Final Project Submission

Please fill out:

- Student name: Group 5 -
- Student pace: part time
- Scheduled project review date/time: 03/01/2024
- Instructor name:
- Blog post URL:

Business Understanding

To develop a predictive model that accurately estimates the sale price of houses in King County, Washington, based on various features such as the number of bedrooms, bathrooms, square footage, location attributes (like waterfront proximity), house condition, grade, year built and renovated. This model aims to assist potential buyers, sellers, and real estate stakeholders in understanding and predicting house prices within the region.

Importing Dependancies

In [1]:
```
import itertools
import numpy as np
import pandas as pd
from numbers import Number
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import pickle

import warnings
warnings.filterwarnings('ignore')
```

Data Understanding

In [2]:
```
# Load the dataset
house_data = pd.read_csv("data/kc_house_data.csv")
```

In [3]:
```
# structure of dataset
house_data.shape
```

Out[3]:
```
(21597, 21)
```

In [4]:
```
# Get information about the dataset, including data types and missing values
house_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 1   date           21597 non-null  object 
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64  
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64  
 6   sqft_lot       21597 non-null  int64  
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  object 
 9   view           21534 non-null  object 
 10  condition      21597 non-null  object 
 11  grade          21597 non-null  object 
 12  sqft_above     21597 non-null  int64  
 13  sqft_basement  21597 non-null  object 
 14  yr_built       21597 non-null  int64  
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64  
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64  
 20  sqft_lot15     21597 non-null  int64  
dtypes: float64(6), int64(9), object(6)
memory usage: 3.5+ MB
```

In [5]:
```
# Summary statistics of numerical columns
house_data.describe().transpose()
```

Out[5]:

	count	mean	std	min	25%	5
id	21597.0	4.580474e+09	2.876736e+09	1.000102e+06	2.123049e+09	3.904930e+
price	21597.0	5.402966e+05	3.673681e+05	7.800000e+04	3.220000e+05	4.500000e+
bedrooms	21597.0	3.373200e+00	9.262989e-01	1.000000e+00	3.000000e+00	3.000000e+
bathrooms	21597.0	2.115826e+00	7.689843e-01	5.000000e-01	1.750000e+00	2.250000e+
sqft_living	21597.0	2.080322e+03	9.181061e+02	3.700000e+02	1.430000e+03	1.910000e+
sqft_lot	21597.0	1.509941e+04	4.141264e+04	5.200000e+02	5.040000e+03	7.618000e+
floors	21597.0	1.494096e+00	5.396828e-01	1.000000e+00	1.000000e+00	1.500000e+
sqft_above	21597.0	1.788597e+03	8.277598e+02	3.700000e+02	1.190000e+03	1.560000e+
yr_built	21597.0	1.971000e+03	2.937523e+01	1.900000e+03	1.951000e+03	1.975000e+
yr_renovated	17755.0	8.363678e+01	3.999464e+02	0.000000e+00	0.000000e+00	0.000000e+
zipcode	21597.0	9.807795e+04	5.351307e+01	9.800100e+04	9.803300e+04	9.806500e+
lat	21597.0	4.756009e+01	1.385518e-01	4.715590e+01	4.747110e+01	4.757180e+
long	21597.0	-1.222140e+02	1.407235e-01	-1.225190e+02	-1.223280e+02	-1.222310e+
sqft_living15	21597.0	1.986620e+03	6.852305e+02	3.990000e+02	1.490000e+03	1.840000e+
sqft_lot15	21597.0	1.275828e+04	2.727444e+04	6.510000e+02	5.100000e+03	7.620000e+

Data Preparation

In [6]:
```
#check for duplicate values
house_data.duplicated().sum()
```

Out[6]:
```
0
```

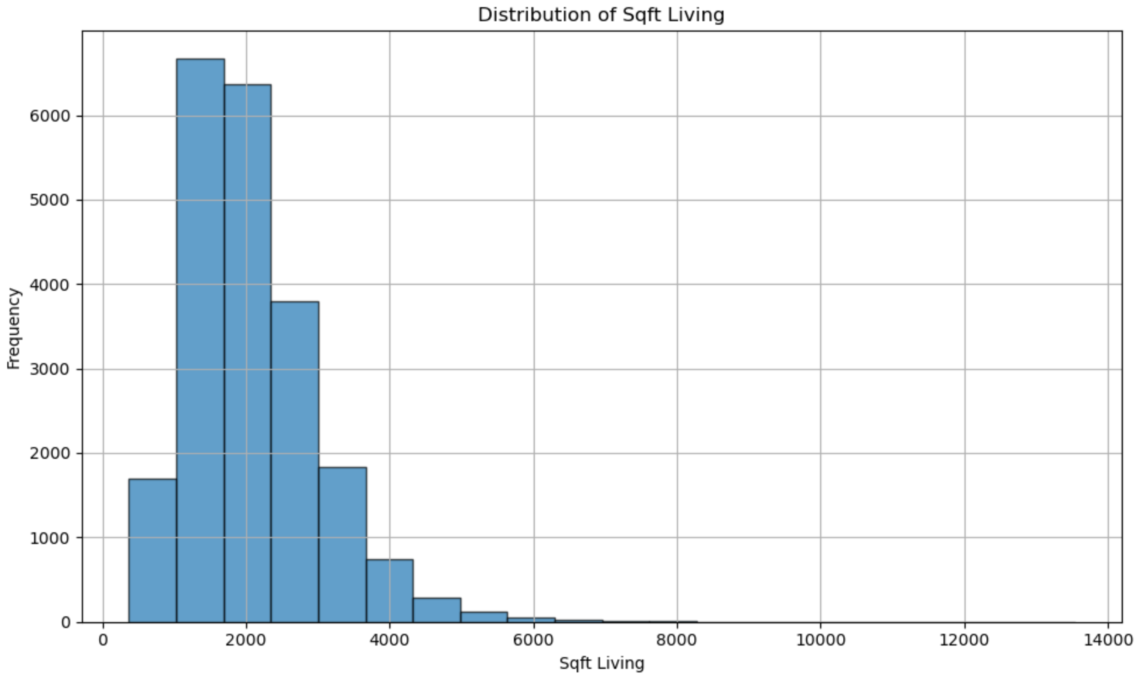
In [7]:
```
# Check for missing values in the dataset
house_data.isnull().sum()
```

Out[7]:
```
id                  0
date                0
price               0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront       2376
view               63
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_built            0
yr_renovated     3842
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
dtype: int64
```

In [8]:
```
# fill null values in waterfront column with the mode
house_data['waterfront'].fillna(house_data['waterfront'].mode()[0], inplace=

# Convert 'yr_renovated' column from float to integer, handling NaN values
house_data['yr_renovated'] = house_data['yr_renovated'].fillna(0).astype(int

# Filling missing values for 'yr_renovated' with 0
house_data['yr_renovated'].fillna(house_data['yr_renovated'].median(), inpla
```

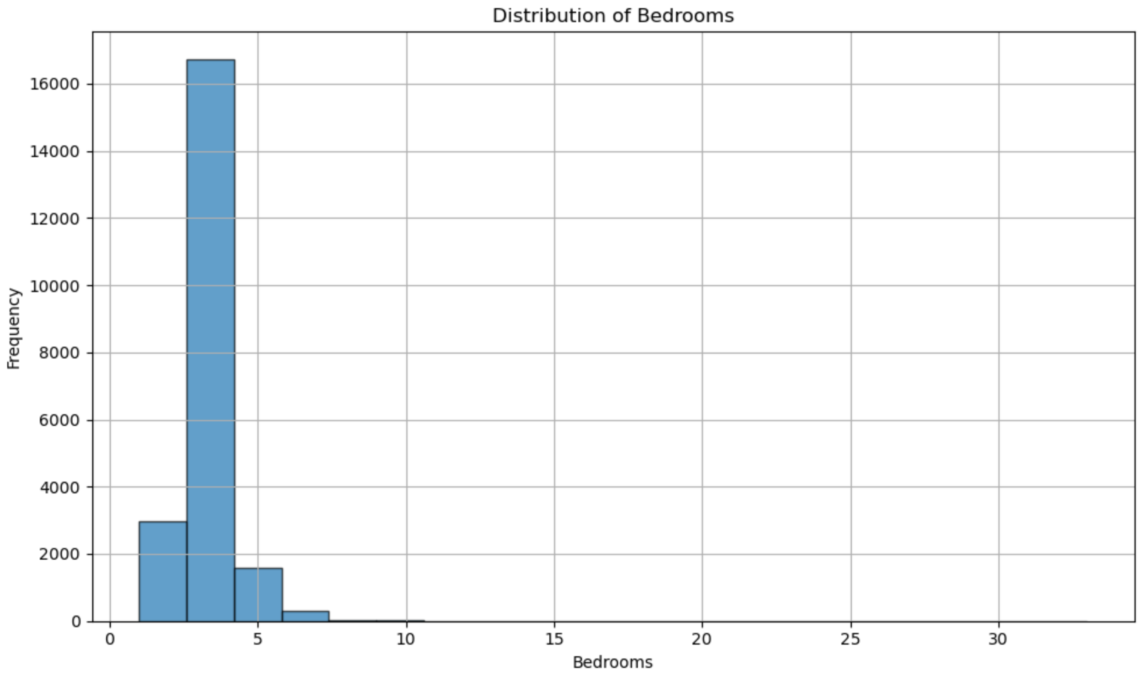
Visualizations

In [9]:
```
sqft_living = house_data["sqft_living"]

# Create the histogram
plt.figure(figsize=(10, 6))
plt.hist(sqft_living, bins=20, edgecolor="black", alpha=0.7)
plt.xlabel("Sqft Living")
plt.ylabel("Frequency")
plt.title("Distribution of Sqft Living")

plt.grid(True)
plt.tight_layout()

plt.show()
```



