Final Project Submission

Please fill out:

- Group 2 Student names: Yvonne Mwangi, Frida Oyucho, Charles Egambi, Myles Kiganane , Sonia Ojay, Thomas Otiende
- Student pace: self paced / part time / full time: Part time
- Scheduled project review date/time: 7th April 2024
- Instructor name: William Okomba, Noah Kandie, Samuel G Mwangi
- Blog post URL:

Introduction

The global real estate market has experienced significant shifts and challenges in recent years, with areas of rapid population growth and economic fluctuations at the forefront of these changes. This global trend has particular resonance in King County, Washington DC, where the housing market is undergoing substantial challenges.

King County is characterized by its diverse neighbourhoods', fluctuating property values, and a dynamic real estate market. Real estate plays a pivotal role in the county's economic growth and community development, involving a broad spectrum of stakeholders such as real estate agents, property owners, homebuyers, investors, regulatory bodies, and the local community at large.

A central challenge within King County's real estate sector is the accurate prediction of house prices. The county's housing market is highly dynamic, influenced by factors like population growth, economic fluctuations, and changing buyer preferences. This volatility makes it challenging for both real estate agents and homeowners to set competitive prices that truly reflect the value of their properties and meet market demand. Without precise price predictions, stakeholders in King County may face difficulties in selling properties efficiently, maximizing returns on investments, and maintaining competitiveness in the market. Addressing this issue requires the development of robust predictive models and leveraging data-driven insights to guide pricing decisions effectively within King County's real estate market.

This research aims to explore the global trends impacting the real estate market and delve into the specific challenges and opportunities within King County, ultimately proposing innovative solutions to foster a more transparent, efficient, and competitive housing market in the county.

Business Understanding:

King County, located in Washington DC, faces significant housing challenges due to its rapidly increasing population. The area is characterized by diverse neighborhoods, varying property values, and fluctuating market dynamics. Real estate is a crucial sector in King County, influencing both economic growth and community development. Key stakeholders include real estate agents, property owners, homebuyers, investors, regulatory bodies, and the local community. Understanding the intricacies of the housing market, including factors affecting property prices, buyer preferences, and market trends, is essential for making informed decisions and strategies in the real estate sector of King County

Business Problem: One of the primary challenges facing stakeholders in King County's real estate market is accurately predicting house prices. The dynamic nature of the housing market, coupled with factors such as population growth, economic fluctuations, and changing buyer preferences, makes it difficult to determine optimal pricing strategies for properties. Real estate agents and homeowners often struggle to set competitive prices that reflect the true value of their properties and meet market demand. Without accurate price predictions, stakeholders may encounter difficulties in selling properties efficiently, maximizing returns on investments, and maintaining competitiveness in the market. Addressing this business problem requires developing robust predictive models and leveraging data-driven insights to guide pricing decisions effectively in King County's real estate market.

Stakeholders: Real estate agents, Property owners, Homebuyers, Investors and Regulatory bodies

Research Questions

- What are the key factors influencing house prices in King County, Washington DC?
- How do factors such as the number of bedrooms, bathrooms, and overall grade of the property influence house prices in King County?
- How accurate is the price prediction when a single feature is considered as compared to multiple housing features?

Data Cleaning

In [122]:
```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

In [123]:
```
# Loading data
df = pd.read_csv('kc_house_data.csv')
df
```

Out[123]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfron
0	7129300520	10/13/2014	221900.00000	3	1.00000	1180	5650	1.00000	Nal
1	6414100192	12/9/2014	538000.00000	3	2.25000	2570	7242	2.00000	NC
2	5631500400	2/25/2015	180000.00000	2	1.00000	770	10000	1.00000	NC
3	2487200875	12/9/2014	604000.00000	4	3.00000	1960	5000	1.00000	NC
4	1954400510	2/18/2015	510000.00000	3	2.00000	1680	8080	1.00000	NC
...	...	...	...	...	...	...	...	...	.
21592	263000018	5/21/2014	360000.00000	3	2.50000	1530	1131	3.00000	NC
21593	6600060120	2/23/2015	400000.00000	4	2.50000	2310	5813	2.00000	NC
21594	1523300141	6/23/2014	402101.00000	2	0.75000	1020	1350	2.00000	NC
21595	291310100	1/16/2015	400000.00000	3	2.50000	1600	2388	2.00000	Nal
21596	1523300157	10/15/2014	325000.00000	2	0.75000	1020	1076	2.00000	NC

21597 rows × 21 columns

In [124]:
```
# Checking the shape of the dataset
df.shape
```

Out[124]:
```
(21597, 21)
```

In [125]:
```
# Checking available columns in the dataset
df.columns
```

Out[125]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

In [126]:
```
# Checking the structure of the dataset(missingness & datatype)
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
```

Identifying Missing Data

In [127]:
```
# Identifying missingness
df.isnull().sum()
```

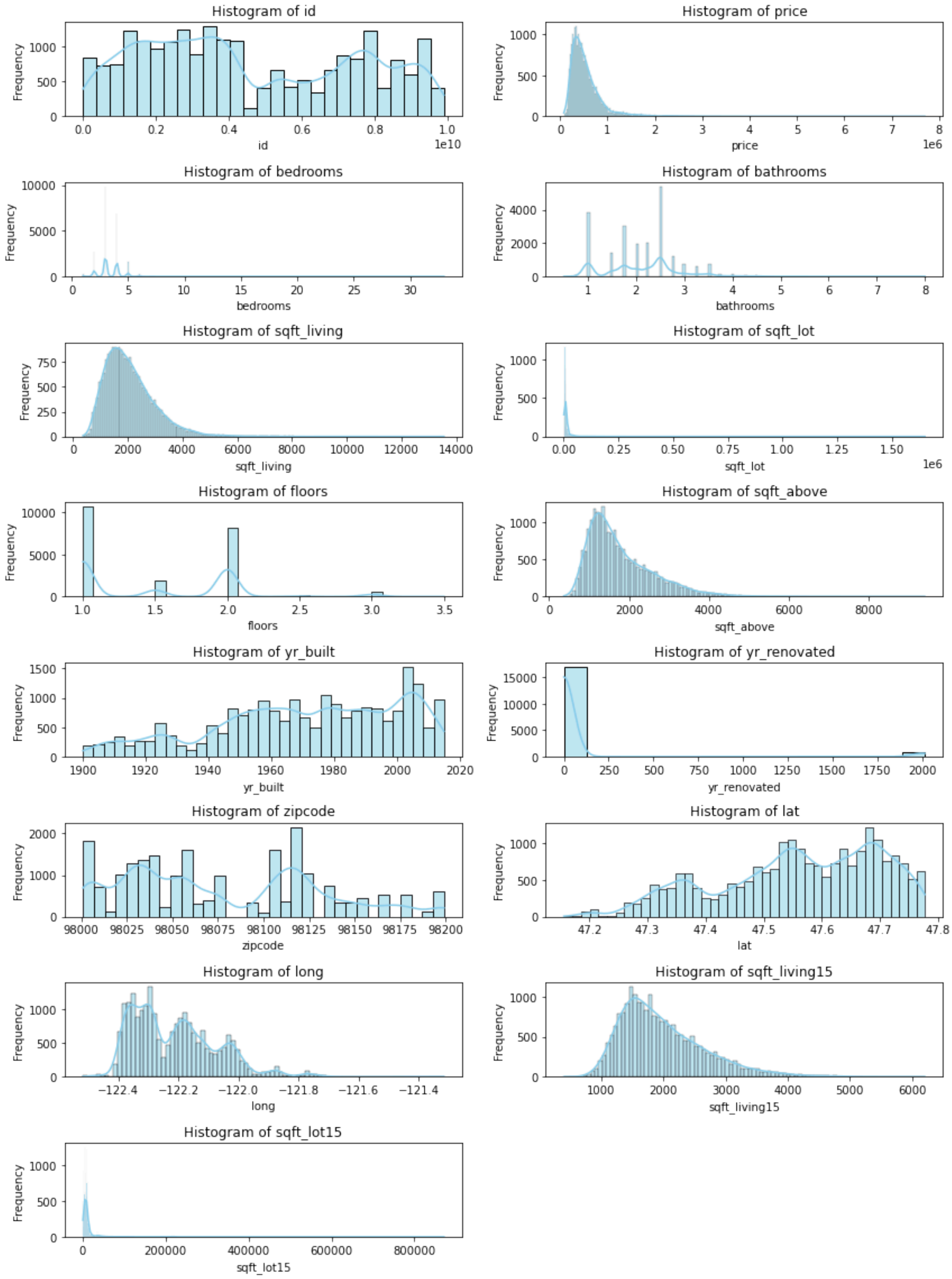
Out[127]:
```
id                  0
date                0
price               0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront       2376
view               63
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_built            0
yr_renovated     3842
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
dtype: int64
```

In [128]:
```
# Handling missingness
df.isnull().mean()
```

Out[128]:
```
id               0.00000
date             0.00000
price            0.00000
bedrooms         0.00000
bathrooms        0.00000
sqft_living      0.00000
sqft_lot         0.00000
floors           0.00000
waterfront       0.11002
view             0.00292
condition        0.00000
grade            0.00000
sqft_above       0.00000
sqft_basement    0.00000
yr_built         0.00000
yr_renovated     0.17790
zipcode          0.00000
lat              0.00000
long             0.00000
sqft_living15    0.00000
sqft_lot15       0.00000
dtype: float64
```

In [129]:

```
# Visualizing the distribution of the dataset to check for skewness that will guide in
numerical_cols = df.select_dtypes(include=['float64', 'int64'])

# Create a new figure
plt.figure(figsize=(12, 16))

# Loop through each numerical column
for i, col in enumerate(numerical_cols.columns):
    plt.subplot(len(numerical_cols.columns)//2 + 1, 2, i+1)
    sns.histplot(df[col], kde=True, color='skyblue')
    plt.title(f'Histogram of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')

# Adjust layout to prevent overlapping
plt.tight_layout()

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



Handling missing data

In [130]:
```
# Imputing waterfront column with mode
mode_waterfront = df['waterfront'].mode()[0]
df['waterfront'].fillna(mode_waterfront, inplace=True)
```

In [131]:
```
# Dropping rows with missing 'View' values
df.dropna(subset=['view'], inplace=True)
```

In [132]:

```
# Imputing rows with missing 'yr_renovated' values
median_year = df['yr_renovated']. median()

df['yr_renovated'].fillna(median_year, inplace=True)
# If your data is normally distributed and does not have outliers affecting the mean si
# If your data is skewed or has outliers, imputing with the median might be more robust
```

Identyfing & Handling Duplicates

In [133]:

