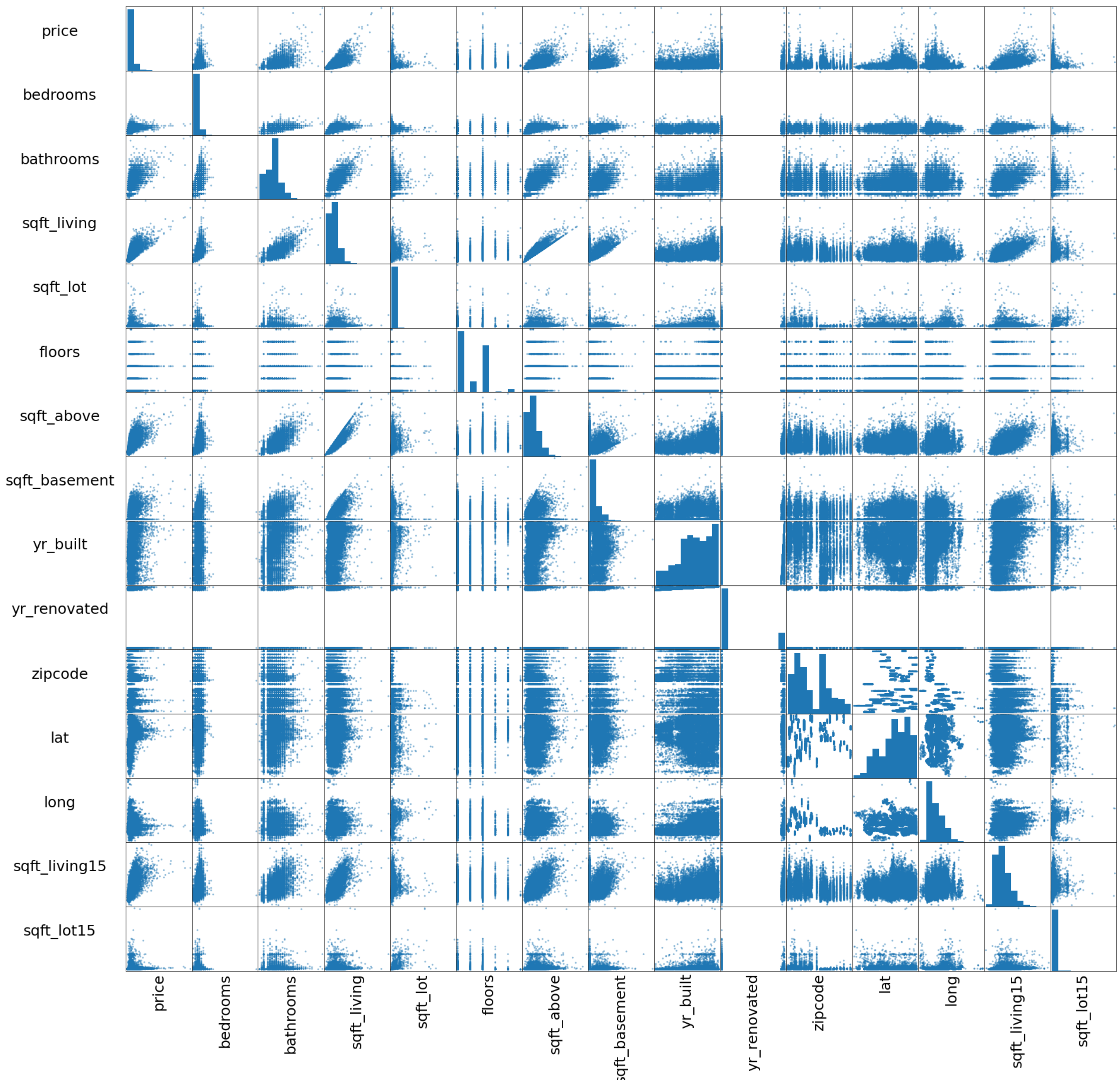
pairs	cc
(sqft_above, sqft_living)	0.88
(sqft_living, sqft_living15)	0.76
(bathrooms, sqft_living)	0.76
(sqft_above, sqft_living15)	0.73
(sqft_lot, sqft_lot15)	0.72
(price, sqft_living)	0.70
(bathrooms, sqft_above)	0.69
(sqft_above, price)	0.61

5.3.5 Scatter matrix

Create a scatter matrix for the King County House data. (This takes awhile (~25 s) to run and generate a plot)

In [26]:

```
%%time
# create scatter matrix
smat = pd.plotting.scatter_matrix(df, figsize=[30, 30]);


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

#Hide all ticks
[s.set_xticks(()) for s in smat.reshape(-1)]
[s.set_yticks(()) for s in smat.reshape(-1)]
[plt.setp(item.xaxis.get_label(), "size", 25) for item in smat.ravel()]
[plt.setp(item.yaxis.get_label(), "size", 25) for item in smat.ravel()]

plt.show()
```

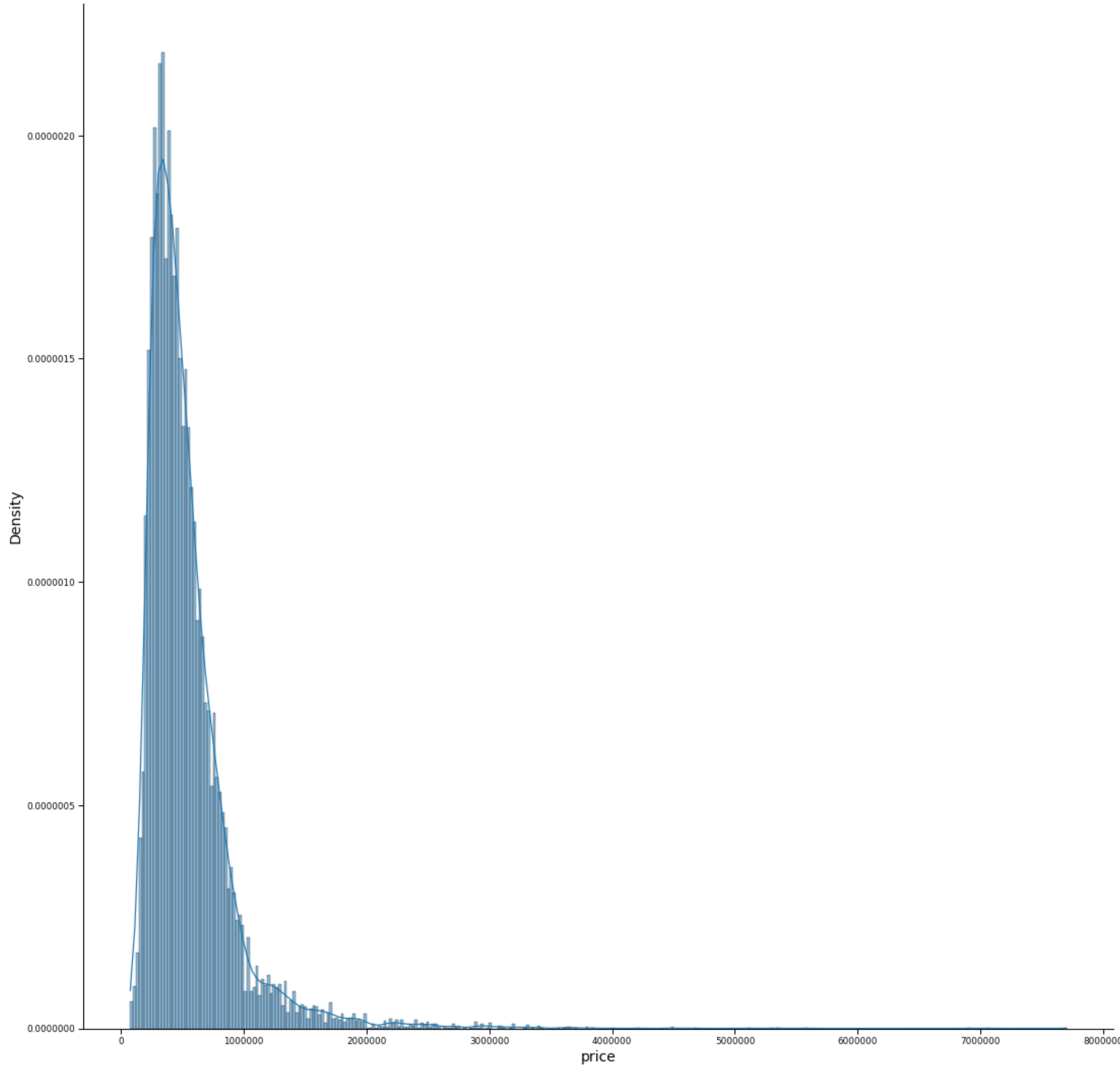



```
Wall time: 1min
```

Visualize target using `seaborn` figure level plot called `displot`.