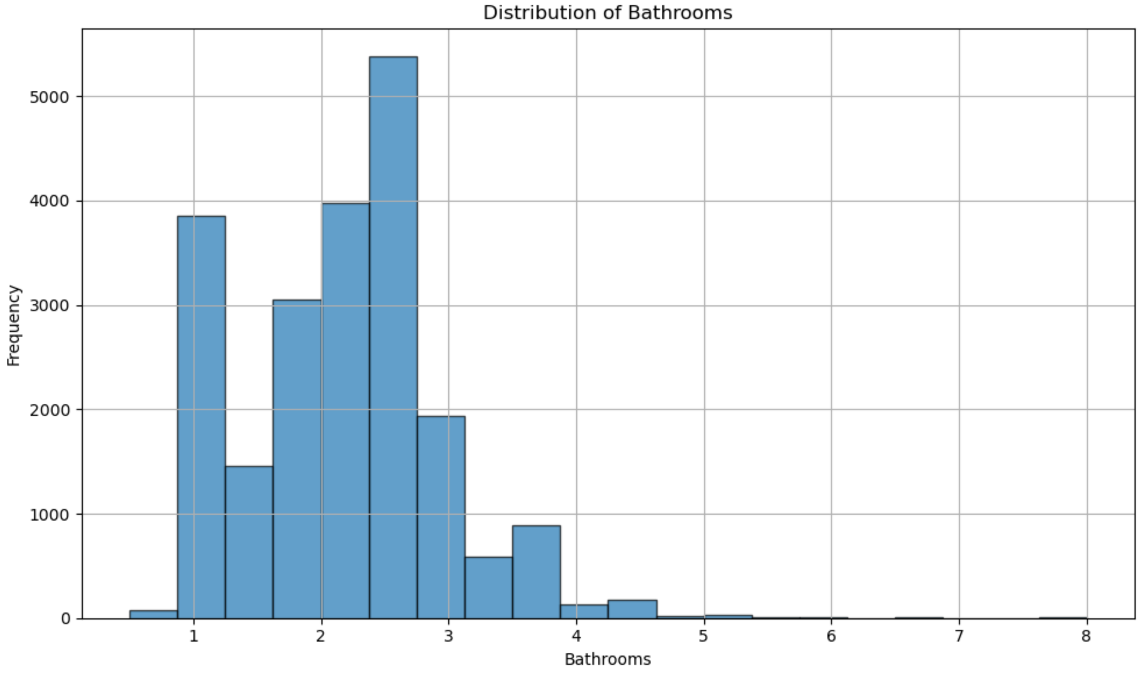
It shows a positive skew, meaning that while most homes fall within a smaller square footage range, there are a few homes with exceptionally large square footage. These larger values, though fewer, extend the tail of the distribution to the right, indicating a positive skew. This suggests that while the majority of homes are smaller, there are a few larger homes that are exceptions.

In [10]:
```
bedrooms = house_data['bedrooms']

# Create the histogram
plt.figure(figsize=(10, 6))
plt.hist(bedrooms, bins=20, edgecolor="black", alpha=0.7)
plt.xlabel("Bedrooms")
plt.ylabel("Frequency")
plt.title("Distribution of Bedrooms")

plt.grid(True)
plt.tight_layout()

plt.show()
```



There is a significant peak at 2 to 3 bedrooms, indicating that most homes have this number of bedrooms.

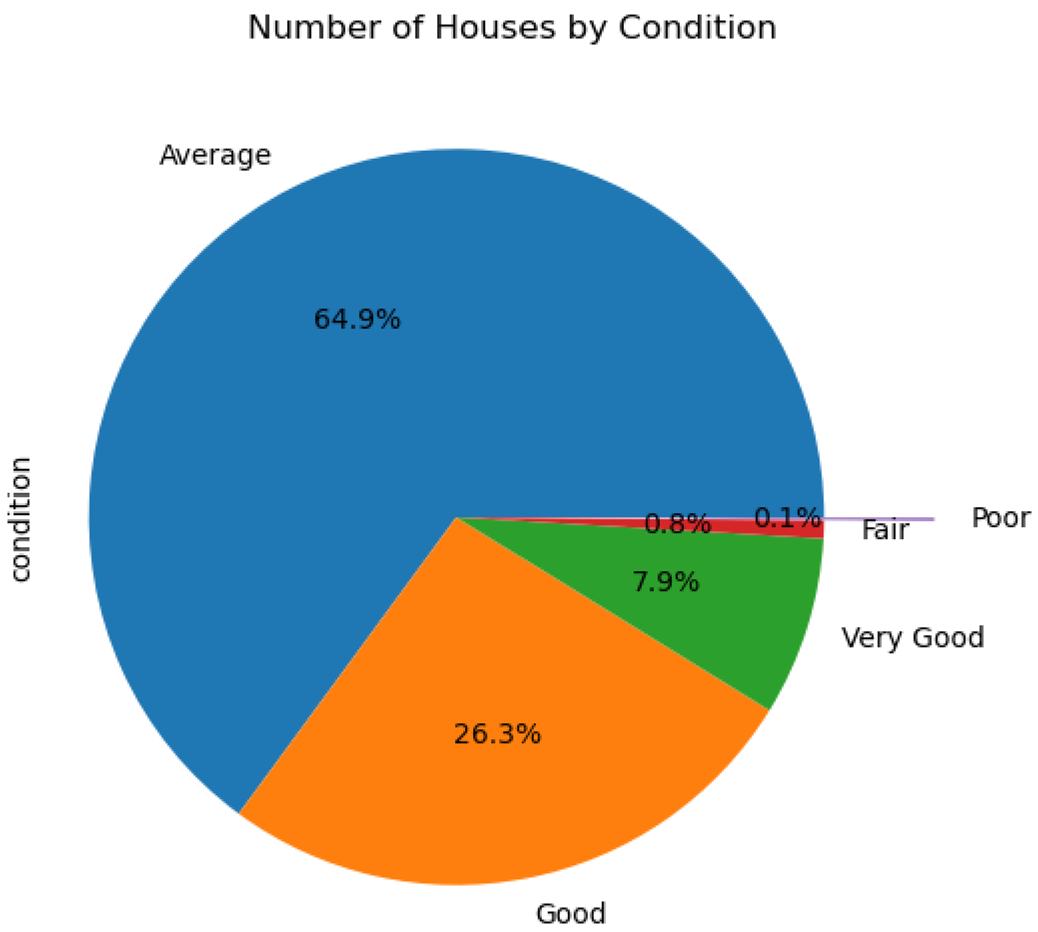
In [11]:
```
#checking the distribution of number of bathrooms

bathrooms = house_data['bathrooms']

# Create the histogram
plt.figure(figsize=(10, 6))
plt.hist(bathrooms, bins=20, edgecolor="black", alpha=0.7)
plt.xlabel("Bathrooms")
plt.ylabel("Frequency")
plt.title("Distribution of Bathrooms")

plt.grid(True)
plt.tight_layout()

plt.show()
```



Most Common: Houses with 3 bathrooms have the highest frequency.

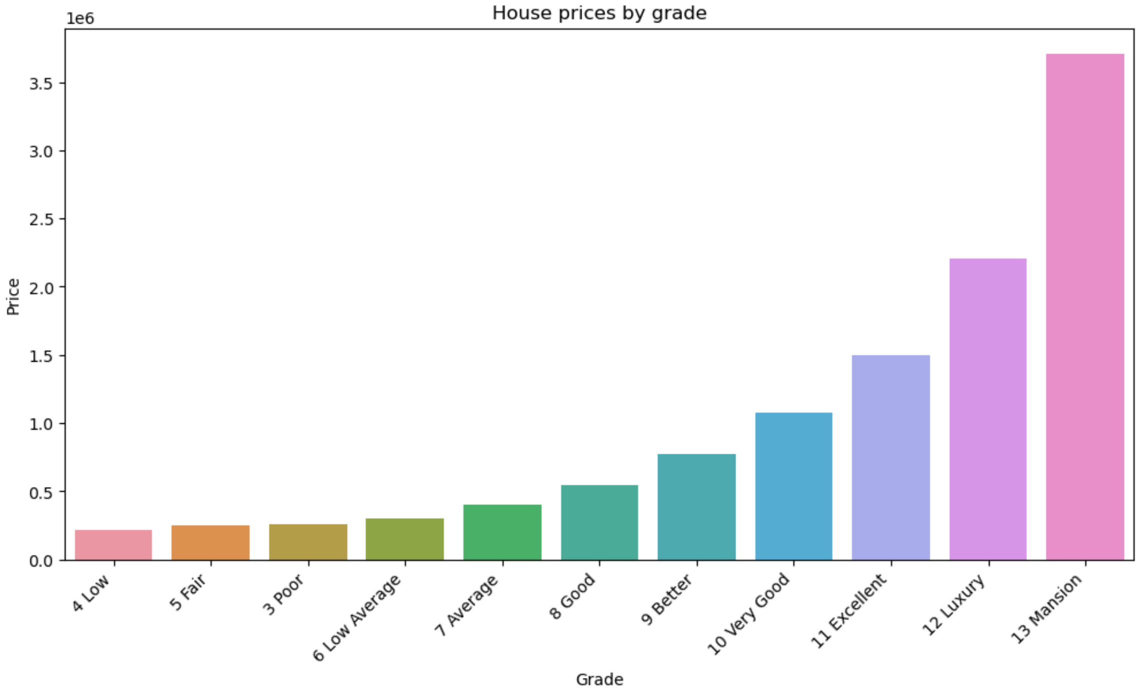
Lower Frequencies: Counts beyond 4 bathrooms have much lower frequencies, suggesting they are less common.