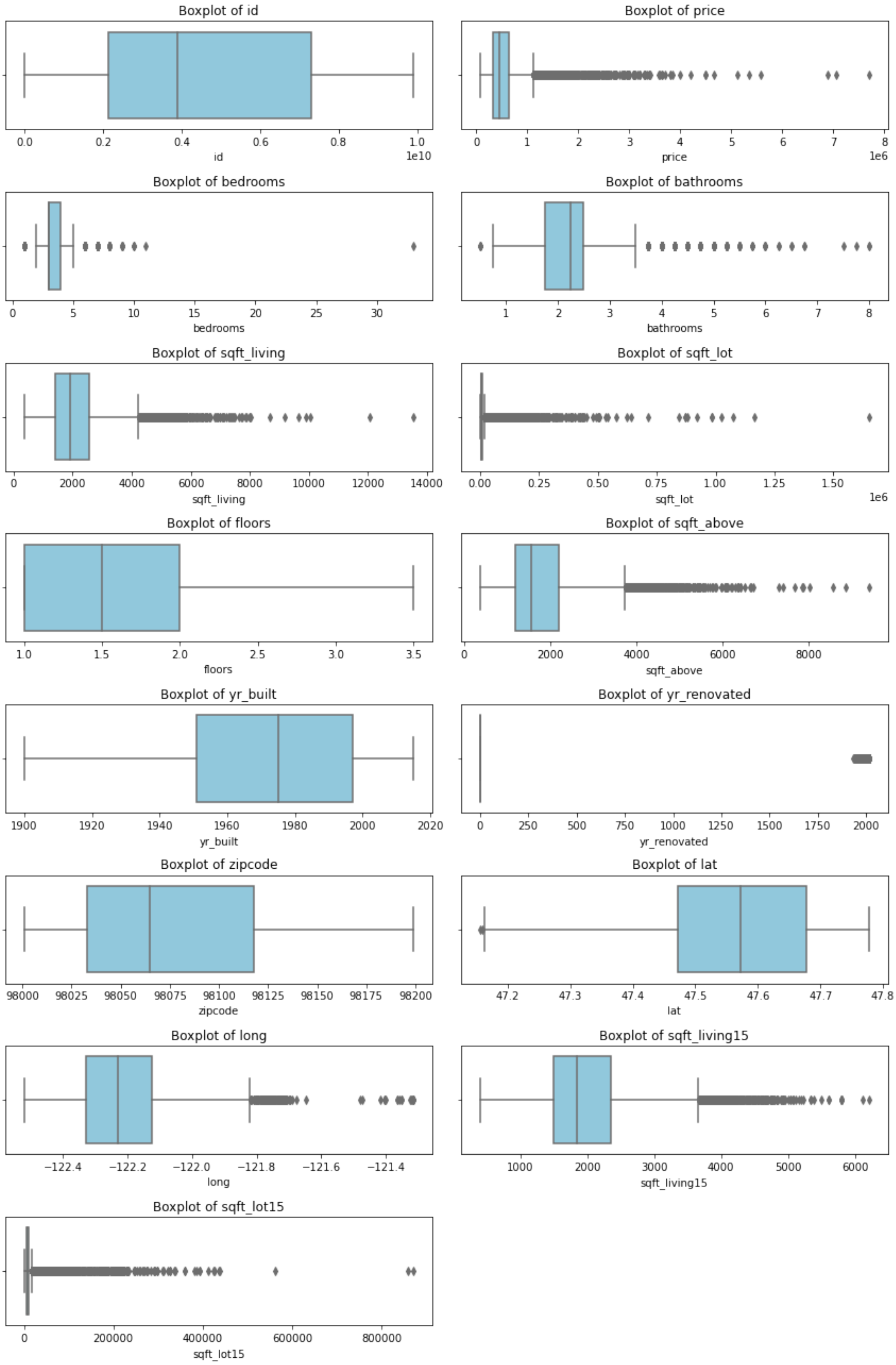
```
df.duplicated().sum()
```

Out[133]: 0

Identifying Outliers

In [134]:

```
# # Visualizing data to check for outliers

# Visualizing the distribution of the dataset to check for skewness that will guide in
numerical_cols = df.select_dtypes(include=['float64', 'int64'])

# Create a new figure
plt.figure(figsize=(12, 18))

# Loop through each numerical column
for i, col in enumerate(numerical_cols.columns):
    plt.subplot(len(numerical_cols.columns)//2 + 1, 2, i+1)
    sns.boxplot(x=df[col], color='skyblue')
    plt.title(f'Boxplot of {col}')
    plt.xlabel(col)

# Adjust layout to prevent overlapping
plt.tight_layout()

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


Boxplot of id
0.0 0.2 0.4 0.6 0.8 1.0
id
1e10
Boxplot of price
0 1 2 3 4 5 6 7 8
price
1e6
Boxplot of bedrooms
0 5 10 15 20 25 30
bedrooms
Boxplot of bathrooms
1 2 3 4 5 6 7 8
bathrooms
Boxplot of sqft_living
0 2000 4000 6000 8000 10000 12000 14000
sqft_living
Boxplot of sqft_lot
0.00 0.25 0.50 0.75 1.00 1.25 1.50
sqft_lot
1e6
Boxplot of floors
1.0 1.5 2.0 2.5 3.0 3.5
floors
Boxplot of sqft_above
0 2000 4000 6000 8000
sqft_above
Boxplot of yr_built
1900 1920 1940 1960 1980 2000 2020
yr_built
Boxplot of yr_renovated
0 250 500 750 1000 1250 1500 1750 2000
yr_renovated
Boxplot of zipcode
98000 98025 98050 98075 98100 98125 98150 98175 98200
zipcode
Boxplot of lat
47.2 47.3 47.4 47.5 47.6 47.7 47.8
lat
Boxplot of long
−122.4 −122.2 −122.0 −121.8 −121.6 −121.4
long
Boxplot of sqft_living15
1000 2000 3000 4000 5000 6000
sqft_living15
Boxplot of sqft_lot15
0 200000 400000 600000 800000
sqft_lot15

In [135]:
```
df.describe()
```

Out[135]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
count	21534.00000	21534.00000	21534.00000	21534.00000	21534.00000	21534.00000	21534.0000(
mean	4582351016.28703	540057.66383	3.37304	2.11571	2079.82785	15090.59636	1.4941:
std	2876779096.96191	366059.58123	0.92641	0.76860	917.44652	41380.20986	0.53981
min	1000102.00000	78000.00000	1.00000	0.50000	370.00000	520.00000	1.0000(
25%	2123212130.25000	322000.00000	3.00000	1.75000	1430.00000	5040.00000	1.0000(
50%	3904945195.00000	450000.00000	3.00000	2.25000	1910.00000	7617.00000	1.5000(
75%	7312175032.50000	645000.00000	4.00000	2.50000	2550.00000	10687.75000	2.0000(
max	9900000190.00000	7700000.00000	33.00000	8.00000	13540.00000	1651359.00000	3.5000(

We identified outliers in the following columns = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'yr_built', 'yr_renovated', 'sqft_living15', 'sqft_lot15']. We deployed the use of InterQuartile Range to handle the outliers

Handling Outliers

In [136]:
```
# Assuming df is your DataFrame containing the columns mentioned
columns = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
           'floors', 'yr_built', 'yr_renovated', 'sqft_living15', 'sqft_lot15']

# Calculate the quartiles
Q1 = df[columns].quantile(0.25)
Q3 = df[columns].quantile(0.75)

# Calculate the IQR
IQR = Q3 - Q1

# Identify outliers
outliers = ((df[columns] < (Q1 - 1.5 * IQR)) | (df[columns] > (Q3 + 1.5 * IQR))).any(ax
# Remove outliers
df_filtered = df[~outliers]

# Display the filtered DataFrame
print("Original DataFrame shape:", df.shape)
print("Filtered DataFrame shape:", df_filtered.shape)
```

```
Original DataFrame shape: (21534, 21)
Filtered DataFrame shape: (16856, 21)
```

In [137]:

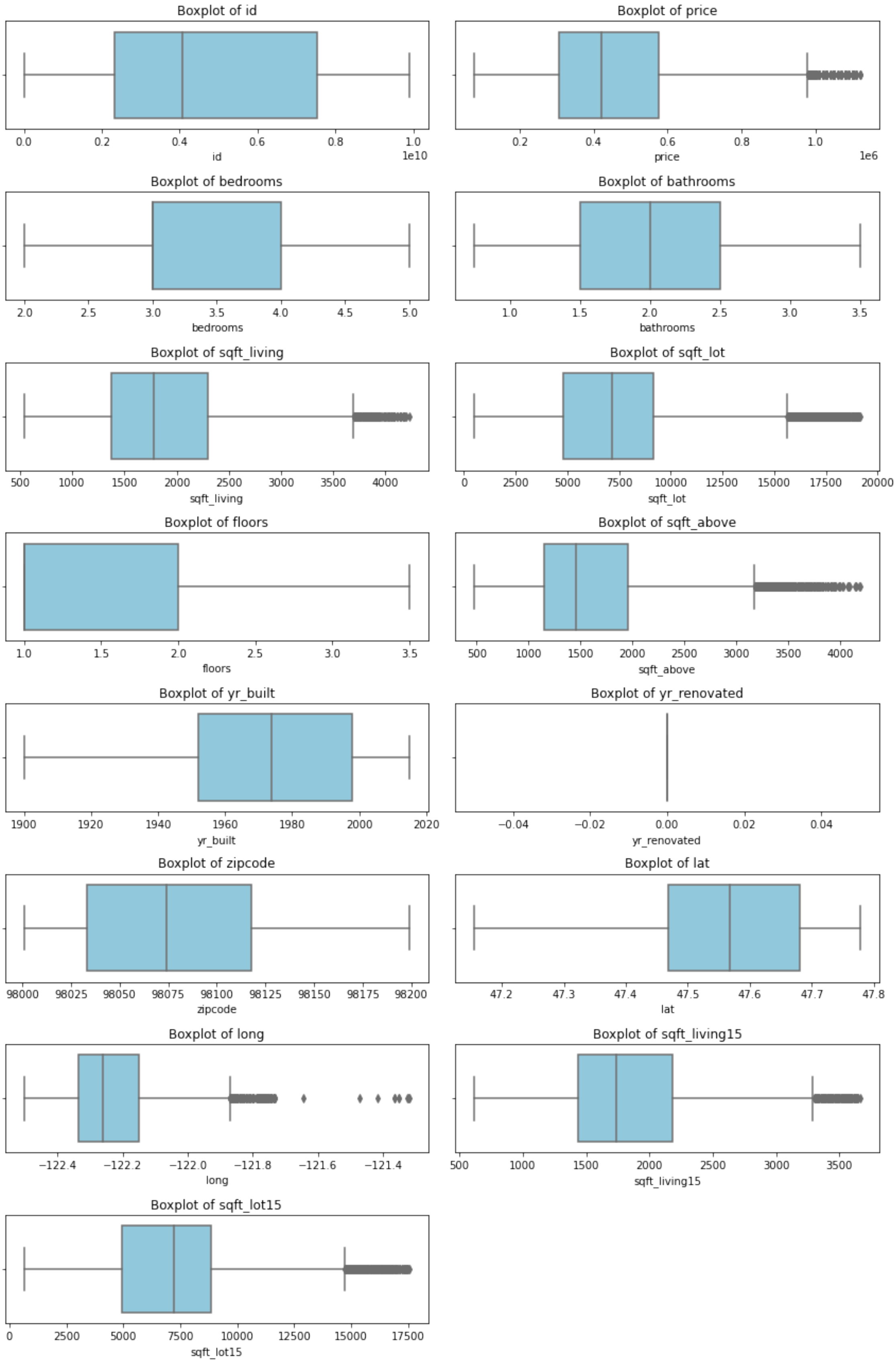
```
df_filtered.describe()
```

Out[137]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
count	16856.00000	16856.00000	16856.00000	16856.00000	16856.00000	16856.00000	16856.00000
mean	4765645346.76442	457700.71879	3.28156	1.99867	1874.11925	7156.29005	1.47209
std	2869688533.01589	196215.26781	0.78376	0.65870	671.83018	3437.87453	0.54561
min	2800031.00000	78000.00000	2.00000	0.75000	540.00000	520.00000	1.00000
25%	2326075090.00000	305498.75000	3.00000	1.50000	1370.00000	4800.00000	1.00000
50%	4077800425.00000	420000.00000	3.00000	2.00000	1780.00000	7155.00000	1.00000
75%	7518503248.75000	575000.00000	4.00000	2.50000	2300.00000	9138.25000	2.00000
max	9900000190.00000	1120000.00000	5.00000	3.50000	4230.00000	19141.00000	3.50000

In [138]:

```
# Visualizing data after handling outliers

# Visualizing the distribution of the dataset to check for skewness that will guide in
numerical_cols = df_filtered.select_dtypes(include=['float64', 'int64'])

