Data Science Part time 06 Phase 2, Group 12

Students names:

- Felix Mburu Njoroge
- Winnie Osolo
- Violet Musyoka
- Alex Mwaura
- Lydia M

Overview

A commercial real estate agency is interested in investing in the real estate within King County. Their main concern is having and overview of prices of houses within the area and identify some of the attributes that influence the pricing strategy.

Business Understanding and Problem

A real estate agency wants to come up with a pricing strategy. They want to do this by analyzing the factors that most significantly impact house prices.

In order to achieve this, we are required to come up with a model that is capable of deducing the main factors that influence the house prices.The client seeks to utilize accurate and representative data pertaining to thne real estate market in King County. This data contains historical sales, size of the property and other pertinet features.

Objectives

To support the real estate agency in developing a pricing strategy through regression analysis, focusing on the factors that significantly impact house prices, the following three objectives or questions can be addressed using the dataset:

1. **What are the key predictors of house prices?**
   - Objective: Identify and quantify the impact of various features (like square footage, number of bedrooms and bathrooms, location, etc.) on the house prices. This involves using regression analysis to determine which variables most strongly correlate with the price of a house.
   - Key Analysis: Conduct multiple linear regression to assess the relationship between house prices and potential predictor variables. Evaluate the coefficients to determine the impact of each feature on the price.
2. **How does location affect house prices within the region?**
   - Objective: Analyze the geographical distribution of house prices to understand how different locations (e.g., zip codes, proximity to waterfronts) influence the valuation of properties. This will help in understanding location premiums or penalties.
3. **Can we predict the price of a house based on its attributes?**

- Objective: Develop a predictive model that estimates the price of a house based on its features, such as size, age, condition, and neighborhood characteristics. The goal is to create a reliable pricing tool that the agency can use for setting competitive prices.
- Key Analysis: Implement a regression model (like multiple linear regression, polynomial regression to predict house prices. Evaluate the model's accuracy and adjust it to improve prediction performance.

These objectives will guide the analysis of the data and the development of a regression-based pricing model, helping the real estate agency to set competitive and market-aligned prices for their listings.

Data Understanding

For the project, we will use the King County House Sales dataset, which contains information about house sales in northwestern county.

Data Preparation

In our data preparation, we imported the necessary libraries, loaded the data, data cleaning which involved dealing with null values, outliers and duplicates then encoding the data.

```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import scipy.stats as stats
%matplotlib inline
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import r2_score, mean_squared_error
from datetime import datetime
pd.options.display.float_format = '{:.2f}'.format

#data loading, loading it as a pandas data frame

df = pd.read_csv(r"C:\Users\Kola500\Downloads\acaa2waaaaw2w2aaa2
x_files\kc_house_data.csv")
```

Previewing the Data

```
#inspect
df.head()

           id       date     price  bedrooms  bathrooms  sqft_living
\
```

```
0  7129300520  10/13/2014 221900.00         3       1.00         1180

1  6414100192   12/9/2014 538000.00         3       2.25         2570

2  5631500400   2/25/2015 180000.00         2       1.00          770

3  2487200875   12/9/2014 604000.00         4       3.00         1960

4  1954400510   2/18/2015 510000.00         3       2.00         1680


   sqft_lot  floors waterfront  view  ...          grade sqft_above  \
0      5650    1.00        NaN  NONE  ...      7 Average       1180
1      7242    2.00         NO  NONE  ...      7 Average       2170
2     10000    1.00         NO  NONE  ...  6 Low Average        770
3      5000    1.00         NO  NONE  ...      7 Average       1050
4      8080    1.00         NO  NONE  ...         8 Good       1680

   sqft_basement yr_built  yr_renovated  zipcode   lat    long
sqft_living15  \
0            0.0     1955          0.00    98178 47.51 -122.26
1340
1          400.0     1951       1991.00    98125 47.72 -122.32
1690
2            0.0     1933           NaN    98028 47.74 -122.23
2720
3          910.0     1965          0.00    98136 47.52 -122.39
1360
4            0.0     1987          0.00    98074 47.62 -122.05
1800

   sqft_lot15
0        5650
1        7639
2        8062
3        5000
4        7503

[5 rows x 21 columns]

df.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
```

```
#check decriptive stats
df.describe()
```

```
                 id      price  bedrooms  bathrooms  sqft_living
sqft_lot  \
count      21597.00   21597.00  21597.00   21597.00     21597.00
21597.00
mean  4580474287.77  540296.57      3.37       2.12      2080.32
15099.41
std   2876735715.75  367368.14      0.93       0.77       918.11
41412.64
min      1000102.00   78000.00      1.00       0.50       370.00
520.00
25%   2123049175.00  322000.00      3.00       1.75      1430.00
5040.00
50%   3904930410.00  450000.00      3.00       2.25      1910.00
7618.00
75%   7308900490.00  645000.00      4.00       2.50      2550.00
10685.00
max   9900000190.00 7700000.00     33.00       8.00     13540.00
1651359.00

        floors  sqft_above  yr_built  yr_renovated  zipcode      lat
long  \
count 21597.00    21597.00  21597.00      17755.00 21597.00 21597.00
21597.00
mean      1.49     1788.60   1971.00         83.64 98077.95    47.56
-122.21
std       0.54      827.76     29.38        399.95    53.51     0.14
0.14
```

```
min        1.00       370.00    1900.00          0.00 98001.00     47.16
-122.52
25%        1.00      1190.00    1951.00          0.00 98033.00     47.47
-122.33
50%        1.50      1560.00    1975.00          0.00 98065.00     47.57
-122.23
75%        2.00      2210.00    1997.00          0.00 98118.00     47.68
-122.12
max        3.50      9410.00    2015.00       2015.00 98199.00     47.78
-121.31

       sqft_living15  sqft_lot15
count       21597.00    21597.00
mean         1986.62    12758.28
std           685.23    27274.44
min           399.00      651.00
25%          1490.00     5100.00
50%          1840.00     7620.00
75%          2360.00    10083.00
max          6210.00   871200.00
```

```
#dropping columns, justfication:
# id and date columns are not useful for analyisis, there are no
insights to get from them
# sqft_basement has too many 0s on inspection
# yr_renovated has too many missing values
df = df.drop(columns=['id', 'date', 'yr_renovated', 'sqft_basement'])
```

Data Cleaning

1. Checking for Missing Values

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

```
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
yr_built            0
zipcode             0
lat                 0
long                0
```

```
sqft_living15       0
sqft_lot15          0
dtype: int64
```

```
#check percentage of missing values
# create a function to check the percentage of missing values
def missing_values(data):
    miss = data.isnull().sum().sort_values(ascending = False)
    percentage_miss = (data.isnull().sum() /
len(data)).sort_values(ascending = False)
    missing = pd.DataFrame({"Missing Values": miss, "Percentage":
percentage_miss}).reset_index()
    missing.drop(missing[missing["Percentage"] == 0].index, inplace =
True)
    return missing

missing_data = missing_values(df)
missing_data
```

	index	Missing Values	Percentage
0	waterfront	2376	0.11
1	view	63	0.00

The nature of the data in waterfront and view means that it would not make much sense to impute because they are categorical variables The best option is to drop the entries with missing values

```
#dropping rows with missing values (waterfront, view)
df = df.dropna()

#checking again for missing values
df.isnull().sum()
```

```
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
yr_built         0
zipcode          0
lat              0
long             0
sqft_living15    0
```

```
sqft_lot15       0
dtype: int64
```

```
df.info()
```

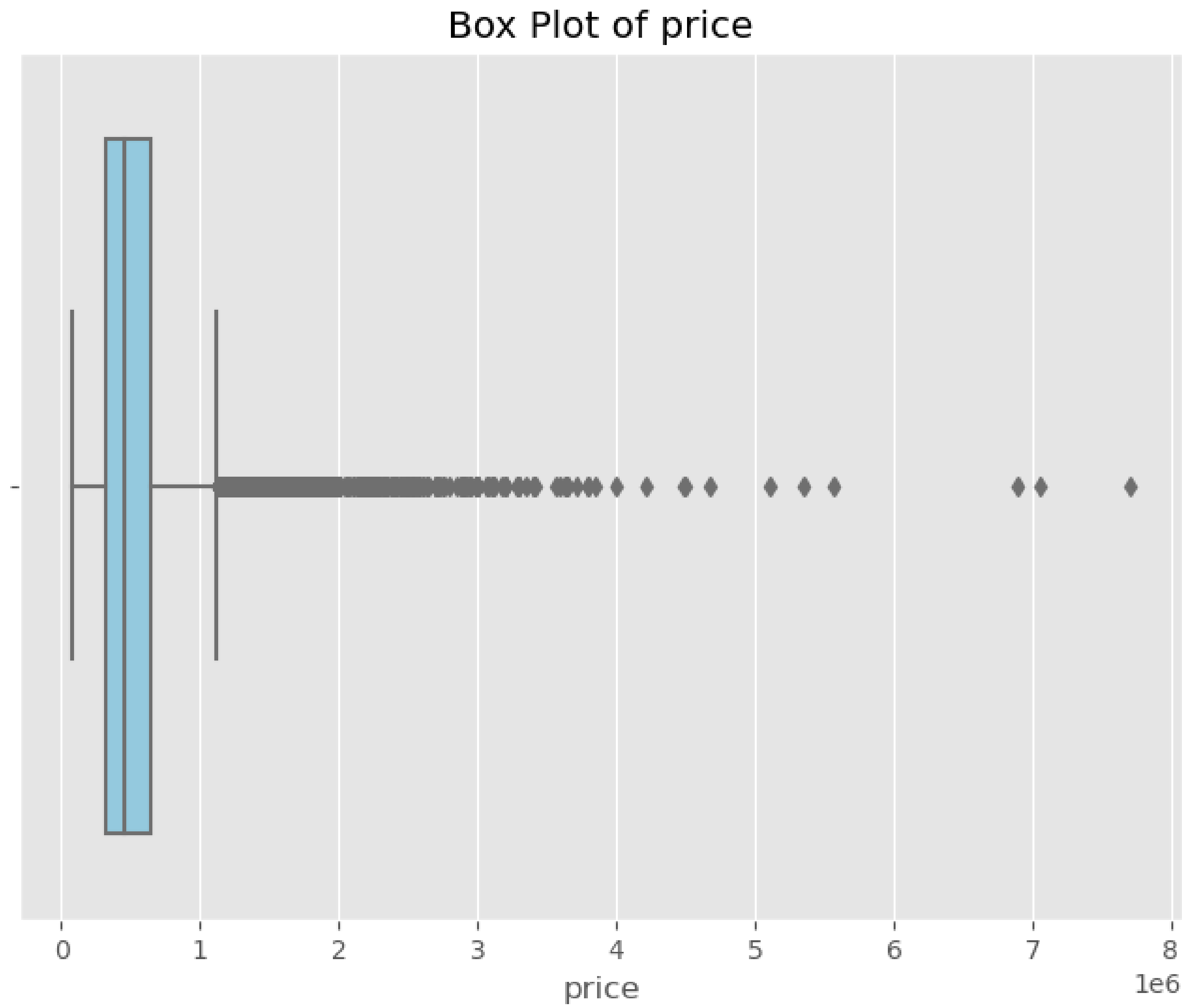
```
<class 'pandas.core.frame.DataFrame'>
Index: 19164 entries, 1 to 21596
Data columns (total 17 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   price          19164 non-null  float64
 1   bedrooms       19164 non-null  int64
 2   bathrooms      19164 non-null  float64
 3   sqft_living    19164 non-null  int64
 4   sqft_lot       19164 non-null  int64
 5   floors         19164 non-null  float64
 6   waterfront     19164 non-null  object
 7   view           19164 non-null  object
 8   condition      19164 non-null  object
 9   grade          19164 non-null  object
 10  sqft_above     19164 non-null  int64
 11  yr_built       19164 non-null  int64
 12  zipcode        19164 non-null  int64
 13  lat            19164 non-null  float64
 14  long           19164 non-null  float64
 15  sqft_living15  19164 non-null  int64
 16  sqft_lot15     19164 non-null  int64
dtypes: float64(5), int64(8), object(4)
memory usage: 2.6+ MB
```

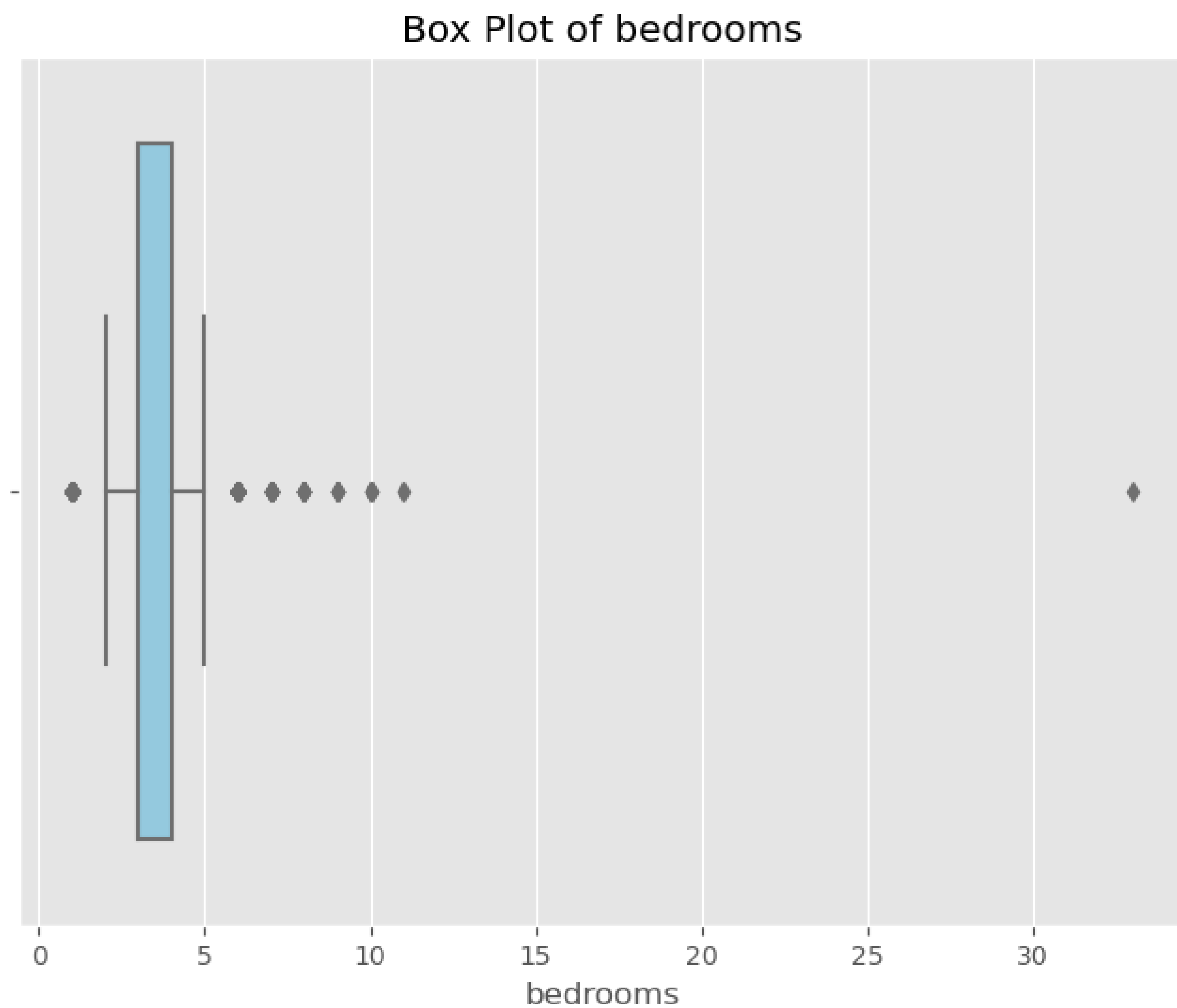
2. Checking for Outliers

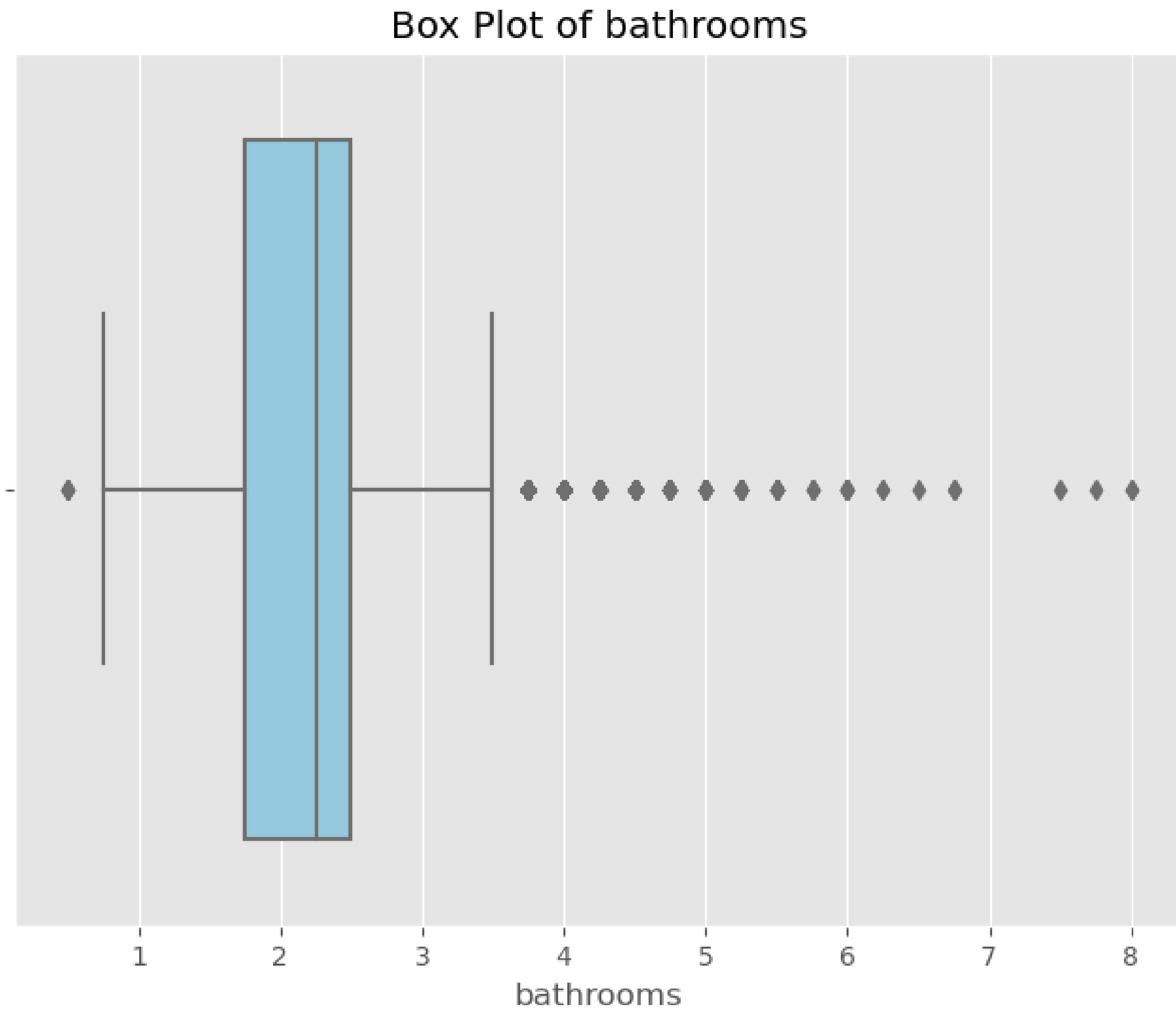
We'll check outliers in the following columns: price, bedrooms, bathrooms, sqft_living, sqft_lot, floors,sqft_above, sqft_basement bacause they are numerical, since they are numerical variables

```
# the columns to check for outliers are price, bedrooms, bathrooms, sqft_living, sqft_lot, floors,sqft_above, sqft_basement
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['price'], color='skyblue')
plt.title('Box Plot of price')
```

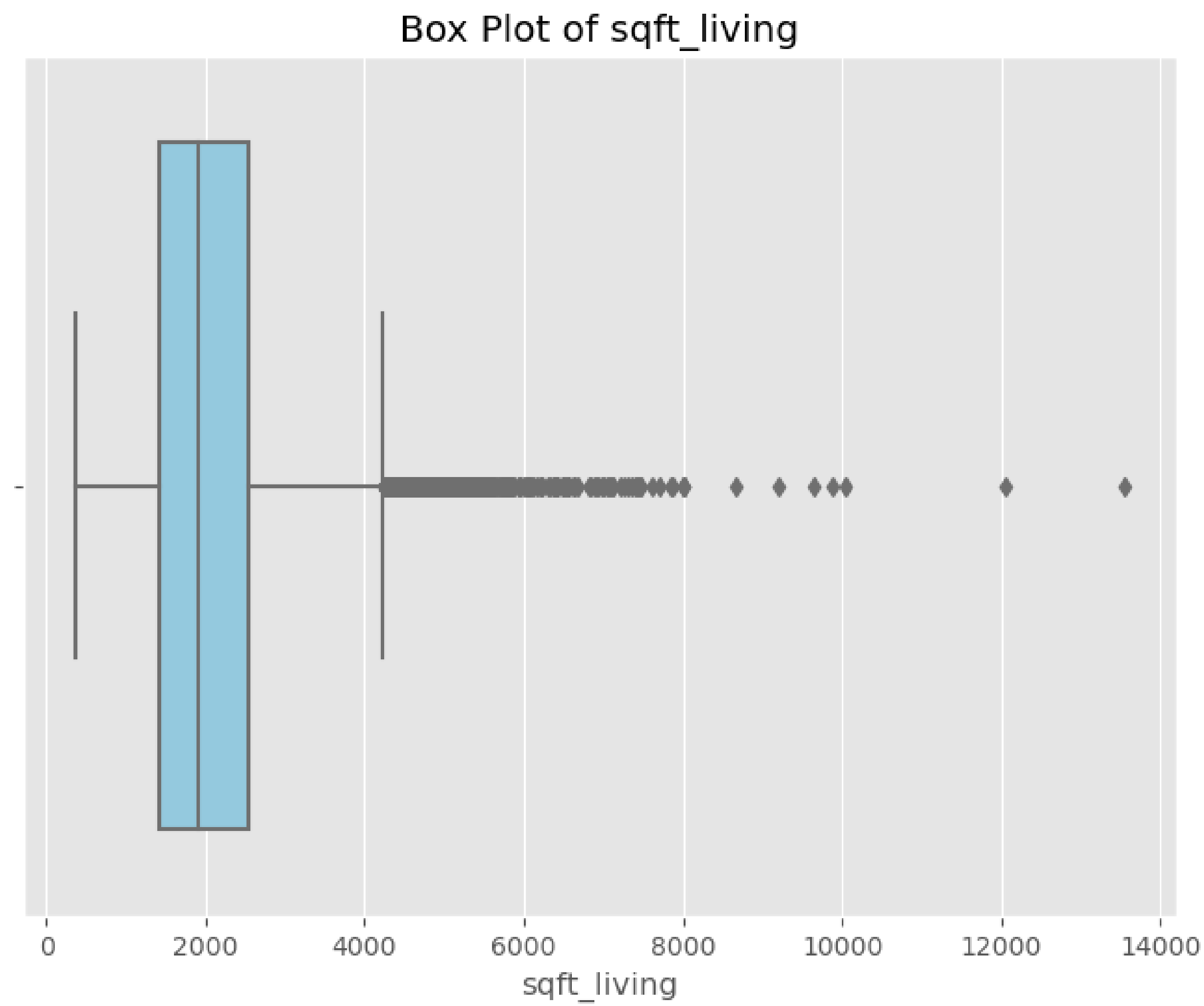
```
plt.show()

plt.figure(figsize=(8, 6))
sns.boxplot(x=df['bedrooms'], color='skyblue')
plt.title('Box Plot of bedrooms')
plt.show()

plt.figure(figsize=(8, 6))
sns.boxplot(x=df['bathrooms'], color='skyblue')
plt.title('Box Plot of bathrooms')
plt.show()

plt.figure(figsize=(8, 6))
sns.boxplot(x=df['sqft_living'], color='skyblue')
plt.title('Box Plot of sqft_living')
plt.show()

plt.figure(figsize=(8, 6))
sns.boxplot(x=df['sqft_lot'], color='skyblue')
plt.title('Box Plot of sqft_lot')
plt.show()

plt.figure(figsize=(8, 6))
sns.boxplot(x=df['floors'], color='skyblue')
plt.title('Box Plot of floors')
plt.show()
```

