In [1]:
```
import pandas as pd


csv_file = "kc_house_data.csv"
# Read the first few rows of the CSV file into a DataFrame
df = pd.read_csv(csv_file)
df.head(50)
```

Out[1]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_bu
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN	0.0	...	7	1180	0.0	195
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	...	7	2170	400.0	195
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	...	6	770	0.0	193
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	...	7	1050	910.0	196
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	...	8	1680	0.0	198
5	7237550310	5/12/2014	1230000.0	4	4.50	5420	101930	1.0	0.0	0.0	...	11	3890	1530.0	200
6	1321400060	6/27/2014	257500.0	3	2.25	1715	6819	2.0	0.0	0.0	...	7	1715	?	199
7	2008000270	1/15/2015	291850.0	3	1.50	1060	9711	1.0	0.0	NaN	...	7	1060	0.0	196
8	2414600126	4/15/2015	229500.0	3	1.00	1780	7470	1.0	0.0	0.0	...	7	1050	730.0	196
9	3793500160	3/12/2015	323000.0	3	2.50	1890	6560	2.0	0.0	0.0	...	7	1890	0.0	200
10	1736800520	4/3/2015	662500.0	3	2.50	3560	9796	1.0	NaN	0.0	...	8	1860	1700.0	196
11	9212900260	5/27/2014	468000.0	2	1.00	1160	6000	1.0	0.0	0.0	...	7	860	300.0	194
12	114101516	5/28/2014	310000.0	3	1.00	1430	19901	1.5	0.0	0.0	...	7	1430	0.0	192
13	6054650070	10/7/2014	400000.0	3	1.75	1370	9680	1.0	0.0	0.0	...	7	1370	0.0	197
14	1175000570	3/12/2015	530000.0	5	2.00	1810	4850	1.5	0.0	0.0	...	7	1810	0.0	190
15	9297300055	1/24/2015	650000.0	4	3.00	2950	5000	2.0	0.0	3.0	...	9	1980	970.0	197
16	1875500060	7/31/2014	395000.0	3	2.00	1890	14040	2.0	0.0	0.0	...	7	1890	0.0	199
17	6865200140	5/29/2014	485000.0	4	1.00	1600	4300	1.5	0.0	0.0	...	7	1600	0.0	191
18	16000397	12/5/2014	189000.0	2	1.00	1200	9850	1.0	0.0	0.0	...	7	1200	?	192
19	7983200060	4/24/2015	230000.0	3	1.00	1250	9774	1.0	0.0	0.0	...	7	1250	0.0	196
20	6300500875	5/14/2014	385000.0	4	1.75	1620	4980	1.0	0.0	0.0	...	7	860	760.0	194
21	2524049179	8/26/2014	2000000.0	3	2.75	3050	44867	1.0	0.0	4.0	...	9	2330	720.0	196
22	7137970340	7/3/2014	285000.0	5	2.50	2270	6300	2.0	0.0	0.0	...	8	2270	0.0	199
23	8091400200	5/16/2014	252700.0	2	1.50	1070	9643	1.0	NaN	0.0	...	7	1070	0.0	198
24	3814700200	11/20/2014	329000.0	3	2.25	2450	6500	2.0	0.0	0.0	...	8	2450	0.0	198
25	1202000200	11/3/2014	233000.0	3	2.00	1710	4697	1.5	0.0	0.0	...	6	1710	0.0	194
26	1794500383	6/26/2014	937000.0	3	1.75	2450	2691	2.0	0.0	0.0	...	8	1750	700.0	191
27	3303700376	12/1/2014	667000.0	3	1.00	1400	1581	1.5	0.0	0.0	...	8	1400	0.0	190
28	5101402488	6/24/2014	438000.0	3	1.75	1520	6380	1.0	0.0	0.0	...	7	790	730.0	194
29	1873100390	3/2/2015	719000.0	4	2.50	2570	7173	2.0	0.0	0.0	...	8	2570	0.0	200
30	8562750320	11/10/2014	580500.0	3	2.50	2320	3980	2.0	0.0	0.0	...	8	2320	0.0	200
31	2426039314	12/1/2014	280000.0	2	1.50	1190	1265	3.0	0.0	0.0	...	7	1190	0.0	200
32	461000390	6/24/2014	687500.0	4	1.75	2330	5000	1.5	0.0	0.0	...	7	1510	820.0	192
33	7589200193	11/10/2014	535000.0	3	1.00	1090	3000	1.5	0.0	0.0	...	8	1090	0.0	192
34	7955080270	12/3/2014	322500.0	4	2.75	2060	6659	1.0	0.0	0.0	...	7	1280	780.0	198
35	9547205180	6/13/2014	696000.0	3	2.50	2300	3060	1.5	0.0	0.0	...	8	1510	790.0	193
36	9435300030	5/28/2014	550000.0	4	1.00	1660	34848	1.0	0.0	0.0	...	5	930	730.0	193
37	2768000400	12/30/2014	640000.0	4	2.00	2360	6000	2.0	0.0	0.0	...	8	2360	0.0	190
38	7895500070	2/13/2015	240000.0	4	1.00	1220	8075	1.0	0.0	0.0	...	7	890	330.0	196
39	2078500320	6/20/2014	605000.0	4	2.50	2620	7553	2.0	0.0	0.0	...	8	2620	0.0	199
40	5547700270	7/15/2014	625000.0	4	2.50	2570	5520	2.0	NaN	0.0	...	9	2570	0.0	200
41	7766200013	8/11/2014	775000.0	4	2.25	4220	24186	1.0	0.0	0.0	...	8	2600	1620.0	198
42	7203220400	7/7/2014	861990.0	5	2.75	3595	5639	2.0	0.0	0.0	...	9	3595	?	201
43	9270200160	10/28/2014	685000.0	3	1.00	1570	2280	2.0	0.0	0.0	...	7	1570	0.0	192
44	1432701230	7/29/2014	309000.0	3	1.00	1280	9656	1.0	0.0	0.0	...	6	920	360.0	195
45	8035350320	7/18/2014	488000.0	3	2.50	3160	13603	2.0	0.0	0.0	...	8	3160	0.0	200
46	8945200830	3/25/2015	210490.0	3	1.00	990	8528	1.0	0.0	0.0	...	6	990	0.0	196
47	4178300310	7/16/2014	785000.0	4	2.50	2290	13416	2.0	0.0	0.0	...	9	2290	0.0	198
48	9215400105	4/28/2015	450000.0	3	1.75	1250	5963	1.0	0.0	0.0	...	7	1250	0.0	195
49	822039084	3/11/2015	1350000.0	3	2.50	2753	65005	1.0	1.0	2.0	...	9	2165	588.0	195

50 rows × 21 columns

In [2]:
```
# the number of rows
num_rows = df.shape[0]

print("Number of Rows in the Dataset:", num_rows)
```

```
Number of Rows in the Dataset: 21597
```

In [3]:
```
# List of columns not to be used
columns_to_delete = ['date', 'view', 'sqft_above', 'sqft_basement', 'yr_renovated', 'zipcode', 'lat', 'long', 's

# Drop the specified columns
df = df.drop(columns=columns_to_delete)

# Display the modified DataFrame
(df.head(50))
```

Out[3]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	grade	yr_built
0	7129300520	221900.0	3	1.00	1180	5650	1.0	NaN	3	7	1955
1	6414100192	538000.0	3	2.25	2570	7242	2.0	0.0	3	7	1951
2	5631500400	180000.0	2	1.00	770	10000	1.0	0.0	3	6	1933
3	2487200875	604000.0	4	3.00	1960	5000	1.0	0.0	5	7	1965
4	1954400510	510000.0	3	2.00	1680	8080	1.0	0.0	3	8	1987
5	7237550310	1230000.0	4	4.50	5420	101930	1.0	0.0	3	11	2001
6	1321400060	257500.0	3	2.25	1715	6819	2.0	0.0	3	7	1995
7	2008000270	291850.0	3	1.50	1060	9711	1.0	0.0	3	7	1963
8	2414600126	229500.0	3	1.00	1780	7470	1.0	0.0	3	7	1960
9	3793500160	323000.0	3	2.50	1890	6560	2.0	0.0	3	7	2003
10	1736800520	662500.0	3	2.50	3560	9796	1.0	NaN	3	8	1965
11	9212900260	468000.0	2	1.00	1160	6000	1.0	0.0	4	7	1942
12	114101516	310000.0	3	1.00	1430	19901	1.5	0.0	4	7	1927
13	6054650070	400000.0	3	1.75	1370	9680	1.0	0.0	4	7	1977
14	1175000570	530000.0	5	2.00	1810	4850	1.5	0.0	3	7	1900
15	9297300055	650000.0	4	3.00	2950	5000	2.0	0.0	3	9	1979
16	1875500060	395000.0	3	2.00	1890	14040	2.0	0.0	3	7	1994
17	6865200140	485000.0	4	1.00	1600	4300	1.5	0.0	4	7	1916
18	16000397	189000.0	2	1.00	1200	9850	1.0	0.0	4	7	1921
19	7983200060	230000.0	3	1.00	1250	9774	1.0	0.0	4	7	1969
20	6300500875	385000.0	4	1.75	1620	4980	1.0	0.0	4	7	1947
21	2524049179	2000000.0	3	2.75	3050	44867	1.0	0.0	3	9	1968
22	7137970340	285000.0	5	2.50	2270	6300	2.0	0.0	3	8	1995
23	8091400200	252700.0	2	1.50	1070	9643	1.0	NaN	3	7	1985
24	3814700200	329000.0	3	2.25	2450	6500	2.0	0.0	4	8	1985
25	1202000200	233000.0	3	2.00	1710	4697	1.5	0.0	5	6	1941
26	1794500383	937000.0	3	1.75	2450	2691	2.0	0.0	3	8	1915
27	3303700376	667000.0	3	1.00	1400	1581	1.5	0.0	5	8	1909
28	5101402488	438000.0	3	1.75	1520	6380	1.0	0.0	3	7	1948
29	1873100390	719000.0	4	2.50	2570	7173	2.0	0.0	3	8	2005
30	8562750320	580500.0	3	2.50	2320	3980	2.0	0.0	3	8	2003
31	2426039314	280000.0	2	1.50	1190	1265	3.0	0.0	3	7	2005
32	461000390	687500.0	4	1.75	2330	5000	1.5	0.0	4	7	1929
33	7589200193	535000.0	3	1.00	1090	3000	1.5	0.0	4	8	1929
34	7955080270	322500.0	4	2.75	2060	6659	1.0	0.0	3	7	1981
35	9547205180	696000.0	3	2.50	2300	3060	1.5	0.0	3	8	1930
36	9435300030	550000.0	4	1.00	1660	34848	1.0	0.0	1	5	1933
37	2768000400	640000.0	4	2.00	2360	6000	2.0	0.0	4	8	1904
38	7895500070	240000.0	4	1.00	1220	8075	1.0	0.0	2	7	1969
39	2078500320	605000.0	4	2.50	2620	7553	2.0	0.0	3	8	1996
40	5547700270	625000.0	4	2.50	2570	5520	2.0	NaN	3	9	2000
41	7766200013	775000.0	4	2.25	4220	24186	1.0	0.0	3	8	1984
42	7203220400	861990.0	5	2.75	3595	5639	2.0	0.0	3	9	2014
43	9270200160	685000.0	3	1.00	1570	2280	2.0	0.0	3	7	1922
44	1432701230	309000.0	3	1.00	1280	9656	1.0	0.0	4	6	1959
45	8035350320	488000.0	3	2.50	3160	13603	2.0	0.0	3	8	2003
46	8945200830	210490.0	3	1.00	990	8528	1.0	0.0	3	6	1966
47	4178300310	785000.0	4	2.50	2290	13416	2.0	0.0	4	9	1981
48	9215400105	450000.0	3	1.75	1250	5963	1.0	0.0	4	7	1953
49	822039084	1350000.0	3	2.50	2753	65005	1.0	1.0	5	9	1953

In [4]:
```
# the number of rows
num_rows = df.shape[0]

print("Number of Rows in the Dataset:", num_rows)
```

```
Number of Rows in the Dataset: 21597
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   id           21597 non-null  int64  
 1   price        21597 non-null  float64
 2   bedrooms     21597 non-null  int64  
 3   bathrooms    21597 non-null  float64
 4   sqft_living  21597 non-null  int64  
 5   sqft_lot     21597 non-null  int64  
 6   floors       21597 non-null  float64
 7   waterfront   19221 non-null  float64
 8   condition    21597 non-null  int64  
 9   grade        21597 non-null  int64  
 10  yr_built     21597 non-null  int64  
dtypes: float64(4), int64(7)
memory usage: 1.8 MB
```

In [6]:
```
import pandas as pd

# Count NaN, null, or empty values for each column
null_counts = df.isnull().sum()

# Print the counts
print("Count of NaN, null, or empty values for every column:")
print(null_counts)
```

```
Count of NaN, null, or empty values for every column:
id                0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront     2376
condition         0
grade             0
yr_built          0
dtype: int64
```

In [7]:
```
# Find the minimum and maximum values in the 'price' column
min_price = df['price'].min()
max_price = df['price'].max()

print("Minimum Price:", min_price)
print("Maximum Price:", max_price)
```

```
Minimum Price: 78000.0
Maximum Price: 7700000.0
```

In [8]:
```
# Filter non-waterfront properties
non_waterfront_properties = df[df['waterfront'] == 0.0]

# Find the range of prices for non-waterfront properties
min_price_non_waterfront = non_waterfront_properties['price'].min()
max_price_non_waterfront = non_waterfront_properties['price'].max()

print("Range of Prices for Non-Waterfront Properties:")
print("Minimum Price:", min_price_non_waterfront)
print("Maximum Price:", max_price_non_waterfront)
```

```
Range of Prices for Non-Waterfront Properties:
Minimum Price: 78000.0
Maximum Price: 7700000.0
```

In [9]:
```
nan_count = df['waterfront'].isna().sum()

print("Number of NaN values in the 'waterfront' column:", nan_count)
```

```
Number of NaN values in the 'waterfront' column: 2376
```

In [10]:
```
# Count the number of waterfront properties with the value 1.0
num_waterfront_properties = df['waterfront'].value_counts().get(1.0, 0)

# Display the count
print("Number of Waterfront Properties:", num_waterfront_properties)
```

```
Number of Waterfront Properties: 146
```

In [11]:
```
# Filter waterfront properties
waterfront_properties = df[df['waterfront'] == 1.0]

# Find the range of prices for waterfront properties
min_price_waterfront = waterfront_properties['price'].min()
max_price_waterfront = waterfront_properties['price'].max()

print("Range of Prices for Waterfront Properties:")
print("Minimum Price:", min_price_waterfront)
print("Maximum Price:", max_price_waterfront)
```

```
Range of Prices for Waterfront Properties:
Minimum Price: 285000.0
Maximum Price: 7060000.0
```

In [12]:
```
# Filter properties with NaN values in the 'waterfront' column
nan_waterfront_properties = df[df['waterfront'].isna()]

#prices for properties with NaN values in the 'waterfront' column
prices_nan_waterfront = nan_waterfront_properties['price']

# Display the prices
print("Prices of Properties with NaN in Waterfront Column:")
print(prices_nan_waterfront)
```

```
Prices of Properties with NaN in Waterfront Column:
0        221900.0
10       662500.0
23       252700.0
40       625000.0
55       885000.0
           ...
21578    350000.0
21582    541800.0
21586    224000.0
21587    507250.0
21595    400000.0
Name: price, Length: 2376, dtype: float64
```

In [13]:
```
# Find the range of prices for properties with NaN values in the 'waterfront' column
min_price_nan_waterfront = prices_nan_waterfront.min()
max_price_nan_waterfront = prices_nan_waterfront.max()

print("Range of Prices for Properties with NaN in Waterfront Column:")
print("Minimum Price:", min_price_nan_waterfront)
print("Maximum Price:", max_price_nan_waterfront)
```

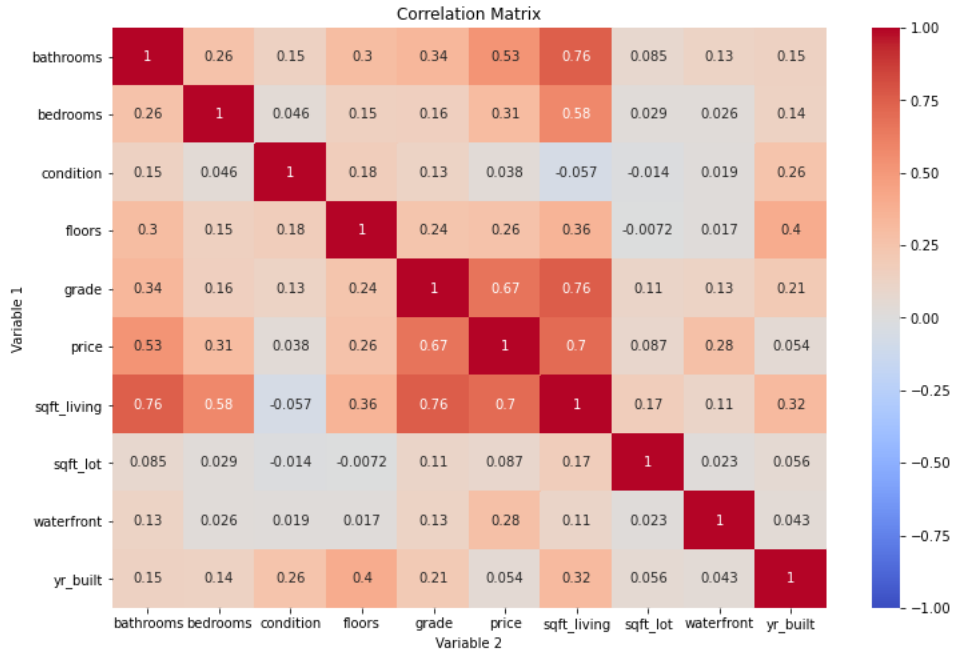
```
Range of Prices for Properties with NaN in Waterfront Column:
Minimum Price: 80000.0
Maximum Price: 3200000.0
```

As the range of prices for waterfront properties with Nan values crosses below the minimum value of known waterfront (ie $280,000) and reaches halfway into the range of known waterfront prices, it cannot be safely imputed that all Nan properties are non-waterfront. So will delete all waterfront properties with nan values.

In [14]:
```
import pandas as pd



# Drop rows with NaN, null, or empty values in the 'waterfront' column
df = df.dropna(subset=['waterfront'])

# Now 'df' will contain the DataFrame with rows having NaN, null, or empty values in the 'waterfront' column rem
import pandas as pd

# Assuming 'df' is your DataFrame containing the columns
# Replace 'df' with your actual DataFrame name

# Print 'waterfront' column
print(df['waterfront'])
```

```
1        0.0
2        0.0
3        0.0
4        0.0
5        0.0
        ...
21591    0.0
21592    0.0
21593    0.0
21594    0.0
21596    0.0
Name: waterfront, Length: 19221, dtype: float64
```

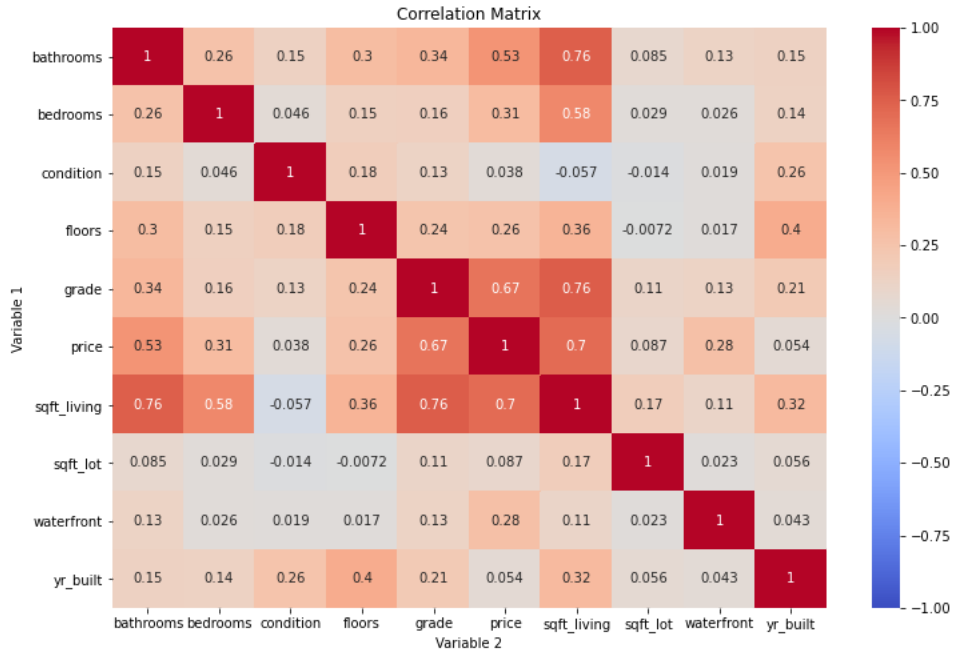
In [15]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import pointbiserialr, chi2_contingency

# Function to calculate Cramer's V
def cramers_v(confusion_matrix):
    chi2 = chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    phi2 = chi2 / n
    r, k = confusion_matrix.shape
    phi2corr = max(0, phi2 - ((k-1)*(r-1))/(n-1))
    rcorr = r - ((r-1)**2)/(n-1)
    kcorr = k - ((k-1)**2)/(n-1)
    return np.sqrt(phi2corr / min((kcorr-1), (rcorr-1)))

# Function to calculate correlation coefficients
def calculate_correlation(var1, var2):
    if var1 in continuous_vars and var2 in continuous_vars:
        # For continuous-continuous variables, use Pearson correlation coefficient
        corr_coef = df[var1].corr(df[var2])
    elif var1 in continuous_vars and var2 in ordinal_categorical_vars:
        # For continuous-ordinal categorical variables, use point-biserial correlation coefficient
        corr_coef, _ = pointbiserialr(df[var2], df[var1])
    elif var1 in ordinal_categorical_vars and var2 in continuous_vars:
        # For ordinal categorical-continuous variables, use point-biserial correlation coefficient
        corr_coef, _ = pointbiserialr(df[var1], df[var2])
    elif var1 in binary_categorical_vars and var2 in continuous_vars:
        # For binary categorical-continuous variables, use point-biserial correlation coefficient
        corr_coef, _ = pointbiserialr(df[var1], df[var2])
    elif var1 in ordinal_categorical_vars and var2 in ordinal_categorical_vars:
        # For ordinal categorical-ordinal categorical variables, calculate Cramer's V
        confusion_matrix = pd.crosstab(df[var1], df[var2])
        corr_coef = cramers_v(confusion_matrix)
    elif var1 in binary_categorical_vars and var2 in binary_categorical_vars:
        # For binary categorical-binary categorical variables, calculate Cramer's V
        confusion_matrix = pd.crosstab(df[var1], df[var2])
        corr_coef = cramers_v(confusion_matrix)
    else:
        # For all other combinations, return None
        corr_coef = None
    return corr_coef

# List of continuous variables
continuous_vars = ['sqft_living', 'sqft_lot','price']

# List of ordinal categorical variables
ordinal_categorical_vars = ['bedrooms', 'bathrooms', 'floors', 'waterfront', 'condition', 'grade', 'yr_built']

# List of binary categorical variables
binary_categorical_vars = []

# List to store correlation coefficients
correlation_matrix = []

# Iterate over all combinations of variables
for var1 in continuous_vars + ordinal_categorical_vars + binary_categorical_vars:
    for var2 in continuous_vars + ordinal_categorical_vars + binary_categorical_vars:
        # Calculate correlation coefficient
        corr_coef = calculate_correlation(var1, var2)
        # Append to correlation matrix
        correlation_matrix.append([var1, var2, corr_coef])

# Convert correlation matrix to DataFrame
correlation_df = pd.DataFrame(correlation_matrix, columns=['Variable 1', 'Variable 2', 'Correlation Coefficient'

# Create a pivot table for heatmap
heatmap_data = correlation_df.pivot(index='Variable 1', columns='Variable 2', values='Correlation Coefficient')

# Create the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(heatmap_data, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix')
plt.show()
```



In [16]:
```
# Calculate summary statistics including the 95th percentile
summary_data_with_95th = df.describe().round(1).append(df.quantile(0.95).rename("95%"))

print(summary_data_with_95th)
```

```
                id      price  bedrooms  bathrooms  sqft_living   sqft_lot  \
count  1.922100e+04    19221.0   19221.0    19221.0       19221.0    19221.0
mean   4.592301e+09   541639.8       3.4        2.1        2082.4    15073.8
std    2.876995e+09   372247.8       0.9        0.8         922.5    40817.3
min    1.000102e+06    78000.0       1.0        0.5         370.0      520.0
25%    2.124049e+09   322000.0       3.0        1.8        1430.0     5040.0
50%    3.905081e+09   450000.0       3.0        2.2        1920.0     7620.0
75%    7.334501e+09   644000.0       4.0        2.5        2550.0    10716.0
max    9.900000e+09  7700000.0      33.0        8.0       13540.0  1651359.0
95%    9.297300e+09  1170000.0       5.0        3.5        3770.0    43560.0

        floors  waterfront  condition    grade  yr_built
count  19221.0     19221.0    19221.0  19221.0   19221.0
mean       1.5         0.0        3.4      7.7    1971.0
std        0.5         0.1        0.7      1.2      29.4
min        1.0         0.0        1.0      3.0    1900.0
25%        1.0         0.0        3.0      7.0    1951.0
50%        1.5         0.0        3.0      7.0    1975.0
75%        2.0         0.0        4.0      8.0    1997.0
max        3.5         1.0        5.0     13.0    2015.0
95%        2.0         0.0        5.0     10.0    2011.0
```

In [17]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import pointbiserialr, chi2_contingency

# Function to calculate Cramer's V
def cramers_v(confusion_matrix):
    chi2 = chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    phi2 = chi2 / n
    r, k = confusion_matrix.shape
    phi2corr = max(0, phi2 - ((k-1)*(r-1))/(n-1))
    rcorr = r - ((r-1)**2)/(n-1)
    kcorr = k - ((k-1)**2)/(n-1)
    return np.sqrt(phi2corr / min((kcorr-1), (rcorr-1)))

# Function to calculate correlation coefficients
def calculate_correlation(var1, var2):
    if var1 in continuous_vars and var2 in continuous_vars:
        # For continuous-continuous variables, use Pearson correlation coefficient
        corr_coef = df[var1].corr(df[var2])
    elif var1 in continuous_vars and var2 in ordinal_categorical_vars:
        # For continuous-ordinal categorical variables, use point-biserial correlation coefficient
        corr_coef, _ = pointbiserialr(df[var2], df[var1])
    elif var1 in ordinal_categorical_vars and var2 in continuous_vars:
        # For ordinal categorical-continuous variables, use point-biserial correlation coefficient
        corr_coef, _ = pointbiserialr(df[var1], df[var2])
    elif var1 in binary_categorical_vars and var2 in continuous_vars:
        # For binary categorical-continuous variables, use point-biserial correlation coefficient
        corr_coef, _ = pointbiserialr(df[var1], df[var2])
    elif var1 in ordinal_categorical_vars and var2 in ordinal_categorical_vars:
        # For ordinal categorical-ordinal categorical variables, calculate Cramer's V
        confusion_matrix = pd.crosstab(df[var1], df[var2])
        corr_coef = cramers_v(confusion_matrix)
    elif var1 in binary_categorical_vars and var2 in binary_categorical_vars:
        # For binary categorical-binary categorical variables, calculate Cramer's V
        confusion_matrix = pd.crosstab(df[var1], df[var2])
        corr_coef = cramers_v(confusion_matrix)
    else:
        # For all other combinations, return None
        corr_coef = None
    return corr_coef

# List of continuous variables
continuous_vars = ['sqft_living', 'sqft_lot', 'price']

# List of ordinal categorical variables
ordinal_categorical_vars = ['bedrooms', 'bathrooms', 'floors', 'waterfront', 'condition', 'grade', 'yr_built']

# List of binary categorical variables
binary_categorical_vars = []

# List of predictor variables including dependent variable
predictor_vars = continuous_vars + ordinal_categorical_vars + binary_categorical_vars

# List to store correlation coefficients
correlation_matrix = []

# Iterate over all combinations of variables
for var1 in predictor_vars:
    for var2 in predictor_vars:
        # Calculate correlation coefficient
        corr_coef = calculate_correlation(var1, var2)
        # Append to correlation matrix
        correlation_matrix.append([var1, var2, corr_coef])

# Convert correlation matrix to DataFrame
correlation_df = pd.DataFrame(correlation_matrix, columns=['Variable 1', 'Variable 2', 'Correlation Coefficient'

# Create a pivot table for heatmap
heatmap_data = correlation_df.pivot(index='Variable 1', columns='Variable 2', values='Correlation Coefficient')

# Create the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(heatmap_data, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix')
plt.show()
```