# Create a new figure
plt.figure(figsize=(12, 18))

# Loop through each numerical column
for i, col in enumerate(numerical_cols.columns):
    plt.subplot(len(numerical_cols.columns)//2 + 1, 2, i+1)
    sns.boxplot(x=df_filtered[col], color='skyblue')
    plt.title(f'Boxplot of {col}')
    plt.xlabel(col)

# Adjust layout to prevent overlapping
plt.tight_layout()

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


Boxplot of id
0.0 0.2 0.4 0.6 0.8 1.0
id
1e10
Boxplot of price
0.2 0.4 0.6 0.8 1.0
price
1e6
Boxplot of bedrooms
2.0 2.5 3.0 3.5 4.0 4.5 5.0
bedrooms
Boxplot of bathrooms
1.0 1.5 2.0 2.5 3.0 3.5
bathrooms
Boxplot of sqft_living
500 1000 1500 2000 2500 3000 3500 4000
sqft_living
Boxplot of sqft_lot
0 2500 5000 7500 10000 12500 15000 17500 20000
sqft_lot
Boxplot of floors
1.0 1.5 2.0 2.5 3.0 3.5
floors
Boxplot of sqft_above
500 1000 1500 2000 2500 3000 3500 4000
sqft_above
Boxplot of yr_built
1900 1920 1940 1960 1980 2000 2020
yr_built
Boxplot of yr_renovated
−0.04 −0.02 0.00 0.02 0.04
yr_renovated
Boxplot of zipcode
98000 98025 98050 98075 98100 98125 98150 98175 98200
zipcode
Boxplot of lat
47.2 47.3 47.4 47.5 47.6 47.7 47.8
lat
Boxplot of long
−122.4 −122.2 −122.0 −121.8 −121.6 −121.4
long
Boxplot of sqft_living15
500 1000 1500 2000 2500 3000 3500
sqft_living15
Boxplot of sqft_lot15
0 2500 5000 7500 10000 12500 15000 17500
sqft_lot15

In [139]:
```
df_filtered.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16856 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             16856 non-null  int64  
 1   date           16856 non-null  object 
 2   price          16856 non-null  float64
 3   bedrooms       16856 non-null  int64  
 4   bathrooms      16856 non-null  float64
 5   sqft_living    16856 non-null  int64  
 6   sqft_lot       16856 non-null  int64  
 7   floors         16856 non-null  float64
 8   waterfront     16856 non-null  object 
 9   view           16856 non-null  object 
 10  condition      16856 non-null  object 
 11  grade          16856 non-null  object 
 12  sqft_above     16856 non-null  int64  
 13  sqft_basement  16856 non-null  object 
 14  yr_built       16856 non-null  int64  
 15  yr_renovated   16856 non-null  float64
 16  zipcode        16856 non-null  int64  
 17  lat            16856 non-null  float64
 18  long           16856 non-null  float64
 19  sqft_living15  16856 non-null  int64  
 20  sqft_lot15     16856 non-null  int64  
dtypes: float64(6), int64(9), object(6)
memory usage: 2.8+ MB
```

In [140]:
```
# Exporting the cleaned dataset
df_filtered.to_csv('cleaned_dataset.csv', index=False)
```

In [141]:
```
df_filtered.head()
```

Out[141]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	
0	7129300520	10/13/2014	221900.00000	3	1.00000	1180	5650	1.00000	NO	N
2	5631500400	2/25/2015	180000.00000	2	1.00000	770	10000	1.00000	NO	N
3	2487200875	12/9/2014	604000.00000	4	3.00000	1960	5000	1.00000	NO	N
4	1954400510	2/18/2015	510000.00000	3	2.00000	1680	8080	1.00000	NO	N
6	1321400060	6/27/2014	257500.00000	3	2.25000	1715	6819	2.00000	NO	N

5 rows × 21 columns

In [142]:
```
df = pd.read_csv('cleaned_dataset.csv')

df.sample(2)
```

Out[142]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
6224	6928600330	8/20/2014	278000.00000	5	1.75000	2170	9752	1.00000	NO
4319	6204400170	6/20/2014	477000.00000	3	1.75000	1780	8085	1.00000	NO

2 rows × 21 columns

Exploratory Data Analysis (EDA) for the cleaned Dataset

In [143]:
```
#1. General Information
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16856 entries, 0 to 16855
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             16856 non-null  int64
 1   date           16856 non-null  object
 2   price          16856 non-null  float64
 3   bedrooms       16856 non-null  int64
 4   bathrooms      16856 non-null  float64
 5   sqft_living    16856 non-null  int64
 6   sqft_lot       16856 non-null  int64
 7   floors         16856 non-null  float64
 8   waterfront     16856 non-null  object
 9   view           16856 non-null  object
 10  condition      16856 non-null  object
 11  grade          16856 non-null  object
 12  sqft_above     16856 non-null  int64
 13  sqft_basement  16856 non-null  object
```

In [144]:
```
# Droping id, date, yr_renovated and zipcode
df.drop('id', axis = 1, inplace = True)
df.drop('date', axis = 1, inplace = True)
df.drop('yr_renovated', axis = 1 , inplace = True)
df.drop('zipcode', axis = 1 , inplace = True)
```

In [145]:
```
df.shape
print(f"Number of rows: {df.shape[0]}")
print(f"Number of columns: {df.shape[1]}")
```

```
Number of rows: 16856
Number of columns: 17
```

In [146]:
```
#2. Check for missing values
print(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
sqft_basement    0
yr_built         0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

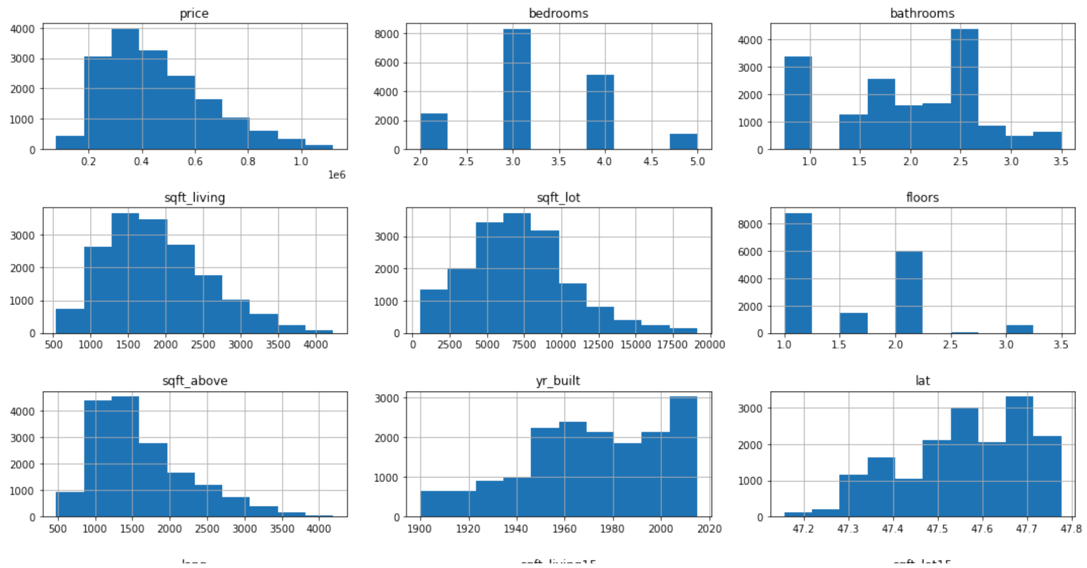
No missing values

In [147]:
```
#3. Descriptive Statistics
pd.set_option('display.float_format',lambda x: '%.5f' %x)
df.describe()
```

Out[147]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above	y
count	16856.00000	16856.00000	16856.00000	16856.00000	16856.00000	16856.00000	16856.00000	16856
mean	457700.71879	3.28156	1.99867	1874.11925	7156.29005	1.47209	1622.11171	1971
std	196215.26781	0.78376	0.65870	671.83018	3437.87453	0.54561	641.49376	29
min	78000.00000	2.00000	0.75000	540.00000	520.00000	1.00000	480.00000	1900
25%	305498.75000	3.00000	1.50000	1370.00000	4800.00000	1.00000	1150.00000	1952
50%	420000.00000	3.00000	2.00000	1780.00000	7155.00000	1.00000	1460.00000	1974
75%	575000.00000	4.00000	2.50000	2300.00000	9138.25000	2.00000	1960.25000	1998
max	1120000.00000	5.00000	3.50000	4230.00000	19141.00000	3.50000	4190.00000	2015

In [148]:
```
# Data Visualization
df.hist(figsize=(15, 10))
plt.tight_layout()
plt.show()
```



In [149]:
```
df.columns
```

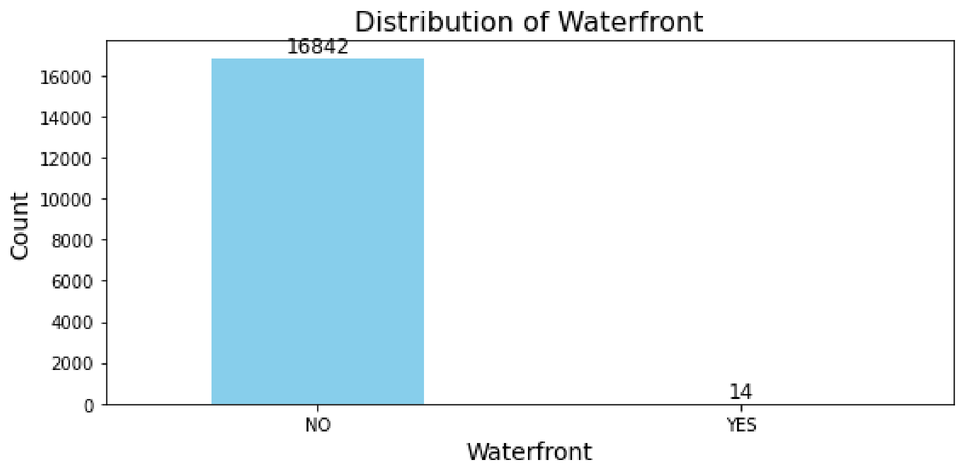
Out[149]:
```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'lat', 'long', 'sqft_living15',
       'sqft_lot15'],
      dtype='object')
```

In [150]:

```
categorical_columns = ['waterfront', 'view', 'condition', 'grade']

for column in categorical_columns:
    counts = df[column].value_counts()
    print(f"Counts for {column}:\n{counts}\n")
```

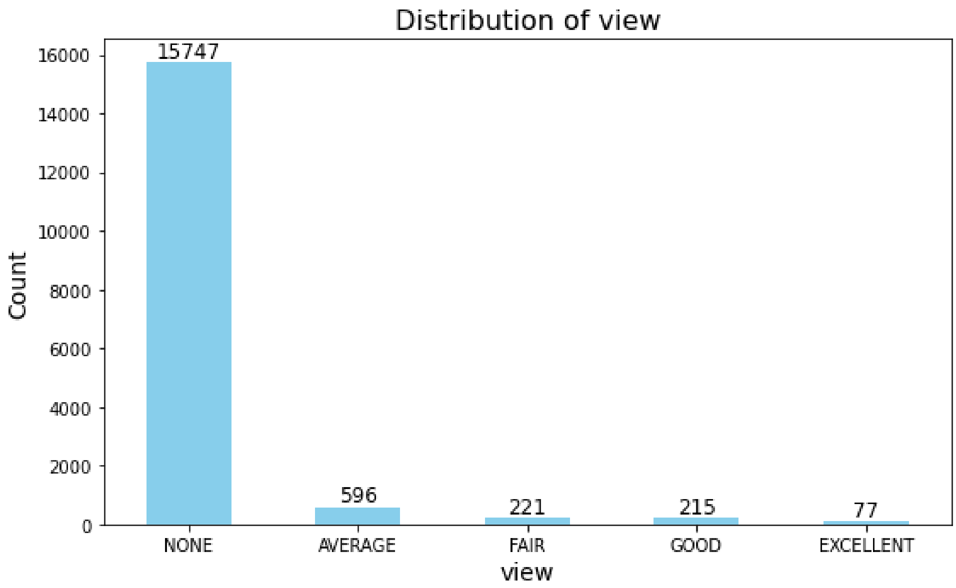
```
Counts for waterfront:
NO     16842
YES       14
Name: waterfront, dtype: int64