In [12]:
```
# Pie chart to show number of houses by condition
house_counts = house_data['condition'].value_counts()
plt.figure(figsize=(6, 6))
explode = (0, 0, 0, 0, 0.3)
house_counts.plot(kind='pie', autopct="%1.1f%%", explode=explode, title="Nur
plt.axis('equal')
plt.show()
```



Most of the houses are in average to good condition.

In [13]:
```
# Distribution of houses by grade
ave_prices = house_data[['grade', 'price']].groupby('grade').mean().sort_val
plt.figure(figsize=(12, 6))
sns.barplot(x='grade', y='price', data=ave_prices, label='Price')
plt.xticks(rotation=45, ha='right')
plt.xlabel("Grade")
plt.ylabel("Price")
plt.title('House prices by grade')
plt.show()
```



Houses under average

Feature Engineering

In [14]:
```
# Explore specific columns related to the features mentioned in the business
relevant_columns = ['date','price', 'bedrooms', 'bathrooms', 'floors', 'sqft
                    'waterfront', 'condition', 'grade', 'yr_built', 'yr_reno
relevant_data = house_data[relevant_columns]
relevant_data
```

Out[14]:

	date	price	bedrooms	bathrooms	floors	sqft_living	sqft_lot	waterfront	c
0	10/13/2014	221900.0	3	1.00	1.0	1180	5650	NO	
1	12/9/2014	538000.0	3	2.25	2.0	2570	7242	NO	
2	2/25/2015	180000.0	2	1.00	1.0	770	10000	NO	
3	12/9/2014	604000.0	4	3.00	1.0	1960	5000	NO	
4	2/18/2015	510000.0	3	2.00	1.0	1680	8080	NO	
...	...	...	...	...	...	...	...	...	
21592	5/21/2014	360000.0	3	2.50	3.0	1530	1131	NO	
21593	2/23/2015	400000.0	4	2.50	2.0	2310	5813	NO	
21594	6/23/2014	402101.0	2	0.75	2.0	1020	1350	NO	
21595	1/16/2015	400000.0	3	2.50	2.0	1600	2388	NO	
21596	10/15/2014	325000.0	2	0.75	2.0	1020	1076	NO	

21597 rows × 12 columns

In [15]:
```
# identify unique values in the columns
print(house_data["waterfront"].unique())
print()
print(house_data["condition"].unique())
print()
print(house_data["grade"].unique())
```

```
['NO' 'YES']

['Average' 'Very Good' 'Good' 'Poor' 'Fair']

['7 Average' '6 Low Average' '8 Good' '11 Excellent' '9 Better' '5 Fair'
 '10 Very Good' '12 Luxury' '4 Low' '3 Poor' '13 Mansion']
```

In [16]:
```
# Create a new column indicating if renovation happened (1) or not (0)
relevant_data['renovated'] = (relevant_data['yr_renovated'] != 0).astype(int
relevant_data
```

Out[16]:

	date	price	bedrooms	bathrooms	floors	sqft_living	sqft_lot	waterfront	c
0	10/13/2014	221900.0	3	1.00	1.0	1180	5650	NO	
1	12/9/2014	538000.0	3	2.25	2.0	2570	7242	NO	
2	2/25/2015	180000.0	2	1.00	1.0	770	10000	NO	
3	12/9/2014	604000.0	4	3.00	1.0	1960	5000	NO	
4	2/18/2015	510000.0	3	2.00	1.0	1680	8080	NO	
...	...	...	...	...	...	...	...	...	
21592	5/21/2014	360000.0	3	2.50	3.0	1530	1131	NO	
21593	2/23/2015	400000.0	4	2.50	2.0	2310	5813	NO	
21594	6/23/2014	402101.0	2	0.75	2.0	1020	1350	NO	
21595	1/16/2015	400000.0	3	2.50	2.0	1600	2388	NO	
21596	10/15/2014	325000.0	2	0.75	2.0	1020	1076	NO	

21597 rows × 13 columns

In [17]:
```
# Convert 'date' column to datetime
relevant_data['date'] = pd.to_datetime(relevant_data['date'])

# Define a function to get the season based on the month
def get_season(month):
    if month in [3, 4, 5]:  # Spring: March, April, May
        return 'Spring'
    elif month in [6, 7, 8]:  # Summer: June, July, August
        return 'Summer'
    elif month in [9, 10, 11]:  # Fall: September, October, November
        return 'Fall'
    else:  # Winter: December, January, February
        return 'Winter'

# Create 'season' column based on 'date'
relevant_data['season'] = relevant_data['date'].dt.month.apply(get_season).c
relevant_data.head(15)
```

Out[17]:

	date	price	bedrooms	bathrooms	floors	sqft_living	sqft_lot	waterfront	conditior
0	2014-10-13	221900.0	3	1.00	1.0	1180	5650	NO	Average
1	2014-12-09	538000.0	3	2.25	2.0	2570	7242	NO	Average
2	2015-02-25	180000.0	2	1.00	1.0	770	10000	NO	Average
3	2014-12-09	604000.0	4	3.00	1.0	1960	5000	NO	Very Gooc
4	2015-02-18	510000.0	3	2.00	1.0	1680	8080	NO	Average
5	2014-05-12	1230000.0	4	4.50	1.0	5420	101930	NO	Average
6	2014-06-27	257500.0	3	2.25	2.0	1715	6819	NO	Average
7	2015-01-15	291850.0	3	1.50	1.0	1060	9711	NO	Average
8	2015-04-15	229500.0	3	1.00	1.0	1780	7470	NO	Average
9	2015-03-12	323000.0	3	2.50	2.0	1890	6560	NO	Average
10	2015-04-03	662500.0	3	2.50	1.0	3560	9796	NO	Average
11	2014-05-27	468000.0	2	1.00	1.0	1160	6000	NO	Gooc
12	2014-05-28	310000.0	3	1.00	1.5	1430	19901	NO	Gooc
13	2014-10-07	400000.0	3	1.75	1.0	1370	9680	NO	Gooc
14	2015-03-12	530000.0	5	2.00	1.5	1810	4850	NO	Average

In [18]:
```
# Replace 'No' with 0 and 'Yes' with 1 in the 'waterfront' column
relevant_data['waterfront'] = relevant_data['waterfront'].replace({'NO': 0,
relevant_data
```

Out[18]:

	date	price	bedrooms	bathrooms	floors	sqft_living	sqft_lot	waterfront	conditi
0	2014-10-13	221900.0	3	1.00	1.0	1180	5650	0	Avera
1	2014-12-09	538000.0	3	2.25	2.0	2570	7242	0	Avera
2	2015-02-25	180000.0	2	1.00	1.0	770	10000	0	Avera
3	2014-12-09	604000.0	4	3.00	1.0	1960	5000	0	V Gc
4	2015-02-18	510000.0	3	2.00	1.0	1680	8080	0	Avera
...	...	...	...	...	...	...	...	...	
21592	2014-05-21	360000.0	3	2.50	3.0	1530	1131	0	Avera
21593	2015-02-23	400000.0	4	2.50	2.0	2310	5813	0	Avera
21594	2014-06-23	402101.0	2	0.75	2.0	1020	1350	0	Avera
21595	2015-01-16	400000.0	3	2.50	2.0	1600	2388	0	Avera
21596	2014-10-15	325000.0	2	0.75	2.0	1020	1076	0	Avera

21597 rows × 14 columns

In [19]:
```
# Extract grade number and condition using regex
# Add grade number (grade_no) column
import re

relevant_data[['grade_no', 'condition']] = relevant_data['grade'].str.extrac
relevant_data
```

Out[19]:

	date	price	bedrooms	bathrooms	floors	sqft_living	sqft_lot	waterfront	conditi
0	2014-10-13	221900.0	3	1.00	1.0	1180	5650	0	Avera
1	2014-12-09	538000.0	3	2.25	2.0	2570	7242	0	Avera
2	2015-02-25	180000.0	2	1.00	1.0	770	10000	0	L Avera
3	2014-12-09	604000.0	4	3.00	1.0	1960	5000	0	Avera
4	2015-02-18	510000.0	3	2.00	1.0	1680	8080	0	Gc
...	...	...	...	...	...	...	...	...	
21592	2014-05-21	360000.0	3	2.50	3.0	1530	1131	0	Gc
21593	2015-02-23	400000.0	4	2.50	2.0	2310	5813	0	Gc
21594	2014-06-23	402101.0	2	0.75	2.0	1020	1350	0	Avera
21595	2015-01-16	400000.0	3	2.50	2.0	1600	2388	0	Gc
21596	2014-10-15	325000.0	2	0.75	2.0	1020	1076	0	Avera

21597 rows × 15 columns

In [20]:
```
# convert grade_no values to int
relevant_data['grade_no'] = relevant_data['grade_no'].astype('int64')
```

In [21]:
```
print(relevant_data.columns)
```

```
Index(['date', 'price', 'bedrooms', 'bathrooms', 'floors', 'sqft_living',
       'sqft_lot', 'waterfront', 'condition', 'grade', 'yr_built',
       'yr_renovated', 'renovated', 'season', 'grade_no'],
      dtype='object')
```

In [22]:
```
# Create a new feature for the age of the house
current_year = pd.Timestamp.now().year  # Get the current year
relevant_data['house_age'] = current_year - relevant_data['yr_built']

relevant_data.head(15)
```