In [18]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm

# Assuming df is your DataFrame containing the data

# Add constant term for intercept
X = sm.add_constant(df[['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'condition',

# Dependent variable
y = df['price']

# Fit OLS regression model
model = sm.OLS(y, X).fit()

# Print regression results
print(model.summary())
```

```
bedrooms    -3.972e+04   2344.674    -16.939      0.000   -4.43e+04   -3.51e+04
bathrooms   -1.188e+04   3770.557     -3.150      0.002   -1.93e+04   -4487.812
sqft_living   214.1508      3.721     57.548      0.000     206.857     221.445
sqft_lot       -0.3736      0.043     -8.731      0.000      -0.458      -0.290
floors      -2.006e+04   3881.385     -5.169      0.000   -2.77e+04   -1.25e+04
waterfront    8.15e+05   1.99e+04     41.047      0.000    7.76e+05    8.54e+05
condition    5.955e+04   2739.447     21.739      0.000    5.42e+04    6.49e+04
grade        1.045e+05   2420.595     43.181      0.000    9.98e+04    1.09e+05
==============================================================================
Omnibus:                    13807.120   Durbin-Watson:                   1.977
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           759636.350
Skew:                           2.885   Prob(JB):                         0.00
Kurtosis:                      33.252   Cond. No.                     5.06e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.06e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [19]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm

# Assuming df is your DataFrame containing the data

# Add constant term for intercept
X = sm.add_constant(df[['bedrooms', 'bathrooms', 'sqft_living', 'floors', 'waterfront', 'condition', 'grade',]])

# Dependent variable
y = df['price']

# Fit OLS regression model
model = sm.OLS(y, X).fit()

# Print regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.594
Model:                            OLS   Adj. R-squared:                  0.594
Method:                 Least Squares   F-statistic:                     4024.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:48:51   Log-Likelihood:            -2.6515e+05
No. Observations:               19221   AIC:                         5.303e+05
Df Residuals:                   19213   BIC:                         5.304e+05
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -7.296e+05   1.88e+04    -38.808      0.000   -7.66e+05   -6.93e+05
bedrooms   -3.799e+04   2340.898    -16.229      0.000   -4.26e+04   -3.34e+04
bathrooms  -1.092e+04   3776.318     -2.891      0.004   -1.83e+04   -3514.295
sqft_living  208.9293      3.680     56.773      0.000     201.716     216.143
floors     -1.811e+04   3882.488     -4.664      0.000   -2.57e+04   -1.05e+04
waterfront  8.158e+05   1.99e+04     41.005      0.000    7.77e+05    8.55e+05
condition   6.006e+04   2744.187     21.886      0.000    5.47e+04    6.54e+04
grade       1.049e+05   2424.983     43.251      0.000       1e+05     1.1e+05
==============================================================================
Omnibus:                    13910.275   Durbin-Watson:                   1.977
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           782854.793
Skew:                           2.912   Prob(JB):                         0.00
Kurtosis:                      33.718   Cond. No.                     2.65e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.65e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [20]:
```
import statsmodels.api as sm

# Define predictor variables
predictors = ['bedrooms', 'bathrooms', 'grade', 'sqft_living', 'waterfront']

# Add constant term for the intercept
X = sm.add_constant(df[predictors])

# Define dependent variable
y = df['price']

# Fit OLS model
model = sm.OLS(y, X).fit()

# Print model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.583
Model:                            OLS   Adj. R-squared:                  0.582
Method:                 Least Squares   F-statistic:                     5363.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:48:52   Log-Likelihood:            -2.6543e+05
No. Observations:               19221   AIC:                         5.309e+05
Df Residuals:                   19215   BIC:                         5.309e+05
Df Model:                           5
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const       -4.853e+05   1.53e+04    -31.619      0.000   -5.15e+05   -4.55e+05
bedrooms    -3.373e+04   2367.287    -14.249      0.000   -3.84e+04   -2.91e+04
bathrooms   -2.579e+04   3589.244     -7.186      0.000   -3.28e+04   -1.88e+04
grade        9.589e+04   2384.346     40.216      0.000    9.12e+04    1.01e+05
sqft_living   218.2298      3.691     59.132      0.000     210.996     225.464
waterfront     8.3e+05   2.02e+04     41.140      0.000     7.9e+05     8.7e+05
==============================================================================
Omnibus:                    13735.505   Durbin-Watson:                   1.973
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           726462.986
Skew:                           2.875   Prob(JB):                         0.00
Kurtosis:                      32.564   Cond. No.                     2.65e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.65e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [21]:
```
import statsmodels.api as sm

# Define predictor variable
predictor = 'sqft_living'

# Add constant term for the intercept
X = sm.add_constant(df[predictor])

# Define dependent variable
y = df['price']

# Fit OLS model
model = sm.OLS(y, X).fit()

# Print model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.497
Model:                            OLS   Adj. R-squared:                  0.497
Method:                 Least Squares   F-statistic:                 1.895e+04
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:48:52   Log-Likelihood:            -2.6723e+05
No. Observations:               19221   AIC:                         5.345e+05
Df Residuals:                   19219   BIC:                         5.345e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -5.045e+04   4703.746    -10.726      0.000   -5.97e+04   -4.12e+04
sqft_living   284.3280      2.065    137.675      0.000     280.280     288.376
==============================================================================
Omnibus:                    13337.882   Durbin-Watson:                   1.989
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           507333.623
Skew:                           2.859   Prob(JB):                         0.00
Kurtosis:                      27.511   Cond. No.                     5.62e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.62e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [22]:
```
import statsmodels.api as sm

# Define predictor variables
predictors = ['grade', 'sqft_living', 'waterfront']

# Add constant term for the intercept
X = sm.add_constant(df[predictors])

# Define dependent variable
y = df['price']

# Fit OLS model
model = sm.OLS(y, X).fit()

# Print model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.576
Model:                            OLS   Adj. R-squared:                  0.576
Method:                 Least Squares   F-statistic:                     8700.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:48:52   Log-Likelihood:            -2.6558e+05
No. Observations:               19221   AIC:                         5.312e+05
Df Residuals:                   19217   BIC:                         5.312e+05
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -5.876e+05   1.37e+04    -42.936      0.000   -6.14e+05   -5.61e+05
grade        9.75e+04   2304.784     42.301      0.000     9.3e+04    1.02e+05
sqft_living   180.4716      2.945     61.287      0.000     174.700     186.243
waterfront   8.578e+05   2.03e+04     42.333      0.000    8.18e+05    8.98e+05
==============================================================================
Omnibus:                    14084.230   Durbin-Watson:                   1.969
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           799379.849
Skew:                           2.967   Prob(JB):                         0.00
Kurtosis:                      34.031   Cond. No.                     2.64e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.64e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

import pandas as pd import numpy as np import seaborn as sns import matplotlib.pyplot as plt from scipy.stats import pointbiserialr, chi2_contingency

Function to calculate Cramer's V

def cramers_v(confusion_matrix): chi2 = chi2_contingency(confusion_matrix)[0] n = confusion_matrix.sum().sum() phi2 = chi2 / n r, k = confusion_matrix.shape phi2corr = max(0, phi2 - ((k-1)*(r-1))/(n-1)) rcorr = r - ((r-1)*2)/(n-1) kcorr = k - ((k-1)**2)/(n-1) return np.sqrt(phi2corr / min((kcorr-1), (rcorr-1)))

Function to calculate correlation coefficients

def calculate_correlation(var1, var2): if var1 in continuous_vars and var2 in continuous_vars: # For continuous-continuous variables, use Pearson correlation coefficient corr_coef = df[var1].corr(df[var2]) elif var1 in continuous_vars and var2 in ordinal_categorical_vars: # For continuous-ordinal categorical variables, use point-biserial correlation coefficient corr_coef, _ = pointbiserialr(df[var2], df[var1]) elif var1 in ordinal_categorical_vars and var2 in continuous_vars: # For ordinal categorical-continuous variables, use point-biserial correlation coefficient corr_coef, _ = pointbiserialr(df[var1], df[var2]) elif var1 in binary_categorical_vars and var2 in continuous_vars: # For binary categorical-continuous variables, use point-biserial correlation coefficient corr_coef, _ = pointbiserialr(df[var1], df[var2]) elif var1 in ordinal_categorical_vars and var2 in ordinal_categorical_vars: # For ordinal categorical-ordinal categorical variables, calculate Cramer's V confusion_matrix = pd.crosstab(df[var1], df[var2]) corr_coef = cramers_v(confusion_matrix) elif var1 in binary_categorical_vars and var2 in binary_categorical_vars: # For binary categorical-binary categorical variables, calculate Cramer's V confusion_matrix = pd.crosstab(df[var1], df[var2]) corr_coef = cramers_v(confusion_matrix) else: # For all other combinations, return None corr_coef = None return corr_coef

List of continuous variables

continuous_vars = ['sqft_living', 'sqft_lot', 'price']

List of ordinal categorical variables

ordinal_categorical_vars = ['bedrooms', 'bathrooms', 'floors', 'waterfront', 'condition', 'grade', 'yr_built']

List of binary categorical variables

binary_categorical_vars = []

List of predictor variables including dependent variable

predictor_vars = continuous_vars + ordinal_categorical_vars + binary_categorical_vars

List to store correlation coefficients

correlation_matrix = []

Iterate over all combinations of variables

for var1 in predictor_vars: for var2 in predictor_vars: # Calculate correlation coefficient corr_coef = calculate_correlation(var1, var2) # Append to correlation matrix correlation_matrix.append([var1, var2, corr_coef])

Convert correlation matrix to DataFrame

correlation_df = pd.DataFrame(correlation_matrix, columns=['Variable 1', 'Variable 2', 'Correlation Coefficient'])

Create a pivot table for heatmap

heatmap_data = correlation_df.pivot(index='Variable 1', columns='Variable 2', values='Correlation Coefficient')

Create the heatmap

plt.figure(figsize=(12, 8)) sns.heatmap(heatmap_data, annot=True, cmap='coolwarm', vmin=-1, vmax=1) plt.title('Correlation Matrix') plt.show()

import matplotlib.pyplot as plt import seaborn as sns

columns_to_plot = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'condition', 'grade', 'yr_built', 'price']

fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(20, 8))

for col, ax in zip(columns_to_plot, axes.flatten()): sns.scatterplot(x=col, y='price', data=df, ax=ax, alpha=0.4, color='b') ax.set_title(f'Scatter Plot: {col}') ax.set_xlabel(col) ax.set_ylabel('Price')

plt.tight_layout() plt.show()

In [23]:
```
#Looks like not many values above 10000 sq feet so will check further and remove
```

In [24]:
```
import pandas as pd


# Count values above 10000 in the 'sqft_living' column
count_above_10000 = (df['sqft_living'] > 10000).sum()

# Print the count
print("Count of sqft_living values above 10000:", count_above_10000)
```

```
Count of sqft_living values above 10000: 3
```

In [25]:
```
import pandas as pd

# Remove rows where 'sqft_living' values are above 10000
df = df[df['sqft_living'] <= 10000]

# Now 'df' will contain rows where 'sqft_living' values are not above 10000
```

In [26]:
```
import pandas as pd

# Count values above 125000 in the 'sqft_lot' column
count_above_125 = (df['sqft_lot'] > 125000).sum()

# Print the count
print("Count of sqft_lot values above 125000:", count_above_125)
```

```
Count of sqft_lot values above 125000: 331
```

In [27]:
```
import pandas as pd

# Remove rows where 'sqft_lot' values are above 125000
df = df[df['sqft_lot'] <= 125000]
```

From the start it is important to lower the risk of any property investing/renovating. By removing outliers early, this will likely reduce the capital risk and may help improve the accuracy of the modelling which inturn cna help to lower the risk further by producing more precise predictions.

In [28]:
```
bedrooms_range = df['bedrooms'].max() - df['bedrooms'].min()

print("Range of values in the 'bedrooms' column:", bedrooms_range)
```

```
Range of values in the 'bedrooms' column: 32
```

In [29]:
```
properties_more_than_6_bedrooms = (df['bedrooms'] > 6).sum()

print("Number of properties with more than 6 bedrooms:", properties_more_than_6_bedrooms)
```

```
Number of properties with more than 6 bedrooms: 56
```

In [30]:
```
properties_more_than_4_bedrooms = (df['bedrooms'] <= 4).sum()

print("Number of properties with 4 or fewer bedrooms:", properties_more_than_4_bedrooms)
```

```
Number of properties with 4 or fewer bedrooms: 17191
```

In [31]:
```
df = df[df['bedrooms'] <=7]
```

In [32]:
```
properties_less_than_2_bedrooms = (df['bedrooms'] < 2).sum()

print("Number of properties with less than 2 bedrooms:", properties_less_than_2_bedrooms)
```

```
Number of properties with less than 2 bedrooms: 172
```

In [33]:
```
properties_less_than_1_bedroom = df[df['bedrooms'] < 1]

# print the properties with less than 1 bedroom

print(properties_less_than_1_bedroom)
```

```
Empty DataFrame
Columns: [id, price, bedrooms, bathrooms, sqft_living, sqft_lot, floors, waterfront, condition, grade, yr_buil
t]
Index: []
```

In [34]:
```
nan_values_sqft_living = df['sqft_living'].isna().sum()

print("Number of NaN values in the 'sqft_living' column:", nan_values_sqft_living)
```

```
Number of NaN values in the 'sqft_living' column: 0
```

In [35]:
```
lowest_sqft_living = df['sqft_living'].min()
highest_sqft_living = df['sqft_living'].max()

print("Lowest value in 'sqft_living':", lowest_sqft_living)
print("Highest value in 'sqft_living':", highest_sqft_living)
```

```
Lowest value in 'sqft_living': 370
Highest value in 'sqft_living': 9890
```

In [36]:
```
num_bathrooms_greater_than_5 = df[df['bathrooms'] > 5].shape[0]
print("Number of bathrooms greater than 5:", num_bathrooms_greater_than_5)
```

```
Number of bathrooms greater than 5: 29
```

In [37]:
```
num_bathrooms_greater_than_6 = df[df['bathrooms'] > 6].shape[0]
print("Number of bathrooms greater than 6:", num_bathrooms_greater_than_6)
```

```
Number of bathrooms greater than 6: 5
```

In [38]:
```
df = df[df['bathrooms'] <= 6]
```

In [ ]:
```
```

In [39]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm

# Assuming df is your DataFrame containing the data

# Add constant term for intercept
X = sm.add_constant(df['bedrooms'])

# Dependent variable
y = df['price']

# Fit OLS regression model
model = sm.OLS(y, X).fit()

# Print regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.100
Model:                            OLS   Adj. R-squared:                  0.100
Method:                 Least Squares   F-statistic:                     2098.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:48:52   Log-Likelihood:            -2.6703e+05
No. Observations:               18860   AIC:                         5.341e+05
Df Residuals:                   18858   BIC:                         5.341e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        1.05e+05   9757.245     10.758      0.000    8.58e+04    1.24e+05
bedrooms    1.284e+05   2803.672     45.799      0.000    1.23e+05    1.34e+05
==============================================================================
Omnibus:                    14363.820   Durbin-Watson:                   1.978
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           431045.428
```

In [40]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm

# Assuming df is your DataFrame containing the data

# Add constant term for intercept
X = sm.add_constant(df['bathrooms'])

# Dependent variable
y = df['price']

# Fit OLS regression model
model = sm.OLS(y, X).fit()

# Print regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.267
Model:                            OLS   Adj. R-squared:                  0.267
Method:                 Least Squares   F-statistic:                     6870.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:48:53   Log-Likelihood:            -2.6509e+05
No. Observations:               18860   AIC:                         5.302e+05
Df Residuals:                   18858   BIC:                         5.302e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2.001e+04   6629.164      3.019      0.003    7018.064     3.3e+04
bathrooms   2.454e+05   2960.843     82.884      0.000     2.4e+05    2.51e+05
==============================================================================
Omnibus:                    13000.402   Durbin-Watson:                   1.967
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           326941.718
```

In [41]:
```
min_year_built = df['yr_built'].min()
max_year_built = df['yr_built'].max()

print("Range of values in yr_built column:", min_year_built, "-", max_year_built)
```

```
Range of values in yr_built column: 1900 - 2015
```

In [42]:
```
# Define the number of desired groups
num_groups = 5

# Calculate the bin width
bin_width = (max_year_built - min_year_built) / num_groups

# Create the bins
bins = [min_year_built + i * bin_width for i in range(num_groups)]
bins.append(max_year_built)

# Use pd.cut to assign each value in 'yr_built' to its respective bin
df['yr_built_group'] = pd.cut(df['yr_built'], bins=bins, labels=[f'Group {i+1}' for i in range(num_groups)], inc

# Display the count of values in each group
print("Number of properties in each group:")
print(df['yr_built_group'].value_counts())
```

```
Number of properties in each group:
Group 5    5438
Group 3    5101
Group 4    4590
Group 2    2143
Group 1    1588
Name: yr_built_group, dtype: int64
```

In [43]:
```
# Convert the categorical variable 'yr_built_group' into dummy variables
dummy_variables = pd.get_dummies(df['yr_built_group'], prefix='yr_built', drop_first=True)

# Concatenate the dummy variables with the original dataframe
df = pd.concat([df, dummy_variables], axis=1)

# Drop the original 'yr_built' and 'yr_built_group' columns
df.drop(columns=['yr_built', 'yr_built_group'], inplace=True)
```

In [44]:
```
import pandas as pd

# Filter out rows where the number of bedrooms is not equal to 1
filtered_df = df[df['bedrooms'] != 1]

# List of categorical variables to encode
categorical_variables = ['bedrooms']

# Iterate over each categorical variable
for var in categorical_variables:
    # Convert the categorical variable into dummy variables
    dummy_variables = pd.get_dummies(df[var], prefix=var, drop_first=True)

    # Concatenate the dummy variables with the original dataframe
    df = pd.concat([df, dummy_variables], axis=1)



# Display the updated dataframe
print(df)
```

```
              id      price  bedrooms  bathrooms  sqft_living  sqft_lot  \
1     6414100192   538000.0         3       2.25         2570      7242
2     5631500400   180000.0         2       1.00          770     10000
3     2487200875   604000.0         4       3.00         1960      5000
4     1954400510   510000.0         3       2.00         1680      8080
5     7237550310  1230000.0         4       4.50         5420    101930
...          ...        ...       ...        ...          ...       ...
21591 2997800021   475000.0         3       2.50         1310      1294
21592  263000018   360000.0         3       2.50         1530      1131
21593 6600060120   400000.0         4       2.50         2310      5813
21594 1523300141   402101.0         2       0.75         1020      1350
21596 1523300157   325000.0         2       0.75         1020      1076

       floors  waterfront  condition  grade  yr_built_Group 2  \
1         2.0         0.0          3      7                  0
2         1.0         0.0          3      6                  1
3         1.0         0.0          5      7                  0
4         1.0         0.0          3      8                  0
5         1.0         0.0          3     11                  0
...       ...         ...        ...    ...                ...
21591     2.0         0.0          3      8                  0
21592     3.0         0.0          3      8                  0
21593     2.0         0.0          3      8                  0
21594     2.0         0.0          3      7                  0
21596     2.0         0.0          3      7                  0

       yr_built_Group 3  yr_built_Group 4  yr_built_Group 5  bedrooms_2  \
1                     1                 0                 0           0
2                     0                 0                 0           1
3                     1                 0                 0           0
4                     0                 1                 0           0
5                     0                 0                 1           0
...                 ...               ...               ...         ...
21591                 0                 0                 1           0
21592                 0                 0                 1           0
21593                 0                 0                 1           0
21594                 0                 0                 1           1
21596                 0                 0                 1           1

       bedrooms_3  bedrooms_4  bedrooms_5  bedrooms_6  bedrooms_7
1               1           0           0           0           0
2               0           0           0           0           0
3               0           1           0           0           0
4               1           0           0           0           0
5               0           1           0           0           0
...           ...         ...         ...         ...         ...
21591           1           0           0           0           0
21592           1           0           0           0           0
21593           0           1           0           0           0
21594           0           0           0           0           0
21596           0           0           0           0           0

[18860 rows x 20 columns]
```

In [45]:
```
# List of categorical variables to encode
categorical_variables = [ 'bathrooms', 'floors', 'waterfront', 'condition', 'grade']

# Iterate over each categorical variable
for var in categorical_variables:
    # Convert the categorical variable into dummy variables
    dummy_variables = pd.get_dummies(df[var], prefix=var, drop_first=True)

    # Concatenate the dummy variables with the original dataframe
    df = pd.concat([df, dummy_variables], axis=1)



# Display the updated dataframe
print(df)
```

```
              id      price  bedrooms  bathrooms  sqft_living  sqft_lot  \
1      6414100192   538000.0         3       2.25         2570      7242
2      5631500400   180000.0         2       1.00          770     10000
3      2487200875   604000.0         4       3.00         1960      5000
4      1954400510   510000.0         3       2.00         1680      8080
5      7237550310  1230000.0         4       4.50         5420    101930
...           ...        ...       ...        ...          ...       ...
21591  2997800021   475000.0         3       2.50         1310      1294
21592   263000018   360000.0         3       2.50         1530      1131
21593  6600060120   400000.0         4       2.50         2310      5813
21594  1523300141   402101.0         2       0.75         1020      1350
21596  1523300157   325000.0         2       0.75         1020      1076

       floors  waterfront  condition  grade  ...  grade_4  grade_5  grade_6  \
1         2.0         0.0          3      7  ...        0        0        0
2         1.0         0.0          3      6  ...        0        0        1
3         1.0         0.0          5      7  ...        0        0        0
4         1.0         0.0          3      8  ...        0        0        0
5         1.0         0.0          3     11  ...        0        0        0
...       ...         ...        ...    ...  ...      ...      ...      ...
21591     2.0         0.0          3      8  ...        0        0        0
21592     3.0         0.0          3      8  ...        0        0        0
21593     2.0         0.0          3      8  ...        0        0        0
21594     2.0         0.0          3      7  ...        0        0        0
21596     2.0         0.0          3      7  ...        0        0        0

       grade_7  grade_8  grade_9  grade_10  grade_11  grade_12  grade_13
1            1        0        0         0         0         0         0
2            0        0        0         0         0         0         0
3            1        0        0         0         0         0         0
4            0        1        0         0         0         0         0
5            0        0        0         0         1         0         0
...        ...      ...      ...       ...       ...       ...       ...
21591        0        1        0         0         0         0         0
21592        0        1        0         0         0         0         0
21593        0        1        0         0         0         0         0
21594        1        0        0         0         0         0         0
21596        1        0        0         0         0         0         0

[18860 rows x 62 columns]
```

In [46]:
```
# Print column names
print(df.columns)
```

```
Index(['id', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'condition', 'grade', 'yr_built_Group 2',
       'yr_built_Group 3', 'yr_built_Group 4', 'yr_built_Group 5',
       'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5', 'bedrooms_6',
       'bedrooms_7', 'bathrooms_0.75', 'bathrooms_1.0', 'bathrooms_1.25',
       'bathrooms_1.5', 'bathrooms_1.75', 'bathrooms_2.0', 'bathrooms_2.25',
       'bathrooms_2.5', 'bathrooms_2.75', 'bathrooms_3.0', 'bathrooms_3.25',
       'bathrooms_3.5', 'bathrooms_3.75', 'bathrooms_4.0', 'bathrooms_4.25',
       'bathrooms_4.5', 'bathrooms_4.75', 'bathrooms_5.0', 'bathrooms_5.25',
       'bathrooms_5.5', 'bathrooms_5.75', 'bathrooms_6.0', 'floors_1.5',
       'floors_2.0', 'floors_2.5', 'floors_3.0', 'floors_3.5',
       'waterfront_1.0', 'condition_2', 'condition_3', 'condition_4',
       'condition_5', 'grade_4', 'grade_5', 'grade_6', 'grade_7', 'grade_8',
       'grade_9', 'grade_10', 'grade_11', 'grade_12', 'grade_13'],
      dtype='object')
```

In [47]:
```
import statsmodels.api as sm

# Define the predictor variables
X = df[['sqft_living', 'sqft_lot', 'yr_built_Group 2',
       'yr_built_Group 3', 'yr_built_Group 4', 'yr_built_Group 5',
       'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5', 'bedrooms_6',
       'bedrooms_7', 'bathrooms_0.75', 'bathrooms_1.0', 'bathrooms_1.25',
       'bathrooms_1.5', 'bathrooms_1.75', 'bathrooms_2.0', 'bathrooms_2.25',
       'bathrooms_2.5', 'bathrooms_2.75', 'bathrooms_3.0', 'bathrooms_3.25',
       'bathrooms_3.5', 'bathrooms_3.75', 'bathrooms_4.0', 'bathrooms_4.25',
       'bathrooms_4.5', 'bathrooms_4.75', 'bathrooms_5.0', 'bathrooms_5.25',
       'bathrooms_5.5', 'bathrooms_5.75', 'bathrooms_6.0', 'floors_1.5',
       'floors_2.0', 'floors_2.5', 'floors_3.0', 'floors_3.5',
       'waterfront_1.0', 'condition_2', 'condition_3', 'condition_4',
       'condition_5', 'grade_4', 'grade_5', 'grade_6', 'grade_7', 'grade_8',
       'grade_9', 'grade_10', 'grade_11', 'grade_12', 'grade_13']]

