Final Project Submission

Please fill out:

- Student name:Cynthia Chiuri
- Student pace: part time
- Scheduled project review date/time:
- Instructor name: Noah Kandie
- Blog post URL:https:https://datascience942.wordpress.com/2023/06/02/predicting-housing-prices-insights-and-recommendations-for-homeowners/ (https://datascience942.wordpress.com/2023/06/02/predicting-housing-prices-insights-and-recommendations-for-homeowners/)
- GROUP 8

Housing Image

Column Names and Descriptions for King County Data Set

- `id` - Unique identifier for a house
- `date` - Date house was sold
- `price` - Sale price (prediction target)
- `bedrooms` - Number of bedrooms
- `bathrooms` - Number of bathrooms
- `sqft_living` - Square footage of living space in the home
- `sqft_lot` - Square footage of the lot
- `floors` - Number of floors (levels) in house
- `waterfront` - Whether the house is on a waterfront
  - Includes Duwamish, Elliott Bay, Puget Sound, Lake Union, Ship Canal, Lake Washington, Lake Sammamish, other lake, and river/slough waterfronts
- `view` - Quality of view from house
  - Includes views of Mt. Rainier, Olympics, Cascades, Territorial, Seattle Skyline, Puget Sound, Lake Washington, Lake Sammamish, small lake / river / creek, and other
- `condition` - How good the overall condition of the house is. Related to maintenance of house.
  - See the King County Assessor Website (https://info.kingcounty.gov/assessor/esales/Glossary.aspx?type=r) for further explanation of each condition code
- `grade` - Overall grade of the house. Related to the construction and design of the house.
  - See the King County Assessor Website (https://info.kingcounty.gov/assessor/esales/Glossary.aspx?type=r) for further explanation of each building grade code
- `sqft_above` - Square footage of house apart from basement
- `sqft_basement` - Square footage of the basement

- `yr_built` - Year when house was built
- `yr_renovated` - Year when house was renovated
- `zipcode` - ZIP Code used by the United States Postal Service
- `lat` - Latitude coordinate
- `long` - Longitude coordinate
- `sqft_living15` - The square footage of interior housing living space for the nearest 15 neighbors
- `sqft_lot15` - The square footage of the land lots of the nearest 15 neighbors

Predictive analysis of House prices in King County

Renovations: Worth the Investment or a Risky Gamble?

Overview

This project uses linear regression analysis to infer how certain variables impact housing prices and by how much. The aim is to gain insights and make predictions about the factors that affect house sales in King County area as well as lucrative neighbourhoods to invest in while using statistical techniques to support relevant recommendations.

Business problem

The real estate agency wants to provide homeowners with advice on how home renovations can potentially increase the estimated value of their homes and by what amount. The agency aims to offer valuable insights to homeowners, helping them make informed decisions about renovation projects that can maximize their return on investment when selling their properties.

Business objectives

The analysis aims to answer below questions in trying to predict the prices;

1. To determine how much would adding an extension to the lot area of the home likely increase sale price?

2. To examine how much would adding an additional bathroom likely increase sale price?

3. To determine how much would adding an extension to the living area of the home likely increase sale price?

4. To examine how much would adding an additional floor to a house likely increase sale price?

Metric of Success

Our metric of success will be the R-Squared and the Root Mean Square of Errors(RMSE). This will be the final step in evaluating the performance of the model by doing a train-test split, which will give us an idea of how the model would perform with new data for the same variables that the model will be trained on, and another set that it will be tested on. By default, the function takes 80% of the data as the training subset and the other 20% as its test subset.

Data understanding

The dataset used for predicting the sales price of houses in King County is found in `kc_house_data.csv` . It comprises 21,597 observations and consists of 20 house features along with a column indicating the house price. The data covers homes sold between May 2014 and May 2015. Out of the 20 features, eight are continuous numerical variables that provide information about the area dimensions and geographical location of the house. These variables offer a general overview of the house's structure and characteristics. The remaining attributes are discrete variables, which offer more detailed information about specific components of the house. The discrete variables include quantifications of various items within the house, such as the number of bedrooms, bathrooms, presence of a waterfront, and floor level. Some attributes also provide background information about the house, such as the year of construction, year of innovation, previous selling price, and date of sale.

Importing the relevant libraries and loading the dataset from `kc_house_data.csv` .

In [77]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
import statsmodels
import math
import matplotlib.pyplot as plt
from scipy.special import logsumexp
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
import sklearn.metrics as metrics
from scipy import stats as stats
from statsmodels.stats.outliers_influence import variance_inflation_factor
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from statsmodels.formula.api import ols
```

In [2]:
```
# displaying the DataFrame
df = pd.read_csv("data/kc_house_data.csv")
df
```

Out[2]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	
...	...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	2.0	

21597 rows × 21 columns

In [3]:
```
#Checking on the columns in our dataset
df.columns
```

Out[3]:
```
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 [4]:
```
# checking the number of rows and columns
df.shape
```

Out[4]:
```
(21597, 21)
```

We have 21,597 rows of data, meaning we have information about 21,597 homes. That is plenty of data with which to build a model. However, not every row has complete information about a given home, such as `yr_renovated` having fewer than 21,597 records.

In [5]:
```
# checking the summary statistics
df.describe()
```

Out[5]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	215
mean	4.580474e+09	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	
std	2.876736e+09	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	
25%	2.123049e+09	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	

This gives us a great overview of the data we have. A few key takeaways are:

- Homes are priced between 78,000 and 7,700,000 dollars
- Most homes are between 322,000 and 645,000 dollars
- The avereage home has 3.3 bedrooms and 2.1 bathrooms, with about 2,080 living square footage
- All homes have between 1 and 3.5 floors
- The average home was built around 1971, but some are over 100 years old
- We noticed that there is a home listed as having 33 bedrooms. Either that's an extreme outlier, or some sort of input error. We will investigate that later.

So now that we have a basic understanding of the data we're working with, we can dive into some more information that we will need in order to build a model later. By using the `.info()` method, we can pull up information about missing data values, how many rows of data we have, and whether values are being read as text or as numerical data.

In [6]:
```
# checking the metadata of our data
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
```

From the metadata, not every row has complete information about a given home, such as `yr_renovated` having fewer than 21,597 entries.

Furthermore, not all columns of data are being read as quantitative data. In this case, some columns are being read as numbers, whether that's in integer form or float (numbers with decimals) form, while others are being read as text inputs, or objects.

It looks like we'll have to convert some columns with qualitative data (such as view, waterfront, and condition) into integers or floats so we can build models with them. We'll also have to replace null values for the waterfront, view, and yr_renovated columns.

In [7]:
```
# checking for the total number of null values per column
df.isna().sum()
```

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

Based on the dataset waterfront, view and yr_renovated have the summation of 2,376, 63 and 3,842 null values respectively.

In [8]:
```
# dropping null values
df.dropna(inplace=True)
```

In [9]:
```
# checking if the null values are successfully droped.
df.isna().sum()
```

Out[9]:
```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

In [10]:
```
# checking on duplicated values in id column.
duplicated=df["id"].duplicated().sum()
duplicated
```

Out[10]:
```
86
```

This shows that there are 86 duplicates in the `id` column. This is equivalent to 86 houses from the the dataset. Dropping the mentioned number may not skew the dataset.

In [11]:
```
# dropping the duplicates
df.drop_duplicates(subset='id', inplace=True)
```

In [12]:
```
# confirming that the duplicates have been dropped successfully
duplicated=df["id"].duplicated().sum()
duplicated
```

Out[12]:
```
0
```

Exploratory Data Analysis (EDA)

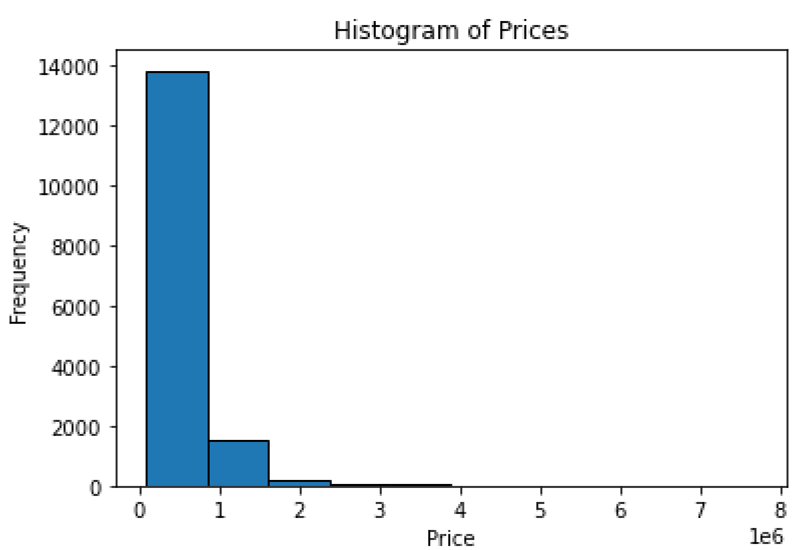
Univariate analysis

The stage involve exploration process, which involves generating and plotting histograms and box plots. This crucial step allows us to gain insight into the distribution patterns of the data for each variable. By visualizing the histograms, we can better comprehend the spread and frequency of values within each variable, providing a foundation for further analysis. Box plots help us identify potential outliers.

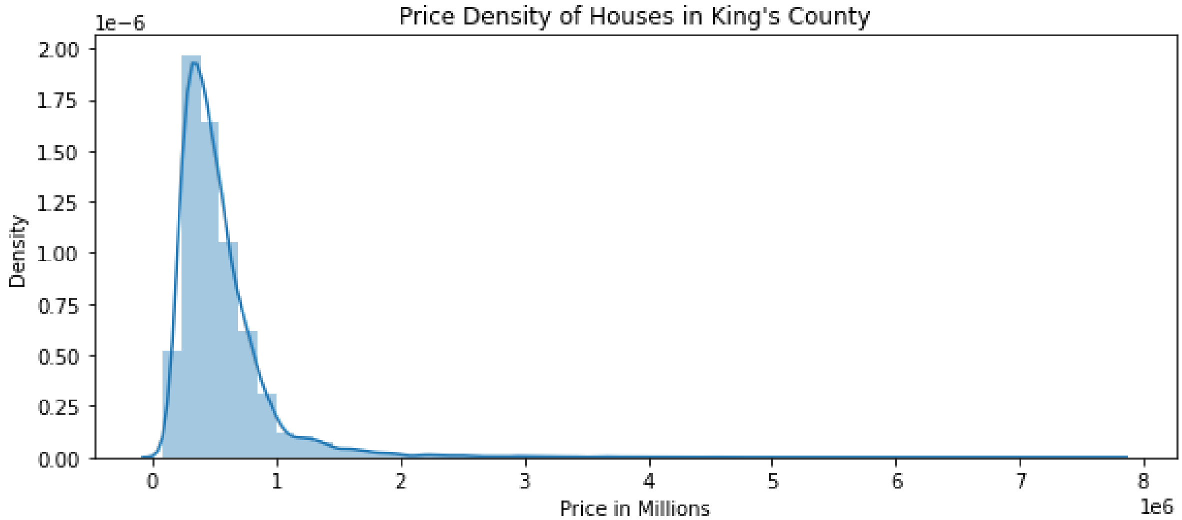
In [13]:
```
# Checking on measures of central tendency and dispersion

price_mean = df["price"].mean()
price_mode = df["price"].mode()[0]
price_median = df["price"].median()
price_std = df["price"].std()

print("Mean:", price_mean)
print("Mode:", price_mode)
print("Median:", price_median)
print("Standard Deviation:", price_std)
```

```
Mean: 541492.6832737944
Mode: 350000.0
Median: 450000.0
Standard Deviation: 372603.68455896684
```

In [14]:
```
# Plotting a histogram of price

plt.hist(df["price"], bins=10, edgecolor='black')
plt.xlabel("Price")
plt.ylabel("Frequency")
plt.title("Histogram of Prices")
plt.show()
```

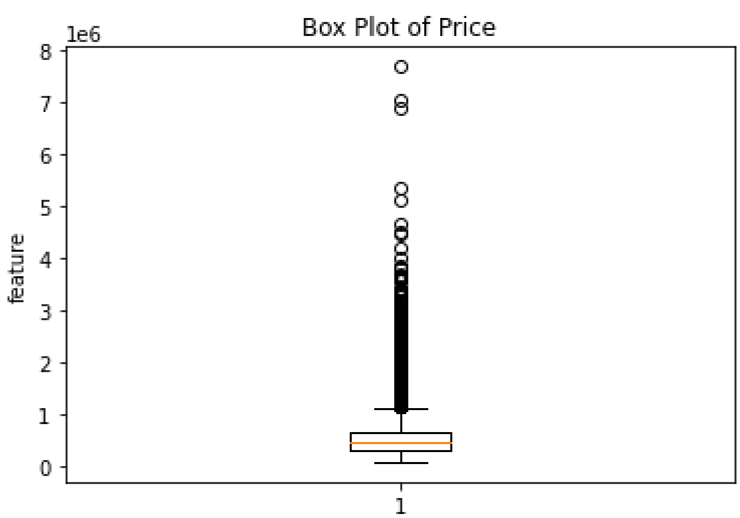


In [15]:
```
# Plotting a histogram/kernel density estimate of price

plt.figure(figsize=(10,4))

price_dist = sns.distplot(df["price"])
price_dist.set(xlabel="Price in Millions", title="Price Density of Houses in Ki
plt.show()
```



As we can see, the distribution of house prices is right-skewed. This means that there are a large number of houses that are relatively inexpensive, but there are also a small number of houses that are very expensive.

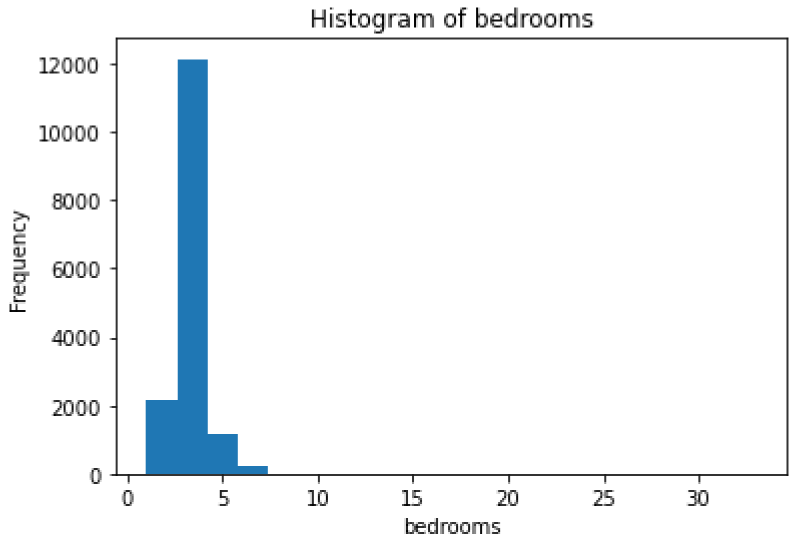
In [16]:
```
# Checking on outliers in the price variable

plt.figure()
plt.boxplot(df['price'])
plt.ylabel("feature")
plt.title('Box Plot of Price')
plt.show()
```

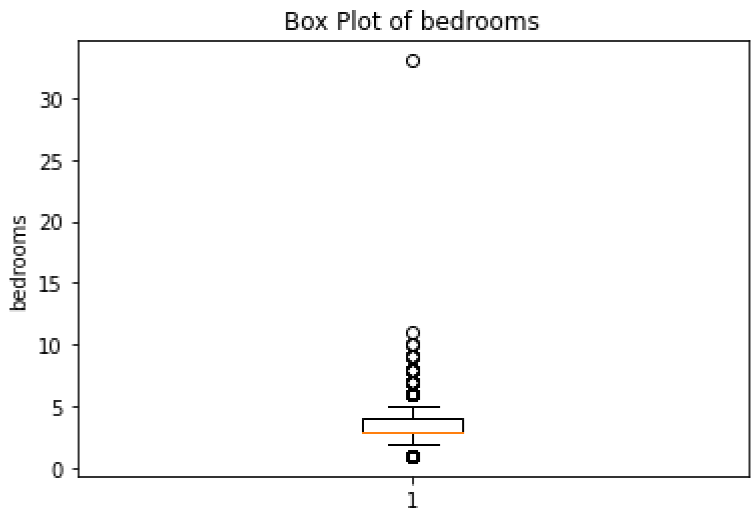


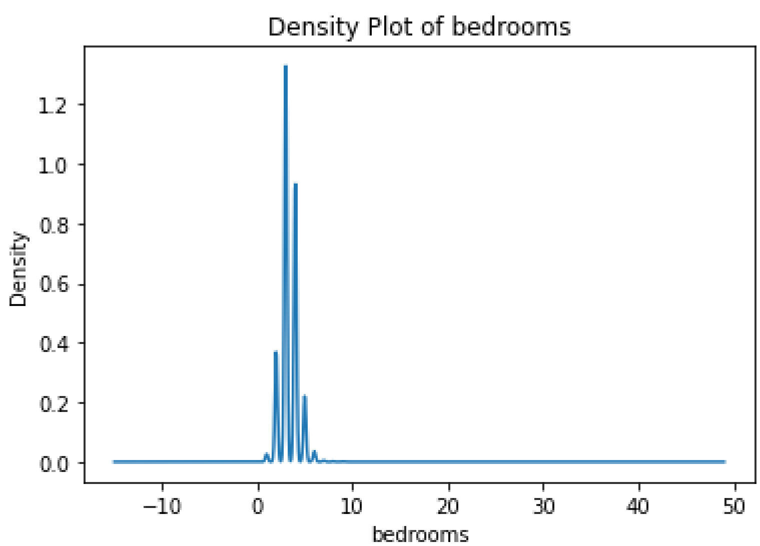
Based on the box plot there is presence of outliers but we decided to keep them based on the assumption that they are a true representation of the real-world dataset.

In [17]:
```
# Plotting Histogram, density plots and box plot