Out[22]:

	date	price	bedrooms	bathrooms	floors	sqft_living	sqft_lot	waterfront	conditior
0	2014-10-13	221900.0	3	1.00	1.0	1180	5650	0	Average
1	2014-12-09	538000.0	3	2.25	2.0	2570	7242	0	Average
2	2015-02-25	180000.0	2	1.00	1.0	770	10000	0	Low Average
3	2014-12-09	604000.0	4	3.00	1.0	1960	5000	0	Average
4	2015-02-18	510000.0	3	2.00	1.0	1680	8080	0	Gooc
5	2014-05-12	1230000.0	4	4.50	1.0	5420	101930	0	Excellen
6	2014-06-27	257500.0	3	2.25	2.0	1715	6819	0	Average
7	2015-01-15	291850.0	3	1.50	1.0	1060	9711	0	Average
8	2015-04-15	229500.0	3	1.00	1.0	1780	7470	0	Average
9	2015-03-12	323000.0	3	2.50	2.0	1890	6560	0	Average
10	2015-04-03	662500.0	3	2.50	1.0	3560	9796	0	Gooc
11	2014-05-27	468000.0	2	1.00	1.0	1160	6000	0	Average
12	2014-05-28	310000.0	3	1.00	1.5	1430	19901	0	Average
13	2014-10-07	400000.0	3	1.75	1.0	1370	9680	0	Average
14	2015-03-12	530000.0	5	2.00	1.5	1810	4850	0	Average

In [23]:
```
# Drop irrelevant columns and assign the result back to engineered_data
relevant_data = relevant_data.drop(['date', 'grade', 'yr_renovated', 'yr_bui
relevant_data
```

Out[23]:

	price	bedrooms	bathrooms	floors	sqft_living	sqft_lot	waterfront	renovated	s
0	221900.0	3	1.00	1.0	1180	5650	0	0	
1	538000.0	3	2.25	2.0	2570	7242	0	1	
2	180000.0	2	1.00	1.0	770	10000	0	0	
3	604000.0	4	3.00	1.0	1960	5000	0	0	
4	510000.0	3	2.00	1.0	1680	8080	0	0	
...	...	...	...	...	...	...	...	...	
21592	360000.0	3	2.50	3.0	1530	1131	0	0	
21593	400000.0	4	2.50	2.0	2310	5813	0	0	
21594	402101.0	2	0.75	2.0	1020	1350	0	0	Su
21595	400000.0	3	2.50	2.0	1600	2388	0	0	
21596	325000.0	2	0.75	2.0	1020	1076	0	0	

21597 rows × 11 columns

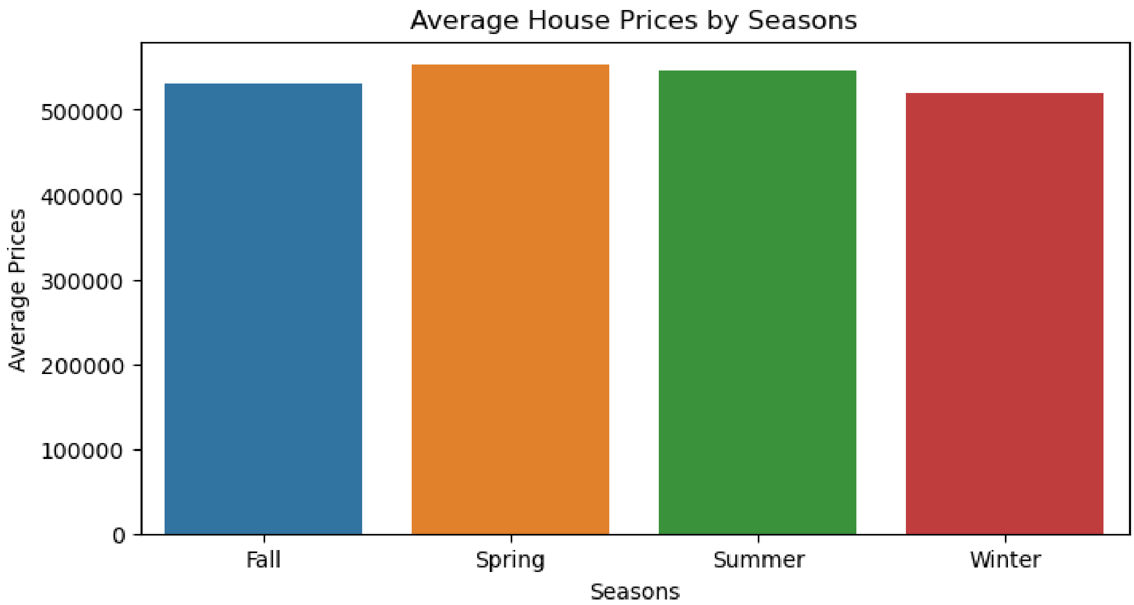
Data Visualizations

In [24]:
```
# # Pairplot
# sns.pairplot(relevant_data)
# plt.show()
```

In [25]:
```
relevant_data.price.describe()
```

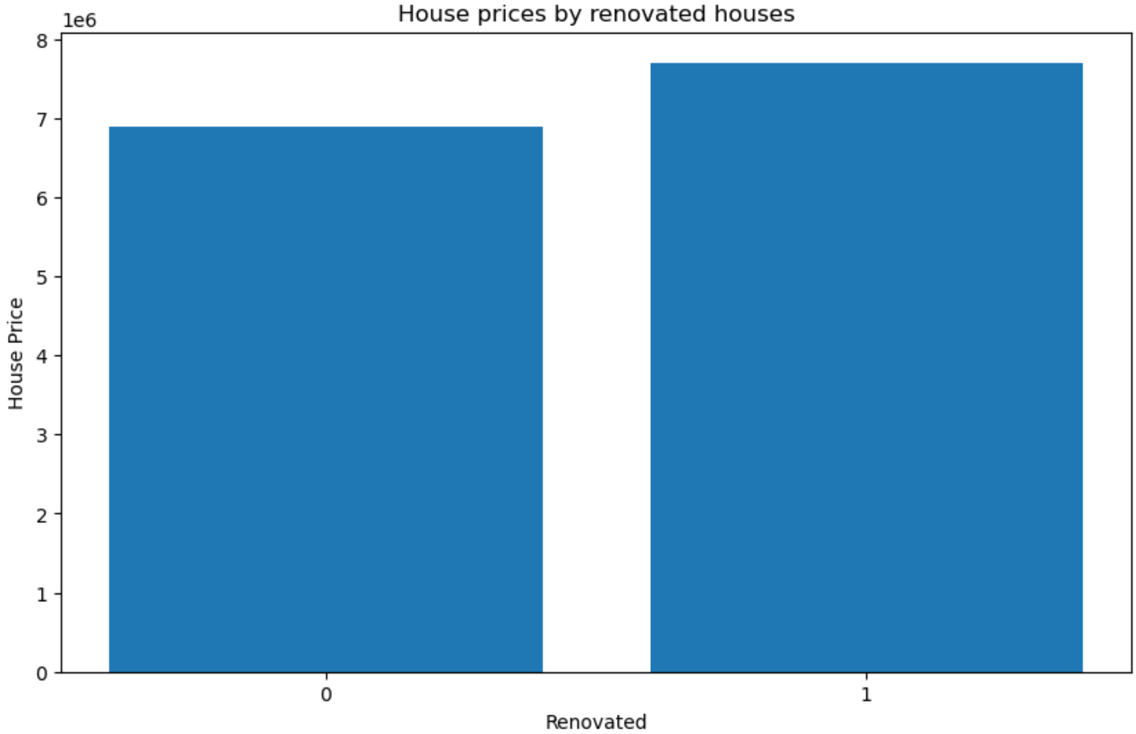
Out[25]:
```
count    2.159700e+04
mean     5.402966e+05
std      3.673681e+05
min      7.800000e+04
25%      3.220000e+05
50%      4.500000e+05
75%      6.450000e+05
max      7.700000e+06
Name: price, dtype: float64
```

In [26]:
```
# check the average house prices by season
average_prices = relevant_data.groupby('season')['price'].mean().reset_index

# Bar plot using Seaborn with average prices
plt.figure(figsize=(8, 4))
sns.barplot(x='season', y='price', data=average_prices)
plt.title('Average House Prices by Seasons')
plt.xlabel('Seasons')
plt.ylabel('Average Prices')
plt.savefig('house_grade.png', transparent=True)
plt.show()
```



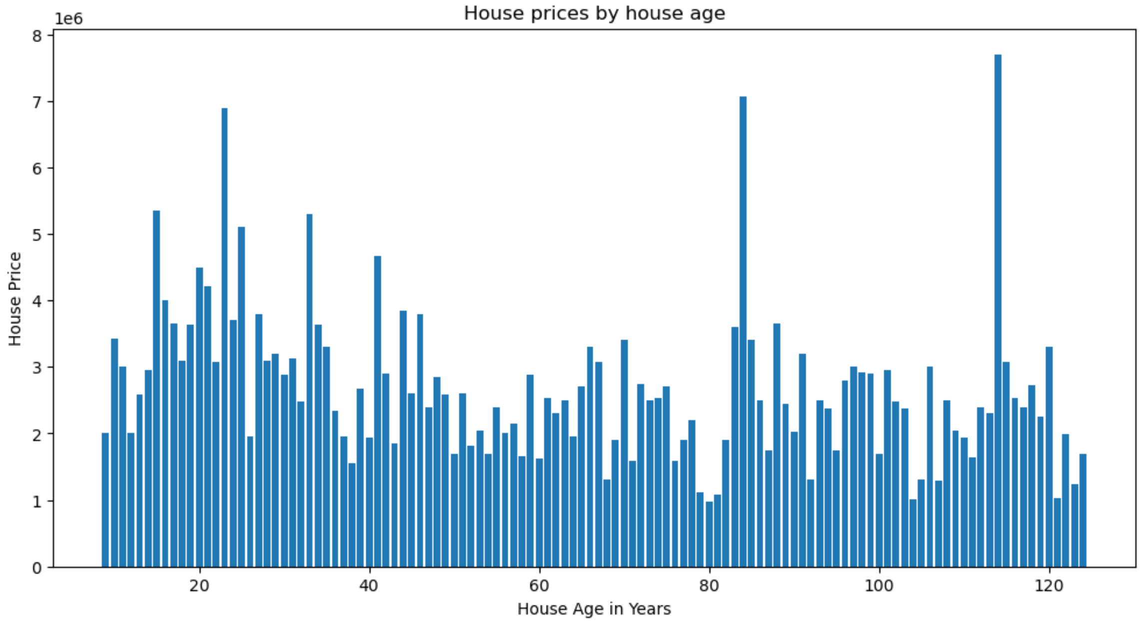
Average house prices are highest during spring and slightly lower during winter.