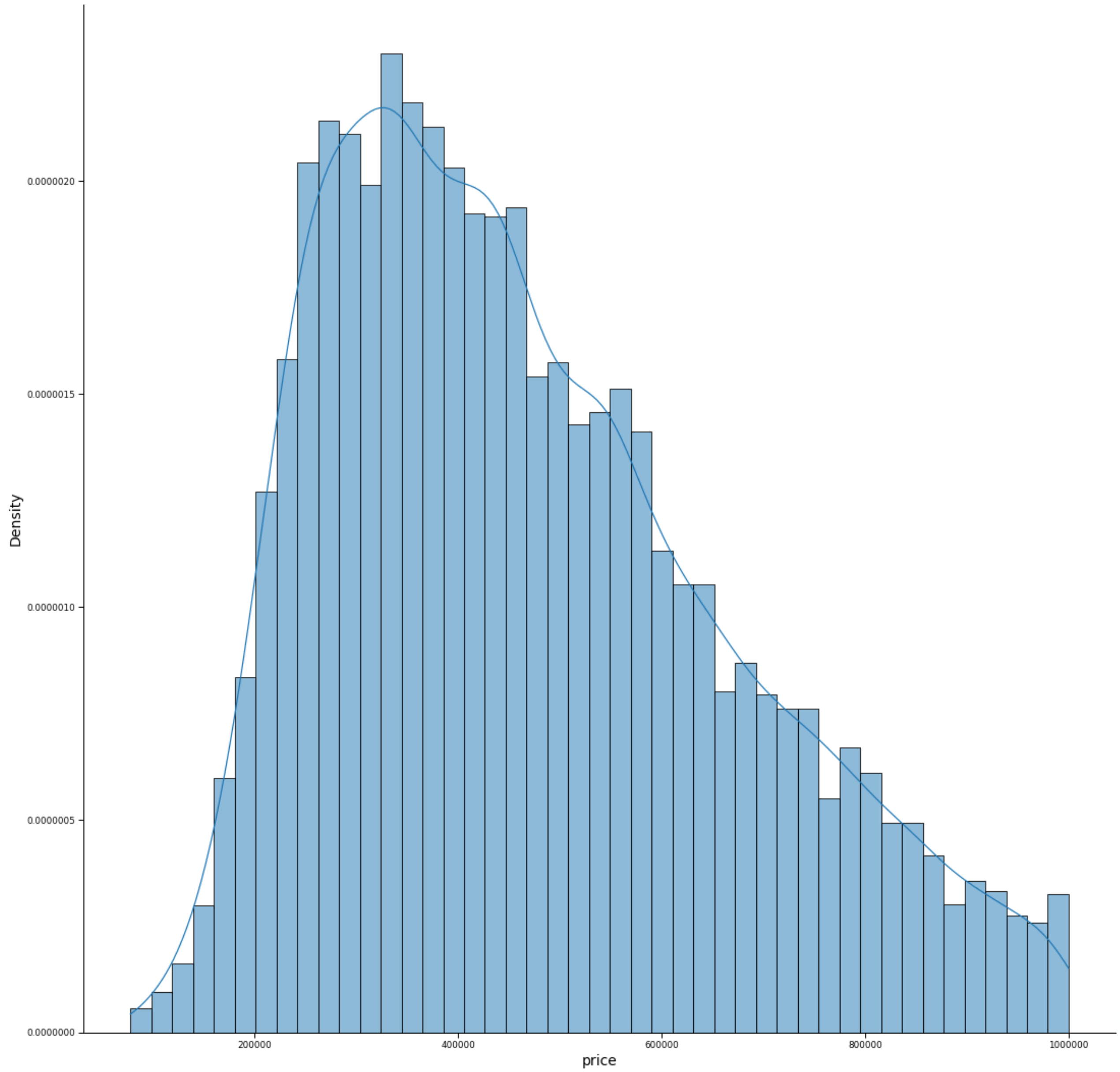
In [27]:

```
sns.set_context("paper", rc={"font.size":12,"axes.labelsize":14})
sns.displot(df['price'], stat = 'density', kde = True, height = 15)
plt.ticklabel_format(style='plain', axis= 'both')
plt.show();
```



The displot shows that most of the houses in the KC House data set are priced below 1 million dollars.

In [28]:
```python
# Rough filter investigating dataframe upper limit
new_df = df[df['price'] <= 1000000]
sns.set_context("paper", rc={"font.size":12,"axes.labelsize":14})
sns.displot(new_df['price'], stat = 'density', kde = True, height = 15)
plt.ticklabel_format(style='plain', axis= 'both')
plt.show();
```



In [29]:
```python
# Identify the name of the predictor column with strongest correlation
most_correlated = 'sqft_living'
```

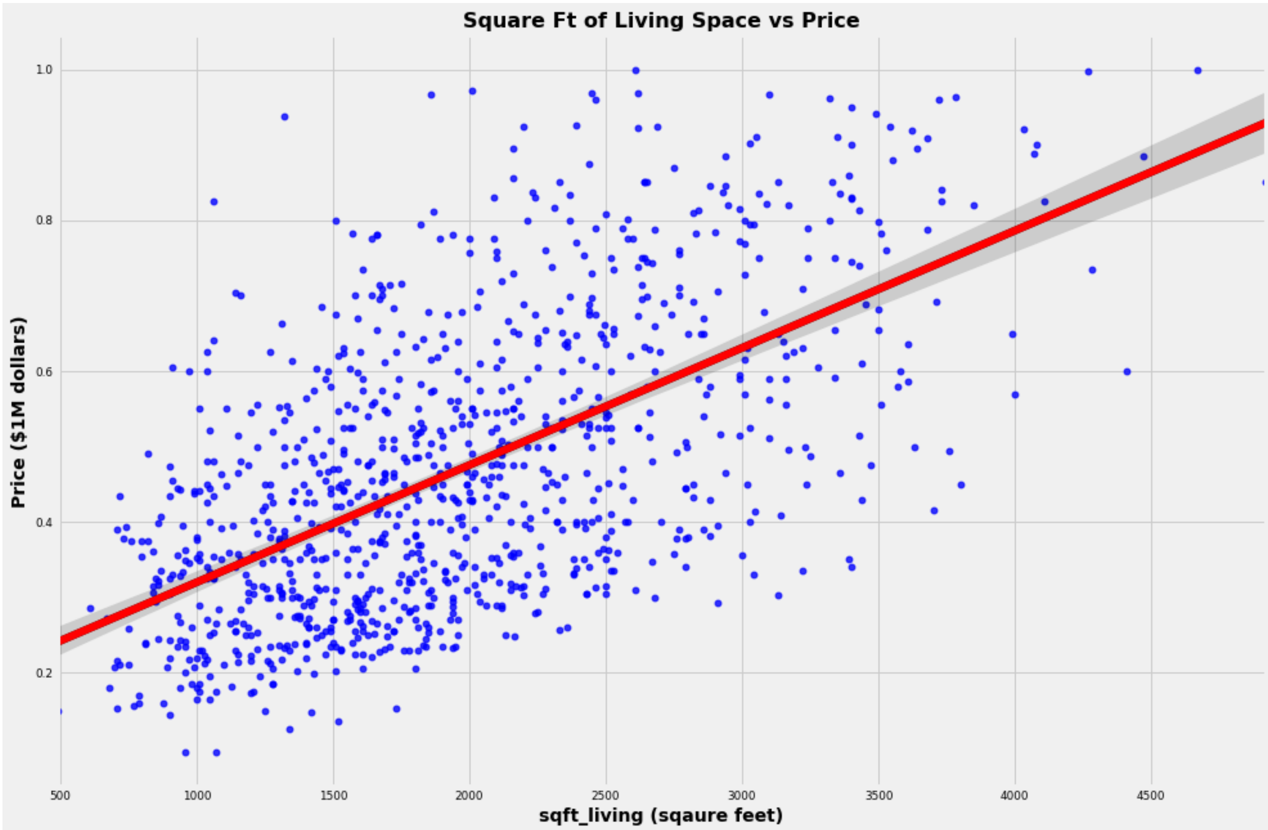
From reviewing dataframe df with correlation coefficient values between 0.6 and 1, we can drop the following features from our original dataframe (`df` to create a new dataframe called `df_subset` :

- `sqft_above`
- `sqft_living15`
- `bathrooms`
- `sqft_lot`
- `sqft_lot15`

In [30]:
```python
df_subset = new_df.drop(["sqft_above", "sqft_living15", "bathrooms"
                                   , "sqft_lot", "sqft_lot15"], axis = 1)
```

In [31]:

```
plt.style.use('fivethirtyeight')
values = np.array(df_subset.sample(1000, random_state = 1).price)/1000000
# Set up plot
fig, ax = plt.subplots(figsize = (15, 10))
sns.regplot(data = df_subset.sample(1000, random_state = 1), x = most_correlated, y = values
                 , scatter = True, ci = 95, fit_reg = True,scatter_kws={"color": "blue"}, line_kws={"color": "black"} );
sns.regplot(data = df_subset.sample(1000, random_state = 1), x = most_correlated, y = values, scatter = False
                 , ci = 0, fit_reg = True, color = 'red');
ax.set_title("Square Ft of Living Space vs Price", weight = 'bold').set_fontsize('16');
ax.set_xlabel("sqft_living (sqaure feet)", fontsize = '14', weight = 'bold');
ax.set_ylabel("Price ($1M dollars)" , fontsize = '14', weight = 'bold');
```



6 Build a Simple Linear Regression model

First, I will set the dependent variable (`y`) to be the `price` . Next I will choose the most highly correlated features from the dataframe to be the baseline independent variable (`X`). Afterwards, I will:

- Build a linear regression using `statsModels`
- Describe the overall model performance
- Interpret its coefficients.