Counts for view:
NONE         15747
AVERAGE        596
FAIR           221
GOOD           215
EXCELLENT       77
Name: view, dtype: int64

Counts for condition:
Average      10903
Good          4461
Very Good     1359
Fair           116
Poor            17
Name: condition, dtype: int64

Counts for grade:
7 Average        7885
8 Good           4966
6 Low Average    1716
9 Better         1691
10 Very Good      397
5 Fair            161
11 Excellent       31
4 Low               9
Name: grade, dtype: int64
```

In [151]:
```
# Calculate frequency of each category
value_counts = df['waterfront'].value_counts()
# Plot bar chart
plt.figure(figsize=(8, 4))
bar_chart = value_counts.plot(kind='bar', color='skyblue')

# Add labels and title
for i, v in enumerate(value_counts):
    plt.text(i, v + 10, str(v), ha='center', va='bottom', fontsize=12)

plt.xlabel('Waterfront', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.title('Distribution of Waterfront', fontsize=16)
plt.xticks(rotation=0)
plt.tight_layout()
# Show plot
plt.show()
```



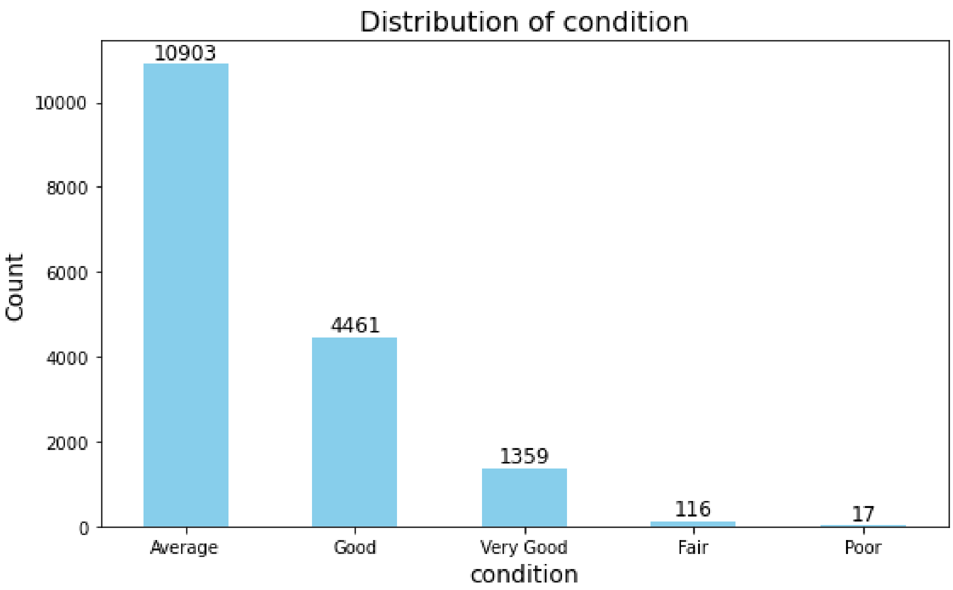
In [152]:

```
# Calculate frequency of each category
value_counts = df['view'].value_counts()
# Plot bar chart
plt.figure(figsize=(8, 5))
bar_chart = value_counts.plot(kind='bar', color='skyblue')

# Add labels and title
for i, v in enumerate(value_counts):
    plt.text(i, v + 10, str(v), ha='center', va='bottom', fontsize=12)

plt.xlabel('view', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.title('Distribution of view', fontsize=16)
plt.xticks(rotation=0)
plt.tight_layout()
# Show plot
plt.show()
```



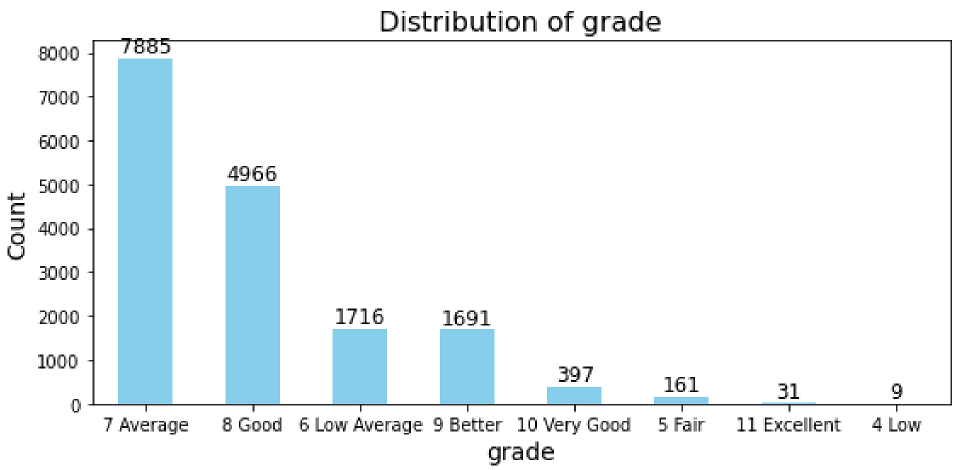
In [153]:

```
# Calculate frequency of each category
value_counts = df['condition'].value_counts()
# Plot bar chart
plt.figure(figsize=(8, 5))
bar_chart = value_counts.plot(kind='bar', color='skyblue')

# Add labels and title
for i, v in enumerate(value_counts):
    plt.text(i, v + 10, str(v), ha='center', va='bottom', fontsize=12)

plt.xlabel('condition', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.title('Distribution of condition', fontsize=16)
plt.xticks(rotation=0)
plt.tight_layout()
# Show plot
plt.show()
```



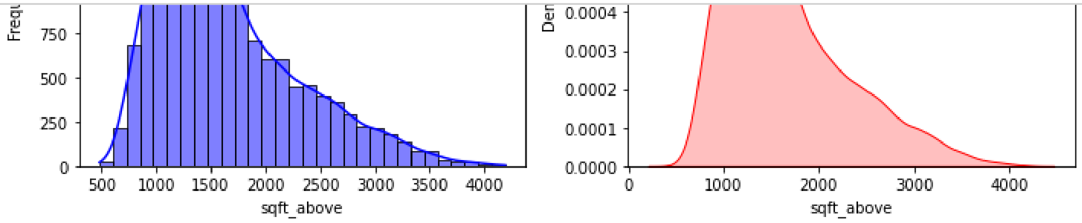
In [154]:

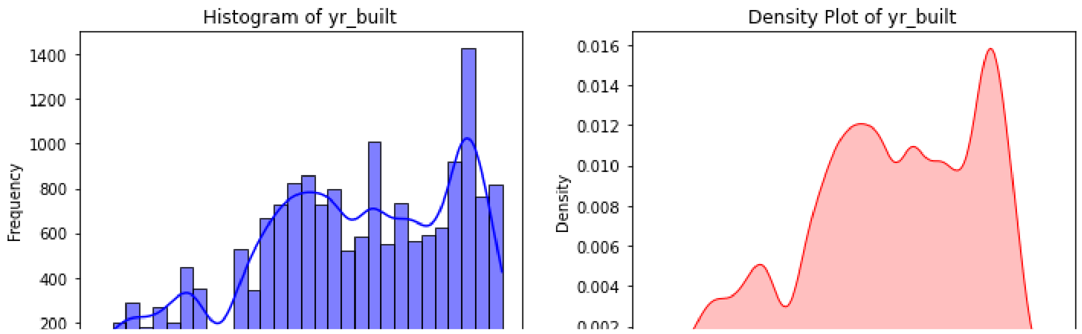
```
# Calculate frequency of each category
value_counts = df['grade'].value_counts()
# Plot bar chart
plt.figure(figsize=(8, 4))
bar_chart = value_counts.plot(kind='bar', color='skyblue')

# Add labels and title
for i, v in enumerate(value_counts):
    plt.text(i, v + 10, str(v), ha='center', va='bottom', fontsize=12)

plt.xlabel('grade', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.title('Distribution of grade', fontsize=16)
plt.xticks(rotation=0)
plt.tight_layout()
# Show plot
plt.show()
```



In [155]:

```
# 3.Normality and spread (standard deviation, skewness and kurtosis)

numerical_columns = df.select_dtypes(include=[np.number]).columns

# Create histograms and density plots
for column in numerical_columns:
    plt.figure(figsize=(10, 4))

    # Histogram
    plt.subplot(1, 2, 1)
    sns.histplot(df[column], kde=True, bins=30, color='blue')
    plt.title(f'Histogram of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')

    # Density Plot
    plt.subplot(1, 2, 2)
    sns.kdeplot(df[column], color='red', fill=True)
    plt.title(f'Density Plot of {column}')
    plt.xlabel(column)
    plt.ylabel('Density')

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





In [156]:

```
# 3.Normality and spread (standard deviation, skewness and kurtosis)

# Compute and print the standard deviation, skewness, and kurtosis
pd.set_option('display.float_format',lambda x: '%.5f' %x)
normality_spread = {
    'standard_deviation': df[numerical_columns].std(),
    'skewness': df[numerical_columns].skew(),
    'kurtosis': df[numerical_columns].kurt()
}

for key, value in normality_spread.items():
    print(f"{key}:")
    print(value)
    print("-----------------------")
```