In [27]:
```
# checking the distribution of house prices by renovated houses
Renovated = relevant_data['renovated']
Price = relevant_data['price']
plt.figure(figsize=(10,6))
plt.bar(Renovated, Price)
plt.xlabel('Renovated')
plt.ylabel('House Price')
plt.title('House prices by renovated houses')
plt.xticks(ticks=[0, 1])
plt.show()
```



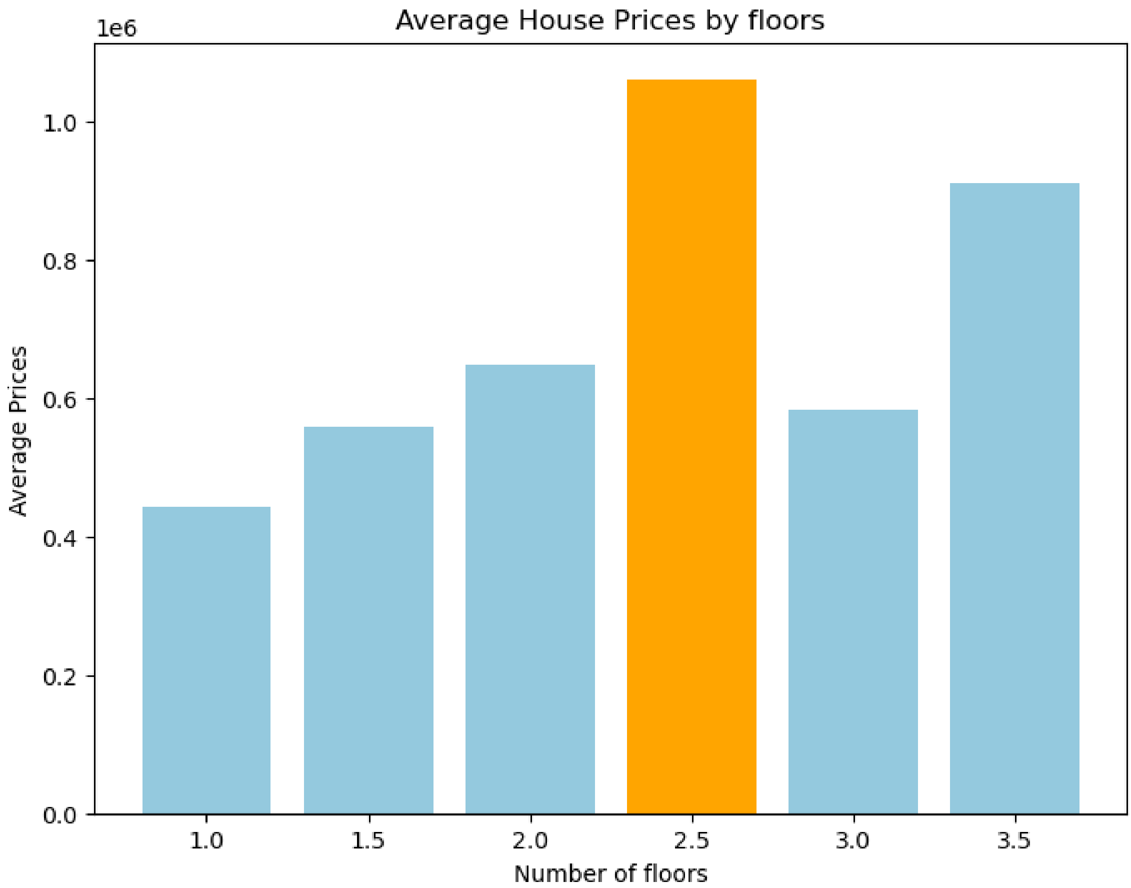
Renovated houses have a higher average price indicating renaovations had a small impact on the value of the house.

In [28]:
```
# Distribution of house price by age
House_age = relevant_data['house_age']
Price = relevant_data['price']
plt.figure(figsize=(12,6))
plt.bar(House_age, Price)
plt.xlabel('House Age in Years')
plt.ylabel('House Price')
plt.title('House prices by house age')

plt.show()
```



The houses are priced at 2 million dollars and above where the age is below 50 years. On the contrary majority of the houses over 50 years are priced below 2 million dollars.

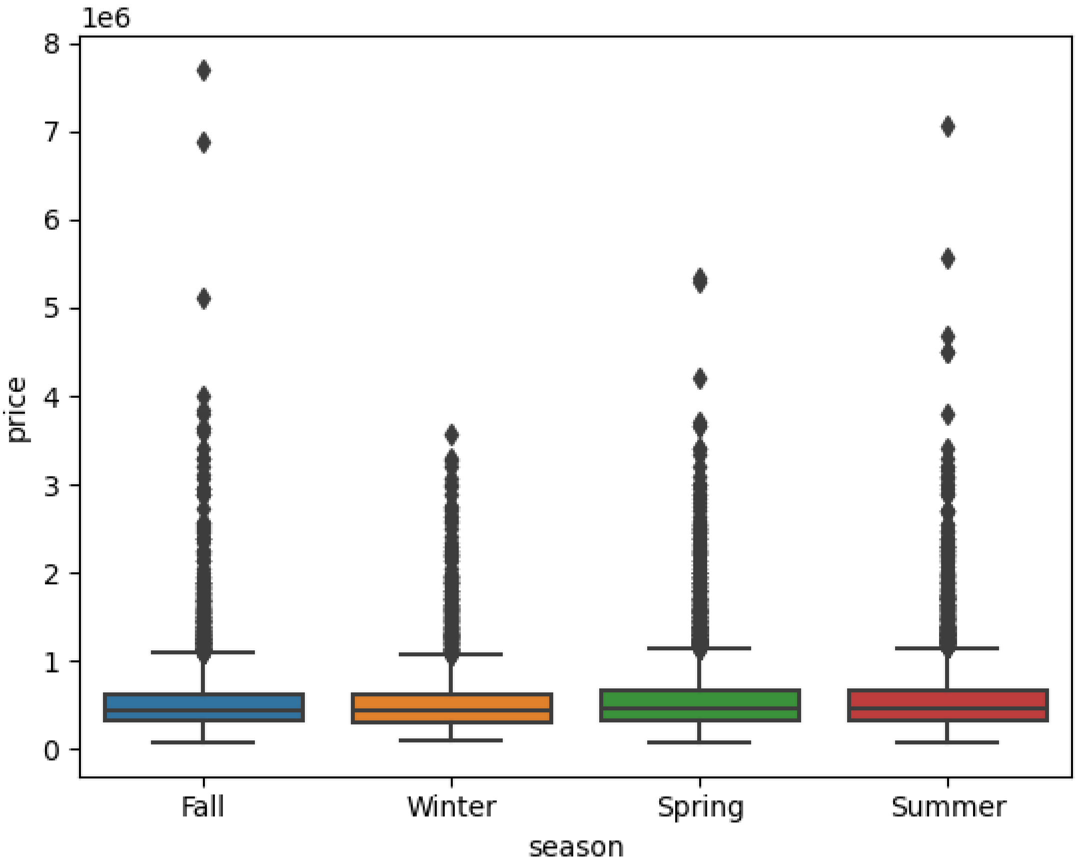
In [29]:
```
# Distribution of houses by floors
# Get the number of houses with each number of floors
average_prices = relevant_data.groupby('floors')['price'].mean().reset_index

# Find the index of the maximum value
max_index = average_prices['price'].idxmax()

# Bar plot using Seaborn with average prices
plt.figure(figsize=(8, 6))
sns.barplot(x='floors', y='price', data=average_prices, color='skyblue')

# Highlight the bar with the maximum value
plt.bar(max_index, average_prices.loc[max_index, 'price'], color='orange')

plt.title('Average House Prices by floors')
plt.xlabel('Number of floors')
plt.ylabel('Average Prices')

plt.show()
```