In [32]:

```
# Explore correlation to find a good starting point
df_subset.corr()['price'].sort_values()
```

Out[32]:

```
zipcode          -0.02
yr_renovated      0.03
yr_built          0.06
long              0.08
sqft_basement     0.22
floors            0.27
bedrooms          0.29
lat               0.44
sqft_living       0.60
price             1.00
Name: price, dtype: float64
```

In [33]:

```
# Set price as the dependent variable
y = df_subset["price"]
```

In [34]:
```
# sqft_living had the highest correlation
X_baseline = df_subset[[most_correlated]]
```

6.1 Creating and Fitting Simple Linear Regression

In [35]:
```
baseline_model = sm.OLS(y, sm.add_constant(X_baseline))
baseline_results = baseline_model.fit()
```

7 Evaluate and Interpret Baseline Model Results

In [36]:
```
print(baseline_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.365
Model:                            OLS   Adj. R-squared:                  0.365
Method:                 Least Squares   F-statistic:                 1.158e+04
Date:                Sun, 22 Jan 2023   Prob (F-statistic):               0.00
Time:                        10:12:12   Log-Likelihood:            -2.6946e+05
No. Observations:               20139   AIC:                         5.389e+05
Df Residuals:                   20137   BIC:                         5.389e+05
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const        1.607e+05   3061.929     52.468      0.000    1.55e+05    1.67e+05
sqft_living   157.1496      1.461    107.596      0.000     154.287     160.012
==============================================================================
Omnibus:                      704.434   Durbin-Watson:                   1.961
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              780.694
Skew:                           0.482   Prob(JB):                    2.98e-170
Kurtosis:                       2.973   Cond. No.                     5.82e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.82e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

This simple linear regression model is statistically significant overall, and explains **49.3% of the variance in house price**. Both the intercept and the coefficient for `sqft_living` are statistically significant.

The intercept is a small negative number, meaning a home with 0 square feet of living would cost around $0.

The coefficient for `sqft_living` is about 280, which means that for each additional square foot of living space, I expect the price to increase about **$280**.

8 Prepare Categorical Features for Multiple Regression Modeling

The categorical features that need to be replaced with dummies are `grade` , `view` , `waterfront` , and `zipcode` . First, let's review how many values each of these features has.

In [37]:
```
new_df["grade"].value_counts()
```

Out[37]:
```
7 Average        8951
8 Good           5873
9 Better         2224
6 Low Average    2033
10 Very Good      694
5 Fair            242
11 Excellent       92
4 Low              27
12 Luxury           2
3 Poor              1
Name: grade, dtype: int64
```

In [38]:
```python
new_df["view"].value_counts()
```

Out[38]:
```
NONE         18611
AVERAGE        763
GOOD           319
FAIR           260
EXCELLENT      129
0               57
Name: view, dtype: int64
```

In [39]:
```python
new_df["waterfront"].value_counts()
```

Out[39]:
```
NO     17862
0       2227
YES       50
Name: waterfront, dtype: int64
```

In [40]:
```python
new_df["zipcode"].value_counts()
```

Out[40]:
```
98038    586
98103    581
98052    553
98115    549
98042    547
        ...
98109     82
98102     81
98024     72
98148     57
98039      6
Name: zipcode, Length: 70, dtype: int64
```

Next, lets make a new dataframe that includes these categoricals called `df_updated` .

In [41]:
```python
df_updated = new_df[['bedrooms','sqft_living', 'lat', 'long', "grade", "view", "zipcode", "waterfront"]]
df_updated.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20139 entries, 7129300520 to 1523300157
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   bedrooms     20139 non-null  int64  
 1   sqft_living  20139 non-null  int64  
 2   lat          20139 non-null  float64
 3   long         20139 non-null  float64
 4   grade        20139 non-null  object 
 5   view         20139 non-null  object 
 6   zipcode      20139 non-null  int64  
 7   waterfront   20139 non-null  object 
dtypes: float64(2), int64(3), object(3)
memory usage: 1.4+ MB
```

Now let's use **dummy variables** to make this dataframe usuable for a linnear regression model.

In [42]:
```python
# Create a new data frame with view dummy variables
df_with_dummies = pd.get_dummies(data = df_updated, columns = ["view", "zipcode", "grade", "waterfront"], drop_first = True)
```

In [43]:
```python
df_with_dummies.head()
```

Out[43]:

	bedrooms	sqft_living	lat	long	view_AVERAGE	view_EXCELLENT	view_FAIR	view_GOOD	view_NONE	zipcode_98002	...	grade_12 Luxury
id												
7129300520	3	1180	47.51	-122.26	0	0	0	0	1	0	...	0
6414100192	3	2570	47.72	-122.32	0	0	0	0	1	0	...	0
5631500400	2	770	47.74	-122.23	0	0	0	0	1	0	...	0
2487200875	4	1960	47.52	-122.39	0	0	0	0	1	0	...	0
1954400510	3	1680	47.62	-122.05	0	0	0	0	1	0	...	0

5 rows × 89 columns

In [44]:
```python
target = new_df["price"]
```

In [45]:
```python
target
```

Out[45]:
```
id
7129300520    221900.00
6414100192    538000.00
5631500400    180000.00
2487200875    604000.00
1954400510    510000.00
                 ...
263000018     360000.00
6600060120    400000.00
1523300141    402101.00
291310100     400000.00
1523300157    325000.00
Name: price, Length: 20139, dtype: float64
```

In [46]:
```python
iterated_model = sm.OLS(target, sm.add_constant(df_with_dummies))
iterated_results = iterated_model.fit()
```

In [47]:
```python
print(iterated_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.813
Model:                            OLS   Adj. R-squared:                  0.812
Method:                 Least Squares   F-statistic:                     978.0
Date:                Sun, 22 Jan 2023   Prob (F-statistic):               0.00
Time:                        10:12:13   Log-Likelihood:            -2.5717e+05
No. Observations:               20139   AIC:                         5.145e+05
Df Residuals:                   20049   BIC:                         5.152e+05
Df Model:                          89
Covariance Type:            nonrobust
=========================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
const                 -1.142e+07   3.34e+06     -3.419      0.001     -1.8e+07   -4.87e+06
bedrooms               -428.4762    860.000     -0.498      0.618    -2114.147    1257.195
sqft_living             109.9166      1.387     79.261      0.000      107.198     112.635
lat                    1.465e+05   3.47e+04      4.223      0.000     7.85e+04    2.14e+05
long                   -3.85e+04   2.47e+04     -1.560      0.119    -8.69e+04    9863.715
```

8.1 Remove the values with high p-values

In [48]:
```python
revised_df = df_with_dummies.drop(["view_AVERAGE", "view_FAIR", "zipcode_98002"
                                  , "zipcode_98003", "zipcode_98011", "zipcode_98014"
                                  ,"zipcode_98077","zipcode_98028","zipcode_98030"
                                  ,"zipcode_98031", "zipcode_98032", "zipcode_98042"
                                  ,"zipcode_98055", "zipcode_98058", "zipcode_98077", "zipcode_98188"
                                  , "zipcode_98148","zipcode_98168","zipcode_98092","zipcode_98198"
                                  ,"zipcode_98070","grade_3 Poor","waterfront_NO"], axis = 1)
```

In [49]:
```python
revised_model = sm.OLS(target, sm.add_constant(revised_df))
revised_results = revised_model.fit()
```

In [50]:

```
print(revised_results.summary())
```

OLS Regression Results

Dep. Variable:	price	R-squared:	0.810
Model:	OLS	Adj. R-squared:	0.809
Method:	Least Squares	F-statistic:	1274.
Date:	Sun, 22 Jan 2023	Prob (F-statistic):	0.00
Time:	10:12:13	Log-Likelihood:	-2.5734e+05
No. Observations:	20139	AIC:	5.148e+05
Df Residuals:	20071	BIC:	5.153e+05
Df Model:	67		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.736e+07	1.55e+06	-11.190	0.000	-2.04e+07	-1.43e+07
bedrooms	-1100.0639	864.564	-1.272	0.203	-2794.680	594.553
sqft_living	110.9095	1.395	79.485	0.000	108.174	113.644
lat	3.45e+05	8638.089	39.943	0.000	3.28e+05	3.62e+05
long	-1.065e+04	1.16e+04	-0.920	0.358	-3.33e+04	1.2e+04
view_EXCELLENT	9.417e+04	9003.015	10.460	0.000	7.65e+04	1.12e+05
view_GOOD	2.571e+04	5505.471	4.669	0.000	1.49e+04	3.65e+04
view_NONE	-6.075e+04	2779.500	-21.858	0.000	-6.62e+04	-5.53e+04
zipcode_98004	4.075e+05	7588.574	53.699	0.000	3.93e+05	4.22e+05
zipcode_98005	2.463e+05	7371.820	33.414	0.000	2.32e+05	2.61e+05
zipcode_98006	1.987e+05	4725.756	42.041	0.000	1.89e+05	2.08e+05
zipcode_98007	1.649e+05	7683.570	21.455	0.000	1.5e+05	1.8e+05
zipcode_98008	1.513e+05	5694.763	26.569	0.000	1.4e+05	1.62e+05
zipcode_98010	9.98e+04	9111.519	10.953	0.000	8.19e+04	1.18e+05
zipcode_98019	-3.58e+04	7246.918	-4.940	0.000	-5e+04	-2.16e+04
zipcode_98022	6.779e+04	6769.352	10.014	0.000	5.45e+04	8.11e+04
zipcode_98023	-1.83e+04	4495.090	-4.071	0.000	-2.71e+04	-9490.531
zipcode_98024	9.049e+04	1.08e+04	8.397	0.000	6.94e+04	1.12e+05
zipcode_98027	1.297e+05	4906.830	26.437	0.000	1.2e+05	1.39e+05
zipcode_98029	1.443e+05	5598.263	25.779	0.000	1.33e+05	1.55e+05
zipcode_98033	1.932e+05	5219.876	37.007	0.000	1.83e+05	2.03e+05
zipcode_98034	4.701e+04	4654.291	10.101	0.000	3.79e+04	5.61e+04
zipcode_98038	2.925e+04	4376.620	6.682	0.000	2.07e+04	3.78e+04
zipcode_98039	5.598e+05	3.51e+04	15.936	0.000	4.91e+05	6.29e+05
zipcode_98040	3.479e+05	7302.947	47.638	0.000	3.34e+05	3.62e+05
zipcode_98045	6.236e+04	7893.578	7.900	0.000	4.69e+04	7.78e+04
zipcode_98052	1.361e+05	4407.164	30.880	0.000	1.27e+05	1.45e+05
zipcode_98053	1.245e+05	5343.599	23.295	0.000	1.14e+05	1.35e+05
zipcode_98056	3.89e+04	4518.904	8.609	0.000	3e+04	4.78e+04
zipcode_98059	5.175e+04	4361.658	11.864	0.000	4.32e+04	6.03e+04
zipcode_98065	6.351e+04	6434.366	9.870	0.000	5.09e+04	7.61e+04
zipcode_98072	3.898e+04	6071.862	6.421	0.000	2.71e+04	5.09e+04
zipcode_98074	1.155e+05	4954.809	23.307	0.000	1.06e+05	1.25e+05
zipcode_98075	1.573e+05	5560.670	28.293	0.000	1.46e+05	1.68e+05
zipcode_98102	2.692e+05	9928.257	27.119	0.000	2.5e+05	2.89e+05
zipcode_98103	1.877e+05	4731.373	39.666	0.000	1.78e+05	1.97e+05
zipcode_98105	2.555e+05	7113.987	35.911	0.000	2.42e+05	2.69e+05
zipcode_98106	2.181e+04	5294.405	4.120	0.000	1.14e+04	3.22e+04
zipcode_98107	1.888e+05	6270.634	30.116	0.000	1.77e+05	2.01e+05
zipcode_98108	2.576e+04	6626.665	3.887	0.000	1.28e+04	3.87e+04
zipcode_98109	2.914e+05	9922.805	29.370	0.000	2.72e+05	3.11e+05
zipcode_98112	2.993e+05	7421.443	40.329	0.000	2.85e+05	3.14e+05
zipcode_98115	1.88e+05	4660.527	40.344	0.000	1.79e+05	1.97e+05
zipcode_98116	1.962e+05	5723.631	34.270	0.000	1.85e+05	2.07e+05
zipcode_98117	1.769e+05	5056.509	34.991	0.000	1.67e+05	1.87e+05
zipcode_98118	7.413e+04	4271.551	17.353	0.000	6.58e+04	8.25e+04
zipcode_98119	2.776e+05	7731.911	35.899	0.000	2.62e+05	2.93e+05
zipcode_98122	2.003e+05	5760.419	34.776	0.000	1.89e+05	2.12e+05
zipcode_98125	5.19e+04	5292.516	9.806	0.000	4.15e+04	6.23e+04
zipcode_98126	1.005e+05	5291.665	18.983	0.000	9.01e+04	1.11e+05
zipcode_98133	3779.5949	5221.256	0.724	0.469	-6454.496	1.4e+04
zipcode_98136	1.635e+05	6074.849	26.913	0.000	1.52e+05	1.75e+05
zipcode_98144	1.342e+05	5347.013	25.095	0.000	1.24e+05	1.45e+05
zipcode_98146	3.219e+04	5630.675	5.717	0.000	2.12e+04	4.32e+04
zipcode_98155	-1.072e+04	5314.075	-2.017	0.044	-2.11e+04	-301.328
zipcode_98166	5.433e+04	5892.126	9.221	0.000	4.28e+04	6.59e+04
zipcode_98177	7.358e+04	6822.137	10.785	0.000	6.02e+04	8.69e+04
zipcode_98178	-2.26e+04	5557.390	-4.067	0.000	-3.35e+04	-1.17e+04
zipcode_98199	2.256e+05	6371.699	35.413	0.000	2.13e+05	2.38e+05
grade_11 Excellent	5.693e+04	9570.733	5.948	0.000	3.82e+04	7.57e+04
grade_12 Luxury	-2.176e+05	6.11e+04	-3.561	0.000	-3.37e+05	-9.78e+04
grade_4 Low	-1.767e+05	1.72e+04	-10.285	0.000	-2.1e+05	-1.43e+05
grade_5 Fair	-1.836e+05	7084.645	-25.918	0.000	-1.98e+05	-1.7e+05
grade_6 Low Average	-1.76e+05	4683.078	-37.575	0.000	-1.85e+05	-1.67e+05
grade_7 Average	-1.543e+05	4057.989	-38.012	0.000	-1.62e+05	-1.46e+05
grade_8 Good	-1.173e+05	3822.404	-30.684	0.000	-1.25e+05	-1.1e+05
grade_9 Better	-4.505e+04	3850.252	-11.701	0.000	-5.26e+04	-3.75e+04
waterfront_YES	1.851e+05	1.4e+04	13.260	0.000	1.58e+05	2.13e+05

Omnibus:	1674.547	Durbin-Watson:	1.985
Prob(Omnibus):	0.000	Jarque-Bera (JB):	5577.670
Skew:	0.406	Prob(JB):	0.00

```
Kurtosis:                       5.447   Cond. No.                     5.38e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.38e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [51]:

```
baseline_results.rsquared_adj, revised_results.rsquared_adj
```

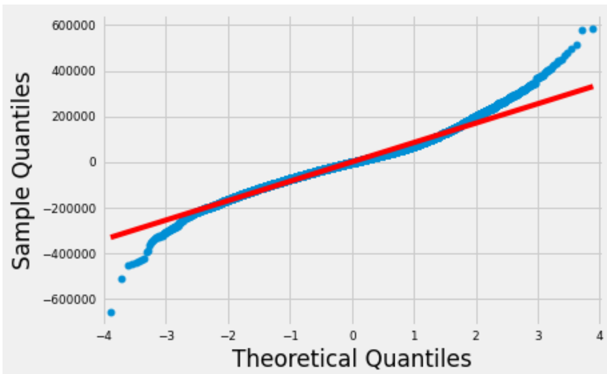
Out[51]:

```
(0.365011142951163, 0.8089660021397984)
```

In [52]:

```
# check for the normality of the residuals
sm.qqplot(iterated_results.resid, line='s')
# also check that the mean of the residuals is approx. 0.
mean_residuals = sum(iterated_results.resid)/ len(iterated_results.resid)
print("The mean of the residuals is {:.4}".format(mean_residuals))
```

```
The mean of the residuals is 1.99e-07
```