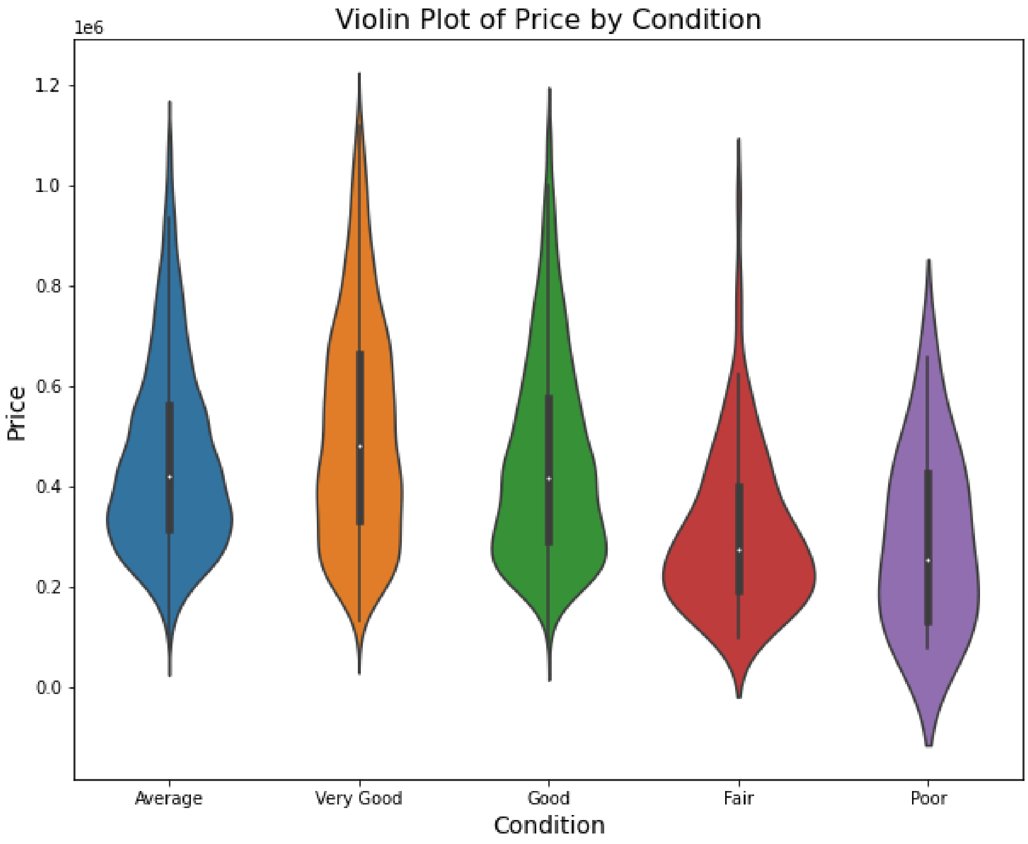
```
standard_deviation:
price           196215.26781
bedrooms             0.78376
bathrooms            0.65870
sqft_living        671.83018
sqft_lot          3437.87453
floors               0.54561
sqft_above         641.49376
yr_built            29.37420
lat                  0.13916
long                 0.13269
sqft_living15      548.65798
sqft_lot15        3059.72350
dtype: float64
-----------------------
skewness:
price            0.80387
bedrooms         0.22383
bathrooms       -0.07454
sqft_living      0.60650
sqft_lot         0.54922
floors           0.75234
sqft_above       1.01696
yr_built        -0.43612
lat             -0.45213
long             0.96807
sqft_living15    0.75328
sqft_lot15       0.34497
dtype: float64
-----------------------
kurtosis:
price            0.19802
bedrooms        -0.32701
bathrooms       -0.76630
sqft_living     -0.03210
sqft_lot         0.43316
floors          -0.25896
sqft_above       0.60689
yr_built        -0.67909
lat             -0.75922
long             1.20574
sqft_living15    0.17706
sqft_lot15       0.21554
dtype: float64
-----------------------
```

Bivariate Analysis

In [157]:
```
df.columns
```

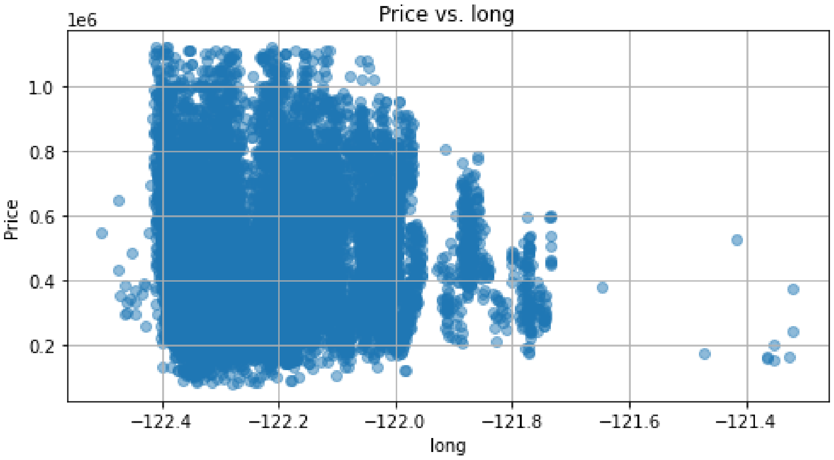
Out[157]:
```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'lat', 'long', 'sqft_living15',
       'sqft_lot15'],
      dtype='object')
```

In [158]:
```
# Create violin plot
plt.figure(figsize=(10, 8))
sns.violinplot(x='condition', y='price', data=df)

# Add labels and title
plt.xlabel('Condition', fontsize=14)
plt.ylabel('Price', fontsize=14)
plt.title('Violin Plot of Price by Condition', fontsize=16)

# Show plot
plt.show()
```



In [159]:

```
#Price against Long
# Scatter plot
plt.figure(figsize=(8, 4))
plt.scatter(df['long'], df['price'], alpha=0.5)
plt.title('Price vs. long')
plt.xlabel('long')
plt.ylabel('Price')
plt.grid(True)
plt.show()
```



In [160]:

```
df.columns
```

Out[160]:

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

In [161]:

```
# Dropping the date  and id Columns
```

In [162]:

```
df.head()
```

Out[162]:

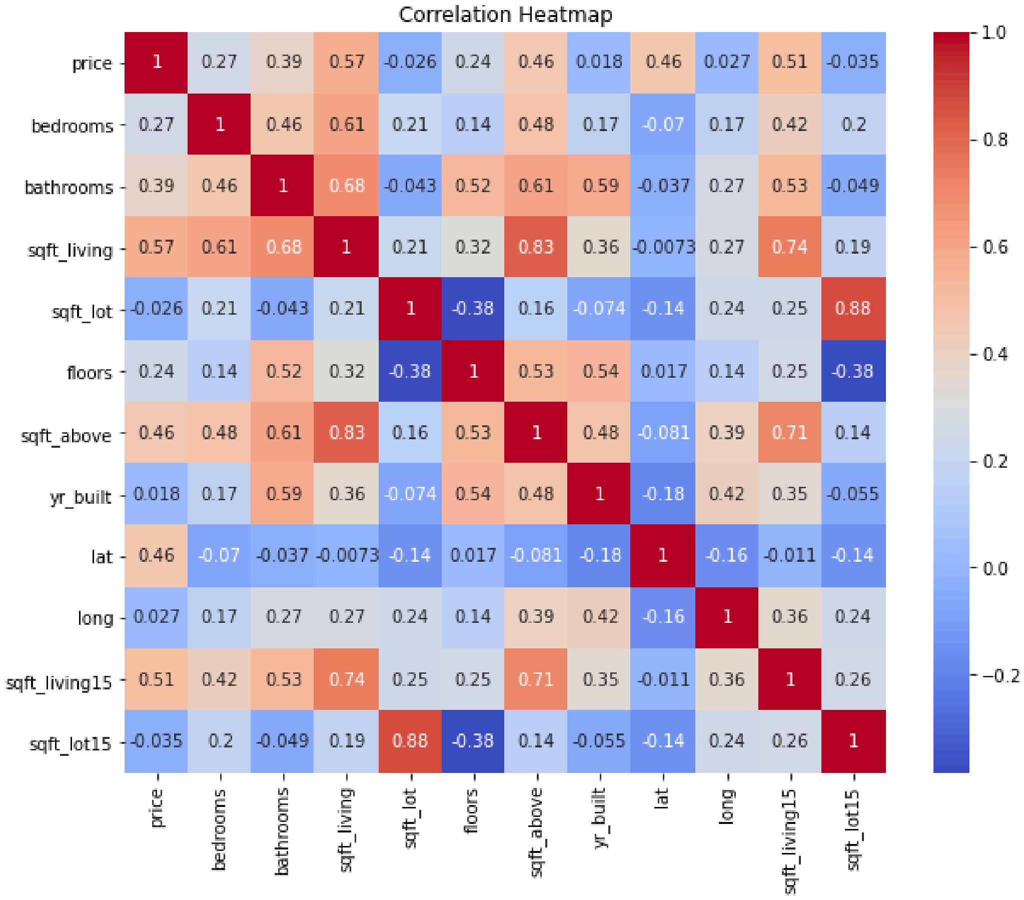
	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade
0	221900.00000	3	1.00000	1180	5650	1.00000	NO	NONE	Average	7 Average
1	180000.00000	2	1.00000	770	10000	1.00000	NO	NONE	Average	6 Low Average
2	604000.00000	4	3.00000	1960	5000	1.00000	NO	NONE	Very Good	7 Average
3	510000.00000	3	2.00000	1680	8080	1.00000	NO	NONE	Average	8 Good
4	257500.00000	3	2.25000	1715	6819	2.00000	NO	NONE	Average	7 Average

Multivariate Analysis

***Checking for Correlations**

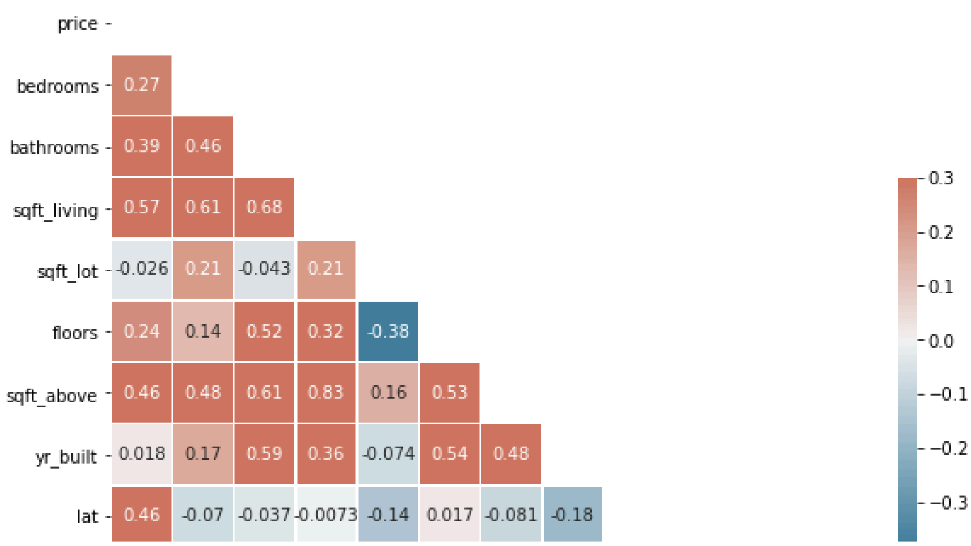
In [163]:

```
corr_matrix = df.corr()

# Create heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



In [164]:

```
# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(10, 8))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr_matrix, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, annot=True , linewidths=.5, cbar_kws={"shrink": .5})

plt.show()
```



Hypothesis testing

Null Hypothesis (H0): There is no significant relationship between the various housing features and house prices in King County's real estate market.

Alternate Hypothesis (H1): There is a significant relationship between the various housing features and house prices in King County's real estate market.

In [165]:

```
import statsmodels.api as sm
from statsmodels.formula.api import ols

# where 'Price' is the dependent variable and other columns are predictors

# Define the formula for the ANOVA model
formula = 'price ~ bedrooms + bathrooms + sqft_living + sqft_lot + floors + waterfront

# Fit the ANOVA model
model = ols(formula, data=df).fit()

# Perform ANOVA
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

```
                            sum_sq          df          F  PR(>F)
waterfront       464634044603.99353     1.00000   39.50206 0.00000
view            5045252365447.17383     4.00000  107.23378 0.00000
condition       5711310620626.59668     4.00000  121.39044 0.00000
grade          33926470133297.21484     7.00000  412.04944 0.00000
sqft_basement   5311068772892.15723   226.00000    1.99794 0.00000
bedrooms         282515277209.35437     1.00000   24.01876 0.00000
bathrooms        809171913677.27893     1.00000   68.79383 0.00000
sqft_living      264262708732.59076     1.00000   22.46697 0.00000
sqft_lot         176957102443.26331     1.00000   15.04446 0.00011
floors           279388340684.16693     1.00000   23.75292 0.00000
sqft_above        10474531366.70181     1.00000    0.89052 0.34535
yr_built       17688915111019.32812     1.00000 1503.86860 0.00000
lat            78457892140581.51562     1.00000 6670.29941 0.00000
long             301507451243.91327     1.00000   25.63343 0.00000
sqft_living15   4119115357445.20898     1.00000  350.19718 0.00000
sqft_lot15       939594228063.77527     1.00000   79.88202 0.00000
Residual      195277279937174.00000 16602.00000        nan     nan
```

Conclusion: Based on the provide d ANOVA table, we reject the null hypothesis (H0) that there is no significant relationship between the various housing features and house prices in King County's real estate market. Therefore, we accept the alternate hypothesis (H1) that there is a significant relationship between the various housing features and house prices in King County's real estate market

Linear Regression

We will build **two models — one simple linear regression model and one multiple linear regresssion model**

There are two relevant components of interpreting the model summaries: model **metrics** such as r-squared and p-values, which tell us how well our model is fit to the data, and model **parameters** (intercept and coefficients), which tell us how the model is using the feature(s) to predict the target.

