Team 1 Housing Data Analysis

Section 1 Data Importing and Preprocessing

Importing Libraries

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading data

We used a dataset of house_sales in a comma seperated value format. We imported the .csv file using the pandas .read_csv method to load the data into a dataframe

In [ ]:
```
#creating the dataframe by importing the csv file
housing_data = pd.read_csv('house_sales.csv')
#displaying the first 5 rows of the dataframe
print(housing_data.head())
```

```
           id             date     price  bedrooms  bathrooms  sqft_living  \
0  7129300520  20141013T000000  221900.0       3.0       1.00       1180.0
1  6414100192  20141209T000000  538000.0       3.0       2.25       2570.0
2  5631500400  20150225T000000  180000.0       2.0       1.00        770.0
3  2487200875  20141209T000000  604000.0       4.0       3.00       1960.0
4  1954400510  20150218T000000  510000.0       3.0       2.00       1680.0

   sqft_lot  floors  waterfront  view  ...  grade  sqft_above  sqft_basement  \
0    5650.0     1.0           0     0  ...      7        1180              0
1    7242.0     2.0           0     0  ...      7        2170            400
2   10000.0     1.0           0     0  ...      6         770              0
3    5000.0     1.0           0     0  ...      7        1050            910
4    8080.0     1.0           0     0  ...      8        1680              0

   yr_built  yr_renovated  zipcode      lat     long  sqft_living15  \
0      1955             0    98178  47.5112 -122.257           1340
1      1951          1991    98125  47.7210 -122.319           1690
2      1933             0    98028  47.7379 -122.233           2720
3      1965             0    98136  47.5208 -122.393           1360
4      1987             0    98074  47.6168 -122.045           1800

   sqft_lot15
0        5650
1        7639
2        8062
3        5000
4        7503

[5 rows x 21 columns]
```

Now we are going to get the dimensions of the newly imported dataframe named housing_data

In [ ]:
```
#dataframe dimensions
print(f"The dimensions of the initial dataframe are {housing_data.shape}")
```

```
The dimensions of the initial dataframe are (21613, 21)
```

Next we will get a sense for the columns in the dataframe and their data types

In [ ]:
```
#printing columns in dataframe
print(housing_data.columns)
```

```
Index(['id', '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')
```

In [ ]:
```
#printing the data types of the columns
print(housing_data.dtypes)
```

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

Next we will view summary statistics of the initial dataset to get a feel for it.

In [ ]:
```
#printing the descriptive statistics of the dataframe
print(housing_data.describe())
```

```
                 id         price      bedrooms     bathrooms   sqft_living  \
count  2.161300e+04  2.161300e+04  20479.000000  20545.000000  20503.000000   
mean   4.580302e+09  5.400881e+05      3.372821      2.113507   2081.073697   
std    2.876566e+09  3.671272e+05      0.930711      0.768913    915.043176   
min    1.000102e+06  7.500000e+04      0.000000      0.000000    290.000000   
25%    2.123049e+09  3.219500e+05      3.000000      1.500000   1430.000000   
50%    3.904930e+09  4.500000e+05      3.000000      2.250000   1920.000000   
75%    7.308900e+09  6.450000e+05      4.000000      2.500000   2550.000000   
max    9.900000e+09  7.700000e+06     33.000000      8.000000  12050.000000   

           sqft_lot        floors    waterfront          view     condition  \
count  2.056900e+04  21613.000000  21613.000000  21613.000000  21613.000000   
mean   1.517982e+04      1.494309      0.007542      0.234303      3.409430   
std    4.148617e+04      0.539989      0.086517      0.766318      0.650743   
min    5.200000e+02      1.000000      0.000000      0.000000      1.000000   
25%    5.040000e+03      1.000000      0.000000      0.000000      3.000000   
50%    7.620000e+03      1.500000      0.000000      0.000000      3.000000   
75%    1.070800e+04      2.000000      0.000000      0.000000      4.000000   
max    1.651359e+06      3.500000      1.000000      4.000000      5.000000   

              grade    sqft_above  sqft_basement      yr_built  yr_renovated  \
count  21613.000000  21613.000000   21613.000000  21613.000000  21613.000000   
mean       7.656873   1788.390691     291.509045   1971.005136     84.402258   
std        1.175459    828.090978     442.575043     29.373411    401.679240   
min        1.000000    290.000000       0.000000   1900.000000      0.000000   
25%        7.000000   1190.000000       0.000000   1951.000000      0.000000   
50%        7.000000   1560.000000       0.000000   1975.000000      0.000000   
75%        8.000000   2210.000000     560.000000   1997.000000      0.000000   
max       13.000000   9410.000000    4820.000000   2015.000000   2015.000000   

            zipcode           lat          long  sqft_living15     sqft_lot15  
count  21613.000000  21613.000000  21613.000000   21613.000000   21613.000000  
mean   98077.939805     47.560053   -122.213896    1986.552492   12768.455652  
std       53.505026      0.138564      0.140828     685.391304   27304.179631  
min    98001.000000     47.155900   -122.519000     399.000000     651.000000  
25%    98033.000000     47.471000   -122.328000    1490.000000    5100.000000  
50%    98065.000000     47.571800   -122.230000    1840.000000    7620.000000  
75%    98118.000000     47.678000   -122.125000    2360.000000   10083.000000  
max    98199.000000     47.777600   -121.315000    6210.000000  871200.000000  
```

Missing Data

In this section we will explore the initial dataframe and understand missing values, performing data cleaning, wrangling and transformation of data where appropriate.

In [ ]:
```python
#Getting the number of missing values in each column
print(housing_data.isnull().sum())
```

```
id                  0
date                0
price               0
bedrooms         1134
bathrooms        1068
sqft_living      1110
sqft_lot         1044
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
```

Now I want to get a feel for how many 0's are in each column of the dataset

In [ ]:
```python
#Counting the number of zeros in each column
print(housing_data.isin([0]).sum())
```

```
id                   0
date                 0
price                0
bedrooms            11
bathrooms            8
sqft_living          0
sqft_lot             0
floors               0
waterfront       21450
view             19489
condition            0
grade                0
sqft_above           0
sqft_basement    13126
yr_built             0
yr_renovated     20699
zipcode              0
lat                  0
long                 0
sqft_living15        0
sqft_lot15           0
dtype: int64
```

At this point I am going to take advantage, and create two new dataframes, one will be cleaned and imputed data, the other will be simply removing all null values from the dataframe. We will start first with the simple removing all nulls values from the dataframe.

In [ ]:
```python
# Removing all NaN values from the original dataframe
housing_data_dropped_nan = housing_data.dropna()
print(housing_data_dropped_nan.shape)
```

```
(17618, 21)
```

At this point we grouped missing values for bedrooms and bathrooms by zipcode and imputed the most common bedroom or bathroom for that zipcode to fill in missing NaN values. By using the mode we get the most common

In [ ]:
```
# Moving the original dataset to a cleaned dataset to avoid modifying the original dataset
cleaned_data = housing_data.copy()

# Handling the missing values in bedrooms column by grouping by zip code and using the mode of the bedrooms in each zip code
# Grouping by zip code and passing bedrooms into the transform function
cleaned_data['bedrooms'] = cleaned_data.groupby('zipcode')['bedrooms'].transform(lambda x: x.fillna(x.mode()[0]))
#count number of missing values in the bedrooms column
print(cleaned_data['bedrooms'].isnull().sum())

# Handling the missing values in the bathrooms column by grouping by zip code then using the mode of the bathrooms in each zip code
# Grouping by zip code and bedroom and passing bathrooms into the transform function
cleaned_data['bathrooms'] = cleaned_data.groupby(['zipcode'])['bathrooms'].transform(lambda x: x.fillna(x.mode()[0]))
#count number of missing values in the bathrooms column
print(cleaned_data['bathrooms'].isnull().sum())
```

```
0
0
```

Now for the missing values in the columns 'sqft_living' and 'sqft_lot' we will use the same groupby approach, leveraging the zip codes. However, for these values it would be acceptable to use the mean as opposed to the mode of the column.

In [ ]:
```
#Handling the missing values from the 'sqft_living' column by grouping by zip code and using the mean of the 'sqft_living' in each zip code
# Grouping by zip code and passing sqft_living into the transform function
cleaned_data['sqft_living'] = cleaned_data.groupby('zipcode')['sqft_living'].transform(lambda x: x.fillna(x.mean()))
#count number of missing values in the sq_ft column
print(cleaned_data['sqft_living'].isnull().sum())

#Handling the missing values from the 'sqft_lot' column by grouping by zip code and using the mean of the 'sqft_lot' in each zip code
# Grouping by zip code and passing sqft_lot into the transform function
cleaned_data['sqft_lot'] = cleaned_data.groupby('zipcode')['sqft_lot'].transform(lambda x: x.fillna(x.mean()))
#count number of missing values in the sqft_lot column
print(cleaned_data['sqft_lot'].isnull().sum())

print(cleaned_data.isnull().sum())
print(cleaned_data.shape)

#Changing values greater than 1 in 'view' to 1
cleaned_data['view'] = cleaned_data['view'].apply(lambda x: 1 if x > 0 else x)
```

```
0
0
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
(21613, 21)
```

Section 2: Data Analysis and Visualizations

Identify Categorical, Numerical, and Ordinal Data

Categorical data includes waterfront, yr_built, yr_renovated, and zip code.

Numerical data includes date, price, sqft_living, sqft_lot, sqft_above, sqft_basement, lat, long, sqft_living15, sqft_lot15.

Ordinal data includes bedroom, bathroom, floors, view, condition and grade.

Measures of Centrality

In [ ]:
```
#printing the descriptive statistics of the cleaned and imputed dataframe
print(cleaned_data.describe())

#calculating mode and median for price
print("\nMode for price is ", cleaned_data['price'].mode()[0])
print("Median for price is ", cleaned_data['price'].median())
#calculating 25% and 75% quartile
pq3,pq1 = np.percentile(cleaned_data['price'], [75,25])
#calculating IQR
iqr = pq3-pq1
print("IQR for price is ", iqr)

#calculating mode and median for sqft_living
print("\nMode for sqft_living is ", cleaned_data['sqft_living'].mode()[0])
print("Median for sqft_living is ", cleaned_data['sqft_living'].median())
#calculating 25% and 75% quartile
sq3,sq1 = np.percentile(cleaned_data['sqft_living'], [75,25])
#calculating IQR
sq_iqr = sq3-sq1
print("IQR for sqft_living is ", sq_iqr)

#calculating mode and median for sqft_lot
print("\nMode for sqft_lot is ", cleaned_data['sqft_lot'].mode()[0])
print("Median for sqft_lot is ", cleaned_data['sqft_lot'].median())
#calculating 25% and 75% quartile
sql3,sql1 = np.percentile(cleaned_data['sqft_lot'], [75,25])
#calculating IQR
sql_iqr = sql3-sql1
print("IQR for sqft_lot is ", sql_iqr)

#calculating mode and median for sqft_above
```

```python
print("\nMode for sqft_above is ", cleaned_data['sqft_above'].mode()[0])
print("Median for sqft_above is ", cleaned_data['sqft_above'].median())
#calculating 25% and 75% quartile
sqa3,sqa1 = np.percentile(cleaned_data['sqft_above'], [75,25])
#calculating IQR
sqa_iqr = sqa3-sqa1
print("IQR for sqft_above is ", sqa_iqr)

#calculating mode and median for sqft_basement
print("\nMode for sqft_basement is ", cleaned_data['sqft_basement'].mode()[0])
print("Median for sqft_basement is ", cleaned_data['sqft_basement'].median())
#calculating 25% and 75% quartile
sqb3,sqb1 = np.percentile(cleaned_data['sqft_basement'], [75,25])
sqb_iqr = sqb3-sqb1
print("IQR for sqft_basement is ", sqb_iqr)
```

```
                 id         price      bedrooms     bathrooms   sqft_living  \
count  2.161300e+04  2.161300e+04  21613.000000  21613.000000  21613.000000   
mean   4.580302e+09  5.400881e+05      3.360015      2.102438   2080.786053   
std    2.876566e+09  3.671272e+05      0.910834      0.768081    895.764068   
min    1.000102e+06  7.500000e+04      0.000000      0.000000    290.000000   
25%    2.123049e+09  3.219500e+05      3.000000      1.500000   1450.000000   
50%    3.904930e+09  4.500000e+05      3.000000      2.250000   1930.000000   
75%    7.308900e+09  6.450000e+05      4.000000      2.500000   2530.000000   
max    9.900000e+09  7.700000e+06     33.000000      8.000000  12050.000000   

           sqft_lot        floors    waterfront          view     condition  \
count  2.161300e+04  21613.000000  21613.000000  21613.000000  21613.000000   
mean   1.519622e+04      1.494309      0.007542      0.098274      3.409430   
std    4.062404e+04      0.539989      0.086517      0.297692      0.650743   
min    5.200000e+02      1.000000      0.000000      0.000000      1.000000   
25%    5.100000e+03      1.000000      0.000000      0.000000      3.000000   
50%    7.700000e+03      1.500000      0.000000      0.000000      3.000000   
75%    1.089100e+04      2.000000      0.000000      0.000000      4.000000   
max    1.651359e+06      3.500000      1.000000      1.000000      5.000000   

              grade    sqft_above  sqft_basement      yr_built  yr_renovated  \
count  21613.000000  21613.000000   21613.000000  21613.000000  21613.000000   
mean       7.656873   1788.390691     291.509045   1971.005136     84.402258   
std        1.175459    828.090978     442.575043     29.373411    401.679240   
min        1.000000    290.000000       0.000000   1900.000000      0.000000   
25%        7.000000   1190.000000       0.000000   1951.000000      0.000000   
50%        7.000000   1560.000000       0.000000   1975.000000      0.000000   
75%        8.000000   2210.000000     560.000000   1997.000000      0.000000   
max       13.000000   9410.000000    4820.000000   2015.000000   2015.000000   

            zipcode           lat          long  sqft_living15     sqft_lot15  
count  21613.000000  21613.000000  21613.000000   21613.000000   21613.000000  
mean   98077.939805     47.560053   -122.213896    1986.552492   12768.455652  
std       53.505026      0.138564      0.140828     685.391304   27304.179631  
min    98001.000000     47.155900   -122.519000     399.000000     651.000000  
25%    98033.000000     47.471000   -122.328000    1490.000000    5100.000000  
50%    98065.000000     47.571800   -122.230000    1840.000000    7620.000000  
75%    98118.000000     47.678000   -122.125000    2360.000000   10083.000000  
max    98199.000000     47.777600   -121.315000    6210.000000  871200.000000  

Mode for price is  350000.0
Median for price is  450000.0
IQR for price is  323050.0

Mode for sqft_living is  1300.0
Median for sqft_living is  1930.0
IQR for sqft_living is  1080.0

Mode for sqft_lot is  5000.0
Median for sqft_lot is  7700.0
IQR for sqft_lot is  5791.0

Mode for sqft_above is  1300
Median for sqft_above is  1560.0
IQR for sqft_above is  1020.0

Mode for sqft_basement is  0
Median for sqft_basement is  0.0
IQR for sqft_basement is  560.0
```

Distribution Visualizations

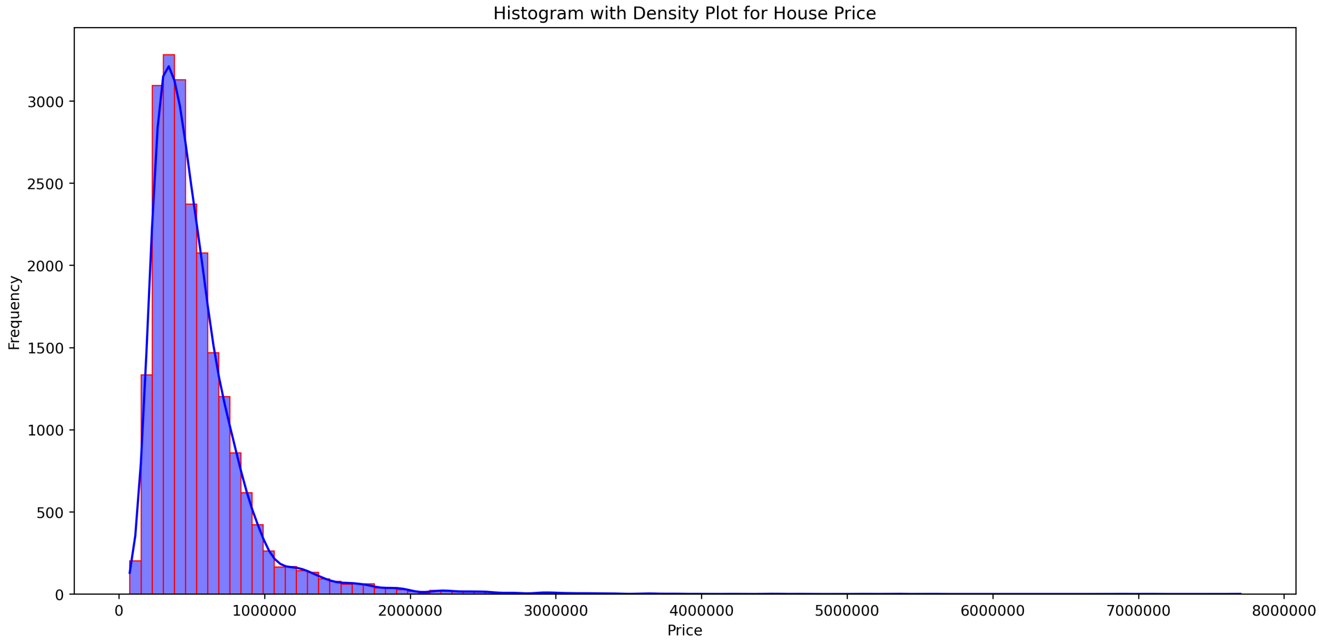
In this section, we will create visualizations for the distribution of data.