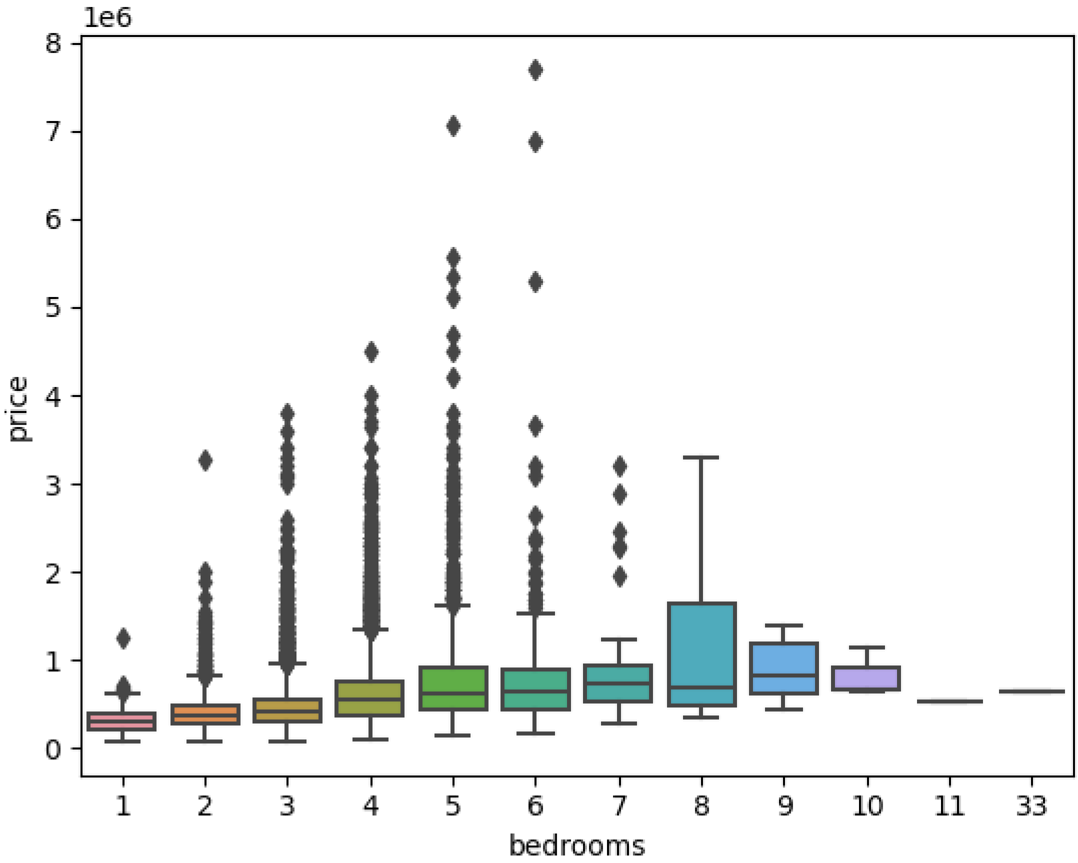
Houses with 2.5 floors are the most priced at slightly above 1 million dollars and the least priced houses have 1 floor.

Checking for outliers

In [30]:
```
# seasons and price
sns.boxplot(x='season', y='price', data=relevant_data)
plt.show()
```

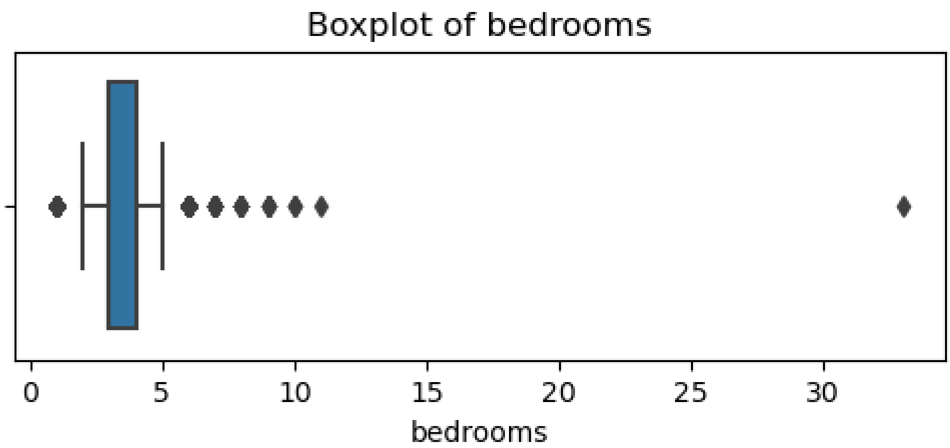


In [31]:
```
# bedrooms and price
sns.boxplot(x=relevant_data['bedrooms'], y=relevant_data['price'])
plt.show()
```



Removing outliers using z-score

In [32]:
```
plt.figure(figsize=(6, 2))
sns.boxplot(x='bedrooms', data=relevant_data)
plt.title('Boxplot of bedrooms')
plt.show()
```

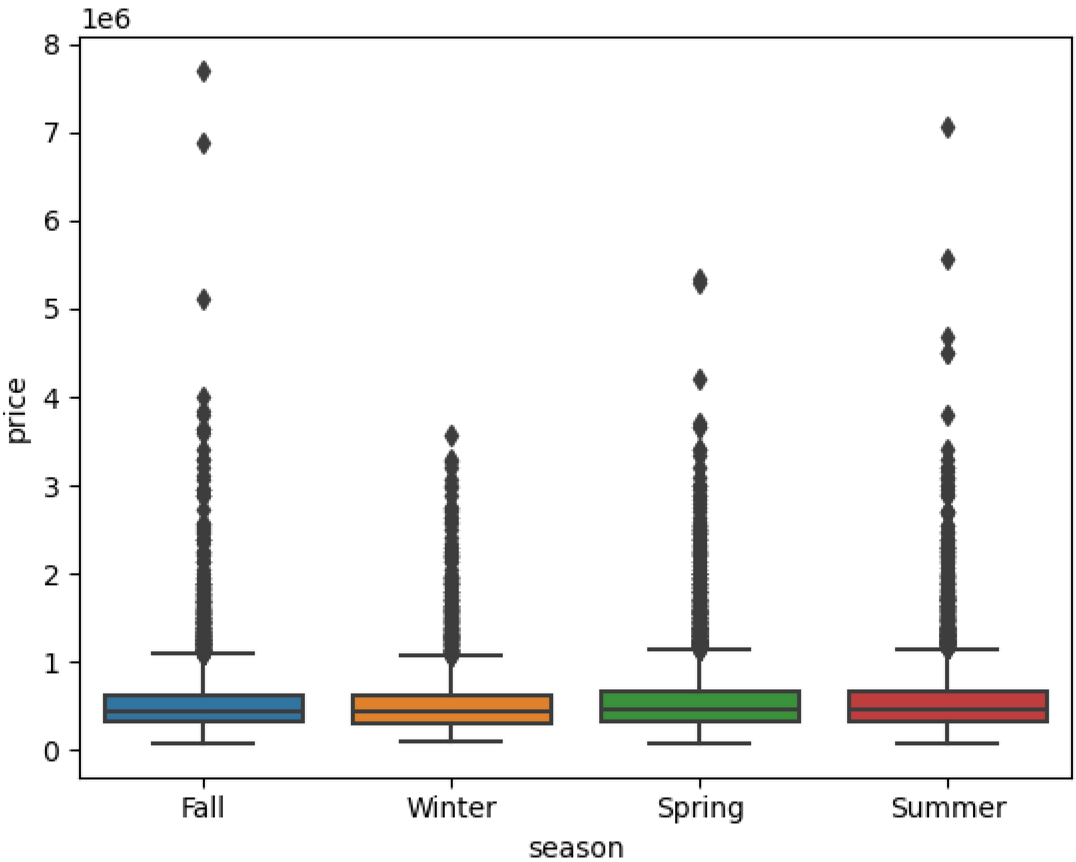


In [33]:
```
#removing outlier with '33' bedrooms
relevant_data=relevant_data[relevant_data['bedrooms']!=33]
```

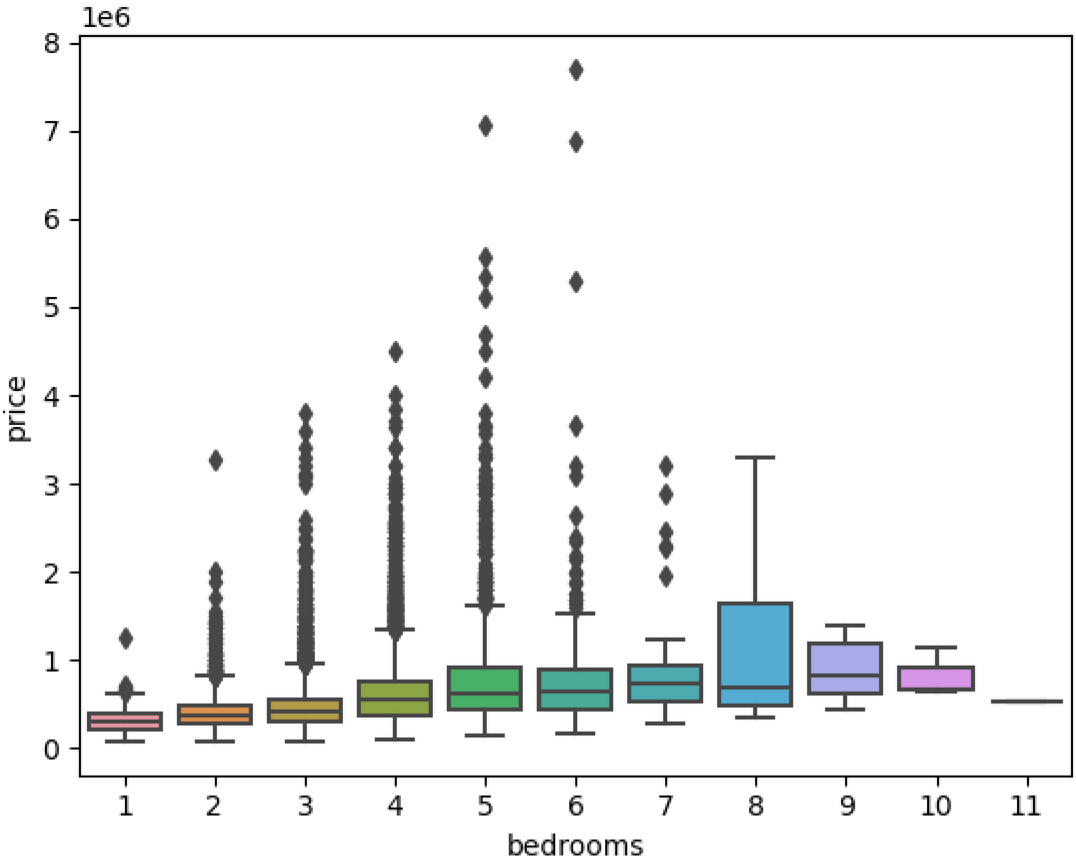
In [34]:
```
# Calculate Z-score for 'price'
z_scores = stats.zscore(relevant_data['price'])
threshold = 3  # Define a threshold for extreme values (e.g., 3 standard dev

# Detect outliers
outliers = relevant_data['price'][np.abs(z_scores) > threshold]

# Remove outliers
relevant_data1 = relevant_data[np.abs(z_scores) <= threshold]
```