# Select the desired features
features = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors','zipc
plt.figure(figsize=(12, 8))
ncols=3
nrows=4
# Perform univariate analysis for each feature
for feature in features:
    # Descriptive Statistics
    print('Descriptive Statistics for', feature)
    print(df[feature].describe())
    print()
    # Histogram
    plt.figure()
    plt.hist(df[feature], bins=20)
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.title('Histogram of ' + feature)
    plt.show()
    # Box Plot
    plt.figure()
    plt.boxplot(df[feature])
    plt.ylabel(feature)
    plt.title('Box Plot of ' + feature)
    plt.show()
    # Density Plot
    plt.figure()
    df[feature].plot(kind='density')
    plt.xlabel(feature)
    plt.ylabel('Density')
    plt.title('Density Plot of ' + feature)
    plt.show()
```

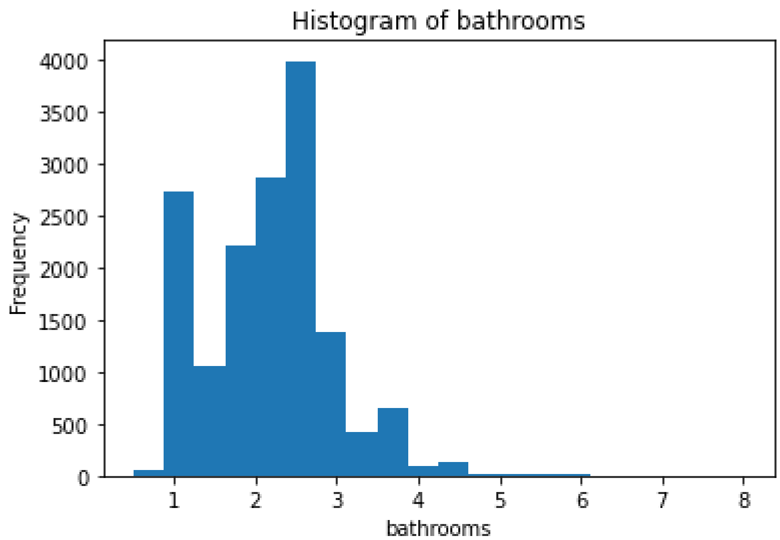
```
Descriptive Statistics for bedrooms
count    15676.000000
mean         3.379434
std          0.935193
min          1.000000
25%          3.000000
50%          3.000000
75%          4.000000
max         33.000000
Name: bedrooms, dtype: float64


<Figure size 864x576 with 0 Axes>
```

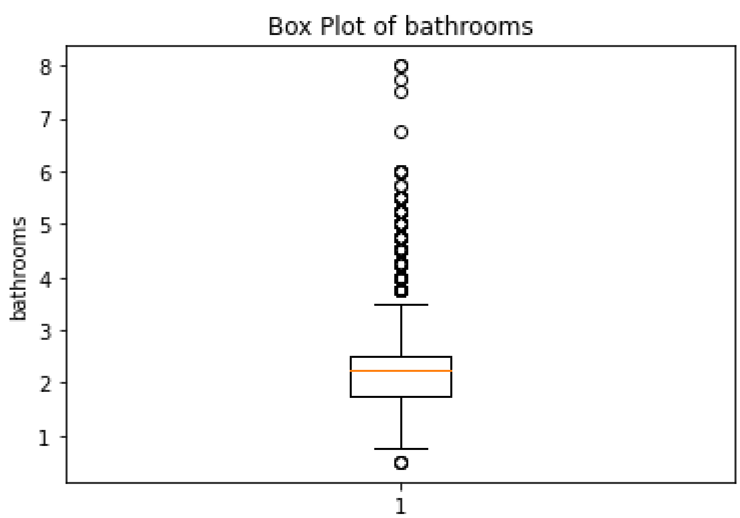

Histogram of bedrooms
12000
10000
8000
6000
4000
2000
0
Frequency
0
5
10
15
20
25
30
bedrooms

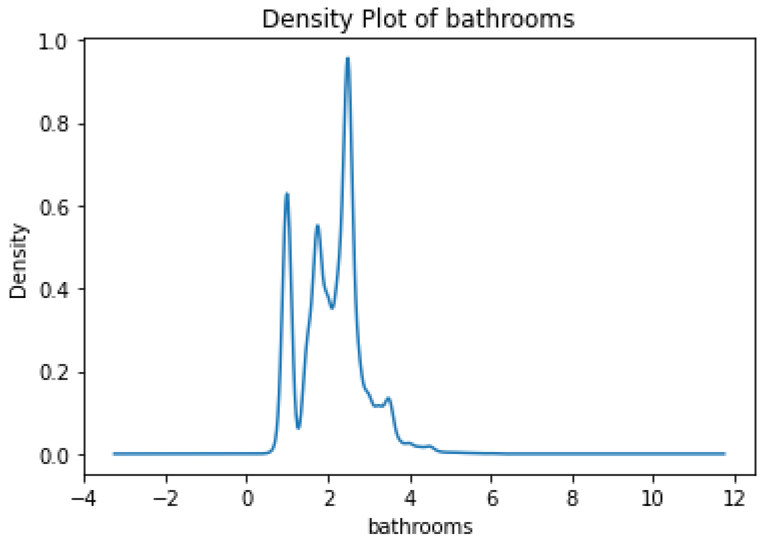

Box Plot of bedrooms
30
25
20
15
10
5
0
bedrooms
1


Density Plot of bedrooms
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Density
−10
0
10
20
30
40
50
bedrooms

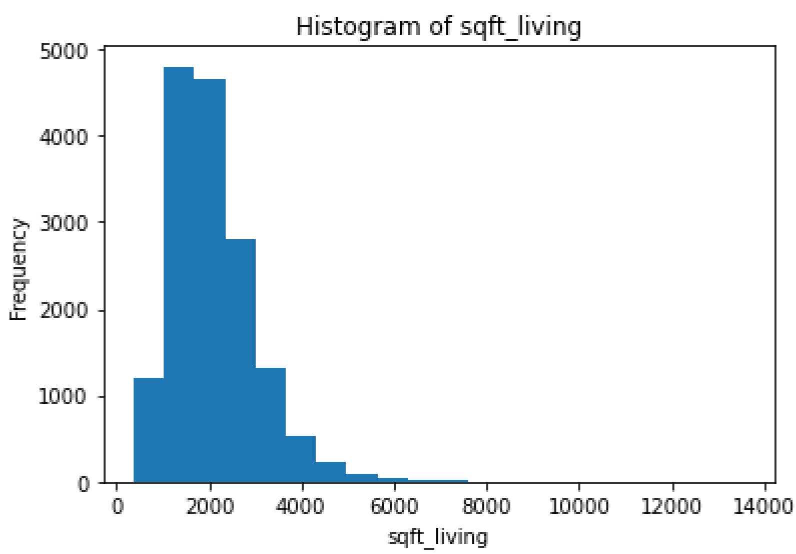
```
Descriptive Statistics for bathrooms
count    15676.000000
mean         2.122066
std          0.766735
min          0.500000
25%          1.750000
50%          2.250000
75%          2.500000
max          8.000000
Name: bathrooms, dtype: float64
```

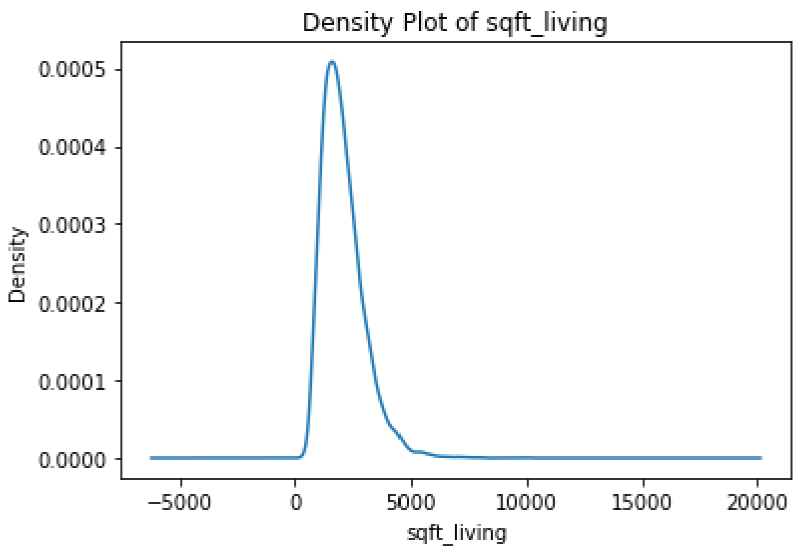




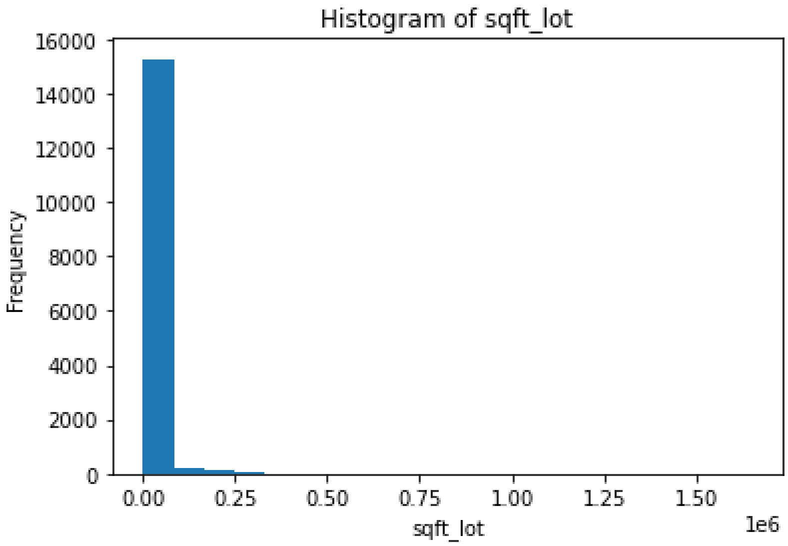


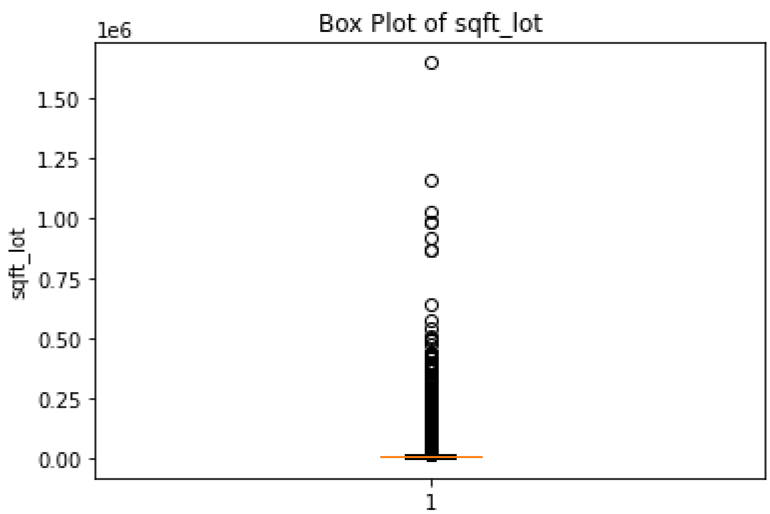

```
Descriptive Statistics for sqft_living
count    15676.000000
mean      2086.057285
std        918.753332
min        370.000000
25%       1430.000000
50%       1920.000000
75%       2550.000000
max      13540.000000
Name: sqft_living, dtype: float64
```

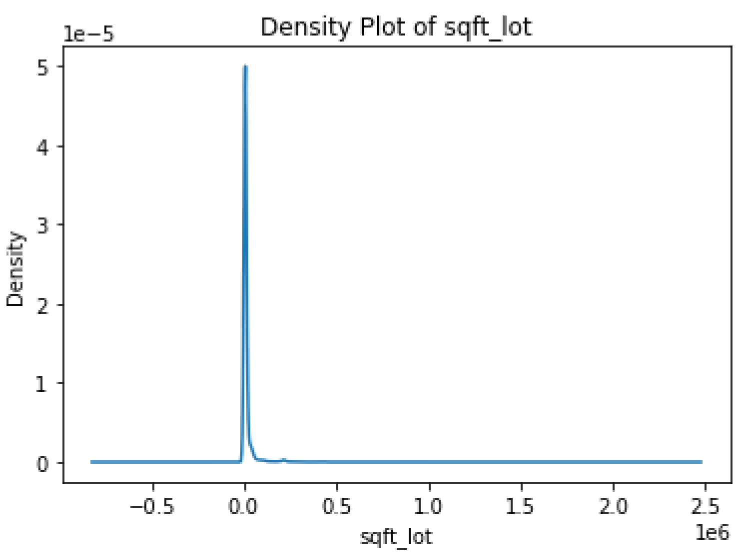




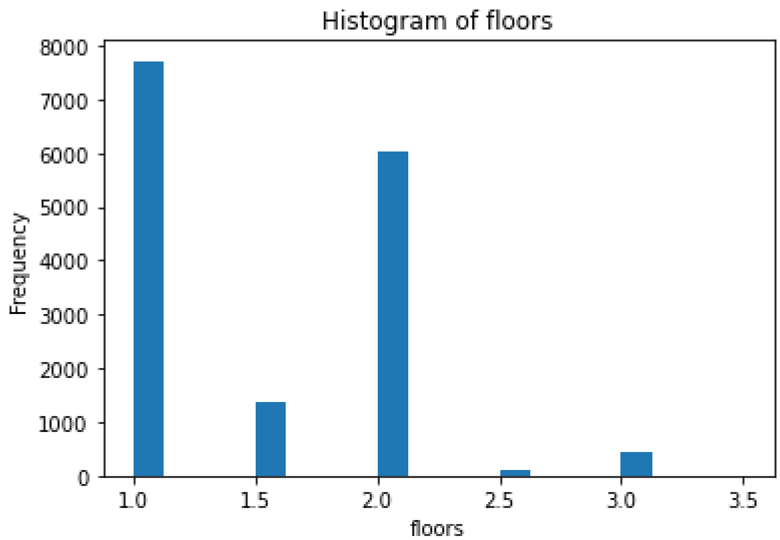