In [53]:

```
revised_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20139 entries, 7129300520 to 1523300157
Data columns (total 67 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   bedrooms            20139 non-null  int64  
 1   sqft_living         20139 non-null  int64  
 2   lat                 20139 non-null  float64
 3   long                20139 non-null  float64
 4   view_EXCELLENT      20139 non-null  uint8  
 5   view_GOOD           20139 non-null  uint8  
 6   view_NONE           20139 non-null  uint8  
 7   zipcode_98004       20139 non-null  uint8  
 8   zipcode_98005       20139 non-null  uint8  
 9   zipcode_98006       20139 non-null  uint8  
 10  zipcode_98007       20139 non-null  uint8  
 11  zipcode_98008       20139 non-null  uint8  
 12  zipcode_98010       20139 non-null  uint8  
 13  zipcode_98019       20139 non-null  uint8  
 14  zipcode_98022       20139 non-null  uint8  
 15  zipcode_98023       20139 non-null  uint8  
 16  zipcode_98024       20139 non-null  uint8  
 17  zipcode_98027       20139 non-null  uint8  
 18  zipcode_98029       20139 non-null  uint8  
 19  zipcode_98033       20139 non-null  uint8  
 20  zipcode_98034       20139 non-null  uint8  
 21  zipcode_98038       20139 non-null  uint8  
 22  zipcode_98039       20139 non-null  uint8  
 23  zipcode_98040       20139 non-null  uint8  
 24  zipcode_98045       20139 non-null  uint8  
 25  zipcode_98052       20139 non-null  uint8  
 26  zipcode_98053       20139 non-null  uint8  
 27  zipcode_98056       20139 non-null  uint8  
 28  zipcode_98059       20139 non-null  uint8  
 29  zipcode_98065       20139 non-null  uint8  
 30  zipcode_98072       20139 non-null  uint8  
 31  zipcode_98074       20139 non-null  uint8  
 32  zipcode_98075       20139 non-null  uint8  
 33  zipcode_98102       20139 non-null  uint8  
 34  zipcode_98103       20139 non-null  uint8  
 35  zipcode_98105       20139 non-null  uint8  
 36  zipcode_98106       20139 non-null  uint8  
 37  zipcode_98107       20139 non-null  uint8  
 38  zipcode_98108       20139 non-null  uint8  
 39  zipcode_98109       20139 non-null  uint8  
 40  zipcode_98112       20139 non-null  uint8  
 41  zipcode_98115       20139 non-null  uint8  
 42  zipcode_98116       20139 non-null  uint8  
 43  zipcode_98117       20139 non-null  uint8  
 44  zipcode_98118       20139 non-null  uint8  
 45  zipcode_98119       20139 non-null  uint8  
 46  zipcode_98122       20139 non-null  uint8  
 47  zipcode_98125       20139 non-null  uint8  
 48  zipcode_98126       20139 non-null  uint8  
 49  zipcode_98133       20139 non-null  uint8  
 50  zipcode_98136       20139 non-null  uint8  
 51  zipcode_98144       20139 non-null  uint8  
 52  zipcode_98146       20139 non-null  uint8  
 53  zipcode_98155       20139 non-null  uint8  
 54  zipcode_98166       20139 non-null  uint8  
 55  zipcode_98177       20139 non-null  uint8  
 56  zipcode_98178       20139 non-null  uint8  
 57  zipcode_98199       20139 non-null  uint8  
 58  grade_11 Excellent  20139 non-null  uint8  
 59  grade_12 Luxury     20139 non-null  uint8  
 60  grade_4 Low         20139 non-null  uint8  
 61  grade_5 Fair        20139 non-null  uint8  
 62  grade_6 Low Average 20139 non-null  uint8  
 63  grade_7 Average     20139 non-null  uint8  
 64  grade_8 Good        20139 non-null  uint8  
 65  grade_9 Better      20139 non-null  uint8  
 66  waterfront_YES      20139 non-null  uint8  
dtypes: float64(2), int64(2), uint8(63)
memory usage: 2.0 MB
```

In [54]:
```
final_df = pd.concat([target, revised_df], axis=1)
final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20139 entries, 7129300520 to 1523300157
Data columns (total 68 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   price               20139 non-null  float64
 1   bedrooms            20139 non-null  int64  
 2   sqft_living         20139 non-null  int64  
 3   lat                 20139 non-null  float64
 4   long                20139 non-null  float64
 5   view_EXCELLENT      20139 non-null  uint8  
 6   view_GOOD           20139 non-null  uint8  
 7   view_NONE           20139 non-null  uint8  
 8   zipcode_98004       20139 non-null  uint8  
 9   zipcode_98005       20139 non-null  uint8  
 10  zipcode_98006       20139 non-null  uint8  
 11  zipcode_98007       20139 non-null  uint8  
 12  zipcode_98008       20139 non-null  uint8  
 13  zipcode_98010       20139 non-null  uint8  
 14  zipcode_98019       20139 non-null  uint8  
 15  zipcode_98022       20139 non-null  uint8  
 16  zipcode_98023       20139 non-null  uint8  
 17  zipcode_98024       20139 non-null  uint8  
 18  zipcode_98027       20139 non-null  uint8  
 19  zipcode_98029       20139 non-null  uint8  
 20  zipcode_98033       20139 non-null  uint8  
 21  zipcode_98034       20139 non-null  uint8  
 22  zipcode_98038       20139 non-null  uint8  
 23  zipcode_98039       20139 non-null  uint8  
 24  zipcode_98040       20139 non-null  uint8  
 25  zipcode_98045       20139 non-null  uint8  
 26  zipcode_98052       20139 non-null  uint8  
 27  zipcode_98053       20139 non-null  uint8  
 28  zipcode_98056       20139 non-null  uint8  
 29  zipcode_98059       20139 non-null  uint8  
 30  zipcode_98065       20139 non-null  uint8  
 31  zipcode_98072       20139 non-null  uint8  
 32  zipcode_98074       20139 non-null  uint8  
 33  zipcode_98075       20139 non-null  uint8  
 34  zipcode_98102       20139 non-null  uint8  
 35  zipcode_98103       20139 non-null  uint8  
 36  zipcode_98105       20139 non-null  uint8  
 37  zipcode_98106       20139 non-null  uint8  
 38  zipcode_98107       20139 non-null  uint8  
 39  zipcode_98108       20139 non-null  uint8  
 40  zipcode_98109       20139 non-null  uint8  
 41  zipcode_98112       20139 non-null  uint8  
 42  zipcode_98115       20139 non-null  uint8  
 43  zipcode_98116       20139 non-null  uint8  
 44  zipcode_98117       20139 non-null  uint8  
 45  zipcode_98118       20139 non-null  uint8  
 46  zipcode_98119       20139 non-null  uint8  
 47  zipcode_98122       20139 non-null  uint8  
 48  zipcode_98125       20139 non-null  uint8  
 49  zipcode_98126       20139 non-null  uint8  
 50  zipcode_98133       20139 non-null  uint8  
 51  zipcode_98136       20139 non-null  uint8  
 52  zipcode_98144       20139 non-null  uint8  
 53  zipcode_98146       20139 non-null  uint8  
 54  zipcode_98155       20139 non-null  uint8  
 55  zipcode_98166       20139 non-null  uint8  
 56  zipcode_98177       20139 non-null  uint8  
 57  zipcode_98178       20139 non-null  uint8  
 58  zipcode_98199       20139 non-null  uint8  
 59  grade_11 Excellent  20139 non-null  uint8  
 60  grade_12 Luxury     20139 non-null  uint8  
 61  grade_4 Low         20139 non-null  uint8  
 62  grade_5 Fair        20139 non-null  uint8  
 63  grade_6 Low Average 20139 non-null  uint8  
 64  grade_7 Average     20139 non-null  uint8  
 65  grade_8 Good        20139 non-null  uint8  
 66  grade_9 Better      20139 non-null  uint8  
 67  waterfront_YES      20139 non-null  uint8  
dtypes: float64(3), int64(2), uint8(63)
memory usage: 2.1 MB
```

In [55]:

```python
bedrooms_counts = final_df['bedrooms'].value_counts()
print(bedrooms_counts)
```

```
3     9519
4     6147
2     2727
5     1286
6      218
1      195
7       30
8        9
9        4
10       2
11       1
33       1
Name: bedrooms, dtype: int64
```

In [56]:

```python
# Remove Bedrooms that are 9 or larger from final dataframe
final_df = final_df[final_df['bedrooms'] < 9]
final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20131 entries, 7129300520 to 1523300157
Data columns (total 68 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   price               20131 non-null  float64
 1   bedrooms            20131 non-null  int64  
 2   sqft_living         20131 non-null  int64  
 3   lat                 20131 non-null  float64
 4   long                20131 non-null  float64
 5   view_EXCELLENT      20131 non-null  uint8  
 6   view_GOOD           20131 non-null  uint8  
 7   view_NONE           20131 non-null  uint8  
 8   zipcode_98004       20131 non-null  uint8  
 9   zipcode_98005       20131 non-null  uint8  
 10  zipcode_98006       20131 non-null  uint8  
 11  zipcode_98007       20131 non-null  uint8  
 12  zipcode_98008       20131 non-null  uint8  
 13  zipcode_98010       20131 non-null  uint8  
 14  zipcode_98019       20131 non-null  uint8  
 15  zipcode_98022       20131 non-null  uint8  
 16  zipcode_98023       20131 non-null  uint8  
 17  zipcode_98024       20131 non-null  uint8  
 18  zipcode_98027       20131 non-null  uint8  
 19  zipcode_98029       20131 non-null  uint8  
 20  zipcode_98033       20131 non-null  uint8  
 21  zipcode_98034       20131 non-null  uint8  
 22  zipcode_98038       20131 non-null  uint8  
 23  zipcode_98039       20131 non-null  uint8  
 24  zipcode_98040       20131 non-null  uint8  
 25  zipcode_98045       20131 non-null  uint8  
 26  zipcode_98052       20131 non-null  uint8  
 27  zipcode_98053       20131 non-null  uint8  
 28  zipcode_98056       20131 non-null  uint8  
 29  zipcode_98059       20131 non-null  uint8  
 30  zipcode_98065       20131 non-null  uint8  
 31  zipcode_98072       20131 non-null  uint8  
 32  zipcode_98074       20131 non-null  uint8  
 33  zipcode_98075       20131 non-null  uint8  
 34  zipcode_98102       20131 non-null  uint8  
 35  zipcode_98103       20131 non-null  uint8  
 36  zipcode_98105       20131 non-null  uint8  
 37  zipcode_98106       20131 non-null  uint8  
 38  zipcode_98107       20131 non-null  uint8  
 39  zipcode_98108       20131 non-null  uint8  
 40  zipcode_98109       20131 non-null  uint8  
 41  zipcode_98112       20131 non-null  uint8  
 42  zipcode_98115       20131 non-null  uint8  
 43  zipcode_98116       20131 non-null  uint8  
 44  zipcode_98117       20131 non-null  uint8  
 45  zipcode_98118       20131 non-null  uint8  
 46  zipcode_98119       20131 non-null  uint8  
 47  zipcode_98122       20131 non-null  uint8  
 48  zipcode_98125       20131 non-null  uint8  
 49  zipcode_98126       20131 non-null  uint8  
 50  zipcode_98133       20131 non-null  uint8  
 51  zipcode_98136       20131 non-null  uint8  
 52  zipcode_98144       20131 non-null  uint8  
 53  zipcode_98146       20131 non-null  uint8  
 54  zipcode_98155       20131 non-null  uint8  
 55  zipcode_98166       20131 non-null  uint8  
 56  zipcode_98177       20131 non-null  uint8  
 57  zipcode_98178       20131 non-null  uint8  
 58  zipcode_98199       20131 non-null  uint8  
 59  grade_11 Excellent  20131 non-null  uint8  
 60  grade_12 Luxury     20131 non-null  uint8  
 61  grade_4 Low         20131 non-null  uint8  
 62  grade_5 Fair        20131 non-null  uint8  
 63  grade_6 Low Average 20131 non-null  uint8  
 64  grade_7 Average     20131 non-null  uint8  
 65  grade_8 Good        20131 non-null  uint8  
 66  grade_9 Better      20131 non-null  uint8  
 67  waterfront_YES      20131 non-null  uint8  
dtypes: float64(3), int64(2), uint8(63)
memory usage: 2.1 MB
```