# Add a constant term to the predictor variables
X = sm.add_constant(X)

# Define the dependent variable
y = df['price']

# Fit the multiple linear regression model
model = sm.OLS(y, X).fit()

# Print the model summary
print(model.summary())
```

OLS Regression Results

Dep. Variable:	price	R-squared:	0.686
Model:	OLS	Adj. R-squared:	0.685
Method:	Least Squares	F-statistic:	760.4
Date:	Mon, 26 Feb 2024	Prob (F-statistic):	0.00
Time:	03:48:54	Log-Likelihood:	-2.5710e+05
No. Observations:	18860	AIC:	5.143e+05
Df Residuals:	18805	BIC:	5.147e+05
Df Model:	54		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.087e+04	2.31e+05	0.177	0.860	-4.12e+05	4.94e+05
sqft_living	152.5730	3.597	42.421	0.000	145.523	159.623
sqft_lot	-1.2164	0.123	-9.919	0.000	-1.457	-0.976
yr_built_Group 2	-3.404e+04	6746.394	-5.046	0.000	-4.73e+04	-2.08e+04
yr_built_Group 3	-1.316e+05	6443.611	-20.425	0.000	-1.44e+05	-1.19e+05
yr_built_Group 4	-2.305e+05	6882.345	-33.489	0.000	-2.44e+05	-2.17e+05
yr_built_Group 5	-2.514e+05	7424.064	-33.867	0.000	-2.66e+05	-2.37e+05
bedrooms_2	-4415.0584	1.66e+04	-0.266	0.790	-3.7e+04	2.81e+04
bedrooms_3	-4.363e+04	1.67e+04	-2.618	0.009	-7.63e+04	-1.1e+04
bedrooms_4	-7.701e+04	1.7e+04	-4.522	0.000	-1.1e+05	-4.36e+04
bedrooms_5	-8.122e+04	1.79e+04	-4.528	0.000	-1.16e+05	-4.61e+04
bedrooms_6	-1.332e+05	2.22e+04	-6.014	0.000	-1.77e+05	-8.98e+04
bedrooms_7	-1.657e+05	3.99e+04	-4.153	0.000	-2.44e+05	-8.75e+04
bathrooms_0.75	8.033e+04	1.04e+05	0.770	0.441	-1.24e+05	2.85e+05
bathrooms_1.0	1.037e+05	1.01e+05	1.026	0.305	-9.44e+04	3.02e+05
bathrooms_1.25	1.43e+05	1.24e+05	1.156	0.248	-9.94e+04	3.85e+05
bathrooms_1.5	1.085e+05	1.01e+05	1.072	0.284	-8.99e+04	3.07e+05
bathrooms_1.75	1.181e+05	1.01e+05	1.167	0.243	-8.02e+04	3.16e+05
bathrooms_2.0	1.197e+05	1.01e+05	1.183	0.237	-7.86e+04	3.18e+05
bathrooms_2.25	1.5e+05	1.01e+05	1.482	0.138	-4.84e+04	3.48e+05
bathrooms_2.5	1.166e+05	1.01e+05	1.152	0.249	-8.17e+04	3.15e+05
bathrooms_2.75	1.376e+05	1.01e+05	1.358	0.174	-6.1e+04	3.36e+05
bathrooms_3.0	1.647e+05	1.01e+05	1.624	0.104	-3.41e+04	3.64e+05
bathrooms_3.25	2.296e+05	1.02e+05	2.260	0.024	3.05e+04	4.29e+05
bathrooms_3.5	1.809e+05	1.02e+05	1.782	0.075	-1.81e+04	3.8e+05
bathrooms_3.75	3.134e+05	1.03e+05	3.048	0.002	1.12e+05	5.15e+05
bathrooms_4.0	2.95e+05	1.03e+05	2.861	0.004	9.29e+04	4.97e+05
bathrooms_4.25	3.977e+05	1.04e+05	3.809	0.000	1.93e+05	6.02e+05
bathrooms_4.5	2.739e+05	1.04e+05	2.638	0.008	7.04e+04	4.77e+05
bathrooms_4.75	6.214e+05	1.11e+05	5.623	0.000	4.05e+05	8.38e+05
bathrooms_5.0	5.504e+05	1.14e+05	4.846	0.000	3.28e+05	7.73e+05
bathrooms_5.25	4.859e+05	1.2e+05	4.040	0.000	2.5e+05	7.22e+05
bathrooms_5.5	8.491e+05	1.26e+05	6.751	0.000	6.03e+05	1.1e+06
bathrooms_5.75	1.358e+06	1.8e+05	7.562	0.000	1.01e+06	1.71e+06
bathrooms_6.0	1.314e+06	1.46e+05	9.030	0.000	1.03e+06	1.6e+06
floors_1.5	1768.3333	6045.380	0.293	0.770	-1.01e+04	1.36e+04
floors_2.0	-3875.2680	4814.848	-0.805	0.421	-1.33e+04	5562.267
floors_2.5	9.827e+04	1.81e+04	5.443	0.000	6.29e+04	1.34e+05
floors_3.0	8.6e+04	1.02e+04	8.427	0.000	6.6e+04	1.06e+05
floors_3.5	7.552e+04	9.05e+04	0.835	0.404	-1.02e+05	2.53e+05
waterfront_1.0	6.927e+05	1.73e+04	39.966	0.000	6.59e+05	7.27e+05
condition_2	-6654.6048	4.47e+04	-0.149	0.882	-9.43e+04	8.1e+04
condition_3	7139.5171	4.16e+04	0.172	0.864	-7.44e+04	8.87e+04
condition_4	3.272e+04	4.16e+04	0.786	0.432	-4.89e+04	1.14e+05
condition_5	7.725e+04	4.19e+04	1.846	0.065	-4793.922	1.59e+05
grade_4	3.961e+04	2.07e+05	0.192	0.848	-3.65e+05	4.44e+05
grade_5	7612.2011	2.04e+05	0.037	0.970	-3.92e+05	4.08e+05
grade_6	6.53e+04	2.04e+05	0.320	0.749	-3.34e+05	4.65e+05
grade_7	1.699e+05	2.04e+05	0.833	0.405	-2.3e+05	5.7e+05
grade_8	2.758e+05	2.04e+05	1.352	0.176	-1.24e+05	6.76e+05
grade_9	4.306e+05	2.04e+05	2.110	0.035	3.07e+04	8.31e+05
grade_10	6.081e+05	2.04e+05	2.979	0.003	2.08e+05	1.01e+06
grade_11	8.116e+05	2.04e+05	3.969	0.000	4.11e+05	1.21e+06
grade_12	1.301e+06	2.06e+05	6.321	0.000	8.97e+05	1.7e+06
grade_13	1.744e+06	2.15e+05	8.097	0.000	1.32e+06	2.17e+06

Omnibus:	7710.065	Durbin-Watson:	1.982
Prob(Omnibus):	0.000	Jarque-Bera (JB):	136465.868
Skew:	1.509	Prob(JB):	0.00
Kurtosis:	15.828	Cond. No.	7.87e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.87e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

In [48]:
```
import statsmodels.api as sm

# Define the predictor variables
X = df[['sqft_living', 'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5', 'bedrooms_6',
       'bedrooms_7', 'bathrooms_0.75', 'bathrooms_1.0', 'bathrooms_1.25',
       'bathrooms_1.5', 'bathrooms_1.75', 'bathrooms_2.0', 'bathrooms_2.25',
       'bathrooms_2.5', 'bathrooms_2.75', 'bathrooms_3.0', 'bathrooms_3.25',
       'bathrooms_3.5', 'bathrooms_3.75', 'bathrooms_4.0', 'bathrooms_4.25',
       'bathrooms_4.5', 'bathrooms_4.75', 'bathrooms_5.0', 'bathrooms_5.25',
       'bathrooms_5.5', 'bathrooms_5.75', 'bathrooms_6.0', 'floors_1.5',
       'floors_2.0', 'floors_2.5', 'floors_3.0', 'floors_3.5',
       'waterfront_1.0', 'grade_4', 'grade_5', 'grade_6', 'grade_7', 'grade_8',
       'grade_9', 'grade_10', 'grade_11', 'grade_12', 'grade_13']]

# Add a constant term to the predictor variables
X = sm.add_constant(X)

# Define the dependent variable
y = df['price']

# Fit the multiple linear regression model
model = sm.OLS(y, X).fit()

# Print the model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.644
Model:                            OLS   Adj. R-squared:                  0.643
Method:                 Least Squares   F-statistic:                     756.8
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:48:55   Log-Likelihood:           -2.5828e+05
No. Observations:               18860   AIC:                            5.166e+05
Df Residuals:                   18814   BIC:                            5.170e+05
Df Model:                          45
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const           1.232e+05   2.42e+05      0.510      0.610    -3.5e+05    5.97e+05
sqft_living       161.1718      3.724     43.282      0.000     153.873     168.471
bedrooms_2     -2739.6418   1.76e+04     -0.155      0.877   -3.73e+04    3.18e+04
bedrooms_3     -6.374e+04   1.77e+04     -3.610      0.000   -9.83e+04   -2.91e+04
bedrooms_4     -8.498e+04    1.8e+04     -4.714      0.000    -1.2e+05   -4.96e+04
bedrooms_5     -8.368e+04    1.9e+04     -4.409      0.000   -1.21e+05   -4.65e+04
bedrooms_6     -1.135e+05   2.35e+04     -4.839      0.000   -1.59e+05   -6.75e+04
bedrooms_7     -1.464e+05   4.24e+04     -3.456      0.001   -2.29e+05   -6.34e+04
bathrooms_0.75  5.501e+04   1.11e+05      0.496      0.620   -1.62e+05    2.72e+05
bathrooms_1.0   9.661e+04   1.07e+05      0.899      0.369   -1.14e+05    3.07e+05
bathrooms_1.25  8.612e+04   1.32e+05      0.654      0.513   -1.72e+05    3.44e+05
bathrooms_1.5   8.035e+04   1.08e+05      0.746      0.456   -1.31e+05    2.91e+05
bathrooms_1.75  8.131e+04   1.08e+05      0.756      0.450    -1.3e+05    2.92e+05
bathrooms_2.0   8.536e+04   1.08e+05      0.793      0.428   -1.26e+05    2.96e+05
bathrooms_2.25  8.074e+04   1.08e+05      0.750      0.453    -1.3e+05    2.92e+05
bathrooms_2.5   2.759e+04   1.08e+05      0.256      0.798   -1.83e+05    2.39e+05
bathrooms_2.75  6.746e+04   1.08e+05      0.626      0.531   -1.44e+05    2.79e+05
bathrooms_3.0   9.863e+04   1.08e+05      0.914      0.361   -1.13e+05     3.1e+05
bathrooms_3.25  1.603e+05   1.08e+05      1.483      0.138   -5.16e+04    3.72e+05
bathrooms_3.5   9.122e+04   1.08e+05      0.844      0.398   -1.21e+05    3.03e+05
bathrooms_3.75  2.424e+05   1.09e+05      2.216      0.027     2.8e+04    4.57e+05
bathrooms_4.0    2.23e+05    1.1e+05      2.032      0.042    7921.300    4.38e+05
bathrooms_4.25  3.247e+05   1.11e+05      2.924      0.003    1.07e+05    5.42e+05
bathrooms_4.5   1.762e+05    1.1e+05      1.595      0.111   -4.03e+04    3.93e+05
bathrooms_4.75  5.485e+05   1.18e+05      4.665      0.000    3.18e+05    7.79e+05
bathrooms_5.0   4.304e+05   1.21e+05      3.563      0.000    1.94e+05    6.67e+05
bathrooms_5.25  3.813e+05   1.28e+05      2.981      0.003    1.31e+05    6.32e+05
bathrooms_5.5   7.359e+05   1.34e+05      5.501      0.000    4.74e+05    9.98e+05
bathrooms_5.75  1.266e+06   1.91e+05      6.630      0.000    8.92e+05    1.64e+06
bathrooms_6.0   1.194e+06   1.55e+05      7.715      0.000    8.91e+05     1.5e+06
floors_1.5      8.899e+04   5835.377     15.249      0.000    7.75e+04      1e+05
floors_2.0     -4.224e+04   4486.826     -9.414      0.000    -5.1e+04   -3.34e+04
floors_2.5       1.55e+05    1.9e+04      8.173      0.000    1.18e+05    1.92e+05
floors_3.0      2.733e+04   1.02e+04      2.691      0.007    7422.255    4.72e+04
floors_3.5      1.891e+04   9.62e+04      0.197      0.844    -1.7e+05    2.07e+05
waterfront_1.0  7.282e+05   1.84e+04     39.608      0.000    6.92e+05    7.64e+05
grade_4        -1.022e+05    2.2e+05     -0.465      0.642   -5.33e+05    3.28e+05
grade_5        -1.232e+05   2.17e+05     -0.568      0.570   -5.48e+05    3.02e+05
grade_6        -8.075e+04   2.17e+05     -0.372      0.710   -5.06e+05    3.44e+05
grade_7        -1.183e+04   2.17e+05     -0.055      0.956   -4.37e+05    4.13e+05
grade_8         7.795e+04   2.17e+05      0.359      0.719   -3.47e+05    5.03e+05
grade_9         2.215e+05   2.17e+05      1.021      0.307   -2.04e+05    6.47e+05
grade_10        3.845e+05   2.17e+05      1.771      0.077    -4.1e+04     8.1e+05
grade_11        5.648e+05   2.17e+05      2.598      0.009    1.39e+05    9.91e+05
grade_12        1.041e+06   2.19e+05      4.755      0.000    6.12e+05    1.47e+06
grade_13        1.488e+06   2.29e+05      6.496      0.000    1.04e+06    1.94e+06
==============================================================================
Omnibus:                     7562.277   Durbin-Watson:                   1.980
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           105726.554
Skew:                           1.542   Prob(JB):                         0.00
Kurtosis:                      14.181   Cond. No.                     1.04e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.04e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [49]:
```
print(df.columns)
```

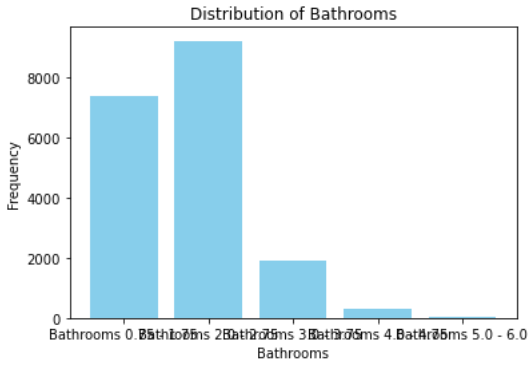
```
Index(['id', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'condition', 'grade', 'yr_built_Group 2',
       'yr_built_Group 3', 'yr_built_Group 4', 'yr_built_Group 5',
       'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5', 'bedrooms_6',
       'bedrooms_7', 'bathrooms_0.75', 'bathrooms_1.0', 'bathrooms_1.25',
       'bathrooms_1.5', 'bathrooms_1.75', 'bathrooms_2.0', 'bathrooms_2.25',
       'bathrooms_2.5', 'bathrooms_2.75', 'bathrooms_3.0', 'bathrooms_3.25',
       'bathrooms_3.5', 'bathrooms_3.75', 'bathrooms_4.0', 'bathrooms_4.25',
       'bathrooms_4.5', 'bathrooms_4.75', 'bathrooms_5.0', 'bathrooms_5.25',
       'bathrooms_5.5', 'bathrooms_5.75', 'bathrooms_6.0', 'floors_1.5',
       'floors_2.0', 'floors_2.5', 'floors_3.0', 'floors_3.5',
       'waterfront_1.0', 'condition_2', 'condition_3', 'condition_4',
       'condition_5', 'grade_4', 'grade_5', 'grade_6', 'grade_7', 'grade_8',
       'grade_9', 'grade_10', 'grade_11', 'grade_12', 'grade_13'],
      dtype='object')
```

In [50]:
```
# Define the bins for grouping bathrooms
bins = {
    'Bathrooms 0.75 - 1.75': ['bathrooms_0.75', 'bathrooms_1.0', 'bathrooms_1.25', 'bathrooms_1.5', 'bathrooms_1
    'Bathrooms 2.0 - 2.75': ['bathrooms_2.0', 'bathrooms_2.25', 'bathrooms_2.5', 'bathrooms_2.75'],
    'Bathrooms 3.0 - 3.75': ['bathrooms_3.0', 'bathrooms_3.25', 'bathrooms_3.5', 'bathrooms_3.75'],
    'Bathrooms 4.0 - 4.75': ['bathrooms_4.0', 'bathrooms_4.25', 'bathrooms_4.5', 'bathrooms_4.75'],
    'Bathrooms 5.0 - 6.0': ['bathrooms_5.0', 'bathrooms_5.25', 'bathrooms_5.5', 'bathrooms_5.75', 'bathrooms_6.0
}

# Iterate over the bins and create new columns
for bin_name, bin_vars in bins.items():
    df[bin_name] = df[bin_vars].sum(axis=1)
```

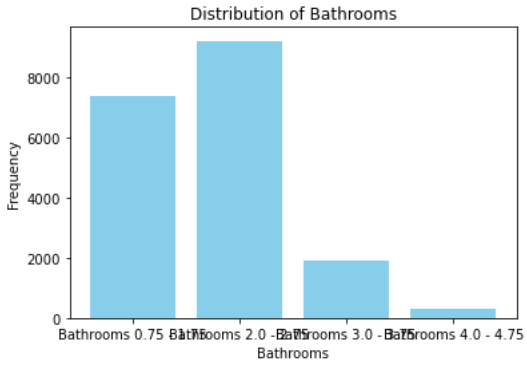
In [51]:
```
print (df.columns)
```

```
Index(['id', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'condition', 'grade', 'yr_built_Group 2',
       'yr_built_Group 3', 'yr_built_Group 4', 'yr_built_Group 5',
       'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5', 'bedrooms_6',
       'bedrooms_7', 'bathrooms_0.75', 'bathrooms_1.0', 'bathrooms_1.25',
       'bathrooms_1.5', 'bathrooms_1.75', 'bathrooms_2.0', 'bathrooms_2.25',
       'bathrooms_2.5', 'bathrooms_2.75', 'bathrooms_3.0', 'bathrooms_3.25',
       'bathrooms_3.5', 'bathrooms_3.75', 'bathrooms_4.0', 'bathrooms_4.25',
       'bathrooms_4.5', 'bathrooms_4.75', 'bathrooms_5.0', 'bathrooms_5.25',
       'bathrooms_5.5', 'bathrooms_5.75', 'bathrooms_6.0', 'floors_1.5',
       'floors_2.0', 'floors_2.5', 'floors_3.0', 'floors_3.5',
       'waterfront_1.0', 'condition_2', 'condition_3', 'condition_4',
       'condition_5', 'grade_4', 'grade_5', 'grade_6', 'grade_7', 'grade_8',
       'grade_9', 'grade_10', 'grade_11', 'grade_12', 'grade_13',
       'Bathrooms 0.75 - 1.75', 'Bathrooms 2.0 - 2.75', 'Bathrooms 3.0 - 3.75',
       'Bathrooms 4.0 - 4.75', 'Bathrooms 5.0 - 6.0'],
      dtype='object')
```

In [52]:
```
import matplotlib.pyplot as plt
import numpy as np

categories = ['Bathrooms 0.75 - 1.75', 'Bathrooms 2.0 - 2.75', 'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75', 'B
labels = np.arange(len(categories)) + 1  # Adjusted to start from 1

# Calculate the frequency of each category
category_counts = [df[col_name].sum() for col_name in categories]

# Create a bar plot
plt.bar(labels, category_counts, tick_label=categories, color='skyblue')

# Add labels and title
plt.xlabel('Bathrooms')
plt.ylabel('Frequency')
plt.title('Distribution of Bathrooms')

# Show plot
plt.show()
```



In [53]:
```
import matplotlib.pyplot as plt
import numpy as np

# Define the categories (excluding 'Bathrooms 5.0 - 6.0') and their corresponding integer labels
categories = ['Bathrooms 0.75 - 1.75', 'Bathrooms 2.0 - 2.75', 'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75']
labels = np.arange(len(categories)) + 1  # Adjusted to start from 1

# Calculate the frequency of each category
category_counts = [df[col_name].sum() for col_name in categories]

# Create a bar plot
plt.bar(labels, category_counts, tick_label=categories, color='skyblue')

# Add labels and title
plt.xlabel('Bathrooms')
plt.ylabel('Frequency')
plt.title('Distribution of Bathrooms')

# Show plot
plt.show()
```



In [54]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Define predictors (X) and target variable (y)
X = df[['Bathrooms 0.75 - 1.75', 'Bathrooms 2.0 - 2.75', 'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75']]
y = df['price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error:', mse)
```

```
Mean Squared Error: 92696109693.1135
```

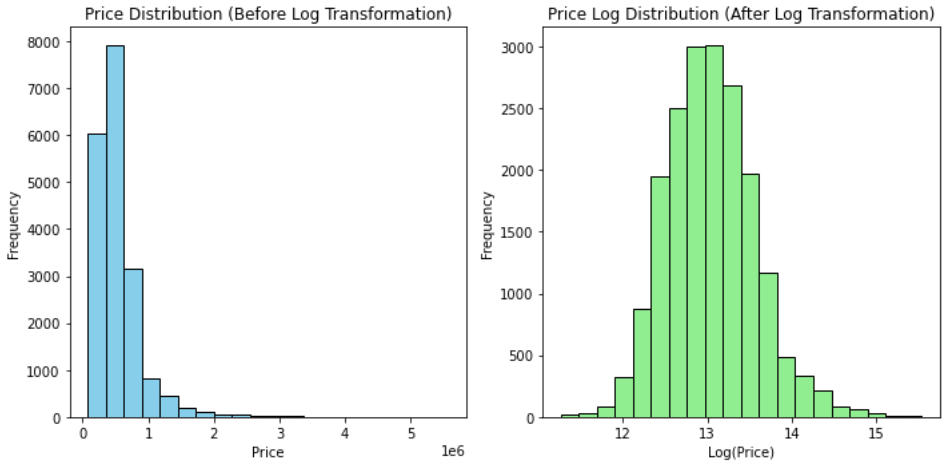
In [ ]:

In [ ]:

In [55]:
```
import numpy as np

# Log transformation of the price variable
df['price_log'] = np.log(df['price'])

# Check the first few rows to confirm the transformation
print(df[['price', 'price_log']].head())
```

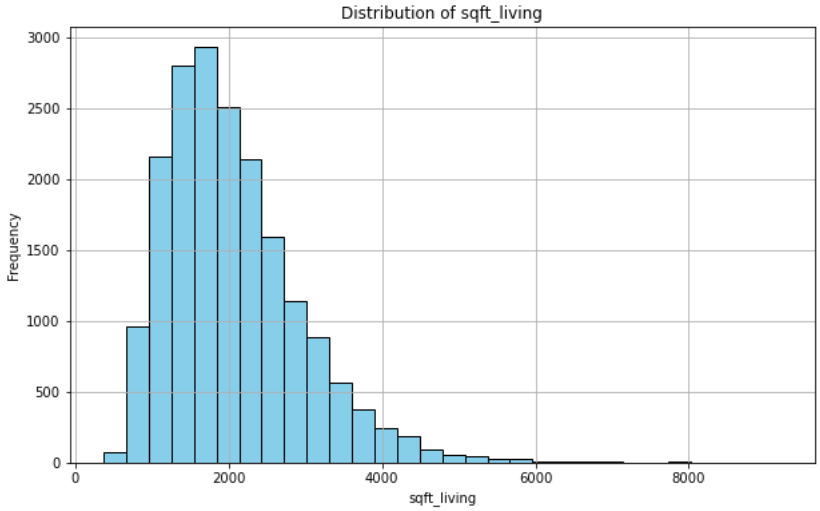
```
       price  price_log
1   538000.0  13.195614
2   180000.0  12.100712
3   604000.0  13.311329
4   510000.0  13.142166
5  1230000.0  14.022525
```

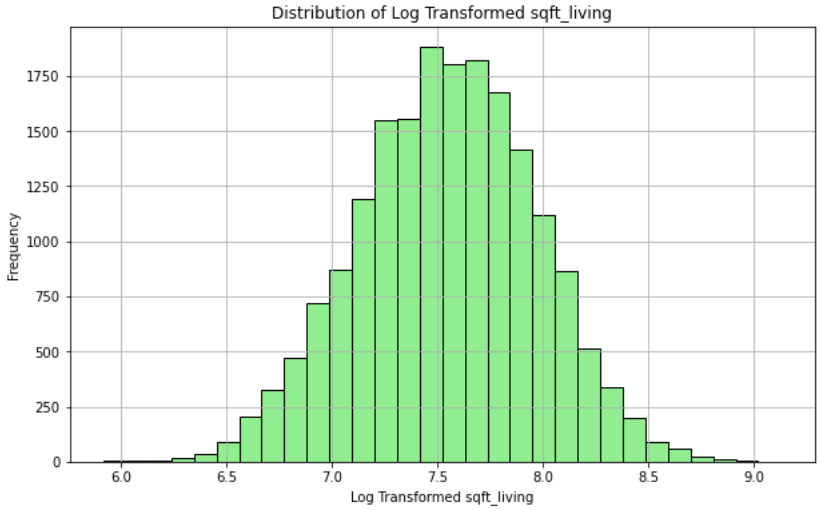
In [56]:
```
import matplotlib.pyplot as plt

# Before log transformation
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.hist(df['price'], bins=20, color='skyblue', edgecolor='black')
plt.title('Price Distribution (Before Log Transformation)')
plt.xlabel('Price')
plt.ylabel('Frequency')

# After log transformation
plt.subplot(1, 2, 2)
plt.hist(df['price_log'], bins=20, color='lightgreen', edgecolor='black')
plt.title('Price Log Distribution (After Log Transformation)')
plt.xlabel('Log(Price)')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



In [ ]:

In [57]:
```
import numpy as np
import matplotlib.pyplot as plt

# Visualize the distribution of sqft_living
plt.figure(figsize=(10, 6))
plt.hist(df['sqft_living'], bins=30, color='skyblue', edgecolor='black')
plt.title('Distribution of sqft_living')
plt.xlabel('sqft_living')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

# Apply log transformation to sqft_living
df['sqft_living_log'] = np.log1p(df['sqft_living'])

# Visualize the transformed distribution
plt.figure(figsize=(10, 6))
plt.hist(df['sqft_living_log'], bins=30, color='lightgreen', edgecolor='black')
plt.title('Distribution of Log Transformed sqft_living')
plt.xlabel('Log Transformed sqft_living')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```





In [58]:
```
import pandas as pd