```
Descriptive Statistics for sqft_lot
count    1.567600e+04
mean     1.529400e+04
std      4.189635e+04
min      5.200000e+02
25%      5.045250e+03
50%      7.600000e+03
75%      1.071700e+04
max      1.651359e+06
Name: sqft_lot, dtype: float64
```

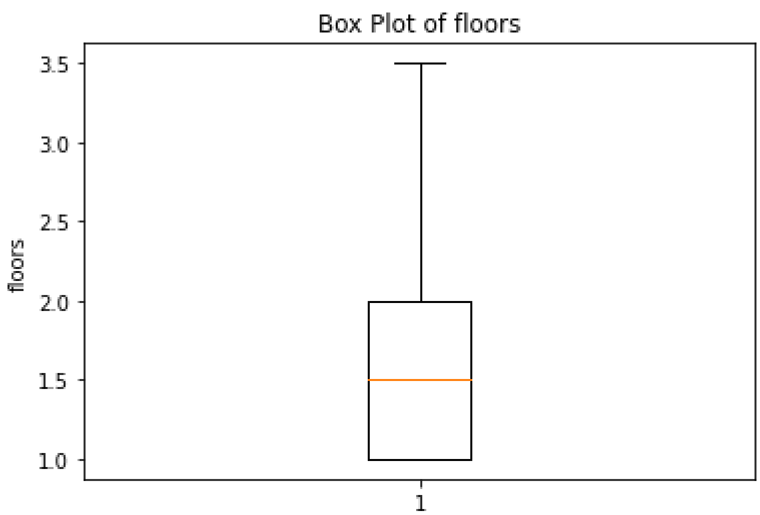

Histogram of sqft_lot
Frequency
sqft_lot
1e6
0
2000
4000
6000
8000
10000
12000
14000
16000
0.00
0.25
0.50
0.75
1.00
1.25
1.50

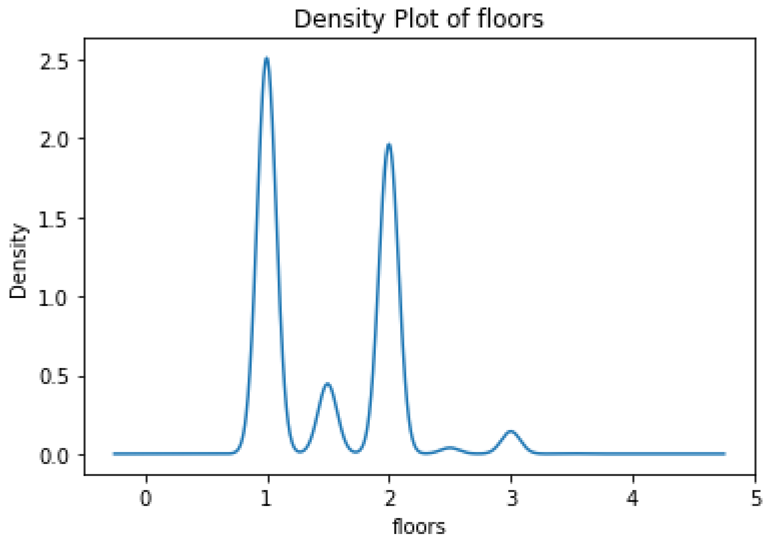

Box Plot of sqft_lot
1e6
sqft_lot
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1


Density Plot of sqft_lot
1e−5
Density
sqft_lot
1e6
0
1
2
3
4
5
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

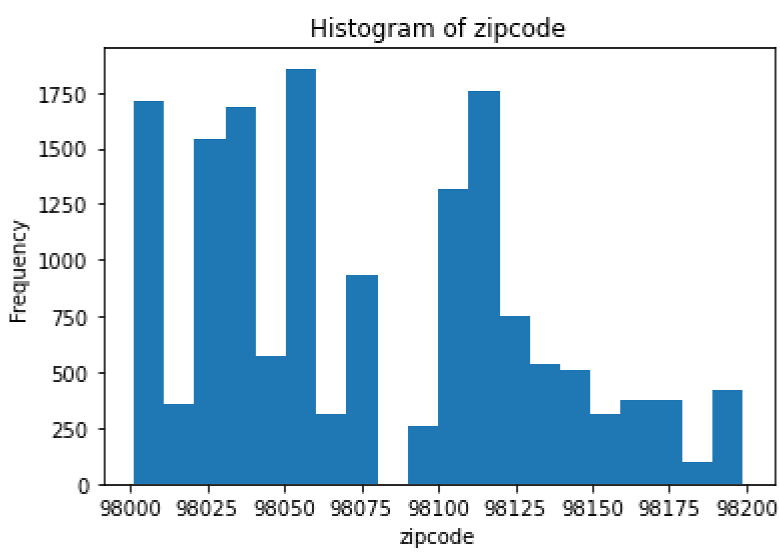
```
Descriptive Statistics for floors
count    15676.000000
mean         1.496587
std          0.539689
min          1.000000
25%          1.000000
50%          1.500000
75%          2.000000
max          3.500000
Name: floors, dtype: float64
```

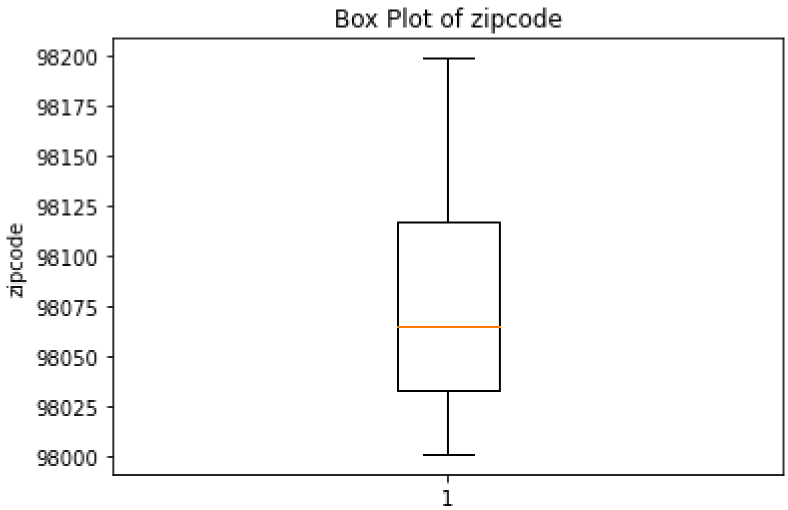


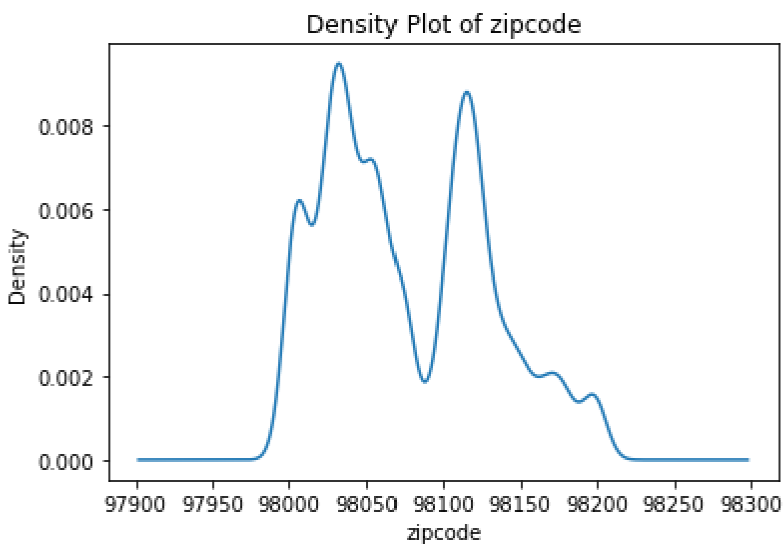




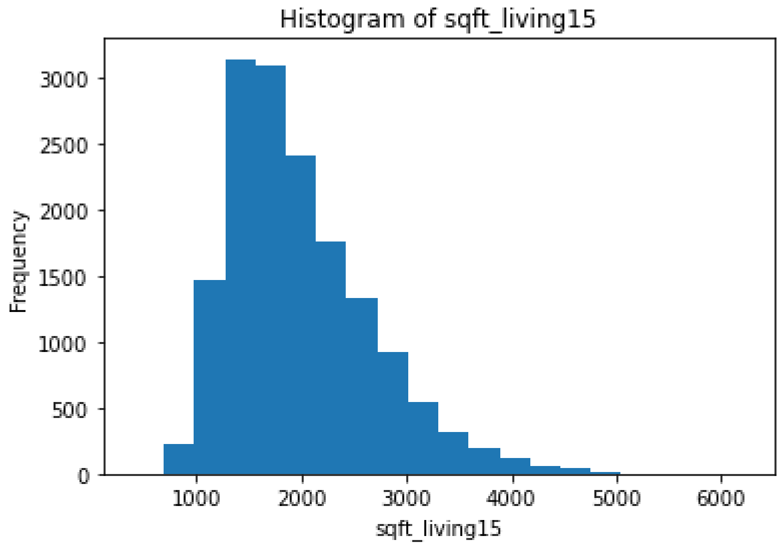

```
Descriptive Statistics for zipcode
count    15676.000000
mean     98077.487114
std         53.366170
min      98001.000000
25%      98033.000000
50%      98065.000000
75%      98117.000000
max      98199.000000
Name: zipcode, dtype: float64
```

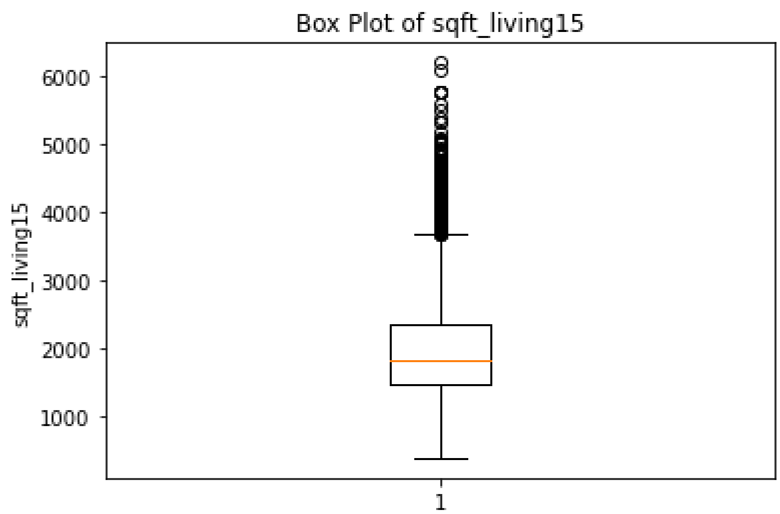


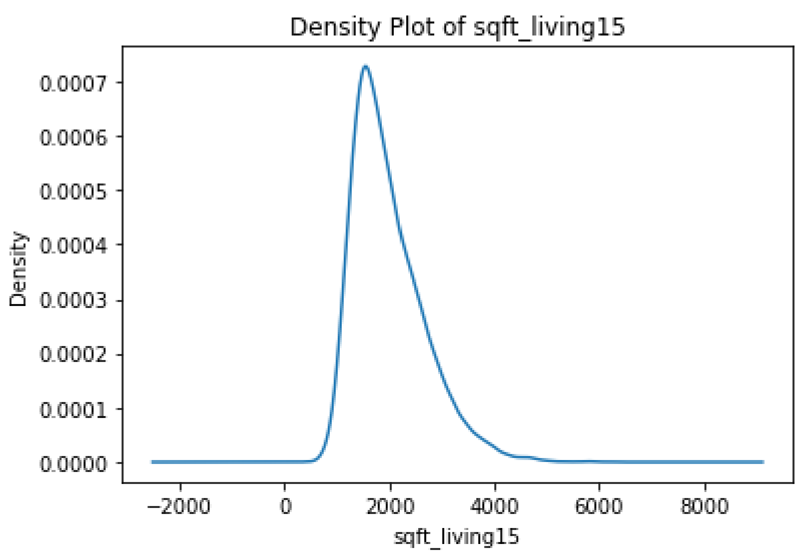




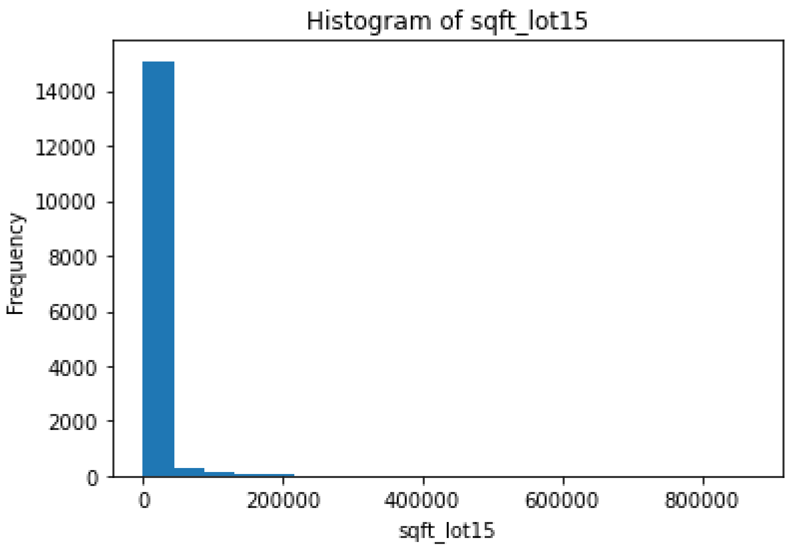

```
Descriptive Statistics for sqft_living15
count    15676.000000
mean      1991.289168
std        684.179299
min        399.000000
25%       1490.000000
50%       1850.000000
75%       2370.000000
max       6210.000000
Name: sqft_living15, dtype: float64
```

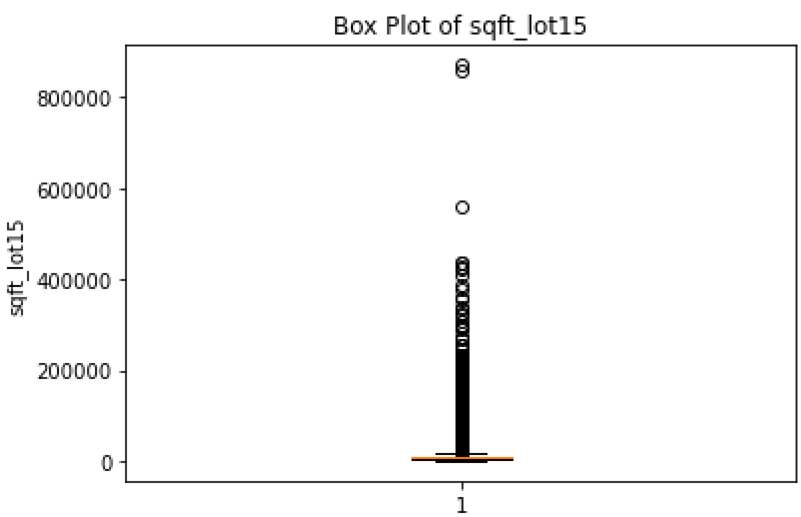

Histogram of sqft_living15
Frequency
sqft_living15
0
500
1000
1500
2000
2500
3000
1000
2000
3000
4000
5000
6000

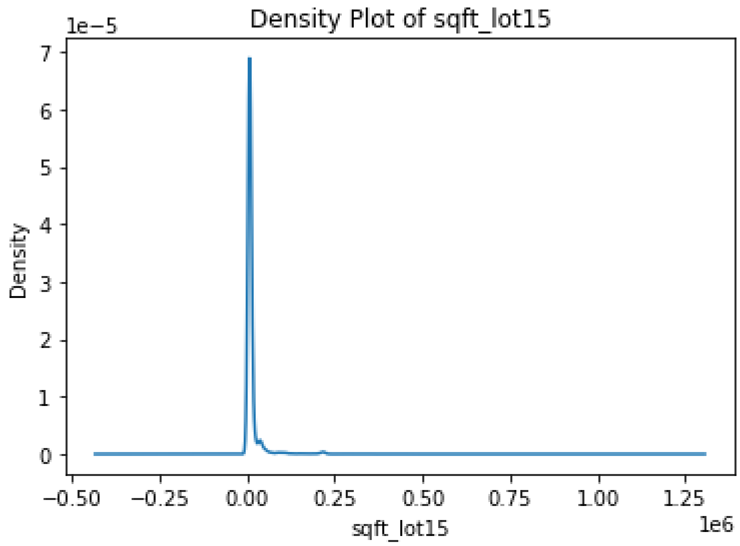

Box Plot of sqft_living15
sqft_living15
1000
2000
3000
4000
5000
6000
1


Density Plot of sqft_living15
Density
sqft_living15
0.0000
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006
0.0007
−2000
0
2000
4000
6000
8000