In [57]:
```
bedrooms_counts = final_df['bedrooms'].value_counts()
print(bedrooms_counts)
```

```
3    9519
4    6147
2    2727
5    1286
6     218
1     195
7      30
8       9
Name: bedrooms, dtype: int64
```

In [58]:
```
final_df.describe()
```

Out[58]:

	price	bedrooms	sqft_living	lat	long	view_EXCELLENT	view_GOOD	view_NONE	zipcode_98004	zipcode_98005	...	zipcode_9819
count	20131.00	20131.00	20131.00	20131.00	20131.00	20131.00	20131.00	20131.00	20131.00	20131.00	...	20131.0
mean	467900.80	3.32	1955.09	47.56	-122.21	0.01	0.02	0.92	0.01	0.01	...	0.0
std	196484.62	0.88	755.07	0.14	0.14	0.08	0.12	0.26	0.08	0.08	...	0.1
min	78000.00	1.00	370.00	47.16	-122.52	0.00	0.00	0.00	0.00	0.00	...	0.0
25%	314500.00	3.00	1390.00	47.46	-122.33	0.00	0.00	1.00	0.00	0.00	...	0.0
50%	434900.00	3.00	1840.00	47.57	-122.23	0.00	0.00	1.00	0.00	0.00	...	0.0
75%	594000.00	4.00	2410.00	47.68	-122.12	0.00	0.00	1.00	0.00	0.00	...	0.0
max	1000000.00	8.00	7480.00	47.78	-121.31	1.00	1.00	1.00	1.00	1.00	...	1.0

8 rows × 68 columns

8.2 Exploratory Data Analysis Q1

8.3 Which neighborhoods have the highest average home price?

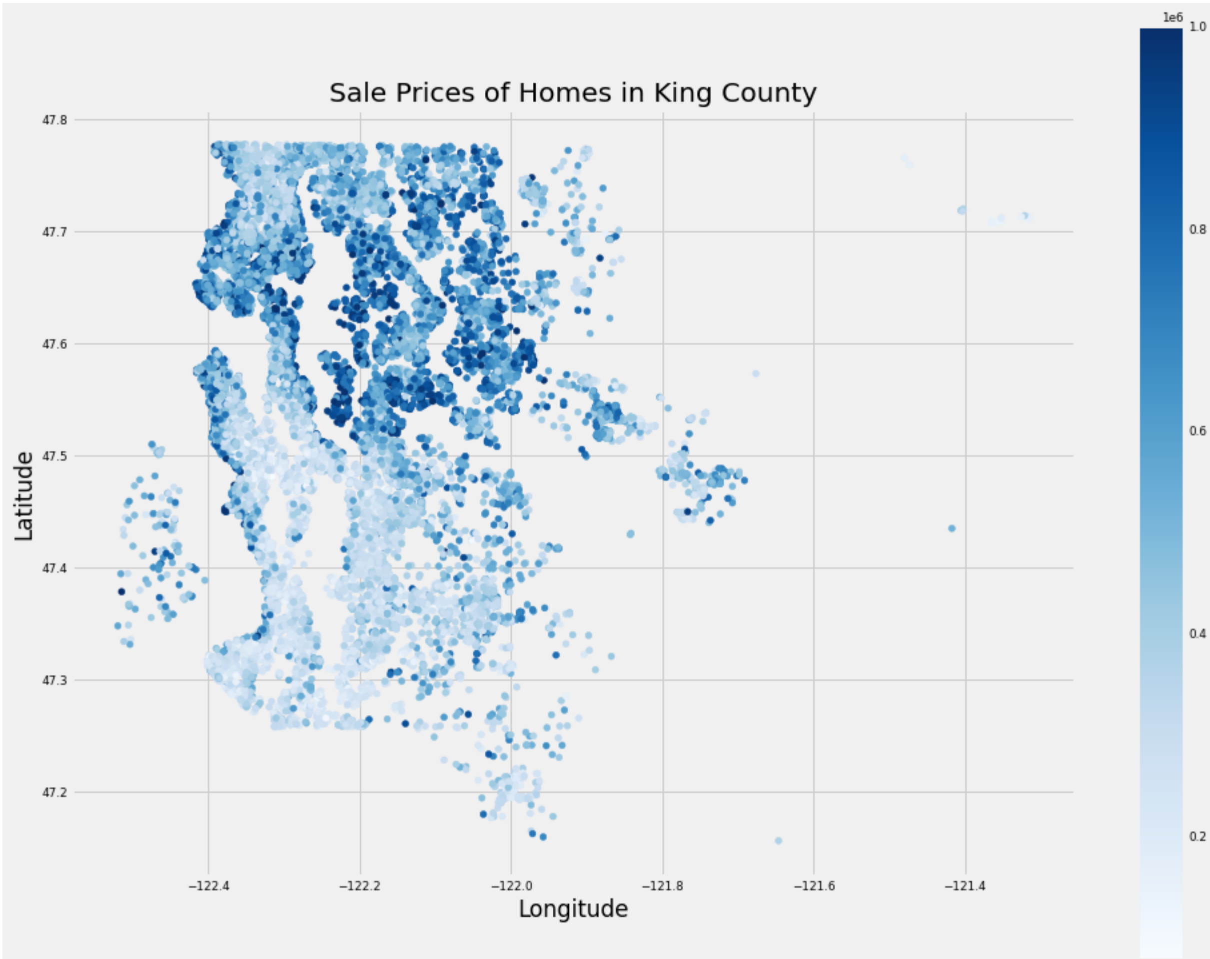
The first thought I had when trying to answer this question was that I should leverage the location of homes in the dataset and use a sequential color palette. The first way that I tried to visualize which neighborhoods have the highest home price was to plot the longitude and lattitude against one another so that I could get a quick geographical visual on where the more expensive homes were located.

In [59]:
```
import geopandas as gpd
def sale_price_map(df):
    """
    Creates a map of sale prices for homes in king county dataset using geopandas

    Inputs:
    df: pandas dataframe with columns 'bedrooms', 'lat', 'long', and 'price'

    Output:
    map of sale prices for homes in king county
    """
    # Set up plot
    #plt.figure(figsize = (20,15))
    # Create a GeoDataFrame from the pandas dataframe
    gdf = gpd.GeoDataFrame(df, geometry=gpd.points_from_xy(df.long, df.lat))
    # Set the CRS (coordinate reference system) of the GeoDataFrame
    gdf.crs = {'init': 'epsg:4326'}
    # Create a map of sale prices for homes in king county
    gdf.plot(column='price', cmap='Blues', legend=True, figsize = (15,12))
    plt.title("Sale Prices of Homes in King County")
    plt.xlabel("Longitude")
    plt.ylabel("Latitude")
    plt.show()
```

In [60]:

```
sale_price_map(df_subset)
```

```
C:\ProgramData\Anaconda3\envs\learn-env\lib\site-packages\pyproj\crs\crs.py:141: FutureWarning: '+init=<authority>:<code>' s
yntax is deprecated. '<authority>:<code>' is the preferred initialization method. When making the change, be mindful of axis
order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-in-proj-6 (https://pyproj4.github.io/
pyproj/stable/gotchas.html#axis-order-changes-in-proj-6)
  in_crs_string = _prepare_from_proj_string(in_crs_string)
```