# Create boolean columns based on conditions
df['bedrooms_less_than_8_true'] = df['bedrooms'] < 8
df['bedrooms_less_than_8_false'] = df['bedrooms'] >= 8
df['grade_less_than_8_true'] = df['grade'] < 8
df['grade_less_than_8_false'] = df['grade'] >= 8

# Convert boolean columns to numerical (0 and 1)
df['bedrooms_less_than_8_true'] = df['bedrooms_less_than_8_true'].astype(int)
df['bedrooms_less_than_8_false'] = df['bedrooms_less_than_8_false'].astype(int)
df['grade_less_than_8_true'] = df['grade_less_than_8_true'].astype(int)
df['grade_less_than_8_false'] = df['grade_less_than_8_false'].astype(int)

# Print the DataFrame with the new columns
print(df)
```

```
              id      price  bedrooms  bathrooms  sqft_living  sqft_lot  \
1      6414100192   538000.0         3       2.25         2570      7242
2      5631500400   180000.0         2       1.00          770     10000
3      2487200875   604000.0         4       3.00         1960      5000
4      1954400510   510000.0         3       2.00         1680      8080
5      7237550310  1230000.0         4       4.50         5420    101930
...           ...        ...       ...        ...          ...       ...
21591  2997800021   475000.0         3       2.50         1310      1294
21592   263000018   360000.0         3       2.50         1530      1131
21593  6600060120   400000.0         4       2.50         2310      5813
21594  1523300141   402101.0         2       0.75         1020      1350
21596  1523300157   325000.0         2       0.75         1020      1076

       floors  waterfront  condition  grade  ...  Bathrooms 2.0 - 2.75  \
1         2.0         0.0          3      7  ...                     1
2         1.0         0.0          3      6  ...                     0
3         1.0         0.0          5      7  ...                     0
4         1.0         0.0          3      8  ...                     1
5         1.0         0.0          3     11  ...                     0
...       ...         ...        ...    ...  ...                   ...
21591     2.0         0.0          3      8  ...                     1
21592     3.0         0.0          3      8  ...                     1
21593     2.0         0.0          3      8  ...                     1
21594     2.0         0.0          3      7  ...                     0
21596     2.0         0.0          3      7  ...                     0

       Bathrooms 3.0 - 3.75  Bathrooms 4.0 - 4.75  Bathrooms 5.0 - 6.0  \
1                         0                     0                    0
2                         0                     0                    0
3                         1                     0                    0
4                         0                     0                    0
5                         0                     1                    0
...                     ...                   ...                  ...
21591                     0                     0                    0
21592                     0                     0                    0
21593                     0                     0                    0
21594                     0                     0                    0
21596                     0                     0                    0

       price_log  sqft_living_log  bedrooms_less_than_8_true  \
1      13.195614         7.852050                          1
2      12.100712         6.647688                          1
3      13.311329         7.581210                          1
4      13.142166         7.427144                          1
5      14.022525         8.598036                          1
...          ...              ...                        ...
21591  13.071070         7.178545                          1
21592  12.793859         7.333676                          1
21593  12.899220         7.745436                          1
21594  12.904459         6.928538                          1
21596  12.691580         6.928538                          1

       bedrooms_less_than_8_false  grade_less_than_8_true  \
1                               0                       1
2                               0                       1
3                               0                       1
4                               0                       0
5                               0                       0
...                           ...                     ...
21591                           0                       0
21592                           0                       0
21593                           0                       0
21594                           0                       1
21596                           0                       1

       grade_less_than_8_false
1                            0
2                            0
3                            0
4                            1
5                            1
...                        ...
21591                        1
21592                        1
21593                        1
21594                        0
21596                        0

[18860 rows x 73 columns]
```

In [59]:

```
df_encoded = pd.get_dummies(df, columns=['bedrooms_less_than_8_true', 'bedrooms_less_than_8_false', 'grade_less_

# Print the DataFrame with the new columns
print(df_encoded)
```

```
              id      price  bedrooms  bathrooms  sqft_living  sqft_lot  \
1     6414100192   538000.0         3       2.25         2570      7242
2     5631500400   180000.0         2       1.00          770     10000
3     2487200875   604000.0         4       3.00         1960      5000
4     1954400510   510000.0         3       2.00         1680      8080
5     7237550310  1230000.0         4       4.50         5420    101930
...          ...        ...       ...        ...          ...       ...
21591 2997800021   475000.0         3       2.50         1310      1294
21592  263000018   360000.0         3       2.50         1530      1131
21593 6600060120   400000.0         4       2.50         2310      5813
21594 1523300141   402101.0         2       0.75         1020      1350
21596 1523300157   325000.0         2       0.75         1020      1076

       floors  waterfront  condition  grade  ...  grade_13  \
1         2.0         0.0          3      7  ...         0
2         1.0         0.0          3      6  ...         0
3         1.0         0.0          5      7  ...         0
4         1.0         0.0          3      8  ...         0
5         1.0         0.0          3     11  ...         0
...       ...         ...        ...    ...  ...       ...
21591     2.0         0.0          3      8  ...         0
21592     3.0         0.0          3      8  ...         0
21593     2.0         0.0          3      8  ...         0
21594     2.0         0.0          3      7  ...         0
21596     2.0         0.0          3      7  ...         0

       Bathrooms 0.75 - 1.75  Bathrooms 2.0 - 2.75  Bathrooms 3.0 - 3.75  \
1                          0                     1                     0
2                          1                     0                     0
3                          0                     0                     1
4                          0                     1                     0
5                          0                     0                     0
...                      ...                   ...                   ...
21591                      0                     1                     0
21592                      0                     1                     0
21593                      0                     1                     0
21594                      1                     0                     0
21596                      1                     0                     0

       Bathrooms 4.0 - 4.75  Bathrooms 5.0 - 6.0  price_log  sqft_living_log  \
1                         0                    0  13.195614         7.852050
2                         0                    0  12.100712         6.647688
3                         0                    0  13.311329         7.581210
4                         0                    0  13.142166         7.427144
5                         1                    0  14.022525         8.598036
...                     ...                  ...        ...              ...
21591                     0                    0  13.071070         7.178545
21592                     0                    0  12.793859         7.333676
21593                     0                    0  12.899220         7.745436
21594                     0                    0  12.904459         6.928538
21596                     0                    0  12.691580         6.928538

       grade_less_than_8_1  grade_8_and_above_1
1                        1                    0
2                        1                    0
3                        1                    0
4                        0                    1
5                        0                    1
...                    ...                  ...
21591                    0                    1
21592                    0                    1
21593                    0                    1
21594                    1                    0
21596                    1                    0

[18860 rows x 71 columns]
```

In [ ]:

In [60]:
```
import pandas as pd
import statsmodels.api as sm

# Assuming df is your DataFrame containing the data
X = df[['floors_2.0', 'floors_2.5', 'floors_3.0', 'floors_3.5',
        'waterfront_1.0', 'condition_2', 'condition_3', 'condition_4',
        'condition_5', 'grade_4', 'grade_5', 'grade_6', 'grade_7', 'grade_8',
        'grade_9', 'grade_10', 'grade_11', 'grade_12', 'grade_13',
        'sqft_living_log', 'Bathrooms 0.75 - 1.75', 'Bathrooms 2.0 - 2.75',
        'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75', 'Bathrooms 5.0 - 6.0',
        'yr_built_Group 2', 'yr_built_Group 3', 'yr_built_Group 4', 'yr_built_Group 5',
        'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5', 'bedrooms_6', 'bedrooms_7']]
y = df['price_log']

# Add constant term for intercept
X = sm.add_constant(X)

# Fit OLS regression model
model = sm.OLS(y, X).fit()

# Print regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.646
Model:                            OLS   Adj. R-squared:                  0.646
Method:                 Least Squares   F-statistic:                     982.7
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:04   Log-Likelihood:                -4815.5
No. Observations:               18860   AIC:                             9703.
Df Residuals:                   18824   BIC:                             9985.
Df Model:                          35
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                            8.9780      0.362     24.783      0.000       8.268       9.688
floors_2.0                      -0.0033      0.007     -0.467      0.641      -0.017       0.011
floors_2.5                       0.0445      0.028      1.605      0.109      -0.010       0.099
floors_3.0                       0.1733      0.016     11.114      0.000       0.143       0.204
floors_3.5                       0.1475      0.140      1.052      0.293      -0.127       0.422
waterfront_1.0                   0.5396      0.027     20.166      0.000       0.487       0.592
condition_2                     -0.0346      0.069     -0.500      0.617      -0.170       0.101
condition_3                      0.1039      0.064      1.612      0.107      -0.022       0.230
condition_4                      0.1431      0.065      2.219      0.027       0.017       0.270
condition_5                      0.2121      0.065      3.271      0.001       0.085       0.339
grade_4                         -0.1498      0.319     -0.469      0.639      -0.776       0.476
grade_5                         -0.1535      0.314     -0.488      0.625      -0.769       0.462
grade_6                          0.0463      0.314      0.147      0.883      -0.569       0.661
grade_7                          0.3235      0.314      1.031      0.303      -0.292       0.939
grade_8                          0.5618      0.314      1.789      0.074      -0.054       1.177
grade_9                          0.8146      0.314      2.593      0.010       0.199       1.430
grade_10                         1.0114      0.314      3.218      0.001       0.395       1.627
grade_11                         1.1835      0.315      3.761      0.000       0.567       1.800
grade_12                         1.4228      0.317      4.494      0.000       0.802       2.043
grade_13                         1.6551      0.330      5.010      0.000       1.008       2.303
sqft_living_log                  0.4719      0.011     43.548      0.000       0.451       0.493
Bathrooms 0.75 - 1.75            0.3338      0.157      2.132      0.033       0.027       0.641
Bathrooms 2.0 - 2.75             0.3680      0.157      2.349      0.019       0.061       0.675
Bathrooms 3.0 - 3.75             0.4614      0.157      2.941      0.003       0.154       0.769
Bathrooms 4.0 - 4.75             0.5364      0.158      3.394      0.001       0.227       0.846
Bathrooms 5.0 - 6.0              0.6116      0.165      3.696      0.000       0.287       0.936
yr_built_Group 2                -0.0691      0.010     -6.634      0.000      -0.090      -0.049
yr_built_Group 3                -0.2840      0.009    -30.588      0.000      -0.302      -0.266
yr_built_Group 4                -0.4347      0.010    -44.257      0.000      -0.454      -0.415
yr_built_Group 5                -0.4461      0.011    -40.901      0.000      -0.467      -0.425
bedrooms_2                      -0.0397      0.025     -1.558      0.119      -0.090       0.010
bedrooms_3                      -0.1486      0.026     -5.795      0.000      -0.199      -0.098
bedrooms_4                      -0.1782      0.026     -6.761      0.000      -0.230      -0.127
bedrooms_5                      -0.1759      0.028     -6.329      0.000      -0.230      -0.121
bedrooms_6                      -0.2123      0.034     -6.198      0.000      -0.279      -0.145
bedrooms_7                      -0.2194      0.061     -3.578      0.000      -0.340      -0.099
==============================================================================
Omnibus:                       54.872   Durbin-Watson:                   1.977
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               64.066
Skew:                          -0.074   Prob(JB):                     1.23e-14
Kurtosis:                       3.245   Cond. No.                     3.55e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.55e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [61]:
```
import pandas as pd
import statsmodels.api as sm

# Assuming df is your DataFrame containing the data
X = df[['floors_2.0', 'floors_2.5', 'floors_3.0', 'floors_3.5',
        'waterfront_1.0', 'condition_2', 'condition_3', 'condition_4',
        'condition_5', 'grade_4', 'grade_5', 'grade_6', 'grade_7', 'grade_8',
        'grade_9', 'grade_10', 'grade_11', 'grade_12', 'grade_13',
        'sqft_living_log', 'Bathrooms 0.75 - 1.75', 'Bathrooms 2.0 - 2.75',
        'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75', 'Bathrooms 5.0 - 6.0',
        'yr_built_Group 2', 'yr_built_Group 3', 'yr_built_Group 4', 'yr_built_Group 5',
        'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5', 'bedrooms_6', 'bedrooms_7']]
y = df['price_log']

# Add constant term for intercept
X = sm.add_constant(X)

# Fit OLS regression model
model = sm.OLS(y, X).fit()

# Print regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.646
Model:                            OLS   Adj. R-squared:                  0.646
Method:                 Least Squares   F-statistic:                     982.7
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:05   Log-Likelihood:                -4815.5
No. Observations:               18860   AIC:                             9703.
Df Residuals:                   18824   BIC:                             9985.
Df Model:                          35
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                        8.9780      0.362     24.783      0.000       8.268       9.688
floors_2.0                  -0.0033      0.007     -0.467      0.641      -0.017       0.011
floors_2.5                   0.0445      0.028      1.605      0.109      -0.010       0.099
floors_3.0                   0.1733      0.016     11.114      0.000       0.143       0.204
floors_3.5                   0.1475      0.140      1.052      0.293      -0.127       0.422
waterfront_1.0               0.5396      0.027     20.166      0.000       0.487       0.592
condition_2                 -0.0346      0.069     -0.500      0.617      -0.170       0.101
condition_3                  0.1039      0.064      1.612      0.107      -0.022       0.230
condition_4                  0.1431      0.065      2.219      0.027       0.017       0.270
condition_5                  0.2121      0.065      3.271      0.001       0.085       0.339
grade_4                     -0.1498      0.319     -0.469      0.639      -0.776       0.476
grade_5                     -0.1535      0.314     -0.488      0.625      -0.769       0.462
grade_6                      0.0463      0.314      0.147      0.883      -0.569       0.661
grade_7                      0.3235      0.314      1.031      0.303      -0.292       0.939
grade_8                      0.5618      0.314      1.789      0.074      -0.054       1.177
grade_9                      0.8146      0.314      2.593      0.010       0.199       1.430
grade_10                     1.0114      0.314      3.218      0.001       0.395       1.627
grade_11                     1.1835      0.315      3.761      0.000       0.567       1.800
grade_12                     1.4228      0.317      4.494      0.000       0.802       2.043
grade_13                     1.6551      0.330      5.010      0.000       1.008       2.303
sqft_living_log              0.4719      0.011     43.548      0.000       0.451       0.493
Bathrooms 0.75 - 1.75        0.3338      0.157      2.132      0.033       0.027       0.641
Bathrooms 2.0 - 2.75         0.3680      0.157      2.349      0.019       0.061       0.675
Bathrooms 3.0 - 3.75         0.4614      0.157      2.941      0.003       0.154       0.769
Bathrooms 4.0 - 4.75         0.5364      0.158      3.394      0.001       0.227       0.846
Bathrooms 5.0 - 6.0          0.6116      0.165      3.696      0.000       0.287       0.936
yr_built_Group 2            -0.0691      0.010     -6.634      0.000      -0.090      -0.049
yr_built_Group 3            -0.2840      0.009    -30.588      0.000      -0.302      -0.266
yr_built_Group 4            -0.4347      0.010    -44.257      0.000      -0.454      -0.415
yr_built_Group 5            -0.4461      0.011    -40.901      0.000      -0.467      -0.425
bedrooms_2                  -0.0397      0.025     -1.558      0.119      -0.090       0.010
bedrooms_3                  -0.1486      0.026     -5.795      0.000      -0.199      -0.098
bedrooms_4                  -0.1782      0.026     -6.761      0.000      -0.230      -0.127
bedrooms_5                  -0.1759      0.028     -6.329      0.000      -0.230      -0.121
bedrooms_6                  -0.2123      0.034     -6.198      0.000      -0.279      -0.145
bedrooms_7                  -0.2194      0.061     -3.578      0.000      -0.340      -0.099
==============================================================================
Omnibus:                       54.872   Durbin-Watson:                   1.977
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               64.066
Skew:                          -0.074   Prob(JB):                     1.23e-14
Kurtosis:                       3.245   Cond. No.                     3.55e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.55e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [62]:
```
import statsmodels.api as sm

# Define the independent variables
independent_vars = ['Bathrooms 0.75 - 1.75', 'Bathrooms 2.0 - 2.75', 'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.

# Add a constant term for the intercept
X = sm.add_constant(df[independent_vars])

# Define the dependent variable
y = df['price_log']

# Create and fit the OLS model
model = sm.OLS(y, X)
results = model.fit()

# Print the summary of the model
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.255
Model:                            OLS   Adj. R-squared:                  0.255
Method:                 Least Squares   F-statistic:                     1616.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:05   Log-Likelihood:                -11836.
No. Observations:               18860   AIC:                         2.368e+04
Df Residuals:                   18855   BIC:                         2.372e+04
Df Model:                           4
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                       14.1935      0.068    207.706      0.000      14.060      14.327
Bathrooms 0.75 - 1.75       -1.4026      0.069    -20.465      0.000      -1.537      -1.268
Bathrooms 2.0 - 2.75        -1.0880      0.068    -15.884      0.000      -1.222      -0.954
Bathrooms 3.0 - 3.75        -0.6418      0.069     -9.285      0.000      -0.777      -0.506
Bathrooms 4.0 - 4.75        -0.1755      0.073     -2.391      0.017      -0.319      -0.032
==============================================================================
Omnibus:                      102.886   Durbin-Watson:                   1.974
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              105.676
Skew:                           0.170   Prob(JB):                     1.13e-23
Kurtosis:                       3.136   Cond. No.                         55.3
==============================================================================

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

In [63]:
```
import pandas as pd


correlation = df[['Bathrooms 0.75 - 1.75', 'Bathrooms 2.0 - 2.75',
                  'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75',
                  'Bathrooms 5.0 - 6.0', 'price_log']].corr()

# Selecting the correlation coefficient between 'Bathrooms' and 'price'
bathrooms_price_corr = correlation.loc['price_log', :'Bathrooms 5.0 - 6.0']

# Printing the correlation coefficient value
print("Correlation coefficient between Bathrooms and price_log:")
print(bathrooms_price_corr)
```

```
Correlation coefficient between Bathrooms and price_log:
Bathrooms 0.75 - 1.75   -0.386334
Bathrooms 2.0 - 2.75     0.115781
Bathrooms 3.0 - 3.75     0.323956
Bathrooms 4.0 - 4.75     0.230272
Bathrooms 5.0 - 6.0      0.117537
Name: price_log, dtype: float64
```

In [ ]:

In [64]:
```
import statsmodels.api as sm

# Define the independent variable
X = sm.add_constant(df['sqft_living_log'])

# Define the dependent variable
y = df['price_log']

# Create and fit the OLS model
model = sm.OLS(y, X)
results = model.fit()

# Print the summary of the model
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.448
Model:                            OLS   Adj. R-squared:                  0.448
Method:                 Least Squares   F-statistic:                 1.528e+04
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:05   Log-Likelihood:                -9019.0
No. Observations:               18860   AIC:                             1.804e+04
Df Residuals:                   18858   BIC:                             1.806e+04
Df Model:                           1
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const              6.7402      0.051    131.983      0.000       6.640       6.840
sqft_living_log    0.8354      0.007    123.617      0.000       0.822       0.849
==============================================================================
Omnibus:                      117.618   Durbin-Watson:                   1.989
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              105.701
Skew:                           0.140   Prob(JB):                     1.12e-23
Kurtosis:                       2.764   Cond. No.                         138.
==============================================================================

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

In [ ]:
```
```

In [65]:
```
import pandas as pd
import statsmodels.api as sm

# Assuming df is your DataFrame containing the data
X = df[[  'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75', 'grade_8','grade_9','grade_10', 'grade_11', 'grade_12'
y = df['price_log']

# Add constant term for intercept
X = sm.add_constant(X)

# Fit OLS regression model
model = sm.OLS(y, X).fit()

# Print regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.542
Model:                            OLS   Adj. R-squared:                  0.541
Method:                 Least Squares   F-statistic:                     2474.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:05   Log-Likelihood:                -7260.5
No. Observations:               18860   AIC:                         1.454e+04
Df Residuals:                   18850   BIC:                         1.462e+04
Df Model:                           9
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                        9.4019      0.064    146.793      0.000       9.276       9.527
Bathrooms 3.0 - 3.75         0.0912      0.010      9.556      0.000       0.072       0.110
Bathrooms 4.0 - 4.75         0.1917      0.023      8.357      0.000       0.147       0.237
grade_8                      0.1983      0.007     29.523      0.000       0.185       0.211
grade_9                      0.4152      0.010     41.931      0.000       0.396       0.435
grade_10                     0.6100      0.014     43.045      0.000       0.582       0.638
grade_11                     0.7849      0.022     35.026      0.000       0.741       0.829
grade_12                     1.0961      0.044     25.054      0.000       1.010       1.182
grade_13                     1.3671      0.113     12.059      0.000       1.145       1.589
sqft_living_log              0.4604      0.009     52.562      0.000       0.443       0.478
==============================================================================
Omnibus:                       60.679   Durbin-Watson:                   1.982
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               57.696
Skew:                           0.111   Prob(JB):                     2.96e-13
Kurtosis:                       2.844   Cond. No.                         335.
==============================================================================

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

In [66]:
```
import pandas as pd
import statsmodels.api as sm

# Assuming df is your DataFrame containing the data
X = df[[ 'waterfront_1.0', 'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75', 'grade_8','grade_9','grade_10', 'grad
y = df['price_log']

# Add constant term for intercept
X = sm.add_constant(X)

# Fit OLS regression model
model = sm.OLS(y, X).fit()

# Print regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.553
Model:                            OLS   Adj. R-squared:                  0.552
Method:                 Least Squares   F-statistic:                     2328.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:06   Log-Likelihood:                -7030.4
No. Observations:               18860   AIC:                         1.408e+04
Df Residuals:                   18849   BIC:                         1.417e+04
Df Model:                          10
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                        9.4297      0.063    149.000      0.000       9.306       9.554
waterfront_1.0               0.6445      0.030     21.577      0.000       0.586       0.703
Bathrooms 3.0 - 3.75         0.0914      0.009      9.700      0.000       0.073       0.110
Bathrooms 4.0 - 4.75         0.1775      0.023      7.830      0.000       0.133       0.222
grade_8                      0.1982      0.007     29.871      0.000       0.185       0.211
grade_9                      0.4138      0.010     42.304      0.000       0.395       0.433
grade_10                     0.5983      0.014     42.705      0.000       0.571       0.626
grade_11                     0.7597      0.022     34.271      0.000       0.716       0.803
grade_12                     1.0264      0.043     23.683      0.000       0.941       1.111
grade_13                     1.3795      0.112     12.318      0.000       1.160       1.599
sqft_living_log              0.4563      0.009     52.719      0.000       0.439       0.473
==============================================================================
Omnibus:                       55.369   Durbin-Watson:                   1.978
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               47.702
Skew:                           0.069   Prob(JB):                     4.38e-11
Kurtosis:                       2.796   Cond. No.                         335.
==============================================================================

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

In [67]:
```
import pandas as pd
import statsmodels.api as sm

# Assuming df is your DataFrame containing the data
X = df[['bedrooms_3', 'bedrooms_4', 'bedrooms_5', 'bedrooms_6', 'bedrooms_7', 'waterfront_1.0', 'Bathrooms 3.0 -
y = df['price_log']

# Add constant term for intercept
X = sm.add_constant(X)

# Fit OLS regression model
model = sm.OLS(y, X).fit()

# Print regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.559
Model:                            OLS   Adj. R-squared:                  0.559
Method:                 Least Squares   F-statistic:                     1593.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:06   Log-Likelihood:                -6894.8
No. Observations:               18860   AIC:                         1.382e+04
Df Residuals:                   18844   BIC:                         1.395e+04
Df Model:                          15
Covariance Type:            nonrobust
============================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                      8.9085      0.076    117.474      0.000       8.760       9.057
bedrooms_3                -0.1356      0.009    -15.631      0.000      -0.153      -0.119
bedrooms_4                -0.1608      0.010    -15.363      0.000      -0.181      -0.140
bedrooms_5                -0.1410      0.014     -9.992      0.000      -0.169      -0.113
bedrooms_6                -0.1411      0.026     -5.369      0.000      -0.193      -0.090
bedrooms_7                -0.1303      0.062     -2.089      0.037      -0.252      -0.008
waterfront_1.0             0.6126      0.030     20.596      0.000       0.554       0.671
Bathrooms 3.0 - 3.75       0.0851      0.010      8.950      0.000       0.066       0.104
Bathrooms 4.0 - 4.75       0.1544      0.023      6.700      0.000       0.109       0.200
grade_8                    0.1898      0.007     28.580      0.000       0.177       0.203
grade_9                    0.3942      0.010     39.940      0.000       0.375       0.413
grade_10                   0.5658      0.014     39.874      0.000       0.538       0.594
grade_11                   0.7175      0.022     32.134      0.000       0.674       0.761
grade_12                   0.9690      0.043     22.390      0.000       0.884       1.054
grade_13                   1.2925      0.111     11.606      0.000       1.074       1.511
sqft_living_log            0.5431      0.011     50.167      0.000       0.522       0.564
==============================================================================
Omnibus:                       31.796   Durbin-Watson:                   1.977
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               28.879
Skew:                           0.058   Prob(JB):                     5.36e-07
Kurtosis:                       2.848   Cond. No.                         337.
==============================================================================

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

In [ ]:

In [68]:
```
import pandas as pd
import statsmodels.api as sm

# Assuming df is your DataFrame containing the data
X = df[[ 'bedrooms', 'waterfront_1.0', 'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75', 'grade_8','grade_9','grad
y = df['price_log']

# Add constant term for intercept
X = sm.add_constant(X)

# Fit OLS regression model
model = sm.OLS(y, X).fit()

# Print regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.555
Model:                            OLS   Adj. R-squared:                  0.554
Method:                 Least Squares   F-statistic:                     2134.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:06   Log-Likelihood:                -6988.3
No. Observations:               18860   AIC:                         1.400e+04
Df Residuals:                   18848   BIC:                         1.409e+04
Df Model:                          11
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                        9.1143      0.072    126.794      0.000       8.973       9.255
bedrooms                    -0.0359      0.004     -9.183      0.000      -0.044      -0.028
waterfront_1.0               0.6264      0.030     20.971      0.000       0.568       0.685
Bathrooms 3.0 - 3.75         0.0998      0.009     10.563      0.000       0.081       0.118
Bathrooms 4.0 - 4.75         0.1964      0.023      8.645      0.000       0.152       0.241
grade_8                      0.1906      0.007     28.561      0.000       0.178       0.204
grade_9                      0.3995      0.010     40.421      0.000       0.380       0.419
grade_10                     0.5737      0.014     40.315      0.000       0.546       0.602
grade_11                     0.7282      0.022     32.530      0.000       0.684       0.772
grade_12                     0.9873      0.043     22.719      0.000       0.902       1.072
grade_13                     1.3392      0.112     11.975      0.000       1.120       1.558
sqft_living_log              0.5147      0.011     47.981      0.000       0.494       0.536
==============================================================================
Omnibus:                       53.700   Durbin-Watson:                   1.977
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               46.593
Skew:                           0.070   Prob(JB):                     7.63e-11
Kurtosis:                       2.801   Cond. No.                         368.
==============================================================================

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

In [69]:
```
import pandas as pd
import statsmodels.api as sm