```
Descriptive Statistics for sqft_lot15
count     15676.000000
mean      12911.040125
std       28037.170327
min         659.000000
25%        5100.000000
50%        7620.000000
75%       10102.250000
max      871200.000000
Name: sqft_lot15, dtype: float64
```







Descriptive Statistics

In [18]:
```
# To acertain the meadian points of the dataset

df[['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'sqft_living
```

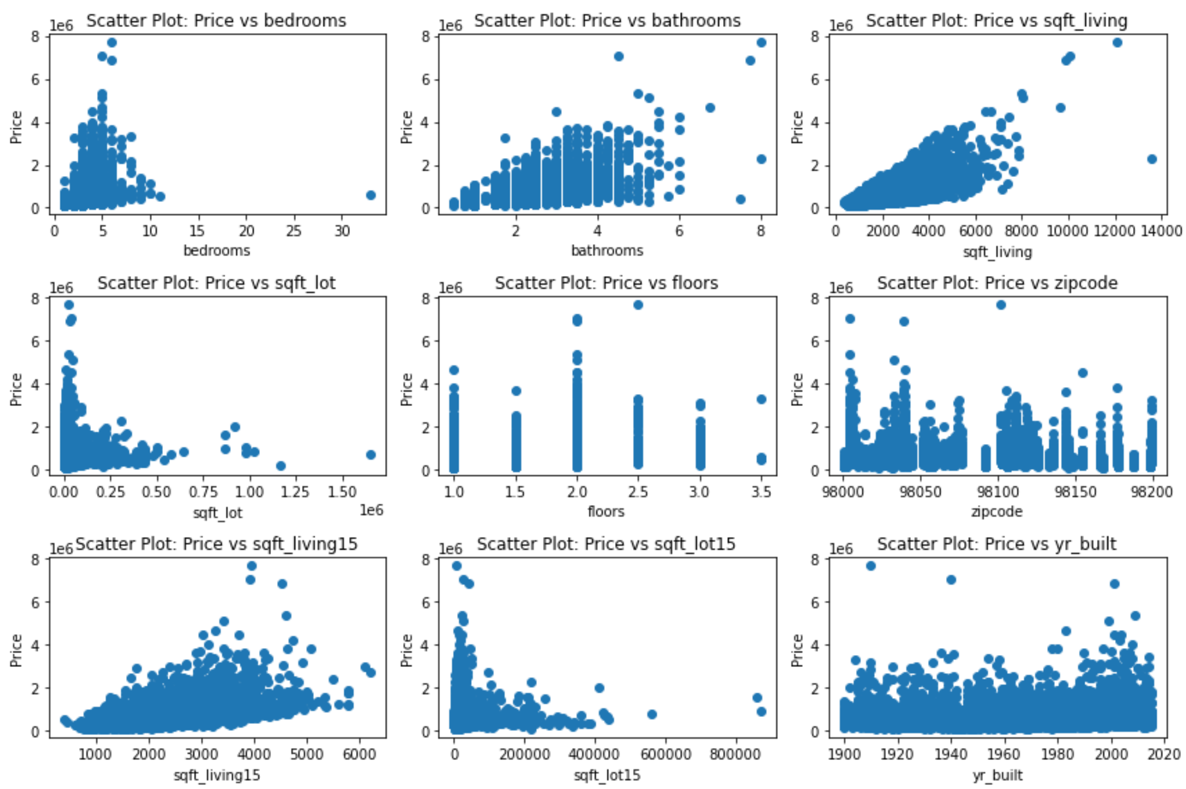
Out[18]:
```
bedrooms           3.00
bathrooms          2.25
sqft_living     1920.00
sqft_lot        7600.00
floors             1.50
sqft_living15   1850.00
sqft_lot15      7620.00
dtype: float64
```

- Based on the above plots,bedrooms shows out of 15,676 counts the mean mean is 3.37, std of 0.935 and the median of 3.0. This shows most of the houses have 3 or 4 bedrooms with an exception of an outlier which exist in the dataset.The dataset is also distributed uniformly around the mean.
- Bathrooms has a mean of 2.12, median of 2.25 and std of 0.77. The dataset is rightly skewed and the dataset is distributed around the mean.Most of the houses have 2 bathrooms.
- Sqft_living shows that it has a mean of 2080, std of 918.1 and median of 1910 depicting that the data is distributed around the mean with slight deviations. most of the houses covers 2080 square feets space.
- Sqft_lot has rightly skewed dataset with the presence of outlier. It has a mean of 15,099.41 feets, median of 7,618 which shows that few data points are aound the mean.
- Sqft_living15 dataset shows that the dataset has mean of 1,986 feats of living space, median of 1,840 which shows that most of the houses have living space of 1,986 feets and since most of the datapoints are distributed around the mean with a deviation of 685 feets only.
- Descriptive Statistics for floors depicts that relatively few houses have 1 to 2 foors. The mean is 1.5, median of 1.5. the data points are scattered with most points a 1 and 2 based on the density curve.

Bivariate Analysis

Bivariate analysis focuses on determining the correlation between two variables. At this stage we will use a heatmap and scatterplot to check on correlation and collinearity of the variables.

In [19]:
```
features = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
            'zipcode', 'sqft_living15','sqft_lot15','yr_built']

# Set the figure size and grid layout
fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(12, 8))

# Perform bivariate analysis for each feature
for i, feature in enumerate(features):
    # Calculate the row and column index
    row = i // 3
    col = i % 3

    # Scatter Plot
    axs[row, col].scatter(df[feature], df['price'])
    axs[row, col].set_xlabel(feature)
    axs[row, col].set_ylabel('Price')
    axs[row, col].set_title('Scatter Plot: Price vs ' + feature)

# Adjust the spacing between subplots
plt.tight_layout()

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



- Square foot of living has a STRONG correlation with price; we can assume that as the square foot of living increases, so does price.
- Square foot of lot has a high number of 0's. What does this mean? Does this indicate apartment building homes, which is more expansive vertically rather than horizontally (compared to regular flat homes), thus requiring not that much square foot of lot.

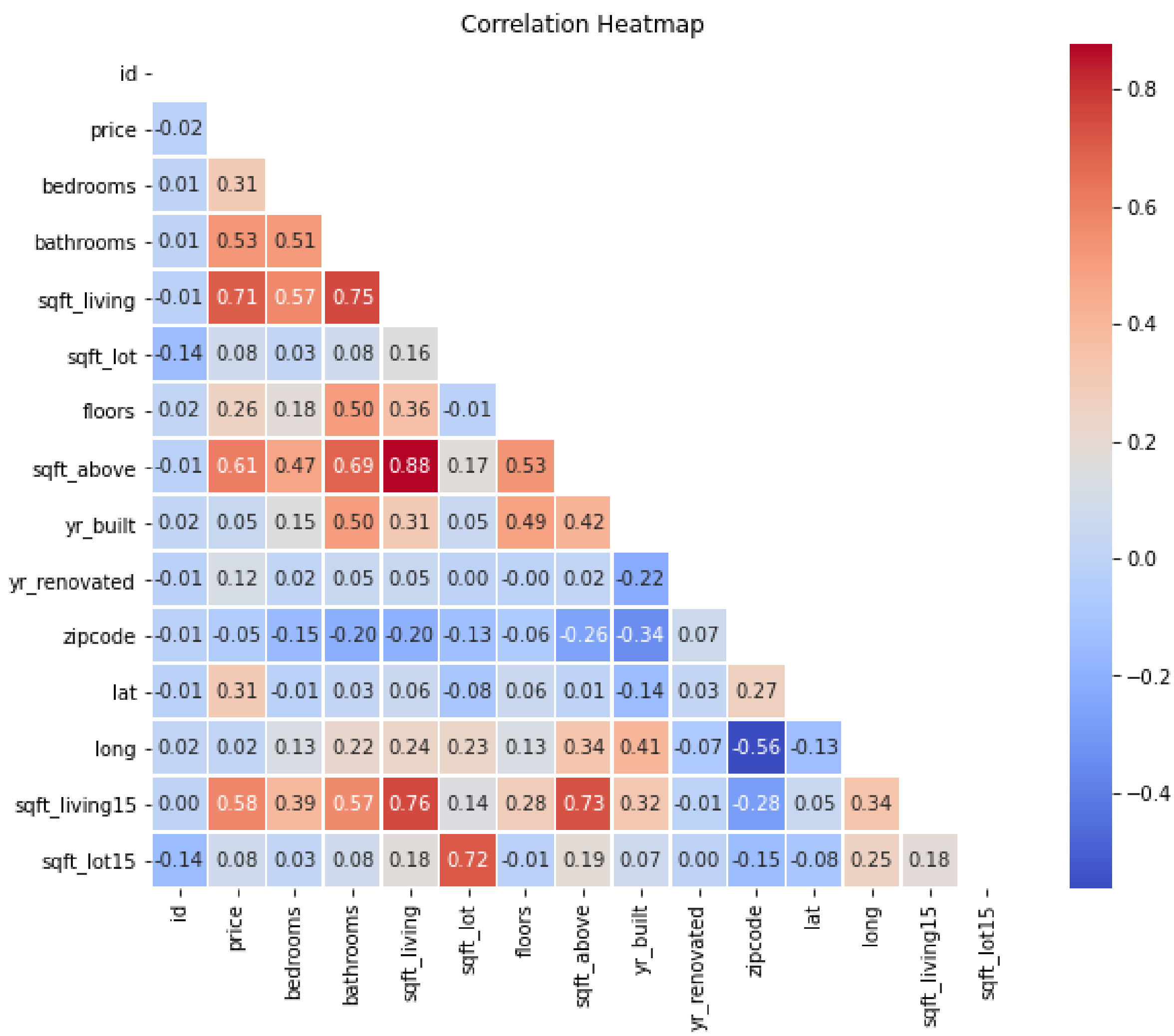
In [20]:
```
# Selecting the numeric columns

numeric_columns = df.select_dtypes(include='number').columns
numeric_df = df[numeric_columns]
```

In [21]:
```
# Computing the correlation matrix to check for Linearity

numeric_df.corr()
```

Out[21]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_
id	1.000000	-0.016236	0.007883	0.005406	-0.008858	-0.136009	0.020083	-0.0
price	-0.016236	1.000000	0.305947	0.526228	0.705975	0.083572	0.259193	0.6
bedrooms	0.007883	0.305947	1.000000	0.512488	0.574179	0.025684	0.180158	0.4
bathrooms	0.005406	0.526228	0.512488	1.000000	0.753613	0.080027	0.504916	0.6
sqft_living	-0.008858	0.705975	0.574179	0.753613	1.000000	0.164512	0.358657	0.8
sqft_lot	-0.136009	0.083572	0.025684	0.080027	0.164512	1.000000	-0.010454	0.1
floors	0.020083	0.259193	0.180158	0.504916	0.358657	-0.010454	1.000000	0.5
sqft_above	-0.009551	0.611886	0.474835	0.685456	0.876260	0.173422	0.528179	1.0
yr_built	0.024011	0.048672	0.153048	0.504193	0.313206	0.051256	0.486854	0.4
yr_renovated	-0.010419	0.123077	0.016632	0.047255	0.049992	0.002169	-0.001287	0.0
zipcode	-0.007812	-0.048661	-0.148417	-0.198798	-0.195836	-0.129495	-0.057011	-0.2
lat	-0.006173	0.306058	-0.007583	0.029184	0.057228	-0.084771	0.058032	0.0
long	0.018679	0.020241	0.129424	0.221825	0.238786	0.231748	0.128729	0.3
sqft_living15	0.000362	0.580963	0.392272	0.569053	0.756576	0.144640	0.281330	0.7
sqft_lot15	-0.141551	0.078972	0.025342	0.081837	0.176506	0.718327	-0.013882	0.1

In [22]:
```
# Creating a heatmap using seaborn

columns=['price', 'bedrooms', 'grade_no', 'yr_built', 'sqft_living', 'floors',
         'bathrooms', 'cond_avg', 'cond_fair', 'cond_good', 'cond_poor', 'cond_
index=['price', 'bedrooms', 'grade_no', 'yr_built', 'sqft_living', 'floors',
       'bathrooms', 'cond_avg', 'cond_fair', 'cond_good', 'cond_poor', 'cond_v

corr_matrix = numeric_df.corr()
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
fig, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(corr_matrix, mask=mask, annot=True, cmap='coolwarm', fmt=".2f", li
ax.set_title('Correlation Heatmap')
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.show()
```



Data Pre-processing before fitting our Regression Model

This invloves techniques such as:

1. Deal with null values
2. Encoding categorical variables

3. Feature engineering
4. Transformations
5. Feature scaling

In [23]:
```
# converting sqft_basement and waterfront which involves using OneHotEncorder.
df['sqft_basement'] = pd.to_numeric(df['sqft_basement'], errors='coerce')
```

Categorical columns include `condition` and `waterfront`.

One Hot Encoding the Categorical Variables

In [24]:
```
# One_Hot_Encoding the categorical variables

df["grade_no"]= pd.to_numeric(df['grade'].str.split().str[0])

condition = df[['condition']]
ohe = OneHotEncoder(categories="auto", sparse=False, handle_unknown="ignore")
ohe.fit(condition)
condition_enc = ohe.transform(condition)
condition_enc = pd.DataFrame(condition_enc,
                             columns=['cond_avg','cond_fair','cond_good','cond_
                             index=df.index)
df.drop('condition', axis=1, inplace=True)
df = pd.concat([df, condition_enc], axis=1)
```

In [25]:
```
# Selecting our features of relevance

df_values= df[['price','bedrooms','bathrooms','sqft_living','sqft_basement',
               'sqft_lot15', 'grade_no', 'cond_avg', 'cond_fair','cond_good',
               'cond_poor', 'cond_verygood']]
```

In [26]:
```
# Confirming if there are any null values

df.isna().sum()
```

Out[26]:
```
id                 0
date               0
price              0
bedrooms           0
bathrooms          0
sqft_living        0
sqft_lot           0
floors             0
waterfront         0
view               0
grade              0
sqft_above         0
sqft_basement    332
yr_built           0
yr_renovated       0
zipcode            0
lat                0
long               0
sqft_living15      0
sqft_lot15         0
grade_no           0
cond_avg           0
cond_fair          0
cond_good          0
cond_poor          0
cond_verygood      0
dtype: int64
```

In [27]:
```
# Replacing the the null values with 0

df['sqft_basement'] = df['sqft_basement'].fillna(0)
```

In [28]:
```
# Checking if the null values have been replaced with 0

df.isna().sum()
```

Out[28]:
```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
grade_no         0
cond_avg         0
cond_fair        0
cond_good        0
cond_poor        0
cond_verygood    0
dtype: int64
```

In [29]:
```
# Displaying our final df before modeling

df_values
```

Out[29]:

	price	bedrooms	bathrooms	sqft_living	sqft_basement	sqft_lot15	grade_no	cond_
1	538000.0	3	2.25	2570	400.0	7639	7	
3	604000.0	4	3.00	1960	910.0	5000	7	
4	510000.0	3	2.00	1680	0.0	7503	8	
5	1230000.0	4	4.50	5420	1530.0	101930	11	
6	257500.0	3	2.25	1715	NaN	6819	7	
...	...	...	...	...	...	...	...	
21591	475000.0	3	2.50	1310	130.0	1265	8	
21592	360000.0	3	2.50	1530	0.0	1509	8	
21593	400000.0	4	2.50	2310	0.0	7200	8	
21594	402101.0	2	0.75	1020	0.0	2007	7	
21596	325000.0	2	0.75	1020	0.0	1357	7	

15676 rows × 12 columns

LINEAR MODELING

Checking for the Linearity Assumption.