Here we are creating a histogram with a density plot for house price. Our data is positively skewed and most of the house price is between $321,950 to$645,000.

In [ ]:

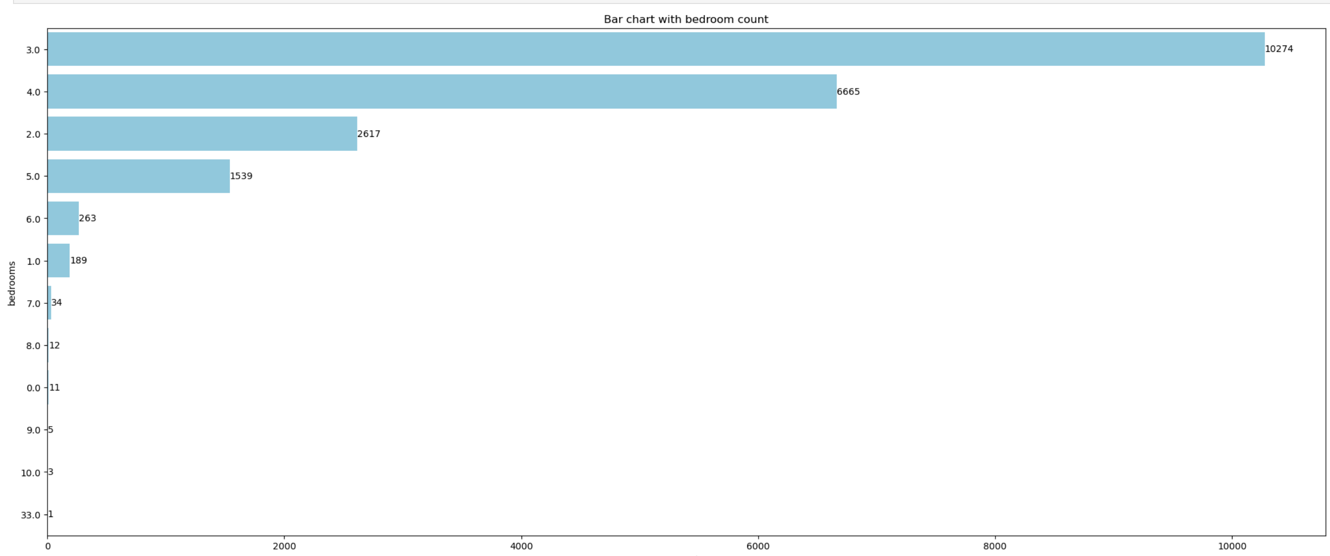
```python
from matplotlib.pyplot import figure
figure(num=None, figsize=(15,7), dpi=256, facecolor='w', edgecolor='r')

#Creating a histogram with a density plot
p1 = sns.histplot(cleaned_data['price'], bins=100, kde=True, color = 'blue', edgecolor='red')

#Adding labels and title
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.title("Histogram with Density Plot for House Price")
plt.ticklabel_format(style='plain')

#showing the number of counts for each xlabel
#for i in p1.containers:
    #p1.bar_label(i,)
```

```
c:\Users\19noa\miniconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to Na
N before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



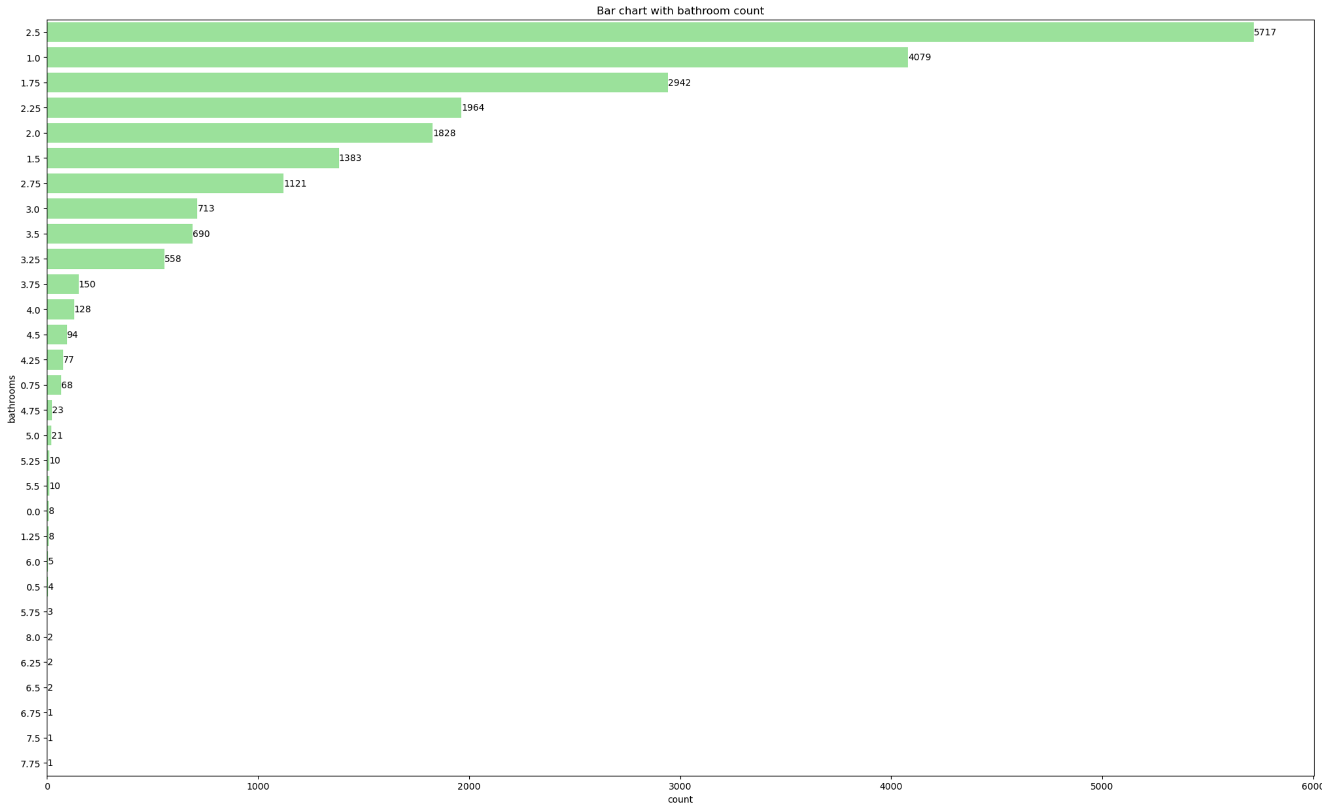
We will now create a bar chart that shows the frequency for the number of bedrooms in a house.

In [ ]:
```
plt.figure(figsize = (25,10))
#print(cleaned_data['bedrooms'].value_counts())

#Bar chart that shows the frequency for the number of bedrooms in a house
bed1 = sns.countplot(y='bedrooms', data = cleaned_data, color='skyblue', order = cleaned_data['bedrooms'].value_counts().index)

#scaling it with logarithmic scale to clearly show the bar plot
#bed1.set_xscale("log")
#ticks = [1, 10, 100, 1000, 10000, 13000]
#bed1.set_xticks(ticks)
#bed1.set_xticklabels(ticks)
plt.title("Bar chart with bedroom count")

#showing the number of counts for each xlabel
for i in bed1.containers:
    bed1.bar_label(i,)
```



We will now create a bar chart that shows the frequency for the number of bathrooms in a house.

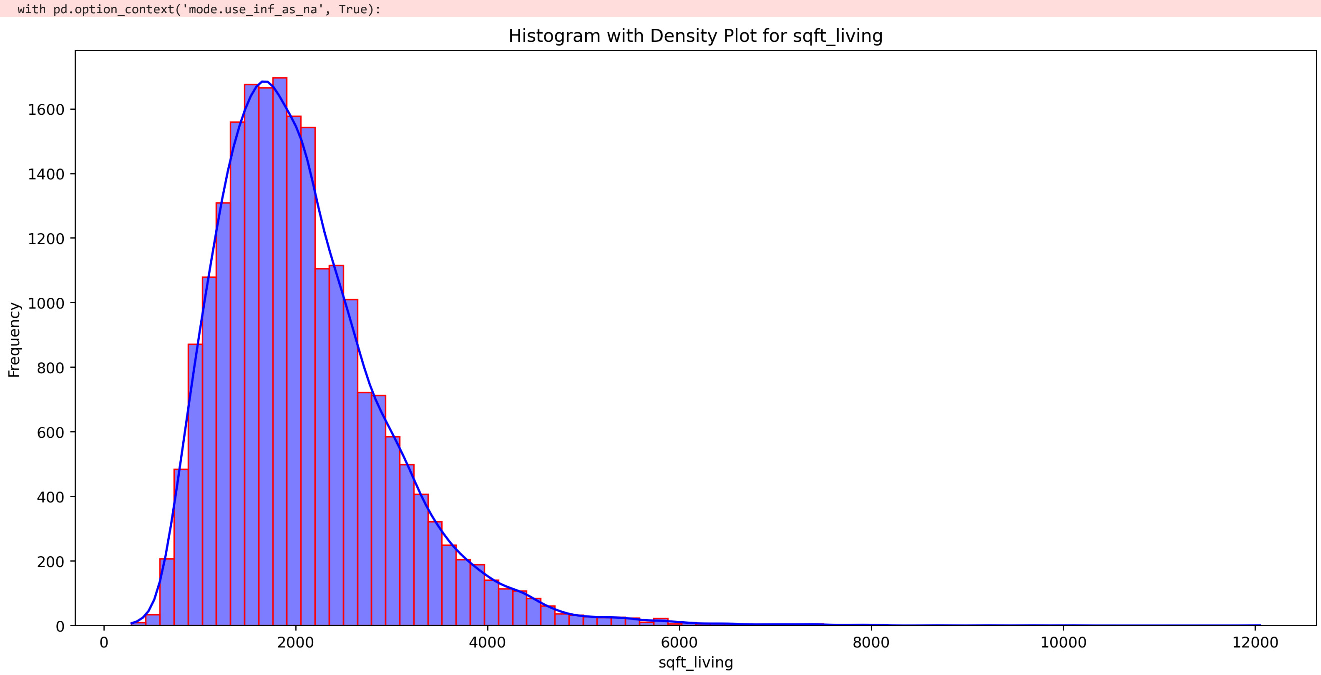
In [ ]:
```
plt.figure(figsize = (25,15))
#print(cleaned_data['bathrooms'].value_counts())

#Bar chart that shows the frequency for the number of bathrooms in a house.
bath1 = sns.countplot(y='bathrooms', data = cleaned_data, color='lightgreen', order = cleaned_data['bathrooms'].value_counts().index)

#scaling it with logarithmic scale to clearly show the bar plot
#bath1.set_xscale("log")
#ticks = [1, 10, 100, 1000, 10000]
#bath1.set_xticks(ticks)
#bath1.set_xticklabels(ticks)
plt.title("Bar chart with bathroom count")

#showing the number of counts for each xlabel
for i in bath1.containers:
    bath1.bar_label(i,)
```



We will now create a histogram with a density plot for sqft_living. The data is positively skewed and most of the sqft_living is between 290 square feet to 2530 square feet.

In [ ]:
```
figure(num=None, figsize=(15,7), dpi=256, facecolor='w', edgecolor='r')

#Creating a histogram with a density plot
living1 = sns.histplot(cleaned_data['sqft_living'], bins=80, kde=True, color = 'blue', edgecolor='red')

#Adding labels and title
plt.xlabel('sqft_living')
plt.ylabel('Frequency')
plt.title("Histogram with Density Plot for sqft_living")
plt.ticklabel_format(style='plain')

#showing the number of counts for each xlabel
#for i in living1.containers:
    #living1.bar_label(i,)
```

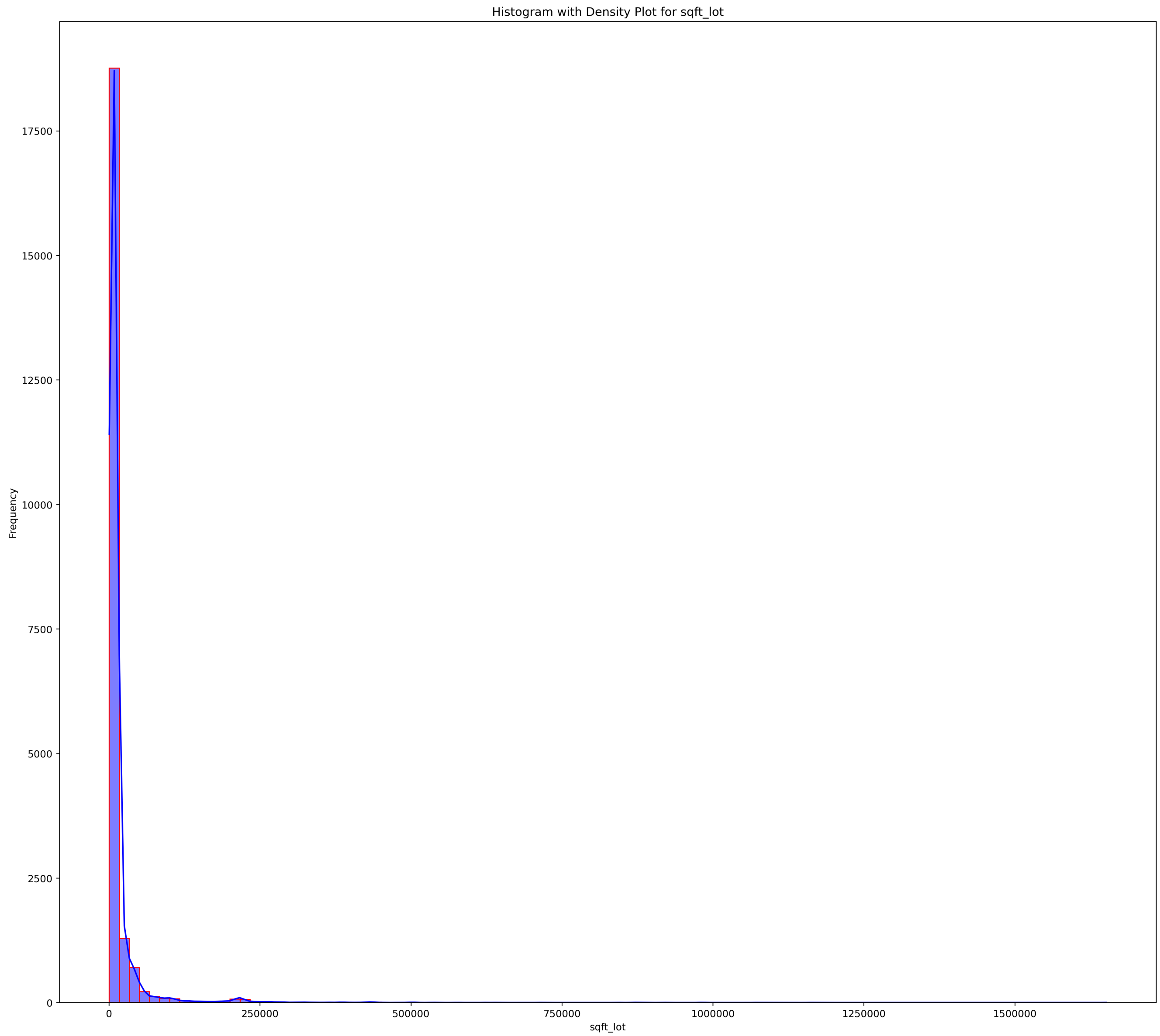
```
c:\Users\19noa\miniconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to Na
N before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



We will now create a histogram with a density plot for sqft_lot. The data is positively skewed and most of the sqft_lot is between 5100 square feet to 10891 square feet.