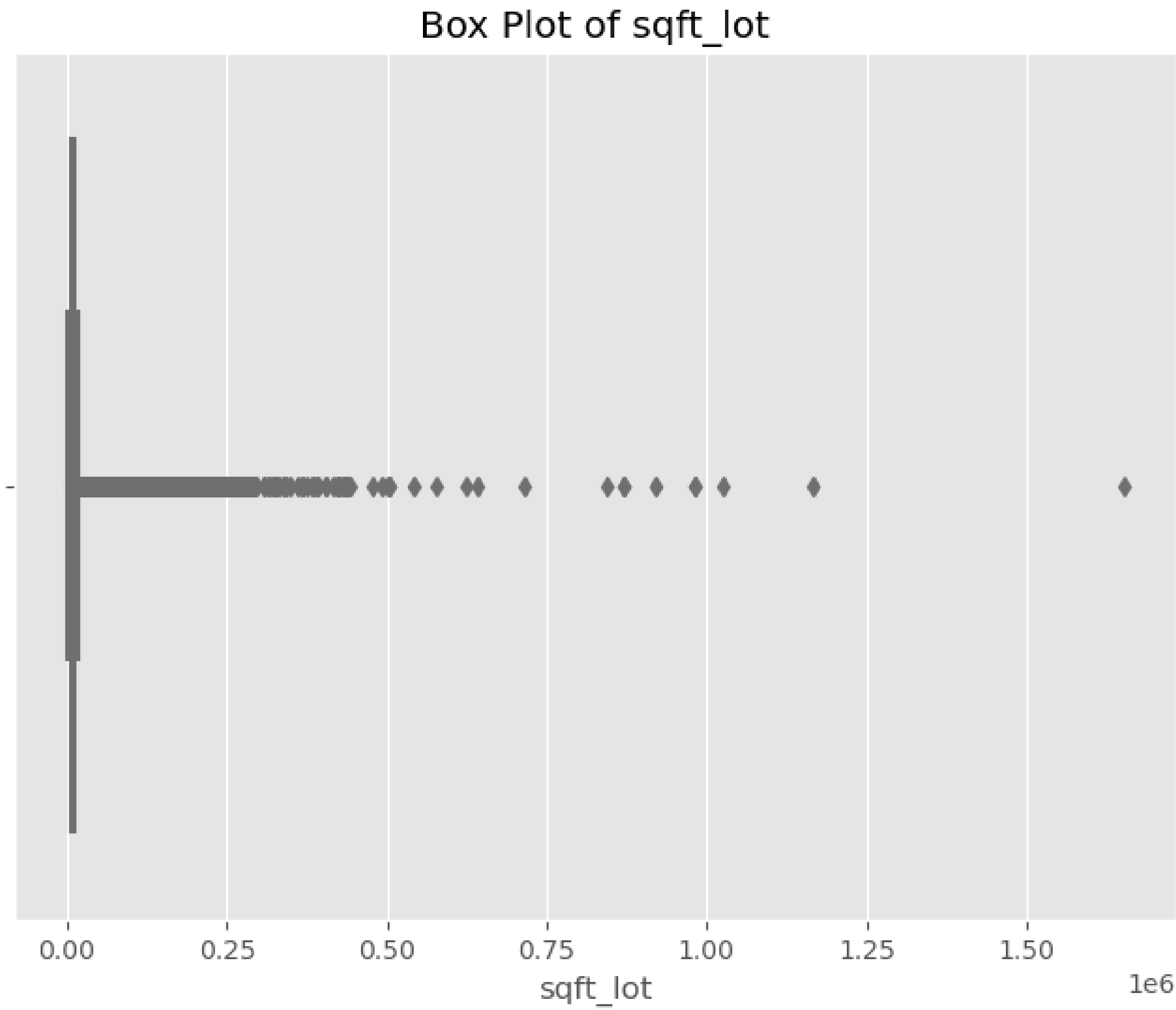

Box Plot of price
0
1
2
3
4
5
6
7
8
1e6
price

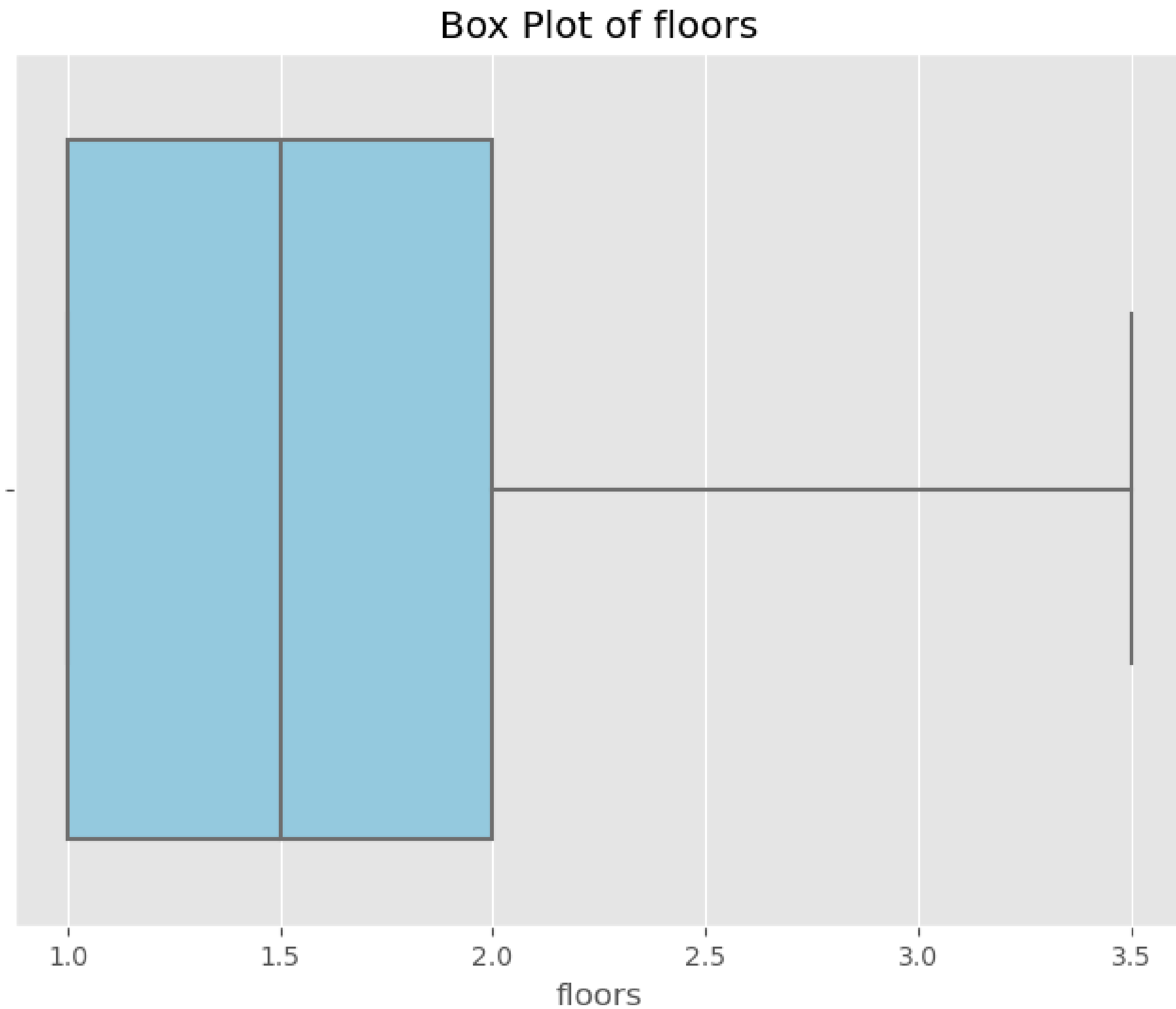

Box Plot of bedrooms
0
5
10
15
20
25
30
bedrooms



Box Plot of bathrooms
1
2
3
4
5
6
7
8
bathrooms

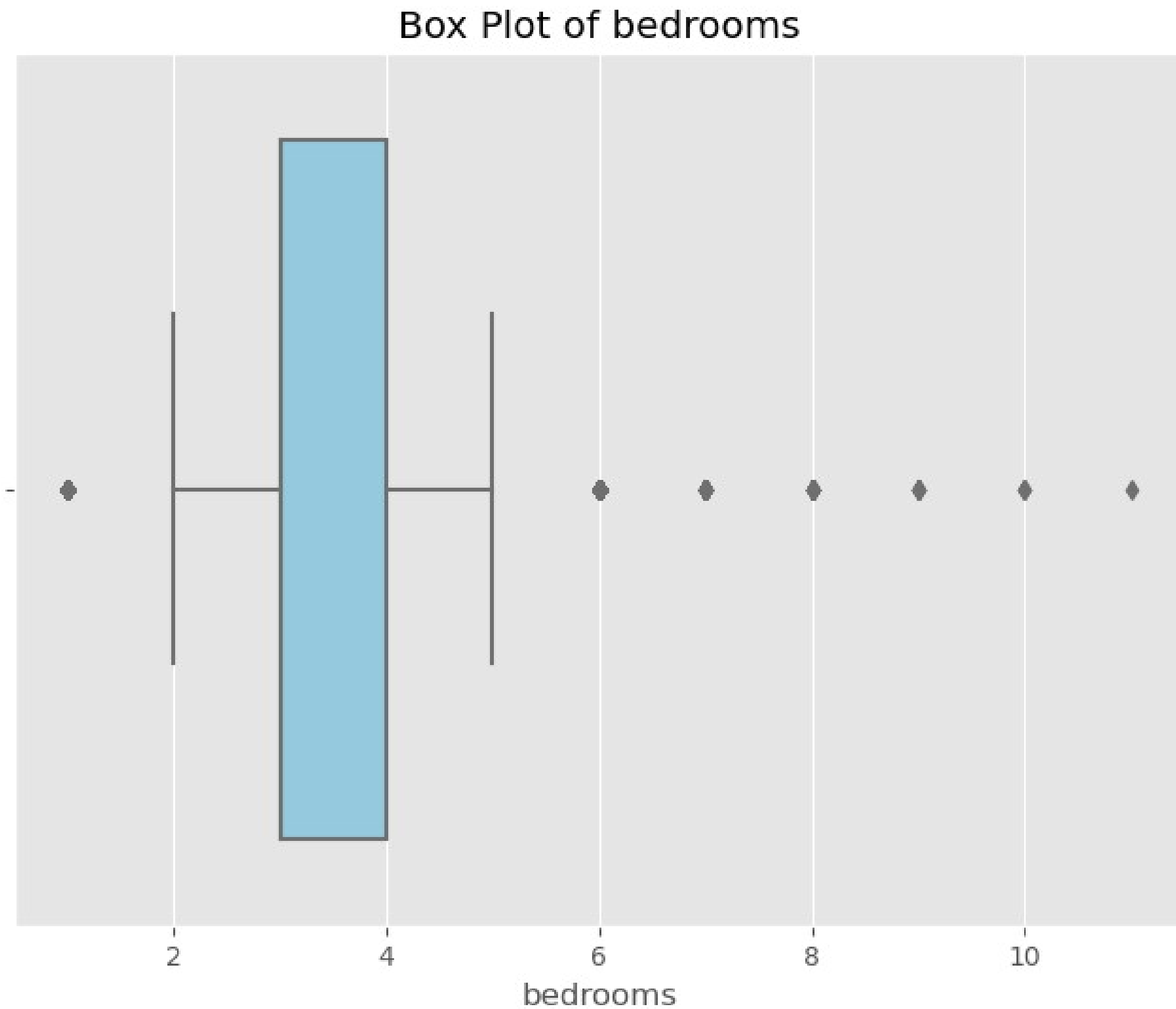

Box Plot of sqft_living
0
2000
4000
6000
8000
10000
12000
14000
sqft_living


Box Plot of sqft_lot
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1e6
sqft_lot



The outlier that appear unusual is 30 bedrooms. This is quite extreme. Although the other columns have outliers, it doesn't mean they have to be dropped, they are well within the realm of possibility

```
#dropping the outlier with more than 30 bedrooms since it's extreme
df = df[df['bedrooms'] <= 30]

#bedrooms after removing outliers
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['bedrooms'], color='skyblue')
plt.title('Box Plot of bedrooms')
plt.show()
```



Although the other columns have outliers, it doesn't mean they have to be dropped

3. Checking for Duplicate Rows

```
df.duplicated()
```

```
1        False
2        False
3        False
4        False
5        False
         ...
21591    False
21592    False
21593    False
21594    False
21596    False
Length: 19163, dtype: bool
```

There are no duplicates

4. Ensuring Correct Data Type

```
df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 19163 entries, 1 to 21596
Data columns (total 17 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          19163 non-null  float64
 1   bedrooms       19163 non-null  int64  
 2   bathrooms      19163 non-null  float64
 3   sqft_living    19163 non-null  int64  
 4   sqft_lot       19163 non-null  int64  
 5   floors         19163 non-null  float64
 6   waterfront     19163 non-null  object 
 7   view           19163 non-null  object 
 8   condition      19163 non-null  object 
 9   grade          19163 non-null  object 
 10  sqft_above     19163 non-null  int64  
 11  yr_built       19163 non-null  int64  
 12  zipcode        19163 non-null  int64  
 13  lat            19163 non-null  float64
 14  long           19163 non-null  float64
 15  sqft_living15  19163 non-null  int64  
 16  sqft_lot15     19163 non-null  int64  
dtypes: float64(5), int64(8), object(4)
memory usage: 2.6+ MB
```

Floors and bathrooms are named as float data type. This is incorrect and needs to be addressed since they can only be integers.

```
df['floors'] = df['floors'].astype(int)
df['bathrooms'] = df['bathrooms'].astype(int)
```

Exploratory Data Analysis (EDA)

In this section we will perforn EDA to understand the data better and discover any patterns, trends that may exist using univariate, bivariate and multivariate analysis. We will use desriptive statistics and visualizations to summarise the main characteristics and examine relationships between the features and our target variable(price).

- Measures of Central Tendancy and Dispersion
- Univariate
- Bivariate - correlation
- Feature engineering
- Categorical Data Encoding- one hot encoding, and label Encoder

Analysis of the Numerical and Descriptive data

```
num_attributes = df.select_dtypes(include=['int64','float64'])
cat_attributes = df.select_dtypes(exclude=['int64','float64'])

num_attributes.sample()
```

```
          price  bedrooms  sqft_living  sqft_lot  sqft_above  yr_built  \
11149 422250.00         3         1650      7145        1300      1977   

       zipcode   lat    long  sqft_living15  sqft_lot15  
11149    98155 47.77 -122.32           1760        7206  
```

```
cat_attributes.sample()
```

```
    bathrooms  floors waterfront  view condition      grade
149         1       1         NO  NONE   Average  7 Average
```

```
df.describe()
```

```
           price  bedrooms  bathrooms  sqft_living   sqft_lot   floors  \
count   19163.00  19163.00   19163.00     19163.00   19163.00 19163.00   
mean   541443.82      3.37       1.75      2082.06   15062.21     1.45   
std    370909.93      0.90       0.73       921.94   40773.16     0.55   
min     78000.00      1.00       0.00       370.00     520.00     1.00   
25%    322000.00      3.00       1.00      1430.00    5040.00     1.00   
50%    450000.00      3.00       2.00      1920.00    7620.00     1.00   
75%    643975.00      4.00       2.00      2550.00   10720.00     2.00   
max   7700000.00     11.00       8.00     13540.00 1651359.00     3.00   

       sqft_above  yr_built  zipcode      lat     long  sqft_living15  \
count    19163.00  19163.00 19163.00 19163.00 19163.00       19163.00   
mean      1791.46   1971.04 98077.73    47.56  -122.21        1987.26   
std        831.78     29.39    53.45     0.14     0.14         684.79   
min        370.00   1900.00 98001.00    47.16  -122.52         399.00   
25%       1200.00   1951.00 98033.00    47.47  -122.33        1490.00   
```

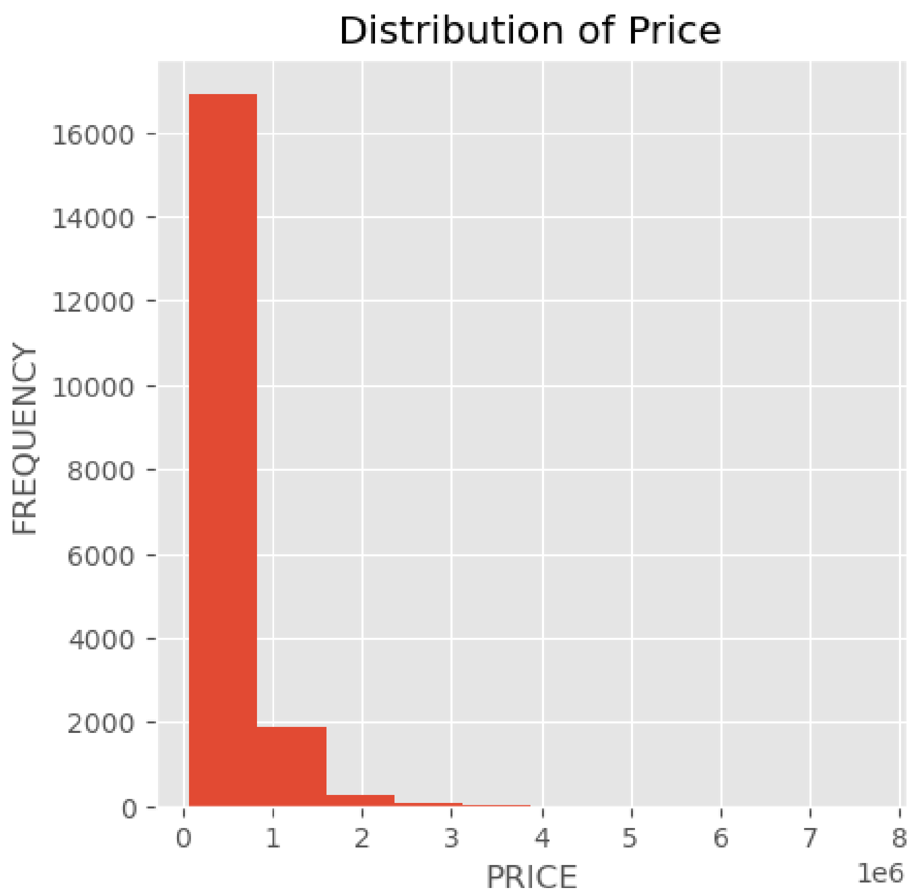
```
50%         1560.00    1975.00 98065.00     47.57   -122.23          1840.00

75%         2218.50    1997.00 98117.00     47.68   -122.12          2360.00

max         9410.00    2015.00 98199.00     47.78   -121.31          6210.00


       sqft_lot15
count    19163.00
mean     12798.08
std      27553.29
min        651.00
25%       5100.00
50%       7620.00
75%      10093.50
max     871200.00
```

1.Univariate Analysis

- Summary ststistic of our main variable of interest - Price

```
# Extract the price Column
price = df['price']

# Price statistics
price_stats = price.describe()
print(price_stats)

# Histogram of the price statistics
plt.style.use('ggplot')
price.hist(figsize=(5,5))
plt.xlabel ('PRICE')
plt.ylabel('FREQUENCY')
plt.title ('Distribution of Price')
```

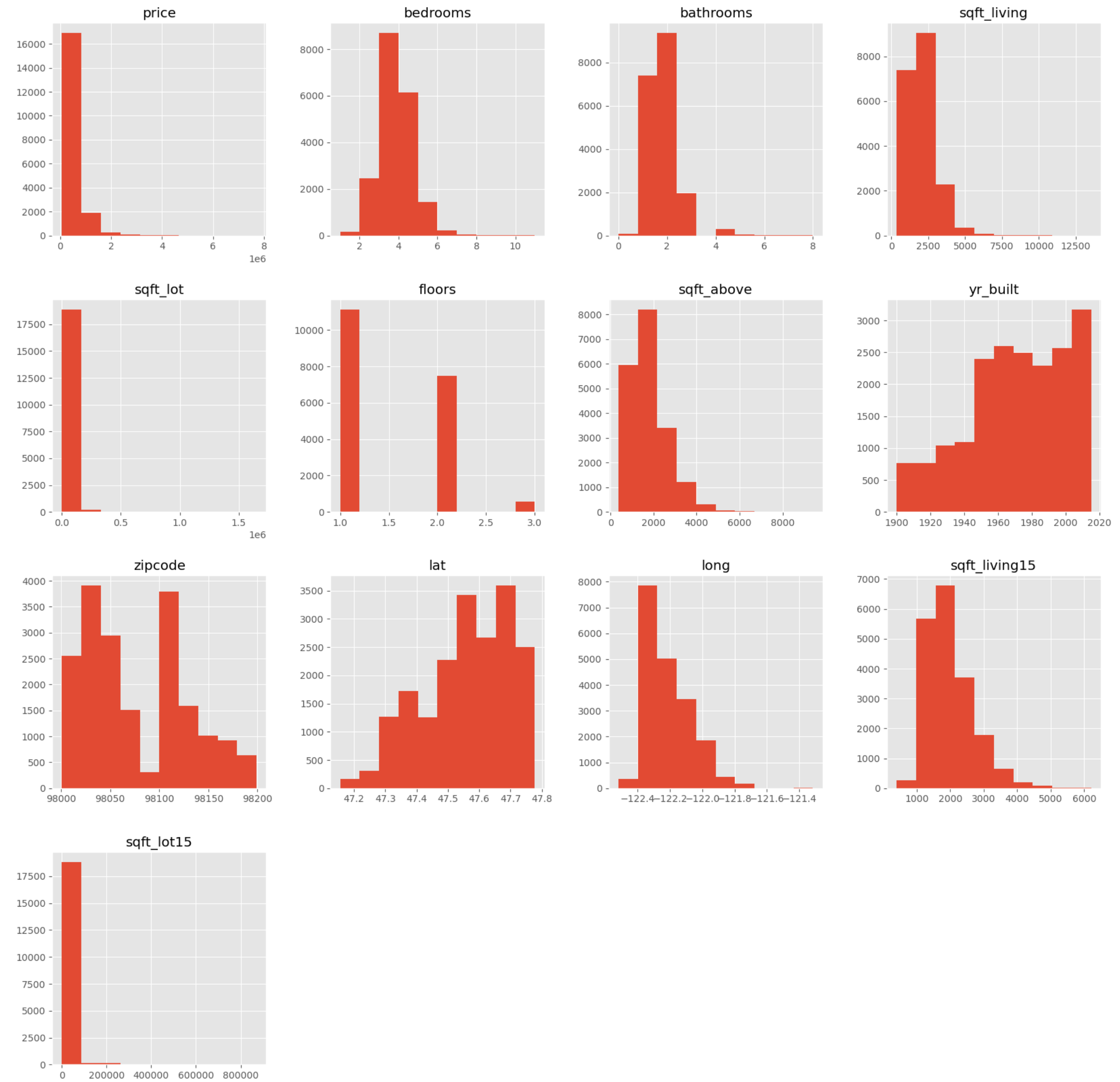
```
count      19163.00
mean      541443.82
std       370909.93
min        78000.00
25%       322000.00
50%       450000.00
75%       643975.00
max      7700000.00
Name: price, dtype: float64

Text(0.5, 1.0, 'Distribution of Price')
```



- The histogram shows that the distribution of the house prices in trhe dataset is right skewed, which means thet there are more houses with lower prices.
- The summary statistics provides insights on the mean price whish is at USD 541443 and the median price at USD 450000. The standard devistion is USD 370909.The high standard deviation implies that there are significant differences in housing prices across the dataset. Some houses may be priced much lower than the average, while others may be priced much higher. This variability can be influenced by various factors which will be analysed .
- The maximum housing price recorded is $7,700,000. This represents the highest observed price in the dataset.

```
#distribution of the other variables
df.hist(figsize=(20,20))

array([[<Axes: title={'center': 'price'}>,
        <Axes: title={'center': 'bedrooms'}>,
        <Axes: title={'center': 'bathrooms'}>,
        <Axes: title={'center': 'sqft_living'}>],
       [<Axes: title={'center': 'sqft_lot'}>,
        <Axes: title={'center': 'floors'}>,
        <Axes: title={'center': 'sqft_above'}>,
        <Axes: title={'center': 'yr_built'}>],
```

```
 [<Axes: title={'center': 'zipcode'}>,
  <Axes: title={'center': 'lat'}>,
  <Axes: title={'center': 'long'}>,
  <Axes: title={'center': 'sqft_living15'}>],
 [<Axes: title={'center': 'sqft_lot15'}>, <Axes: >, <Axes: >,
  <Axes: >]], dtype=object)
```



majority of the distributions are skewed

Bivariate Analysis

In this section we will examine the relationships between our target variable-price and other attributes in the dataset. This will help us understand how one variable affects or is affected by the other varriables.