Here, we assert two things before building our model;

1. We want to include the features which have the highest correlation with our target variable(price).

2. While following the condition above, we want our features not to be multicorrelated with each other.

In [30]:
```
# checking for correlations between our features and the target variable
# from the highest to the lowest

df.corr()['price'].sort_values(ascending=False).head(15)
```

Out[30]:
```
price            1.000000
sqft_living      0.705975
grade_no         0.664092
sqft_above       0.611886
sqft_living15    0.580963
bathrooms        0.526228
sqft_basement    0.315663
lat              0.306058
bedrooms         0.305947
floors           0.259193
yr_renovated     0.123077
sqft_lot         0.083572
sqft_lot15       0.078972
cond_verygood    0.055422
yr_built         0.048672
Name: price, dtype: float64
```

In [31]:
```
# Checking for Multicollinearity in our predictors
corr_df = df.corr().abs().stack().reset_index().sort_values(0, ascending=False
corr_df['pairs'] = list(zip(corr_df.level_0, corr_df.level_1))

# Dropping 'level_0' and 'level_1'
corr_df.set_index(['pairs'], inplace=True)
corr_df.drop(columns=['level_0', 'level_1'], inplace=True)

# Renaming our column
corr_df.columns = ["corr_coef"]

# Veiwing the highly correlated predictor pairs
# (our threshold is features with a value above 80%)

corr_df[(corr_df.corr_coef > 0.80) & (corr_df.corr_coef < 1)]
```

Out[31]:

	corr_coef
pairs	
(sqft_living, sqft_above)	0.876260
(sqft_above, sqft_living)	0.876260
(cond_avg, cond_good)	0.811063
(cond_good, cond_avg)	0.811063

In [32]:
```
# Dropping unnecessary columns
df.drop(columns=['id','date','grade','yr_built','yr_renovated', 'lat', 'long',
                 'cond_fair','cond_good','cond_poor','cond_verygood'], inplace
```

In [33]:
```
# Checking the metadata of the remaining columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15676 entries, 1 to 21596
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          15676 non-null  float64
 1   bedrooms       15676 non-null  int64  
 2   bathrooms      15676 non-null  float64
 3   sqft_living    15676 non-null  int64  
 4   sqft_lot       15676 non-null  int64  
 5   floors         15676 non-null  float64
 6   waterfront     15676 non-null  object 
 7   view           15676 non-null  object 
 8   sqft_above     15676 non-null  int64  
 9   sqft_basement  15676 non-null  float64
 10  zipcode        15676 non-null  int64  
 11  sqft_living15  15676 non-null  int64  
 12  sqft_lot15     15676 non-null  int64  
 13  grade_no       15676 non-null  int64  
dtypes: float64(4), int64(8), object(2)
memory usage: 1.8+ MB
```

Defining our Functions for use

In [34]:
```
# Defining a function for fitting our model
def run_model(data):
    x = data.drop('price', axis=1)
    y = data['price']
    linreg = LinearRegression()
    crossvalidation = KFold(n_splits = 10, shuffle = True, random_state = 1)
    mean_r2 = np.mean(cross_val_score(linreg, x, y, scoring='r2', cv=crossvali
    mse = np.mean(cross_val_score(linreg, x, y, scoring='neg_mean_squared_erro
    rmse = np.sqrt(mse)

    x_cols = data.drop('price', axis=1).columns
    y_col = 'price'
    plus = '+'.join(x_cols)
    formula = y_col + '~' + plus
    model = ols(formula=formula, data=data).fit()
    print('The mean r^2 for a KFold test with 10 splits is {} \n'.format(mean_
    print('The mean RMSE for a KFold test with 10 splits is {} \n'.format(rmse
    print(model.summary())

    # Testing for homoscedasticity
    residuals = model.resid
    fig, ax = plt.subplots(figsize=(15,8))
    plt.scatter(model.predict(x), residuals)
    plt.plot(model.predict(x), [np.mean(residuals) for i in range(len(data))])
    ax.set_title('Homoscedasticity')
    plt.show()
    print('\n')

    # Testing for normality using a QQ-plot
    fig, ax = plt.subplots(figsize=(15,8))
    sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True, ax=ax)
    ax.set_title('QQ Plot')
    plt.show()
```

In [35]:
```
# Defining a function to perform log transformations
def log_transform(features, df):
    for feat in features:
        df[feat] = df[feat].map(lambda x: np.log(x))
    return df
```

In [36]:
```
# Defining a function to generate a heatmap
def heatmap(data):
    corr = data.corr()
    fig, ax = plt.subplots(figsize=(12,12))
    sns.heatmap(corr, cmap='Reds', annot=True, ax=ax);
```

In [37]:
```
# Defining a function to remove outliers from our features
def outliers(features, data):
    for feat in features:
        mu = np.mean(data[feat])
        std = np.std(data[feat])
        outlier = 3*std
        data = data[(data[feat] <= mu+outlier) & (data[feat] >= mu-outlier)]
    return data
```

In [38]:
```
# Defining a function to perform OneHotEncoding
def scale_ohe(ohe_feature, data):
    ohe = pd.get_dummies(data[ohe_feature], prefix=ohe_feature, drop_first=Tru
    no_ohe = data.drop(ohe_feature, axis=1)
    no_ohe_scale = no_ohe.apply(scale)
    return pd.concat([no_ohe_scale, ohe], axis=1)
```

In [39]:
```
# Defining a function for getting the coefficients of features
def get_coefficients_continuous(scaled_coefs, features):
    for i, feat in enumerate(features):
        maximum = df_log['price'].max()
        minimum = df_log['price'].min()
        range_feat = df_no_outlier[feat].max() - df_no_outlier[feat].min()
        unscale = abs(scaled_coefs[i])*(maximum-minimum)+minimum
        unlog = math.exp(unscale)

        slope_actual = unlog/range_feat

        if scaled_coefs[i] >= 0:
            print('Coefficient for {} is ${}'.format(feat, slope_actual))
        else:
            print('Coefficient for {} is ${}'.format(feat, slope_actual*-1))
```

In [40]:
```
df = df[df['sqft_basement'] != '?']
df['sqft_basement'] = df['sqft_basement'].astype(float)
```

In [41]:
```
df['sqft_basement'] = df['sqft_basement'].astype(float)
```

In [42]:
```
df['basement'] = np.where(df['sqft_basement'] > 0, 1, 0)
df.head()
```

Out[42]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	sqft_above
1	538000.0	3	2.25	2570	7242	2.0	NO	NONE	2170
3	604000.0	4	3.00	1960	5000	1.0	NO	NONE	1050
4	510000.0	3	2.00	1680	8080	1.0	NO	NONE	1680
5	1230000.0	4	4.50	5420	101930	1.0	NO	NONE	3890
6	257500.0	3	2.25	1715	6819	2.0	NO	NONE	1715

In [43]:
```
df.drop(columns=["sqft_basement","waterfront","view"], inplace=True)
```

Building the Baseline model

For the baseline model, we will do a simple linear regression, using the most highly correlated feature and then we improve our model from there through an iterative process whereby we perform techniques such as:

1. Dealing with outliers, i.e. either removing outliers or apply transformations to make the data more robust to outliers.
2. Transormations e.g. log transformations of our features.
3. Feature Scaling, i.e. to ensure that all features are on a similar scale. Common scaling techniques include standardization (mean normalization) or normalization (min-max scaling). This will aid in direct comparison of our features and determine which has the highest impact on our target variable.

In [44]:
```
# Assigning our features and target variables
X = df["sqft_living"]
y = df['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando

# Baseline Model with statsmodels
X_train_with_intercept = sm.add_constant(X_train)
baseline_model = sm.OLS(y_train, X_train_with_intercept)
baseline_results = baseline_model.fit()
baseline_predictions = baseline_results.predict(sm.add_constant(X_test))
baseline_rmse = mean_squared_error(y_test, baseline_predictions, squared=False

print("Baseline Model RMSE:", baseline_rmse)
print(baseline_results.summary())
```

```
Baseline Model RMSE: 271201.25051764137
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  price   R-squared:                       0.49
5
Model:                            OLS   Adj. R-squared:                  0.49
5
Method:                 Least Squares   F-statistic:                 1.229e+0
4
Date:                Thu, 01 Jun 2023   Prob (F-statistic):              0.0
0
Time:                        12:26:49   Log-Likelihood:            -1.7425e+0
5
No. Observations:               12540   AIC:                         3.485e+0
5
Df Residuals:                   12538   BIC:                         3.485e+0
5
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
==
                  coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
const       -5.235e+04   5849.737     -8.949      0.000   -6.38e+04   -4.09e+
04
sqft_living   285.1177      2.572    110.841      0.000     280.076     290.1
60
==============================================================================
=
Omnibus:                     8675.250   Durbin-Watson:                   2.00
5
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           331404.03
7
Skew:                           2.839   Prob(JB):                        0.0
0
Kurtosis:                      27.536   Cond. No.                     5.68e+0
3
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 5.68e+03. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

Interpretation of results

1. The model is generally statistically significant with an F-statistic p_value of 0.0 at a significance level of 0.05

2. The R-squared value is 0.495, indicating that approximately 49.5% of the variation in the price can be explained by the sqft_living variable. This value is very low and the model needs improving.

3. The coefficient of the constant term (const) is -5.235e+04, and the coefficient of the sqft_living variable is 285.1177. These coefficients represent the expected change in the price for a one-unit change in the corresponding predictor variable, assuming other variables are held constant,e.g. For a one-unit increase in square-foot living area, we see an associated increase in around 285 dollars in selling price of the houses.

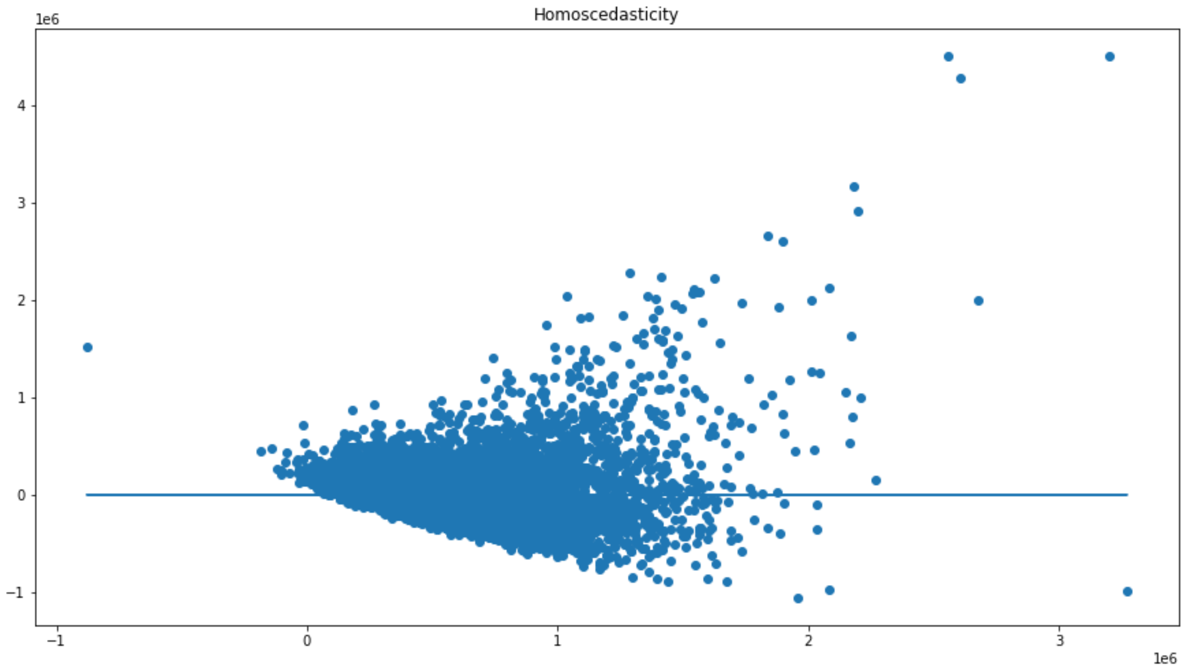
Iteration 1

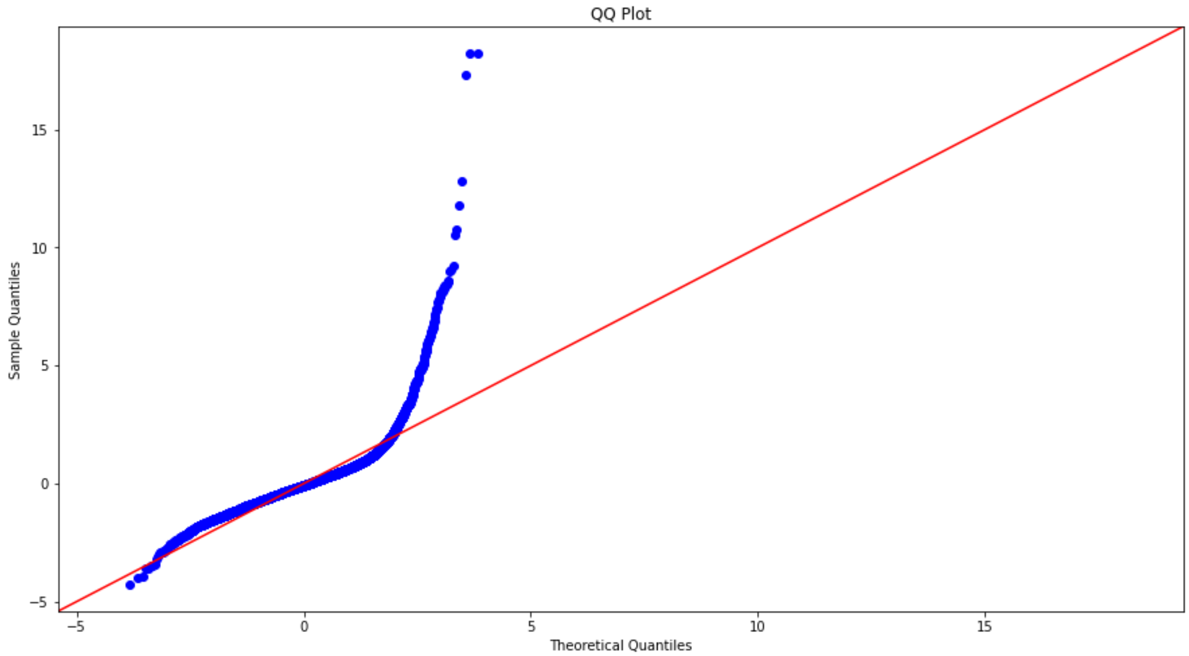
Here we perform the first iteration whereby we have included more features into the model. We also perform a KFold test with 10 splits and get the mean r-squared as well as the mean RMSE of our model.

In [45]:
```
# Fit our model using the defined function

run_model(df)
```

```
The mean r^2 for a KFold test with 10 splits is 0.558608548218698

The mean RMSE for a KFold test with 10 splits is 247521.86276668686

                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  price   R-squared:                       0.56
1
Model:                            OLS   Adj. R-squared:                  0.56
1
Method:                 Least Squares   F-statistic:                     181
9.
Date:                Thu, 01 Jun 2023   Prob (F-statistic):              0.0
0
Time:                        12:26:50   Log-Likelihood:           -2.1689e+0
5
No. Observations:               15676   AIC:                       4.338e+0
5
Df Residuals:                   15664   BIC:                       4.339e+0
5
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
====
                    coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
Intercept     -6.288e+07   3.92e+06    -16.060      0.000   -7.06e+07   -5.52
e+07
bedrooms       -4.35e+04   2676.343    -16.255      0.000   -4.88e+04   -3.83
e+04
bathrooms     -1.275e+04   4454.806     -2.863      0.004   -2.15e+04   -402
1.685
sqft_living     255.7224      8.380     30.514      0.000     239.296     27
2.149
sqft_lot          0.0359      0.068      0.529      0.597      -0.097
0.169
floors        -2.511e+04   5016.612     -5.005      0.000   -3.49e+04   -1.53
e+04
sqft_above      -40.1904      9.273     -4.334      0.000     -58.366     -2
2.015
zipcode         635.9363     39.909     15.935      0.000     557.710     71
4.162
sqft_living15    26.4631      4.864      5.440      0.000      16.928      3
5.998
sqft_lot15       -0.6038      0.102     -5.904      0.000      -0.804      -
0.403
grade_no       9.843e+04   3008.590     32.717      0.000    9.25e+04    1.04
e+05
basement       5038.0897   7251.047      0.695      0.487   -9174.799    1.93
e+04
==============================================================================
=
Omnibus:                    12544.377   Durbin-Watson:                   1.97
6
```

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          823131.54
1
Skew:                           3.365   Prob(JB):                        0.0
0
Kurtosis:                      37.856   Cond. No.                   2.00e+0
8
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large,  2e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
```





Interpretation of results

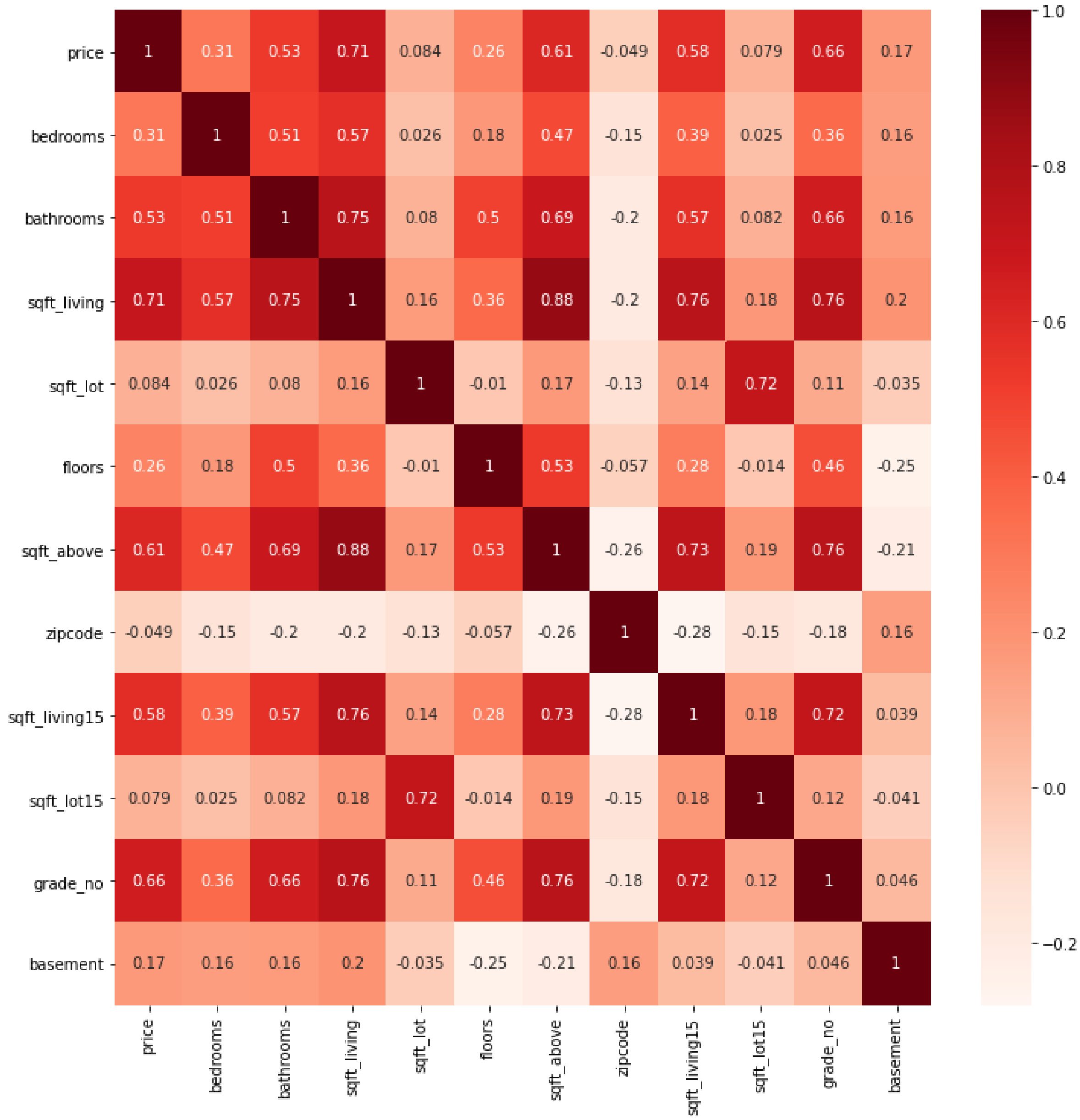
1. The model is generally statistically significant with an F-statistic p_value of 0.0 at a significance level of 0.05

2. The R-squared value is 0.561, indicating that approximately 56.1% of the variation in the price can be explained by the model. This value indicates an improvement of the baseline model.