In [ ]:
```
figure(num=None, figsize=(20,18), dpi=256, facecolor='w', edgecolor='r')

#Creating a histogram with a density plot
lot1 = sns.histplot(cleaned_data['sqft_lot'], bins=99, kde=True, color = 'blue', edgecolor='red')

#Adding labels and title
plt.xlabel('sqft_lot')
plt.ylabel('Frequency')
plt.title("Histogram with Density Plot for sqft_lot")
plt.ticklabel_format(style='plain')

#showing the number of counts for each xlabel
#for i in lot1.containers:
    #lot1.bar_label(i,)
```

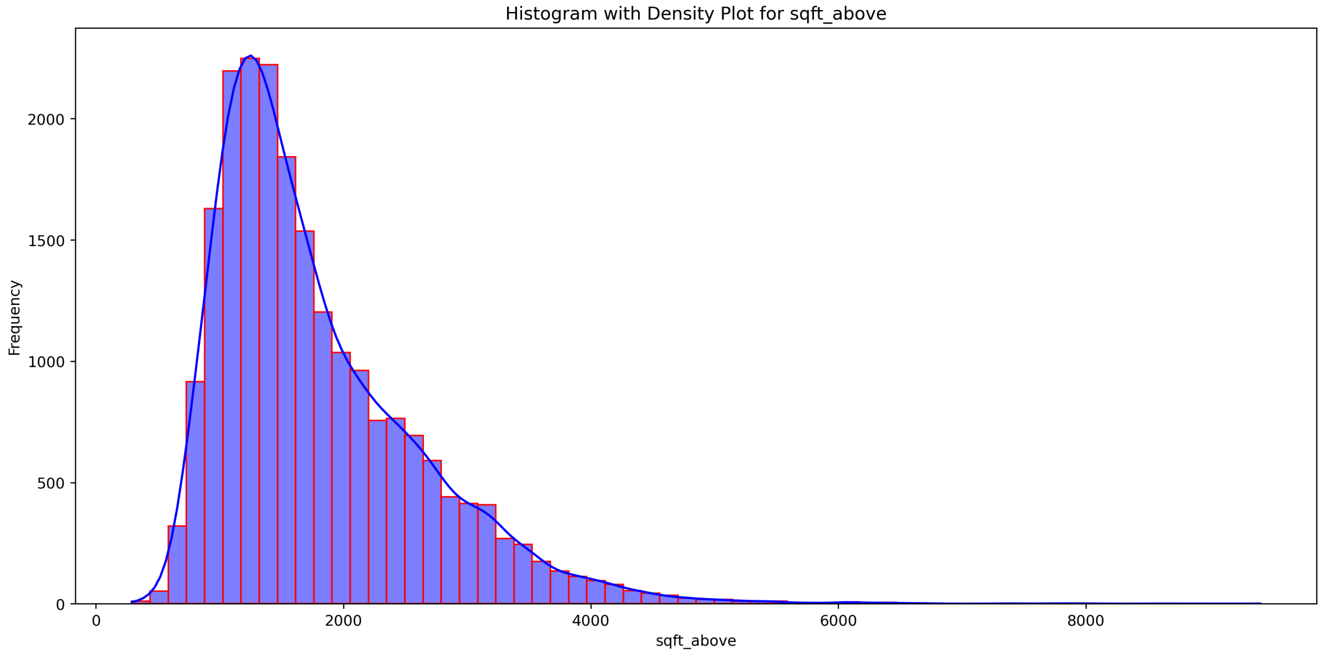
```
c:\Users\19noa\miniconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to Na
N before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



We will now create a histogram with a density plot for sqft_above. The data is positively skewed and most of the sqft_above is between 1190 square feet to 2210 square feet.

In [ ]:
```
figure(num=None, figsize=(15,7), dpi=256, facecolor='w', edgecolor='r')

#Creating a histogram with a density plot
above1 = sns.histplot(cleaned_data['sqft_above'], bins=62, kde=True, color = 'blue', edgecolor='red')

#Adding labels and title
plt.xlabel('sqft_above')
plt.ylabel('Frequency')
plt.title("Histogram with Density Plot for sqft_above")
plt.ticklabel_format(style='plain')

#showing the number of counts for each xlabel
#for i in above1.containers:
    #above1.bar_label(i,)
```

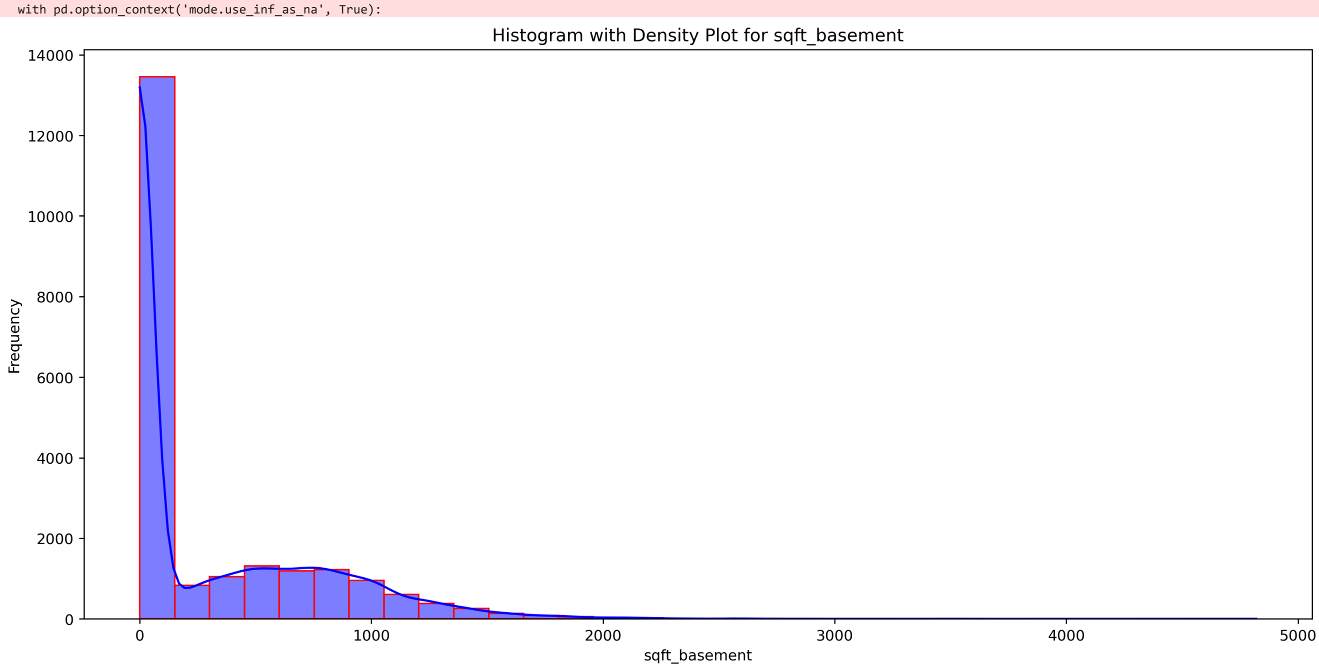
```
c:\Users\19noa\miniconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to Na
N before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



We will now create a histogram with a density plot for sqft_basement. The data is positively skewed and most of the sqft_basement is between 0 square feet to 560 square feet.

In [ ]:
```
figure(num=None, figsize=(15,7), dpi=256, facecolor='w', edgecolor='r')

#Creating a histogram with a density plot
base1 = sns.histplot(cleaned_data['sqft_basement'], bins=32, kde=True, color = 'blue', edgecolor='red')

#Adding labels and title
plt.xlabel('sqft_basement')
plt.ylabel('Frequency')
plt.title("Histogram with Density Plot for sqft_basement")
plt.ticklabel_format(style='plain')

#showing the number of counts for each xlabel
#for i in base1.containers:
    #base1.bar_label(i,)
```

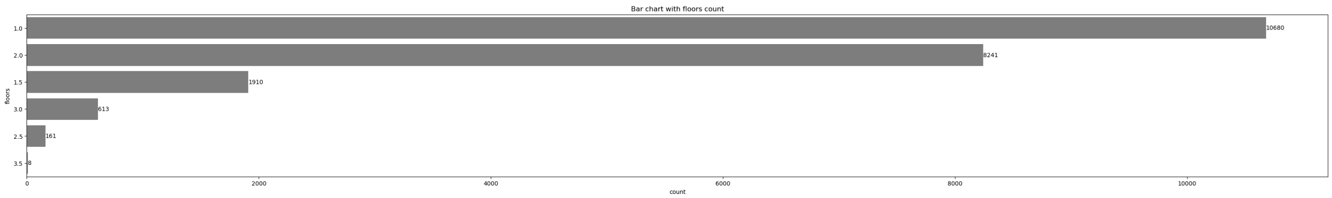
```
c:\Users\19noa\miniconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to Na
N before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



We will now create a bar chart that shows the frequency for the number of floors in a house.

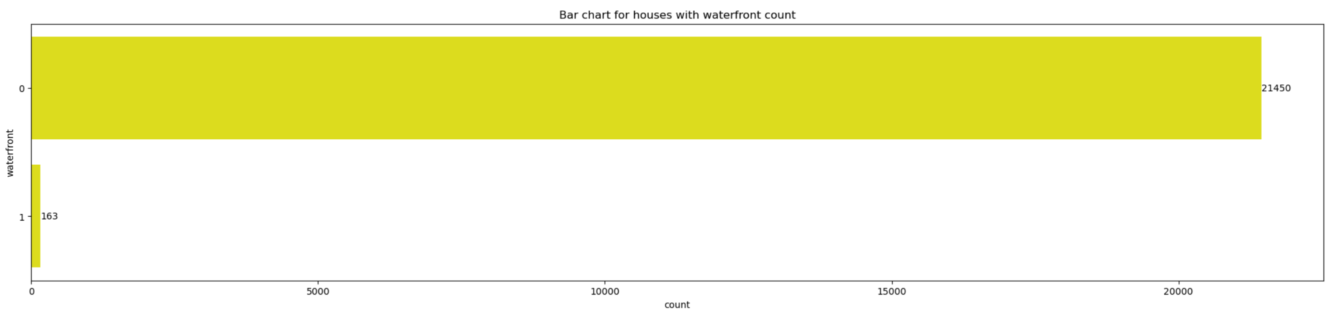
In [ ]:
```
plt.figure(figsize = (40,5))

#Bar chart that shows the frequency for the number of floors in a house.
floors1 = sns.countplot(y='floors', data = cleaned_data, color='grey', order = cleaned_data['floors'].value_counts().index)

#scaling it with logarithmic scale to clearly show the bar plot
#floors1.set_xscale("log")
#ticks = [1, 10, 100, 1000, 10000, 12000]
#floors1.set_xticks(ticks)
#floors1.set_xticklabels(ticks)
```

```
plt.title("Bar chart with floors count")

#showing the number of counts for each xlabel
for i in floors1.containers:
    floors1.bar_label(i,)
```



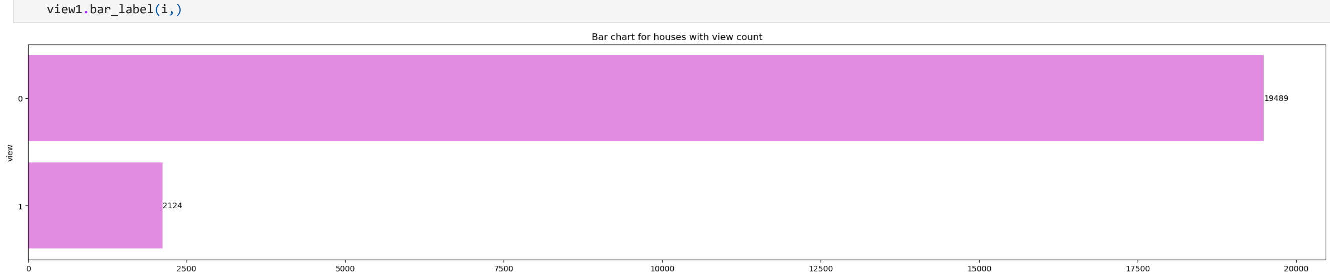
We will now create a bar chart that shows the frequency of there being a waterfront in a house.

In [ ]:
```
plt.figure(figsize = (25,5))
#print(cleaned_data['waterfront'].value_counts())

#Bar chart that shows the frequency of there being a waterfront in a house.
water1 = sns.countplot(y='waterfront', data = cleaned_data, color='yellow', order = cleaned_data['waterfront'].value_counts().index)

#scaling it with logarithmic scale to clearly show the bar plot
#water1.set_xscale("log")
#ticks = [1, 10, 100, 1000, 10000, 25000]
#water1.set_xticks(ticks)
#water1.set_xticklabels(ticks)
plt.title("Bar chart for houses with waterfront count")