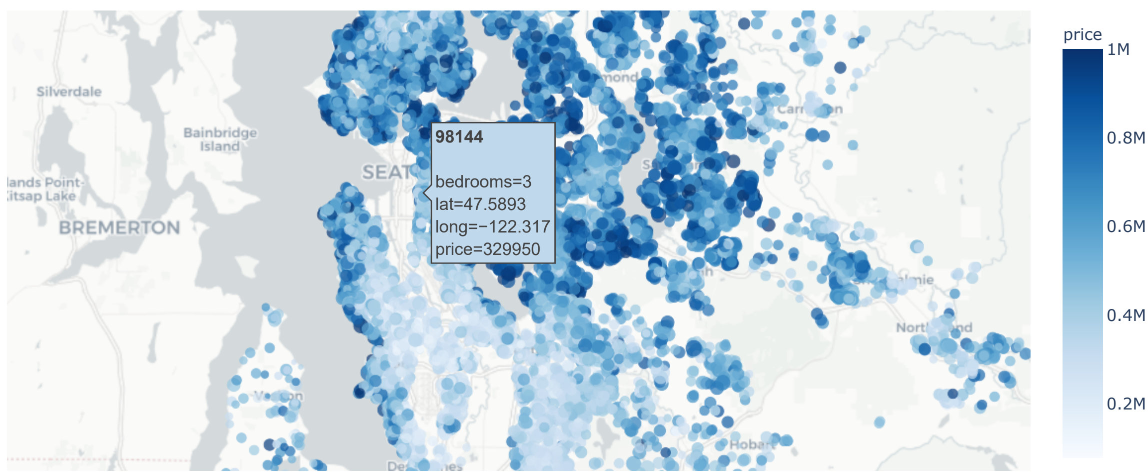


This visualization gave a general idea that the neighborhoods in the North Eastern section of King County were the most pricey and the neighborhoods to the South and Western sections of King County were more affordable. I wanted to make connections at the neighborhood level based on client need, so I used `plotly.express` as `px` and created a figure using a `scatter_mapbox` where I could zoom and hover over individual homes in different neighborhoods.

In [61]:

```
import plotly.express as px
import pandas as pd

# Load the data
data = df_subset

# create the map
fig = px.scatter_mapbox(data_frame=data,
                        lat='lat', # column in data that contains latitude
                        lon='long', # column in data that contains longitude
                        color='price', # column in data that contains housing prices
                        title='Housing Prices in King County by Neighborhood Location',
                        hover_name='zipcode', # column in data that contains address of the property
                        size='bedrooms', # column in data that contains square footage of the property
                        zoom=9,
                        color_continuous_scale="Blues",
                        mapbox_style='carto-positron')

fig.show()
```

Housing Prices in King County by Neighborhood Location



8.4 Exploratory Data Analysis Q2

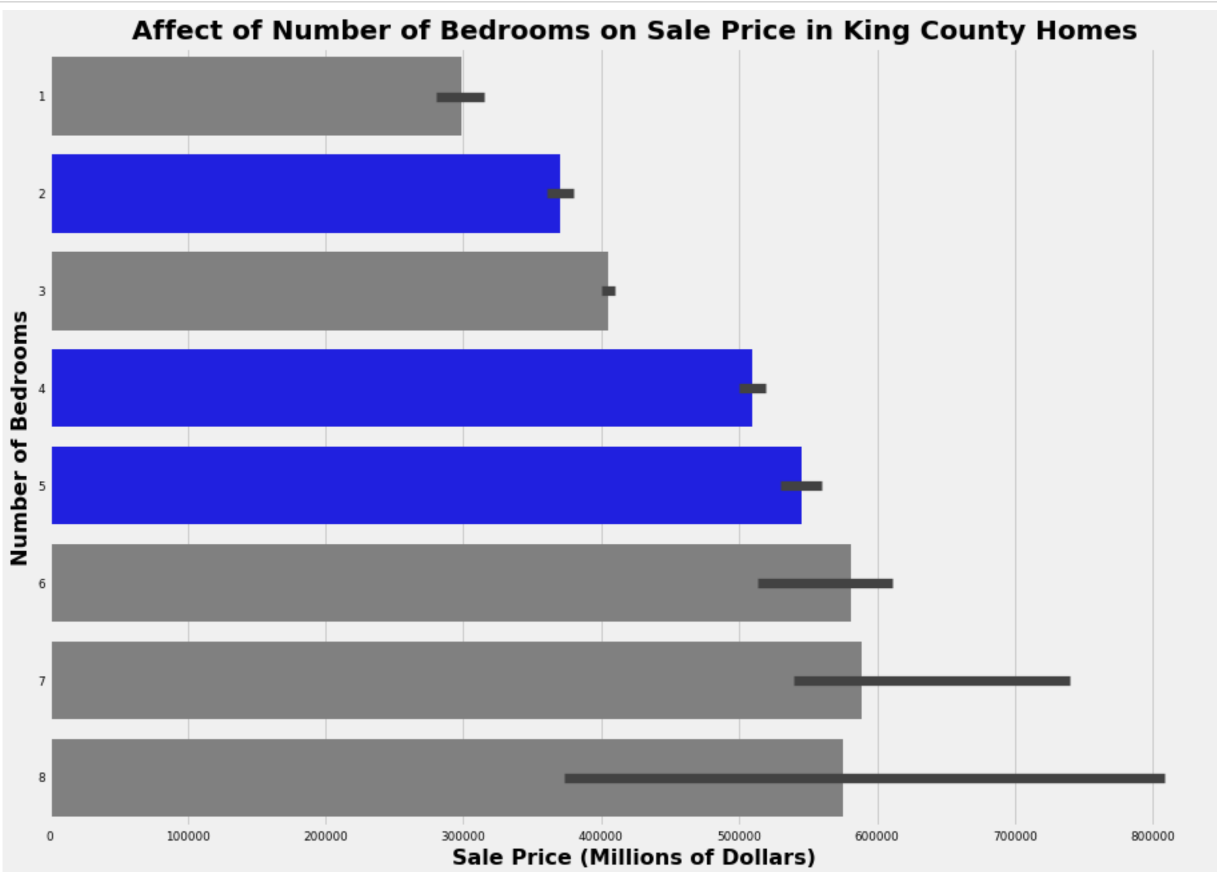
8.5 How does the number of bedrooms affect the sale price of a home?

In [62]:

```
def plot_bedrooms_price(df):
    """
    Plot bar graph of number of bedrooms vs price

    Inputs:
    df: pandas dataframe
    lim: integer
    xlabel: independent variable label
    ylabel: dependent  variable label

    Output:
    bar plot
    """
    # Create plot variables
    prices = np.array(df.price)
    bedrooms = np.array(df.bedrooms)
    median_price = np.median(prices)

    # Create plot labels
    x_label = "Sale Price (Millions of Dollars)"
    y_label = "Number of Bedrooms"
    title = "Affect of Number of Bedrooms on Sale Price in King County Homes"

    # Create a palette that highlights the top 5 sale price values as a function of the number of bedrooms"
    clrs = ['grey' if (x < median_price) else 'blue' for x in prices]
    # Set up plot figure size
    plt.figure(figsize = (14,10))

    # Plot bar using input of bedrooms, price, index and palette
    ax = sns.barplot(x = prices, y = bedrooms, palette = clrs, estimator = np.median, orient = 'h')
    # Setup titles and axes labels
    ax.set_title(title, weight='bold').set_fontsize('20')
    ax.set_ylabel(y_label, fontsize='16', weight='bold')
    ax.set_xlabel(x_label, fontsize='16', weight='bold')
```

In [63]:

```
plot_bedrooms_price(final_df)
```



In [64]:
```
final_df.describe()
```

Out[64]:

	price	bedrooms	sqft_living	lat	long	view_EXCELLENT	view_GOOD	view_NONE	zipcode_98004	zipcode_98005	...	zipcode_9819
count	20131.00	20131.00	20131.00	20131.00	20131.00	20131.00	20131.00	20131.00	20131.00	20131.00	...	20131.0
mean	467900.80	3.32	1955.09	47.56	-122.21	0.01	0.02	0.92	0.01	0.01	...	0.0
std	196484.62	0.88	755.07	0.14	0.14	0.08	0.12	0.26	0.08	0.08	...	0.1
min	78000.00	1.00	370.00	47.16	-122.52	0.00	0.00	0.00	0.00	0.00	...	0.0
25%	314500.00	3.00	1390.00	47.46	-122.33	0.00	0.00	1.00	0.00	0.00	...	0.0
50%	434900.00	3.00	1840.00	47.57	-122.23	0.00	0.00	1.00	0.00	0.00	...	0.0
75%	594000.00	4.00	2410.00	47.68	-122.12	0.00	0.00	1.00	0.00	0.00	...	0.0
max	1000000.00	8.00	7480.00	47.78	-121.31	1.00	1.00	1.00	1.00	1.00	...	1.0

8 rows × 68 columns

8.6 Exploratory Data Analysis Q3

8.7 How does proximity to a highly rated school affect the sale price of a home?

In [65]:
```
# Read Tables from the GreatSchools Website
param = {'lat': 47.5480339, 'locationLabel': 'King County, WA, USA',
         'locationType': 'street_address', 'lon': -121.9836029,
         'st[]': ['public_charter', 'public'], 'state': 'WA', 'view': 'table'}

url = "https://www.greatschools.org/search/search.page?" + urlencode(param)
driver = webdriver.Chrome('C:/chromedriver_win32/chromedriver.exe')

try:
    driver.get(url)
except WebDriverException:
    print("page down")

html = driver.page_source

table = pd.read_html(html)
king_county_schools_df = table[0]

driver.close()
```

In [66]:
```python
king_county_schools_df.head(28)
```

Out[66]:

	School	Type	Grades	Total students enrolled	Students per teacher	Reviews	District
0	Assigned school7/10Above averageGrand Ridge El...	Public district	K-5	747.00	19:1	8 Reviews	Issaquah School District
1	Assigned school7/10Above averagePacific Cascad...	Public district	6-8	1000.00	23:1	6 Reviews	Issaquah School District
2	Assigned school8/10Above averageIssaquah High ...	Public district	9-12	2417.00	26:1	23 Reviews	Issaquah School District
3	10/10Above averageSkyline High School2 awardsA...	Public district	9-12	2169.00	25:1	21 Reviews	Issaquah School District
4	9/10Above averageSunny Hills Elementary School...	Public district	K-5	732.00	18:1	11 Reviews	Issaquah School District
5	9/10Above averageDiscovery Elementary School23...	Public district	PK-5	686.00	18:1	6 Reviews	Issaquah School District
6	8/10Above averageFall City Elementary School33...	Public district	K-5	544.00	19:1	8 Reviews	Snoqualmie Valley School District
7	8/10Above averageCascade Ridge Elementary Scho...	Public district	K-5	487.00	17:1	6 Reviews	Issaquah School District
8	8/10Above averageCreekside Elementary School20...	Public district	K-5	723.00	18:1	14 Reviews	Issaquah School District
9	8/10Above averageIssaquah High School2 awardsA...	Public district	9-12	2417.00	26:1	23 Reviews	Issaquah School District
10	8/10Above averagePine Lake Middle School3095 I...	Public district	6-8	943.00	24:1	13 Reviews	Issaquah School District
11	8/10Above averageEndeavour Elementary School26...	Public district	K-5	598.00	19:1	9 Reviews	Issaquah School District
12	7/10Above averageGrand Ridge Elementary School...	Public district	K-5	747.00	19:1	8 Reviews	Issaquah School District
13	7/10Above averagePacific Cascade Middle School...	Public district	6-8	1000.00	23:1	6 Reviews	Issaquah School District
14	7/10Above averageIssaquah Middle School600 2nd...	Public district	6-8	988.00	22:1	13 Reviews	Issaquah School District
15	7/10Above averageBeaver Lake Middle School2502...	Public district	6-8	852.00	22:1	8 Reviews	Issaquah School District
16	6/10AverageIssaquah Valley Elementary School55...	Public district	K-5	620.00	16:1	13 Reviews	Issaquah School District
17	6/10AverageClark Elementary School335 1st Aven...	Public district	K-5	739.00	16:1	14 Reviews	Issaquah School District
18	6/10AverageChallenger Elementary School25200 S...	Public district	K-5	558.00	17:1	10 Reviews	Issaquah School District
19	5/10AverageChief Kanim Middle School32627 Redm...	Public district	6-8	795.00	21:1	12 Reviews	Snoqualmie Valley School District
20	Currently unratedSnoqualmie Valley Christian S...	Private	PK-7	nan	NaN	0 Reviews	NaN
21	Currently unratedEastside Catholic School232 2...	Private	6-12	841.00	NaN	17 Reviews	NaN
22	Currently unratedArbor Schools1107 228th Avenu...	Private	PK-6	108.00	NaN	16 Reviews	NaN
23	Currently unratedEmerald Heights Academy1420 N...	Private	PK-8	50.00	NaN	17 Reviews	NaN
24	Currently unratedSt Joseph School220 Mountain ...	Private	PK-8	169.00	NaN	9 Reviews	NaN
25	Currently unratedIssaquah Montessori School243...	Private	PK-K	88.00	NaN	3 Reviews	NaN
26	Currently unratedDartmoor School22500 SE 64th ...	Private	1-12	nan	NaN	0 Reviews	NaN
27	Currently unratedSnoqualmie Springs School2523...	Private	PK-2	55.00	NaN	8 Reviews	NaN

In [67]:
```python
king_county_schools_df['School'][0]
```

Out[67]:
```
'Assigned school7/10Above averageGrand Ridge Elementary School1739 Northeast Park Drive, Issaquah, WA, 98029 Homes for sale'
```

In [68]:

```
split_df = king_county_schools_df['School'].str.split(" ", expand=True)
print("Shape of split dataframe:", split_df.shape)
# create a list of column names that matches the number of elements in the split dataframe
column_names = ["col{}".format(i) for i in range(split_df.shape[1])]
# assign the split dataframe to the target dataframe with the new column names
king_county_schools_df[column_names] = split_df
# drop the original column
king_county_schools_df.drop('School', axis=1, inplace=True)
```

```
Shape of split dataframe: (28, 24)
```

In [69]:

```
king_county_schools_df.head(28)
```

Out[69]:

	Type	Grades	Total students enrolled	Students per teacher	Reviews	District	col0	col1	col2	col3	...	col14
0	Public district	K-5	747.00	19:1	8 Reviews	Issaquah School District	Assigned	school7/10Above	averageGrand	Ridge	...	sale
1	Public district	6-8	1000.00	23:1	6 Reviews	Issaquah School District	Assigned	school7/10Above	averagePacific	Cascade	...	for
2	Public district	9-12	2417.00	26:1	23 Reviews	Issaquah School District	Assigned	school8/10Above	averageIssaquah	High	...	Washington700
3	Public district	9-12	2169.00	25:1	21 Reviews	Issaquah School District	10/10Above	averageSkyline	High	School2	...	228th
4	Public district	K-5	732.00	18:1	11 Reviews	Issaquah School District	9/10Above	averageSunny	Hills	Elementary	...	sale
5	Public district	PK-5	686.00	18:1	6 Reviews	Issaquah School District	9/10Above	averageDiscovery	Elementary	School2300	...	None
6	Public district	K-5	544.00	19:1	8 Reviews	Snoqualmie Valley School District	8/10Above	averageFall	City	Elementary	...	sale
7	Public district	K-5	487.00	17:1	6 Reviews	Issaquah School District	8/10Above	averageCascade	Ridge	Elementary	...	None
8	Public district	K-5	723.00	18:1	14 Reviews	Issaquah School District	8/10Above	averageCreekside	Elementary	School20777	...	None
9	Public district	9-12	2417.00	26:1	23 Reviews	Issaquah School District	8/10Above	averageIssaquah	High	School2	...	2nd
10	Public district	6-8	943.00	24:1	13 Reviews	Issaquah School District	8/10Above	averagePine	Lake	Middle	...	sale
11	Public district	K-5	598.00	19:1	9 Reviews	Issaquah School District	8/10Above	averageEndeavour	Elementary	School26205	...	None
12	Public district	K-5	747.00	19:1	8 Reviews	Issaquah School District	7/10Above	averageGrand	Ridge	Elementary	...	None
13	Public district	6-8	1000.00	23:1	6 Reviews	Issaquah School District	7/10Above	averagePacific	Cascade	Middle	...	sale
14	Public district	6-8	988.00	22:1	13 Reviews	Issaquah School District	7/10Above	averageIssaquah	Middle	School600	...	None
15	Public district	6-8	852.00	22:1	8 Reviews	Issaquah School District	7/10Above	averageBeaver	Lake	Middle	...	None
16	Public district	K-5	620.00	16:1	13 Reviews	Issaquah School District	6/10AverageIssaquah	Valley	Elementary	School555	...	None
17	Public district	K-5	739.00	16:1	14 Reviews	Issaquah School District	6/10AverageClark	Elementary	School335	1st	...	None
18	Public district	K-5	558.00	17:1	10 Reviews	Issaquah School District	6/10AverageChallenger	Elementary	School25200	Southeast	...	None
19	Public district	6-8	795.00	21:1	12 Reviews	Snoqualmie Valley School District	5/10AverageChief	Kanim	Middle	School32627	...	sale
20	Private	PK-7	nan	NaN	0 Reviews	NaN	Currently	unratedSnoqualmie	Valley	Christian	...	None
21	Private	6-12	841.00	NaN	17 Reviews	NaN	Currently	unratedEastside	Catholic	School232	...	None
22	Private	PK-6	108.00	NaN	16 Reviews	NaN	Currently	unratedArbor	Schools1107	228th	...	None
23	Private	PK-8	50.00	NaN	17 Reviews	NaN	Currently	unratedEmerald	Heights	Academy1420	...	None
24	Private	PK-8	169.00	NaN	9 Reviews	NaN	Currently	unratedSt	Joseph	School220	...	None
25	Private	PK-K	88.00	NaN	3 Reviews	NaN	Currently	unratedIssaquah	Montessori	School24326	...	None
26	Private	1-12	nan	NaN	0 Reviews	NaN	Currently	unratedDartmoor	School22500	SE	...	None

	Type	Grades	Total students enrolled	Students per teacher	Reviews	District	col0	col1	col2	col3	...	col14
27	Private	PK-2	55.00	NaN	8 Reviews	NaN	Currently	unratedSnoqualmie	Springs	School25237	...	sale

28 rows × 30 columns

In [70]:

```
from haversine import haversine

def distance_to_school(lat, long, school_lat, school_long):
    # Create a tuple for the home coordinates
    home_coords = (home_lat, home_long)

    # Create a tuple for the school coordinates
    school_coords = (school_lat, school_long)

    # Use the haversine function to calculate the distance between the two points
    distance = haversine(home_coords, school_coords)

    # Return the distance in miles
    return distance
```

9 Insights

10 Recommendations