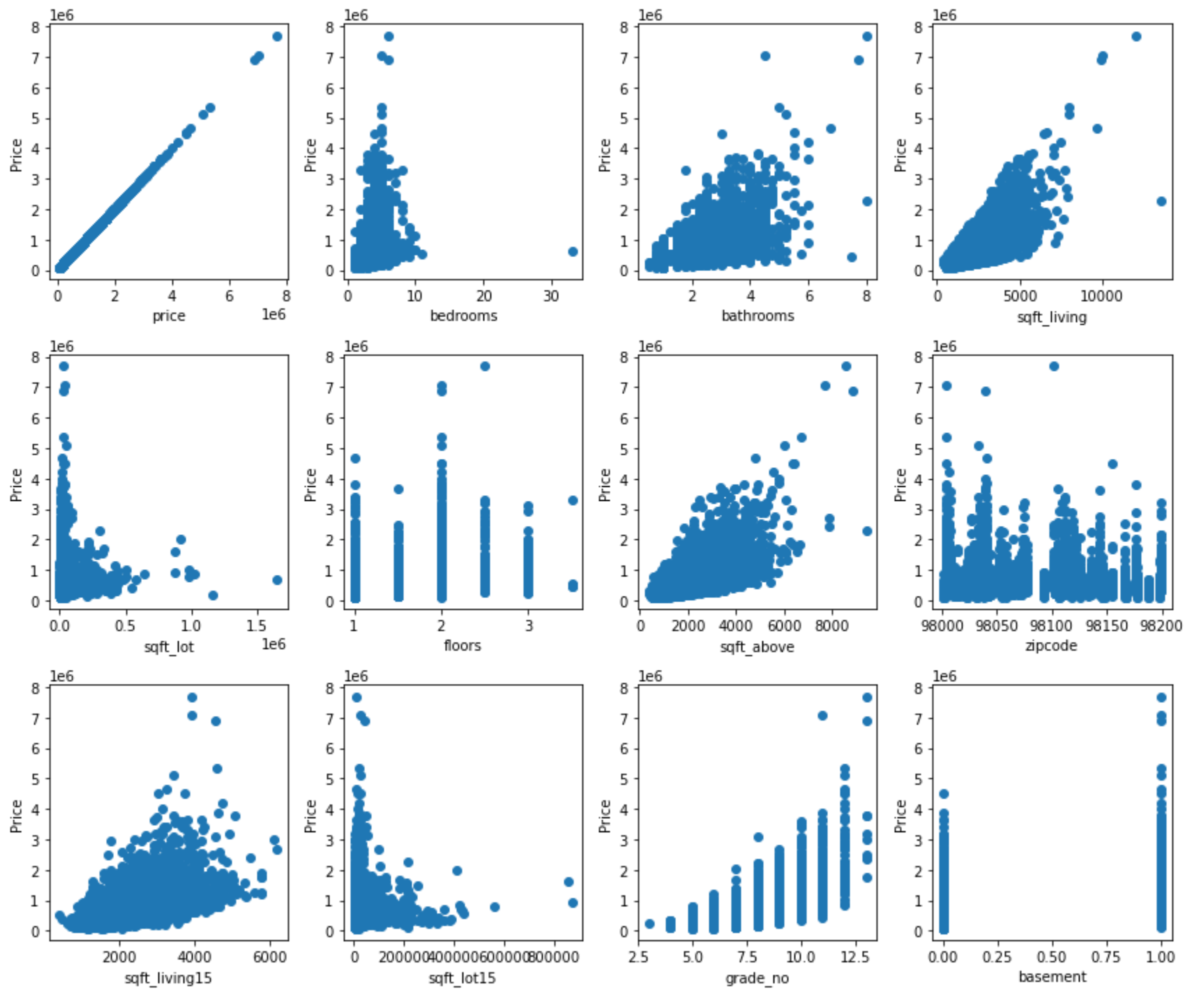
3. The plot to test for homoscedasticity reveals that the residuals are somewhat heteroscedastic because they are diverging/variating. This is an indication of skewness/heavy-tailed dataset/presence of outliers.

4. The QQ-plot is used to test for normality of residuals. In this case, the residuals appear not to be normal because they are diverging off the line.

In [46]:
```
# Generating the heatmap
heatmap(df)
```



In [47]:
```
# Plot scatter plots against "price"
X = df
y = df["price"]

fig, axes = plt.subplots(nrows=3, ncols=4, figsize=(12, 10))
flatten_axes = axes.flatten()

for i, column in enumerate(X.columns):
    flatten_axes[i].scatter(X[column], y)
    flatten_axes[i].set_xlabel(column)
    flatten_axes[i].set_ylabel("Price")

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



In [48]:
```
continuous = ['price', 'sqft_living', 'sqft_lot', 'sqft_living15', 'sqft_lot15
df_no_outlier = outliers(continuous, df)
df_no_outlier.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14582 entries, 1 to 21596
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          14582 non-null  float64
 1   bedrooms       14582 non-null  int64  
 2   bathrooms      14582 non-null  float64
 3   sqft_living    14582 non-null  int64  
 4   sqft_lot       14582 non-null  int64  
 5   floors         14582 non-null  float64
 6   sqft_above     14582 non-null  int64  
 7   zipcode        14582 non-null  int64  
 8   sqft_living15  14582 non-null  int64  
 9   sqft_lot15     14582 non-null  int64  
 10  grade_no       14582 non-null  int64  
 11  basement       14582 non-null  int32  
dtypes: float64(3), int32(1), int64(8)
memory usage: 1.4 MB
```

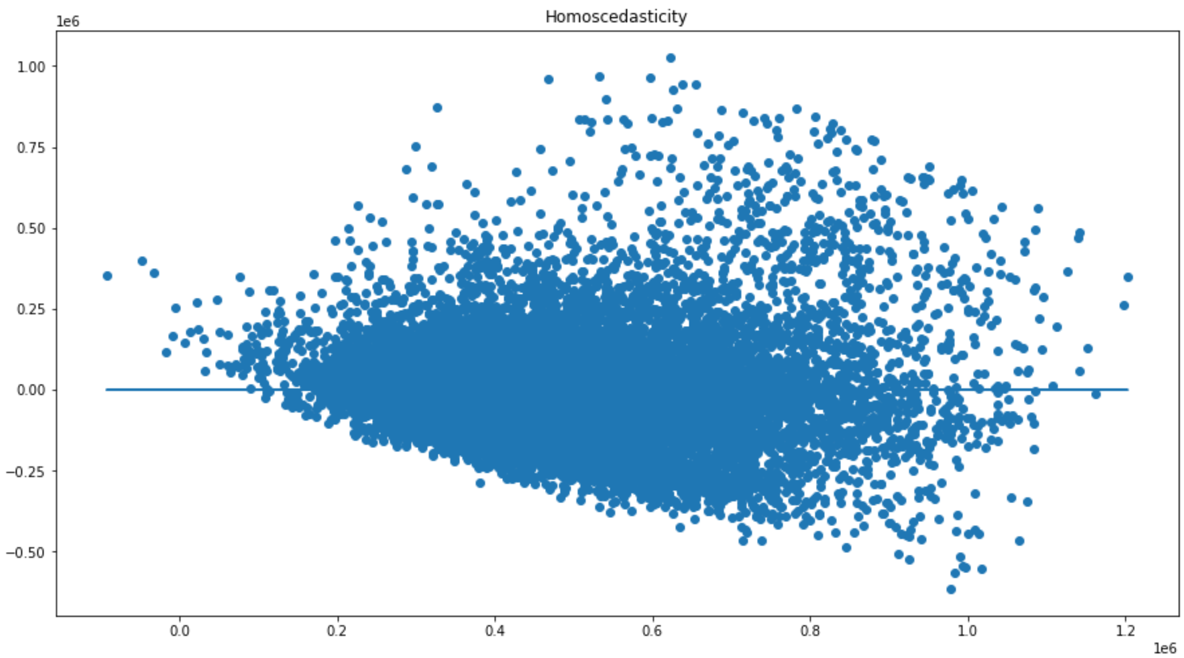
Iteration 2

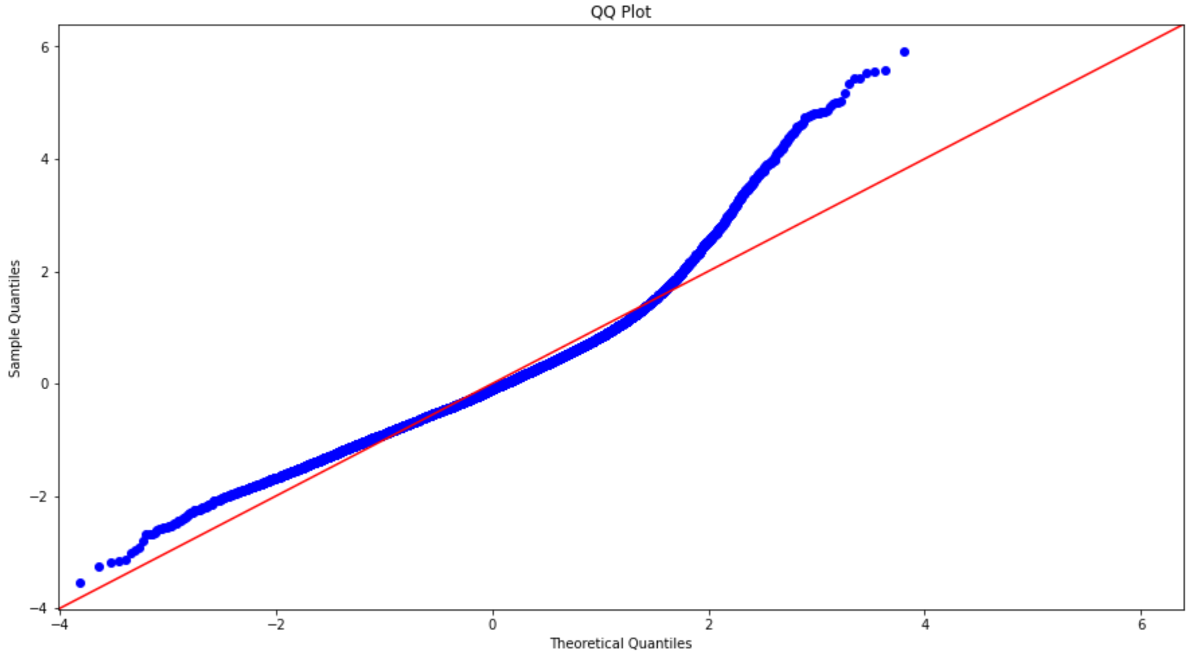
In this iteration, we tried to remove outliers from our data to see the impact on our model's performance.

In [49]:
```
# Fitting our model without outliers