#showing the number of counts for each xlabel
for i in water1.containers:
    water1.bar_label(i,)
```



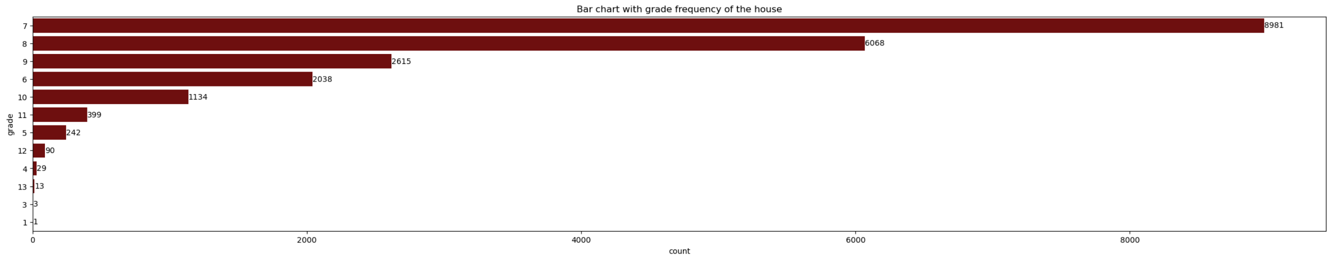
We will now create a bar chart that shows the frequency of there being a view in the house.

In [ ]:
```
plt.figure(figsize = (30,5))
#print(cleaned_data['view'].value_counts())

#Bar chart that shows the frequency of there being a view in the house.
view1 = sns.countplot(y='view', data = cleaned_data, color='violet', order = cleaned_data['view'].value_counts().index)

#scaling it with logarithmic scale to clearly show the bar plot
#view1.set_xscale("log")
#ticks = [1, 10, 100, 1000, 10000, 25000]
#view1.set_xticks(ticks)
#view1.set_xticklabels(ticks)
plt.title("Bar chart for houses with view count")

#showing the number of counts for each xlabel
for i in view1.containers:
    view1.bar_label(i,)
```



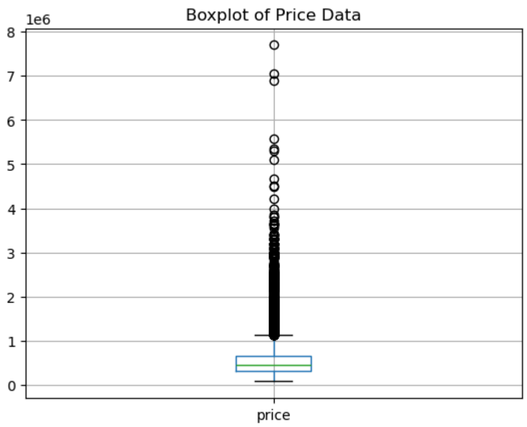
We will now create a bar chart that shows the frequency of the grade of the house.

In [ ]:
```
plt.figure(figsize = (30,5))
#print(cleaned_data['grade'].value_counts())

#Bar that shows the frequency of the grade of the house.
grade1 = sns.countplot(y='grade', data = cleaned_data, color='maroon', order = cleaned_data['grade'].value_counts().index)

#scaling it with logarithmic scale to clearly show the bar plot
#grade1.set_xscale("log")
#ticks = [1, 10, 100, 1000, 10000]
#grade1.set_xticks(ticks)
#grade1.set_xticklabels(ticks)
plt.title("Bar chart with grade frequency of the house")

#showing the number of counts for each xlabel
for i in grade1.containers:
    grade1.bar_label(i,)
```



Using a boxplot to identify outliers, there are approximately 1146 rows that contains an outlier for price and will be kept in the dataset.

In [ ]:
```
price_boxplot = cleaned_data.boxplot(column=['price'])

# plot title
plt.title ('Boxplot of Price Data')
plt.show()

#calculating 25% and 75% quartile
price3,price1 = np.percentile(cleaned_data['price'], [75,25])
price_iqr = price3-price1
#calculating lower and upper bound
price_lower = cleaned_data['price'].min()
price_upper = price3 + 1.5*price_iqr
print("Any value for price less than", price_lower, "dollars is an outlier", "\nAny values greater than",price_upper, "dollars is an outlier.")
cleaned_data[cleaned_data.price > price_upper]
```




```
Any value for price less than 75000.0 dollars is an outlier
Any values greater than 1129575.0 dollars is an outlier.
```

Out[ ]:

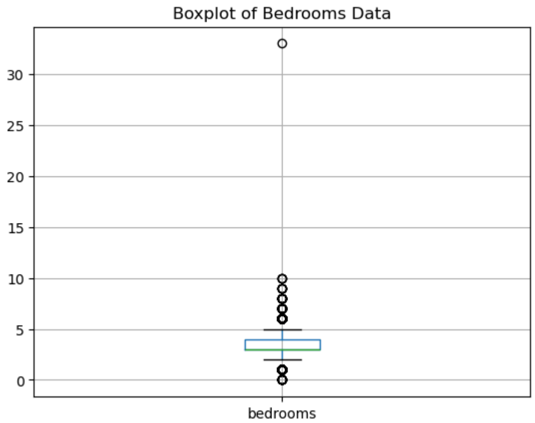
	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	
5	7237550310	20140512T000000	1225000.0	4.0	4.50	5420.000000	101930.000000	1.0	0	0	...	11	3890	1530	2001	0	98053	47
21	2524049179	20140826T000000	2000000.0	3.0	2.75	3050.000000	44867.000000	1.0	0	1	...	9	2330	720	1968	0	98040	47
49	822039084	20150311T000000	1350000.0	3.0	2.50	2753.000000	65005.000000	1.0	1	1	...	9	2165	588	1953	0	98070	47
69	1802000060	20140612T000000	1325000.0	5.0	2.25	3200.000000	12925.178451	1.0	0	0	...	8	1600	1600	1965	0	98004	47
125	4389200955	20150302T000000	1450000.0	4.0	2.75	2750.000000	17789.000000	1.5	0	0	...	8	1980	770	1914	1992	98004	47
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
21568	524059330	20150130T000000	1700000.0	4.0	3.50	3830.000000	8963.000000	2.0	0	0	...	10	3120	710	2014	0	98004	47
21576	9253900271	20150107T000000	3567000.0	5.0	4.50	4850.000000	10584.000000	2.0	1	1	...	10	3540	1310	2007	0	98008	47
21590	7430200100	20140514T000000	1222500.0	4.0	3.50	2634.544153	9444.000000	1.5	0	0	...	11	3110	1800	2007	0	98074	47
21597	191100405	20150421T000000	1575000.0	4.0	3.25	3410.000000	10125.000000	2.0	0	0	...	10	3410	0	2007	0	98040	47
21600	249000205	20141015T000000	1537000.0	5.0	3.75	4470.000000	8088.000000	2.0	0	0	...	11	4470	0	2008	0	98004	47

1146 rows × 21 columns

Using the boxplot, there are approximately 518 rows with outliers within the dataset. The extreme outlier of 33 bedrooms was analyzed and updated to be 3 bedrooms as there seems to be a typo. The sqft_lving of 1620 and sqft_lot of 8049 with 1 floor does not accurately represent a 33 bedroom house. The 0 bedrooms were also removed.

In [ ]:
```
bedrooms_boxplot = cleaned_data.boxplot(column=['bedrooms'])

# plot title
plt.title ('Boxplot of Bedrooms Data')
plt.show()

#calculating 25% and 75% quartile
bed3,bed1 = np.percentile(cleaned_data['bedrooms'], [75,25])
bed_iqr = bed3-bed1
#calculating lower and upper bound
bed_lower = bed1 - 1.5*bed_iqr
bed_upper = bed3 + 1.5*bed_iqr
print("Any value for bedrooms less than",bed_lower, "bedrooms is an outlier", "\nAny values greater than",bed_upper, "bedrooms is an outlier.")
print(cleaned_data[(cleaned_data.bedrooms < bed_lower) | (cleaned_data.bedrooms > bed_upper)].sort_values('bedrooms', ascending = True))

#replacing the 33 bedroom as a 3 bedroom
cleaned_data['bedrooms'].replace(33, 3, inplace=True)
print(cleaned_data[(cleaned_data.bedrooms < bed_lower) | (cleaned_data.bedrooms > bed_upper)].sort_values('bedrooms', ascending = True))

#removing all 0 bedrooms
cleaned_data = cleaned_data.loc[~(cleaned_data['bedrooms'] == 0)]
```

```
print(cleaned_data['bedrooms'].value_counts())
cleaned_data.sort_values('bedrooms', ascending = True)
```



```
Any value for bedrooms less than 1.5 bedrooms is an outlier
Any values greater than 5.5 bedrooms is an outlier.
               id             date      price  bedrooms  bathrooms  \
4868   6896300380  20141002T000000   228000.0       0.0       1.00
18379  1222029077  20141029T000000   265000.0       0.0       0.75
12653  7849202299  20150218T000000   320000.0       0.0       2.50
14423  9543000205  20150413T000000   139950.0       0.0       2.50
9773   3374500520  20150429T000000   355000.0       0.0       0.00
...           ...              ...        ...       ...        ...
4235   2902200015  20150106T000000   700000.0       9.0       3.00
13314   627300145  20140814T000000  1148000.0      10.0       5.25
19254  8812401450  20141229T000000   660000.0      10.0       3.00
15161  5566100170  20141029T000000   650000.0      10.0       2.00
15870  2402100895  20140625T000000   640000.0      33.0       1.75

       sqft_living  sqft_lot  floors  waterfront  view  ...  grade  \
4868         390.0    5900.0     1.0           0     0  ...      4
18379        384.0  213444.0     1.0           0     0  ...      4
12653       1490.0    7111.0     2.0           0     0  ...      7
14423        844.0    4269.0     1.0           0     0  ...      7
9773        2460.0    8049.0     2.0           0     0  ...      8
...            ...       ...     ...         ...   ...  ...    ...
4235        3680.0    4400.0     2.0           0     0  ...      7
13314       4590.0   10920.0     1.0           0     1  ...      9
19254       2920.0    3745.0     2.0           0     0  ...      7
15161       3610.0   11914.0     2.0           0     0  ...      7
15870       1620.0    6000.0     1.0           0     0  ...      7

       sqft_above  sqft_basement  yr_built  yr_renovated  zipcode      lat  \
4868          390              0      1953             0    98118  47.5260
18379         384              0      2003             0    98070  47.4177
12653        1490              0      1999             0    98065  47.5261
14423         844              0      1913             0    98001  47.2781
9773         2460              0      1990             0    98031  47.4095
...           ...            ...       ...           ...      ...      ...
4235         2830            850      1908             0    98102  47.6374
13314        2500           2090      2008             0    98004  47.5861
19254        1860           1060      1913             0    98105  47.6635
15161        3010            600      1958             0    98006  47.5705
15870        1040            580      1947             0    98103  47.6878

          long  sqft_living15  sqft_lot15
4868  -122.261           2170        6000
18379 -122.491           1920      224341
12653 -121.826           1500        4675
14423 -122.250           1380        9600
9773  -122.168           2520        8050
...        ...            ...         ...
4235  -122.324           1960        2450
13314 -122.113           2730       10400
19254 -122.320           1810        3745
15161 -122.175           2040       11914
15870 -122.331           1330        4700

[518 rows x 21 columns]
               id             date      price  bedrooms  bathrooms  \
4868   6896300380  20141002T000000   228000.0       0.0       1.00
18379  1222029077  20141029T000000   265000.0       0.0       0.75
12653  7849202299  20150218T000000   320000.0       0.0       2.50
14423  9543000205  20150413T000000   139950.0       0.0       2.50
9773   3374500520  20150429T000000   355000.0       0.0       0.00
...           ...              ...        ...       ...        ...
16844  8823900290  20150317T000000  1400000.0       9.0       4.00
4235   2902200015  20150106T000000   700000.0       9.0       3.00
13314   627300145  20140814T000000  1148000.0      10.0       5.25
15161  5566100170  20141029T000000   650000.0      10.0       2.00
19254  8812401450  20141229T000000   660000.0      10.0       3.00

       sqft_living  sqft_lot  floors  waterfront  view  ...  grade  \
4868         390.0    5900.0     1.0           0     0  ...      4
18379        384.0  213444.0     1.0           0     0  ...      4
12653       1490.0    7111.0     2.0           0     0  ...      7
14423        844.0    4269.0     1.0           0     0  ...      7
9773        2460.0    8049.0     2.0           0     0  ...      8
...            ...       ...     ...         ...   ...  ...    ...
16844       4620.0    5508.0     2.5           0     0  ...     11
4235        3680.0    4400.0     2.0           0     0  ...      7
13314       4590.0   10920.0     1.0           0     1  ...      9
15161       3610.0   11914.0     2.0           0     0  ...      7
19254       2920.0    3745.0     2.0           0     0  ...      7

       sqft_above  sqft_basement  yr_built  yr_renovated  zipcode      lat  \
4868          390              0      1953             0    98118  47.5260
18379         384              0      2003             0    98070  47.4177
12653        1490              0      1999             0    98065  47.5261
14423         844              0      1913             0    98001  47.2781
9773         2460              0      1990             0    98031  47.4095
...           ...            ...       ...           ...      ...      ...
16844        3870            750      1915             0    98105  47.6684
4235         2830            850      1908             0    98102  47.6374
13314        2500           2090      2008             0    98004  47.5861
15161        3010            600      1958             0    98006  47.5705
19254        1860           1060      1913             0    98105  47.6635

          long  sqft_living15  sqft_lot15
4868  -122.261           2170        6000
18379 -122.491           1920      224341
12653 -121.826           1500        4675
14423 -122.250           1380        9600
9773  -122.168           2520        8050
...        ...            ...         ...
16844 -122.309           2710        4320
4235  -122.324           1960        2450
13314 -122.113           2730       10400
15161 -122.175           2040       11914
19254 -122.320           1810        3745

[517 rows x 21 columns]
bedrooms
3.0     10275
4.0      6665
2.0      2617
```

```
5.0      1539
6.0       263
1.0       189
7.0        34
8.0        12
9.0         5
10.0        3
Name: count, dtype: int64
```

```
C:\Users\19noa\AppData\Local\Temp\ipykernel_52056\2335008484.py:17: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inp
lace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inp
lace on the original object.


  cleaned_data['bedrooms'].replace(33, 3, inplace=True)
```

Out[ ]:

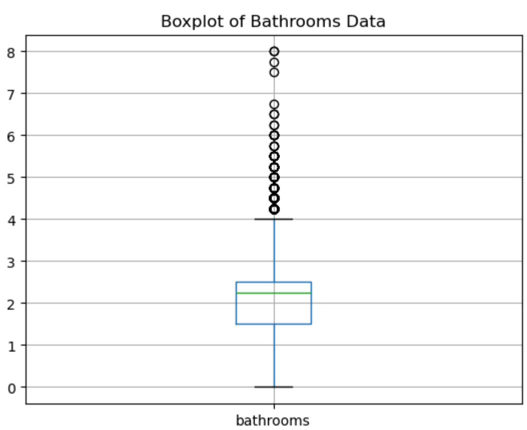
	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat
860	1723049033	20140620T000000	245000.0	1.0	0.75	380.0	15000.0000	1.0	0	0	...	5	380	0	1963	0	98168	47.4810
14366	3333002450	20140708T000000	165000.0	1.0	1.00	850.0	8050.0000	1.0	0	0	...	6	850	0	1906	0	98118	47.5427
18052	1352300580	20141114T000000	247000.0	1.0	1.00	460.0	10225.6875	1.0	0	0	...	4	460	0	1937	0	98055	47.4868
21240	7174800094	20150420T000000	525000.0	1.0	1.50	1030.0	5923.0000	1.0	0	0	...	8	1030	0	1940	0	98105	47.6653
18059	1773101530	20141218T000000	275000.0	1.0	1.00	520.0	4800.0000	1.0	0	0	...	5	520	0	1930	0	98106	47.5533
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6079	9822700190	20140808T000000	1280000.0	9.0	4.50	3650.0	5000.0000	2.0	0	0	...	8	2530	1120	1915	2010	98105	47.6604
18443	8823901445	20150313T000000	934000.0	9.0	1.75	2820.0	4480.0000	2.0	0	0	...	7	1880	940	1918	0	98105	47.6654
19254	8812401450	20141229T000000	660000.0	10.0	3.00	2920.0	3745.0000	2.0	0	0	...	7	1860	1060	1913	0	98105	47.6635
13314	627300145	20140814T000000	1148000.0	10.0	5.25	4590.0	10920.0000	1.0	0	1	...	9	2500	2090	2008	0	98004	47.5861
15161	5566100170	20141029T000000	650000.0	10.0	2.00	3610.0	11914.0000	2.0	0	0	...	7	3010	600	1958	0	98006	47.5705

21602 rows × 21 columns

Using the boxplot, there are approximately 252 rows with outliers within the dataset. The 0 bathrooms were removed from the dataset.

In [ ]:

```python
bathrooms_boxplot = cleaned_data.boxplot(column=['bathrooms'])

# plot title
plt.title ('Boxplot of Bathrooms Data')
plt.show()

#calculating 25% and 75% quartile
bath3,bath1 = np.percentile(cleaned_data['bathrooms'], [75,25])
bath_iqr = bath3-bath1
#calculating lower and upper bound
bath_lower = bath1 - 1.5*bath_iqr
bath_upper = bath3 + 1.5*bath_iqr
print("Any value for bathrooms less than",bath_lower, "bathrooms is an outlier.", "\nAny values greater than", bath_upper, "bathrooms is an outlier.")
print(cleaned_data[(cleaned_data.bathrooms < bath_lower) | (cleaned_data.bathrooms > bath_upper)].sort_values('bathrooms', ascending = True))

#removing all 0 bathrooms
cleaned_data = cleaned_data.loc[~(cleaned_data['bathrooms'] == 0)]
print(cleaned_data['bathrooms'].value_counts())
cleaned_data.sort_values('bathrooms', ascending = True)
```



```
Any value for bathrooms less than 0.0 bathrooms is an outlier.
Any values greater than 4.0 bathrooms is an outlier.
               id             date      price  bedrooms  bathrooms  \
11685  1126069045  20140620T000000  1135000.0       6.0       4.25   
7710    644200040  20140515T000000  1000000.0       5.0       4.25   
15022  2210500010  20140930T000000  2450000.0       7.0       4.25   
7280    922059169  20141201T000000   800000.0       6.0       4.25   
7236   1245002391  20141022T000000  1400000.0       5.0       4.25   
...           ...              ...        ...       ...        ...   
8092   1924059029  20140617T000000  4668000.0       5.0       6.75   
8546    424049043  20140811T000000   450000.0       9.0       7.50   
9254   9208900037  20140919T000000  6885000.0       6.0       7.75   
12777  1225069038  20140505T000000  2280000.0       3.0       8.00   
7252   6762700020  20141013T000000  7700000.0       6.0       8.00   

        sqft_living  sqft_lot  floors  waterfront  view  ...  grade  \
11685   6900.000000  244716.0     2.0           0     0  ...      9   
7710    3920.000000   16258.0     2.0           0     0  ...      9   
15022   4670.000000   23115.0     2.0           0     1  ...     11   
7280    5480.000000  189050.0     2.0           0     0  ...     10   
7236    4230.000000    6907.0     2.0           0     0  ...     10   
...             ...       ...     ...         ...   ...  ...    ...   
8092    9640.000000   13068.0     1.0           1     1  ...     12   
8546    4050.000000    6504.0     2.0           0     0  ...      7   
9254    3685.520833   31374.0     2.0           0     1  ...     13   
12777   2584.620053  307752.0     3.0           0     1  ...     12   
7252   12050.000000   27600.0     2.5           0     1  ...     13   

       sqft_above  sqft_basement  yr_built  yr_renovated  zipcode      lat  \
11685        4820           2080      2002             0    98077  47.7506   
7710         2900           1020      1990             0    98004  47.5871   
15022        4670              0      1992             0    98039  47.6183   
7280         5140            340      1991             0    98031  47.4120   
7236         3450            780      2008             0    98033  47.6866   
...           ...            ...       ...           ...      ...      ...   
8092         4820           4820      1983          2009    98040  47.5570   
8546         4050              0      1996             0    98144  47.5923   
9254         8860           1030      2001             0    98039  47.6305   
12777        9410           4130      1999             0    98053  47.6675   
7252         8570           3480      1910          1987    98102  47.6298   

          long  sqft_living15  sqft_lot15  
11685 -122.012           4170      266587  
7710  -122.192           2540       12131  
15022 -122.227           3240       13912  
7280  -122.168           2470       10429  
7236  -122.205           2650        8076  
...        ...            ...         ...  
8092  -122.210           3270       10454  
8546  -122.301           1448        3866  
9254  -122.240           4540       42730  
12777 -121.986           4850      217800  
7252  -122.323           3940        8800  

[252 rows x 21 columns]
bathrooms
2.50    5713
1.00    4077
1.75    2942
2.25    1964
2.00    1828
1.50    1383
2.75    1121
3.00     713
3.50     690
3.25     558
3.75     150
4.00     128
4.50      94
4.25      77
0.75      67
4.75      23
5.00      21
5.25      10
5.50      10
1.25       8
6.00       5
0.50       4
5.75       3
8.00       2
6.25       2
6.50       2
6.75       1
7.50       1
7.75       1
Name: count, dtype: int64
```

Out[ ]:

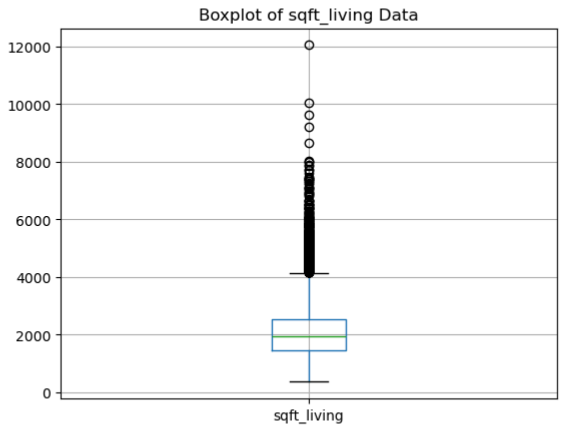
	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat
10424	7129800036	20150114T000000	109000.0	2.0	0.50	580.000000	6900.0	1.0	0	0	...	5	580	0	1941	0	98118	47.5135
12041	2991000160	20141212T000000	312500.0	4.0	0.50	2300.000000	5570.0	2.0	0	0	...	8	2300	0	1996	0	98092	47.3285
11674	7987400316	20140814T000000	255000.0	1.0	0.50	880.000000	1642.0	1.0	0	0	...	6	500	380	1910	0	98126	47.5732
2261	3971701455	20141003T000000	273000.0	2.0	0.50	1180.000000	7750.0	1.0	0	0	...	6	590	590	1945	0	98155	47.7690
21612	1523300157	20141015T000000	325000.0	2.0	0.75	1020.000000	1076.0	2.0	0	0	...	7	1020	0	2008	0	98144	47.5941
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8092	1924059029	20140617T000000	4668000.0	5.0	6.75	9640.000000	13068.0	1.0	1	1	...	12	4820	4820	1983	2009	98040	47.5570
8546	424049043	20140811T000000	450000.0	9.0	7.50	4050.000000	6504.0	2.0	0	0	...	7	4050	0	1996	0	98144	47.5923
9254	9208900037	20140919T000000	6885000.0	6.0	7.75	3685.520833	31374.0	2.0	0	1	...	13	8860	1030	2001	0	98039	47.6305
12777	1225069038	20140505T000000	2280000.0	3.0	8.00	2584.620053	307752.0	3.0	0	1	...	12	9410	4130	1999	0	98053	47.6675
7252	6762700020	20141013T000000	7700000.0	6.0	8.00	12050.000000	27600.0	2.5	0	1	...	13	8570	3480	1910	1987	98102	47.6298

21598 rows × 21 columns

Using the boxplot, there are approximately 601 rows with outliers within the sqft_living column and will be kept in the dataset.

In [ ]:

```
sqft_living_boxplot = cleaned_data.boxplot(column=['sqft_living'])

# plot title
plt.title ('Boxplot of sqft_living Data')
plt.show()

#calculating 25% and 75% quartile
sqft3,sqft1 = np.percentile(cleaned_data['sqft_living'], [75,25])
sqft_iqr = sqft3-sqft1
#calculating lower and upper bound
sqft_lower = cleaned_data['sqft_living'].min()
sqft_upper = sqft3 + 1.5*sqft_iqr
print("Any values less than",sqft_lower, "square foot is an outlier.", "\nAny values greater than",sqft_upper,"square foot is an outlier.")
cleaned_data[(cleaned_data.sqft_living > sqft_upper)].sort_values('sqft_living', ascending = False)
```




```
Any values less than 370.0 square foot is an outlier.
Any values greater than 4150.0 square foot is an outlier.
```

Out[ ]:

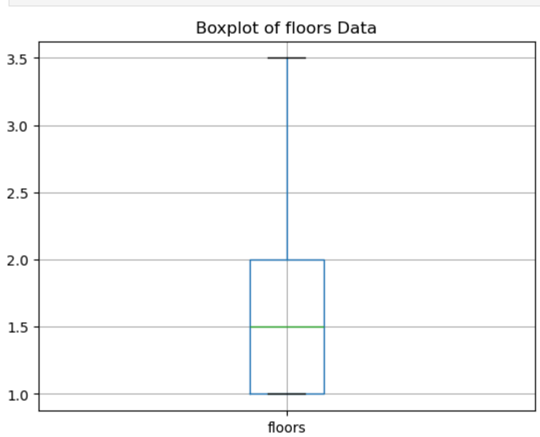
	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	l
7252	6762700020	20141013T000000	7700000.0	6.0	8.00	12050.0	27600.000000	2.5	0	1	...	13	8570	3480	1910	1987	98102	47.62
3914	9808700762	20140611T000000	7062500.0	5.0	4.50	10040.0	37325.000000	2.0	1	1	...	11	7680	2360	1940	2001	98004	47.65
8092	1924059029	20140617T000000	4668000.0	5.0	6.75	9640.0	13068.000000	1.0	1	1	...	12	4820	4820	1983	2009	98040	47.55
4411	2470100110	20140804T000000	5570000.0	5.0	5.75	9200.0	16813.145833	2.0	0	0	...	13	6200	3000	2001	0	98039	47.62
14556	2303900035	20140611T000000	2888000.0	5.0	6.25	8670.0	64033.000000	2.0	0	1	...	13	6120	2550	1965	2003	98177	47.72
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
11947	3303980140	20150402T000000	1150000.0	4.0	3.00	4160.0	13170.000000	2.0	0	0	...	11	3040	1120	2001	0	98059	47.51
11233	2655500241	20140814T000000	1699000.0	3.0	3.25	4160.0	35153.000000	3.0	0	1	...	12	3690	470	2001	0	98040	47.57
5961	8155800050	20150422T000000	1110000.0	3.0	4.00	4160.0	31796.000000	2.0	0	0	...	11	4160	0	1989	0	98053	47.66
19681	1266200140	20150506T000000	1850000.0	4.0	3.25	4160.0	10335.000000	2.0	0	0	...	10	4160	0	2014	0	98004	47.62
13448	2426059124	20141216T000000	1045000.0	4.0	3.25	4160.0	47480.000000	2.0	0	0	...	10	4160	0	1995	0	98072	47.72

601 rows × 21 columns

Using the boxplot, there are no outliers in the floors column.

In [ ]:

```
floors_boxplot = cleaned_data.boxplot(column=['floors'])

# plot title
plt.title ('Boxplot of floors Data')
plt.show()

#calculating 25% and 75% quartile
floors3,floors1 = np.percentile(cleaned_data['floors'], [75,25])
floors_iqr = floors3-floors1
#calculating lower and upper bound
floors_lower = cleaned_data['floors'].min()
floors_upper = floors3 + 1.5*floors_iqr
print("Any value less than",floors_lower, "floors is an outlier.", "\nAny values greater than" ,floors_upper, "floors is an outlier.")
cleaned_data[(cleaned_data.floors > floors_upper)].sort_values('floors', ascending = False)
```



```
Any value less than 1.0 floors is an outlier.
Any values greater than 3.5 floors is an outlier.
```

Out[ ]:

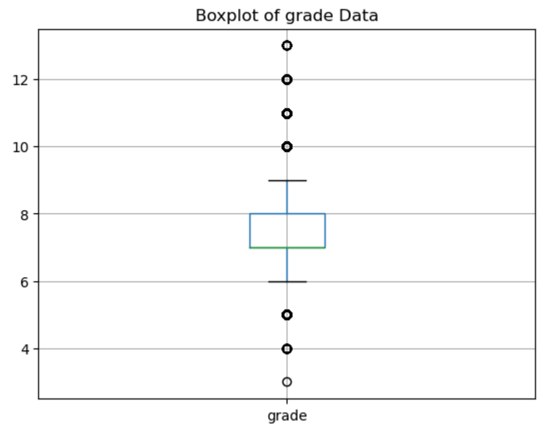
id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15

0 rows × 21 columns

Using the boxplot, there are approximately 1905 rows with outliers within the grade column and will be kept from the dataset.

In [ ]:

```python
grade_boxplot = cleaned_data.boxplot(column=['grade'])

# plot title
plt.title ('Boxplot of grade Data')
plt.show()

#calculating 25% and 75% quartile
grade3,grade1 = np.percentile(cleaned_data['grade'], [75,25])
grade_iqr = grade3-grade1
#calculating lower and upper bound
grade_lower = grade1 - 1.5*grade_iqr
grade_upper = grade3 + 1.5*grade_iqr
print("Any values less than",grade_lower, "for grade is an outlier.", "\nAny values greater than",grade_upper, "for grade is an outlier.")
cleaned_data[(cleaned_data.grade < grade_lower) | (cleaned_data.grade > grade_upper)].sort_values('grade', ascending = False)
```




```
Any values less than 5.5 for grade is an outlier.
Any values greater than 9.5 for grade is an outlier.
```

Out[ ]:

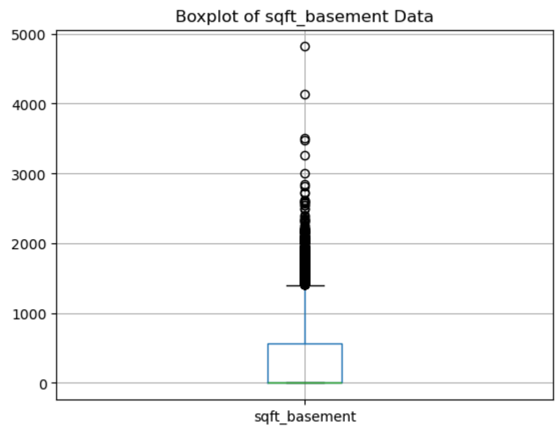
	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	
6041	1725059316	20141120T000000	2385000.0	4.0	4.00	2382.779376	13296.000000	2.0	0	1	...	13	4900	1430	2000	0	98033	47
5451	7237501190	20141010T000000	1780000.0	4.0	3.25	4890.000000	13402.000000	2.0	0	0	...	13	4890	0	2004	0	98059	47
7252	6762700020	20141013T000000	7700000.0	6.0	8.00	12050.000000	27600.000000	2.5	0	1	...	13	8570	3480	1910	1987	98102	47
4411	2470100110	20140804T000000	5570000.0	5.0	5.75	9200.000000	16813.145833	2.0	0	0	...	13	6200	3000	2001	0	98039	47
19017	2303900100	20140911T000000	3800000.0	3.0	4.25	5510.000000	35000.000000	2.0	0	1	...	13	4910	600	1997	0	98177	47
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
9794	3760500240	20150512T000000	435000.0	2.0	0.75	750.000000	16321.000000	1.0	0	1	...	4	750	0	1936	0	98034	47
465	8658300340	20140523T000000	80000.0	1.0	0.75	430.000000	5050.000000	1.0	0	0	...	4	430	0	1912	0	98014	47
16340	6146600170	20140703T000000	100000.0	2.0	0.75	660.000000	5240.000000	1.0	0	0	...	4	660	0	1912	0	98032	47
7973	3122069029	20140619T000000	120000.0	2.0	1.00	990.000000	39964.000000	1.0	0	0	...	4	990	0	1945	0	98042	47
3223	2420069251	20150225T000000	262000.0	1.0	0.75	1841.699115	12981.000000	1.0	0	0	...	3	520	0	1920	0	98022	47

1905 rows × 21 columns

Using the boxplot, there are approximately 496 rows with outliers within the sqft_basement column and will be kept in the dataset.

In [ ]:

```python
sqft_base_boxplot = cleaned_data.boxplot(column=['sqft_basement'])

# plot title
plt.title ('Boxplot of sqft_basement Data')
plt.show()

#calculating 25% and 75% quartile
sqft_base3,sqft_base1 = np.percentile(cleaned_data['sqft_basement'], [75,25])
sqft_base_iqr = sqft_base3-sqft_base1
#calculating lower and upper bound
sqft_base_lower = cleaned_data['sqft_basement'].min()
sqft_base_upper = sqft_base3 + 1.5*sqft_base_iqr
print("Any values less than", sqft_base_lower, "square foot is an outlier", "\nAny values greater than", sqft_base_upper, "square foot is an outlier.")
cleaned_data[(cleaned_data.sqft_basement < sqft_base_lower) | (cleaned_data.sqft_basement > sqft_base_upper)].sort_values('sqft_basement', ascending = False)
```




```
Any values less than 0 square foot is an outlier
Any values greater than 1400.0 square foot is an outlier.
```

Out[ ]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat
8092	1924059029	20140617T000000	4668000.0	5.0	6.75	9640.000000	13068.0	1.0	1	1	...	12	4820	4820	1983	2009	98040	47.5570
12777	1225069038	20140505T000000	2280000.0	3.0	8.00	2584.620053	307752.0	3.0	0	1	...	12	9410	4130	1999	0	98053	47.6675
15482	624069108	20140812T000000	3200000.0	4.0	3.25	7000.000000	28206.0	1.0	1	1	...	12	3500	3500	1991	0	98075	47.5928
7252	6762700020	20141013T000000	7700000.0	6.0	8.00	12050.000000	27600.0	2.5	0	1	...	13	8570	3480	1910	1987	98102	47.6298
10085	7767000060	20140912T000000	1900000.0	5.0	4.25	6510.000000	16471.0	2.0	0	1	...	11	3250	3260	1980	0	98040	47.5758
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2209	269000240	20141030T000000	1050000.0	5.0	2.25	2168.729642	7680.0	1.0	0	1	...	8	1550	1410	1958	0	98199	47.6456
4827	7366100080	20140731T000000	318000.0	5.0	2.50	2820.000000	9956.0	1.0	0	0	...	7	1410	1410	1967	0	98168	47.4715
8976	1126049095	20140926T000000	450000.0	3.0	2.50	2820.000000	10208.0	1.0	0	1	...	8	1410	1410	1954	0	98028	47.7609
4336	7738500475	20141212T000000	485000.0	3.0	3.25	2820.000000	6611.0	1.0	0	0	...	7	1410	1410	1958	0	98155	47.7473
1539	3425059222	20141124T000000	1300000.0	6.0	3.50	2640.696203	32670.0	2.0	0	0	...	10	5153	1410	2002	0	98005	47.6078

496 rows × 21 columns

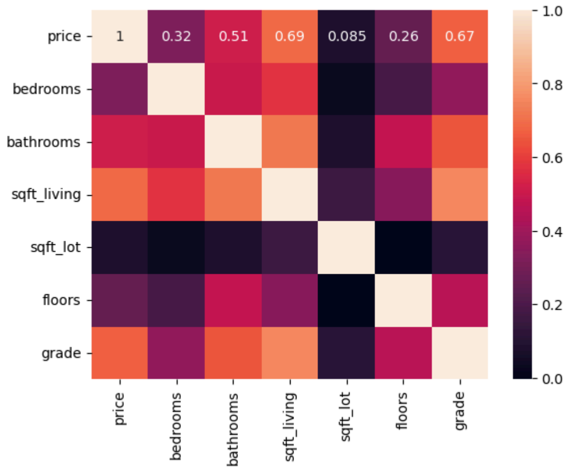
Correlation Matrix

This correlation matrix below uses the correlation function to show the correlation between all the variables. The heat map displays higher correlated items.

In [ ]:

```python
import seaborn as sb
columns_to_plot = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'grade']

chosen_data = cleaned_data[columns_to_plot]

correlation = chosen_data.corr()
corr_plot = sb.heatmap(correlation, annot=True)
print('Correlation Matrix: ')
print(correlation)
```

```
Correlation Matrix: 
                price  bedrooms  bathrooms  sqft_living  sqft_lot    floors  \
price        1.000000  0.315289   0.511176     0.688973  0.084613  0.256815   
bedrooms     0.315289  1.000000   0.499751     0.572469  0.029216  0.180476   
bathrooms    0.511176  0.499751   1.000000     0.717912  0.085412  0.480464   
sqft_living  0.688973  0.572469   0.717912     1.000000  0.163590  0.346291   
sqft_lot     0.084613  0.029216   0.085412     0.163590  1.000000 -0.004534   
floors       0.256815  0.180476   0.480464     0.346291 -0.004534  1.000000   
grade        0.667921  0.362937   0.646311     0.751033  0.109960  0.458806   

                grade  
price        0.667921  
bedrooms     0.362937  
bathrooms    0.646311  
sqft_living  0.751033  
sqft_lot     0.109960  
floors       0.458806  
grade        1.000000  
```



Identifying the Independent and Dependent Variables

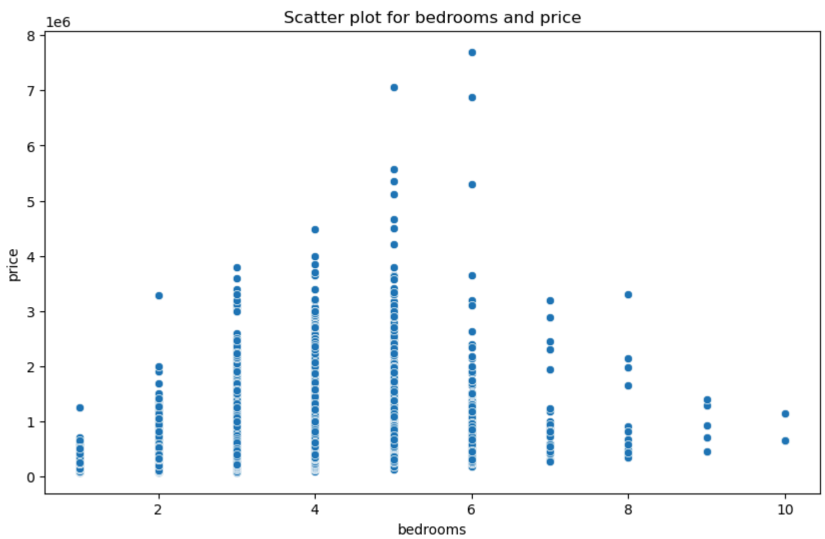
The independent variables include bedrooms, bathrooms, sqft_living, sqft_lot, floors, waterfront, view, and grade.

The dependent variable is price.

Exploratory Analysis via Visualizations

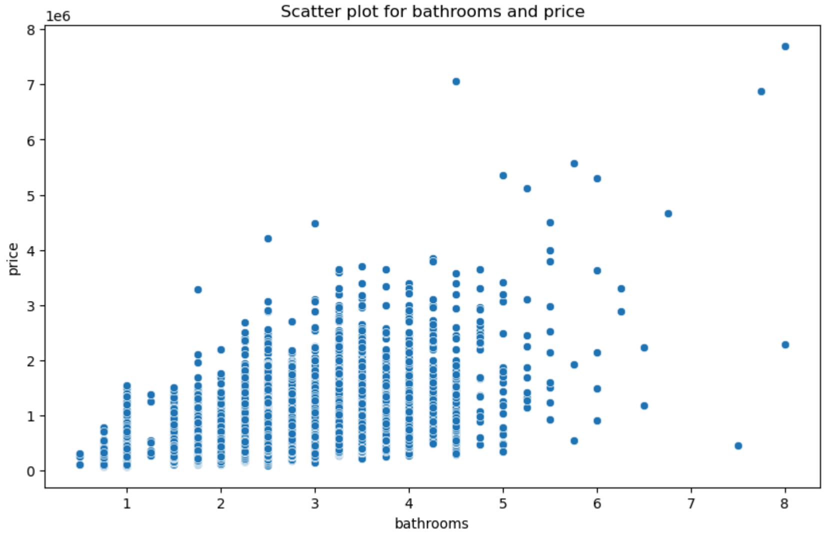
The below scatter plot displays the relationship between price and the number of bedrooms

In [ ]:
```
#Price and Bedrooms
plt.figure(figsize=(10,6))
sns.scatterplot(x = 'bedrooms', y = 'price', data=cleaned_data)
plt.title('Scatter plot for bedrooms and price')
plt.xlabel('bedrooms')
plt.ylabel('price')

plt.show()
```



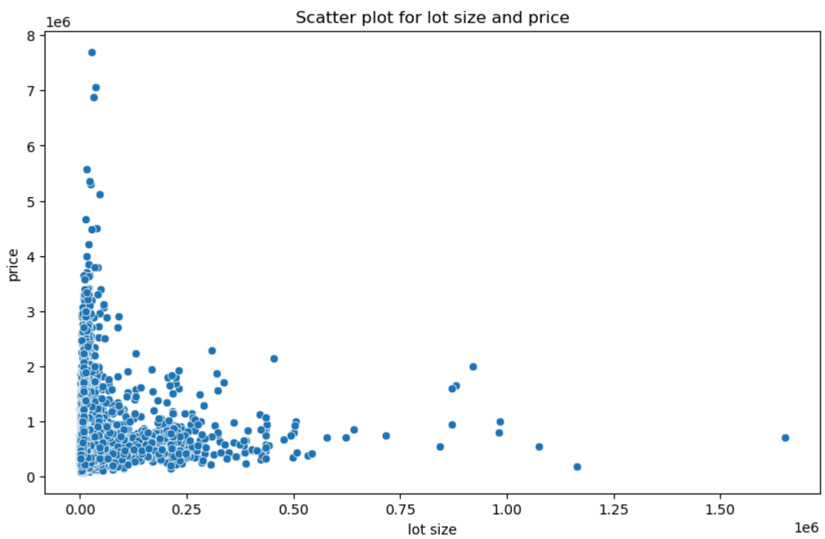
This scatter plot shows Price and Bathrooms

In [ ]:
```
#Scatter for price and bathrooms

plt.figure(figsize=(10,6))
sns.scatterplot(x = 'bathrooms', y = 'price', data=cleaned_data)
plt.title('Scatter plot for bathrooms and price')
plt.xlabel('bathrooms')
plt.ylabel('price')

plt.show()
```



This plot shows how the price and lot size are related. Oddly they don't seem to be positively correlated.

In [ ]:
```python
#Scatter for price and lot size
plt.figure(figsize=(10,6))
sns.scatterplot(x = 'sqft_lot', y = 'price', data=cleaned_data)
plt.title('Scatter plot for lot size and price')
plt.xlabel('lot size')
plt.ylabel('price')

plt.show()
```

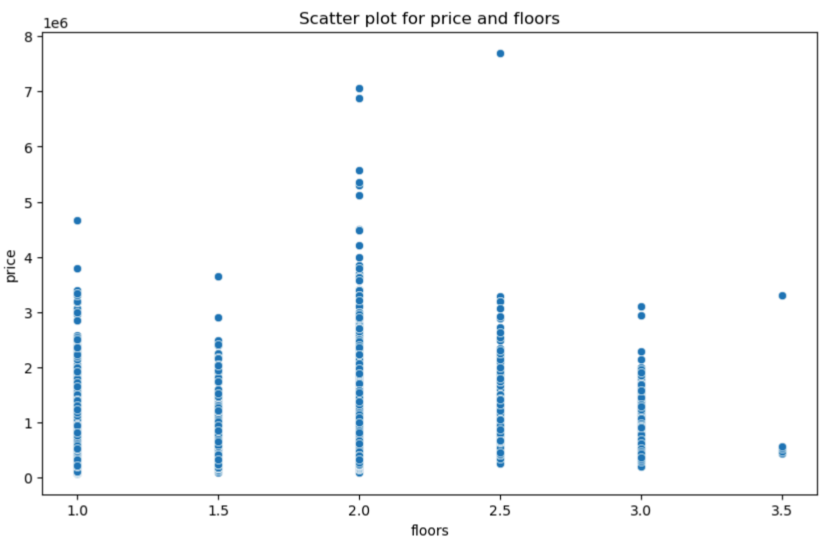


This plot shows the relationship between price and house square footage. This clearly shows a positive correlation between the square feet of the house and the price.

In [ ]:
```python
#Scatter for price and square foot living
plt.figure(figsize=(10,6))
sns.scatterplot(x = 'sqft_living', y = 'price', data=cleaned_data)
plt.title('Scatter plot for square foot living and price')
plt.xlabel('square foot living')
plt.ylabel('price')

plt.show()
```

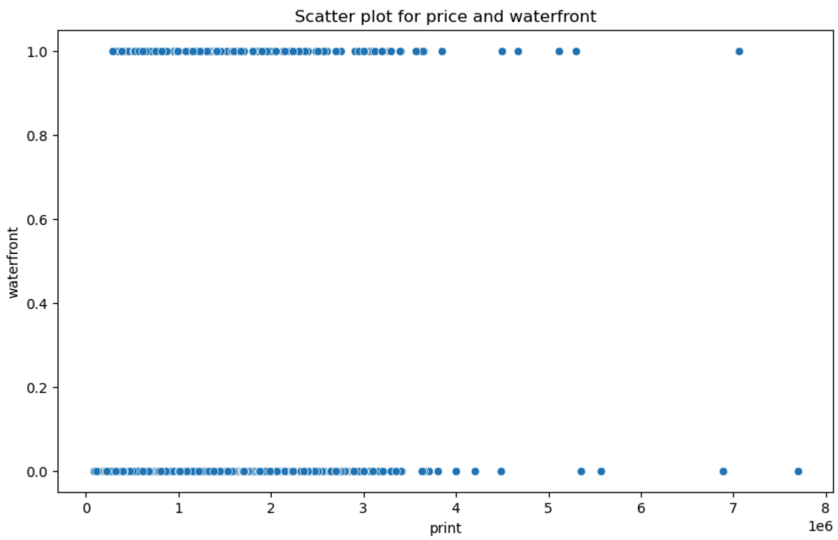
This scatter plot shows how floors and price are related. Strangely it looks like just because there are more floors the price doesn't necessarily increase. We see the majority of the higher priced houses are in the 2 floor category.

In [ ]:
```
#Price and Floors EA scatter
plt.figure(figsize=(10,6))
sns.scatterplot(x = 'floors', y = 'price', data=cleaned_data)
plt.title('Scatter plot for price and floors')
plt.xlabel('floors')
plt.ylabel('price')