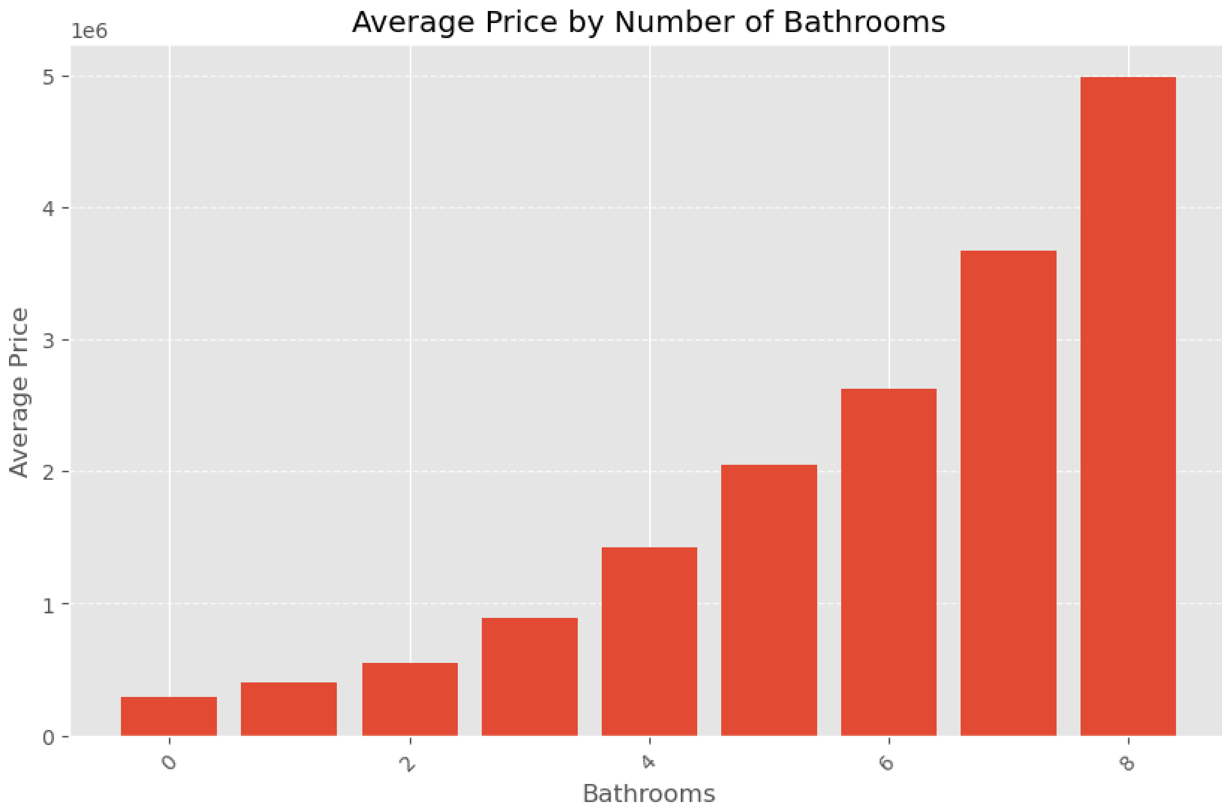
DESCRIPTIVE ANALYISIS

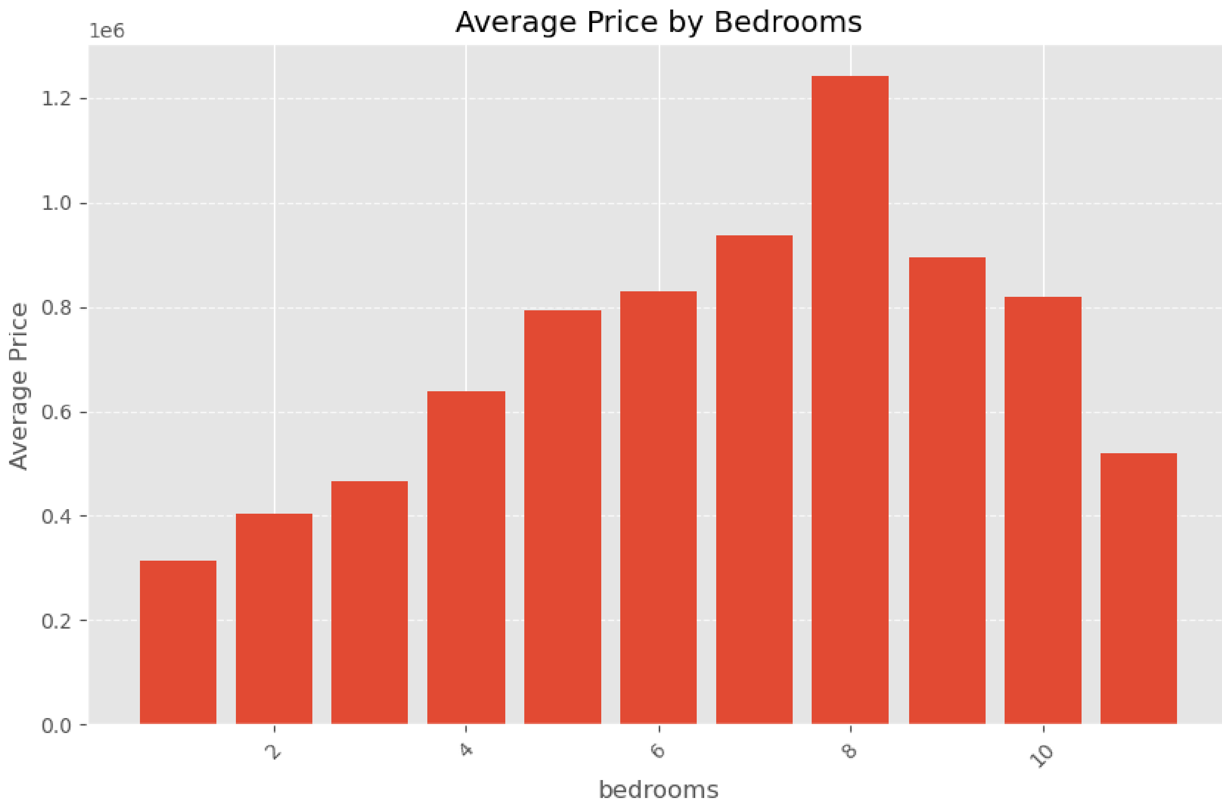
```
# The relationship between number of bedrooms, bathrooms, sqft_living,
sqft and price
average_price_by_bedrooms = df.groupby('bedrooms')['price'].mean()
avg_price_by_bathrooms = df.groupby('bathrooms')['price'].mean()
avg_price_by_sqft_living = df.groupby('sqft_living')['price'].mean()
avg_price_by_sqft_lot = df.groupby('sqft_lot')['price'].mean()
avg_price_by_floors = df.groupby('floors')['price'].mean()
avg_price_by_waterfront     = df.groupby('waterfront')['price'].mean()
avg_price_by_view = df.groupby('view')
['price'].mean().sort_values(ascending=True)
avg_price_by_condition = df.groupby('condition')
['price'].mean().sort_values(ascending=True)
avg_price_by_grade = df.groupby('grade')
['price'].mean().sort_values(ascending=True)
avg_price_by_sqft_above = df.groupby('sqft_above')['price'].mean()
avg_price_by_sqft_living15 = df.groupby('sqft_living15')
['price'].mean()
avg_price_by_sqft_lot15 = df.groupby('sqft_lot15')['price'].mean()
avg_price_by_yr_built = df.groupby('yr_built')['price'].mean()

# Plotting the graph
# bathrooms
plt.figure(figsize=(10, 6))
plt.bar(avg_price_by_bathrooms.index, avg_price_by_bathrooms.values)
plt.xlabel('Bathrooms')
plt.ylabel('Average Price')
plt.title('Average Price by Number of Bathrooms')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add gridlines for
better reference

#bedrooms
plt.figure(figsize=(10, 6))
plt.bar(average_price_by_bedrooms.index,
average_price_by_bedrooms.values)
plt.xlabel('bedrooms')
plt.ylabel('Average Price')
plt.title('Average Price by Bedrooms')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add gridlines for
better reference
```

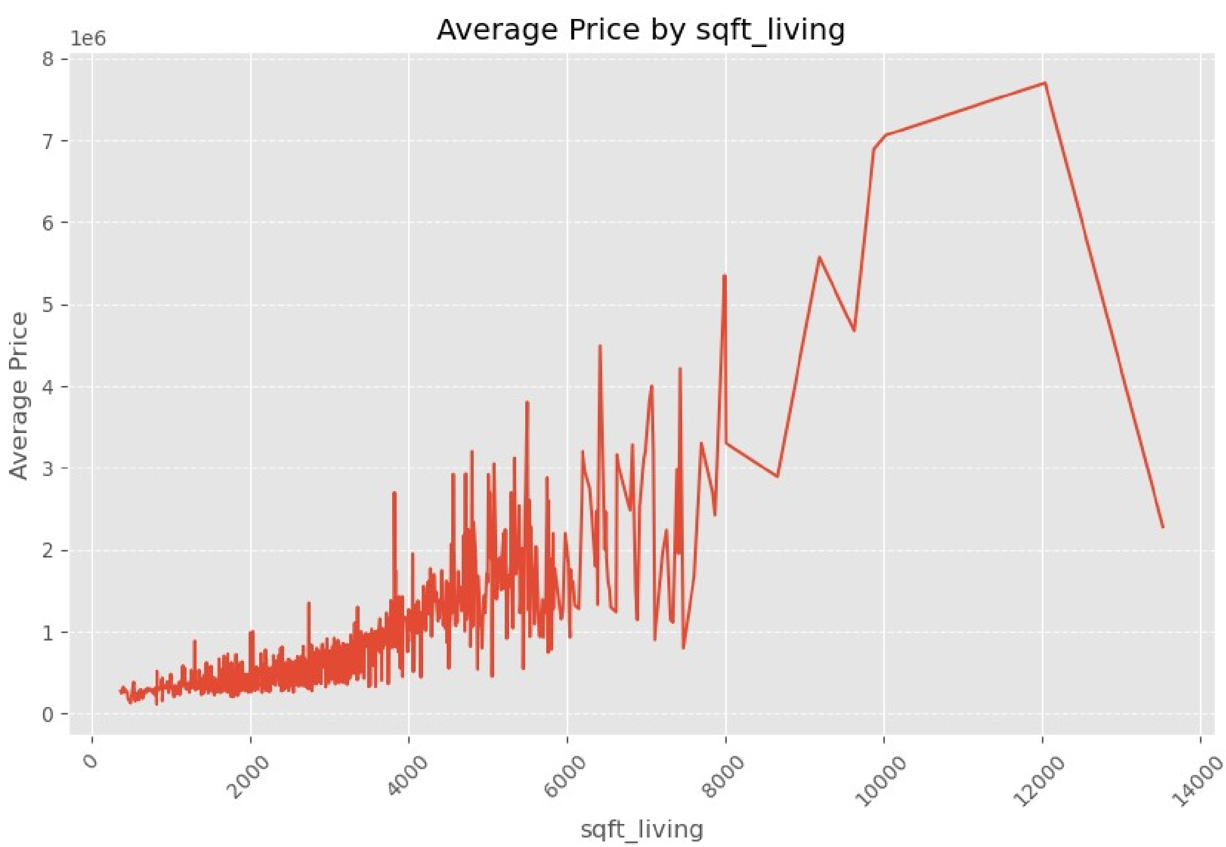
```python
#sqft_living
plt.figure(figsize=(10, 6))
plt.plot(avg_price_by_sqft_living.index,
avg_price_by_sqft_living.values)
plt.xlabel('sqft_living')
plt.ylabel('Average Price')
plt.title('Average Price by sqft_living')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add gridlines for
better reference

#sqft_lot
plt.figure(figsize=(10, 6))
plt.plot(avg_price_by_sqft_lot.index, avg_price_by_sqft_lot.values)
plt.xlabel('sqft_lot')
plt.ylabel('Average Price')
plt.title('Average Price by sqft_lot')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add gridlines for
better reference

#floors
plt.figure(figsize=(10, 6))
plt.bar(avg_price_by_floors.index, avg_price_by_floors.values)
plt.xlabel('floors')
plt.ylabel('Average Price')
plt.title('Average Price by floors')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add gridlines for
better reference

#waterfront
plt.figure(figsize=(10, 6))
plt.bar(avg_price_by_waterfront.index, avg_price_by_waterfront.values)
plt.xlabel('waterfront')
plt.ylabel('Average Price')
plt.title('Average Price by waterfront')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add gridlines for
better reference

#view
plt.figure(figsize=(10, 6))
plt.bar(avg_price_by_view.index, avg_price_by_view.values)
plt.xlabel('view')
plt.ylabel('Average Price')
plt.title('Average Price by view')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add gridlines for
better reference
```

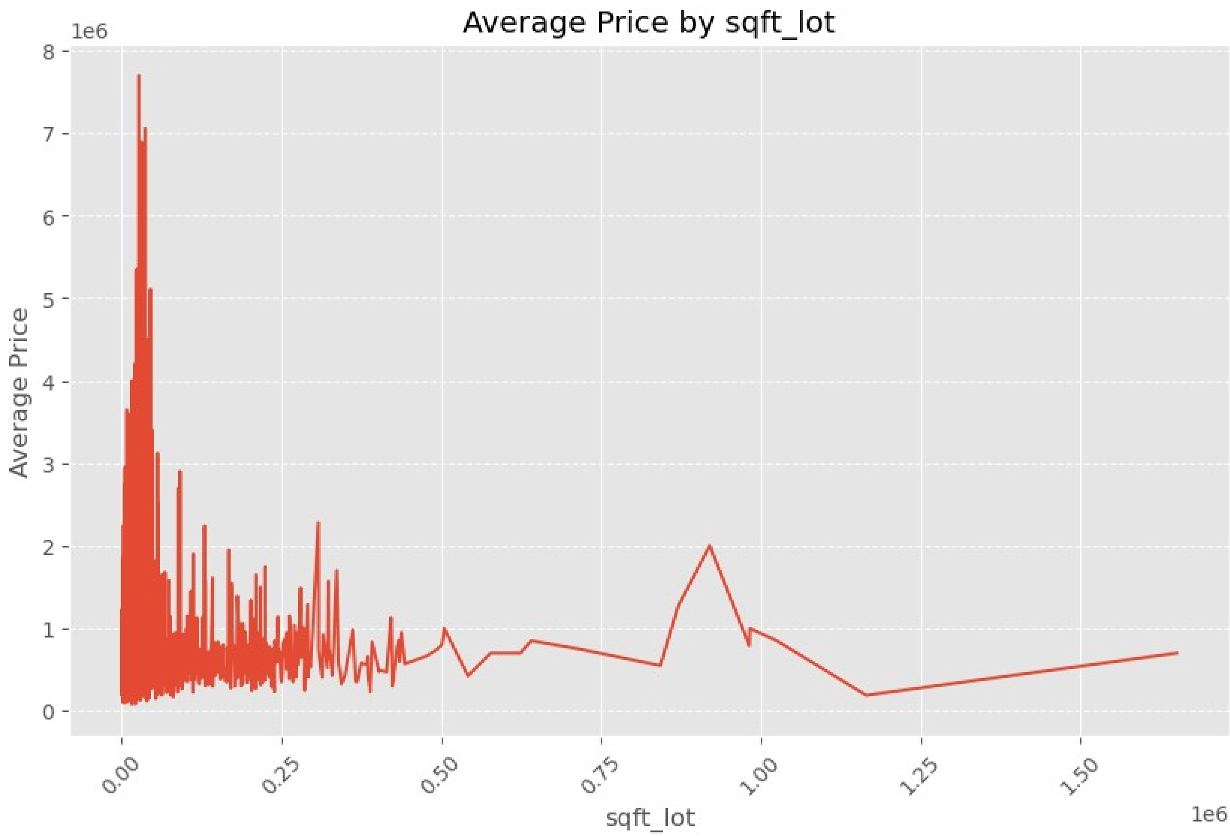
```
#condition
plt.figure(figsize=(10, 6))
plt.bar(avg_price_by_condition.index, avg_price_by_condition.values)
plt.xlabel('condition')
plt.ylabel('Average Price')
plt.title('Average Price by condition')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add gridlines for
better reference

#grade
plt.figure(figsize=(10, 6))
plt.bar(avg_price_by_grade.index, avg_price_by_grade.values)
plt.xlabel('grade')
plt.ylabel('Average Price')
plt.title('Average Price by grade')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add gridlines for
better reference

#yr_built
plt.figure(figsize=(10, 6))
plt.plot(avg_price_by_yr_built.index, avg_price_by_yr_built.values)
plt.xlabel('yr_built')
plt.ylabel('Average Price')
plt.title('Average Price by yr_built')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)

#sqft_above
plt.figure(figsize=(10, 6))
plt.plot(avg_price_by_sqft_above.index,
avg_price_by_sqft_above.values)
plt.xlabel('sqft_above')
plt.ylabel('Average Price')
plt.title('Average Price by Number of sqft_above')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)

#sqft_living15
plt.figure(figsize=(10, 6))
plt.plot(avg_price_by_sqft_living15.index,
avg_price_by_sqft_living15.values)
plt.xlabel('sqft_living15')
plt.ylabel('Average Price')
plt.title('Average Price by sqft_living15')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add gridlines for
better reference
```