Simple Linear Regression

The formula for a simple linear regression is: $y=mx+b$ where y is the price, m is the slope of sqft_living, x is sqft_living, and b is the y-intercept (the value of y when x is 0).

In [166]:
```
df.corr()["price"]
```

Out[166]:
```
price            1.00000
bedrooms         0.26537
bathrooms        0.38689
sqft_living      0.57173
sqft_lot        -0.02554
floors           0.24017
sqft_above       0.45643
yr_built         0.01767
lat              0.46286
long             0.02653
sqft_living15    0.51273
sqft_lot15      -0.03528
Name: price, dtype: float64
```

The sqft_living feature has the highest correlation with price, so we will use it to build a simple linear regression model.

In [167]:
```
#we import the necessary libraries

from statsmodels.formula.api import ols


# Replace None with appropriate code
simple_formula = 'price ~ sqft_living'
simple_model = ols(formula=simple_formula, data = df).fit()
simple_model_summary = simple_model.summary()


simple_model_summary
```

Out[167]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.327
Model:	OLS	**Adj. R-squared:**	0.327
Method:	Least Squares	**F-statistic:**	8184.
Date:	Sun, 07 Apr 2024	**Prob (F-statistic):**	0.00
Time:	17:03:26	**Log-Likelihood:**	-2.2600e+05
No. Observations:	16856	**AIC:**	4.520e+05
Df Residuals:	16854	**BIC:**	4.520e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.448e+05	3674.675	39.395	0.000	1.38e+05	1.52e+05
sqft_living	166.9787	1.846	90.467	0.000	163.361	170.597

Omnibus:	867.752	**Durbin-Watson:**	1.972
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1005.835
Skew:	0.588	**Prob(JB):**	3.85e-219
Kurtosis:	3.226	**Cond. No.**	5.90e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.9e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

The R squared value of our regression model is 32.7%. This value indicates that approximately 32.7% of the variability in the dependent variable can be explained by the independent variable(s) included in our model. The remaining 67.3% of the variability is not accounted for by the predictors in the model.

The R squared value of 32.7% suggests a moderate level of explanatory power of our regression model. This result highlights the need for further research to explore additional variables and factors that may influence the dependent variable and to improve the predictive accuracy and explanatory power of the model.

Is the model statistically significant at α=0.05? # Compare the probability of the f-statistic to the alpha. The p-value of 0.000 is less than the α=0.05 therefore the model is statistically significant.

In [168]:
```
# Slope (coefficient of sqft_living)
m = simple_model.params['sqft_living']
m

# Intercept (coefficient of Intercept)
b = simple_model.params['Intercept']

# YOUR CODE HERE
# raise NotImplementedError()

print(f"""
Our simple linear regression model found a y-intercept of ${round(b, 2)}, theis means t
living area, the price increases by ${round(m, 2)}
""")
```

```
Our simple linear regression model found a y-intercept of $144762.76, theis means that
for every increase of 1 square foot
living area, the price increases by $166.98
```

Features Engineering

In [169]:
```
# classify the houses based on their grades as suppar, average and superior
```

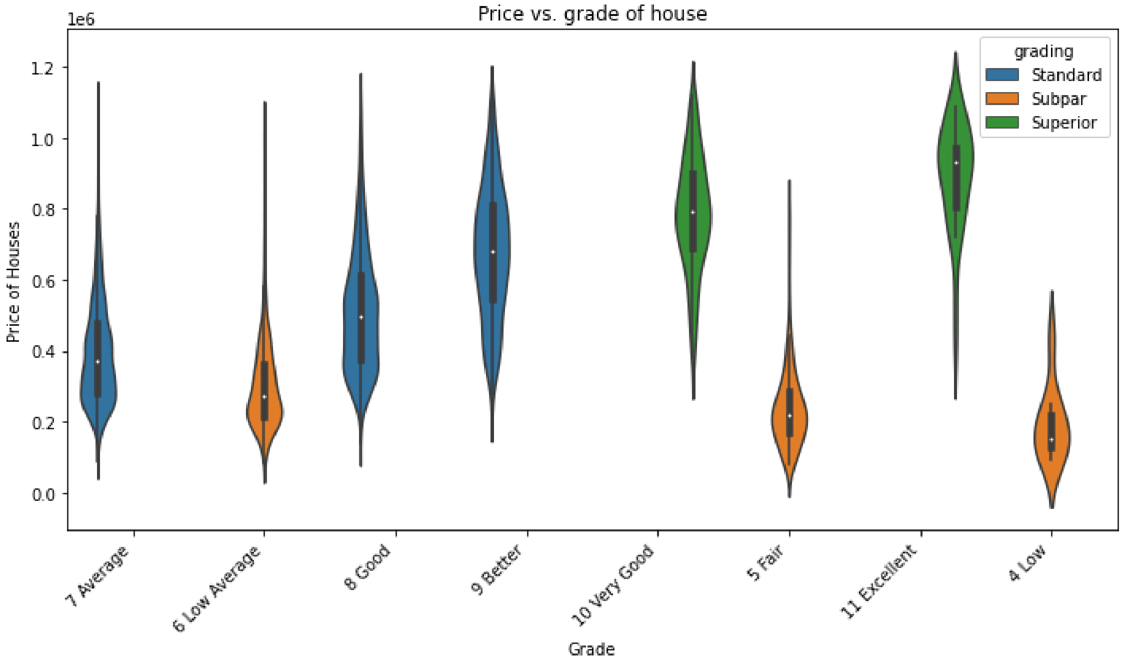
In [170]:
```
def grading (grade):
    if grade == "4 Low" or grade == "5 Fair"or grade == "6 Low Average":
        return "Subpar"
    elif grade == "7 Average" or grade == "8 Good" or grade == "9 Better":
        return "Standard"
    else:
        return "Superior"
```

In [171]:
```
df['grading'] = df['grade'].apply(grading)
```

In [172]:
```
# inspect the gradng column

grad = df['grading'].value_counts()
print(f"Counts for grading:\n{grad}\n")
```

```
Counts for grading:
Standard    14542
Subpar       1886
Superior      428
Name: grading, dtype: int64
```

In [173]:
```
# compare the grading against price
plt.figure(figsize=(10, 6))

sns.violinplot(x='grade', y='price', hue='grading', data=df) #, alpha=0.7, s=80)

plt.title("Price vs. grade of house")
plt.xlabel("Grade")
plt.ylabel("Price of Houses")

plt.xticks(rotation=45, ha='right')


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



Dealing with categorical data

One hot encoding

In [174]:
```
# Perform one-hot encoding for each categorical column
one_hot_encoded_data = pd.get_dummies(df, columns=categorical_columns)

# Separate numerical columns
numerical_columns = [col for col in df.columns if col not in categorical_columns]

# Print the first few rows of the one-hot encoded DataFrame
print(one_hot_encoded_data.head())
```

```
          price  bedrooms  bathrooms  sqft_living  sqft_lot  floors  \
0 221900.00000         3    1.00000         1180      5650 1.00000
1 180000.00000         2    1.00000          770     10000 1.00000
2 604000.00000         4    3.00000         1960      5000 1.00000
3 510000.00000         3    2.00000         1680      8080 1.00000
4 257500.00000         3    2.25000         1715      6819 2.00000

   sqft_above sqft_basement  yr_built      lat  ...  condition_Poor  \
0        1180           0.0      1955 47.51120  ...               0
1         770           0.0      1933 47.73790  ...               0
2        1050         910.0      1965 47.52080  ...               0
3        1680           0.0      1987 47.61680  ...               0
4        1715             ?      1995 47.30970  ...               0

   condition_Very Good  grade_10 Very Good grade_11 Excellent  grade_4 Low  \
0                    0                   0                  0            0
1                    0                   0                  0            0
2                    1                   0                  0            0
3                    0                   0                  0            0
4                    0                   0                  0            0

   grade_5 Fair  grade_6 Low Average  grade_7 Average  grade_8 Good  \
0             0                    0                1             0
1             0                    1                0             0
2             0                    0                1             0
3             0                    0                0             1
4             0                    0                1             0

   grade_9 Better
0               0
1               0
2               0
3               0
4               0

[5 rows x 34 columns]
```

In [175]:
```
# Concatenate one-hot encoded features with numerical features
X = pd.concat([one_hot_encoded_data[numerical_columns], one_hot_encoded_data.drop(numer
```

In [176]:
```
df1 = one_hot_encoded_data
```

In [177]:
```
df1.columns
```

Out[177]:
```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'sqft_above', 'sqft_basement', 'yr_built', 'lat', 'long',
       'sqft_living15', 'sqft_lot15', 'grading', 'waterfront_NO',
       'waterfront_YES', 'view_AVERAGE', 'view_EXCELLENT', 'view_FAIR',
       'view_GOOD', 'view_NONE', 'condition_Average', 'condition_Fair',
       'condition_Good', 'condition_Poor', 'condition_Very Good',
       'grade_10 Very Good', 'grade_11 Excellent', 'grade_4 Low',
       'grade_5 Fair', 'grade_6 Low Average', 'grade_7 Average',
       'grade_8 Good', 'grade_9 Better'],
      dtype='object')
```

Normalization

We will normalize (or standardize) our data for this linear regression because our data is in different Scales/ units. This will help us

1. Handle the data better
2. Ease interpretation of the model since the coefficients in our linear regression model represent the change in the price associated with a one-unit change in the standardized predictor variable we are reviewing.
3. Reduce the impact of multicollinearity on the regression coefficients and their interpretability.

Standardization / normalization of the data results in a mean of zero and a standard deviation of 1

In [178]:

```
X = df1[['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'sqft_above', 'sqft_basement', 'yr_built', 'lat', 'long',
       'sqft_living15', 'sqft_lot15', 'waterfront_NO',
       'waterfront_YES', 'view_AVERAGE', 'view_EXCELLENT', 'view_FAIR',
       'view_GOOD', 'view_NONE', 'condition_Average', 'condition_Fair',
       'condition_Good', 'condition_Poor', 'condition_Very Good',
       'grade_10 Very Good', 'grade_11 Excellent', 'grade_4 Low',
       'grade_5 Fair', 'grade_6 Low Average', 'grade_7 Average',
       'grade_8 Good', 'grade_9 Better']]

# we only consider columns with numericals
numerical_cols1 = X.select_dtypes(include=['float64', 'int64', 'uint8']).columns

# Select only numerical columns
X = X[numerical_cols1]
```

In [179]:

```
bsmt = df1['sqft_basement'].value_counts()
print(f"Counts for basement:\n{bsmt}\n")
```

```
Counts for basement:
0.0       10227
?           347
500.0       182
600.0       173
700.0       170
          ...
1520.0        1
65.0          1
506.0         1
243.0         1
143.0         1
Name: sqft_basement, Length: 227, dtype: int64
```

In [180]:

```
# Convert column to numeric (to calculate median)
df1['sqft_basement'] = pd.to_numeric(df1['sqft_basement'], errors='coerce')

# Calculate median excluding NaN values
median_value = df1['sqft_basement'].median()

# Replace NaN values with median
df1['sqft_basement'] = df1['sqft_basement'].fillna(median_value)
```

In [181]:

```
from sklearn.preprocessing  import StandardScaler

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler to the data and transform the data
standardized_data = scaler.fit_transform(X)