plt.show()
```



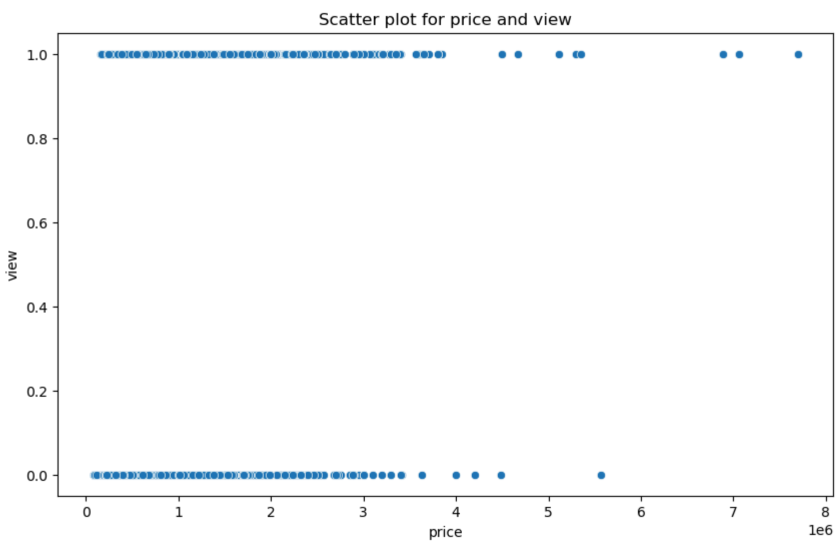
Here is the relationship between price and whether the house is waterfront or not and again these results were surprising. It doesn't show that being waterfront increases the price of the house.

In [ ]:
```
#Scatter for price and waterfront
plt.figure(figsize=(10,6))
sns.scatterplot(x = 'price', y = 'waterfront', data=cleaned_data)
plt.title('Scatter plot for price and waterfront')
plt.xlabel('print')
plt.ylabel('waterfront')

plt.show()
```



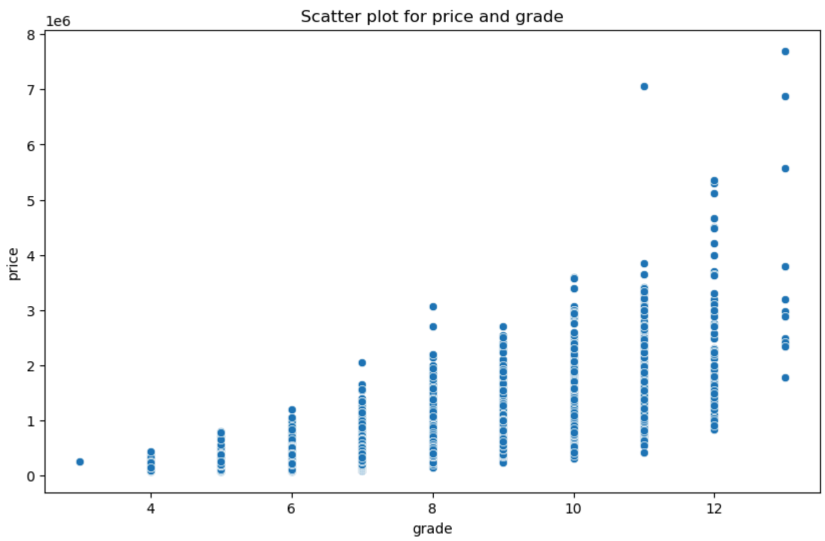
This shows price and view's relationship and it does seem that having a good view increases the value which is what we would probably assume.

In [ ]:
```
#Scatter for price and view
plt.figure(figsize=(10,6))
sns.scatterplot(x = 'price', y = 'view', data=cleaned_data)
plt.title('Scatter plot for price and view')
plt.xlabel('price')
plt.ylabel('view')

plt.show()
```



This below plot shows the relationship between price and grade which as expected appears to have a strong positive correlation

In [ ]:
```
#Scatter for price and grade
plt.figure(figsize=(10,6))
sns.scatterplot(x = 'grade', y = 'price', data=cleaned_data)
plt.title('Scatter plot for price and grade')
plt.xlabel('grade')
plt.ylabel('price')

plt.show()
```



Section 3: Data Analytics - test

Supervised vs Unsupervised Learning

This is the regression analysis for sqft lot and sqft living against the price. R-squared value shows about 48% of the variability in the price depends on sqft living and lot. F-statistic is 9793 so with the number of observations and degrees of freedom we can determine this regression is most likely not statistically significatn.

In [ ]:
```
#regression for sqft_lot and sqft_living against price

import statsmodels.api as sm
```

In [ ]:
```
X = cleaned_data[['sqft_lot', 'sqft_living']]
Y = cleaned_data['price']

X = sm. add_constant (X)

linear_model = sm. OLS (Y, X)

results = linear_model. fit()

print (results. summary ())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.475
Model:                            OLS   Adj. R-squared:                  0.475
Method:                 Least Squares   F-statistic:                     9789.
Date:                Sat, 13 Apr 2024   Prob (F-statistic):               0.00
Time:                        19:27:27   Log-Likelihood:            -3.0042e+05
No. Observations:               21598   AIC:                         6.009e+05
Df Residuals:                   21595   BIC:                         6.009e+05
Df Model:                           2
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const       -4.774e+04   4577.872    -10.429      0.000   -5.67e+04   -3.88e+04
sqft_lot       -0.2609      0.045     -5.779      0.000      -0.349      -0.172
sqft_living   284.4136      2.048    138.861      0.000     280.399     288.428
==============================================================================
Omnibus:                    16351.480   Durbin-Watson:                   1.975
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           914606.580
Skew:                           3.134   Prob(JB):                         0.00
Kurtosis:                      34.258   Cond. No.                     1.10e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.1e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

This is for the relation ship between bedrooms and price,R-squared is .099 and Fstatistic is 2387.

In [ ]:
```
#Regression Model for price and bedrooms

X = cleaned_data[['bedrooms']]
Y = cleaned_data['price']

X = sm.add_constant(X)

linear_model = sm.OLS(Y, X)

results = linear_model.fit()

print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.099
Model:                            OLS   Adj. R-squared:                  0.099
Method:                 Least Squares   F-statistic:                     2384.
Date:                Sat, 13 Apr 2024   Prob (F-statistic):               0.00
Time:                        19:27:27   Log-Likelihood:            -3.0626e+05
No. Observations:               21598   AIC:                         6.125e+05
Df Residuals:                   21596   BIC:                         6.125e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       1.007e+05   9309.337     10.814      0.000    8.24e+04    1.19e+05
bedrooms    1.308e+05   2678.819     48.824      0.000    1.26e+05    1.36e+05
==============================================================================
Omnibus:                    18899.226   Durbin-Watson:                   1.962
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1162326.629
Skew:                           3.936   Prob(JB):                         0.00
Kurtosis:                      38.066   Cond. No.                         14.7
==============================================================================

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

This shows the relationship between Bathrooms and Price and we see an R-squared of .261 and F statistic is 7642

In [ ]:

```python
#Supervised regression for bathroom and price

X = cleaned_data[['bathrooms']]
Y = cleaned_data['price']

X = sm.add_constant(X)

linear_model = sm.OLS(Y, X)

results = linear_model.fit()

print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.261
Model:                            OLS   Adj. R-squared:                  0.261
Method:                 Least Squares   F-statistic:                     7639.
Date:                Sat, 13 Apr 2024   Prob (F-statistic):               0.00
Time:                        19:27:27   Log-Likelihood:            -3.0412e+05
No. Observations:               21598   AIC:                         6.082e+05
Df Residuals:                   21596   BIC:                         6.083e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2.566e+04   6266.320      4.095      0.000    1.34e+04    3.79e+04
bathrooms   2.446e+05   2798.923     87.402      0.000    2.39e+05     2.5e+05
==============================================================================
Omnibus:                    17317.711   Durbin-Watson:                   1.956
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           879539.354
Skew:                           3.476   Prob(JB):                         0.00
Kurtosis:                      33.480   Cond. No.                         7.71
==============================================================================

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

Here is the regression for floors and prices R-squared is .066 and F-statistic is 1525.

In [ ]:

```python
#Supervised regression for price and floors

X = cleaned_data[['floors']]
Y = cleaned_data['price']

X = sm.add_constant(X)

linear_model = sm.OLS(Y, X)

results = linear_model.fit()

print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.066
Model:                            OLS   Adj. R-squared:                  0.066
Method:                 Least Squares   F-statistic:                     1525.
Date:                Sat, 13 Apr 2024   Prob (F-statistic):          2.27e-322
Time:                        19:27:27   Log-Likelihood:            -3.0665e+05
No. Observations:               21598   AIC:                         6.133e+05
Df Residuals:                   21596   BIC:                         6.133e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2.792e+05   7106.838     39.284      0.000    2.65e+05    2.93e+05
floors      1.747e+05   4473.784     39.050      0.000    1.66e+05    1.83e+05
==============================================================================
Omnibus:                    19369.210   Durbin-Watson:                   1.973
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1260508.231
Skew:                           4.079   Prob(JB):                         0.00
Kurtosis:                      39.526   Cond. No.                         6.37
==============================================================================

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

This is the regression model for if the house is on the waterfront and the price. The R-squared is .071 and the F-statistic is 1650

In [ ]:

```python
#Supervised Regression for price and waterfront
```

```
X = cleaned_data[['waterfront']]
Y = cleaned_data['price']

X = sm.add_constant(X)

linear_model = sm.OLS(Y, X)

results = linear_model.fit()

print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.071
Model:                            OLS   Adj. R-squared:                  0.071
Method:                 Least Squares   F-statistic:                     1650.
Date:                Sat, 13 Apr 2024   Prob (F-statistic):               0.00
Time:                        19:27:27   Log-Likelihood:            -3.0660e+05
No. Observations:               21598   AIC:                         6.132e+05
Df Residuals:                   21596   BIC:                         6.132e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       5.317e+05   2416.981    219.973      0.000    5.27e+05    5.36e+05
waterfront   1.13e+06   2.78e+04     40.623      0.000    1.08e+06    1.18e+06
==============================================================================
Omnibus:                    17744.222   Durbin-Watson:                   1.962
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           924924.672
Skew:                           3.607   Prob(JB):                         0.00
Kurtosis:                      34.237   Cond. No.                         11.6
==============================================================================

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

Below is the regression for if the house has a view against the price. The R-squared value is .129 and the Fstatistic is 3196.

In [ ]:
```
#Supervised regression for price and view

X = cleaned_data[['view']]
Y = cleaned_data['price']

X = sm.add_constant(X)

linear_model = sm.OLS(Y, X)

results = linear_model.fit()

print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.129
Model:                            OLS   Adj. R-squared:                  0.129
Method:                 Least Squares   F-statistic:                     3198.
Date:                Sat, 13 Apr 2024   Prob (F-statistic):               0.00
Time:                        19:27:27   Log-Likelihood:            -3.0590e+05
No. Observations:               21598   AIC:                         6.118e+05
Df Residuals:                   21596   BIC:                         6.118e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       4.967e+05   2455.206    202.297      0.000    4.92e+05    5.01e+05
view         4.43e+05   7832.889     56.551      0.000    4.28e+05    4.58e+05
==============================================================================
Omnibus:                    18360.975   Durbin-Watson:                   1.960
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1111436.469
Skew:                           3.756   Prob(JB):                         0.00
Kurtosis:                      37.331   Cond. No.                         3.40
==============================================================================

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

This below regression goes over the relationship between price and grade. The R-squared value for this regression is .446 and the Fstatistic is 1.74.

In [ ]:
```
#Supervised regression for price and grade

X = cleaned_data[['grade']]
Y = cleaned_data['price']

X = sm.add_constant(X)

linear_model = sm.OLS(Y, X)

results = linear_model.fit()

print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.446
Model:                            OLS   Adj. R-squared:                  0.446
Method:                 Least Squares   F-statistic:                 1.739e+04
Date:                Sat, 13 Apr 2024   Prob (F-statistic):               0.00
Time:                        19:27:27   Log-Likelihood:            -3.0101e+05
No. Observations:               21598   AIC:                         6.020e+05
Df Residuals:                   21596   BIC:                         6.020e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       -1.06e+06   1.23e+04    -86.368      0.000   -1.08e+06   -1.04e+06
grade        2.09e+05   1584.788    131.887      0.000    2.06e+05    2.12e+05
==============================================================================
Omnibus:                    19899.052   Durbin-Watson:                   1.968
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          2054020.764
Skew:                           4.086   Prob(JB):                         0.00
Kurtosis:                      50.071   Cond. No.                         52.0
==============================================================================

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

Combined Regression

Below is the summary of the regression plotted using the Ordinary Least Squares method. For this regression there there was a comparison performed for all of the independent variables we use against the price.

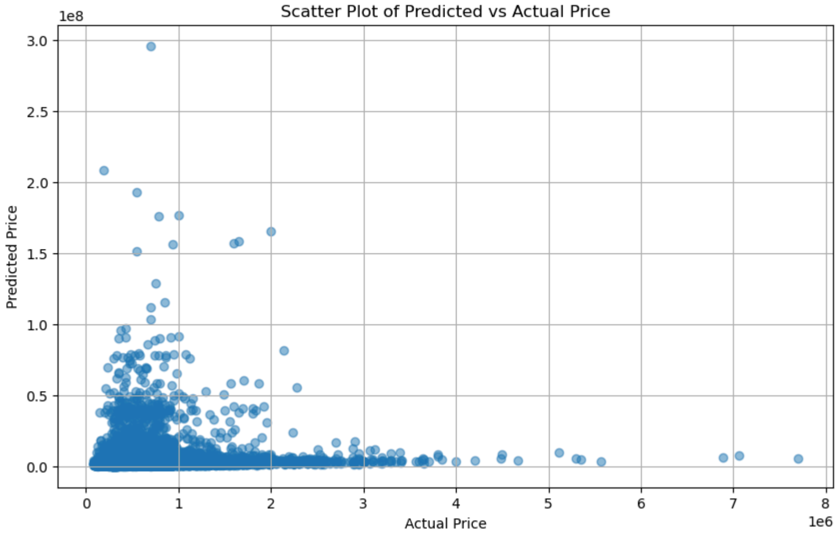
In [ ]:

```python
#collection regression
X = cleaned_data[['bedrooms', 'bathrooms', 'sqft_lot', 'sqft_living', 'floors', 'waterfront', 'view', 'grade']]
Y = cleaned_data['price']

X = sm.add_constant(X)

linear_model = sm.OLS(Y, X)

results = linear_model.fit()

print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.585
Model:                            OLS   Adj. R-squared:                  0.585
Method:                 Least Squares   F-statistic:                     3800.
Date:                Sat, 13 Apr 2024   Prob (F-statistic):               0.00
Time:                        19:27:27   Log-Likelihood:            -2.9790e+05
No. Observations:               21598   AIC:                         5.958e+05
Df Residuals:                   21589   BIC:                         5.959e+05
Df Model:                           8
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const       -5.409e+05    1.4e+04    -38.627      0.000   -5.68e+05   -5.13e+05
bedrooms    -2.469e+04   2297.994    -10.746      0.000   -2.92e+04   -2.02e+04
bathrooms    -387.2910   3318.709     -0.117      0.907   -6892.207    6117.625
sqft_lot       -0.3162      0.040     -7.825      0.000      -0.395      -0.237
sqft_living   179.0091      3.421     52.320      0.000     172.303     185.715
floors      -3.215e+04   3551.185     -9.054      0.000   -3.91e+04   -2.52e+04
waterfront   6.787e+05   1.93e+04     35.088      0.000    6.41e+05    7.17e+05
view         1.571e+05   5845.404     26.871      0.000    1.46e+05    1.69e+05
grade        1.077e+05   2248.253     47.900      0.000    1.03e+05    1.12e+05
==============================================================================
Omnibus:                    16760.556   Durbin-Watson:                   1.972
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1324799.148
Skew:                           3.139   Prob(JB):                         0.00
Kurtosis:                      40.851   Cond. No.                     5.23e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.23e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [ ]:

```python
independent_vars = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view', 'grade']
X = cleaned_data[independent_vars]

X = sm.add_constant(X)


predicted_prices = results.predict(X)


plt.figure(figsize=(10, 6))
plt.scatter(cleaned_data['price'], predicted_prices, alpha=0.5)
plt.title('Scatter Plot of Predicted vs Actual Price')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.grid(True)
plt.show()
```



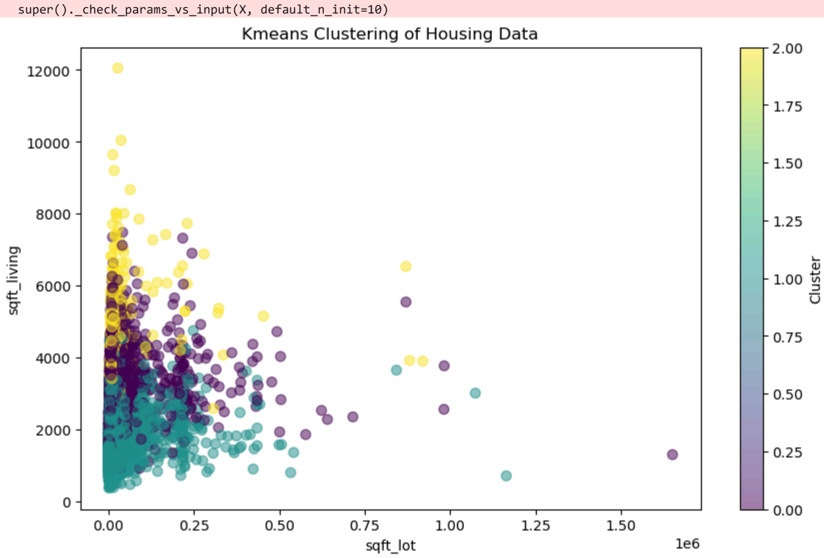
Unsupervised

Below we have applied K-means clustering to compare the independent variables and the dependent variables.

This first cluster compares the sqft_living and sqft_lot with the price. The clustering is fairly spread out for this indicating the clustering wasn't very effective. It is however aparent the majority of the clustering is 2000 to 4000 sqft_living and .175 sqft_lot.

```
In [ ]: #Price with sqft_lot and sqft_living

import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

X = cleaned_data[['price', 'sqft_lot', 'sqft_living']]

kmeans = KMeans(n_clusters=3)

kmeans.fit(X)

cluster_labels = kmeans.labels_

cleaned_data['cluster'] = cluster_labels

plt.figure(figsize=(10,6))

plt.scatter(cleaned_data['sqft_lot'], cleaned_data['sqft_living'], c=cleaned_data['cluster'], cmap='viridis', s=50, alpha=0.5)

plt.xlabel('sqft_lot')
plt.ylabel('sqft_living')
plt.title('Kmeans Clustering of Housing Data')

plt.colorbar(label='Cluster')

plt.show()
```

```
c:\Users\19noa\miniconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

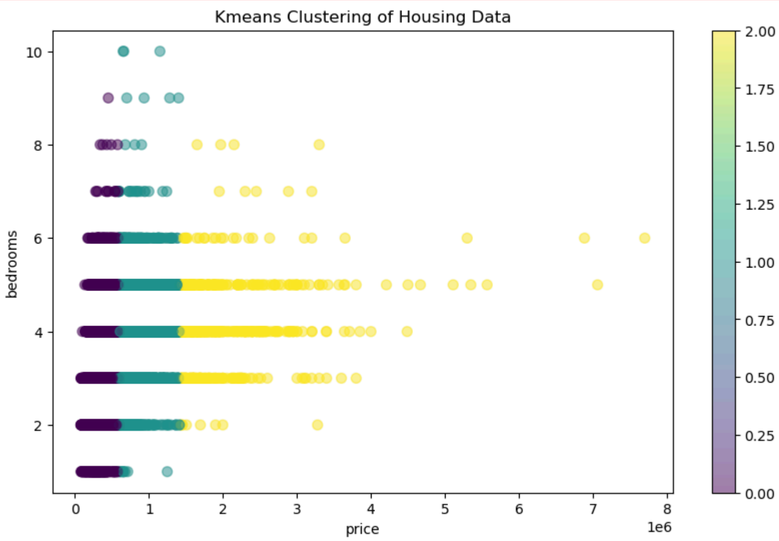


This below K-means cluster shows the relationship between price and number of bedrooms. The majority of the higher priced items and the majority of the datapoints are grouped in the middle and displays that just because there are more bedrooms, the price doesn't necesarily increase.

This cluster shows the relationship between bedrooms and price. This conflicts with the bathroom data where we don't see the number of bedrooms necessarily leading to a higher price but rather the clustered items in the 4 and 6 bedroom cluster show the highest prices and the most data as well.

```
In [ ]: #Price and bedrooms
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

X = cleaned_data[['price', 'bedrooms']]

kmeans = KMeans(n_clusters=3)

kmeans.fit(X)

cluster_labels = kmeans.labels_

cleaned_data['cluster'] = cluster_labels

plt.figure(figsize=(10,6))

plt.scatter(cleaned_data['price'], cleaned_data['bedrooms'], c=cleaned_data['cluster'], cmap='viridis', s=50, alpha=0.5)

plt.xlabel('price')
plt.ylabel('bedrooms')
plt.title('Kmeans Clustering of Housing Data')

plt.colorbar(label='Cluster')

plt.show()
```

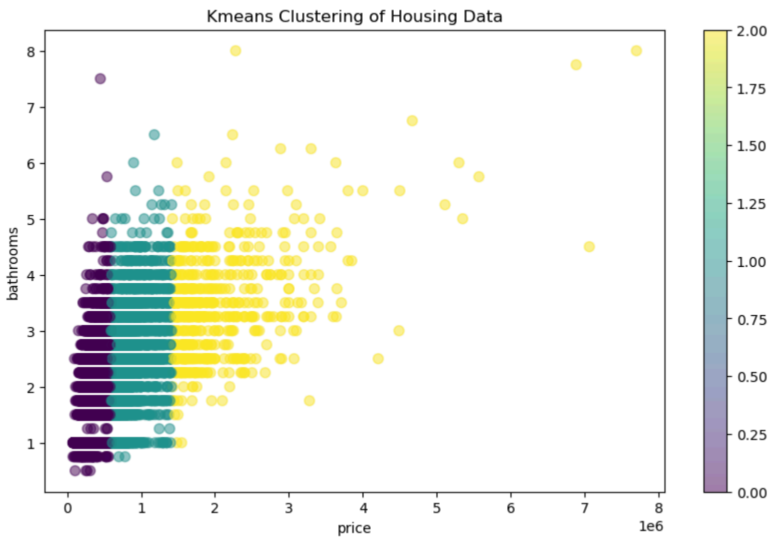
```
c:\Users\19noa\miniconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



This cluster shows the relationship between the price and bathrooms where the majority of the datapoints are clustered between 1 and 4.5 bathrooms with the price highest towards the upper side, indicating that there is a relationship between the number of bathrooms and the price.

```
In [ ]: #Price and bathrooms
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

X = cleaned_data[['price', 'bathrooms']]

kmeans = KMeans(n_clusters=3)

kmeans.fit(X)

cluster_labels = kmeans.labels_

cleaned_data['cluster'] = cluster_labels

plt.figure(figsize=(10,6))

plt.scatter(cleaned_data['price'], cleaned_data['bathrooms'], c=cleaned_data['cluster'], cmap='viridis', s=50, alpha=0.5)

plt.xlabel('price')
plt.ylabel('bathrooms')
plt.title('Kmeans Clustering of Housing Data')

plt.colorbar(label='Cluster')

plt.show()
```

```
c:\Users\19noa\miniconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



This below K-means cluster shows the relationship between price and bedrooms. Based off of the layout of the cluster the majority of the houses are 2 floor houses. These houses always appear to have the highest prices. The higher and lower ends had the most diverse spread.

In [ ]:
```
#price and floors
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

X = cleaned_data[['price', 'bedrooms']]

kmeans = KMeans(n_clusters=3)

kmeans.fit(X)

cluster_labels = kmeans.labels_

cleaned_data['cluster'] = cluster_labels

plt.figure(figsize=(10,6))

plt.scatter(cleaned_data['price'], cleaned_data['floors'], c=cleaned_data['cluster'], cmap='viridis', s=50, alpha=0.5)

plt.xlabel('price')
plt.ylabel('floors')
plt.title('Kmeans Clustering of Housing Data')

plt.colorbar(label='Cluster')

plt.show()
```

```
c:\Users\19noa\miniconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

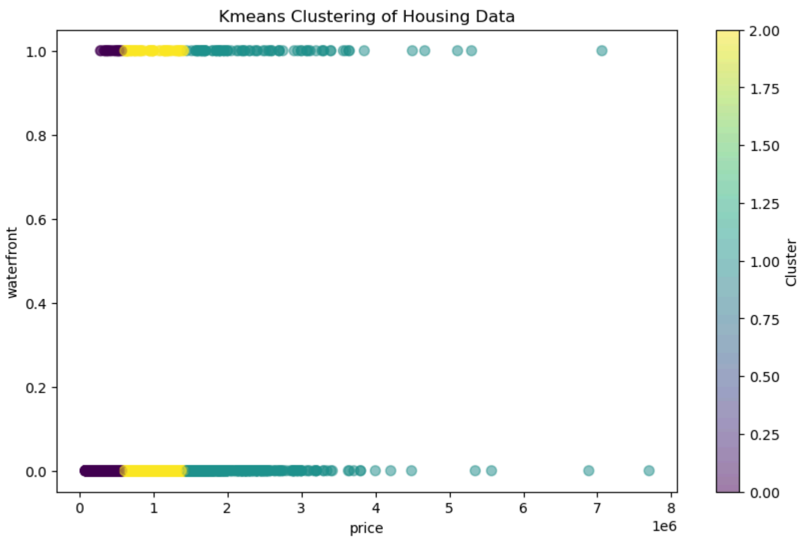
For this K-means cluster, features were clustered by price and waterfront, it doesn't seem that the clusters were

Below is a K-means cluster where the features were clustered by price and waterfront. Based off of the clustering model, it doesn't appear a relationship between waterfront and price is very strong as we see strong clustering in both waterfront properties and not.

In [ ]:
```
#price and waterfront

import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

```
X = cleaned_data[['price', 'waterfront']]

kmeans = KMeans(n_clusters=3)

kmeans.fit(X)

cluster_labels = kmeans.labels_

cleaned_data['cluster'] = cluster_labels

plt.figure(figsize=(10,6))

plt.scatter(cleaned_data['price'], cleaned_data['waterfront'], c=cleaned_data['cluster'], cmap='viridis', s=50, alpha=0.5)

plt.xlabel('price')
plt.ylabel('waterfront')
plt.title('Kmeans Clustering of Housing Data')

plt.colorbar(label='Cluster')

plt.show()
```

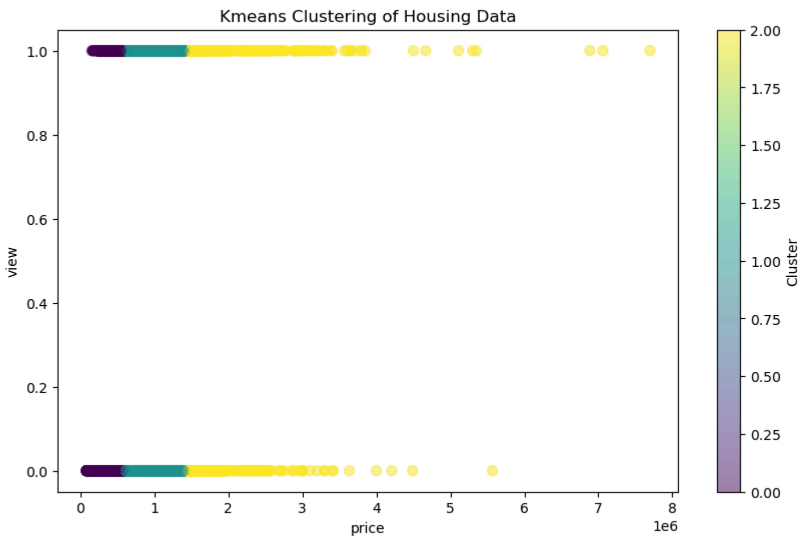
```
c:\Users\19noa\miniconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



Below is a K-means cluster that clusters the pricing and the view together. Based off this cluster it looks like there are futher outliers associated with a nice view and the associated price. Indicating that the price increases with a nice view.

In [ ]:
```
#Price and view
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

X = cleaned_data[['price', 'view']]

kmeans = KMeans(n_clusters=3)

kmeans.fit(X)

cluster_labels = kmeans.labels_

cleaned_data['cluster'] = cluster_labels

plt.figure(figsize=(10,6))

plt.scatter(cleaned_data['price'], cleaned_data['view'], c=cleaned_data['cluster'], cmap='viridis', s=50, alpha=0.5)

plt.xlabel('price')
plt.ylabel('view')
plt.title('Kmeans Clustering of Housing Data')

plt.colorbar(label='Cluster')

plt.show()
```

```
c:\Users\19noa\miniconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



This is a k-means cluster that shows the relationship between price and grade. We can see lots of the clustering in yellow are all high price and also high grade.

In [ ]:
```
#price and grade
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

X = cleaned_data[['price', 'grade']]

kmeans = KMeans(n_clusters=3)

kmeans.fit(X)

cluster_labels = kmeans.labels_

cleaned_data['cluster'] = cluster_labels

plt.figure(figsize=(10,6))

plt.scatter(cleaned_data['price'], cleaned_data['grade'], c=cleaned_data['cluster'], cmap='viridis', s=50, alpha=0.5)

plt.xlabel('price')
plt.ylabel('grade')
plt.title('Kmeans Clustering of Housing Data')

plt.colorbar(label='Cluster')

plt.show()
```

```
c:\Users\19noa\miniconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

All variables used for analysis: price, bedroom, bathroom, sqft_living, sqft_lot, floors, waterfront, view, grade

Train, Test, and Provide Accuracy and Evaluation Metrics

Here we train, test and provide the accuracy and evaluation metrics. The results display the R-Squared at .617 this means that the model built is responsible for 62% of variability in the house prices based on the variables included. The Mean Squared Error means that the squared difference between the predicted house prices and the actual house prices is approximately 236398.63 and for the Mean Absolute Error, on average the absolute difference between the predicted house prices and the actual house prices is approximately 154430.02.

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

X = cleaned_data[['bedrooms', 'bathrooms', 'sqft_lot', 'sqft_living', 'floors', 'waterfront', 'view', 'grade']]
Y = cleaned_data['price']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

lm_model = LinearRegression()
lm_model.fit(X_train, y_train)

y_pred = lm_model.predict(X_test)

accuracy = r2_score(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
mae = mean_absolute_error(y_test, y_pred)

print("R-squared: ", round(accuracy, 4))
print("Root Mean Squared Error: ", round(rmse, 4))
print("Mean absolute Error: ", round(mae, 4))

plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.title('Scatter Plot with Linear Regression Line')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.show()
```

```
R-squared:  0.5816
Root Mean Squared Error:  243348.4396
Mean absolute Error:  153107.9914
```