# Print the number of observations before filtering
print("Number of observations before filtering:", len(df))

# Filter out rows where the number of bedrooms is equal to 1
filtered_df = df[df['bedrooms'] != 1]

# Print the number of observations after filtering
print("Number of observations after filtering:", len(filtered_df))

# Select the independent variables including 'bedrooms' (excluding 1)
X = filtered_df[['bedrooms', 'waterfront_1.0', 'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75',
                 'grade_8', 'grade_9', 'grade_10', 'grade_11', 'grade_12', 'grade_13',
                 'sqft_living_log']]

# Define the dependent variable
y = filtered_df['price_log']

# Add constant term for intercept
X = sm.add_constant(X)

# Fit OLS regression model
model = sm.OLS(y, X).fit()

# Print regression results
print(model.summary())
```

```
Number of observations before filtering: 18860
Number of observations after filtering: 18688
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.554
Model:                            OLS   Adj. R-squared:                  0.554
Method:                 Least Squares   F-statistic:                     2108.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:07   Log-Likelihood:                -6883.2
No. Observations:               18688   AIC:                         1.379e+04
Df Residuals:                   18676   BIC:                         1.388e+04
Df Model:                          11
Covariance Type:            nonrobust
===========================================================================================
                              coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------------
const                       9.1053      0.073    125.071      0.000       8.963       9.248
bedrooms                   -0.0364      0.004     -9.182      0.000      -0.044      -0.029
waterfront_1.0              0.6304      0.030     20.848      0.000       0.571       0.690
Bathrooms 3.0 - 3.75        0.0997      0.009     10.558      0.000       0.081       0.118
Bathrooms 4.0 - 4.75        0.1962      0.023      8.651      0.000       0.152       0.241
grade_8                     0.1907      0.007     28.565      0.000       0.178       0.204
grade_9                     0.3991      0.010     40.348      0.000       0.380       0.419
grade_10                    0.5730      0.014     40.252      0.000       0.545       0.601
grade_11                    0.7272      0.022     32.509      0.000       0.683       0.771
grade_12                    0.9858      0.043     22.719      0.000       0.901       1.071
grade_13                    1.3380      0.112     11.989      0.000       1.119       1.557
sqft_living_log             0.5162      0.011     47.672      0.000       0.495       0.537
==============================================================================
Omnibus:                       58.300   Durbin-Watson:                   1.976
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               51.209
Skew:                           0.080   Prob(JB):                     7.59e-12
Kurtosis:                       2.800   Cond. No.                         367.
==============================================================================

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

In [70]:
```
import statsmodels.api as sm


# Define predictor (X) and target variable (y)
X = df[['bedrooms', 'grade_8','grade_9','grade_10', 'grade_11', 'grade_12', 'grade_13', 'sqft_living_log']]
y = df['price_log']

# Add constant term for intercept
X = sm.add_constant(X)

# Fit OLS regression model
model = sm.OLS(y, X).fit()

# Print regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.540
Model:                            OLS   Adj. R-squared:                  0.540
Method:                 Least Squares   F-statistic:                     2769.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:07   Log-Likelihood:                -7288.7
No. Observations:               18860   AIC:                         1.460e+04
Df Residuals:                   18851   BIC:                         1.467e+04
Df Model:                           8
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const              8.9118      0.072    123.365      0.000       8.770       9.053
bedrooms          -0.0352      0.004     -8.949      0.000      -0.043      -0.027
grade_8            0.1906      0.007     28.140      0.000       0.177       0.204
grade_9            0.4085      0.010     40.902      0.000       0.389       0.428
grade_10           0.6195      0.014     43.809      0.000       0.592       0.647
grade_11           0.8209      0.022     37.402      0.000       0.778       0.864
grade_12           1.1251      0.044     25.804      0.000       1.040       1.211
grade_13           1.3703      0.113     12.076      0.000       1.148       1.593
sqft_living_log    0.5429      0.011     50.204      0.000       0.522       0.564
==============================================================================
Omnibus:                       67.498   Durbin-Watson:                   1.985
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               63.710
Skew:                           0.115   Prob(JB):                     1.46e-14
Kurtosis:                       2.833   Cond. No.                         367.
==============================================================================

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

In [71]:
```
import pandas as pd
import statsmodels.api as sm


# Define predictor (X) and target variable (y)
X = df[['bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5', 'bedrooms_6', 'bedrooms_7', 'sqft_living_log']]
y = df['price_log']
print(X.dtypes)
print(y.dtypes)

 #Add constant term for intercept
X = sm.add_constant(X)

 #Fit OLS regression model
model = sm.OLS(y, X).fit()

#Print regression results
print(model.summary())
```

```
bedrooms_2          uint8
bedrooms_3          uint8
bedrooms_4          uint8
bedrooms_5          uint8
bedrooms_6          uint8
bedrooms_7          uint8
sqft_living_log   float64
dtype: object
float64
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.471
Model:                            OLS   Adj. R-squared:                  0.471
Method:                 Least Squares   F-statistic:                     2395.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:07   Log-Likelihood:                -8615.5
No. Observations:               18860   AIC:                         1.725e+04
Df Residuals:                   18852   BIC:                         1.731e+04
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          6.0300      0.066     91.298      0.000       5.901       6.159
bedrooms_2    -0.0970      0.030     -3.198      0.001      -0.156      -0.038
bedrooms_3    -0.3391      0.030    -11.250      0.000      -0.398      -0.280
bedrooms_4    -0.3935      0.031    -12.657      0.000      -0.454      -0.333
bedrooms_5    -0.3925      0.033    -11.962      0.000      -0.457      -0.328
bedrooms_6    -0.4403      0.041    -10.839      0.000      -0.520      -0.361
bedrooms_7    -0.4470      0.074     -6.064      0.000      -0.592      -0.303
sqft_living_log  0.9730      0.009    110.025      0.000       0.956       0.990
==============================================================================
Omnibus:                       62.904   Durbin-Watson:                   1.987
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               58.400
Skew:                           0.105   Prob(JB):                     2.08e-13
Kurtosis:                       2.825   Cond. No.                         236.
==============================================================================

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

In [72]:
```
import pandas as pd
import statsmodels.api as sm


# Define predictor (X) and target variable (y)
X = df[['bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5', 'bedrooms_6', 'bedrooms_7']]
y = df['price_log']
print(X.dtypes)
print(y.dtypes)

 #Add constant term for intercept
X = sm.add_constant(X)

 #Fit OLS regression model
model = sm.OLS(y, X).fit()

#Print regression results
print(model.summary())
```

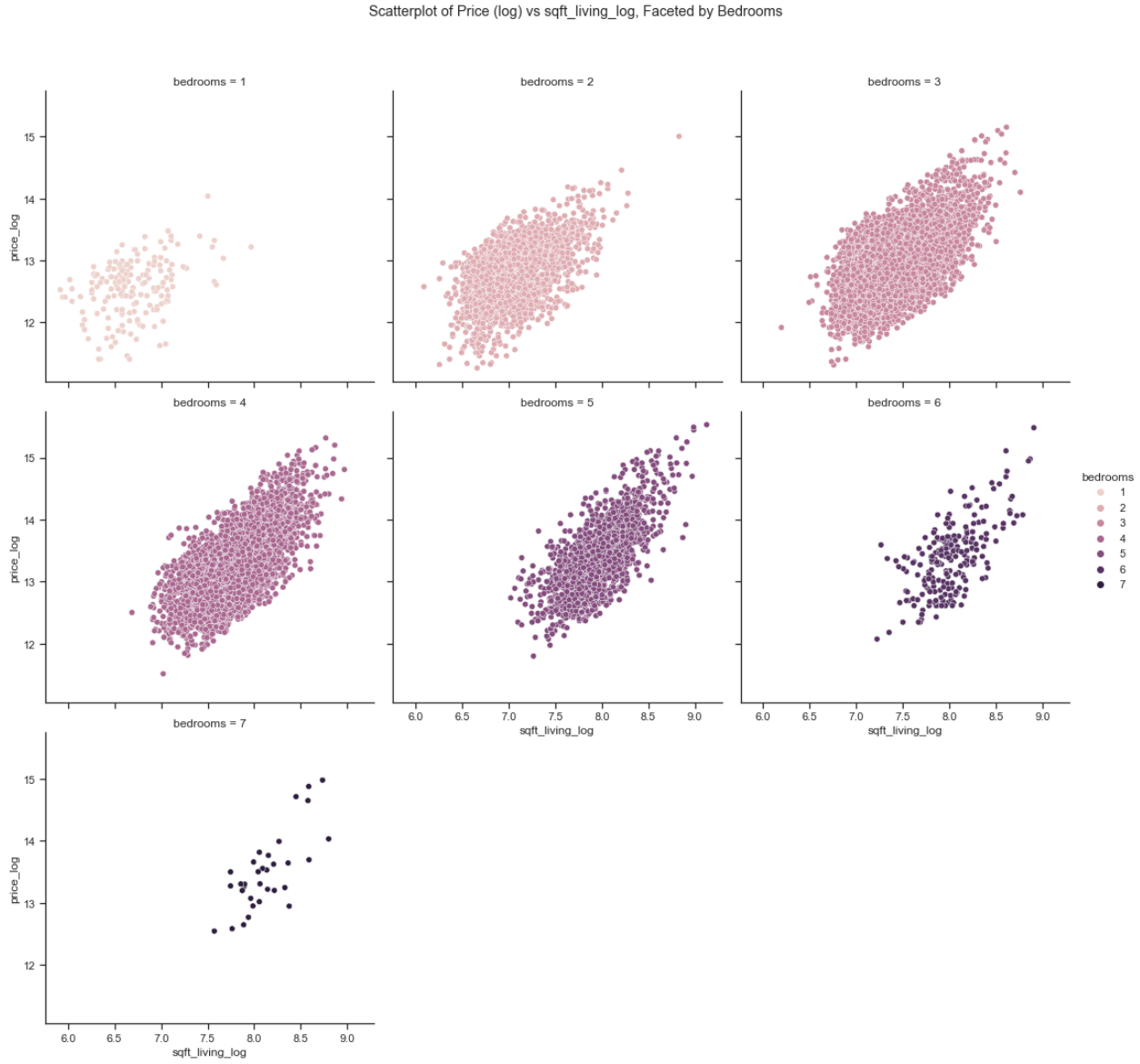
```
Model:                            OLS   Adj. R-squared:                  0.131
Method:                 Least Squares   F-statistic:                     473.3
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:08   Log-Likelihood:                -13293.
No. Observations:               18860   AIC:                         2.660e+04
Df Residuals:                   18853   BIC:                         2.665e+04
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         12.5514      0.037    336.138      0.000      12.478      12.625
bedrooms_2     0.2505      0.039      6.481      0.000       0.175       0.326
bedrooms_3     0.3785      0.038     10.035      0.000       0.305       0.452
bedrooms_4     0.6679      0.038     17.637      0.000       0.594       0.742
bedrooms_5     0.8354      0.040     21.127      0.000       0.758       0.913
bedrooms_6     0.8463      0.050     16.979      0.000       0.749       0.944
bedrooms_7     0.9432      0.093     10.135      0.000       0.761       1.126
==============================================================================
Omnibus:                      486.993   Durbin-Watson:                   1.975
```

In [73]:
```
import pandas as pd
import statsmodels.api as sm


# interaction terms between bedrooms and sqft_living_log for bedrooms_2 through bedrooms_5
for i in range(2, 6):
    df[f'bedrooms_{i}_sqft_interaction'] = df[f'bedrooms_{i}'] * df['sqft_living_log']

# Fit the regression model with the interaction terms
X = df[['sqft_living_log'] + [f'bedrooms_{i}' for i in range(2, 6)] + [f'bedrooms_{i}_sqft_interaction' for i in
X = sm.add_constant(X)  # Add constant for intercept
y = df['price_log']  # Dependent variable

model = sm.OLS(y, X).fit()

# Print the summary of the model
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.480
Model:                            OLS   Adj. R-squared:                  0.480
Method:                 Least Squares   F-statistic:                     1934.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:08   Log-Likelihood:                -8448.7
No. Observations:               18860   AIC:                         1.692e+04
Df Residuals:                   18850   BIC:                         1.700e+04
Df Model:                           9
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                            7.8666      0.189     41.610      0.000       7.496       8.237
sqft_living_log                  0.6930      0.025     27.616      0.000       0.644       0.742
bedrooms_2                      -0.2725      0.251     -1.084      0.278      -0.765       0.220
bedrooms_3                      -1.5362      0.212     -7.241      0.000      -1.952      -1.120
bedrooms_4                      -3.5531      0.227    -15.684      0.000      -3.997      -3.109
bedrooms_5                      -4.6676      0.316    -14.784      0.000      -5.286      -4.049
bedrooms_2_sqft_interaction      0.0447      0.034      1.302      0.193      -0.023       0.112
bedrooms_3_sqft_interaction      0.1940      0.028      6.873      0.000       0.139       0.249
bedrooms_4_sqft_interaction      0.4497      0.030     15.110      0.000       0.391       0.508
bedrooms_5_sqft_interaction      0.5861      0.040     14.491      0.000       0.507       0.665
==============================================================================
Omnibus:                       52.910   Durbin-Watson:                   1.988
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               49.613
Skew:                           0.097   Prob(JB):                     1.69e-11
Kurtosis:                       2.841   Cond. No.                     1.46e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.46e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [ ]:
```

```

In [74]:
```
import seaborn as sns

sns.set(style="ticks")
sns.relplot(x='sqft_living_log', y='price_log', hue='bedrooms', data=df, kind='scatter', col='bedrooms', col_wra
plt.subplots_adjust(top=0.9)
plt.suptitle('Scatterplot of Price (log) vs sqft_living_log, Faceted by Bedrooms')
plt.show()
```



In [ ]:

In [ ]:

In [75]:

```
drooms_5', 'bedrooms_6', 'bedrooms_7', 'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75', 'grade_8','grade_9','grade
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.567
Model:                            OLS   Adj. R-squared:                  0.567
Method:                 Least Squares   F-statistic:                     1235.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:18   Log-Likelihood:                -6715.0
No. Observations:               18860   AIC:                         1.347e+04
Df Residuals:                   18839   BIC:                         1.364e+04
Df Model:                          20
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                        8.6794      0.078    111.375      0.000       8.527       8.832
waterfront_1.0               0.6002      0.029     20.356      0.000       0.542       0.658
floors_2.0                  -0.1021      0.006    -16.493      0.000      -0.114      -0.090
floors_2.5                   0.1194      0.030      3.931      0.000       0.060       0.179
floors_3.0                   0.0624      0.016      3.957      0.000       0.031       0.093
floors_3.5                   0.0309      0.155      0.199      0.842      -0.272       0.334
bedrooms_2                   0.0247      0.028      0.898      0.369      -0.029       0.079
bedrooms_3                  -0.1155      0.028     -4.193      0.000      -0.169      -0.062
bedrooms_4                  -0.1354      0.028     -4.761      0.000      -0.191      -0.080
bedrooms_5                  -0.1279      0.030     -4.252      0.000      -0.187      -0.069
bedrooms_6                  -0.1376      0.037     -3.685      0.000      -0.211      -0.064
bedrooms_7                  -0.1259      0.067     -1.869      0.062      -0.258       0.006
Bathrooms 3.0 - 3.75         0.0900      0.009      9.490      0.000       0.071       0.109
Bathrooms 4.0 - 4.75         0.1547      0.023      6.758      0.000       0.110       0.200
grade_8                      0.2067      0.007     29.943      0.000       0.193       0.220
grade_9                      0.4272      0.010     41.891      0.000       0.407       0.447
grade_10                     0.5986      0.014     41.703      0.000       0.571       0.627
grade_11                     0.7423      0.022     33.290      0.000       0.699       0.786
grade_12                     0.9896      0.043     23.032      0.000       0.905       1.074
grade_13                     1.2706      0.110     11.501      0.000       1.054       1.487
sqft_living_log              0.5740      0.011     52.275      0.000       0.552       0.595
==============================================================================
Omnibus:                       11.655   Durbin-Watson:                   1.979
Prob(Omnibus):                  0.003   Jarque-Bera (JB):               10.906
Skew:                           0.030   Prob(JB):                      0.00428
Kurtosis:                       2.898   Cond. No.                         471.
==============================================================================

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

In [76]:
```
import pandas as pd
import statsmodels.api as sm


# Create interaction terms between bedrooms, grade, and sqft_living_log
for i in range(2, 6):
    for j in range(7, 14):
        df[f'bedrooms_{i}_grade_{j}_sqft_interaction'] = df[f'bedrooms_{i}'] * df[f'grade_{j}'] * df['sqft_livin

# Fit the regression model with the interaction terms
X = df[['sqft_living_log'] + [f'bedrooms_{i}' for i in range(2, 6)] + [f'grade_{j}' for j in range(7, 14)] +
       [f'bedrooms_{i}_grade_{j}_sqft_interaction' for i in range(2, 6) for j in range(7, 14)]]
X = sm.add_constant(X)  # Add constant for intercept
y = df['price_log']  # Dependent variable

model = sm.OLS(y, X).fit()

# Print the summary of the model
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.557
Model:                            OLS   Adj. R-squared:                  0.556
Method:                 Least Squares   F-statistic:                     605.5
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:18   Log-Likelihood:                -6948.4
No. Observations:               18860   AIC:                         1.398e+04
Df Residuals:                   18820   BIC:                         1.429e+04
Df Model:                          39
Covariance Type:            nonrobust
=================================================================================================
                                    coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------------------
const                              9.1903      0.087    105.650      0.000       9.020       9.361
sqft_living_log                    0.4937      0.011     45.054      0.000       0.472       0.515
bedrooms_2                        -0.0572      0.033     -1.749      0.080      -0.121       0.007
bedrooms_3                        -0.2021      0.032     -6.280      0.000      -0.265      -0.139
bedrooms_4                        -0.2143      0.037     -5.723      0.000      -0.288      -0.141
bedrooms_5                        -0.1708      0.061     -2.779      0.005      -0.291      -0.050
grade_7                            0.0653      0.041      1.605      0.109      -0.014       0.145
grade_8                            0.1821      0.045      4.031      0.000       0.094       0.271
grade_9                            0.4590      0.066      6.933      0.000       0.329       0.589
grade_10                           0.7113      0.080      8.915      0.000       0.555       0.868
grade_11                           0.9482      0.109      8.676      0.000       0.734       1.162
grade_12                           1.3218      0.204      6.489      0.000       0.923       1.721
grade_13                           1.4025      0.351      3.991      0.000       0.714       2.091
bedrooms_2_grade_7_sqft_interaction      0.0237      0.006      3.823      0.000       0.012       0.036
bedrooms_2_grade_8_sqft_interaction      0.0211      0.007      3.092      0.002       0.008       0.034
bedrooms_2_grade_9_sqft_interaction      0.0102      0.011      0.962      0.336      -0.011       0.031
bedrooms_2_grade_10_sqft_interaction     0.0004      0.015      0.028      0.978      -0.029       0.030
bedrooms_2_grade_11_sqft_interaction     0.0405      0.032      1.247      0.212      -0.023       0.104
bedrooms_2_grade_12_sqft_interaction    -0.0064      0.049     -0.129      0.897      -0.103       0.090
bedrooms_2_grade_13_sqft_interaction -1.215e-15   4.53e-16     -2.685      0.007    -2.1e-15   -3.28e-16
bedrooms_3_grade_7_sqft_interaction      0.0140      0.006      2.391      0.017       0.003       0.025
bedrooms_3_grade_8_sqft_interaction      0.0255      0.006      4.071      0.000       0.013       0.038
bedrooms_3_grade_9_sqft_interaction      0.0156      0.009      1.785      0.074      -0.002       0.033
bedrooms_3_grade_10_sqft_interaction     0.0095      0.010      0.917      0.359      -0.011       0.030
bedrooms_3_grade_11_sqft_interaction     0.0166      0.015      1.119      0.263      -0.013       0.046
bedrooms_3_grade_12_sqft_interaction     0.0006      0.029      0.021      0.984      -0.056       0.057
bedrooms_3_grade_13_sqft_interaction     0.0589      0.058      1.023      0.307      -0.054       0.172
bedrooms_4_grade_7_sqft_interaction      0.0161      0.006      2.559      0.011       0.004       0.028
bedrooms_4_grade_8_sqft_interaction      0.0209      0.007      3.172      0.002       0.008       0.034
bedrooms_4_grade_9_sqft_interaction      0.0151      0.009      1.711      0.087      -0.002       0.032
bedrooms_4_grade_10_sqft_interaction     0.0098      0.010      0.955      0.340      -0.010       0.030
bedrooms_4_grade_11_sqft_interaction     0.0054      0.014      0.398      0.691      -0.021       0.032
bedrooms_4_grade_12_sqft_interaction    -0.0081      0.025     -0.325      0.745      -0.057       0.041
bedrooms_4_grade_13_sqft_interaction    -0.0023      0.047     -0.049      0.961      -0.095       0.091
bedrooms_5_grade_7_sqft_interaction      0.0089      0.009      0.987      0.324      -0.009       0.027
bedrooms_5_grade_8_sqft_interaction      0.0201      0.009      2.203      0.028       0.002       0.038
bedrooms_5_grade_9_sqft_interaction      0.0224      0.011      2.072      0.038       0.001       0.044
bedrooms_5_grade_10_sqft_interaction     0.0198      0.012      1.645      0.100      -0.004       0.043
bedrooms_5_grade_11_sqft_interaction     0.0040      0.015      0.264      0.792      -0.026       0.034
bedrooms_5_grade_12_sqft_interaction     0.0121      0.026      0.464      0.643      -0.039       0.063
bedrooms_5_grade_13_sqft_interaction     0.0190      0.043      0.437      0.662      -0.066       0.104
==============================================================================
Omnibus:                       51.020   Durbin-Watson:                   1.976
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               51.045
Skew:                           0.121   Prob(JB):                     8.24e-12
Kurtosis:                       2.922   Cond. No.                     1.01e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.19e-26. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [77]:
```
port pandas as pd
port statsmodels.api as sm

= df[[ 'waterfront_1.0', 'floors_2.0', 'floors_2.5', 'floors_3.0', 'bedrooms_2',
      'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75', 'grade_8','grade_9','grade_10', 'grade_11', 'grade_12', 'gr
= df['price_log']

intercept
= sm.add_constant(X)

Fit OLS regression model
del = sm.OLS(y, X).fit()

int(model.summary())
```

```
floors_3.0                0.0829      0.010      4.005      0.000       0.032       0.094
bedrooms_2                0.1366      0.009     15.968      0.000       0.120       0.153
Bathrooms 3.0 - 3.75      0.0907      0.009      9.710      0.000       0.072       0.109
Bathrooms 4.0 - 4.75      0.1575      0.022      7.035      0.000       0.114       0.201
grade_8                   0.2090      0.007     30.390      0.000       0.195       0.222
grade_9                   0.4315      0.010     42.649      0.000       0.412       0.451
grade_10                  0.6063      0.014     42.709      0.000       0.578       0.634
grade_11                  0.7515      0.022     33.979      0.000       0.708       0.795
grade_12                  1.0022      0.043     23.389      0.000       0.918       1.086
grade_13                  1.2910      0.110     11.688      0.000       1.075       1.508
sqft_living_log           0.5515      0.010     57.235      0.000       0.533       0.570
==============================================================================
Omnibus:                       12.878   Durbin-Watson:                   1.980
Prob(Omnibus):                  0.002   Jarque-Bera (JB):               12.034
Skew:                           0.033   Prob(JB):                      0.00244
Kurtosis:                       2.895   Cond. No.                        337.
==============================================================================

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

In [78]:
```
import pandas as pd
import statsmodels.api as sm

X = df[[ 'waterfront_1.0', 'floors_2.0', 'floors_2.5', 'floors_3.0', 'bedrooms_2', 'bedrooms_3', 'bedrooms_4', '
        'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75', 'grade_8','grade_9','grade_10', 'grade_11', 'grade_12',
y = df['price_log']

# intercept
X = sm.add_constant(X)

# Fit OLS regression model
model = sm.OLS(y, X).fit()

print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.567
Model:                            OLS   Adj. R-squared:                  0.567
Method:                 Least Squares   F-statistic:                     1452.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:19   Log-Likelihood:                -6722.0
No. Observations:               18860   AIC:                         1.348e+04
Df Residuals:                   18842   BIC:                         1.362e+04
Df Model:                          17
Covariance Type:            nonrobust
===========================================================================================
                              coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------------
const                       8.6934      0.078    111.658      0.000       8.541       8.846
waterfront_1.0              0.6042      0.029     20.502      0.000       0.546       0.662
floors_2.0                 -0.1017      0.006    -16.418      0.000      -0.114      -0.090
floors_2.5                  0.1169      0.030      3.848      0.000       0.057       0.176
floors_3.0                  0.0614      0.016      3.894      0.000       0.030       0.092
bedrooms_2                  0.1001      0.019      5.352      0.000       0.063       0.137
bedrooms_3                 -0.0359      0.017     -2.057      0.040      -0.070      -0.002
bedrooms_4                 -0.0525      0.018     -2.954      0.003      -0.087      -0.018
bedrooms_5                 -0.0426      0.020     -2.178      0.029      -0.081      -0.004
Bathrooms 3.0 - 3.75        0.0888      0.009      9.365      0.000       0.070       0.107
Bathrooms 4.0 - 4.75        0.1476      0.023      6.486      0.000       0.103       0.192
grade_8                     0.2086      0.007     30.295      0.000       0.195       0.222
grade_9                     0.4317      0.010     42.605      0.000       0.412       0.452
grade_10                    0.6056      0.014     42.542      0.000       0.578       0.634
grade_11                    0.7518      0.022     33.932      0.000       0.708       0.795
grade_12                    1.0007      0.043     23.346      0.000       0.917       1.085
grade_13                    1.2819      0.110     11.605      0.000       1.065       1.498
sqft_living_log             0.5612      0.010     53.769      0.000       0.541       0.582
==============================================================================
Omnibus:                       12.515   Durbin-Watson:                   1.980
Prob(Omnibus):                  0.002   Jarque-Bera (JB):               11.683
Skew:                           0.031   Prob(JB):                      0.00290
Kurtosis:                       2.895   Cond. No.                        338.
==============================================================================

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

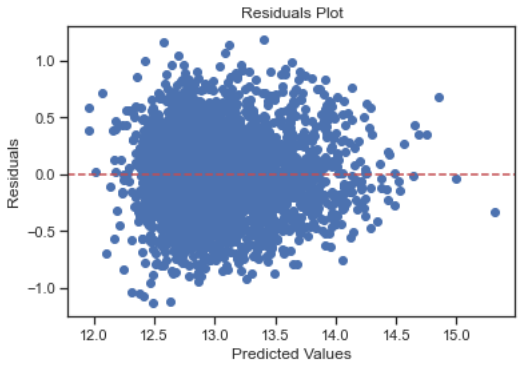
In [79]:
```
import pandas as pd
import statsmodels.api as sm