run_model(df_no_outlier)
```

```
The mean r^2 for a KFold test with 10 splits is 0.5023419026527095

The mean RMSE for a KFold test with 10 splits is 173691.9672560325

                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  price   R-squared:                       0.50
4
Model:                            OLS   Adj. R-squared:                  0.50
4
Method:                 Least Squares   F-statistic:                     134
8.
Date:                Thu, 01 Jun 2023   Prob (F-statistic):               0.0
0
Time:                        12:27:04   Log-Likelihood:            -1.9661e+0
5
No. Observations:               14582   AIC:                        3.932e+0
5
Df Residuals:                   14570   BIC:                        3.933e+0
5
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
====
                    coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
Intercept     -5.059e+07   2.85e+06    -17.728      0.000   -5.62e+07    -4.5
e+07
bedrooms      -1.882e+04   2190.043     -8.595      0.000   -2.31e+04   -1.45
e+04
bathrooms     -2.732e+04   3422.967     -7.982      0.000    -3.4e+04   -2.06
e+04
sqft_living     155.8640      6.919     22.528      0.000     142.302      16
9.426
sqft_lot          0.3337      0.230      1.448      0.148      -0.118
0.786
floors         7402.9042   3829.544      1.933      0.053    -103.488    1.49
e+04
sqft_above      -47.7686      7.463     -6.400      0.000     -62.398      -3
3.139
zipcode         511.4977     29.081     17.589      0.000     454.495      56
8.500
sqft_living15    64.7177      3.987     16.232      0.000      56.903       7
2.533
sqft_lot15       -2.3417      0.326     -7.175      0.000      -2.981       -
1.702
grade_no       9.022e+04   2259.478     39.930      0.000    8.58e+04    9.46
e+04
basement       2.111e+04   5466.456      3.861      0.000    1.04e+04    3.18
e+04
==============================================================================
=
Omnibus:                     2782.963   Durbin-Watson:                   1.96
5
```

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             7255.79
9
Skew:                           1.041   Prob(JB):                         0.0
0
Kurtosis:                       5.759   Cond. No.                     1.97e+0
8
=============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 1.97e+08. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```





Interpretation of results

1. The model is generally statistically significant with an F-statistic p_value of 0.0 at a significance level of 0.05
2. The R-squared value is 0.504, indicating that approximately 50.4% of the variation in the price can be explained by the model. This value indicates a drop from the previous model.
3. The plot to test for homoscedasticity reveals that the residuals are becoming homoscedastic because they are converging and appear to be having an equal variance. So this assumption is satisfied.
4. The QQ-plot is used to test for normality of residuals. In this case, the residuals appear to be somewhat normal but there is still presence of skewness/heavy-tails/outliers.

In [50]:
```
# Displaying the DataFrame
df_no_outlier
```

Out[50]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above	zipcode	sqft_liv
1	538000.0	3	2.25	2570	7242	2.0	2170	98125	
3	604000.0	4	3.00	1960	5000	1.0	1050	98136	
4	510000.0	3	2.00	1680	8080	1.0	1680	98074	
6	257500.0	3	2.25	1715	6819	2.0	1715	98003	
8	229500.0	3	1.00	1780	7470	1.0	1050	98146	
...	...	...	...	...	...	...	...	...	
21591	475000.0	3	2.50	1310	1294	2.0	1180	98116	
21592	360000.0	3	2.50	1530	1131	3.0	1530	98103	
21593	400000.0	4	2.50	2310	5813	2.0	2310	98146	
21594	402101.0	2	0.75	1020	1350	2.0	1020	98144	
21596	325000.0	2	0.75	1020	1076	2.0	1020	98144	

14582 rows × 12 columns

In [51]:
```
df_no_outlier.info()
```

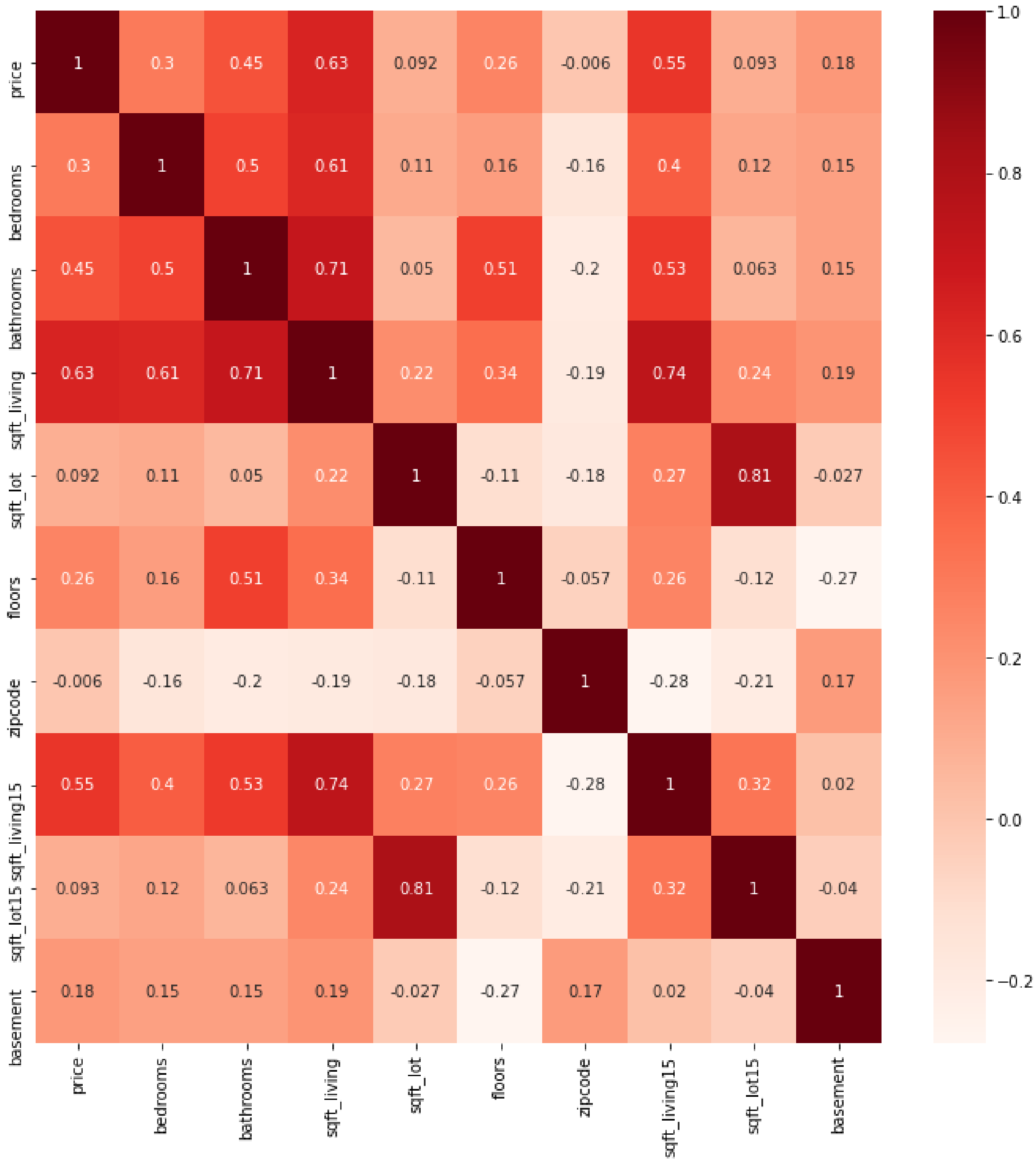
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14582 entries, 1 to 21596
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          14582 non-null  float64
 1   bedrooms       14582 non-null  int64  
 2   bathrooms      14582 non-null  float64
 3   sqft_living    14582 non-null  int64  
 4   sqft_lot       14582 non-null  int64  
 5   floors         14582 non-null  float64
 6   sqft_above     14582 non-null  int64  
 7   zipcode        14582 non-null  int64  
 8   sqft_living15  14582 non-null  int64  
 9   sqft_lot15     14582 non-null  int64  
 10  grade_no       14582 non-null  int64  
 11  basement       14582 non-null  int32  
dtypes: float64(3), int32(1), int64(8)
memory usage: 1.4 MB
```

In [52]:
```
# Dropping unnecessary columns
df_no_outlier.drop(columns=["sqft_above","grade_no"],inplace=True)
```

In [53]:
```
df_no_outlier.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14582 entries, 1 to 21596
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          14582 non-null  float64
 1   bedrooms       14582 non-null  int64  
 2   bathrooms      14582 non-null  float64
 3   sqft_living    14582 non-null  int64  
 4   sqft_lot       14582 non-null  int64  
 5   floors         14582 non-null  float64
 6   zipcode        14582 non-null  int64  
 7   sqft_living15  14582 non-null  int64  
 8   sqft_lot15     14582 non-null  int64  
 9   basement       14582 non-null  int32  
dtypes: float64(3), int32(1), int64(6)
memory usage: 1.2 MB
```

In [54]:
```
# Displaying the heatmap
heatmap(df_no_outlier)
```



Iteration 3

In this iteration, we perform some normalization and log-transformations. This will help to mitigate the presence of outliers in our dataset and hence make the dataset more robust, and also improving the linearity between the target variable(price) and the features.

Normalization and Log_transformation

In [55]:
```
# Displaying the DataFrame
df_no_outlier
```

Out[55]:

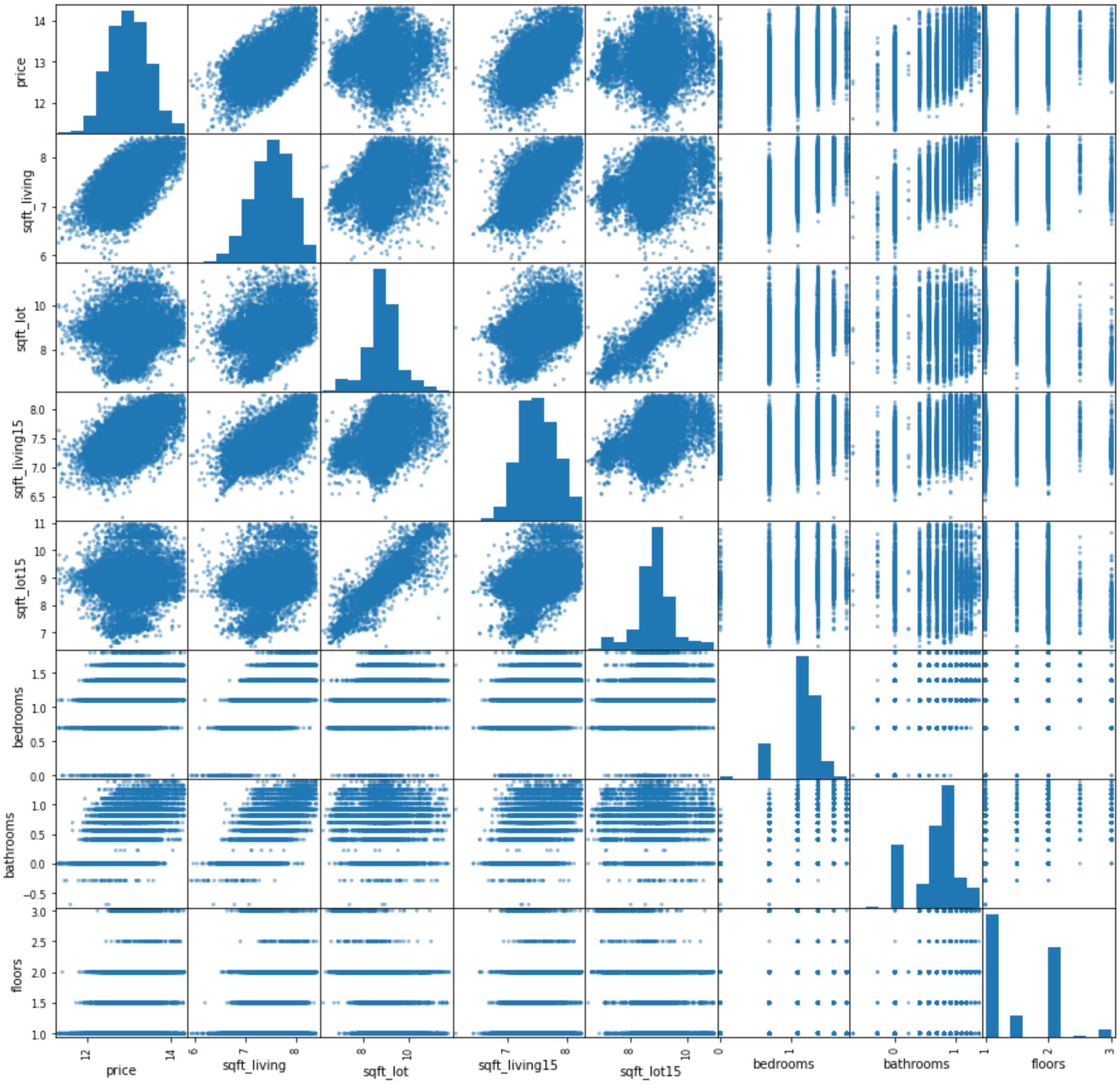
	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	zipcode	sqft_living15	sqft_l
1	538000.0	3	2.25	2570	7242	2.0	98125	1690	
3	604000.0	4	3.00	1960	5000	1.0	98136	1360	
4	510000.0	3	2.00	1680	8080	1.0	98074	1800	
6	257500.0	3	2.25	1715	6819	2.0	98003	2238	
8	229500.0	3	1.00	1780	7470	1.0	98146	1780	
...	...	...	...	...	...	...	...	...	
21591	475000.0	3	2.50	1310	1294	2.0	98116	1330	
21592	360000.0	3	2.50	1530	1131	3.0	98103	1530	
21593	400000.0	4	2.50	2310	5813	2.0	98146	1830	
21594	402101.0	2	0.75	1020	1350	2.0	98144	1020	
21596	325000.0	2	0.75	1020	1076	2.0	98144	1020	

14582 rows × 10 columns

In [56]:
```
# Checking the correlations in descending order
df_no_outlier.corr()["price"].sort_values(ascending=False)
```

Out[56]:
```
price            1.000000
sqft_living      0.627050
sqft_living15    0.550582
bathrooms        0.446199
bedrooms         0.297462
floors           0.256793
basement         0.178915
sqft_lot15       0.093464
sqft_lot         0.091582
zipcode         -0.005953
Name: price, dtype: float64
```

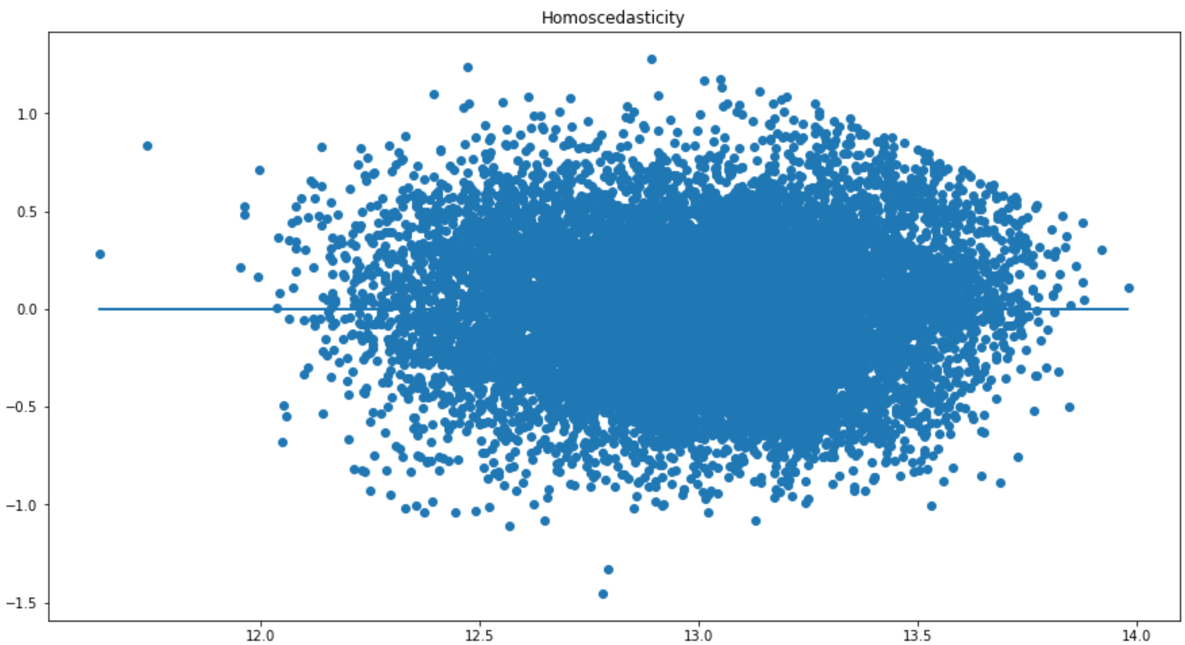
In [57]:
```
# Performing log transformations using our defined function
normalize = ['price', 'sqft_living', 'sqft_lot', 'sqft_living15', 'sqft_lot15'
df_log = log_transform(normalize, df_no_outlier)
pd.plotting.scatter_matrix(df_log[continuous], figsize=(15, 15));
```