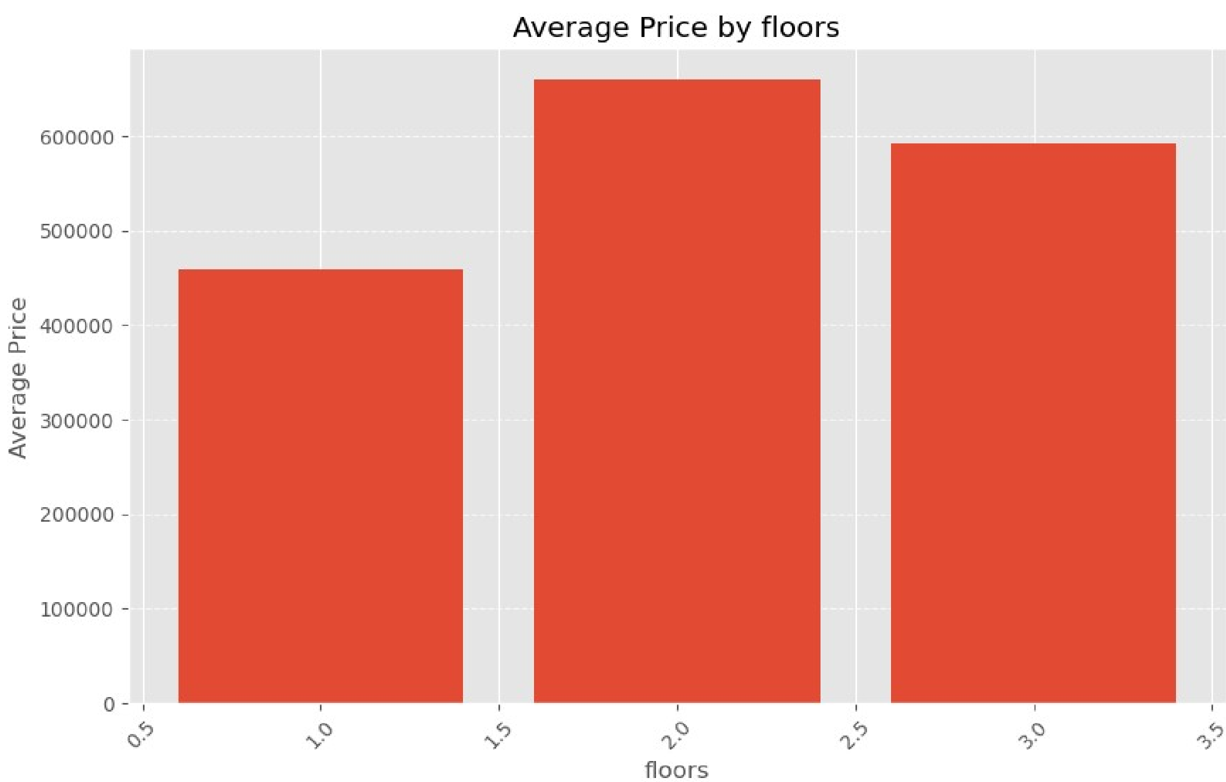
```
#sqft_lot15
plt.figure(figsize=(10, 6))
plt.plot(avg_price_by_sqft_lot15.index,
avg_price_by_sqft_lot15.values)
plt.xlabel('sqft_lot15')
plt.ylabel('Average Price')
plt.title('Average Price by sqft_lot15')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add gridlines for
better reference

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

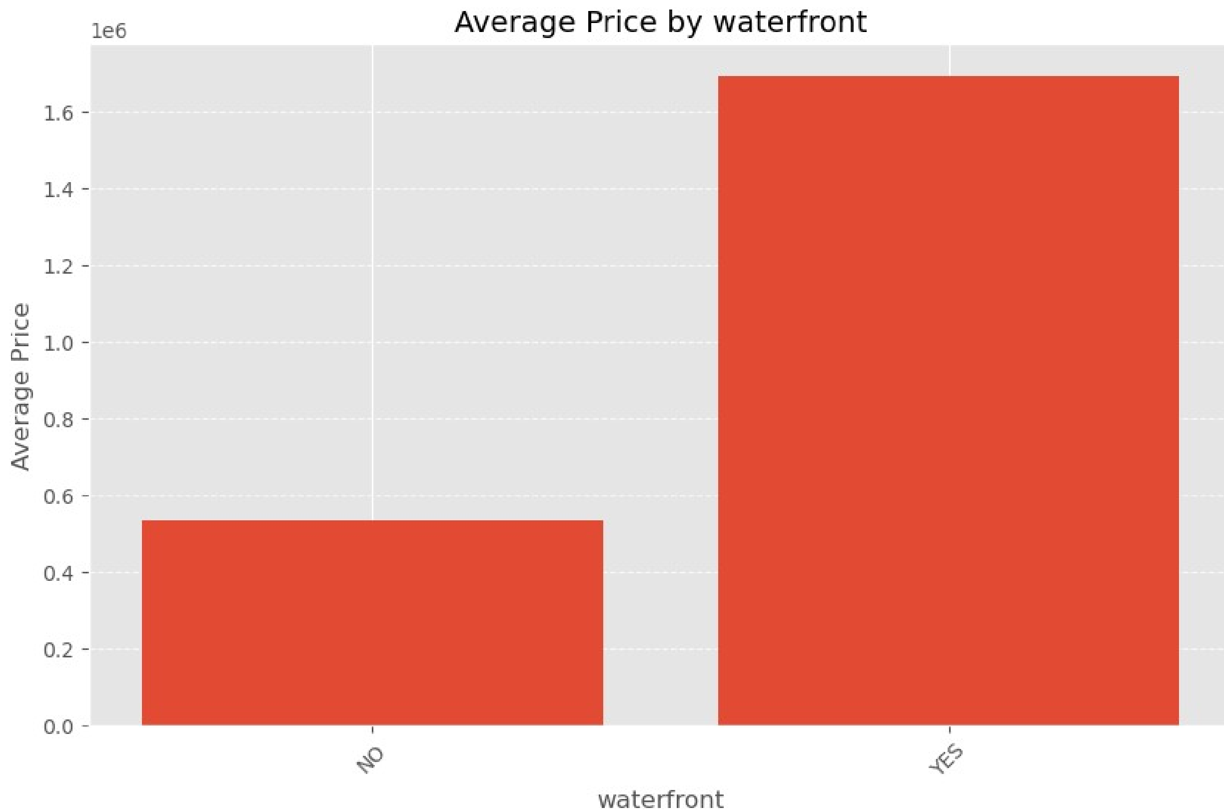



Average Price by Bedrooms
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Average Price
2
4
6
8
10
bedrooms

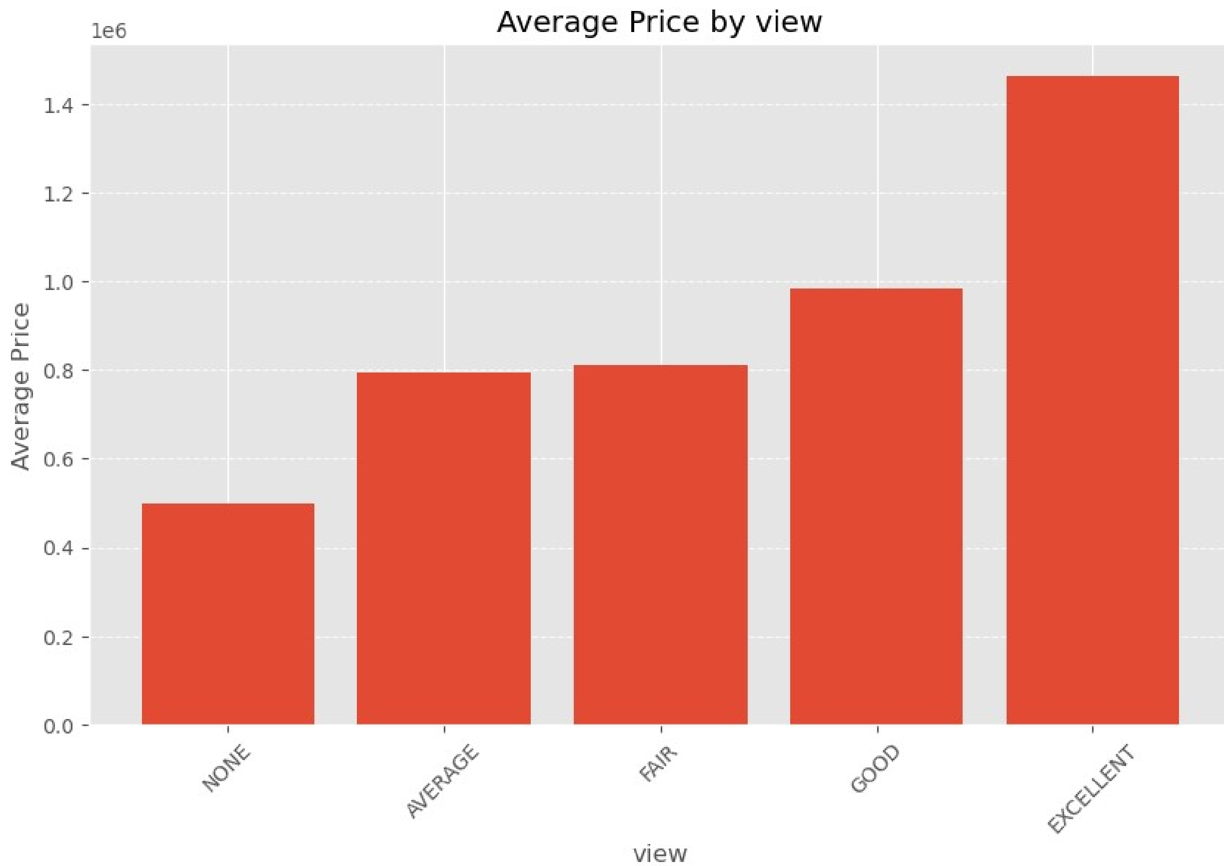

Average Price by sqft_living
1e6
8
7
6
5
4
3
2
1
0
Average Price
0
2000
4000
6000
8000
10000
12000
14000
sqft_living

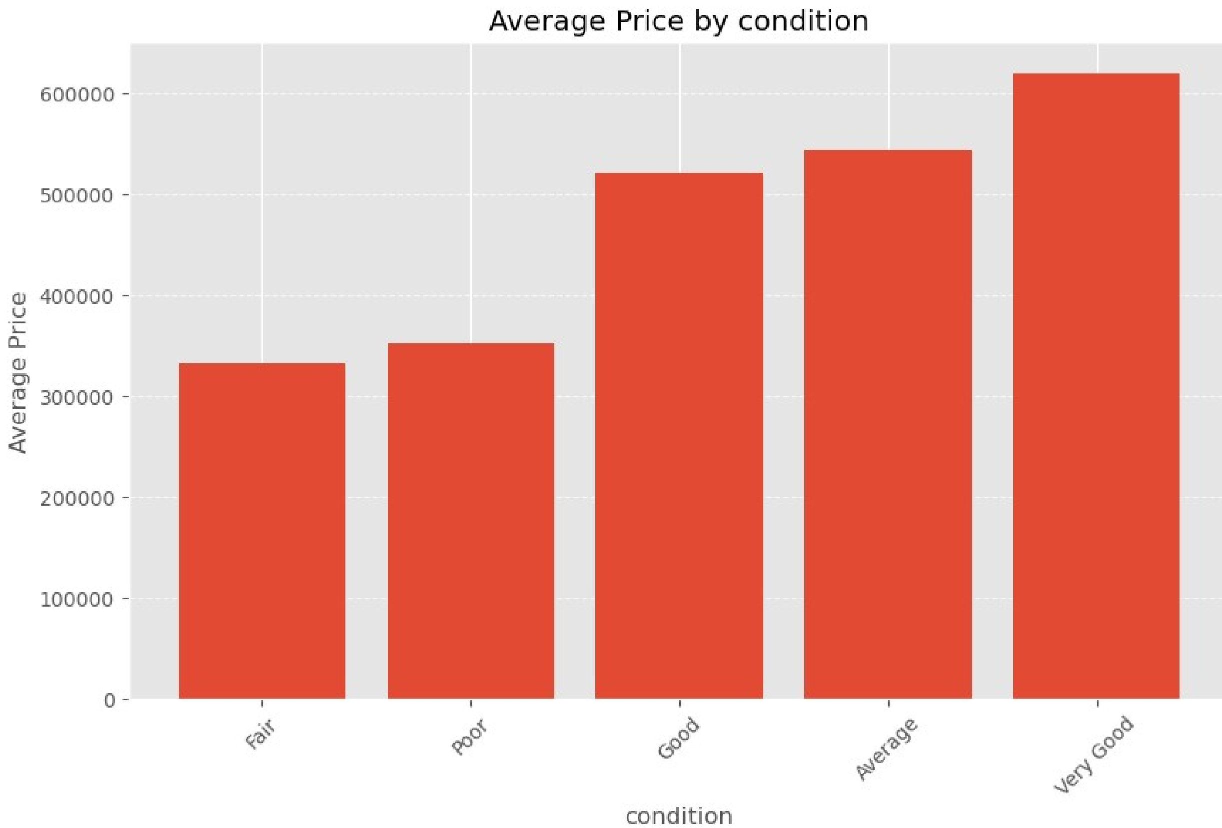

Average Price by sqft_lot
1e6
8
7
6
5
4
3
2
1
0
Average Price
0.00
0.25
0.50
0.75
1.00
1.25
1.50
sqft_lot
1e6

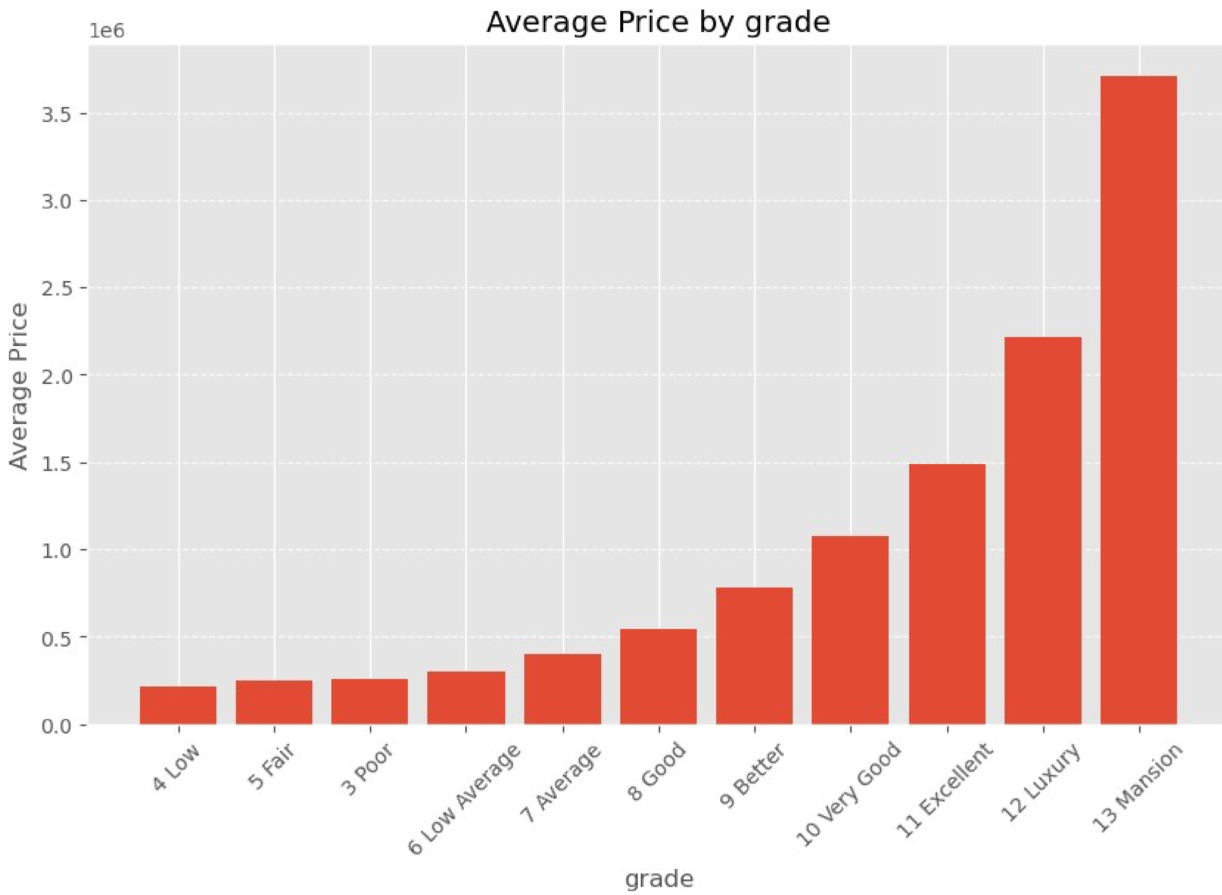

Average Price by floors
600000
500000
400000
300000
200000
100000
0
Average Price
0.5
1.0
1.5
2.0
2.5
3.0
3.5
floors



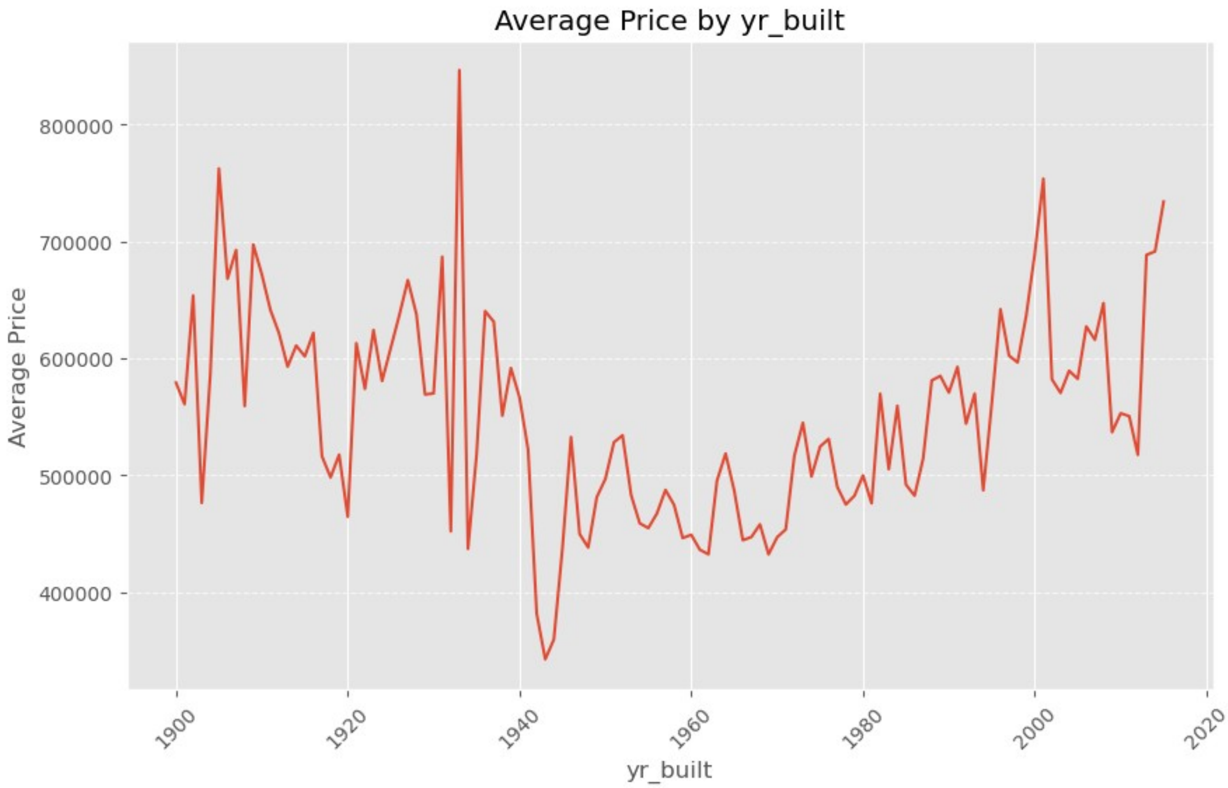
Average Price by waterfront
1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Average Price
NO
YES
waterfront

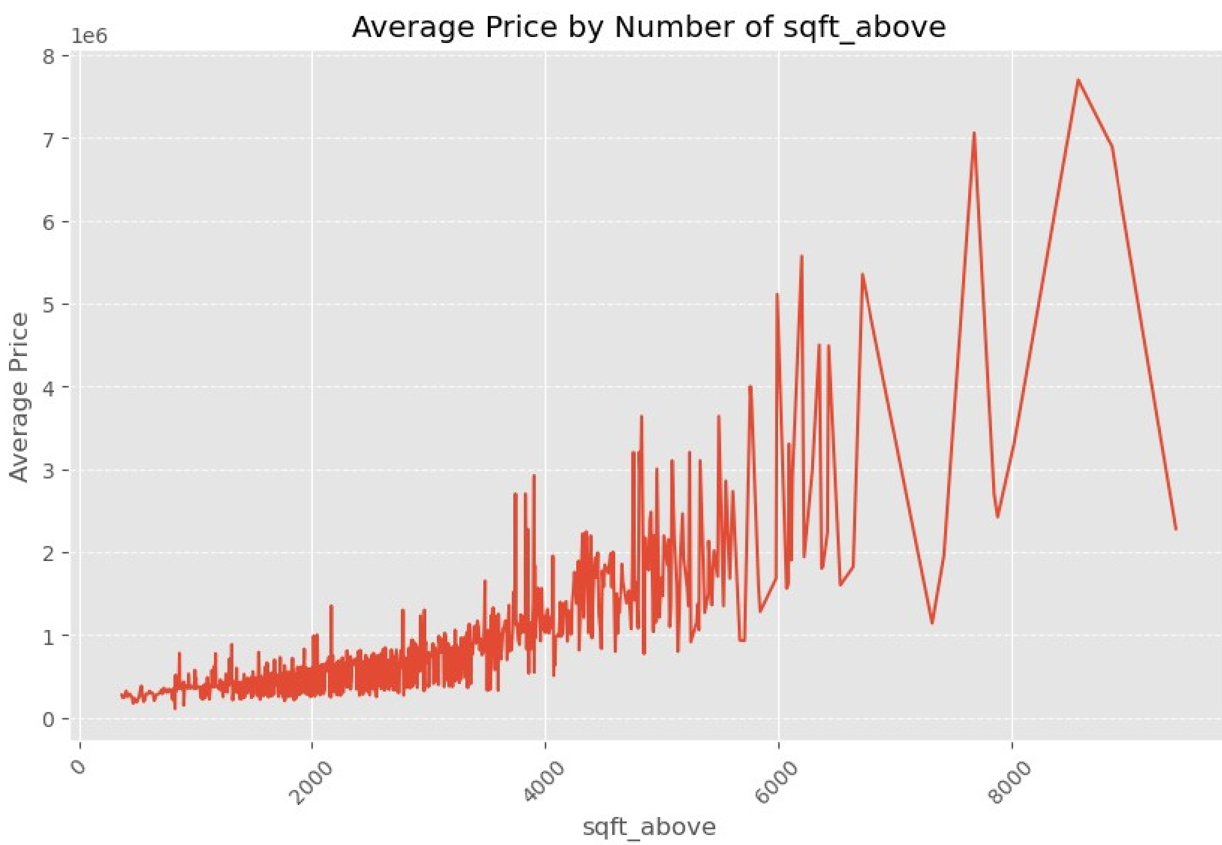

Average Price by view
1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Average Price
NONE
AVERAGE
FAIR
GOOD
EXCELLENT
view

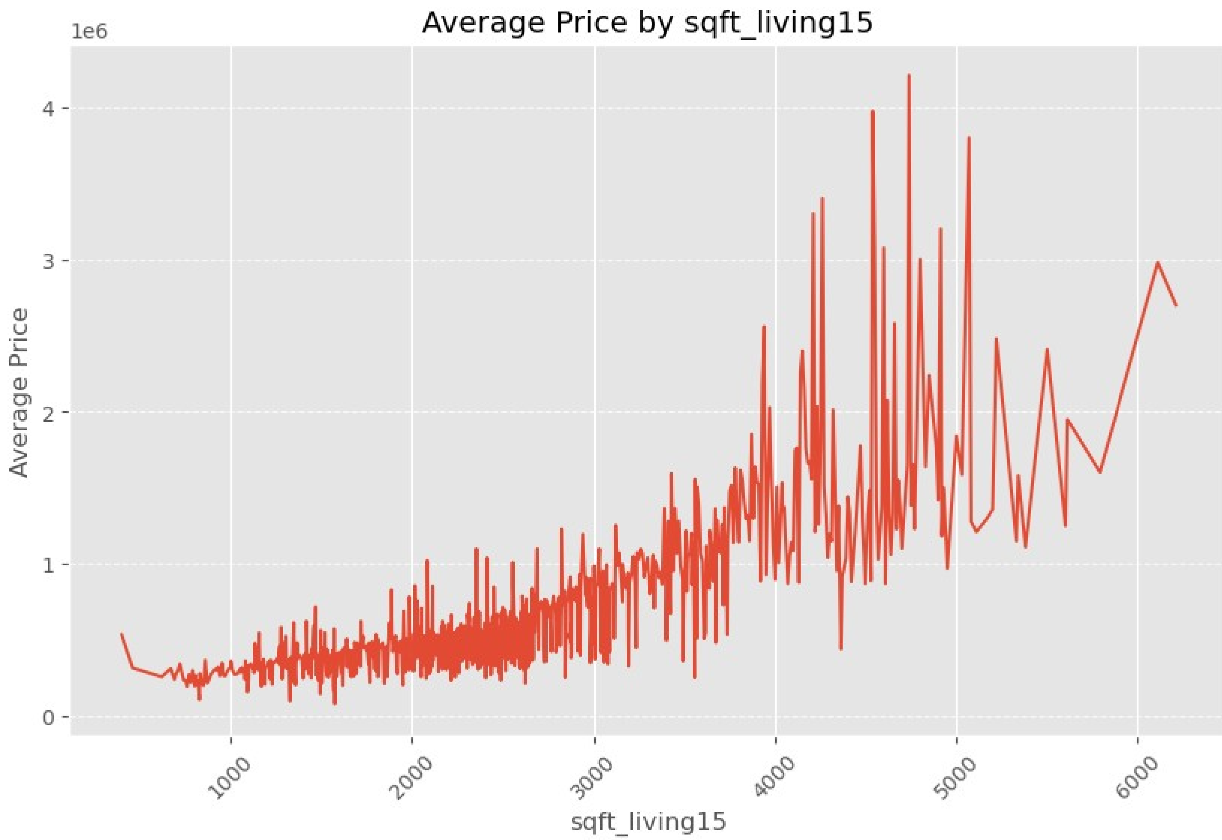

Average Price by condition
600000
500000
400000
300000
200000
100000
0
Average Price
Fair
Poor
Good
Average
Very Good
condition

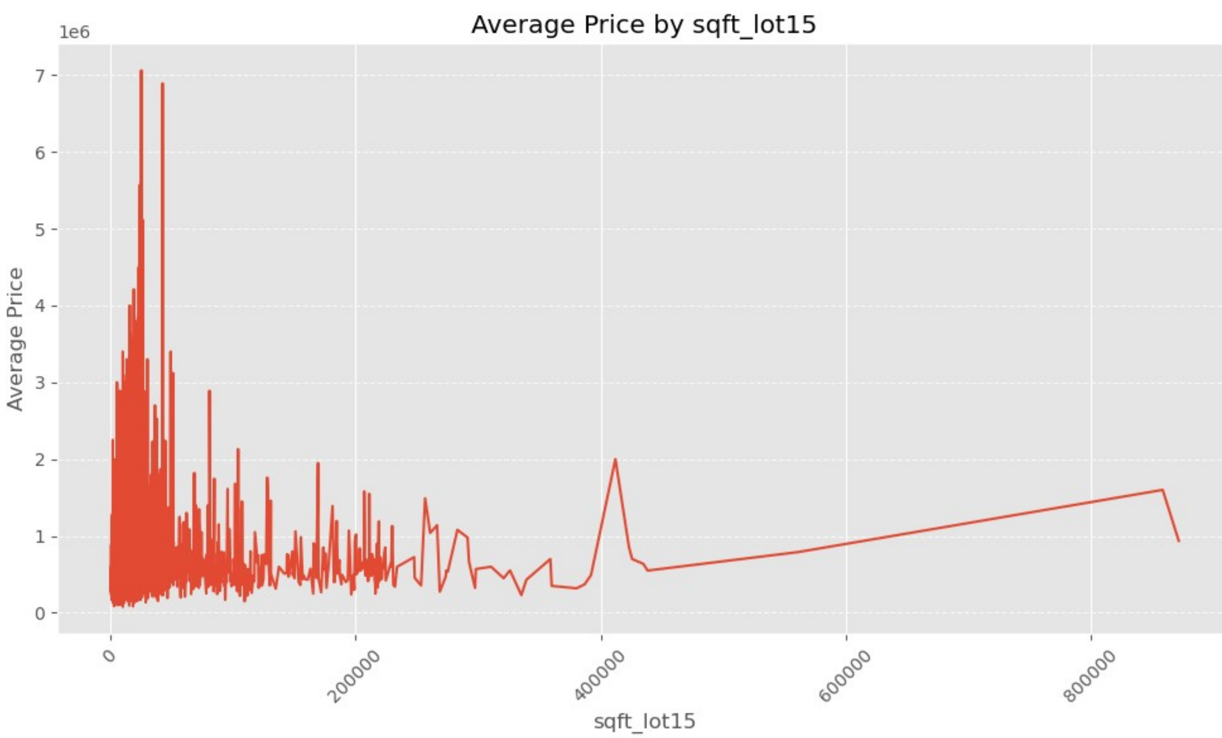


Average Price by grade
1e6
Average Price
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
4 Low
5 Fair
3 Poor
6 Low Average
7 Average
8 Good
9 Better
10 Very Good
11 Excellent
12 Luxury
13 Mansion
grade


Average Price by yr_built
Average Price
800000
700000
600000
500000
400000
1900
1920
1940
1960
1980
2000
2020
yr_built


Average Price by Number of sqft_above
1e6
8
7
6
5
4
3
2
1
0
Average Price
0
2000
4000
6000
8000
sqft_above





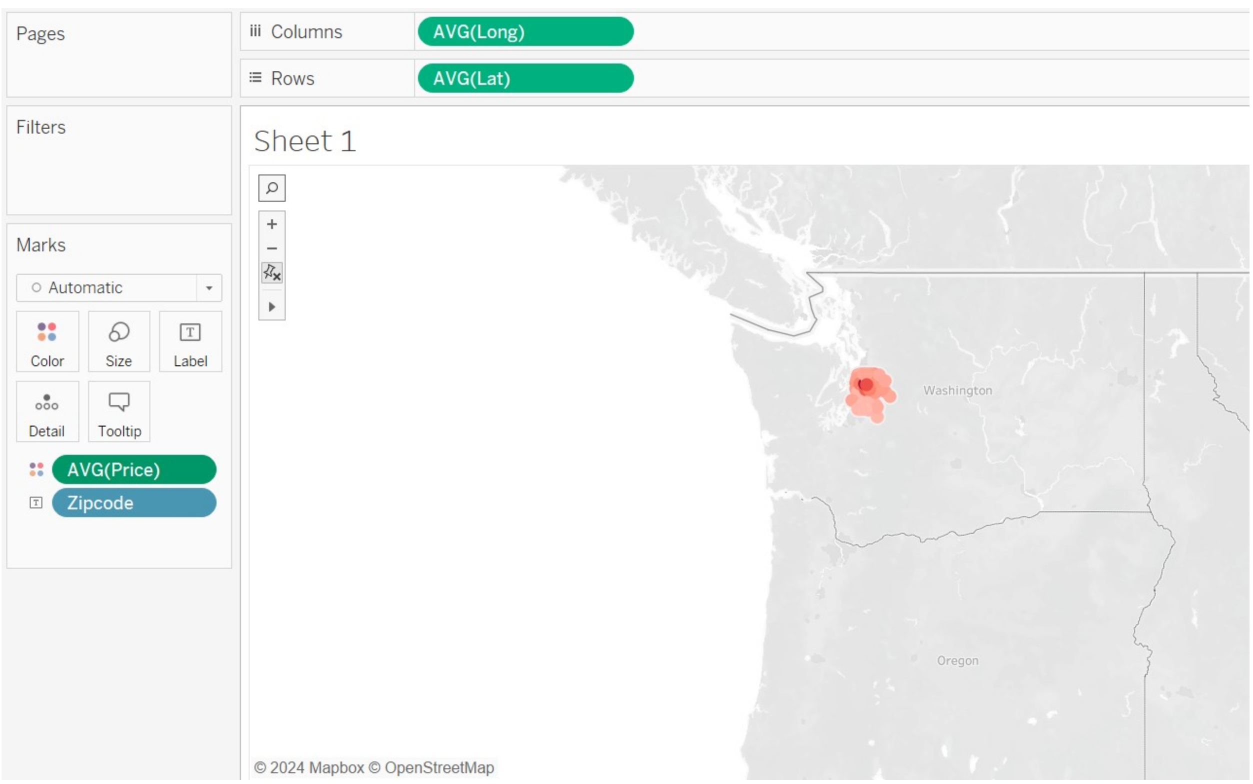
INSIGHTS FROM THIS DESCRIPTIVE ANALYSIS

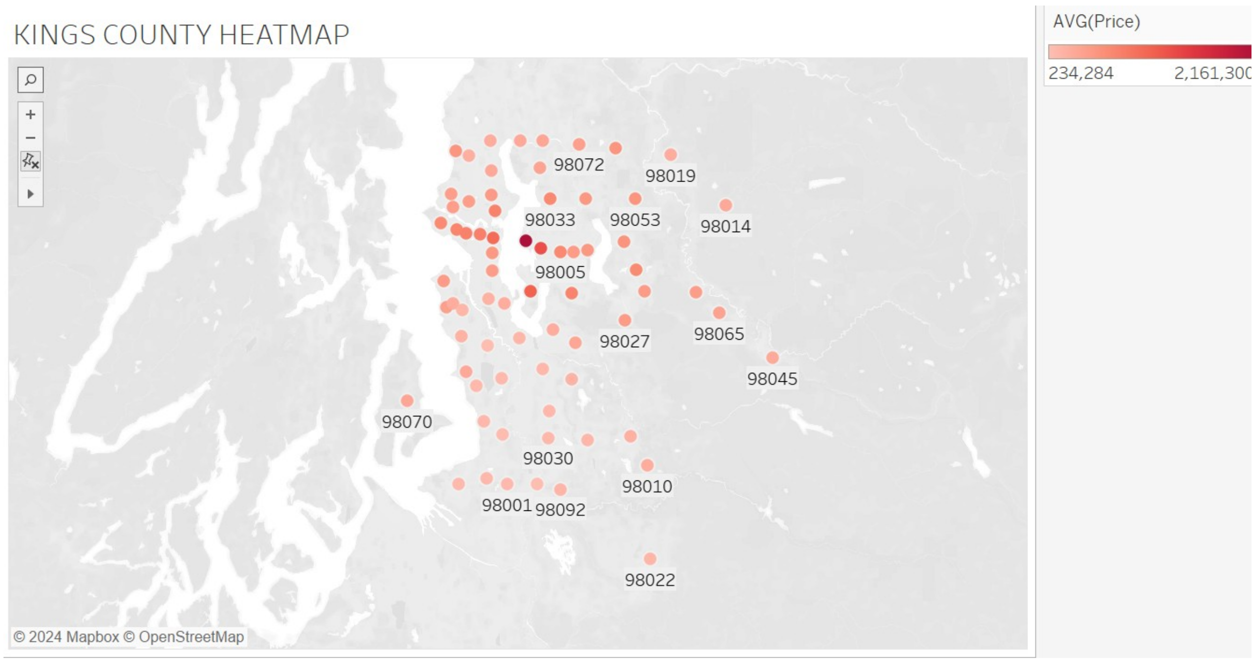
FROM VISUALIZATION OF THE GRAPHS ABOVE, IT IS APPARENT THAT THE AVERAGE PRICE OF THE HOUSES IS HIGHER:

1. WITH INCREASE IN THE NUMBER OF BATHROOMS
2. WITH PRESENCE OF THE WATERFRONT
3. WITH A BETTER VIEW
4. WITH IMPROVEMENT IN THE CONDITION ON THE HOUSE
5. WITH A HIGHER GRADE OF THE HOUSE
6. WITH MORE SQFT_LIVING
7. WITH MORE SQFT_ABOVE

THE AVERAGE PRICE OF HOUSES INCREASES UPTO 8 BEDROOMS, AFTER WHICH IT DECREASES. FROM THE TIME SERIES, THE YEAR A HOUSE WAS BUILT DID NOT APPEAR TO HAVE AN INFLUENCE ON THE PRICE OF THE HOUSES. ON AVERAGE HOUSES WITH 2 FLOORS HAVE A HIGHER PRICE THAN THOSE WITH 3 FLOORS, SO HAVING MORE FLOORS MAY NOT NECESSARILY TRANSLATE TO A HIGHER PRICE OF THE HOUSE.

USING TABLEAU WE'VE PLOTTED A HEAT MAP OF THE PRICES OF HOUSES USING THE LATITUDE, LONGITUDE AND ZIPCODES





USING TABLEAU WE'VE PLOTTED A HEAT MAP OF THE PRICES OF HOUSES USING THE LATITUDE, LONGITUDE AND ZIPCODES

INSIGHTS

FROM THIS HEATMAP IS APPARENT THAT THE BEST PRICED HOUSES (DARKER SHADE OF RED) ARE IN THE CENTRAL REGION OF KINGS COUNTY.

Online research as to why this is the case revealed that: The central region of Kings County's allure stems from proximity to essential amenities like employment centers, schools, and recreational facilities, enhancing overall quality of life.

Historical significance and architectural charm add to its appeal, fostering a sense of community and cultural richness. Urban development initiatives, along with limited land availability, drive competition among buyers, resulting in escalating property values.

Prestigious perception and affluent resident demographics further elevate demand and prices.

These factors, alongside favorable market conditions and investor interest, consolidate the central region's status as a highly coveted residential destination within Kings County.