X = df[[ 'waterfront_1.0', 'floors_2.0', 'floors_2.5', 'floors_3.0', 'bedrooms_2', 'bedrooms_3', 'bedrooms_4', '
          'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75',  'grade_6', 'grade_7', 'grade_8','grade_9','grade_10',
y = df['price_log']

# intercept
X = sm.add_constant(X)

# Fit OLS regression model
model = sm.OLS(y, X).fit()

print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.578
Model:                            OLS   Adj. R-squared:                  0.577
Method:                 Least Squares   F-statistic:                     1357.
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:19   Log-Likelihood:                -6486.5
No. Observations:               18860   AIC:                         1.301e+04
Df Residuals:                   18840   BIC:                         1.317e+04
Df Model:                          19
Covariance Type:            nonrobust
===========================================================================================
                              coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------------
const                       9.0170      0.080    112.799      0.000       8.860       9.174
waterfront_1.0              0.6206      0.029     21.314      0.000       0.564       0.678
floors_2.0                 -0.1121      0.006    -18.265      0.000      -0.124      -0.100
floors_2.5                  0.1077      0.030      3.589      0.000       0.049       0.167
floors_3.0                  0.0293      0.016      1.874      0.061      -0.001       0.060
bedrooms_2                  0.0695      0.019      3.731      0.000       0.033       0.106
bedrooms_3                 -0.0879      0.017     -5.035      0.000      -0.122      -0.054
bedrooms_4                 -0.0913      0.018     -5.156      0.000      -0.126      -0.057
bedrooms_5                 -0.0739      0.019     -3.804      0.000      -0.112      -0.036
Bathrooms 3.0 - 3.75        0.0968      0.009     10.337      0.000       0.078       0.115
Bathrooms 4.0 - 4.75        0.1616      0.022      7.184      0.000       0.117       0.206
grade_6                     0.1491      0.024      6.182      0.000       0.102       0.196
grade_7                     0.3313      0.024     13.950      0.000       0.285       0.378
grade_8                     0.5304      0.025     21.448      0.000       0.482       0.579
grade_9                     0.7740      0.026     29.402      0.000       0.722       0.826
grade_10                    0.9611      0.028     33.757      0.000       0.905       1.017
grade_11                    1.1187      0.033     33.579      0.000       1.053       1.184
grade_12                    1.3796      0.049     27.918      0.000       1.283       1.476
grade_13                    1.6777      0.112     14.959      0.000       1.458       1.898
sqft_living_log             0.4833      0.011     44.294      0.000       0.462       0.505
==============================================================================
Omnibus:                       17.330   Durbin-Watson:                   1.976
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               16.755
Skew:                           0.056   Prob(JB):                     0.000230
Kurtosis:                       2.905   Cond. No.                         354.
==============================================================================

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

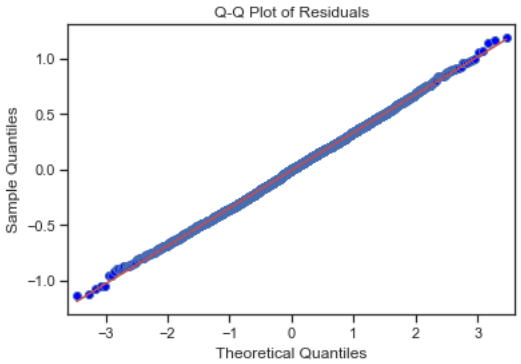
In [80]:
```
klearn.model_selection import train_test_split
klearn.linear_model import LinearRegression
klearn.metrics import mean_squared_error, mean_absolute_error
 pandas as pd
 statsmodels.api as sm

[['waterfront_1.0', 'floors_2.0', 'floors_2.5', 'floors_3.0', 'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms
   'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75', 'grade_6', 'grade_7', 'grade_8','grade_9','grade_10', 'grade_1
['price_log']
t the data into training and testing sets
n, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

n the linear regression model
= LinearRegression()
fit(X_train, y_train)

 predictions
 = model.predict(X_test)

uate the model
mean_squared_error(y_test, y_pred)
mean_absolute_error(y_test, y_pred)
'Mean Squared Error:', mse)
'Mean Absolute Error:', mae)
```

```
Mean Squared Error: 0.11620370039511732
Mean Absolute Error: 0.27426063808399004
```

In [81]:
```
import matplotlib.pyplot as plt

# Calculate residuals
residuals = y_test - y_pred

# Plot residuals
plt.scatter(y_pred, residuals)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residuals Plot')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



In [82]:
```
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Calculate residuals
residuals = y_test - y_pred

# Generate Q-Q plot
fig = sm.qqplot(residuals, line='s')

# Set plot labels and title
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')
plt.title('Q-Q Plot of Residuals')

# Display the plot
plt.show()
```



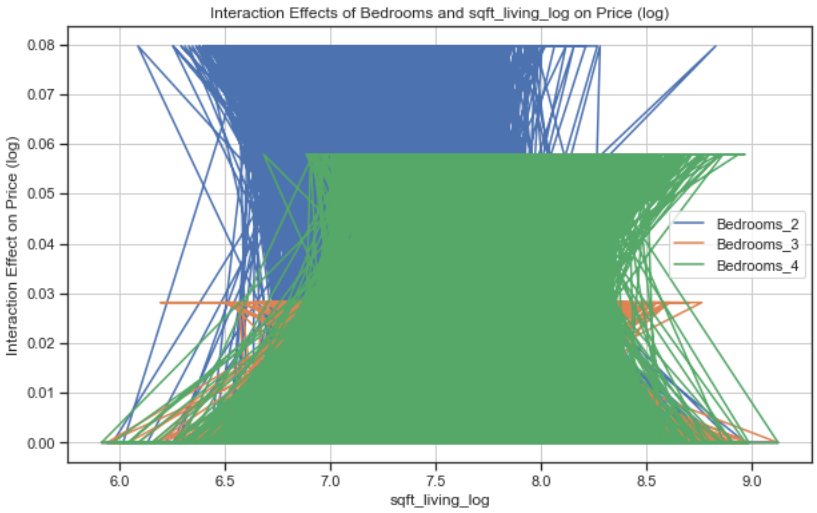
In [83]:
```
import statsmodels.stats.api as sms

# Calculate residuals
residuals = y_test - y_pred

# Perform Jarque-Bera test on residuals
name = ['Jarque-Bera', 'Prob', 'Skew', 'Kurtosis']
jb_test = sms.jarque_bera(residuals)
result = dict(zip(name, jb_test))

# Print the test results
for key, value in result.items():
    print(key + ':', value)
```

```
Jarque-Bera: 3.459706909741273
Prob: 0.1773103920403974
Skew: 0.055548655986612114
Kurtosis: 2.9016619673246113
```

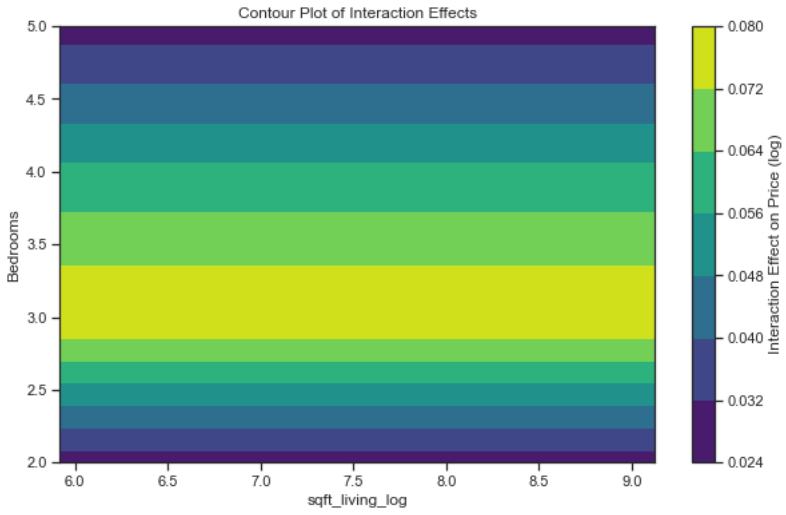
In [84]:
```
from statsmodels.stats.diagnostic import het_goldfeldquandt

# Calculate residuals
residuals = y_test - y_pred

# Perform Goldfeld-Quandt test for heteroscedasticity
gq_test_statistic, gq_p_value, gq_f_value = het_goldfeldquandt(residuals, X_test)

# Print the results
print("Goldfeld-Quandt Test Statistic:", gq_test_statistic)
print("Goldfeld-Quandt p-value:", gq_p_value)
print("Goldfeld-Quandt F-statistic:", gq_f_value)
```

```
Goldfeld-Quandt Test Statistic: 1.0020373431880651
Goldfeld-Quandt p-value: 0.48246549116035825
Goldfeld-Quandt F-statistic: increasing
```

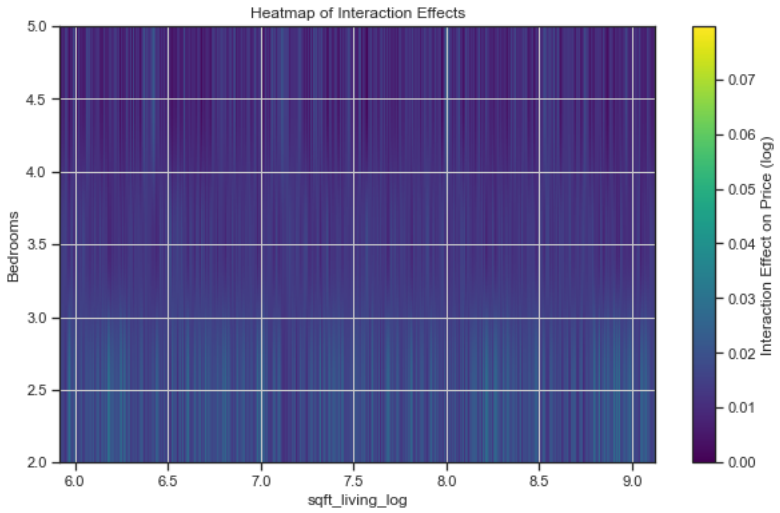
In [85]:
```
import matplotlib.pyplot as plt

# Extract the coefficients from the model
interaction_coeffs = model.coef_[2:5]
# Plot the interaction effects
plt.figure(figsize=(10, 6))

for i, coeff in enumerate(interaction_coeffs):
    plt.plot(df['sqft_living_log'], df[f'bedrooms_{i+2}'] * coeff, label=f'Bedrooms_{i+2}')

plt.xlabel('sqft_living_log')
plt.ylabel('Interaction Effect on Price (log)')
plt.title('Interaction Effects of Bedrooms and sqft_living_log on Price (log)')
plt.legend()
plt.grid(True)
plt.show()
```



In [86]:
```
import numpy as np

# Create a grid of values for sqft_living_log and bedrooms
sqft_living_log_vals = np.linspace(df['sqft_living_log'].min(), df['sqft_living_log'].max(), 100)
bedrooms_vals = np.array([2, 3, 4, 5])

# Create a meshgrid from the values
sqft_living_log_grid, bedrooms_grid = np.meshgrid(sqft_living_log_vals, bedrooms_vals)

# Initialize a grid for interaction effects
interaction_effects_grid = np.zeros_like(sqft_living_log_grid)

# Calculate interaction effects for each combination of sqft_living_log and bedrooms
for i in range(sqft_living_log_grid.shape[1]):
    for j in range(bedrooms_grid.shape[0]):
        for k, coeff in enumerate(interaction_coeffs):
            interaction_effects_grid[j, i] += df[f'bedrooms_{k+2}'].iloc[j] * coeff

# Plot the contour plot
plt.figure(figsize=(10, 6))
plt.contourf(sqft_living_log_grid, bedrooms_grid, interaction_effects_grid, cmap='viridis')
plt.xlabel('sqft_living_log')
plt.ylabel('Bedrooms')
plt.title('Contour Plot of Interaction Effects')
plt.colorbar(label='Interaction Effect on Price (log)')
plt.grid(True)
plt.show()
```



In [87]:
```
import numpy as np

interaction_effects = np.zeros((len(interaction_coeffs), len(df['sqft_living_log'])))

for i, coeff in enumerate(interaction_coeffs):
    interaction_effects[i, :] = df[f'bedrooms_{i+2}'] * coeff

plt.figure(figsize=(10, 6))
plt.imshow(interaction_effects, extent=[min(df['sqft_living_log']), max(df['sqft_living_log']), 2, 5], aspect='a
plt.colorbar(label='Interaction Effect on Price (log)')
plt.xlabel('sqft_living_log')
plt.ylabel('Bedrooms')
plt.title('Heatmap of Interaction Effects')
plt.grid(True)
plt.show()
```



In [88]:
```
import numpy as np

# Calculate descriptive statistics for sqft_living_log
min_sqft_living_log = np.min(df['sqft_living_log'])
max_sqft_living_log = np.max(df['sqft_living_log'])
mean_sqft_living_log = np.mean(df['sqft_living_log'])
std_dev_sqft_living_log = np.std(df['sqft_living_log'])

print("Minimum sqft_living_log:", min_sqft_living_log)
print("Maximum sqft_living_log:", max_sqft_living_log)
print("Mean sqft_living_log:", mean_sqft_living_log)
print("Standard deviation of sqft_living_log:", std_dev_sqft_living_log)
```

```
Minimum sqft_living_log: 5.916202062607435
Maximum sqft_living_log: 9.127067452782361
Mean sqft_living_log: 7.545039279902697
Standard deviation of sqft_living_log: 0.4206099603700284
```

This is the end of the final version of code used in this project. The code below is further investigative code to experiment with different approaches - which did not prove any more useful.

In [89]:
```
# Calculate bin edges based on standard deviations
num_std_devs = 3  # Number of standard deviations

# Calculate bin edges for the initial bins
initial_bin_edges = [mean_sqft_living_log - i * std_dev_sqft_living_log for i in range(num_std_devs, -1, -1)] +
                    [mean_sqft_living_log + i * std_dev_sqft_living_log for i in range(1, num_std_devs + 1)]

# Create additional bins for outliers
additional_bins = 2  # Number of additional bins

# Calculate the range covered by the initial bins
initial_bin_range = initial_bin_edges[-1] - initial_bin_edges[0]

# Calculate the size of each additional bin
additional_bin_size = initial_bin_range / (additional_bins + 1)

# Calculate the bin edges for the additional bins
additional_bin_edges_neg = [initial_bin_edges[0] - (i + 1) * additional_bin_size for i in range(additional_bins)
additional_bin_edges_pos = [initial_bin_edges[-1] + (i + 1) * additional_bin_size for i in range(additional_bins

# Combine initial and additional bin edges
bin_edges = additional_bin_edges_neg + initial_bin_edges + additional_bin_edges_pos

# Ensure all bin edges are in increasing order
bin_edges.sort()

# Assign each data point to a bin
df['sqft_living_log_bin'] = pd.cut(df['sqft_living_log'], bins=bin_edges, labels=False)

# Display the bins
print("Bin Edges:", bin_edges)
```

```
Bin Edges: [4.600769557312498, 5.441989478052554, 6.283209398792612, 6.70381935916264, 7.124429319532668, 7.545
039279902697, 7.965649240272725, 8.386259200642753, 8.806869161012782, 9.64808908175284, 10.489309002492897]
```

In [90]:
```
# Group data by bins
grouped_data = df.groupby('sqft_living_log_bin')
```

In [91]:
```
# Calculate mean price_log for each bin
mean_price_log_by_bin = grouped_data['price_log'].mean()
print(mean_price_log_by_bin)
```

```
sqft_living_log_bin
1    12.334374
2    12.460158
3    12.636771
4    12.845149
5    13.138262
6    13.608943
7    14.214373
8    14.911348
Name: price_log, dtype: float64
```

In [92]:
```
das as pd
tsmodels.api as sm

aterfront_1.0', 'floors_2.0', 'floors_2.5', 'floors_3.0', 'bedrooms_2',
athrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75', 'grade_8', 'grade_9',
rade_10', 'grade_11', 'grade_12', 'grade_13', 'sqft_living_log', 'sqft_living_log_bin']]  # Include sqft_living_l
ice_log']

tant term for intercept
_constant(X)

regression model
_bins = sm.OLS(y, X).fit()

gression results
l_with_bins.summary())
```

OLS Regression Results

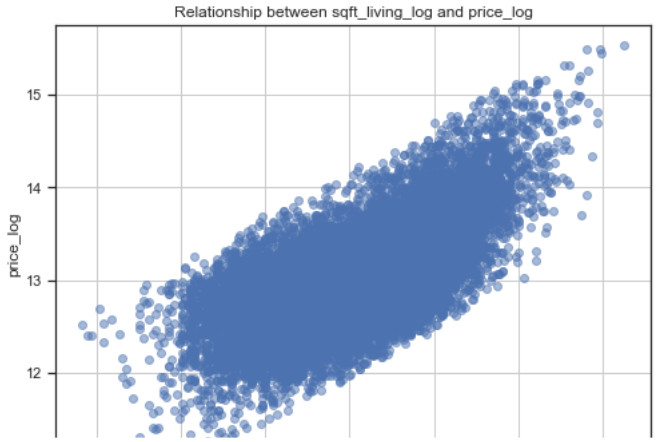
Dep. Variable:	price_log	R-squared:	0.567
Model:	OLS	Adj. R-squared:	0.566
Method:	Least Squares	F-statistic:	1643.
Date:	Mon, 26 Feb 2024	Prob (F-statistic):	0.00
Time:	03:49:32	Log-Likelihood:	-6728.3
No. Observations:	18860	AIC:	1.349e+04
Df Residuals:	18844	BIC:	1.361e+04
Df Model:	15		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	8.7183	0.138	63.242	0.000	8.448	8.988
waterfront_1.0	0.6087	0.029	20.672	0.000	0.551	0.666
floors_2.0	-0.1024	0.006	-16.585	0.000	-0.115	-0.090
floors_2.5	0.1199	0.030	3.948	0.000	0.060	0.179
floors_3.0	0.0630	0.016	4.003	0.000	0.032	0.094
bedrooms_2	0.1366	0.009	15.967	0.000	0.120	0.153
Bathrooms 3.0 - 3.75	0.0906	0.009	9.708	0.000	0.072	0.109
Bathrooms 4.0 - 4.75	0.1575	0.022	7.035	0.000	0.114	0.201
grade_8	0.2090	0.007	30.389	0.000	0.195	0.222
grade_9	0.4315	0.010	42.647	0.000	0.412	0.451
grade_10	0.6062	0.014	42.708	0.000	0.578	0.634
grade_11	0.7515	0.022	33.977	0.000	0.708	0.795
grade_12	1.0022	0.043	23.388	0.000	0.918	1.086
grade_13	1.2911	0.110	11.688	0.000	1.075	1.508
sqft_living_log	0.5528	0.023	24.057	0.000	0.508	0.598
sqft_living_log_bin	-0.0005	0.009	-0.060	0.952	-0.018	0.017

Omnibus:	12.874	Durbin-Watson:	1.980
Prob(Omnibus):	0.002	Jarque-Bera (JB):	12.030
Skew:	0.033	Prob(JB):	0.00244
Kurtosis:	2.895	Cond. No.	496.

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

In [93]:
```
import matplotlib.pyplot as plt

# Scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(df['sqft_living_log'], df['price_log'], alpha=0.5)
plt.title('Relationship between sqft_living_log and price_log')
plt.xlabel('sqft_living_log')
plt.ylabel('price_log')
plt.grid(True)
plt.show()
```



In [94]:
```
variable_titles = df.columns.tolist()

# Print all variable titles
print(variable_titles)
```

```
ms_1.25', 'bathrooms_1.5', 'bathrooms_1.75', 'bathrooms_2.0', 'bathrooms_2.25', 'bathrooms_2.5', 'bathrooms
_2.75', 'bathrooms_3.0', 'bathrooms_3.25', 'bathrooms_3.5', 'bathrooms_3.75', 'bathrooms_4.0', 'bathrooms_
4.25', 'bathrooms_4.5', 'bathrooms_4.75', 'bathrooms_5.0', 'bathrooms_5.25', 'bathrooms_5.5', 'bathrooms_5.
75', 'bathrooms_6.0', 'floors_1.5', 'floors_2.0', 'floors_2.5', 'floors_3.0', 'floors_3.5', 'waterfront_1.
0', 'condition_2', 'condition_3', 'condition_4', 'condition_5', 'grade_4', 'grade_5', 'grade_6', 'grade_7',
'grade_8', 'grade_9', 'grade_10', 'grade_11', 'grade_12', 'grade_13', 'Bathrooms 0.75 - 1.75', 'Bathrooms
2.0 - 2.75', 'Bathrooms 3.0 - 3.75', 'Bathrooms 4.0 - 4.75', 'Bathrooms 5.0 - 6.0', 'price_log', 'sqft_livi
ng_log', 'bedrooms_less_than_8_true', 'bedrooms_less_than_8_false', 'grade_less_than_8_true', 'grade_less_t
han_8_false', 'bedrooms_2_sqft_interaction', 'bedrooms_3_sqft_interaction', 'bedrooms_4_sqft_interaction',
'bedrooms_5_sqft_interaction', 'bedrooms_2_grade_7_sqft_interaction', 'bedrooms_2_grade_8_sqft_interactio
n', 'bedrooms_2_grade_9_sqft_interaction', 'bedrooms_2_grade_10_sqft_interaction', 'bedrooms_2_grade_11_sqf
t_interaction', 'bedrooms_2_grade_12_sqft_interaction', 'bedrooms_2_grade_13_sqft_interaction', 'bedrooms_3
_grade_7_sqft_interaction', 'bedrooms_3_grade_8_sqft_interaction', 'bedrooms_3_grade_9_sqft_interaction',
'bedrooms_3_grade_10_sqft_interaction', 'bedrooms_3_grade_11_sqft_interaction', 'bedrooms_3_grade_12_sqft_i
nteraction', 'bedrooms_3_grade_13_sqft_interaction', 'bedrooms_4_grade_7_sqft_interaction', 'bedrooms_4_gra
de_8_sqft_interaction', 'bedrooms_4_grade_9_sqft_interaction', 'bedrooms_4_grade_10_sqft_interaction', 'bed
rooms_4_grade_11_sqft_interaction', 'bedrooms_4_grade_12_sqft_interaction', 'bedrooms_4_grade_13_sqft_inter
action', 'bedrooms_5_grade_7_sqft_interaction', 'bedrooms_5_grade_8_sqft_interaction', 'bedrooms_5_grade_9_
sqft_interaction', 'bedrooms_5_grade_10_sqft_interaction', 'bedrooms_5_grade_11_sqft_interaction', 'bedroom
```

In [95]:
```
import numpy as np


# Retrieve the coefficient for sqft_living_log
coef_sqft_living_log = 0.8354

# Calculate the mean of sqft_living_log
mean_sqft_living_log = df['sqft_living_log'].mean()

# Exponentiate the coefficient to obtain the factor by which the price changes
factor_price_change = np.exp(coef_sqft_living_log)

# Calculate the mean price of sqft_living_log
mean_price_sqft_living_log = factor_price_change * mean_sqft_living_log

# Calculate the mean of price_log
mean_price_log = df['price_log'].mean()

# Exponentiate the mean of price_log to obtain the mean price
mean_price = np.exp(mean_price_log)

print("Mean of sqft_living_log:", mean_sqft_living_log)
print("Corresponding Mean Price of sqft_living_log:", mean_price_sqft_living_log)
print("Mean Price:", mean_price)
```

```
Mean of sqft_living_log: 7.545039279902697
Corresponding Mean Price of sqft_living_log: 17.396869883586454
Mean Price: 462026.55338403746
```

In [96]:
```
import numpy as np



# Retrieve the coefficient for sqft_living_log
coef_sqft_living_log = 0.8354

# Calculate the mean of sqft_living_log
mean_sqft_living_log = df['sqft_living_log'].mean()

# Exponentiate the coefficient to obtain the factor by which the price changes
factor_price_change = np.exp(coef_sqft_living_log)

# Calculate the increase in sqft from the mean value for every 100 sqft
increase_in_sqft = 100 - mean_sqft_living_log

# Calculate the expected increase in price for every 100 sqft from the mean
price_increase_factor = factor_price_change ** increase_in_sqft

# Assuming mean_price is the mean price of the dataset
mean_price = np.exp(df['price_log'].mean())

# Calculate the expected increase in price in dollars for every 100 sqft from the mean
expected_increase_dollars = mean_price * (price_increase_factor - 1)

print("Expected Increase in Price in dollars for every 100 sqft from the mean:", expected_increase_dollars)
```

```
Expected Increase in Price in dollars for every 100 sqft from the mean: 1.6151586741095248e+39
```

In [97]:
```
import numpy as np


# Retrieve the coefficient for sqft_living_log
coef_sqft_living_log = 0.8354

# Calculate the mean of sqft_living_log
mean_sqft_living_log = df['sqft_living_log'].mean()

# Exponentiate the coefficient to obtain the factor by which the price changes
factor_price_change_per_unit = np.exp(coef_sqft_living_log)

# Calculate the increase in price for one unit increase in sqft_living_log
increase_price_per_unit = mean_price * (factor_price_change_per_unit - 1)

# Calculate the increase in price for every 100 sqft from the mean
increase_price_100_sqft = increase_price_per_unit * 100

print("Expected Increase in Price in dollars for every 100 sqft from the mean:", increase_price_100_sqft)
```

```
Expected Increase in Price in dollars for every 100 sqft from the mean: 60328477.68556732
```

In [98]:
```
import numpy as np

# Coefficient for sqft_living_log
coef_sqft = 0.5515

# Change in square footage (e.g., +100 sq ft)
change_in_sqft = 100

# Baseline price
baseline_price = 6189.01  # Use the baseline price calculated earlier

# Calculate the expected increase in price
price_increase = (coef_sqft * change_in_sqft) * baseline_price

print("Expected Increase in Price:", price_increase)
```

```
Expected Increase in Price: 341323.9015
```

In [99]:
```
import pandas as pd

median_sqft_living = df['sqft_living'].median()

# Filter to get rows with median sqft_living
median_sqft_living_row = df[df['sqft_living'] == median_sqft_living]

# Calculate the median price corresponding to the median sqft_living
median_price_corresponding = median_sqft_living_row['price'].median()

print("Median Price Corresponding to the Median sqft_living:", median_price_corresponding)
```