In [35]:
```
# Pairplot
# sns.pairplot(relevant_data)
# plt.show()
```

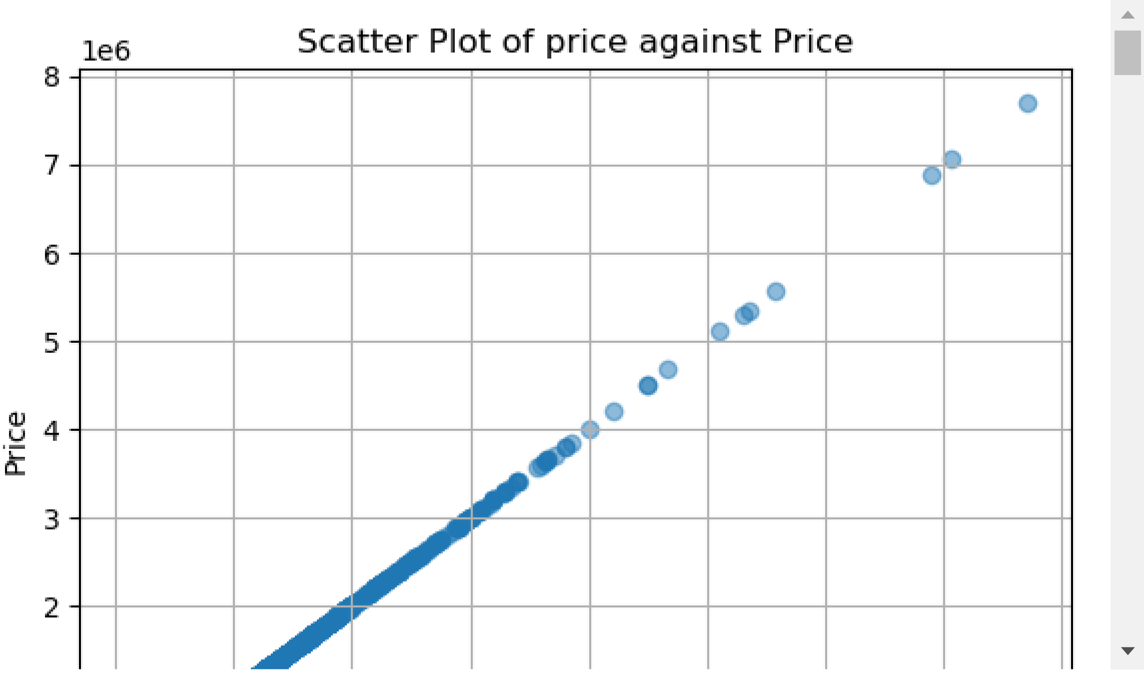
In [36]:
```
# seasons and price
sns.boxplot(x='season', y='price', data=relevant_data)
plt.show()
```



In [37]:
```
sns.boxplot(x='bedrooms', y='price', data=relevant_data)
plt.show()
```



In [38]:
```
#create a scatter plot for the numeric columns to observe their relationship
numeric_columns = relevant_data.select_dtypes(include=['number'])

# Iterate through each numeric column and create scatter plots
for column in numeric_columns.columns:
    plt.scatter(relevant_data[column], relevant_data['price'], alpha=0.5)
    plt.title(f'Scatter Plot of {column} against Price')
    plt.xlabel(column)
    plt.ylabel('Price')
    plt.grid(True)
    plt.show()
```



Correlation

In [39]:
```
relevant_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21596 entries, 0 to 21596
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   price        21596 non-null  float64
 1   bedrooms     21596 non-null  int64  
 2   bathrooms    21596 non-null  float64
 3   floors       21596 non-null  float64
 4   sqft_living  21596 non-null  int64  
 5   sqft_lot     21596 non-null  int64  
 6   waterfront   21596 non-null  int64  
 7   renovated    21596 non-null  int32  
 8   season       21596 non-null  object 
 9   grade_no     21596 non-null  int64  
 10  house_age    21596 non-null  int64  
dtypes: float64(3), int32(1), int64(6), object(1)
memory usage: 1.9+ MB
```

In [40]:
```
numeric_df = relevant_data.select_dtypes(include=['float64', 'int64'])
corr_price = numeric_df.corr()['price'].sort_values()
corr_price
```

Out[40]:
```
house_age     -0.053965
sqft_lot       0.089879
floors         0.256820
waterfront     0.264308
bedrooms       0.315961
bathrooms      0.525915
grade_no       0.667964
sqft_living    0.701929
price          1.000000
Name: price, dtype: float64
```

In [41]:
```
relevant_data.columns
```

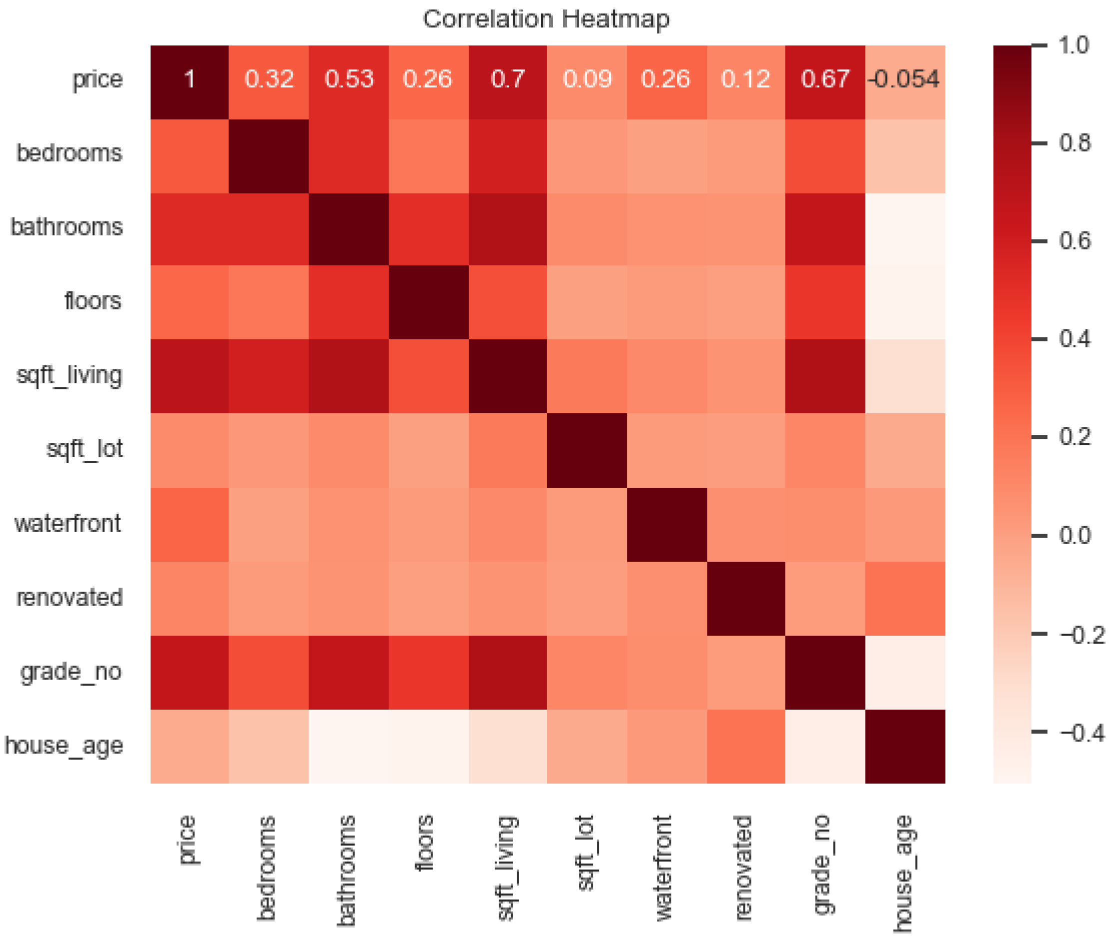
Out[41]:
```
Index(['price', 'bedrooms', 'bathrooms', 'floors', 'sqft_living', 'sqft_lo
t',
       'waterfront', 'renovated', 'season', 'grade_no', 'house_age'],
      dtype='object')
```

In [42]:
```
# Correlation table
cols = ['price', 'bedrooms', 'bathrooms', 'floors', 'sqft_living', 'sqft_lot
       'waterfront', 'renovated', 'grade_no', 'house_age']
corr = relevant_data[cols].corr()
corr
```

Out[42]:

	price	bedrooms	bathrooms	floors	sqft_living	sqft_lot	waterfront	rer
price	1.000000	0.315961	0.525915	0.256820	0.701929	0.089879	0.264308	0
bedrooms	0.315961	1.000000	0.527870	0.183707	0.593178	0.033602	-0.002054	0.
bathrooms	0.525915	0.527870	1.000000	0.502574	0.755755	0.088368	0.063628	0.
floors	0.256820	0.183707	0.502574	1.000000	0.353941	-0.004824	0.020794	0.
sqft_living	0.701929	0.593178	0.755755	0.353941	1.000000	0.173449	0.104635	0.
sqft_lot	0.089879	0.033602	0.088368	-0.004824	0.173449	1.000000	0.021458	0.
waterfront	0.264308	-0.002054	0.063628	0.020794	0.104635	0.021458	1.000000	0.
renovated	0.117546	0.018354	0.046738	0.003705	0.050825	0.005089	0.074267	1.
grade_no	0.667964	0.366174	0.665834	0.458783	0.762776	0.114726	0.082817	0.
house_age	-0.053965	-0.160736	-0.507166	-0.489175	-0.318140	-0.052939	0.024491	0.