```python
# we can now drop longititude, latitude and zipcode columns
df = df.drop(columns=['long', 'lat', 'zipcode'])
```

INFERENTIAL STATISTICS

WE'LL USE STATISTICAL TESTS TO CHECK IF THE DIFFERENCES SHOWN ABOVE ARE STATISTICALLY SIGNIFICANT

IN THIS CASE WE'RE USING MANN WHITNEY U AND KRUSKAL WALLIS TESTS RATHER THAN T-TESTS AND ANOVA BECAUSE MOST OF THE VARIABLES DO NOT SATISFY THE HOMOGENEITY OF VARIANCES ASSUMPTIONS OF T-TESTS AND ANOVA AND THE DISTRIBUTIONS ARE ALSO SKEWED

MANN WHITEY U AND KRUSKAL WALLIS ARE NON PARAMETRIC TESTS THAT DO NOT ASSUME NORMALITY OR EQUAL VARIANCES AND THEY ARE ALSO ROBUST AGAINST SKEWED DATA.

The post hoc pairwise comparisons, conducted after the Kruskal wallis statistical test has been performed to determine which groups differ from each other significantly. Therefore providing insights into the specific differences between groups that may not be apparent from the initial analysis. It is important to know which specific groups have a statistically significant difference

```python
from scipy import stats

# Variables to check
variables = ['bathrooms', 'waterfront', 'view', 'condition', 'grade', 
'bedrooms', 'floors']

for var in variables:
    # Extract data for the variable
    groups = [df[df[var] == val]['price'] for val in df[var].unique()]

    # Normality test (Kolmogorov-Smirnov)
    normality_results = [stats.kstest(group, 'norm') for group in 
groups]
    p_values_normality = [result[1] for result in normality_results]

    # Homogeneity of Variance test (Levene's test)
    homogeneity_result = stats.levene(*groups)

    # Print results
    print(f"\n{var.capitalize()}:")
    print("Normality p-values:", p_values_normality)
    print("Homogeneity of Variance p-value:", 
homogeneity_result.pvalue)
```

```
Bathrooms:
Normality p-values: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Homogeneity of Variance p-value: 0.0

Waterfront:
Normality p-values: [0.0, 0.0]
Homogeneity of Variance p-value: 1.3817112564111914e-144

View:
Normality p-values: [0.0, 0.0, 0.0, 0.0, 0.0]
Homogeneity of Variance p-value: 1.0125963924207291e-288
```

```
Condition:
Normality p-values: [0.0, 0.0, 0.0, 0.0, 0.0]
Homogeneity of Variance p-value: 4.880995092583118e-11

Grade:
Normality p-values: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0]
Homogeneity of Variance p-value: 0.0

Bedrooms:
Normality p-values: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0]
Homogeneity of Variance p-value: 2.0204244112631658e-217

Floors:
Normality p-values: [0.0, 0.0, 0.0]
Homogeneity of Variance p-value: 2.154129038756904e-108
```

```python
import pandas as pd
from scipy.stats import skew, kurtosis

#checking skewness and Kurtosis of all numerical variables

# Calculate skewness, kurtosis, variance, and standard deviation for
each variable
summary_stats = pd.DataFrame({
    'Skewness': num_attributes.apply(skew),
    'Kurtosis': num_attributes.apply(kurtosis),
    'Variance': num_attributes.var(),
    'Std Deviation': num_attributes.std()
})

# Display the summary statistics table
print(summary_stats)
```

```
               Skewness  Kurtosis         Variance  Std Deviation
price              4.08     35.29 137574177905.13      370909.93
bedrooms           0.55      1.89             0.82           0.90
sqft_living        1.50      5.55        849966.43         921.94
sqft_lot          13.26    301.50     1662450593.47       40773.16
sqft_above         1.46      3.50        691858.07         831.78
yr_built          -0.47     -0.66           863.67          29.39
zipcode            0.41     -0.85          2856.88          53.45
lat               -0.49     -0.68             0.02           0.14
long               0.88      1.05             0.02           0.14
sqft_living15      1.12      1.67        468935.20         684.79
sqft_lot15         9.76    159.11      759183840.08       27553.29
```

```python
from scipy import stats

from scipy import stats
```

```
# Assuming you have a DataFrame 'df' with a column 'price' and a column 'waterfront' indicating whether a property has waterfront or not.

# Separate the data into two groups: with waterfront and without waterfront
waterfront_group = df[df['waterfront'] == 'YES']['price']
non_waterfront_group = df[df['waterfront'] == 'NO']['price']

# Perform Mann-Whitney U test
u_statistic, p_value = stats.mannwhitneyu(waterfront_group, non_waterfront_group, alternative='two-sided')

# Print the results
print("Mann-Whitney U statistic:", u_statistic)
print("p-value:", p_value)

# Check if the difference is statistically significant at alpha = 0.05
alpha = 0.05
if p_value < alpha:
    print("The difference in average price between the group with a waterfront and the group without a waterfront is statistically significant.")
else:
    print("There is no statistically significant difference in average price between the group with a waterfront and the group without a waterfront.")
```

```
Mann-Whitney U statistic: 2455123.5
p-value: 3.7024964710112914e-59
The difference in average price between the group with a waterfront
and the group without a waterfront is statistically significant.
```

```
from scipy import stats

# Create lists to hold prices for each group of bathrooms
bathroom_groups = []
for bathrooms in df['bathrooms'].unique():
    bathroom_groups.append(df[df['bathrooms'] == bathrooms]['price'])

# Perform Kruskal-Wallis test
h_statistic, p_value = stats.kruskal(*bathroom_groups)

# Print the results
print("Kruskal-Wallis H statistic:", h_statistic)
print("p-value:", p_value)

# Check if the difference is statistically significant at alpha = 0.05
alpha = 0.05
if p_value < alpha:
```

```
    print("The difference in average price between groups with different numbers of bathrooms is statistically significant.")
else:
    print("There is no statistically significant difference in average price between groups with different numbers of bathrooms.")

# Post-hoc analysis (Conover-Iman test)
posthoc = sm.stats.multicomp.pairwise_tukeyhsd(df['price'], df['bathrooms'])
print("\nPost-Hoc Analysis (Conover-Iman Test):")
print(posthoc)
```

```
Kruskal-Wallis H statistic: 4216.355571576836
p-value: 0.0
The difference in average price between groups with different numbers of bathrooms is statistically significant.

Post-Hoc Analysis (Conover-Iman Test):
       Multiple Comparison of Means - Tukey HSD, FWER=0.05
==================================================================
group1 group2   meandiff   p-adj     lower        upper     reject
------------------------------------------------------------------
     0      1   108333.045 0.0852   -6954.2372  223620.3272  False
     0      2  255962.6379    0.0   140786.984  371138.2918   True
     0      3  597449.7969    0.0  480702.8495  714196.7444   True
     0      4 1136736.9338    0.0 1009702.5683 1263771.2993   True
     0      5 1758899.2287    0.0 1570936.9435 1946861.5139   True
     0      6  2339826.058    0.0 2017288.8309  2662363.285   True
     0      7  3380826.058    0.0 2697102.6026 4064549.5133   True
     0      8  4700826.058    0.0 4017102.6026 5384549.5133   True
     1      2  147629.5929    0.0  132803.4917  162455.6941   True
     1      3  489116.7519    0.0  464945.8652  513287.6386   True
     1      4 1028403.8888    0.0  972797.1261 1084010.6514   True
     1      5 1650566.1837    0.0 1501287.2412 1799845.1263   True
     1      6  2231493.013    0.0 1929856.4023 2533129.6236   True
     1      7  3272493.013    0.0  2598377.188 3946608.8379   True
     1      8  4592493.013    0.0  3918377.188 5266608.8379   True
     2      3   341487.159    0.0  317854.4362  365119.8818   True
     2      4  880774.2959    0.0  825399.3388   936149.253   True
     2      5 1502936.5908    0.0 1353743.8413 1652129.3403   True
     2      6 2083863.4201    0.0 1782269.4568 2385457.3834   True
     2      7 3124863.4201    0.0 2450766.6768 3798960.1633   True
     2      8 4444863.4201    0.0 3770766.6768 5118960.1633   True
     3      4  539287.1369    0.0  480714.0683  597860.2054   True
     3      5 1161449.4318    0.0 1011040.3396 1311858.5239   True
     3      6  1742376.261    0.0 1440178.7475 2044573.7746   True
     3      7  2783376.261    0.0 2109009.2706 3457743.2515   True
     3      8  4103376.261    0.0 3429009.2706 4777743.2515   True
     4      5  622162.2949    0.0  463635.4035  780689.1863   True
     4      6 1203089.1242    0.0  896770.3075 1509407.9409   True
```

```
     4      7 2244089.1242    0.0 1567865.2574  2920312.991   True
     4      8 3564089.1242    0.0 2887865.2574  4240312.991   True
     5      6  580926.8293    0.0  244737.4361  917116.2225   True
     5      7 1621926.8293    0.0  931658.1796 2312195.4789   True
     5      8 2941926.8293    0.0 2251658.1796 3632195.4789   True
     6      7    1041000.0 0.0004  302642.8704 1779357.1296   True
     6      8    2361000.0    0.0 1622642.8704 3099357.1296   True
     7      8    1320000.0 0.0006  366785.0445 2273214.9555   True
------------------------------------------------------------------
```

```python
df['view'].unique()
```

```
array(['NONE', 'GOOD', 'EXCELLENT', 'AVERAGE', 'FAIR'], dtype=object)
```

```python
from scipy import stats

# Create lists to hold prices for each view group
view_groups = []
for view_quality in ['NONE', 'GOOD', 'EXCELLENT', 'AVERAGE', 'FAIR']:
    view_groups.append(df[df['view'] == view_quality]['price'])

# Perform Kruskal-Wallis test
h_statistic, p_value = stats.kruskal(*view_groups)

# Print the results
print("Kruskal-Wallis H statistic:", h_statistic)
print("p-value:", p_value)

# Check if the difference is statistically significant at alpha = 0.05
alpha = 0.05
if p_value < alpha:
    print("The difference in average price between groups with
different views is statistically significant.")
else:
    print("There is no statistically significant difference in average
price between groups with different views.")

# Post-hoc analysis (Conover-Iman test)
posthoc = sm.stats.multicomp.pairwise_tukeyhsd(df['price'],
df['view'])
print("\nPost-Hoc Analysis (Conover-Iman Test):")
print(posthoc)
```

```
Kruskal-Wallis H statistic: 1708.5494629568025
p-value: 0.0
The difference in average price between groups with different views is
statistically significant.