```
Median Price Corresponding to the Median sqft_living: 433250.0
```

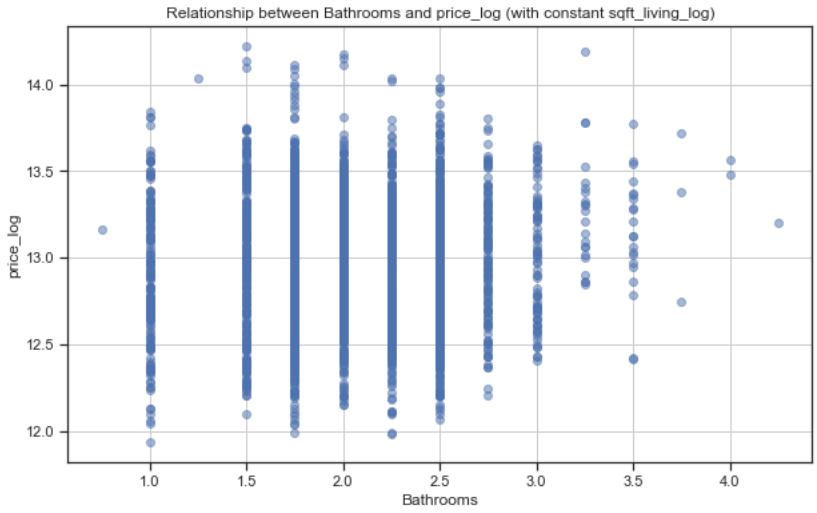
In [100]:
```
import pandas as pd

median_sqft_living = df['sqft_living'].median()
median_price = df['price'].median()

print("Median of sqft_living:", median_sqft_living)
print("Median Price:", median_price)
```

```
Median of sqft_living: 1900.0
Median Price: 450000.0
```

In [101]:
```
# Coefficients for each grade level
coefficients = {
    'grade_8': 0.2090,
    'grade_9': 0.4315,
    'grade_10': 0.6063,
    'grade_11': 0.7515,
    'grade_12': 1.0022,
    'grade_13': 1.2910
}

# Baseline price
baseline_price = 6189.01  # Use the baseline price calculated earlier

# Calculate the expected increase in price for each grade level
for grade, coef in coefficients.items():
    price_increase = baseline_price * coef
    print(f"Expected Increase in Price for {grade}: {price_increase}")
```

```
Expected Increase in Price for grade_8: 1293.50309
Expected Increase in Price for grade_9: 2670.557815
Expected Increase in Price for grade_10: 3752.3967629999997
Expected Increase in Price for grade_11: 4651.041015
Expected Increase in Price for grade_12: 6202.625822
Expected Increase in Price for grade_13: 7990.01191
```

In [102]:
```
import matplotlib.pyplot as plt

# Calculate the mean sqft_living_log
mean_sqft_living_log = df['sqft_living_log'].mean()

# Filter the data to include only rows where sqft_living_log is close to the mean
# Here we use a tolerance value (e.g., 0.1) to define a range around the mean
tolerance = 0.1
filtered_data = df[(df['sqft_living_log'] >= mean_sqft_living_log - tolerance) &
                   (df['sqft_living_log'] <= mean_sqft_living_log + tolerance)]

# Plot the relationship between the number of bathrooms and price_log
plt.figure(figsize=(10, 6))
plt.scatter(filtered_data['bathrooms'], filtered_data['price_log'], alpha=0.5)
plt.title('Relationship between Bathrooms and price_log (with constant sqft_living_log)')
plt.xlabel('Bathrooms')
plt.ylabel('price_log')
plt.grid(True)
plt.show()
```



In [103]:
```
import pandas as pd
import statsmodels.api as sm

X = df[['grade_4', 'grade_5', 'grade_6', 'grade_7', 'grade_8', 'grade_9', 'grade_10', 'grade_11', 'grade_12', 'g
y = df['price_log']

# Add constant term for intercept
X = sm.add_constant(X)

# Fit OLS regression model
model = sm.OLS(y, X).fit()

# Print regression results
print(model.summary())
```

```
grade_4       -0.2990      0.381     -0.784      0.433      -1.046       0.448
grade_5       -0.1517      0.375     -0.405      0.686      -0.886       0.583
grade_6        0.0675      0.374      0.180      0.857      -0.666       0.800
grade_7        0.3579      0.374      0.957      0.339      -0.375       1.091
grade_8        0.6564      0.374      1.756      0.079      -0.076       1.389
grade_9        1.0157      0.374      2.716      0.007       0.283       1.749
grade_10       1.3371      0.374      3.575      0.000       0.604       2.070
grade_11       1.6446      0.374      4.392      0.000       0.911       2.378
grade_12       2.0495      0.376      5.444      0.000       1.312       2.787
grade_13       2.4127      0.392      6.153      0.000       1.644       3.181
==============================================================================
Omnibus:                       94.309   Durbin-Watson:                   1.974
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               95.823
Skew:                           0.169   Prob(JB):                     1.56e-21
Kurtosis:                       3.085   Cond. No.                         519.
==============================================================================

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

In [104]:
```
bedroom_columns = ['bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5', 'bedrooms_6', 'bedrooms_7']

for bedroom_column in bedroom_columns:
    # Filter the dataframe to include only rows where the number of bedrooms corresponds to the current column
    bedroom_df = df[df[bedroom_column] == 1]

    # Print the range of prices for properties with the current number of bedrooms
    print("Price range for properties with", bedroom_column, "bedrooms:")
    print("Minimum price:", bedroom_df['price'].min())
    print('Mean Price:', bedroom_df['price'].mean())
```

```
Price range for properties with bedrooms_2 bedrooms:
Minimum price: 78000.0
Mean Price: 401318.8385761589
Price range for properties with bedrooms_3 bedrooms:
Minimum price: 82000.0
Mean Price: 463354.39038618596
Price range for properties with bedrooms_4 bedrooms:
Minimum price: 100000.0
Mean Price: 633979.8924071618
Price range for properties with bedrooms_5 bedrooms:
Minimum price: 133000.0
Mean Price: 782193.261299435
Price range for properties with bedrooms_6 bedrooms:
Minimum price: 175000.0
Mean Price: 790643.9227272727
Price range for properties with bedrooms_7 bedrooms:
Minimum price: 280000.0
Mean Price: 901182.3333333334
```

In [105]:
```
import pandas as pd
import statsmodels.api as sm

X = df[['bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5', 'bedrooms_6', 'bedrooms_7']]
y = df['price_log']

# Add constant term for intercept
X = sm.add_constant(X)

# Fit OLS regression model
model = sm.OLS(y, X).fit()

# Print regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.131
Model:                            OLS   Adj. R-squared:                  0.131
Method:                 Least Squares   F-statistic:                     473.3
Date:                Mon, 26 Feb 2024   Prob (F-statistic):               0.00
Time:                        03:49:35   Log-Likelihood:                -13293.
No. Observations:               18860   AIC:                         2.660e+04
Df Residuals:                   18853   BIC:                         2.665e+04
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         12.5514      0.037    336.138      0.000      12.478      12.625
bedrooms_2     0.2505      0.039      6.481      0.000       0.175       0.326
bedrooms_3     0.3785      0.038     10.035      0.000       0.305       0.452
bedrooms_4     0.6679      0.038     17.637      0.000       0.594       0.742
bedrooms_5     0.8354      0.040     21.127      0.000       0.758       0.913
bedrooms_6     0.8463      0.050     16.979      0.000       0.749       0.944
bedrooms_7     0.9432      0.093     10.135      0.000       0.761       1.126
==============================================================================
Omnibus:                      486.993   Durbin-Watson:                   1.975
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              566.679
Skew:                           0.354   Prob(JB):                    8.85e-124
Kurtosis:                       3.470   Cond. No.                         35.8
==============================================================================

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

In [106]:
```
import pandas as pd
import numpy as np
from scipy.stats import pointbiserialr, chi2_contingency
from sklearn.preprocessing import LabelEncoder

# Function to calculate Cramer's V
def cramers_v(confusion_matrix):
    chi2 = chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    phi2 = chi2 / n
    r, k = confusion_matrix.shape
    phi2corr = max(0, phi2 - ((k-1)*(r-1))/(n-1))
    rcorr = r - ((r-1)**2)/(n-1)
    kcorr = k - ((k-1)**2)/(n-1)
    return np.sqrt(phi2corr / min((kcorr-1), (rcorr-1)))

# Function to calculate correlation coefficients
def calculate_correlation(var1, var2):
    if var1 in continuous_vars and var2 in continuous_vars:
        # For continuous-continuous variables, use Pearson correlation coefficient
        corr_coef = df[var1].corr(df[var2])
    elif var1 in continuous_vars and var2 in ordinal_categorical_vars:
        # For continuous-ordinal categorical variables, use point-biserial correlation coefficient
        corr_coef, _ = pointbiserialr(df[var2], df[var1])
    elif var1 in ordinal_categorical_vars and var2 in continuous_vars:
        # For ordinal categorical-continuous variables, use point-biserial correlation coefficient
        corr_coef, _ = pointbiserialr(df[var1], df[var2])
    elif var1 in binary_categorical_vars and var2 in continuous_vars:
        # For binary categorical-continuous variables, use point-biserial correlation coefficient
        corr_coef, _ = pointbiserialr(df[var1], df[var2])
    elif var1 in ordinal_categorical_vars and var2 in ordinal_categorical_vars:
        # For ordinal categorical-ordinal categorical variables, calculate Cramer's V
        confusion_matrix = pd.crosstab(df[var1], df[var2])
        corr_coef = cramers_v(confusion_matrix)
    elif var1 in binary_categorical_vars and var2 in binary_categorical_vars:
        # For binary categorical-binary categorical variables, calculate Cramer's V
        confusion_matrix = pd.crosstab(df[var1], df[var2])
        corr_coef = cramers_v(confusion_matrix)
    else:
        # For all other combinations, return None
        corr_coef = None
    return corr_coef

# List of continuous variables
continuous_vars = ['sqft_lot', 'sqft_living']

# List of ordinal categorical variables
ordinal_categorical_vars = ['yr_built_Group 2',
       'yr_built_Group 3', 'yr_built_Group 4', 'yr_built_Group 5',
       'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5', 'bedrooms_6',
       'bedrooms_7', 'bathrooms_0.75', 'bathrooms_1.0', 'bathrooms_1.25',
       'bathrooms_1.5', 'bathrooms_1.75', 'bathrooms_2.0', 'bathrooms_2.25',
       'bathrooms_2.5', 'bathrooms_2.75', 'bathrooms_3.0', 'bathrooms_3.25',
       'bathrooms_3.5', 'bathrooms_3.75', 'bathrooms_4.0', 'bathrooms_4.25',
       'bathrooms_4.5', 'bathrooms_4.75', 'bathrooms_5.0', 'bathrooms_5.25',
       'bathrooms_5.5', 'bathrooms_5.75', 'bathrooms_6.0', 'floors_1.5',
       'floors_2.0', 'floors_2.5', 'floors_3.0', 'floors_3.5',
       'waterfront_1.0', 'condition_2', 'condition_3', 'condition_4',
       'condition_5', 'grade_4', 'grade_5', 'grade_6', 'grade_7', 'grade_8',
       'grade_9', 'grade_10', 'grade_11', 'grade_12', 'grade_13']

# List of binary categorical variables
binary_categorical_vars = ['waterfront_1.0']

# List to store correlation coefficients
correlation_matrix = []

# Iterate over all combinations of variables
for var1 in continuous_vars + ordinal_categorical_vars + binary_categorical_vars:
    for var2 in continuous_vars + ordinal_categorical_vars + binary_categorical_vars:
        # Calculate correlation coefficient
        corr_coef = calculate_correlation(var1, var2)
        # Append to correlation matrix
        correlation_matrix.append([var1, var2, corr_coef])

# Convert correlation matrix to DataFrame
correlation_df = pd.DataFrame(correlation_matrix, columns=['Variable 1', 'Variable 2', 'Correlation Coefficient'

# Display correlation matrix
print(correlation_df)
```

```
          Variable 1        Variable 2  Correlation Coefficient
0           sqft_lot          sqft_lot                1.000000
1           sqft_lot       sqft_living                0.257216
2           sqft_lot  yr_built_Group 2               -0.062984
3           sqft_lot  yr_built_Group 3                0.019700
4           sqft_lot  yr_built_Group 4                0.215156
...              ...               ...                     ...
3020  waterfront_1.0          grade_10                0.055565
3021  waterfront_1.0          grade_11                0.069448
3022  waterfront_1.0          grade_12                0.078818
3023  waterfront_1.0          grade_13                0.008662
3024  waterfront_1.0    waterfront_1.0                0.996452

[3025 rows x 3 columns]
```

In [ ]:

In [107]:
```
import pandas as pd
import numpy as np
from scipy.stats import pearsonr, pointbiserialr, chi2_contingency

# Function to calculate Cramer's V
def cramers_v(confusion_matrix):
    chi2 = chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    phi2 = chi2 / n
    r, k = confusion_matrix.shape
    phi2corr = max(0, phi2 - ((k-1)*(r-1))/(n-1))
    rcorr = r - ((r-1)**2)/(n-1)
    kcorr = k - ((k-1)**2)/(n-1)
    return np.sqrt(phi2corr / min((kcorr-1), (rcorr-1)))

# Function to calculate correlation coefficients and p-values
def calculate_correlation(var1, var2):
    if var1 in continuous_vars and var2 in continuous_vars:
        # For continuous-continuous variables, use Pearson correlation coefficient
        corr_coef, p_value = pearsonr(df[var1], df[var2])
    elif var1 in continuous_vars and var2 in ordinal_categorical_vars:
        # For continuous-ordinal categorical variables, use point-biserial correlation coefficient
        corr_coef, p_value = pointbiserialr(df[var2], df[var1])
    elif var1 in ordinal_categorical_vars and var2 in continuous_vars:
        # For ordinal categorical-continuous variables, use point-biserial correlation coefficient
        corr_coef, p_value = pointbiserialr(df[var1], df[var2])
    elif var1 in binary_categorical_vars and var2 in continuous_vars:
        # For binary categorical-continuous variables, use point-biserial correlation coefficient
        corr_coef, p_value = pointbiserialr(df[var1], df[var2])
    elif var1 in ordinal_categorical_vars and var2 in ordinal_categorical_vars:
        # For ordinal categorical-ordinal categorical variables, calculate Cramer's V
        confusion_matrix = pd.crosstab(df[var1], df[var2])
        corr_coef = cramers_v(confusion_matrix)
        p_value = None  # Cramer's V doesn't have a direct p-value
    elif var1 in binary_categorical_vars and var2 in binary_categorical_vars:
        # For binary categorical-binary categorical variables, calculate Cramer's V
        confusion_matrix = pd.crosstab(df[var1], df[var2])
        corr_coef = cramers_v(confusion_matrix)
        p_value = None  # Cramer's V doesn't have a direct p-value
    else:
        # For all other combinations, return None
        corr_coef = None
        p_value = None
    return corr_coef, p_value

# List of continuous variables
continuous_vars = ['sqft_lot', 'sqft_living']

# List of ordinal categorical variables
ordinal_categorical_vars = ['yr_built_Group 2',
       'yr_built_Group 3', 'yr_built_Group 4', 'yr_built_Group 5',
       'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5', 'bedrooms_6',
       'bedrooms_7', 'bathrooms_0.75', 'bathrooms_1.0', 'bathrooms_1.25',
       'bathrooms_1.5', 'bathrooms_1.75', 'bathrooms_2.0', 'bathrooms_2.25',
       'bathrooms_2.5', 'bathrooms_2.75', 'bathrooms_3.0', 'bathrooms_3.25',
       'bathrooms_3.5', 'bathrooms_3.75', 'bathrooms_4.0', 'bathrooms_4.25',
       'bathrooms_4.5', 'bathrooms_4.75', 'bathrooms_5.0', 'bathrooms_5.25',
       'bathrooms_5.5', 'bathrooms_5.75', 'bathrooms_6.0', 'floors_1.5',
       'floors_2.0', 'floors_2.5', 'floors_3.0', 'floors_3.5',
       'waterfront_1.0', 'condition_2', 'condition_3', 'condition_4',
       'condition_5', 'grade_4', 'grade_5', 'grade_6', 'grade_7', 'grade_8',
       'grade_9', 'grade_10', 'grade_11', 'grade_12', 'grade_13']

# List of binary categorical variables
binary_categorical_vars = ['waterfront_1.0']

# List to store correlation coefficients and p-values
correlation_matrix = []

# Iterate over all combinations of variables
for var1 in continuous_vars + ordinal_categorical_vars + binary_categorical_vars:
    for var2 in continuous_vars + ordinal_categorical_vars + binary_categorical_vars:
        # Calculate correlation coefficient and p-value
        corr_coef, p_value = calculate_correlation(var1, var2)
        # Append to correlation matrix
        correlation_matrix.append([var1, var2, corr_coef, p_value])

# Convert correlation matrix to DataFrame
correlation_df = pd.DataFrame(correlation_matrix, columns=['Variable 1', 'Variable 2', 'Correlation Coefficient'

# Display correlation matrix
print(correlation_df)
```

	Variable 1	Variable 2	Correlation Coefficient	P-Value
0	sqft_lot	sqft_lot	1.000000	0.000000e+00
1	sqft_lot	sqft_living	0.257216	1.127104e-282
2	sqft_lot	yr_built_Group 2	-0.062984	4.814634e-18
3	sqft_lot	yr_built_Group 3	0.019700	6.821015e-03
4	sqft_lot	yr_built_Group 4	0.215156	2.185006e-196
...	...	...	...	...
3020	waterfront_1.0	grade_10	0.055565	NaN
3021	waterfront_1.0	grade_11	0.069448	NaN
3022	waterfront_1.0	grade_12	0.078818	NaN
3023	waterfront_1.0	grade_13	0.008662	NaN
3024	waterfront_1.0	waterfront_1.0	0.996452	NaN

[3025 rows x 4 columns]

In [ ]:

In [108]:
```
rt necessary libraries
 pandas as pd
 numpy as np
cipy.stats import pearsonr, pointbiserialr, chi2_contingency

tion to calculate Cramer's V
amers_v(confusion_matrix):
i2 = chi2_contingency(confusion_matrix)[0]
= confusion_matrix.sum().sum()
i2 = chi2 / n
 k = confusion_matrix.shape
i2corr = max(0, phi2 - ((k-1)*(r-1))/(n-1))
orr = r - ((r-1)**2)/(n-1)
orr = k - ((k-1)**2)/(n-1)
turn np.sqrt(phi2corr / min((kcorr-1), (rcorr-1)))

tion to calculate correlation coefficients and p-values
lculate_correlation(var1, var2):
 var1 in continuous_vars and var2 in continuous_vars:
  # For continuous-continuous variables, use Pearson correlation coefficient
  corr_coef, p_value = pearsonr(df[var1], df[var2])
if var1 in continuous_vars and var2 in ordinal_categorical_vars:
  # For continuous-ordinal categorical variables, use point-biserial correlation coefficient
  corr_coef, p_value = pointbiserialr(df[var2], df[var1])
if var1 in ordinal_categorical_vars and var2 in continuous_vars:
  # For ordinal categorical-continuous variables, use point-biserial correlation coefficient
  corr_coef, p_value = pointbiserialr(df[var1], df[var2])
if var1 in binary_categorical_vars and var2 in continuous_vars:
  # For binary categorical-continuous variables, use point-biserial correlation coefficient
  corr_coef, p_value = pointbiserialr(df[var1], df[var2])
if var1 in ordinal_categorical_vars and var2 in ordinal_categorical_vars:
  # For ordinal categorical-ordinal categorical variables, calculate Cramer's V
  confusion_matrix = pd.crosstab(df[var1], df[var2])
  corr_coef = cramers_v(confusion_matrix)
  p_value = None  # Cramer's V doesn't have a direct p-value
if var1 in binary_categorical_vars and var2 in binary_categorical_vars:
  # For binary categorical-binary categorical variables, calculate Cramer's V
  confusion_matrix = pd.crosstab(df[var1], df[var2])
  corr_coef = cramers_v(confusion_matrix)
  p_value = None  # Cramer's V doesn't have a direct p-value
se:
  # For all other combinations, return None
  corr_coef = None
  p_value = None
turn corr_coef, p_value

 of continuous variables
uous_vars = ['sqft_lot', 'sqft_living']

 of ordinal categorical variables
l_categorical_vars = ['yr_built_Group 2',
 'yr_built_Group 3', 'yr_built_Group 4', 'yr_built_Group 5',
 'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5', 'bedrooms_6',
 'bedrooms_7', 'bathrooms_0.75', 'bathrooms_1.0', 'bathrooms_1.25',
 'bathrooms_1.5', 'bathrooms_1.75', 'bathrooms_2.0', 'bathrooms_2.25',
 'bathrooms_2.5', 'bathrooms_2.75', 'bathrooms_3.0', 'bathrooms_3.25',
 'bathrooms_3.5', 'bathrooms_3.75', 'bathrooms_4.0', 'bathrooms_4.25',
 'bathrooms_4.5', 'bathrooms_4.75', 'bathrooms_5.0', 'bathrooms_5.25',
 'bathrooms_5.5', 'bathrooms_5.75', 'bathrooms_6.0', 'floors_1.5',
 'floors_2.0', 'floors_2.5', 'floors_3.0', 'floors_3.5',
 'waterfront_1.0', 'condition_2', 'condition_3', 'condition_4',
 'condition_5', 'grade_4', 'grade_5', 'grade_6', 'grade_7', 'grade_8',
 'grade_9', 'grade_10', 'grade_11', 'grade_12', 'grade_13']

 of binary categorical variables
_categorical_vars = ['waterfront_1.0']

 to store correlation coefficients and p-values
ation_matrix = []

ate over all combinations of variables
r1 in continuous_vars + ordinal_categorical_vars + binary_categorical_vars:
r var2 in continuous_vars + ordinal_categorical_vars + binary_categorical_vars:
  # Calculate correlation coefficient and p-value
  corr_coef, p_value = calculate_correlation(var1, var2)
  # Append to correlation matrix
  correlation_matrix.append([var1, var2, corr_coef, p_value])

ert correlation matrix to DataFrame
ation_df = pd.DataFrame(correlation_matrix, columns=['Variable 1', 'Variable 2', 'Correlation Coefficient', 'P-Va

er correlation matrix to display only p-values < 0.05
icant_correlations = correlation_df[correlation_df['P-Value'] < 0.05]

lay significant correlations
significant_correlations)
```

```
          Variable 1        Variable 2  Correlation Coefficient        P-Value
0           sqft_lot          sqft_lot                1.000000   0.000000e+00
1           sqft_lot       sqft_living                0.257216  1.127104e-282
2           sqft_lot  yr_built_Group 2               -0.062984   4.814634e-18
3           sqft_lot  yr_built_Group 3                0.019700   6.821015e-03
4           sqft_lot  yr_built_Group 4                0.215156  2.185006e-196
...              ...               ...                     ...           ...
2861        grade_12       sqft_living                0.220004  1.821867e-205
2915        grade_13          sqft_lot                0.023812   1.074233e-03
2916        grade_13       sqft_living                0.118905   2.387630e-60
2970  waterfront_1.0          sqft_lot                0.071088   1.450820e-22
2971  waterfront_1.0       sqft_living                0.107487   1.393830e-49

[188 rows x 4 columns]
```

In [109]:
```
# Filter correlation matrix to display only rows where p-value is not None and p-value < 0.05
significant_correlations = correlation_df[(correlation_df['P-Value'].notna()) & (correlation_df['P-Value'] < 0.0

# Display significant correlations
print(significant_correlations)
```

```
          Variable 1        Variable 2  Correlation Coefficient        P-Value
0           sqft_lot          sqft_lot                1.000000   0.000000e+00
1           sqft_lot       sqft_living                0.257216  1.127104e-282
2           sqft_lot  yr_built_Group 2               -0.062984   4.814634e-18
3           sqft_lot  yr_built_Group 3                0.019700   6.821015e-03
4           sqft_lot  yr_built_Group 4                0.215156  2.185006e-196
...              ...               ...                     ...           ...
2861        grade_12       sqft_living                0.220004  1.821867e-205
2915        grade_13          sqft_lot                0.023812   1.074233e-03
2916        grade_13       sqft_living                0.118905   2.387630e-60
2970  waterfront_1.0          sqft_lot                0.071088   1.450820e-22
2971  waterfront_1.0       sqft_living                0.107487   1.393830e-49

[188 rows x 4 columns]
```

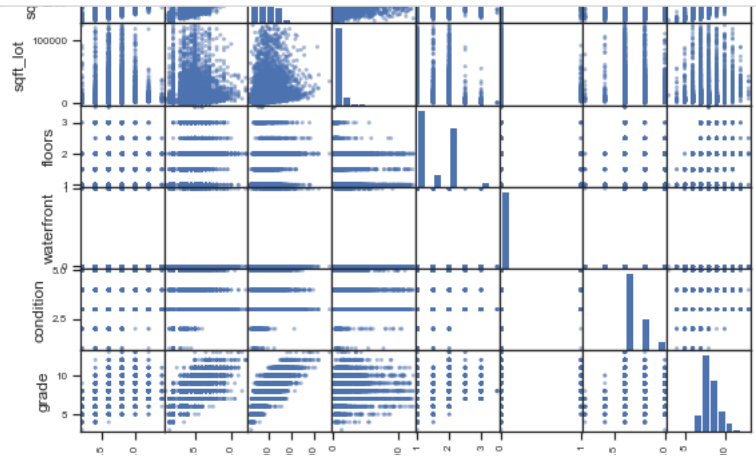
In [110]:
```
# Filter correlation matrix to display only p-values < 0.05
significant_correlations = correlation_df[correlation_df['P-Value'] < 0.05]

# Display significant correlations
print(significant_correlations)
```

```
          Variable 1        Variable 2  Correlation Coefficient        P-Value
0           sqft_lot          sqft_lot                1.000000   0.000000e+00
1           sqft_lot       sqft_living                0.257216  1.127104e-282
2           sqft_lot  yr_built_Group 2               -0.062984   4.814634e-18
3           sqft_lot  yr_built_Group 3                0.019700   6.821015e-03
4           sqft_lot  yr_built_Group 4                0.215156  2.185006e-196
...              ...               ...                     ...           ...
2861        grade_12       sqft_living                0.220004  1.821867e-205
2915        grade_13          sqft_lot                0.023812   1.074233e-03
2916        grade_13       sqft_living                0.118905   2.387630e-60
2970  waterfront_1.0          sqft_lot                0.071088   1.450820e-22
2971  waterfront_1.0       sqft_living                0.107487   1.393830e-49

[188 rows x 4 columns]
```

In [111]:
```
y.stats import pointbiserialr, chi2_contingency
mpy as np

 the list of variables
s_vars = ['sqft_lot', 'sqft_living']
ategorical_vars = ['bedrooms', 'floors', 'condition', 'grade']

n to calculate Point-biserial correlation coefficient
late_point_biserial_corr(var1, var2):
coef, p_val = pointbiserialr(df[var1], df[var2])
n corr_coef, p_val

n to calculate Cramer's V
late_cramers_v(var1, var2):
stat, p_val, dof, expected = chi2_contingency(pd.crosstab(df[var1], df[var2]))
um(pd.crosstab(df[var1], df[var2]).sum())
p.sqrt(chi2_stat / (n * (min(2, len(pd.crosstab(df[var1], df[var2]).index)) - 1)))
n v, p_val

 through each combination of variables
nuous_var in continuous_vars:
rdinal_categorical_var in ordinal_categorical_vars:
 Calculate point-biserial correlation coefficient
orr_coef, p_val = calculate_point_biserial_corr(ordinal_categorical_var, continuous_var)
rint(f"Point-biserial correlation coefficient between {ordinal_categorical_var} and {continuous_var}: {corr_coef}

culate point-biserial correlation coefficient for waterfront with continuous variables
coef, p_val = calculate_point_biserial_corr('waterfront', continuous_var)
(f"Point-biserial correlation coefficient between waterfront and {continuous_var}: {corr_coef}, p-value: {p_val}"