# Convert the standardized data back to a DataFrame
standardized_df = pd.DataFrame(standardized_data, columns=numerical_cols1)

print("Original Data:")
# print(df)
print("\nStandardized Data:")
print(standardized_df)
```

```
Original Data:

Standardized Data:
       bedrooms  bathrooms  sqft_living  sqft_lot   floors  sqft_above  \
0      -0.35925   -1.51615     -1.03321  -0.43816 -0.86527    -0.68921
1      -1.63519   -1.51615     -1.64350   0.82720 -0.86527    -1.32836
2       0.91668    1.52021      0.12783  -0.62723 -0.86527    -0.89187
3      -0.35925    0.00203     -0.28895   0.26869 -0.86527     0.09024
4      -0.35925    0.38157     -0.23685  -0.09811  0.96759     0.14480
...         ...        ...          ...       ...      ...         ...
16851  -0.35925    0.76112     -0.51223  -1.75267  2.80045    -0.14359
16852   0.91668    0.76112      0.64882  -0.39074  0.96759     1.07235
16853  -1.63519   -1.89570     -1.27137  -1.68897  0.96759    -0.93864
16854  -0.35925    0.76112     -0.40803  -1.38703  0.96759    -0.03447
16855  -1.63519   -1.89570     -1.27137  -1.76867  0.96759    -0.93864

       yr_built      lat     long  sqft_living15  ...  condition_Poor  \
0      -0.55201 -0.34384 -0.22833       -0.92432  ...        -0.03177
1      -1.30099  1.28532 -0.04746        1.59098  ...        -0.03177
2      -0.21157 -0.27485 -1.25329       -0.88786  ...        -0.03177
3       0.53741  0.41504  1.36939       -0.08588  ...        -0.03177
4       0.80977 -1.79191 -0.75588        0.71245  ...        -0.03177
...         ...      ...      ...            ...  ...             ...
16851   1.28639  1.00792 -0.89908       -0.57801  ...        -0.03177
16852   1.45661 -0.34744 -1.01966       -0.03120  ...        -0.03177
16853   1.28639  0.25407 -0.54486       -1.50758  ...        -0.03177
16854   1.11617 -0.17640  1.18852       -0.79673  ...        -0.03177
16855   1.25235  0.25191 -0.54486       -1.50758  ...        -0.03177

       condition_Very Good  grade_10 Very Good  grade_11 Excellent  \
0                 -0.29613            -0.15531            -0.04292
1                 -0.29613            -0.15531            -0.04292
2                  3.37687            -0.15531            -0.04292
3                 -0.29613            -0.15531            -0.04292
4                 -0.29613            -0.15531            -0.04292
...                    ...                 ...                 ...
16851             -0.29613            -0.15531            -0.04292
16852             -0.29613            -0.15531            -0.04292
16853             -0.29613            -0.15531            -0.04292
16854             -0.29613            -0.15531            -0.04292
16855             -0.29613            -0.15531            -0.04292

       grade_4 Low  grade_5 Fair  grade_6 Low Average  grade_7 Average  \
0         -0.02311      -0.09820             -0.33666          1.06664
1         -0.02311      -0.09820              2.97033         -0.93752
2         -0.02311      -0.09820             -0.33666          1.06664
3         -0.02311      -0.09820             -0.33666         -0.93752
4         -0.02311      -0.09820             -0.33666          1.06664
...            ...           ...                  ...              ...
16851     -0.02311      -0.09820             -0.33666         -0.93752
16852     -0.02311      -0.09820             -0.33666         -0.93752
16853     -0.02311      -0.09820             -0.33666          1.06664
16854     -0.02311      -0.09820             -0.33666         -0.93752
16855     -0.02311      -0.09820             -0.33666          1.06664

       grade_8 Good  grade_9 Better
0          -0.64627        -0.33393
1          -0.64627        -0.33393
2          -0.64627        -0.33393
3           1.54735        -0.33393
4          -0.64627        -0.33393
...             ...             ...
16851       1.54735        -0.33393
16852       1.54735        -0.33393
16853      -0.64627        -0.33393
16854       1.54735        -0.33393
16855      -0.64627        -0.33393

[16856 rows x 31 columns]
```

Splitting our data for training and testing

In [182]:
```
df1.columns
```

Out[182]:
```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'sqft_above', 'sqft_basement', 'yr_built', 'lat', 'long',
       'sqft_living15', 'sqft_lot15', 'grading', 'waterfront_NO',
       'waterfront_YES', 'view_AVERAGE', 'view_EXCELLENT', 'view_FAIR',
       'view_GOOD', 'view_NONE', 'condition_Average', 'condition_Fair',
       'condition_Good', 'condition_Poor', 'condition_Very Good',
       'grade_10 Very Good', 'grade_11 Excellent', 'grade_4 Low',
       'grade_5 Fair', 'grade_6 Low Average', 'grade_7 Average',
       'grade_8 Good', 'grade_9 Better'],
      dtype='object')
```

In [183]:
```
X
```

Out[183]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above	yr_built	lat	long	sqft_
0	3	1.00000	1180	5650	1.00000	1180	1955	47.51120	-122.25700	
1	2	1.00000	770	10000	1.00000	770	1933	47.73790	-122.23300	
2	4	3.00000	1960	5000	1.00000	1050	1965	47.52080	-122.39300	
3	3	2.00000	1680	8080	1.00000	1680	1987	47.61680	-122.04500	
4	3	2.25000	1715	6819	2.00000	1715	1995	47.30970	-122.32700	
...	...	...	...	...	...	...	...	...	...	
16851	3	2.50000	1530	1131	3.00000	1530	2009	47.69930	-122.34600	
16852	4	2.50000	2310	5813	2.00000	2310	2014	47.51070	-122.36200	
16853	2	0.75000	1020	1350	2.00000	1020	2009	47.59440	-122.29900	
16854	3	2.50000	1600	2388	2.00000	1600	2004	47.53450	-122.06900	
16855	2	0.75000	1020	1076	2.00000	1020	2008	47.59410	-122.29900	

16856 rows × 31 columns

In [184]:
```
Y = df1['price']
```

In [185]:
```
Y
```

Out[185]:
```
0        221900.00000
1        180000.00000
2        604000.00000
3        510000.00000
4        257500.00000
             ...
16851    360000.00000
16852    400000.00000
16853    402101.00000
16854    400000.00000
16855    325000.00000
Name: price, Length: 16856, dtype: float64
```

In [186]:
```
from sklearn.model_selection import train_test_split


# we split our data into 80% train and 20% split

X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size = 0.20, random_sta
```

In [187]:
```
# the features of x_train

X_train
```

Out[187]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above	yr_built	lat	long	sqft_
2076	3	2.25000	1960	17126	1.00000	1400	1966	47.44360	-122.21000	
3951	5	1.75000	2120	8399	1.00000	1320	1942	47.76210	-122.33500	
4858	4	3.00000	3110	7231	2.00000	3110	1997	47.32790	-122.19100	
7713	3	2.50000	1600	1311	3.00000	1600	2005	47.69030	-122.39400	
2347	2	1.00000	960	4920	1.00000	960	1942	47.69460	-122.36200	
...	...	...	...	...	...	...	...	...	...	
2623	3	1.75000	1810	5733	1.00000	1010	1926	47.57090	-122.38800	
12363	4	2.50000	2220	5900	2.00000	2220	2014	47.69560	-122.36000	
5695	3	2.25000	1590	11745	1.00000	1090	1978	47.35530	-122.28000	
8006	3	1.50000	980	7770	1.00000	980	1968	47.69230	-122.20100	
13151	4	2.75000	1820	5400	1.00000	1220	1988	47.52040	-122.20500	

13484 rows × 31 columns

In [188]:
```
# The labels of X_train

Y_train
```

Out[188]:
```
2076     285750.00000
3951     377691.00000
4858     367500.00000
7713     465000.00000
2347     280000.00000
             ...
2623     515000.00000
12363    652600.00000
5695     190000.00000
8006     463800.00000
13151    430000.00000
Name: price, Length: 13484, dtype: float64
```

In [189]:
```
X_test
```

Out[189]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above	yr_built	lat	long	sqft_
7141	3	2.75000	1960	13252	1.00000	1240	1975	47.55820	-122.13900	
3703	3	1.75000	1960	5000	1.00000	980	1911	47.55760	-122.31700	
12646	3	2.50000	1640	7847	2.00000	1640	1987	47.56840	-122.01800	
11393	3	2.25000	2090	15000	1.00000	2090	1961	47.48850	-121.78300	
7477	3	1.75000	2590	8384	1.00000	1590	1971	47.77390	-122.19900	
...	...	...	...	...	...	...	...	...	...	
8744	4	1.75000	1650	7088	1.00000	1650	1973	47.56880	-122.08700	
60	4	2.00000	2750	7807	1.50000	2250	1916	47.71680	-122.28700	
4470	2	2.50000	1560	1222	2.00000	1080	2008	47.60400	-122.30700	
9187	2	1.00000	960	4000	1.00000	960	1918	47.55540	-122.26700	
8528	3	2.00000	2140	4923	1.00000	1070	1928	47.69020	-122.33900	

3372 rows × 31 columns

In [190]:
```
Y_test
```

Out[190]:
```
7141     550000.00000
3703     415000.00000
12646    480000.00000
11393    383000.00000
7477     485000.00000
             ...
8744     559500.00000
60       571000.00000
4470     435000.00000
9187     290000.00000
8528     710000.00000
Name: price, Length: 3372, dtype: float64
```

In [191]:
```
X_train_scaler = scaler.fit_transform(X_train)
X_train_scaler
```

Out[191]:
```
array([[-0.35862708,  0.38327608,  0.1294331 , ..., -0.93687401,
         1.54391996, -0.33093849],
       [ 2.19320806, -0.37478695,  0.36782357, ...,  1.06737938,
        -0.64770197, -0.33093849],
       [ 0.91729049,  1.52037063,  1.84286458, ..., -0.93687401,
         1.54391996, -0.33093849],
       ...,
       [-0.35862708,  0.38327608, -0.42184485, ...,  1.06737938,
        -0.64770197, -0.33093849],
       [-0.35862708, -0.75381846, -1.3307085 , ...,  1.06737938,
        -0.64770197, -0.33093849],
       [ 0.91729049,  1.14133912, -0.07915855, ...,  1.06737938,
        -0.64770197, -0.33093849]])
```

In [192]:
```
X_test_scaler = scaler.transform(X_test)
X_test_scaler
```

Out[192]:
```
array([[-0.35862708,  1.14133912,  0.1294331 , ..., -0.93687401,
         1.54391996, -0.33093849],
       [-0.35862708, -0.37478695,  0.1294331 , ..., -0.93687401,
        -0.64770197, -0.33093849],
       [-0.35862708,  0.7623076 , -0.34734783, ..., -0.93687401,
         1.54391996, -0.33093849],
       ...,
       [-1.63454465,  0.7623076 , -0.46654306, ..., -0.93687401,
         1.54391996, -0.33093849],
       [-1.63454465, -1.5118815 , -1.36050731, ..., -0.93687401,
        -0.64770197, -0.33093849],
       [-0.35862708,  0.00424457,  0.39762238, ..., -0.93687401,
         1.54391996, -0.33093849]])
```

Model training

In [193]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

In [194]:
```
lr.fit(X_train_scaler,Y_train) # fit is for the training the model
```

Out[194]:
```
LinearRegression()
```

In [195]:
```
# We now test out model to see how it predicts the prices of the houses

lr.predict(X_test_scaler) # predict is for the testing the model
```

Out[195]:
```
array([526206.38984857, 508958.38984857, 462273.38984857, ...,
       454766.38984857, 328864.38984857, 670614.38984857])
```

In [196]:
```
X_test
```

Out[196]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above	yr_built	lat	long	sqft_
7141	3	2.75000	1960	13252	1.00000	1240	1975	47.55820	-122.13900	
3703	3	1.75000	1960	5000	1.00000	980	1911	47.55760	-122.31700	
12646	3	2.50000	1640	7847	2.00000	1640	1987	47.56840	-122.01800	
11393	3	2.25000	2090	15000	1.00000	2090	1961	47.48850	-121.78300	
7477	3	1.75000	2590	8384	1.00000	1590	1971	47.77390	-122.19900	
...	...	...	...	...	...	...	...	...	...	
8744	4	1.75000	1650	7088	1.00000	1650	1973	47.56880	-122.08700	
60	4	2.00000	2750	7807	1.50000	2250	1916	47.71680	-122.28700	
4470	2	2.50000	1560	1222	2.00000	1080	2008	47.60400	-122.30700	
9187	2	1.00000	960	4000	1.00000	960	1918	47.55540	-122.26700	
8528	3	2.00000	2140	4923	1.00000	1070	1928	47.69020	-122.33900	

3372 rows × 31 columns

From our model, it shows that the 7141st house that is a 3 bedroom house of 1960 squarefoot, should be sold at USD 464804.23

In [197]:
```
# Y_pred is the model price