Post-Hoc Analysis (Conover-Iman Test):
           Multiple Comparison of Means - Tukey HSD, FWER=0.05
```

```
===================================================================
===
  group1    group2    meandiff   p-adj      lower        upper
reject
-------------------------------------------------------------------
---
  AVERAGE EXCELLENT  667531.7573    0.0   604561.4078  730502.1069
True
  AVERAGE      FAIR   17590.4374 0.9411   -45299.5303    80480.405
False
  AVERAGE      GOOD  188253.5171    0.0   133628.1895  242878.8447
True
  AVERAGE      NONE  -296755.832    0.0  -329475.3765 -264036.2875
True
EXCELLENT      FAIR   -649941.32    0.0   -726608.876  -573273.764
True
EXCELLENT      GOOD -479278.2402    0.0  -549326.0277 -409230.4528
True
EXCELLENT      NONE -964287.5893    0.0 -1018998.9657 -909576.2129
True
     FAIR      GOOD  170663.0797    0.0   100687.5438  240638.6157
True
     FAIR      NONE -314346.2694    0.0  -368965.1106 -259727.4281
True
     GOOD      NONE -485009.3491    0.0  -529864.6297 -440154.0685
True
-------------------------------------------------------------------
---
```

```python
from scipy import stats

# Create lists to hold prices for each condition group
condition_groups = []
for condition in df['condition'].unique():
    condition_groups.append(df[df['condition'] == condition]['price'])

# Perform Kruskal-Wallis test
h_statistic, p_value = stats.kruskal(*condition_groups)

# Print the results
print("Kruskal-Wallis H statistic:", h_statistic)
print("p-value:", p_value)

# Check if the difference is statistically significant at alpha = 0.05
alpha = 0.05
if p_value < alpha:
    print("The difference in average price between groups with
different conditions is statistically significant.")
else:
    print("There is no statistically significant difference in average
```

```
price between groups with different conditions.")

# Post-hoc analysis (Conover-Iman test)
posthoc = sm.stats.multicomp.pairwise_tukeyhsd(df['price'],
df['condition'])
print("\nPost-Hoc Analysis (Conover-Iman Test):")
print(posthoc)
```

```
Kruskal-Wallis H statistic: 230.73692688865907
p-value: 9.16111300461158e-49
The difference in average price between groups with different
conditions is statistically significant.

Post-Hoc Analysis (Conover-Iman Test):
         Multiple Comparison of Means - Tukey HSD, FWER=0.05
======================================================================
 group1   group2    meandiff   p-adj     lower        upper     reject
----------------------------------------------------------------------
Average      Fair -210573.2017    0.0 -292859.1084 -128287.2951   True
Average      Good  -22144.8399 0.0031  -38980.8491   -5308.8307   True
Average      Poor -190679.8004 0.0747 -392551.5305   11191.9297  False
Average Very Good   76123.8004    0.0   48628.3888  103619.2119   True
   Fair      Good  188428.3618    0.0  105416.8217   271439.902   True
   Fair      Poor   19893.4013 0.9991 -197729.3457  237516.1483  False
   Fair Very Good  286697.0021    0.0  200886.3836  372507.6206   True
   Good      Poor -168534.9605 0.1533 -370703.5554   33633.6344  False
   Good Very Good   98268.6403    0.0   68672.3057  127864.9749   True
   Poor Very Good  266803.6008 0.0032   63469.6705  470137.5311   True
----------------------------------------------------------------------
```

```
from scipy import stats

# Create lists to hold prices for each grade group
grade_groups = []
for grade in df['grade'].unique():
    grade_groups.append(df[df['grade'] == grade]['price'])

# Perform Kruskal-Wallis test
h_statistic, p_value = stats.kruskal(*grade_groups)

# Print the results
print("Kruskal-Wallis H statistic:", h_statistic)
print("p-value:", p_value)

# Check if the difference is statistically significant at alpha = 0.05
alpha = 0.05
if p_value < alpha:
    print("The difference in average price between groups with
different grades is statistically significant.")
else:
```

```
    print("There is no statistically significant difference in average
price between groups with different grades.")


# Post-hoc analysis (Conover-Iman test)
posthoc = sm.stats.multicomp.pairwise_tukeyhsd(df['price'],
df['grade'])
print("\nPost-Hoc Analysis (Conover-Iman Test):")
print(posthoc)
```

```
Kruskal-Wallis H statistic: 8614.71244249437
p-value: 0.0
The difference in average price between groups with different grades
is statistically significant.

Post-Hoc Analysis (Conover-Iman Test):
                Multiple Comparison of Means - Tukey HSD, FWER=0.05

======================================================================
=============
    group1        group2       meandiff   p-adj      lower
upper     reject
----------------------------------------------------------------------
-------------
 10 Very Good  11 Excellent   407215.3626    0.0   356298.1242
458132.6009   True
 10 Very Good     12 Luxury  1132422.5904    0.0  1036990.5727
1227854.608   True
 10 Very Good    13 Mansion   2631772.068    0.0  2401074.4549
2862469.6812   True
 10 Very Good        3 Poor  -816997.1627 0.0562 -1643915.4125
9921.087  False
 10 Very Good         4 Low  -865195.1627    0.0 -1032522.6637  -
697867.6617   True
 10 Very Good        5 Fair  -828488.9599    0.0  -890906.5304  -
766071.3894   True
 10 Very Good 6 Low Average  -777175.4637    0.0  -809615.5338  -
744735.3937   True
 10 Very Good     7 Average   -676411.344    0.0  -703972.8597  -
648849.8284   True
 10 Very Good        8 Good   -537085.209    0.0   -565373.706  -
508796.7121   True
 10 Very Good      9 Better  -301440.1952    0.0  -332573.5132  -
270306.8772   True
 11 Excellent     12 Luxury   725207.2278    0.0   623460.1973
826954.2583   True
 11 Excellent    13 Mansion  2224556.7055    0.0  1991175.9502
2457937.4608   True
 11 Excellent        3 Poor -1224212.5253 0.0001 -2051883.3415  -
396541.7091   True
```

```
 11 Excellent         4 Low -1272410.5253    0.0 -1443418.3638 -
1101402.6868   True
 11 Excellent        5 Fair -1235704.3225    0.0 -1307406.0751 -
1164002.5698   True
 11 Excellent 6 Low Average -1184390.8263    0.0 -1232323.5539 -
1136458.0987   True
 11 Excellent     7 Average -1083626.7066    0.0  -1128401.983 -
1038851.4302   True
 11 Excellent        8 Good  -944300.5716    0.0  -989526.9719  -
899074.1712   True
 11 Excellent      9 Better  -708655.5577    0.0  -755713.7315  -
661597.3839   True
    12 Luxury    13 Mansion  1499349.4777    0.0  1252405.5938
1746293.3615   True
    12 Luxury        3 Poor -1949419.7531    0.0 -2781016.8219 -
1117822.6843   True
    12 Luxury         4 Low -1997617.7531    0.0 -2186716.6311 -
1808518.8751   True
    12 Luxury        5 Fair -1960911.5503    0.0 -2068873.7944 -
1852949.3061   True
    12 Luxury 6 Low Average -1909598.0541    0.0 -2003471.6457 -
1815724.4625   True
    12 Luxury     7 Average -1808833.9344    0.0 -1901135.2356 -
1716532.6332   True
    12 Luxury        8 Good -1669507.7994    0.0 -1762028.7814 -
1576986.8174   True
    12 Luxury      9 Better -1433862.7855    0.0 -1527292.8479 -
1340432.7232   True
   13 Mansion        3 Poor -3448769.2308    0.0 -4306480.0233 -
2591058.4383   True
   13 Mansion         4 Low -3496967.2308    0.0 -3779584.4708 -
3214349.9907   True
   13 Mansion        5 Fair -3460261.0279    0.0 -3696417.6812 -
3224104.3746   True
   13 Mansion 6 Low Average -3408947.5318    0.0 -3639004.8506  -
3178890.213   True
   13 Mansion     7 Average -3308183.4121    0.0 -3537603.6573 -
3078763.1668   True
   13 Mansion        8 Good -3168857.2771    0.0 -3398365.9934 -
2939348.5608   True
   13 Mansion      9 Better -2933212.2632    0.0  -3163088.959 -
2703335.5674   True
       3 Poor         4 Low      -48198.0    1.0    -891076.94
794680.94  False
       3 Poor        5 Fair   -11491.7972    1.0  -839949.6223
816966.0279  False
       3 Poor 6 Low Average     39821.699    1.0   -786918.147
866561.545  False
       3 Poor     7 Average   140585.8187    1.0   -685976.975
```

```
967148.6124  False
       3 Poor        8 Good   279911.9537 0.9917  -546675.4004
1106499.3078  False
       3 Poor      9 Better   515556.9675 0.6437  -311132.6346
1342246.5697  False
        4 Low        5 Fair    36706.2028 0.9999  -138070.9947
211483.4003  False
        4 Low 6 Low Average     88019.699 0.8348   -78423.9065
254463.3045  False
        4 Low     7 Average   188783.8187  0.011    23221.8883
354345.7491   True
        4 Low        8 Good   328109.9537    0.0   162425.4501
493794.4573   True
        4 Low      9 Better   563754.9675    0.0   397561.1074
729948.8277   True
       5 Fair 6 Low Average    51313.4962 0.1768    -8694.2893
111321.2817  False
       5 Fair     7 Average   152077.6159    0.0    94560.5461
209594.6857   True
       5 Fair        8 Good   291403.7509    0.0   233534.8021
349272.6996   True
       5 Fair      9 Better   527048.7647    0.0   467737.2172
586360.3122   True
6 Low Average     7 Average   100764.1197    0.0    79209.1119
122319.1275   True
6 Low Average        8 Good   240090.2547    0.0   217613.1491
262567.3602   True
6 Low Average      9 Better   475735.2685    0.0    449768.661
501701.8761   True
    7 Average        8 Good    139326.135    0.0   124748.7802
153903.4898   True
    7 Average      9 Better   374971.1488    0.0   355438.0051
394504.2926   True
       8 Good      9 Better   235645.0139    0.0   215098.8313
256191.1964   True
----------------------------------------------------------------------
-------------
```

```python
from scipy import stats


# Create lists to hold prices for each bedroom group
bedroom_groups = []
for bedrooms in df['bedrooms'].unique():
    bedroom_groups.append(df[df['bedrooms'] == bedrooms]['price'])

# Perform Kruskal-Wallis test
h_statistic, p_value = stats.kruskal(*bedroom_groups)

# Print the results
```

```python
print("Kruskal-Wallis H statistic:", h_statistic)
print("p-value:", p_value)

# Check if the difference is statistically significant at alpha = 0.05
alpha = 0.05
if p_value < alpha:
    print("The difference in average price between groups with
different numbers of bedrooms is statistically significant.")
else:
    print("There is no statistically significant difference in average
price between groups with different numbers of bedrooms.")

# Post-hoc analysis (Conover-Iman test)
posthoc = sm.stats.multicomp.pairwise_tukeyhsd(df['price'],
df['bedrooms'])
print("\nPost-Hoc Analysis (Conover-Iman Test):")
print(posthoc)
```

```
Kruskal-Wallis H statistic: 2346.9091736424602
p-value: 0.0
The difference in average price between groups with different numbers
of bedrooms is statistically significant.

C:\Users\Kola500\.anaconda\ANACONDA\Lib\site-packages\scipy\integrate\
_quadpack_py.py:1233: IntegrationWarning: The integral is probably
divergent, or slowly convergent.
  quad_r = quad(f, low, high, args=args, full_output=self.full_output,


Post-Hoc Analysis (Conover-Iman Test):
        Multiple Comparison of Means - Tukey HSD, FWER=0.05
===================================================================
group1 group2   meandiff   p-adj      lower        upper     reject
-------------------------------------------------------------------
     1      2   89005.2858 0.0461      735.0372  177275.5344   True
     1      3   151176.253    0.0    65049.5041   237303.002   True
     1      4  322890.9542    0.0   236410.4076  409371.5007   True
     1      5  479442.0777    0.0   389126.3885  569757.7669   True
     1      6   516906.181    0.0   403209.4959   630602.866   True
     1      7  623469.4343    0.0   414595.2865  832343.5821   True
     1      8  926965.0525    0.0   576318.8052 1277611.2997   True
     1      9  579783.0676 0.0033   111430.6914 1048135.4438   True
     1     10  505783.2343  0.317  -151055.0221 1162621.4907  False
     1     11  205783.2343    1.0  -925484.4979 1337050.9665  False
     2      3   62170.9672    0.0    36351.3917   87990.5428   True
     2      4  233885.6684    0.0   206909.4098  260861.9269   True
     2      5  390436.7919    0.0   352943.3617  427930.2221   True
     2      6  427900.8952    0.0   349314.8291  506486.9612   True
     2      7  534464.1485    0.0   342429.6641  726498.6329   True
     2      8  837959.7667    0.0   497076.1732 1178843.3602   True
```

```
     2      9  490777.7818 0.0258    29689.0898  951866.4738   True
     2     10  416777.9485 0.6071  -234900.9054 1068456.8024  False
     2     11  116777.9485    1.0 -1011501.9439 1245057.8408  False
     3      4  171714.7011    0.0   152902.5077  190526.8946   True
     3      5  328265.8247    0.0   296142.0421  360389.6072   True
     3      6  365729.9279    0.0   289559.3964  441900.4594   True
     3      7  472293.1812    0.0   281234.4905   663351.872   True
     3      8  775788.7994    0.0   435453.9578  1116123.641   True
     3      9  428606.8146 0.0955   -32076.3325  889289.9617  False
     3     10  354606.9812  0.808  -296784.9965  1005998.959  False
     3     11   54606.9812    1.0 -1073507.2397 1182721.2022  False
     4      5  156551.1235    0.0   123490.4906  189611.7565   True
     4      6  194015.2268    0.0   117444.8806   270585.573   True
     4      7  300578.4801    0.0   109360.0413  491796.9189   True
     4      8  604074.0983    0.0   263649.5509  944498.6457   True
     4      9  256892.1135 0.7836  -203857.3087  717641.5356  False
     4     10  182892.2801 0.9982   -468546.571  834331.1313  False
     4     11 -117107.7199    1.0  -1245249.007 1011033.5672  False
     5      6   37464.1032 0.9232   -43412.6962  118340.9027  False
     5      7  144027.3566 0.3649   -48955.8814  337010.5945  False
     5      8  447522.9748 0.0012   106104.0075  788941.9421   True
     5      9  100340.9899 0.9998  -361143.6456  561825.6254  False
     5     10   26341.1566    1.0  -625617.9031  678300.2163  False
     5     11 -273658.8434 0.9995  -1402100.602  854782.9151  False
     6      7  106563.2533 0.8493   -98406.3605  311532.8671  False
     6      8  410058.8715 0.0071    61724.3718  758393.3712   True
     6      9   62876.8867    1.0  -403747.2461  529501.0194  False
     6     10  -11122.9467    1.0   -666730.015  644484.1216  False
     6     11 -311122.9467 0.9985 -1441676.2691  819430.3758  False
     7      8  303495.6182 0.3011   -86425.3958  693416.6321  False
     7      9  -43686.3667    1.0  -542123.4788  454750.7454  False
     7     10    -117686.2    1.0  -796303.9874  560931.5874  False
     7     11    -417686.2 0.9852 -1561737.0242  726364.6242  False
     8      9 -347181.9848 0.6824  -919688.6973  225324.7276  False
     8     10 -421181.8182 0.7528 -1155924.2782  313560.6418  False
     8     11 -721181.8182 0.6698 -1899390.9123  457027.2759  False
     9     10  -73999.8333    1.0  -871651.1608  723651.4941  False
     9     11 -373999.8333 0.9962  -1592432.362  844432.6954  False
    10     11    -300000.0 0.9997 -1602559.1632 1002559.1632  False
-------------------------------------------------------------------

from scipy import stats
import pandas as pd
from scipy.stats import kruskal


# Assuming you have a DataFrame 'df' with a column 'price' and a
column 'floors' indicating the number of floors.

# Create lists to hold prices for each floor group
```

```
floor_groups = []
for floors in df['floors'].unique():
    floor_groups.append(df[df['floors'] == floors]['price'])

# Perform Kruskal-Wallis test
h_statistic, p_value = stats.kruskal(*floor_groups)

# Print the results
print("Kruskal-Wallis H statistic:", h_statistic)
print("p-value:", p_value)

# Check if the difference is statistically significant at alpha = 0.05
alpha = 0.05
if p_value < alpha:
    print("The difference in average price between groups with
different numbers of floors is statistically significant.")
else:
    print("There is no statistically significant difference in average
price between groups with different numbers of floors.")

# Post-hoc analysis (Conover-Iman test)
posthoc = sm.stats.multicomp.pairwise_tukeyhsd(df['price'],
df['floors'])
print("\nPost-Hoc Analysis (Conover-Iman Test):")
print(posthoc)
```

```
Kruskal-Wallis H statistic: 1745.0997758271121
p-value: 0.0
The difference in average price between groups with different numbers
of floors is statistically significant.

Post-Hoc Analysis (Conover-Iman Test):
      Multiple Comparison of Means - Tukey HSD, FWER=0.05
================================================================
group1 group2   meandiff  p-adj     lower        upper    reject
----------------------------------------------------------------
     1      2 199934.2139    0.0  187385.9523 212482.4755   True
     1      3 132297.0776    0.0   95705.2655 168888.8897   True
     2      3 -67637.1363 0.0001 -104650.2224 -30624.0502   True
----------------------------------------------------------------
```

CORRELATION AND MULTICOLLINEARLITY ANALYSIS

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 19163 entries, 1 to 21596
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
```

```
 0   price          19163 non-null  float64
 1   bedrooms       19163 non-null  int64
 2   bathrooms      19163 non-null  int32
 3   sqft_living    19163 non-null  int64
 4   sqft_lot       19163 non-null  int64
 5   floors         19163 non-null  int32
 6   waterfront     19163 non-null  object
 7   view           19163 non-null  object
 8   condition      19163 non-null  object
 9   grade          19163 non-null  object
 10  sqft_above     19163 non-null  int64
 11  yr_built       19163 non-null  int64
 12  sqft_living15  19163 non-null  int64
 13  sqft_lot15     19163 non-null  int64
dtypes: float64(1), int32(2), int64(7), object(4)
memory usage: 2.0+ MB

continuous_variables = df[['price','sqft_living', 'sqft_lot',
'sqft_above', 'sqft_living15', 'sqft_lot15']]
continuous_variables.head()

       price  sqft_living  sqft_lot  sqft_above  sqft_living15
sqft_lot15
1  538000.00         2570      7242        2170           1690
7639
2  180000.00          770     10000         770           2720
8062
3  604000.00         1960      5000        1050           1360
5000
4  510000.00         1680      8080        1680           1800
7503
5 1230000.00         5420    101930        3890           4760
101930

categorical_variables = df[['bedrooms', 'bathrooms','floors',
'waterfront', 'view', 'condition', 'grade']]
categorical_variables.tail()

       bedrooms  bathrooms  floors waterfront  view condition
grade
21591         3          2       2          NO  NONE   Average     8
Good
21592         3          2       3          NO  NONE   Average     8
Good
21593         4          2       2          NO  NONE   Average     8
Good
21594         2          0       2          NO  NONE   Average   7
Average
21596         2          0       2          NO  NONE   Average   7
Average
```

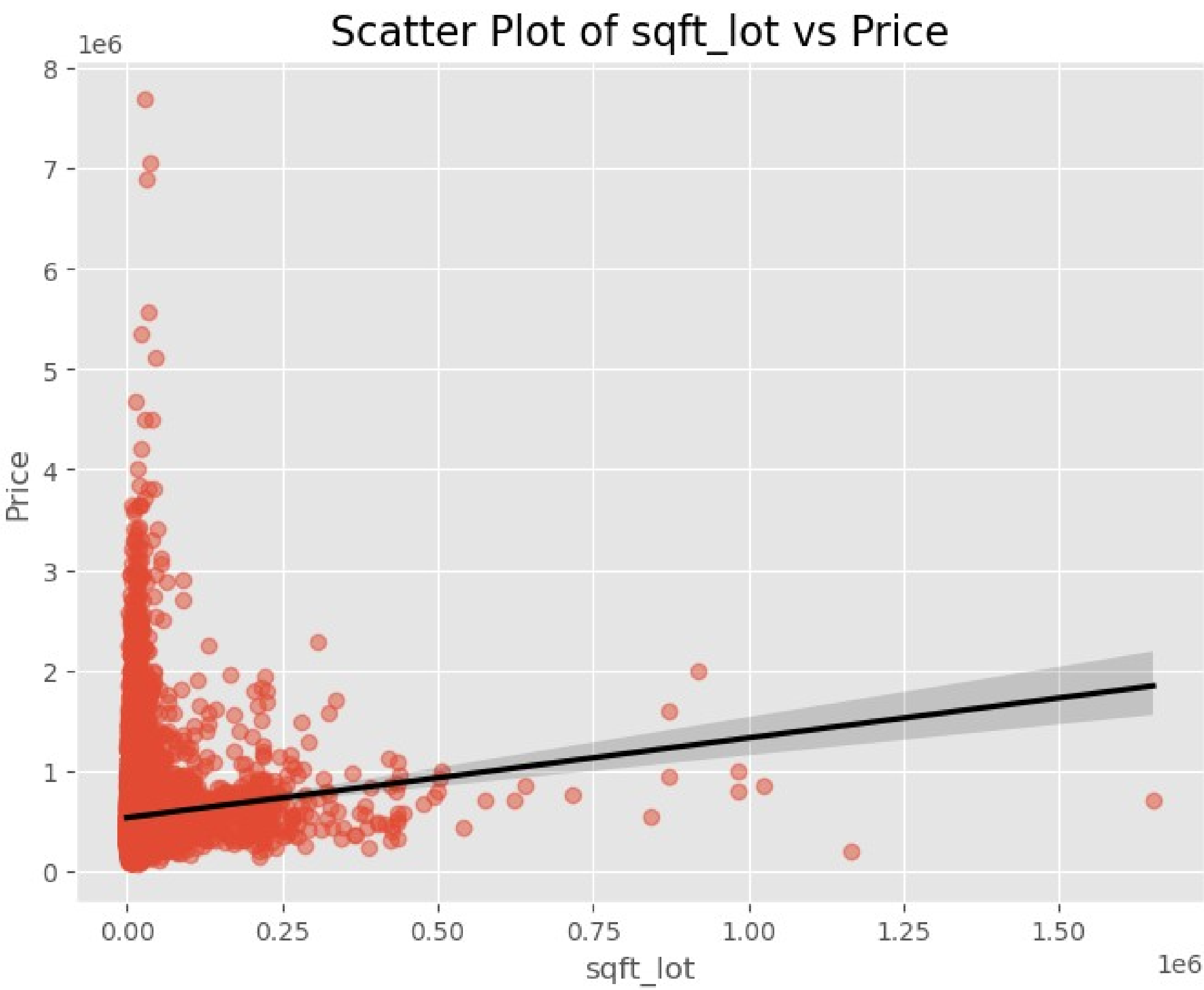
CORRELATION IS ONLY USED FOR CONTINUOUS VARIABLES

```
correlation = continuous_variables.corr()['price']
print(correlation)
```