()

al_categorical_var1 in ordinal_categorical_vars:
rdinal_categorical_var2 in ordinal_categorical_vars:
f ordinal_categorical_var1 != ordinal_categorical_var2:
    # Calculate Cramer's V
    v, p_val = calculate_cramers_v(ordinal_categorical_var1, ordinal_categorical_var2)
    print(f"Cramer's V between {ordinal_categorical_var1} and {ordinal_categorical_var2}: {v}, p-value: {p_val}")

culate Cramer's V for waterfront with ordinal categorical variables
val = calculate_cramers_v(ordinal_categorical_var1, 'waterfront')
(f"Cramer's V between {ordinal_categorical_var1} and waterfront: {v}, p-value: {p_val}")

()
```

```
Point-biserial correlation coefficient between bedrooms and sqft_lot: 0.09693835937949412, p-value: 1.296591958
0917018e-40
Point-biserial correlation coefficient between floors and sqft_lot: -0.07713709043814228, p-value: 2.7215525956
19433e-26
Point-biserial correlation coefficient between condition and sqft_lot: 0.02634265294353161, p-value: 0.00029680
842810290954
Point-biserial correlation coefficient between grade and sqft_lot: 0.17263447188983186, p-value: 4.265787220458
5616e-126
Point-biserial correlation coefficient between waterfront and sqft_lot: 0.0710879680584535, p-value: 1.45082037
8452249e-22

Point-biserial correlation coefficient between bedrooms and sqft_living: 0.6029110869636265, p-value: 0.0
Point-biserial correlation coefficient between floors and sqft_living: 0.35599067956417113, p-value: 0.0
Point-biserial correlation coefficient between condition and sqft_living: -0.05444295706299572, p-value: 7.3312
56267020449e-14
Point-biserial correlation coefficient between grade and sqft_living: 0.7634189684680721, p-value: 0.0
Point-biserial correlation coefficient between waterfront and sqft_living: 0.10748731165572234, p-value: 1.3938
301211213346e-49

Cramer's V between bedrooms and floors: 0.3093287355458438, p-value: 0.0
Cramer's V between bedrooms and condition: 0.09757692222068866, p-value: 8.847076595907641e-26
Cramer's V between bedrooms and grade: 0.5178975057947593, p-value: 0.0
Cramer's V between bedrooms and waterfront: 0.034490184467876, p-value: 0.0010094241682934897

Cramer's V between floors and bedrooms: 0.3093287355458438, p-value: 0.0
Cramer's V between floors and condition: 0.35979749491164753, p-value: 0.0
Cramer's V between floors and grade: 0.5445652417504078, p-value: 0.0
Cramer's V between floors and waterfront: 0.025872743046254445, p-value: 0.02715965081332697

Cramer's V between condition and bedrooms: 0.09757692222068864, p-value: 8.847076595907766e-26
Cramer's V between condition and floors: 0.3597974949116476, p-value: 0.0
Cramer's V between condition and grade: 0.2572064286127341, p-value: 1.2681159509106112e-235
Cramer's V between condition and waterfront: 0.024251124992145694, p-value: 0.025550502361808745

Cramer's V between grade and bedrooms: 0.5178975057947593, p-value: 0.0
Cramer's V between grade and floors: 0.5445652417504078, p-value: 0.0
Cramer's V between grade and condition: 0.2572064286127341, p-value: 1.2681159509106112e-235
Cramer's V between grade and waterfront: 0.13092531416224462, p-value: 1.835882947994246e-63
```

In [112]:
```
# Selecting the predictor variables
data_pred = df[['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'condition', 'grade']]

# Displaying the scatter matrix
pd.plotting.scatter_matrix(data_pred, figsize=[9, 9])
plt.show()
```



In [113]:
```
data_pred.corr()
```

Out[113]:

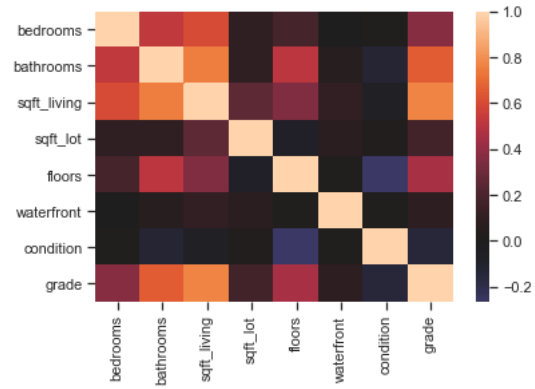
	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	grade
bedrooms	1.000000	0.528292	0.602911	0.096938	0.185432	-0.003409	0.022411	0.376464
bathrooms	0.528292	1.000000	0.749478	0.103413	0.508657	0.065275	-0.126256	0.663595
sqft_living	0.602911	0.749478	1.000000	0.257216	0.355991	0.107487	-0.054443	0.763419
sqft_lot	0.096938	0.103413	0.257216	1.000000	-0.077137	0.071088	0.026343	0.172634
floors	0.185432	0.508657	0.355991	-0.077137	1.000000	0.023838	-0.266019	0.457283
waterfront	-0.003409	0.065275	0.107487	0.071088	0.023838	1.000000	0.017348	0.087597
condition	0.022411	-0.126256	-0.054443	0.026343	-0.266019	0.017348	1.000000	-0.144429
grade	0.376464	0.663595	0.763419	0.172634	0.457283	0.087597	-0.144429	1.000000

In [114]:
```
abs(data_pred.corr()) > 0.70
```

Out[114]:

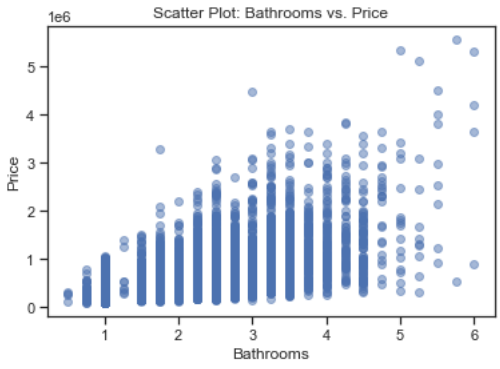
	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	grade
bedrooms	True	False	False	False	False	False	False	False
bathrooms	False	True	True	False	False	False	False	False
sqft_living	False	True	True	False	False	False	False	True
sqft_lot	False	False	False	True	False	False	False	False
floors	False	False	False	False	True	False	False	False
waterfront	False	False	False	False	False	True	False	False
condition	False	False	False	False	False	False	True	False
grade	False	False	True	False	False	False	False	True

In [115]:
```
import seaborn as sns

# Calculate the correlation matrix based on the scatter matrix
corr_matrix = data_pred.corr()

# Create a heatmap
sns.heatmap(corr_matrix, center=0)
plt.show()
```



In [116]:
```
import numpy as np

# Calculate correlation coefficients
corr_bathrooms_price = df['bathrooms'].corr(df['price'])
corr_sqft_living_price = df['sqft_living'].corr(df['price'])

print("Correlation coefficient between bathrooms and price:", corr_bathrooms_price)
print("Correlation coefficient between sqft_living and price:", corr_sqft_living_price)

# Scatter plot for bathrooms vs. price
plt.scatter(df['bathrooms'], df['price'], alpha=0.5)
plt.xlabel('Bathrooms')
plt.ylabel('Price')
plt.title('Scatter Plot: Bathrooms vs. Price')
plt.show()

# Scatter plot for sqft_living vs. price
plt.scatter(df['sqft_living'], df['price'], alpha=0.5)
plt.xlabel('Sqft Living')
plt.ylabel('Price')
plt.title('Scatter Plot: Sqft Living vs. Price')
plt.show()
```

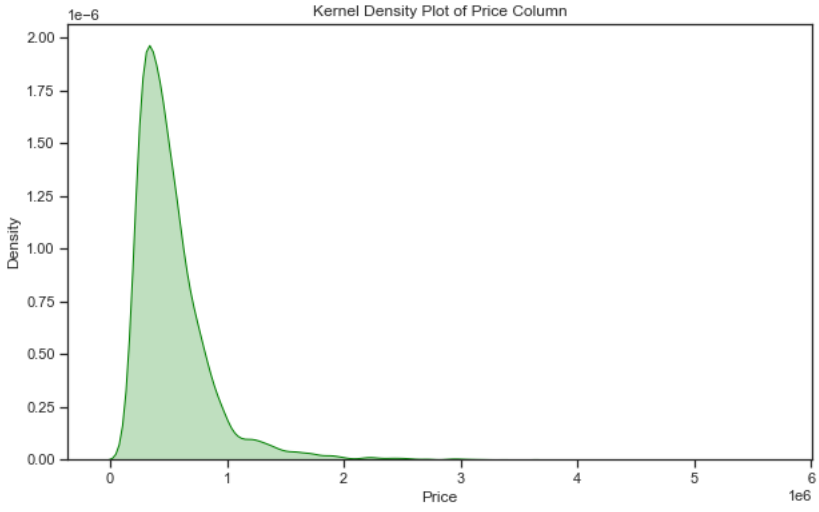
```
Correlation coefficient between bathrooms and price: 0.5167379371417508
Correlation coefficient between sqft_living and price: 0.6966964088093314
```



In [117]:
```
# Calculate correlation coefficients
corr_sqft_living_price = df['sqft_living'].corr(df['price'])
corr_grade_price = df['grade'].corr(df['price'])

print("Correlation coefficient between sqft_living and price:", corr_sqft_living_price)
print("Correlation coefficient between grade and price:", corr_grade_price)

# Scatter plot for sqft_living vs. price
plt.scatter(df['sqft_living'], df['price'], alpha=0.5)
plt.xlabel('Sqft Living')
plt.ylabel('Price')
plt.title('Scatter Plot: Sqft Living vs. Price')
plt.show()

# Scatter plot for grade vs. price
plt.scatter(df['grade'], df['price'], alpha=0.5)
plt.xlabel('Grade')
plt.ylabel('Price')
plt.title('Scatter Plot: Grade vs. Price')
plt.show()
```

```
Correlation coefficient between sqft_living and price: 0.6966964088093314
Correlation coefficient between grade and price: 0.6711287950634893
```

In [118]:
```
import pandas as pd
import matplotlib.pyplot as plt  # Import the plt module
import seaborn as sns

# Create a kernel density plot for the 'price' column
plt.figure(figsize=(10, 6))
sns.kdeplot(df['price'], color='green', fill=True)

# Add labels and title
plt.xlabel('Price')
plt.ylabel('Density')
plt.title('Kernel Density Plot of Price Column')

# Show the plot
plt.show()
```

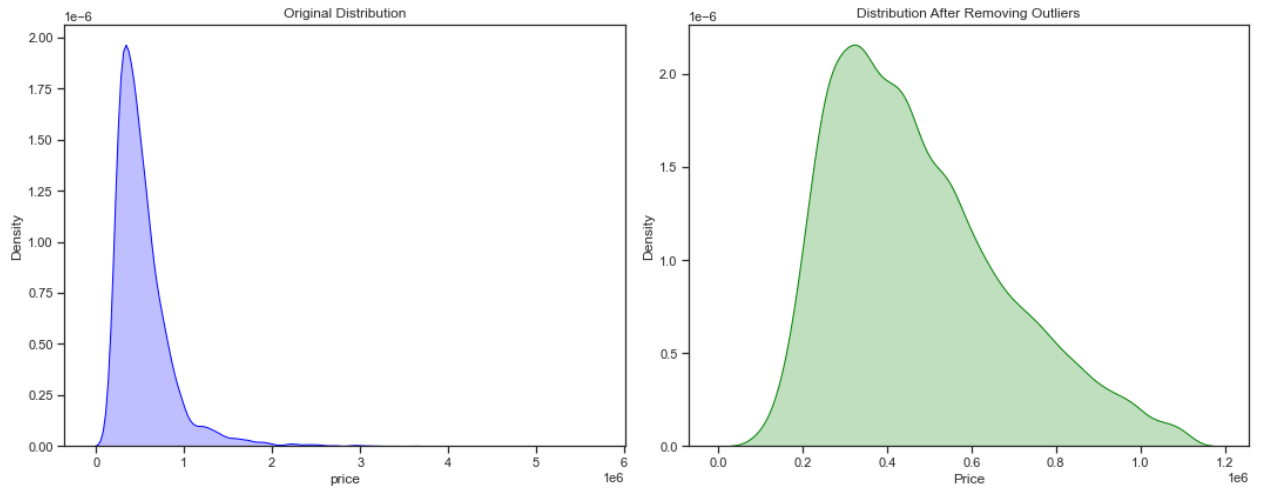


price data is right skewed so I'll remove the data above and below the 99.7 and -99.7 percentiles.

In [ ]:

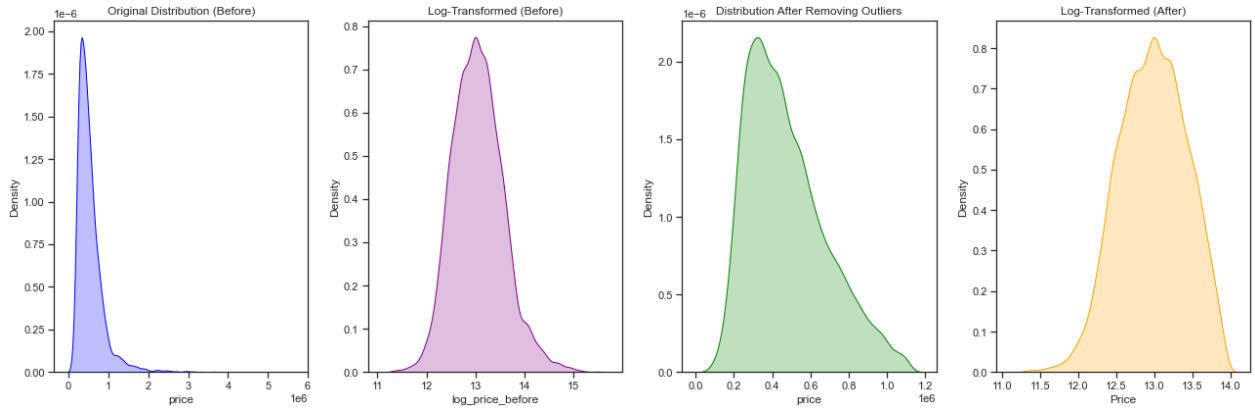
In [119]:
```
import pandas as pd

# Assuming 'df' is your DataFrame with a column 'price'
q1 = df['price'].quantile(0.25)
q3 = df['price'].quantile(0.75)
iqr = q3 - q1

# Define the threshold for outliers
threshold = 1.5 * iqr

# Identify and remove outliers
df_no_outliers = df[(df['price'] >= q1 - threshold) & (df['price'] <= q3 + threshold)]

# 'df_no_outliers' now contains the data without outliers
```

In [120]:
```
import matplotlib.pyplot as plt
import seaborn as sns


# Create a kernel density plot for the 'price' column before and after removing outliers
plt.figure(figsize=(15, 6))

# Original distribution
plt.subplot(1, 2, 1)
sns.kdeplot(df['price'], color='blue', fill=True)
plt.title('Original Distribution')

# Distribution after removing outliers
plt.subplot(1, 2, 2)
sns.kdeplot(df_no_outliers['price'], color='green', fill=True)
plt.title('Distribution After Removing Outliers')

# Add labels and title
plt.xlabel('Price')
plt.ylabel('Density')

# Show the plots
plt.tight_layout()
plt.show()
```



In [121]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import normaltest



# Apply log transformation to the 'price' column before removing outliers
df['log_price_before'] = np.log(df['price'])

# Apply log transformation to the 'price' column after removing outliers
df_no_outliers['log_price_after'] = np.log(df_no_outliers['price'])

# Create a kernel density plot for the log-transformed 'price' column before and after removing outliers
plt.figure(figsize=(18, 6))

# Original distribution (before removing outliers)
plt.subplot(1, 4, 1)
sns.kdeplot(df['price'], color='blue', fill=True)
plt.title('Original Distribution (Before)')

# Log-transformed distribution before removing outliers
plt.subplot(1, 4, 2)
sns.kdeplot(df['log_price_before'], color='purple', fill=True)
plt.title('Log-Transformed (Before)')

# Distribution after removing outliers
plt.subplot(1, 4, 3)
sns.kdeplot(df_no_outliers['price'], color='green', fill=True)
plt.title('Distribution After Removing Outliers')

# Log-transformed distribution after removing outliers
plt.subplot(1, 4, 4)
sns.kdeplot(df_no_outliers['log_price_after'], color='orange', fill=True)
plt.title('Log-Transformed (After)')

# Add labels and title
plt.xlabel('Price')
plt.ylabel('Density')

# Calculate normality tests
normaltest_before = normaltest(df['log_price_before'].dropna())
normaltest_after = normaltest(df_no_outliers['log_price_after'].dropna())

# Print normality test results
print("Normality Test (Before Removing Outliers): p-value =", normaltest_before.pvalue)
print("Normality Test (After Removing Outliers): p-value =", normaltest_after.pvalue)

# Show the plots
plt.tight_layout()
plt.show()
```

```
<ipython-input-121-91bde0314d58>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_no_outliers['log_price_after'] = np.log(df_no_outliers['price'])

Normality Test (Before Removing Outliers): p-value = 1.0328188212206866e-159
Normality Test (After Removing Outliers): p-value = 2.468244431610787e-56
```

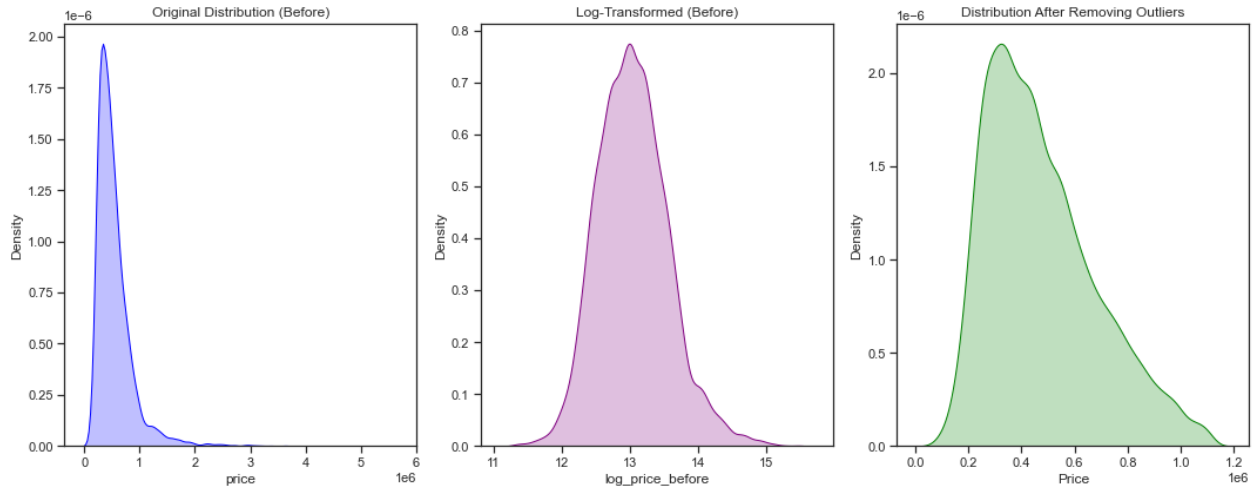


In [122]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import normaltest

# Assuming 'df' is your DataFrame with a column 'price'
# ... (your previous code)

# Apply log transformation to the 'price' column before removing outliers
df['log_price_before'] = np.log(df['price'])

# Apply log transformation to the 'price' column after removing outliers
df_no_outliers['log_price_after'] = np.log(df_no_outliers['price'])

# Create a kernel density plot for the log-transformed 'price' column before and after removing outliers
plt.figure(figsize=(18, 6))

# Original distribution (before removing outliers)
plt.subplot(1, 4, 1)
sns.kdeplot(df['price'], color='blue', fill=True)
plt.title('Original Distribution (Before)')

# Log-transformed distribution before removing outliers
plt.subplot(1, 4, 2)
sns.kdeplot(df['log_price_before'], color='purple', fill=True)
plt.title('Log-Transformed (Before)')

# Distribution after removing outliers
plt.subplot(1, 4, 3)
sns.kdeplot(df_no_outliers['price'], color='green', fill=True)
plt.title('Distribution After Removing Outliers')

# Log-transformed distribution after removing outliers
plt.subplot(1, 4, 4)
sns.kdeplot(df_no_outliers['log_price_after'], color='orange', fill=True)
plt.title('Log-Transformed (After)')

# Add labels and title
plt.xlabel('Price')
plt.ylabel('Density')

# Calculate normality tests
normaltest_before = normaltest(df['log_price_before'].dropna())
normaltest_after = normaltest(df_no_outliers['log_price_after'].dropna())

# Print normality test results
plt.suptitle('Comparison of Log-Transformed Distributions Before and After Removing Outliers')
print("Normality Test (Before Removing Outliers): p-value =", normaltest_before.pvalue)
print("Normality Test (After Removing Outliers): p-value =", normaltest_after.pvalue)

# Show the plots
plt.tight_layout(rect=[0, 0, 1, 0.96])  # Adjusting the layout to include the suptitle
plt.show()
```

```
<ipython-input-122-11d3dbeae7ff>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_no_outliers['log_price_after'] = np.log(df_no_outliers['price'])

Normality Test (Before Removing Outliers): p-value = 1.0328188212206866e-159
Normality Test (After Removing Outliers): p-value = 2.468244431610787e-56
```

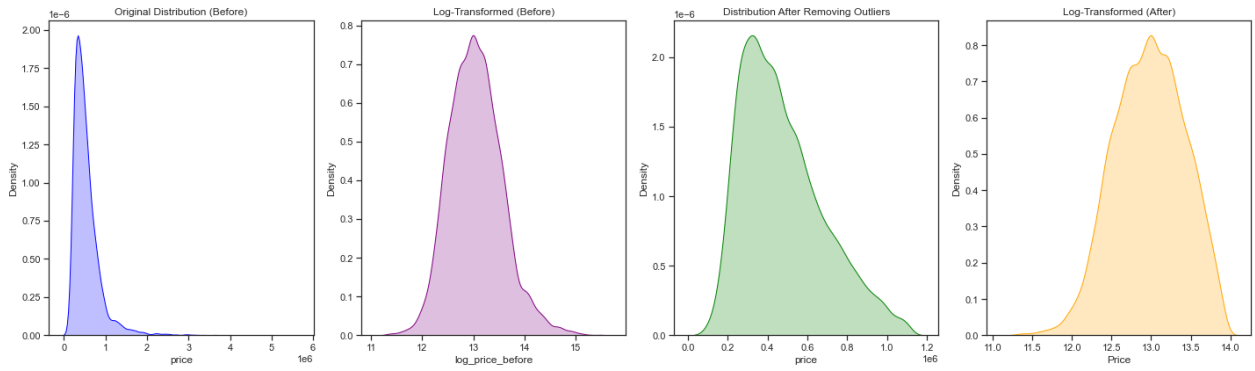
In [123]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import normaltest

# Assuming 'df' is your DataFrame with a column 'price'
# ... (your previous code)

# Create a kernel density plot for the log-transformed 'price' column before and after removing outliers
plt.figure(figsize=(15, 6))

# Original distribution (before removing outliers)
plt.subplot(1, 3, 1)
sns.kdeplot(df['price'], color='blue', fill=True)
plt.title('Original Distribution (Before)')

# Apply log transformation to the 'price' column before removing outliers
df['log_price_before'] = np.log(df['price'])
plt.subplot(1, 3, 2)
sns.kdeplot(df['log_price_before'], color='purple', fill=True)
plt.title('Log-Transformed (Before)')

# Distribution after removing outliers
plt.subplot(1, 3, 3)
sns.kdeplot(df_no_outliers['price'], color='green', fill=True)
plt.title('Distribution After Removing Outliers')

# Calculate normality tests
normaltest_before = normaltest(df['log_price_before'].dropna())
normaltest_after = normaltest(df_no_outliers['log_price_after'].dropna())

# Print normality test results
print("Normality Test (Before Removing Outliers): p-value =", normaltest_before.pvalue)
print("Normality Test (After Removing Outliers): p-value =", normaltest_after.pvalue)

# Add labels and title
plt.xlabel('Price')
plt.ylabel('Density')

# Show the plots
plt.tight_layout()
plt.show()
```

```
Normality Test (Before Removing Outliers): p-value = 1.0328188212206866e-159
Normality Test (After Removing Outliers): p-value = 2.468244431610787e-56
```



In [124]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Assuming 'df' is your DataFrame with a column 'price'
# ... (your previous code)

# Create a kernel density plot for the log-transformed 'price' column before and after removing outliers
plt.figure(figsize=(20, 6))

# Original distribution (before removing outliers)
plt.subplot(1, 4, 1)
sns.kdeplot(df['price'], color='blue', fill=True)
plt.title('Original Distribution (Before)')

# Apply log transformation to the 'price' column before removing outliers
df['log_price_before'] = np.log(df['price'])
plt.subplot(1, 4, 2)
sns.kdeplot(df['log_price_before'], color='purple', fill=True)
plt.title('Log-Transformed (Before)')

# Distribution after removing outliers
plt.subplot(1, 4, 3)
sns.kdeplot(df_no_outliers['price'], color='green', fill=True)
plt.title('Distribution After Removing Outliers')

# Apply log transformation to the 'price' column after removing outliers
df_no_outliers['log_price_after'] = np.log(df_no_outliers['price'])
plt.subplot(1, 4, 4)
sns.kdeplot(df_no_outliers['log_price_after'], color='orange', fill=True)
plt.title('Log-Transformed (After)')

# Add labels and title
plt.xlabel('Price')
plt.ylabel('Density')

# Show the plots
plt.tight_layout()
plt.show()
```

```
<ipython-input-124-0870b333e430>:28: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_no_outliers['log_price_after'] = np.log(df_no_outliers['price'])
```



In [ ]:

now the price has been reduced to just under 1.2 million which allows us to focus the moelling on the less extreme property prices with less examples and higher capital investment requirements.

In [ ]:

In [125]:
```
import matplotlib.pyplot as plt



# Create a box plot for the 'price' column
plt.figure(figsize=(10, 6))
plt.boxplot(df['price'], vert=False)

# Add labels and title
plt.xlabel('Price')
plt.title('Box Plot for Price Column')

# Show the plot
plt.show()
```