In [43]:
```
# heatmap with seaborn
sns.set(font_scale=0.8)
sns.heatmap(corr, annot=True, cmap='Reds')
plt.title("Correlation Heatmap")
plt.show()
```



- The strongest positive correlation (0.7) is between price and square footage of living area, indicating that larger homes tend to have higher prices.
- The second strongest positive correlation (0.67) is between price and the grade_no, suggesting that the higher the grade the higher the price.
- The third strongest positive correlation (0.32) is between price and number of bedrooms, implying that more bedrooms in a home also contribute to its price, but to a lesser extent than bathrooms.
- The weakest positive correlation (0.05) is between price and year built, meaning that newer homes have slightly higher prices than older homes, but the effect is negligible.
- Price and Year Renovated is only (0.12). Reasons could be:
    - Renovations might cater to personal preferences rather than broad market appeal.
    - Recent renovations might indicate potential underlying issues with the property.

Modelling

In [44]:
```
import statsmodels.formula.api as sm
```

In [45]:
```
# To build a regression model we start with the baseline  model.
# This is a simple linear regression
# We start by assigning the dependent variable and the independent variable

y = relevant_data['price']
x_baseline = relevant_data[['sqft_living']]
```

In [46]:
```
# Build the simple linear regression model and display the results of the mo
import statsmodels.api as sm

baseline_model = sm.OLS(y, sm.add_constant(x_baseline))
baseline_results = baseline_model.fit()
print(baseline_results.summary())
```

```
                            OLS Regression Results
==========================================================================
====
Dep. Variable:                  price   R-squared:
0.493
Model:                            OLS   Adj. R-squared:
0.493
Method:                 Least Squares   F-statistic:                 2.097
e+04
Date:                Wed, 03 Jan 2024   Prob (F-statistic):
0.00
Time:                        21:44:56   Log-Likelihood:            -3.0005
e+05
No. Observations:               21596   AIC:                         6.001
e+05
Df Residuals:                   21594   BIC:                         6.001
e+05
Df Model:                           1
Covariance Type:            nonrobust
==========================================================================
=====
                  coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------
-----
const       -4.401e+04   4410.123     -9.980      0.000   -5.27e+04    -3.5
4e+04
sqft_living   280.8688      1.939    144.820      0.000     277.067      28
4.670
==========================================================================
====
Omnibus:                    14801.492   Durbin-Watson:
1.982
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           54264
2.481
Skew:                           2.820   Prob(JB):
0.00
Kurtosis:                      26.901   Cond. No.                     5.63
e+03
==========================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
[2] The condition number is large, 5.63e+03. This might indicate that ther
e are
strong multicollinearity or other numerical problems.
```

In [47]:
```
# Plot the scatter diagram and the best_fit regression line.

fig, ax = plt.subplots()
relevant_data.plot.scatter(x = 'sqft_living', y = 'price', label = 'Data poi
sm.graphics.abline_plot(model_results = baseline_results, label = 'regressic
ax.legend();
```

Intepretation of the Baseline model

- The R squared of the model is 0.49 interpreted as the propotional extent to which independent variable, Squarefoot of living area, explains the change in price.
- The probability of the F Statistic is well below the alpha value of 0.05 which shows that model is stastically significant.
- The coefficients of the model (const (which is the y intercept) and the sqft_living) are both statistically significant with a p value well below 0.05.
- For each increase in sqft_living we see a corresponding increase of 281 in price.

Building a Multiple regression Model

In [48]:
```
relevant_data.columns
```

Out[48]:
```
Index(['price', 'bedrooms', 'bathrooms', 'floors', 'sqft_living', 'sqft_lo
t',
       'waterfront', 'renovated', 'season', 'grade_no', 'house_age'],
      dtype='object')
```

In [49]:
```
y = relevant_data['price']
x_baseline = relevant_data[['grade_no','sqft_living','bathrooms','renovated'
```

In [50]:
```
# Build the simple linear regression model and display the results of the mo
import statsmodels.api as sm


baseline_model = sm.OLS(y, sm.add_constant(x_baseline))
baseline_results = baseline_model.fit()
print(baseline_results.summary())
```

```
                                 OLS Regression Results
==========================================================================
====
Dep. Variable:                  price   R-squared:
0.607
Model:                            OLS   Adj. R-squared:
0.607
Method:                 Least Squares   F-statistic:                      6
672.
Date:                Wed, 03 Jan 2024   Prob (F-statistic):
0.00
Time:                        21:44:58   Log-Likelihood:             -2.9729
e+05
No. Observations:               21596   AIC:                          5.946
e+05
Df Residuals:                   21590   BIC:                          5.946
e+05
Df Model:                           5
Covariance Type:            nonrobust
==========================================================================
=====
                  coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------
-----
const       -1.166e+06   1.55e+04    -75.151      0.000    -1.2e+06   -1.1
4e+06
grade_no     1.417e+05   2200.325     64.409      0.000    1.37e+05    1.4
6e+05
sqft_living   155.1451      3.176     48.843      0.000     148.919     16
1.371
bathrooms    4.247e+04   3484.316     12.188      0.000    3.56e+04    4.9
3e+04
renovated    4.715e+04   8921.922      5.285      0.000    2.97e+04    6.4
6e+04
house_age    3907.1281     67.185     58.155      0.000    3775.440    403
8.816
==========================================================================
====
Omnibus:                    17739.221   Durbin-Watson:
1.977
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          133922
6.121
Skew:                           3.466   Prob(JB):
0.00
Kurtosis:                      40.951   Cond. No.                      2.28
e+04
==========================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
[2] The condition number is large, 2.28e+04. This might indicate that ther
e are
strong multicollinearity or other numerical problems.
```

In [51]:
```
y = relevant_data['price']
x_baseline = relevant_data[['bedrooms', 'bathrooms', 'floors', 'sqft_living'
        'waterfront', 'renovated','grade_no', 'house_age']]
```

In [52]:
```
# Build the simple linear regression model and display the results of the mo
baseline_model = sm.OLS(y, sm.add_constant(x_baseline))
baseline_results = baseline_model.fit()
print(baseline_results.summary())
```

```
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                  price   R-squared:
0.646
Model:                            OLS   Adj. R-squared:
0.645
Method:                 Least Squares   F-statistic:                     4
368.
Date:                Wed, 03 Jan 2024   Prob (F-statistic):
0.00
Time:                        21:44:58   Log-Likelihood:            -2.9618
e+05
No. Observations:               21596   AIC:                         5.924
e+05
Df Residuals:                   21586   BIC:                         5.925
e+05
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
=====
                  coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const       -1.017e+06   1.61e+04    -63.270      0.000   -1.05e+06   -9.8
5e+05
bedrooms    -4.573e+04   2134.899    -21.418      0.000   -4.99e+04   -4.1
5e+04
bathrooms    5.282e+04   3477.102     15.192      0.000     4.6e+04    5.9
6e+04
floors       1.692e+04   3436.267      4.923      0.000    1.02e+04    2.3
7e+04
sqft_living   179.8298      3.324     54.106      0.000     173.315     18
6.344
sqft_lot       -0.2495      0.037     -6.781      0.000      -0.322      -
0.177
waterfront   7.512e+05   1.84e+04     40.838      0.000    7.15e+05    7.8
7e+05
renovated    1.759e+04   8509.590      2.067      0.039     907.193    3.4
3e+04
grade_no     1.291e+05   2159.160     59.791      0.000    1.25e+05    1.3
3e+05
house_age    3957.4099     66.448     59.557      0.000    3827.168    408
7.652
==============================================================================
====
Omnibus:                    15650.960   Durbin-Watson:
1.976
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           96655
6.213
Skew:                           2.888   Prob(JB):
0.00
Kurtosis:                      35.261   Cond. No.                     5.45
e+05
==============================================================================
====

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

```
[2] The condition number is large, 5.45e+05. This might indicate that ther
e are
strong multicollinearity or other numerical problems.
```

Interpretation of results

- R-squared: 0.646 Indicates that the model explains 64.6 % of the variance in the dependent variable (price).

- Adj. R-squared: 0.645 Similar to R-squared but adjusts for the number of predictors in the model.

- Prob (F-statistic): 0.00 The probability associated with the F-statistic, suggesting the overall significance of the model.

- Coefficients: Each predictor variable's coefficient represents the change in the dependent variable (price) per unit change in that predictor, holding other predictors constant.
  - Intercept (const): -1.017e+06
  - bedrooms: -4.573e+04
  - bathrooms: 5.282e+04
  - floors: 1.692e+04
  - sqft_living: 179.8298
  - sqft_lot: -0.2495
  - waterfront: 7.512e+05
  - renovated: 1.759e+04
  - grade_no: 1.291e+05
  - house_age: 3957.4099

- Statistical Significance (P-values): P>|t| values for each coefficient indicate the probability of observing the data if the null hypothesis (that the coefficient is zero) is true.

- Variables with a P-value less than a significance level (commonly 0.05) are typically considered statistically significant. For instance, bathrooms, floors, sqft_living, sqft_lot, waterfront, renovated, grade_no, house_age,seem significant since their P-values are 0.000.

Recomendations

Our potential homeowners are advised to focus on the variables on the final model when looking for competitively priced homes. These are majorly square feet of the living space, bathrooms and bedrooms. Where one is looking to buy a home at favorable prices a potential homeowner will need to compromise on one or two items. e.g. waterfront homes or square feet of living area.Our stakeholders are advised to purchase homes in the spring or summer in order to get a good variety of homes to pick from. Further Data Collection: Explore additional data beyond the current predictors to refine the model. Factors like proximity to amenities, neighborhood trends, or specific architectural styles could further enhance price prediction accuracy.