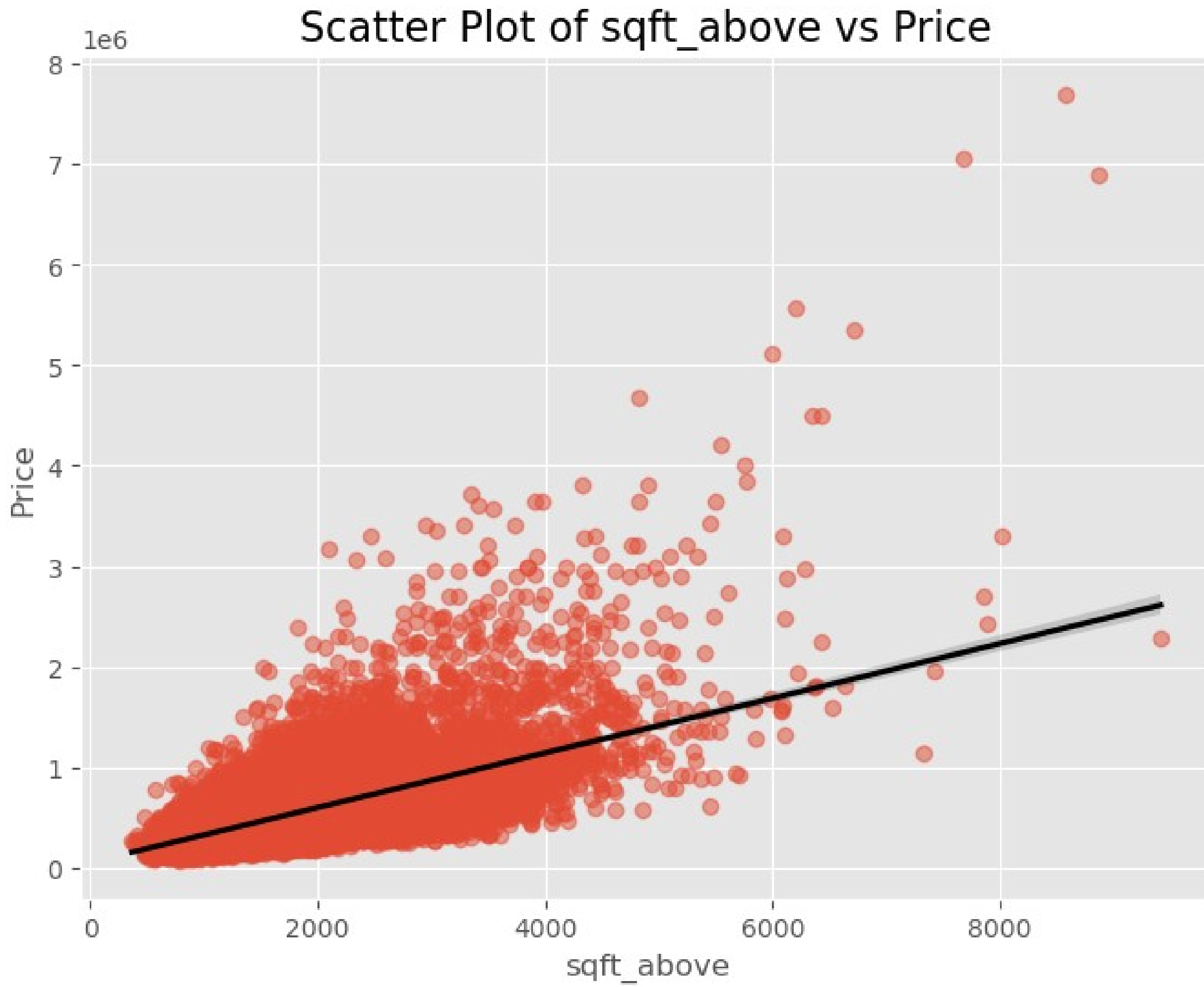
```
price            1.00
sqft_living      0.70
sqft_lot         0.09
sqft_above       0.61
sqft_living15    0.58
sqft_lot15       0.08
Name: price, dtype: float64
```

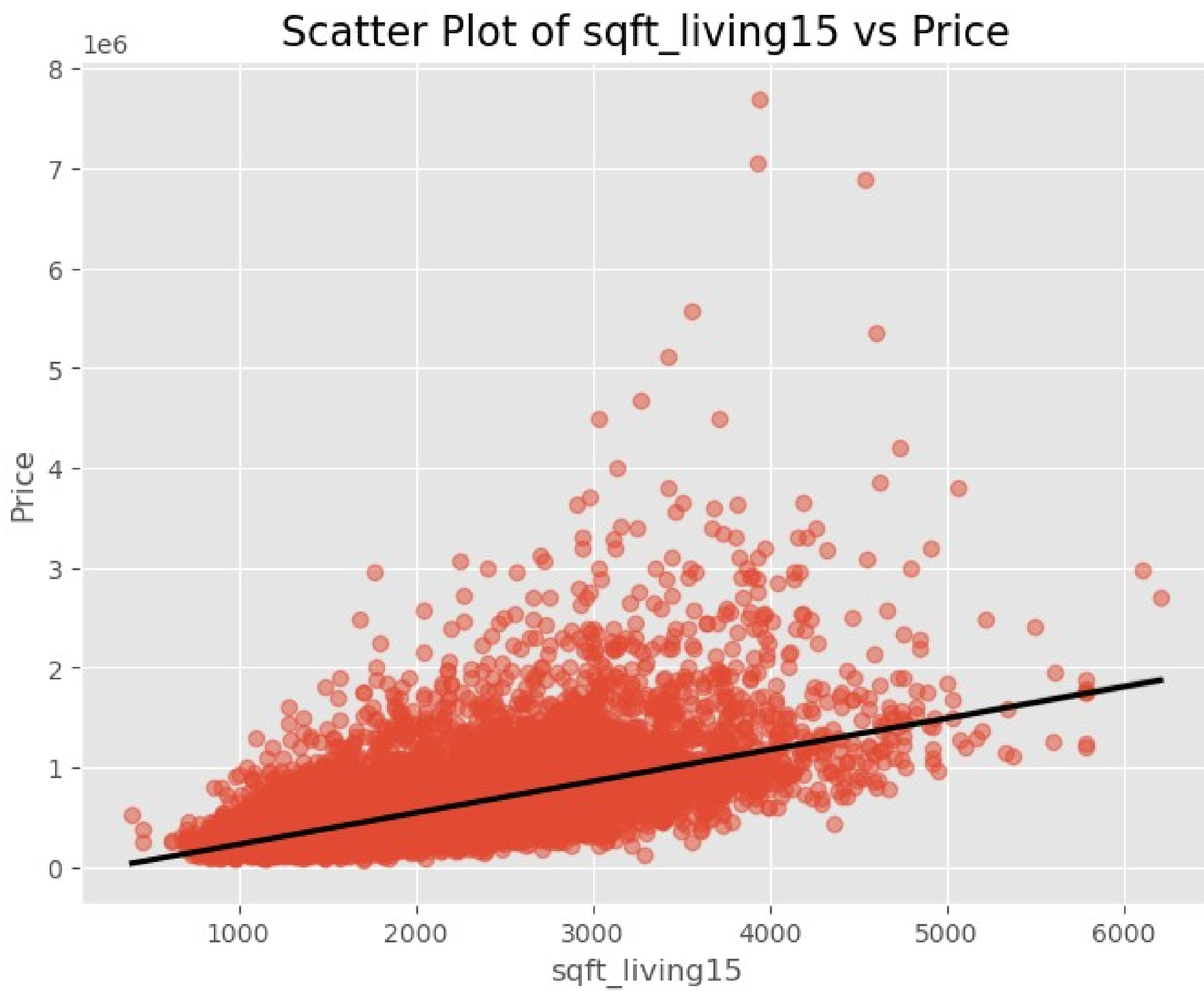
```
# Create scatter plots with black regression lines for each continuous
variable against price
for column in continuous_variables.columns[1:]:  # Exclude 'price'
column
    plt.figure(figsize=(8, 6))
    sns.regplot(x=column, y='price', data=continuous_variables,
scatter_kws={'alpha':0.5}, line_kws={'color':'black'})
    plt.title(f'Scatter Plot of {column} vs Price', fontsize=16)
    plt.xlabel(column, fontsize=12)
    plt.ylabel('Price', fontsize=12)
    plt.grid(True)
    plt.show()
```

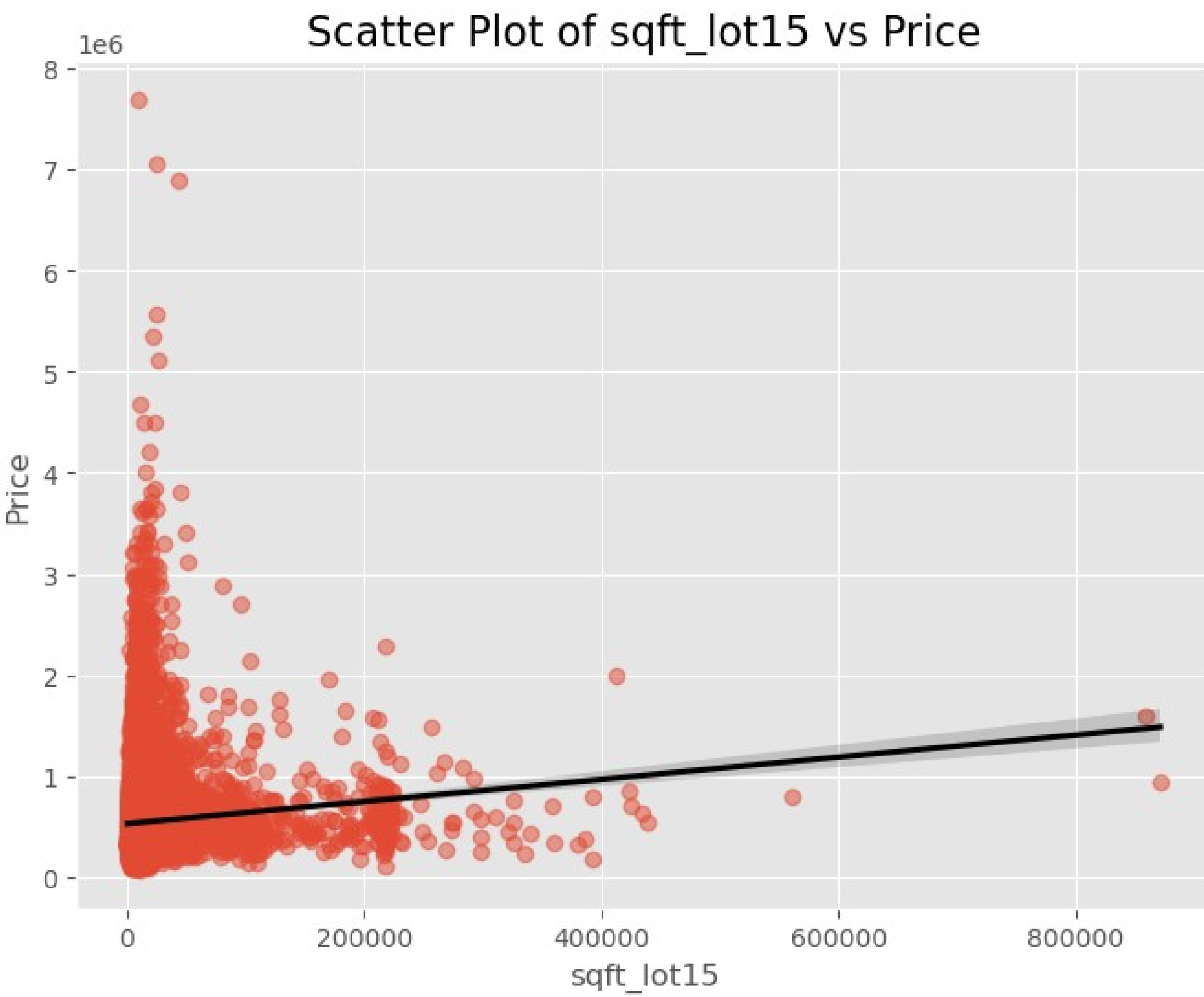

Scatter Plot of sqft_living vs Price
1e6
8
7
6
5
4
3
2
1
0
Price
0
2000
4000
6000
8000
10000
12000
14000
sqft_living



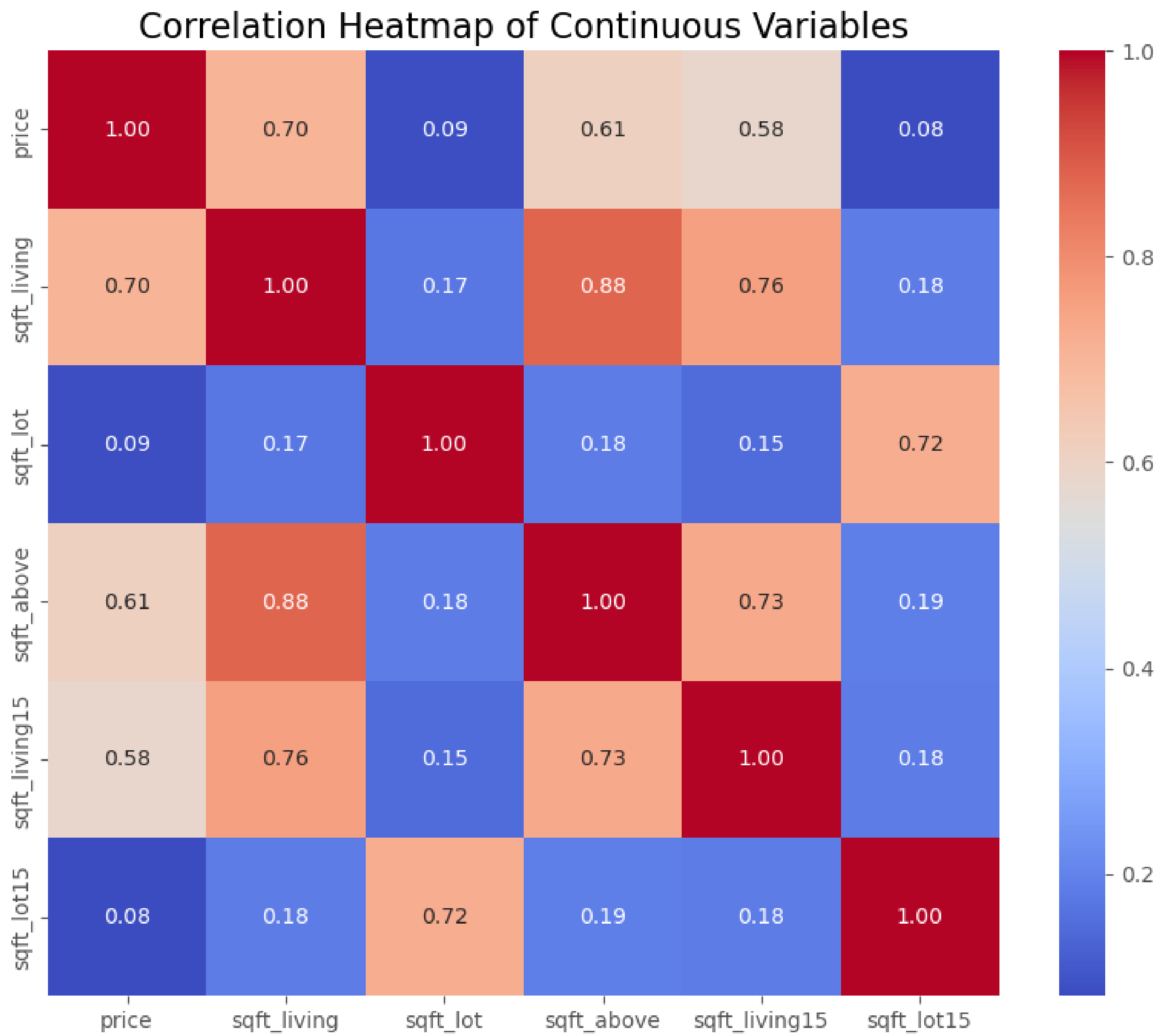
Scatter Plot of sqft_lot vs Price
1e6
8
7
6
5
4
3
2
1
0
Price
0.00
0.25
0.50
0.75
1.00
1.25
1.50
sqft_lot
1e6


Scatter Plot of sqft_above vs Price
1e6
Price
sqft_above
0
1
2
3
4
5
6
7
8
0
2000
4000
6000
8000


Scatter Plot of sqft_living15 vs Price
1e6
8
7
6
5
4
3
2
1
0
Price
1000
2000
3000
4000
5000
6000
sqft_living15




```
# Calculate the correlation matrix
correlation_matrix = continuous_variables.corr()

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f", annot_kws={"size": 10})
plt.title('Correlation Heatmap of Continuous Variables', fontsize=16)
plt.show()
```



INSIGHTS FROM THIS ANALYSIS

THERE'S A RELATIVELY STRONG POSITIVE LINEAR CORRELATION BETWEEN PRICE AND SQFT_LIVING, SQFT_ABOVE AND SQFT_LIVING

THE CORRELATION HEATMAP INDICATES THAT THERE'S A NOTABLE LEVEL OF MULTICOLLINEARLITY BETWEEN sqft_living & sqft_living15, sqft_living & sqft_above, sqft_lot & sqft_lot15

REGRESSION ANALYSIS

Encoding Categorical Variables

Before performing any regression we need to encode the categorical variables.

```
label_encoder = LabelEncoder()
df['grade'] = label_encoder.fit_transform(df['grade'])
df['grade'].value_counts()
```

```
grade
8     7948
9     5398
10    2311
7     1804
0     1014
1      356
6      212
2       81
5       25
3       13
4        1
Name: count, dtype: int64
```

```
df.head()
```

```
       price  bedrooms  bathrooms  sqft_living  sqft_lot  floors
waterfront  \
1  538000.00         3          2         2570      7242       2
NO
2  180000.00         2          1          770     10000       1
NO
3  604000.00         4          3         1960      5000       1
NO
4  510000.00         3          2         1680      8080       1
NO
5 1230000.00         4          4         5420    101930       1
NO

   view  condition  grade  sqft_above  yr_built  sqft_living15
sqft_lot15
1  NONE    Average      8        2170      1951           1690
7639
2  NONE    Average      7         770      1933           2720
8062
3  NONE  Very Good      8        1050      1965           1360
5000
4  NONE    Average      9        1680      1987           1800
7503
5  NONE    Average      1        3890      2001           4760
101930
```

```
#View Column encoding
df['view'].replace(to_replace=
['NONE','FAIR','AVERAGE','GOOD','EXCELLENT'], value = [0,1,2,3,4,],
inplace= True )
```

```
#Waterfront Column encoding
df['waterfront'].replace(to_replace= ['NO','YES'], value = [0,1],
inplace= True )

#Condition Column encoding
df['condition'].replace(to_replace=
['Poor','Fair','Average','Good','Very Good'], value = [0,1,2,3,4,],
inplace= True )

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 19163 entries, 1 to 21596
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   price          19163 non-null  float64
 1   bedrooms       19163 non-null  int64
 2   bathrooms      19163 non-null  int32
 3   sqft_living    19163 non-null  int64
 4   sqft_lot       19163 non-null  int64
 5   floors         19163 non-null  int32
 6   waterfront     19163 non-null  int64
 7   view           19163 non-null  int64
 8   condition      19163 non-null  int64
 9   grade          19163 non-null  int32
 10  sqft_above     19163 non-null  int64
 11  yr_built       19163 non-null  int64
 12  sqft_living15  19163 non-null  int64
 13  sqft_lot15     19163 non-null  int64
dtypes: float64(1), int32(3), int64(10)
memory usage: 2.0 MB
```

```
df.head()
```

```
       price  bedrooms  bathrooms  sqft_living  sqft_lot  floors
waterfront  \
1  538000.00         3          2         2570      7242       2
0
2  180000.00         2          1          770     10000       1
0
3  604000.00         4          3         1960      5000       1
0
4  510000.00         3          2         1680      8080       1
0
5 1230000.00         4          4         5420    101930       1
0

   view  condition  grade  sqft_above  yr_built  sqft_living15
sqft_lot15
```

```
1      0           2       8          2170       1951              1690
7639
2      0           2       7           770       1933              2720
8062
3      0           4       8          1050       1965              1360
5000
4      0           2       9          1680       1987              1800
7503
5      0           2       1          3890       2001              4760
101930
```

ONE HOT ENCODING OF CATEGORICAL VARIABLES, FEATURE SELECTION, CONCATENATION, and SPLITTING THE DATA INTO TRAINING AND TESTING DATA SETS

```
import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import train_test_split


# List of categorical variables to be one-hot encoded
categorical_vars = ['bedrooms', 'bathrooms', 'floors', 'waterfront',
'view', 'condition', 'grade']

# One-hot encode categorical variables
encoded_categorical = pd.get_dummies(df[categorical_vars],
drop_first=True)

# Select continuous variables. sqft_above, sqft_living15 and
sqft_lot15 have not been selected due to the multicollinearlity
observed prior
continuous_variables = df[['sqft_living', 'sqft_lot']]

# Concatenate one-hot encoded categorical variables with continuous
variables
X = pd.concat([encoded_categorical, continuous_variables], axis=1)

# Define the dependent variable
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)

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

# Fit the linear regression model
model = sm.OLS(y_train, X_train).fit()
```