Y_pred = lr.predict(X_test_scaler)
```

In [198]:
```
# Y_test is the actual price from the data

Y_test
```

Out[198]:
```
7141     550000.00000
3703     415000.00000
12646    480000.00000
11393    383000.00000
7477     485000.00000
             ...
8744     559500.00000
60       571000.00000
4470     435000.00000
9187     290000.00000
8528     710000.00000
Name: price, Length: 3372, dtype: float64
```

From Y_test, we see that this house should be sold at USD 550,000.00

In [199]:
```
from sklearn.metrics import mean_absolute_error,r2_score

# computing the mean absolute error from the 3372 entries tested.
mean_absolute_error(Y_test,Y_pred)
```

Out[199]:
```
82730.23033799534
```

This result shows that the difference between the predicted value and the test value(actual data) is USD 82730.23 which is a significant difference

Fitting a Multiple regression model

In [200]:

```
#changing the variables to evaluate if it provides goodness of fit for the model
multiple_formula = 'price ~ X'
multiple_model = ols(formula = multiple_formula, data = df1).fit()
multiple_model_summary = multiple_model.summary()


multiple_model_summary
```

Out[200]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.691
Model:	OLS	**Adj. R-squared:**	0.690
Method:	Least Squares	**F-statistic:**	1393.
Date:	Sun, 07 Apr 2024	**Prob (F-statistic):**	0.00
Time:	17:03:28	**Log-Likelihood:**	-2.1945e+05
No. Observations:	16856	**AIC:**	4.389e+05
Df Residuals:	16828	**BIC:**	4.392e+05
Df Model:	27		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-8.354e+06	5.13e+05	-16.279	0.000	-9.36e+06	-7.35e+06
X[0]	-7509.5608	1419.856	-5.289	0.000	-1.03e+04	-4726.493
X[1]	2.003e+04	2238.944	8.945	0.000	1.56e+04	2.44e+04
X[2]	81.1639	3.197	25.388	0.000	74.898	87.430
X[3]	-2.0159	0.523	-3.853	0.000	-3.041	-0.990
X[4]	1.179e+04	2555.412	4.614	0.000	6782.148	1.68e+04
X[5]	2.7217	3.200	0.851	0.395	-3.550	8.993
X[6]	-1808.5333	46.434	-38.949	0.000	-1899.548	-1717.519
X[7]	5.32e+05	6475.032	82.159	0.000	5.19e+05	5.45e+05
X[8]	3.607e+04	7823.334	4.610	0.000	2.07e+04	5.14e+04
X[9]	50.1800	2.596	19.329	0.000	45.091	55.269
X[10]	-5.6472	0.592	-9.539	0.000	-6.808	-4.487
X[11]	-4.276e+06	2.57e+05	-16.650	0.000	-4.78e+06	-3.77e+06
X[12]	-4.078e+06	2.57e+05	-15.845	0.000	-4.58e+06	-3.57e+06
X[13]	-1.691e+06	1.03e+05	-16.462	0.000	-1.89e+06	-1.49e+06
X[14]	-1.574e+06	1.03e+05	-15.255	0.000	-1.78e+06	-1.37e+06
X[15]	-1.672e+06	1.03e+05	-16.213	0.000	-1.87e+06	-1.47e+06
X[16]	-1.67e+06	1.03e+05	-16.224	0.000	-1.87e+06	-1.47e+06
X[17]	-1.748e+06	1.02e+05	-17.052	0.000	-1.95e+06	-1.55e+06
X[18]	-1.668e+06	1.03e+05	-16.226	0.000	-1.87e+06	-1.47e+06
X[19]	-1.691e+06	1.03e+05	-16.401	0.000	-1.89e+06	-1.49e+06
X[20]	-1.633e+06	1.03e+05	-15.914	0.000	-1.83e+06	-1.43e+06
X[21]	-1.752e+06	1.05e+05	-16.638	0.000	-1.96e+06	-1.55e+06
X[22]	-1.61e+06	1.03e+05	-15.700	0.000	-1.81e+06	-1.41e+06
X[23]	-8.596e+05	6.53e+04	-13.157	0.000	-9.88e+05	-7.32e+05
X[24]	-7.883e+05	6.76e+04	-11.653	0.000	-9.21e+05	-6.56e+05
X[25]	-1.257e+06	7.05e+04	-17.839	0.000	-1.4e+06	-1.12e+06
X[26]	-1.205e+06	6.39e+04	-18.872	0.000	-1.33e+06	-1.08e+06
X[27]	-1.171e+06	6.41e+04	-18.266	0.000	-1.3e+06	-1.05e+06
X[28]	-1.109e+06	6.44e+04	-17.219	0.000	-1.24e+06	-9.83e+05
X[29]	-1.034e+06	6.47e+04	-15.978	0.000	-1.16e+06	-9.07e+05
X[30]	-9.298e+05	6.5e+04	-14.297	0.000	-1.06e+06	-8.02e+05

Omnibus:	1910.863	**Durbin-Watson:**	1.981
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	4037.990

Skew:	0.710	**Prob(JB):**	0.00
Kurtosis:	4.932	**Cond. No.**	1.08e+16

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.92e-20. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [201]:

```
#changing the variables to evaluate if it provides goodness of fit for the model
multiple_formula1 = 'price ~ sqft_living + sqft_above + sqft_living15 + bathrooms + bed
multiple_model1 = ols(formula = multiple_formula1, data = df1).fit()
multiple_model1_summary = multiple_model1.summary()


multiple_model1_summary
```

Out[201]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.487
Model:	OLS	**Adj. R-squared:**	0.487
Method:	Least Squares	**F-statistic:**	941.8
Date:	Sun, 07 Apr 2024	**Prob (F-statistic):**	0.00
Time:	17:03:28	**Log-Likelihood:**	-2.2371e+05
No. Observations:	16856	**AIC:**	4.475e+05
Df Residuals:	16838	**BIC:**	4.476e+05
Df Model:	17		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	2.997e+06	6.21e+04	48.240	0.000	2.88e+06	3.12e+06
grading[T.Subpar]	-9.224e+04	3900.261	-23.650	0.000	-9.99e+04	-8.46e+04
grading[T.Superior]	1.327e+05	7389.518	17.958	0.000	1.18e+05	1.47e+05
sqft_living	133.6416	4.008	33.340	0.000	125.785	141.499
sqft_above	-31.4381	3.870	-8.123	0.000	-39.024	-23.852
sqft_living15	82.4915	3.159	26.111	0.000	76.299	88.684
bathrooms	2.828e+04	2859.302	9.890	0.000	2.27e+04	3.39e+04
bedrooms	-3.133e+04	1791.917	-17.483	0.000	-3.48e+04	-2.78e+04
floors	8.173e+04	2842.114	28.755	0.000	7.62e+04	8.73e+04
yr_built	-2490.3296	54.420	-45.761	0.000	-2597.000	-2383.660
waterfront_NO	1.455e+06	3.4e+04	42.793	0.000	1.39e+06	1.52e+06
waterfront_YES	1.542e+06	4.05e+04	38.106	0.000	1.46e+06	1.62e+06
view_AVERAGE	5.836e+05	1.4e+04	41.646	0.000	5.56e+05	6.11e+05
view_EXCELLENT	6.979e+05	1.85e+04	37.680	0.000	6.62e+05	7.34e+05
view_FAIR	5.915e+05	1.53e+04	38.722	0.000	5.62e+05	6.21e+05
view_GOOD	5.975e+05	1.54e+04	38.909	0.000	5.67e+05	6.28e+05
view_NONE	5.266e+05	1.38e+04	38.185	0.000	5e+05	5.54e+05
condition_Average	-4.204e+04	4334.829	-9.699	0.000	-5.05e+04	-3.35e+04
condition_Fair	-9.319e+04	1.37e+04	-6.827	0.000	-1.2e+05	-6.64e+04
condition_Good	-3.051e+04	4407.844	-6.921	0.000	-3.91e+04	-2.19e+04

Omnibus:	791.660	**Durbin-Watson:**	1.972
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	999.671
Skew:	0.488	**Prob(JB):**	8.40e-218
Kurtosis:	3.685	**Cond. No.**	5.24e+19

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 8.68e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [202]:

```
#changing the variables to evaluate if it provides goodness of fit for the model

multiple_formula2 = 'price ~ sqft_living + sqft_above + sqft_living15 + bathrooms + bed
multiple_model2 = ols(formula = multiple_formula2, data = df1).fit()
multiple_model2_summary = multiple_model2.summary()


multiple_model2_summary
```

Out[202]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.608
Model:	OLS	**Adj. R-squared:**	0.608
Method:	Least Squares	**F-statistic:**	2906.
Date:	Sun, 07 Apr 2024	**Prob (F-statistic):**	0.00
Time:	17:03:28	**Log-Likelihood:**	-2.2144e+05
No. Observations:	16856	**AIC:**	4.429e+05
Df Residuals:	16846	**BIC:**	4.430e+05
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-3.126e+07	1.11e+06	-28.101	0.000	-3.34e+07	-2.91e+07
sqft_living	123.2762	3.506	35.157	0.000	116.403	130.149
sqft_above	1.4104	3.453	0.409	0.683	-5.357	8.178
sqft_living15	88.5361	2.734	32.384	0.000	83.177	93.895
bathrooms	3.47e+04	2470.790	14.045	0.000	2.99e+04	3.95e+04
bedrooms	-2.249e+04	1557.751	-14.437	0.000	-2.55e+04	-1.94e+04
floors	4.638e+04	2548.560	18.198	0.000	4.14e+04	5.14e+04
lat	5.824e+05	7109.393	81.914	0.000	5.68e+05	5.96e+05
long	-5.653e+04	8512.904	-6.641	0.000	-7.32e+04	-3.98e+04
yr_built	-1701.1904	47.212	-36.033	0.000	-1793.730	-1608.650

Omnibus:	1383.029	**Durbin-Watson:**	1.977
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2225.827
Skew:	0.625	**Prob(JB):**	0.00
Kurtosis:	4.268	**Cond. No.**	4.43e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.43e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

The multiple regression model with an R squared value of 69.1% demonstrates substantially improved explanatory power compared to the simple linear regression model with an R squared value of 32.7%. The inclusion of additional independent variables in the multiple regression model has significantly enhanced the model's ability to explain the variability in the price. The R squared values and the comparison between the two models highlight the importance of considering multiple factors and variables in regression analysis to develop a more comprehensive and accurate understanding of the relationship between the independent variables and the price.