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

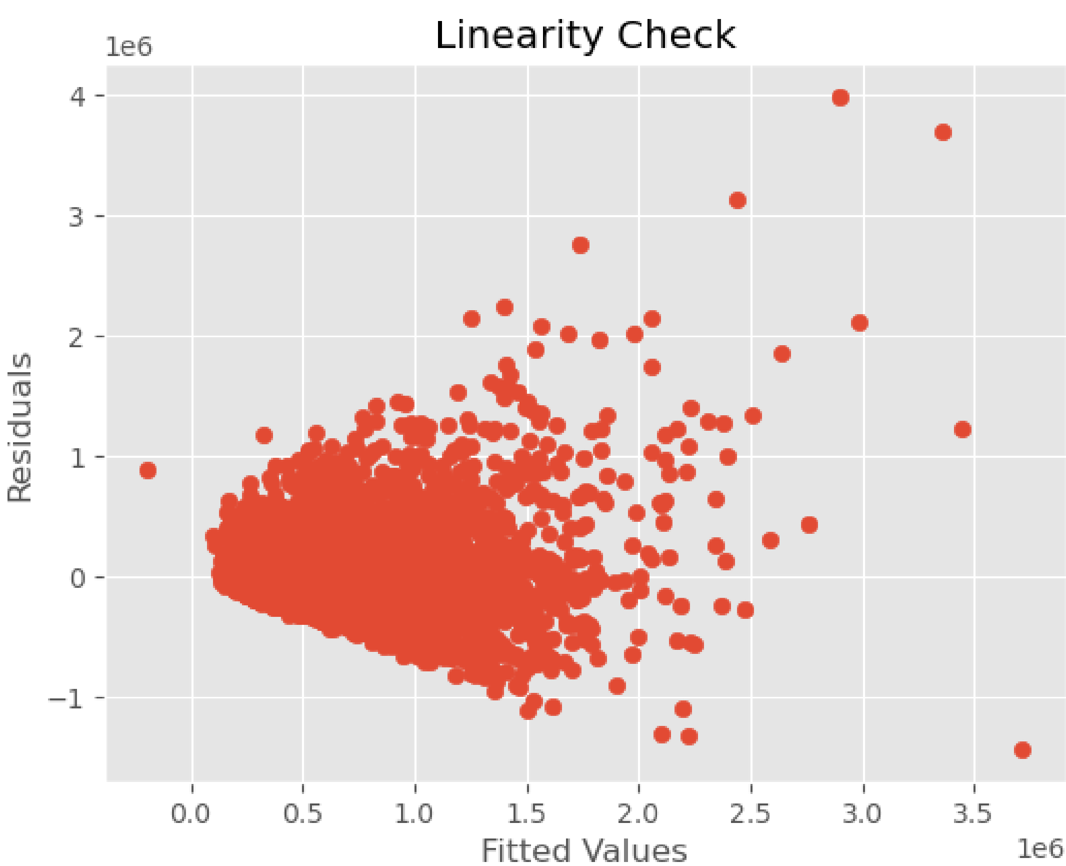
```
                                OLS Regression Results                            
==============================================================================
=========
Dep. Variable:                  price   R-squared:                       
0.583
Model:                            OLS   Adj. R-squared:                  
0.583
Method:                 Least Squares   F-statistic:                     
2384.
Date:                Tue, 09 Apr 2024   Prob (F-statistic):              
0.00
Time:                        21:27:34   Log-Likelihood:             -
2.1134e+05
No. Observations:               15330   AIC:                         
4.227e+05
Df Residuals:                   15320   BIC:                         
4.228e+05
Df Model:                           9                                         

Covariance Type:            nonrobust                                         

==============================================================================
=========
                  coef    std err          t      P>|t|      [0.025      
0.975]
------------------------------------------------------------------------------
---------
const        1.449e+05   1.38e+04     10.487      0.000    1.18e+05    
1.72e+05
bedrooms    -5.076e+04   2723.539    -18.638      0.000   -5.61e+04   
-4.54e+04
bathrooms    3.813e+04   3930.324      9.701      0.000    3.04e+04    
4.58e+04
floors       3791.4612   4125.147      0.919      0.358   -4294.318    
1.19e+04
waterfront   5.779e+05   2.42e+04     23.922      0.000    5.31e+05    
6.25e+05
view         6.442e+04   2844.741     22.647      0.000    5.88e+04    
7e+04
condition    4.683e+04   3066.459     15.270      0.000    4.08e+04    
5.28e+04
grade       -2.039e+04    891.538    -22.866      0.000   -2.21e+04   
-1.86e+04
sqft_living   253.3238      3.490     72.587      0.000     246.483     
260.165
sqft_lot       -0.4224      0.049     -8.622      0.000      -0.518     
```

```
-0.326
==============================================================================
========
Omnibus:                     9175.398   Durbin-Watson:
2.014
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
304820.215
Skew:                           2.325   Prob(JB):
0.00
Kurtosis:                      24.345   Cond. No.
5.40e+05
==============================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 5.4e+05. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

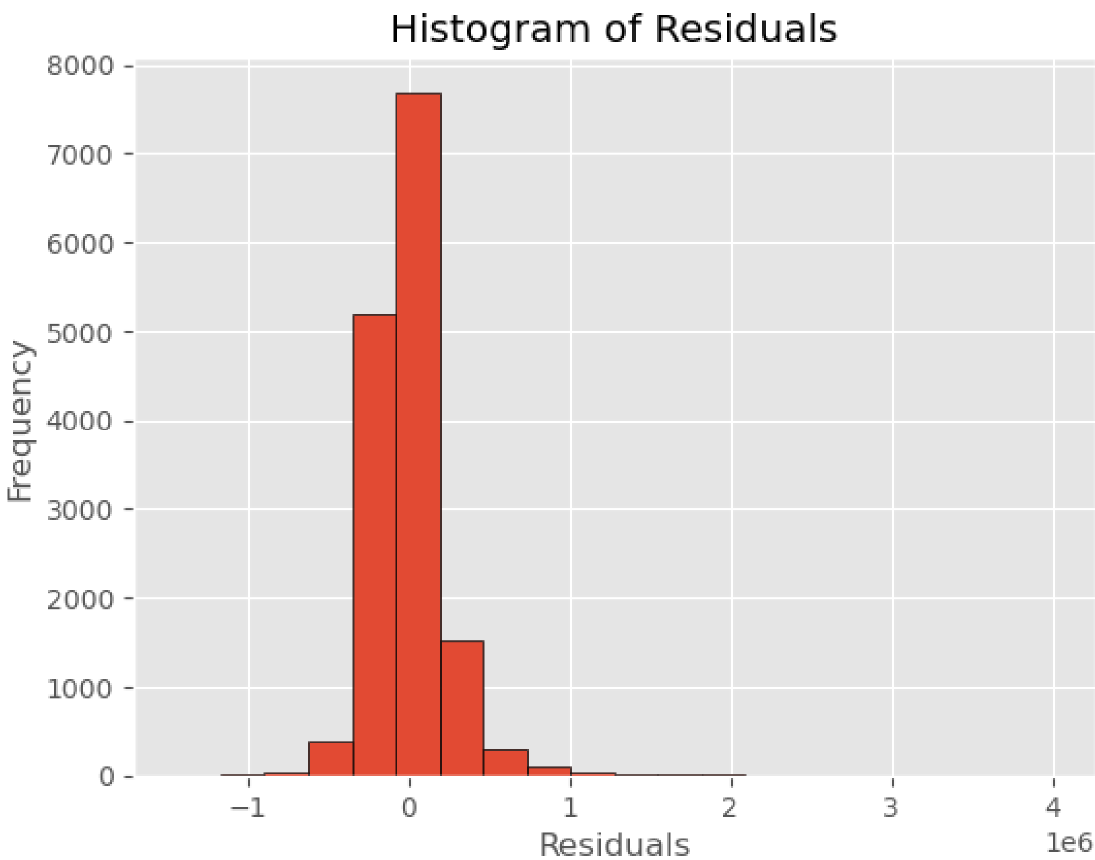
EVALUATE THE ASSUMPTIONS OF LINEAR REGRESSION

```
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.stats.diagnostic import linear_rainbow,
het_breuschpagan
from scipy.stats import shapiro

# Our model is a fitted linear regression model

# Residuals
residuals = model.resid

# Linearity
plt.scatter(model.fittedvalues, residuals)
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Linearity Check')
plt.show()

# Rainbow test for linearity
rainbow_statistic, rainbow_p_value = linear_rainbow(model)
print("Rainbow Test p-value:", rainbow_p_value)

# Independence
dw_test_statistic = sm.stats.stattools.durbin_watson(residuals)
print("Durbin-Watson test statistic:", dw_test_statistic)
```

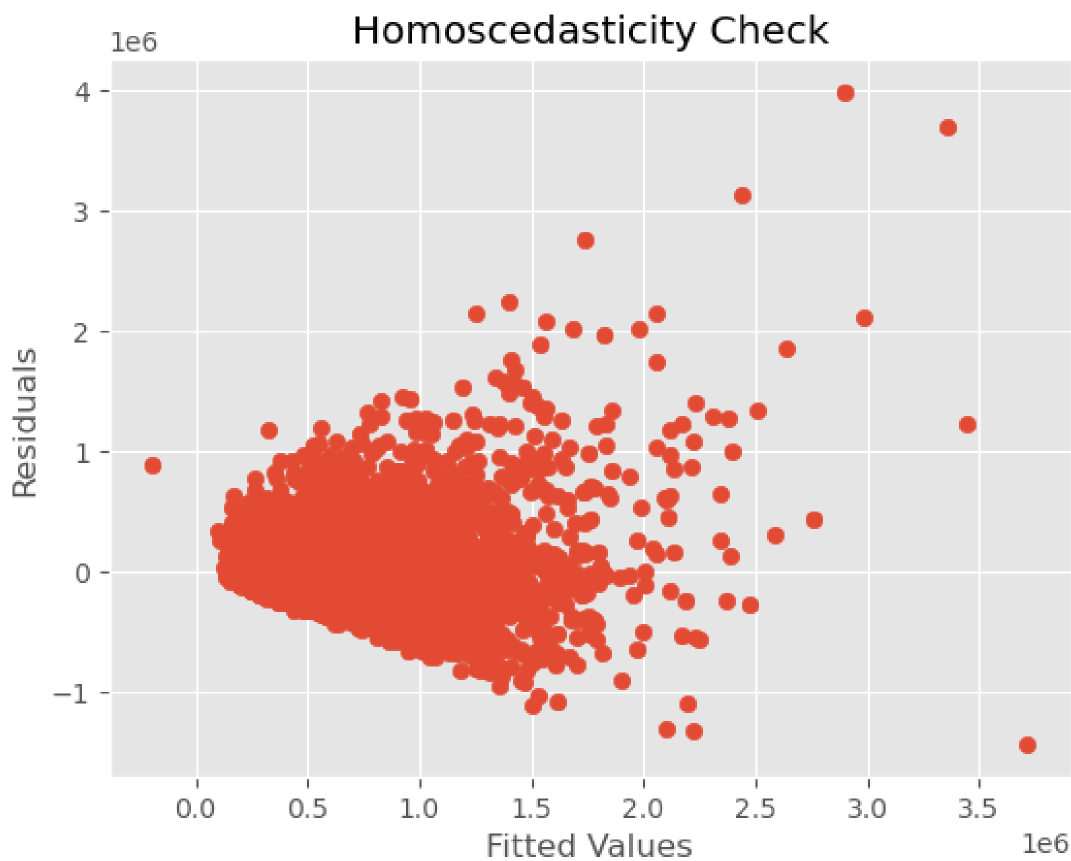
```python
# Normality
plt.hist(residuals, bins=20, edgecolor='k')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Histogram of Residuals')
plt.show()

shapiro_statistic, shapiro_p_value = shapiro(residuals)
print("Shapiro-Wilk Test p-value:", shapiro_p_value)

# Homoscedasticity
plt.scatter(model.fittedvalues, residuals)
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Homoscedasticity Check')
plt.show()

bp_test_statistic, bp_p_value, _, _ = het_breuschpagan(residuals,
model.model.exog)
print("Breusch-Pagan Test p-value:", bp_p_value)
```



```
Rainbow Test p-value: 0.9969011115163554
Durbin-Watson test statistic: 2.013617363633505
```




```
C:\Users\Kola500\.anaconda\ANACONDA\Lib\site-packages\scipy\stats\
_morestats.py:1882: UserWarning: p-value may not be accurate for N >
5000.
  warnings.warn("p-value may not be accurate for N > 5000.")

Shapiro-Wilk Test p-value: 0.0
```




```
Breusch-Pagan Test p-value: 0.0
```

INTERPRETATION

Based on these results:

The relationship between the independent variables and the dependent variable appears to be linear.

There is no significant autocorrelation in the residuals. The residuals are not normally distributed.

There is evidence of heteroscedasticity in the residuals.

Course of action:

Linear regression is relatively robust to violations of normality assumptions and heteroscedasticity, especially with large sample sizes which we have in this case. Linear regression coefficient estimates are also usually still unbiased in the presence of heteroscedasticity. So we'll still proceed with linear regression

Simple Linear Regression Model

```
import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import train_test_split


# Select feature and target variable
feature = 'sqft_living'
target = 'price'

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[[feature]],
df[target], test_size=0.2, random_state=42)

# Add a constant term to the predictor variable
X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test)

# Fit the simple linear regression model
model_simple = sm.OLS(y_train, X_train).fit()

# Print model summary
print("Simple Linear Regression Model Summary:")
print(model_simple.summary())
```

```
Simple Linear Regression Model Summary:
                            OLS Regression Results                            

======================================================================
========
Dep. Variable:                  price   R-squared:                       
0.490
Model:                            OLS   Adj. R-squared:                  
0.489
Method:                 Least Squares   F-statistic:                 
1.470e+04
Date:                Tue, 09 Apr 2024   Prob (F-statistic):              
0.00
Time:                        21:28:11   Log-Likelihood:            -
2.1290e+05
No. Observations:               15330   AIC:                         
4.258e+05
Df Residuals:                   15328   BIC:                         
4.258e+05
Df Model:                           1                                         

Covariance Type:            nonrobust                                         

======================================================================
```

```
=========
                  coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
---------
const       -3.866e+04   5204.447     -7.428      0.000   -4.89e+04
-2.85e+04
sqft_living   277.5570      2.289    121.238      0.000     273.070
282.044
======================================================================
========
Omnibus:                    10428.487   Durbin-Watson:
2.006
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
367419.562
Skew:                           2.798   Prob(JB):
0.00
Kurtosis:                      26.322   Cond. No.
5.63e+03
======================================================================
========

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

In this model, R-squared is 0.490, indicating that approximately 49% of the variance in house prices can be explained by the square footage of living space.

The adjusted R-squared takes into account the number of predictors in the model. It penalizes the addition of unnecessary predictors. In this model, the adjusted R-squared is 0.489.

The F-statistic tests the overall significance of the regression model. A higher F-statistic and a lower p-value indicate a better fit of the model to the data. Here, the F-statistic is 1.470e+04 with a p-value close to 0, suggesting that the model is statistically significant.

Coefficients: The coefficient of sqft_living is 277.5570. This indicates that, on average, for each additional square foot of living space, the house price increases by $277.5570.

Multiple Linear Regression Model with Two Features

```
import pandas as pd
import statsmodels.api as sm
```

```python
from sklearn.model_selection import train_test_split


# Select features and target variable
features = ['sqft_living', 'bedrooms']
target = 'price'

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[features],
df[target], test_size=0.2, random_state=42)

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

# Fit the multiple linear regression model
model_2 = sm.OLS(y_train, X_train).fit()

# Print model summary
print("Multiple Linear Regression Model with Two Features Summary:")
print(model_2.summary())
```

```
Multiple Linear Regression Model with Two Features Summary:
                            OLS Regression Results

======================================================================
========
Dep. Variable:                  price   R-squared:
0.507
Model:                            OLS   Adj. R-squared:
0.507
Method:                 Least Squares   F-statistic:
7886.
Date:                Tue, 09 Apr 2024   Prob (F-statistic):
0.00
Time:                        21:28:23   Log-Likelihood:             -
2.1263e+05
No. Observations:               15330   AIC:
4.253e+05
Df Residuals:                   15327   BIC:
4.253e+05
Df Model:                           2

Covariance Type:            nonrobust

======================================================================
=========
                  coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
```

```
---------
const         1.049e+05   7984.484     13.138      0.000    8.92e+04
1.21e+05
sqft_living   316.4133      2.796    113.184      0.000     310.934
321.893
bedrooms     -6.654e+04   2842.303    -23.411      0.000   -7.21e+04
-6.1e+04
======================================================================
========
Omnibus:                    10003.377   Durbin-Watson:
2.005
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
319712.432
Skew:                           2.660   Prob(JB):
0.00
Kurtosis:                      24.731   Cond. No.
9.13e+03
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 9.13e+03. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

In this model, R-squared is 0.507, indicating that approximately 51% of the variance in house prices can be explained by square footage of living space and the number of bedrooms.

Here, the F-statistic is 7886 with a p-value close to 0, suggesting that the model is statistically significant.

The coefficient of sqft_living is 316.4133. This indicates that, on average, for each additional square foot of living space, the house price increases by $316.4133, holding the number of bedrooms constant.

Multiple Linear Regression Model with Many Features

```python
import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import train_test_split


# Select all available features and target variable
features = ['sqft_living', 'bedrooms', 'bathrooms', 'floors',
```

```python
'waterfront', 'view', 'condition', 'grade']
target = 'price'

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[features],
df[target], test_size=0.2, random_state=42)

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

# Fit the multiple linear regression model
model_all_features = sm.OLS(y_train, X_train).fit()

# Print model summary
print("Multiple Linear Regression Model with All Features Summary:")
print(model_all_features.summary())
```

```
Multiple Linear Regression Model with All Features Summary:
                            OLS Regression Results

======================================================================
========
Dep. Variable:                  price   R-squared:
0.581
Model:                            OLS   Adj. R-squared:
0.581
Method:                 Least Squares   F-statistic:
2660.
Date:                Tue, 09 Apr 2024   Prob (F-statistic):
0.00
Time:                        21:28:34   Log-Likelihood:             -
2.1138e+05
No. Observations:               15330   AIC:
4.228e+05
Df Residuals:                   15321   BIC:
4.228e+05
Df Model:                           8

Covariance Type:            nonrobust

======================================================================
=========
                  coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
---------
const        1.352e+05   1.38e+04      9.789      0.000    1.08e+05
1.62e+05
sqft_living   248.2605      3.448     71.993      0.000     241.501
```

```
255.020
bedrooms     -4.862e+04   2718.716    -17.885      0.000    -5.4e+04
-4.33e+04
bathrooms     3.859e+04   3939.356      9.795      0.000    3.09e+04
4.63e+04
floors        6398.6927   4123.882      1.552      0.121   -1684.607
1.45e+04
waterfront    5.798e+05   2.42e+04     23.942      0.000    5.32e+05
6.27e+05
view          6.421e+04   2851.435     22.519      0.000    5.86e+04
6.98e+04
condition     4.739e+04   3073.080     15.422      0.000    4.14e+04
5.34e+04
grade        -2.027e+04    893.573    -22.688      0.000    -2.2e+04
-1.85e+04
======================================================================
========
Omnibus:                     9245.280   Durbin-Watson:
2.015
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
313571.610
Skew:                           2.344   Prob(JB):
0.00
Kurtosis:                      24.655   Cond. No.
2.90e+04
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 2.9e+04. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

In this model, R-squared is 0.581, indicating that approximately 58.1% of the variance in house prices can be explained by these features.

Here, the F-statistic is 2660 with a p-value close to 0, suggesting that the model is statistically significant.

- **Intercept (const)**: The intercept term represents the estimated house price when all other predictor variables are zero. However, this interpretation is not meaningful in practice, as having zero square footage, bedrooms, bathrooms, etc., is unrealistic.
- **sqft_living**: The coefficient of sqft_living is 248.2605. This means that for every additional square foot of living space, holding all other variables constant, the house price is expected to increase by $248.2605.

- **bedrooms**: The coefficient of bedrooms is -48,620. This indicates that for each additional bedroom, holding all other variables constant, the house price is expected to decrease by $48,620. This negative coefficient suggests that, on average, houses with more bedrooms tend to have lower prices, which could be due to factors like smaller lot sizes or less desirable locations.
- **bathrooms**: The coefficient of bathrooms is 38,590. This means that for each additional bathroom, holding all other variables constant, the house price is expected to increase by $38,590. This positive coefficient suggests that houses with more bathrooms tend to have higher prices, which could be due to increased convenience and luxury.
- **floors**: The coefficient of floors is 6,398.6927. This indicates that for each additional floor, holding all other variables constant, the house price is expected to increase by $6,398.6927. This positive coefficient suggests that houses with more floors tend to have higher prices, which could be due to larger floor area or better views from higher floors.
- **waterfront**: The coefficient of waterfront is 579,800. This means that if a house has waterfront property (i.e., waterfront = 1), holding all other variables constant, the house price is expected to increase by $579,800 compared to a house without waterfront property.
- **view**: The coefficient of view is 64,210. This indicates that for each additional level of view rating, holding all other variables constant, the house price is expected to increase by $64,210. This positive coefficient suggests that houses with better views tend to have higher prices, which could be due to increased desirability.
- **condition**: The coefficient of condition is 47,390. This means that for each additional level of condition rating, holding all other variables constant, the house price is expected to increase by $47,390. This positive coefficient suggests that houses in better condition tend to have higher prices, which could be due to better maintenance and aesthetics.
- **grade**: The coefficient of grade is -20,270. This indicates that for each additional level of grade rating, holding all other variables constant, the house price is expected to decrease by $20,270. This negative coefficient suggests that houses with higher grade ratings tend to have lower prices, which may seem counterintuitive but could be due to other factors not captured in the model.

SUMMARY INSIGHTS

descriptive statistics

the prices of houses in kings county range from **78000 to 7700000** with a median price of 450000.

the house prices in kings county are right skewed, which means that there are **more houses with lower prices.**

descriptive analysis of the average prices of houses revealed that on average houses were priced higher:

- with increase in the number of bathrooms
- with presence of the waterfront
- with a better view
- with improvement in the condition on the house
- with a higher grade of the house
- with more sqft_living
- with more sqft_above

the average price of houses increases upto 8 bedrooms, after which it decreases.

from the time series, **the year a house was built did not appear to have an influence on the price of the houses**.

on average houses with 2 floors have a higher price than those with 3 floors, so having more floors may not necessarily translate to a higher price of the house.

mapping out the various houses in kings county and prices revealed that the **best priced houses are located in the central region of kings county**.

inferential statistics

majority of the differences observed in the means were statistically significant after utilising statistical tests.

correlation analysis revealed there's a relatively strong positive linear correlation between price and sqft_living, sqft_above and sqft_living regression analysis factoring in both categorical and numerical variables revealed that:

that approximately 58.1% of the variance in house prices can be explained by 'sqft_living', 'bedrooms', 'bathrooms', 'floors', 'waterfront', 'view', 'condition', 'grade']

additionally

for every additional square foot of living space, holding all other variables constant, the house price is expected to increase by $248.2605.

for each additional bedroom, holding all other variables constant, the house price is expected to decrease by $48,620. this negative coefficient suggests that, on average, houses with more bedrooms tend to have lower prices, which could be due to factors like smaller lot sizes or less desirable locations.

for each additional bathroom, holding all other variables constant, the house price is expected to increase by $38,590. this positive coefficient suggests that houses with more bathrooms tend to have higher prices, which could be due to increased convenience and luxury.

for each additional floor, holding all other variables constant, the house price is expected to increase by $6,398.6927. this positive coefficient suggests that houses with more floors tend to have higher prices, which could be due to larger floor area or better views from higher floors.

if a house has waterfront property (i.e., waterfront = 1), holding all other variables constant, the house price is expected to increase by $579,800 compared to a house without waterfront property.

for each additional level of view rating, holding all other variables constant, the house price is expected to increase by $64,210. this positive coefficient suggests that houses with better views tend to have higher prices, which could be due to increased desirability.

for each additional level of condition rating, holding all other variables constant, the house price is expected to increase by $47,390. this positive coefficient suggests that houses in better condition tend to have higher prices, which could be due to better maintenance and aesthetics.

for each additional level of grade rating, holding all other variables constant, the house price is expected to decrease by $20,270. this negative coefficient suggests that houses with higher grade ratings tend to have lower prices, which may seem counterintuitive but could be due to other factors not captured in the model.

In conclusion, there are various factors that appear to influence the pricing of houses and therefore the agency should factor them into their pricing model when coming up with prices for their listings.