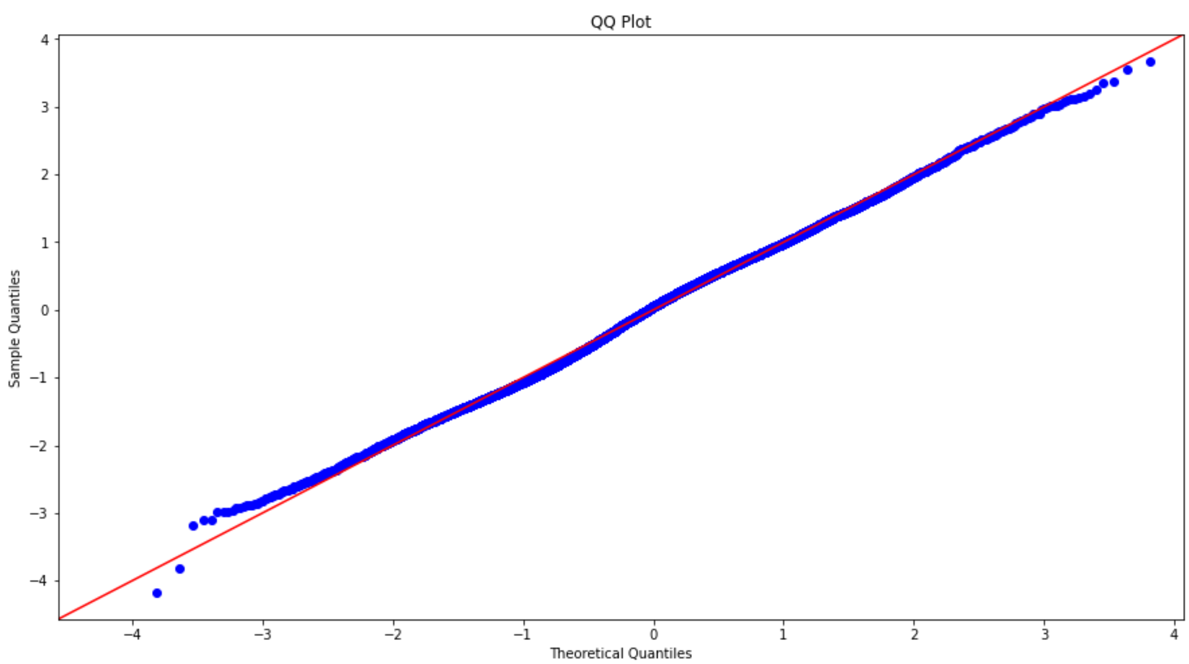


In [58]:
```
# Using our `df_log` we fit our model using our defined function
run_model(df_log)
```

```
The mean r^2 for a KFold test with 10 splits is 0.45540890401512824

The mean RMSE for a KFold test with 10 splits is 0.348485819455707

                            OLS Regression Results                            
==============================================================================
=
Dep. Variable:                  price   R-squared:                       0.45
7
Model:                            OLS   Adj. R-squared:                  0.45
7
Method:                 Least Squares   F-statistic:                     136
2.
Date:                Thu, 01 Jun 2023   Prob (F-statistic):               0.0
0
Time:                        12:27:27   Log-Likelihood:                -5309.
7
No. Observations:               14582   AIC:                         1.064e+0
4
Df Residuals:                   14572   BIC:                         1.072e+0
4
Df Model:                           9                                         
Covariance Type:            nonrobust                                         
==============================================================================
====
                    coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
Intercept      -100.5845      5.805    -17.326      0.000    -111.964     -8
9.205
bedrooms         -0.1838      0.014    -13.162      0.000      -0.211      -
0.156
bathrooms        -0.0522      0.013     -4.114      0.000      -0.077      -
0.027
sqft_living       0.6269      0.016     39.088      0.000       0.595       
0.658
sqft_lot         -0.0268      0.010     -2.766      0.006      -0.046      -
0.008
floors            0.0417      0.008      5.237      0.000       0.026       
0.057
zipcode           0.0011    5.9e-05     18.355      0.000       0.001       
0.001
sqft_living15     0.4777      0.015     32.520      0.000       0.449       
0.507
sqft_lot15       -0.0669      0.011     -6.311      0.000      -0.088      -
0.046
basement          0.0685      0.007      9.315      0.000       0.054       
0.083
==============================================================================
=
Omnibus:                       52.603   Durbin-Watson:                   1.99
1
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               40.05
3
Skew:                          -0.009   Prob(JB):                     2.01e-0
9
```

```
Kurtosis:                       2.744   Cond. No.                    1.97e+0
8
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 1.97e+08. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```





Interpretation of results

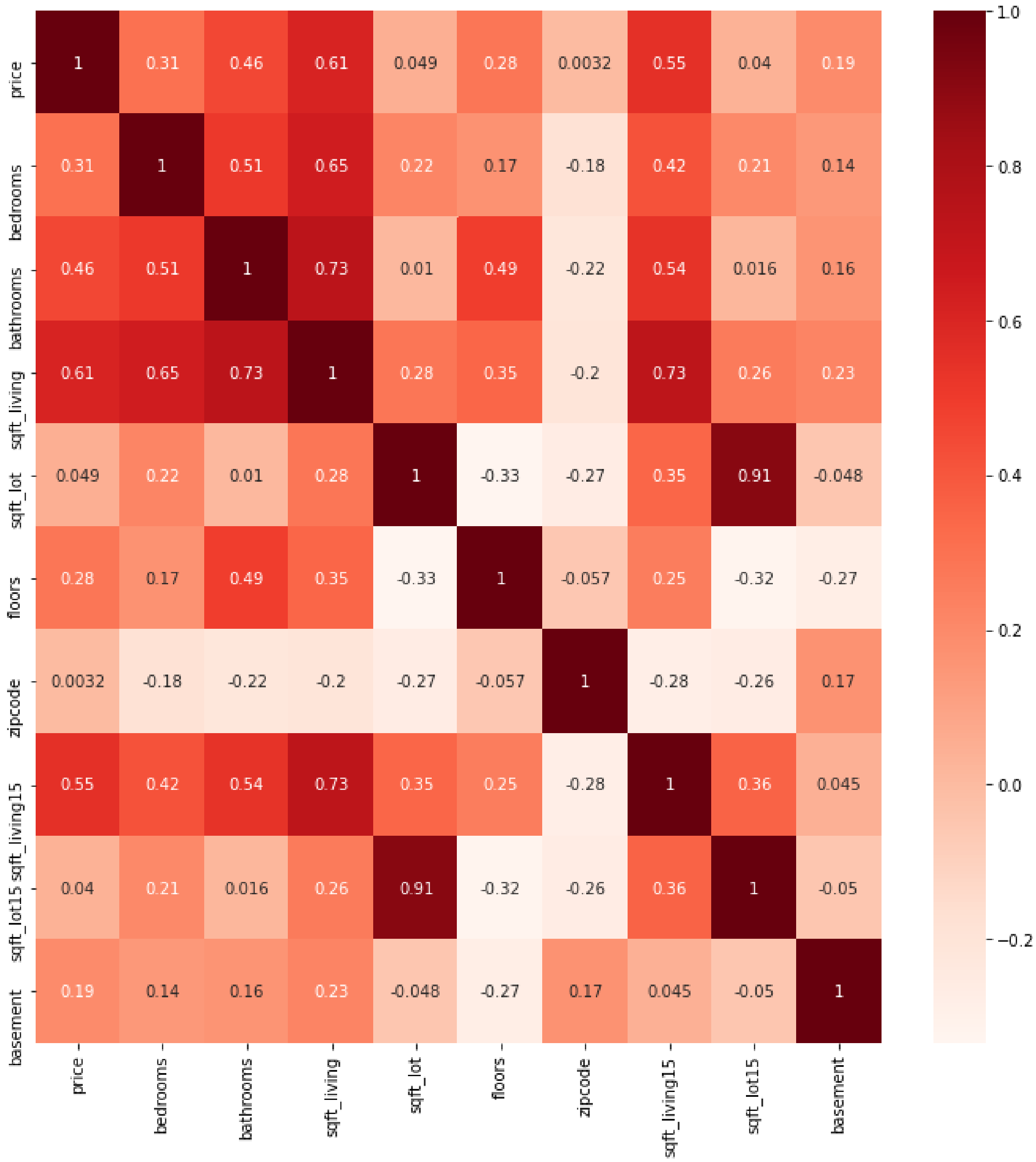
1. The model is generally statistically significant with an F-statistic p_value of 0.0 at a significance level of 0.05

2. The R-squared value is 0.457, indicating that approximately 45.7% of the variation in the price can be explained by the model. This value indicates a drop from the previous model.

3. The plot to test for homoscedasticity reveals that the residuals are now homoscedastic because they are converging and appear to be having an equal variance. So this assumption remains satisfied.

4. The QQ-plot is used to test for normality of residuals. In this case, the residuals appear to be almost perfectly normal as they are following along the line almost neatly.

In [59]:
```
# Displaying the heatmap
heatmap(df_log)
```



Iteration 4 (Final Model)

One hot encode Zipcode

In [60]:
```
# Define a function to perform feature scaling
def scale(feature):
    return (feature-feature.min())/(feature.max()-feature.min())

# OneHotEncoding zipcode
df_scale = scale_ohe('zipcode', df_log)
```

In [61]:
```
df_scale.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14582 entries, 1 to 21596
Data columns (total 78 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          14582 non-null  float64
 1   bedrooms       14582 non-null  float64
 2   bathrooms      14582 non-null  float64
 3   sqft_living    14582 non-null  float64
 4   sqft_lot       14582 non-null  float64
 5   floors         14582 non-null  float64
 6   sqft_living15  14582 non-null  float64
 7   sqft_lot15     14582 non-null  float64
 8   basement       14582 non-null  float64
 9   zipcode_98002  14582 non-null  uint8  
 10  zipcode_98003  14582 non-null  uint8  
 11  zipcode_98004  14582 non-null  uint8  
 12  zipcode_98005  14582 non-null  uint8  
 13  zipcode_98006  14582 non-null  uint8  
 14  zipcode_98007  14582 non-null  uint8  
 15  zipcode_98008  14582 non-null  uint8  
 16  zipcode_98010  14582 non-null  uint8  
 17  zipcode_98011  14582 non-null  uint8  
 18  zipcode_98014  14582 non-null  uint8  
 19  zipcode_98019  14582 non-null  uint8  
 20  zipcode_98022  14582 non-null  uint8  
 21  zipcode_98023  14582 non-null  uint8  
 22  zipcode_98024  14582 non-null  uint8  
 23  zipcode_98027  14582 non-null  uint8  
 24  zipcode_98028  14582 non-null  uint8  
 25  zipcode_98029  14582 non-null  uint8  
 26  zipcode_98030  14582 non-null  uint8  
 27  zipcode_98031  14582 non-null  uint8  
 28  zipcode_98032  14582 non-null  uint8  
 29  zipcode_98033  14582 non-null  uint8  
 30  zipcode_98034  14582 non-null  uint8  
 31  zipcode_98038  14582 non-null  uint8  
 32  zipcode_98039  14582 non-null  uint8  
 33  zipcode_98040  14582 non-null  uint8  
 34  zipcode_98042  14582 non-null  uint8  
 35  zipcode_98045  14582 non-null  uint8  
 36  zipcode_98052  14582 non-null  uint8  
 37  zipcode_98053  14582 non-null  uint8  
 38  zipcode_98055  14582 non-null  uint8  
 39  zipcode_98056  14582 non-null  uint8  
 40  zipcode_98058  14582 non-null  uint8  
 41  zipcode_98059  14582 non-null  uint8  
 42  zipcode_98065  14582 non-null  uint8  
 43  zipcode_98070  14582 non-null  uint8  
 44  zipcode_98072  14582 non-null  uint8  
 45  zipcode_98074  14582 non-null  uint8  
 46  zipcode_98075  14582 non-null  uint8  
 47  zipcode_98077  14582 non-null  uint8  
 48  zipcode_98092  14582 non-null  uint8  
 49  zipcode_98102  14582 non-null  uint8  
 50  zipcode_98103  14582 non-null  uint8  
 51  zipcode_98105  14582 non-null  uint8  
```

```
 52  zipcode_98106  14582 non-null  uint8
 53  zipcode_98107  14582 non-null  uint8
 54  zipcode_98108  14582 non-null  uint8
 55  zipcode_98109  14582 non-null  uint8
 56  zipcode_98112  14582 non-null  uint8
 57  zipcode_98115  14582 non-null  uint8
 58  zipcode_98116  14582 non-null  uint8
 59  zipcode_98117  14582 non-null  uint8
 60  zipcode_98118  14582 non-null  uint8
 61  zipcode_98119  14582 non-null  uint8
 62  zipcode_98122  14582 non-null  uint8
 63  zipcode_98125  14582 non-null  uint8
 64  zipcode_98126  14582 non-null  uint8
 65  zipcode_98133  14582 non-null  uint8
 66  zipcode_98136  14582 non-null  uint8
 67  zipcode_98144  14582 non-null  uint8
 68  zipcode_98146  14582 non-null  uint8
 69  zipcode_98148  14582 non-null  uint8
 70  zipcode_98155  14582 non-null  uint8
 71  zipcode_98166  14582 non-null  uint8
 72  zipcode_98168  14582 non-null  uint8
 73  zipcode_98177  14582 non-null  uint8
 74  zipcode_98178  14582 non-null  uint8
 75  zipcode_98188  14582 non-null  uint8
 76  zipcode_98198  14582 non-null  uint8
 77  zipcode_98199  14582 non-null  uint8
dtypes: float64(9), uint8(69)
memory usage: 2.1 MB
```

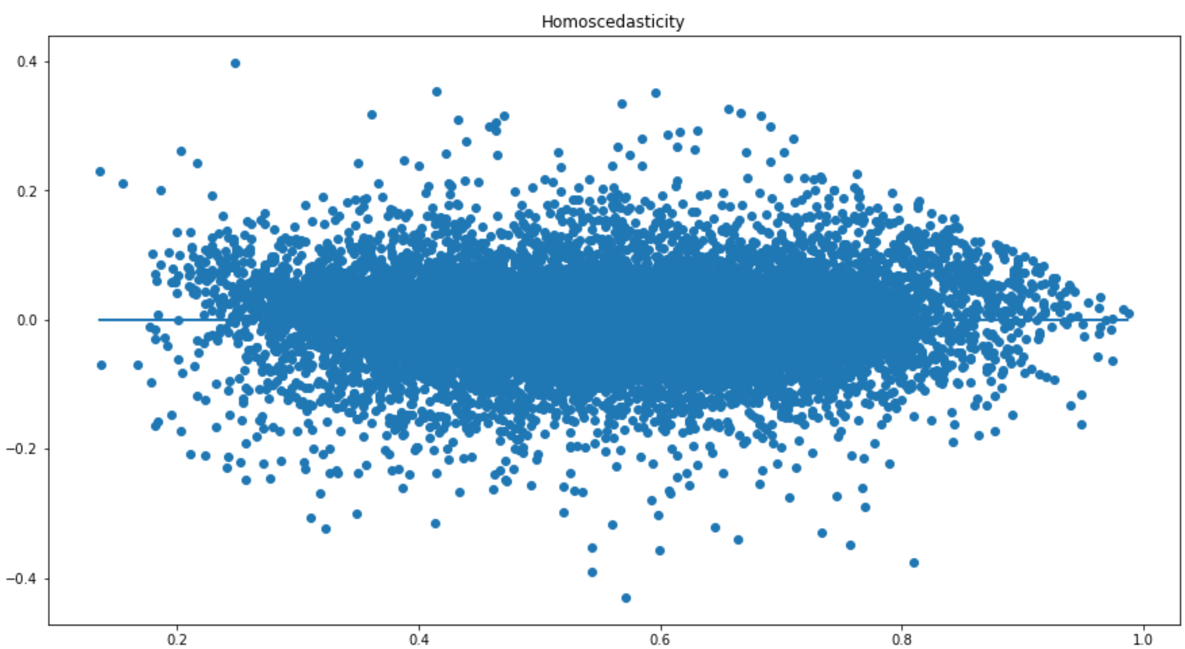
In [62]:
```
# Using our `df_scale` we fit our model using our defined function
run_model(df_scale)
```

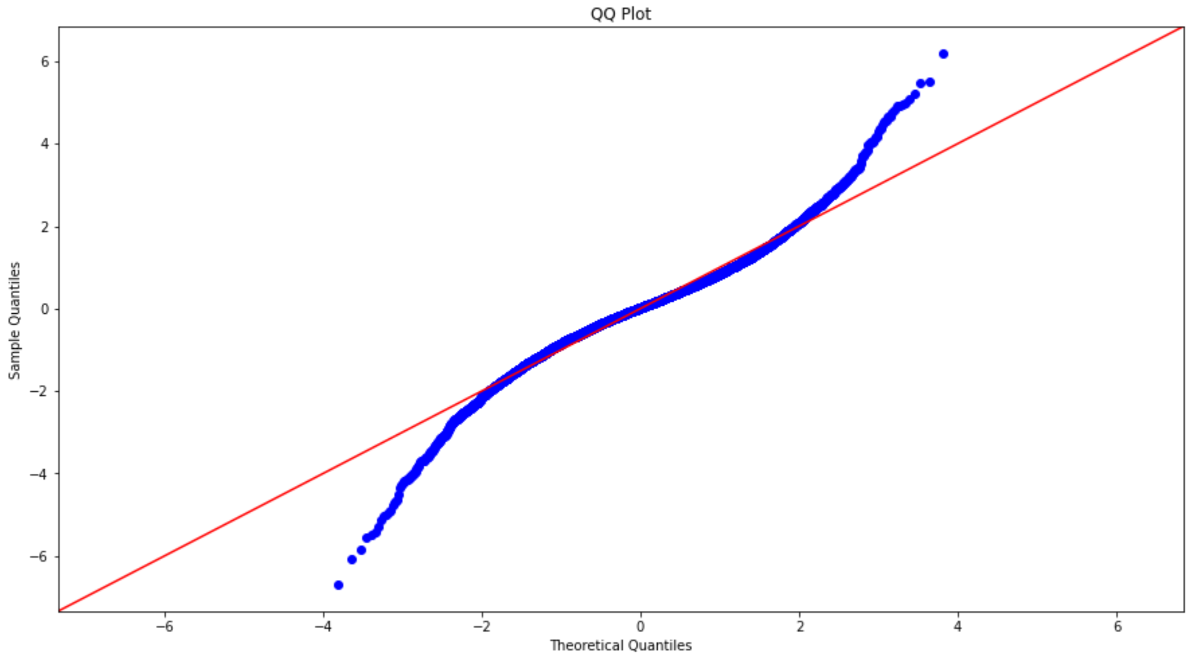
```
The mean r^2 for a KFold test with 10 splits is 0.8313639887029263

The mean RMSE for a KFold test with 10 splits is 0.06464512379839615

                            OLS Regression Results                            
==============================================================================
=
Dep. Variable:                  price   R-squared:                       0.83
3
Model:                            OLS   Adj. R-squared:                  0.83
2
Method:                 Least Squares   F-statistic:                     942.
1
Date:                Thu, 01 Jun 2023   Prob (F-statistic):              0.0
0
Time:                        12:27:36   Log-Likelihood:                 1933
4.
No. Observations:               14582   AIC:                        -3.851e+0
4
Df Residuals:                   14504   BIC:                        -3.792e+0
4
Df Model:                          77                                         
Covariance Type:            nonrobust                                         
==============================================================================
====
                    coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
Intercept        -0.0694      0.006    -12.052      0.000      -0.081      -
0.058
bedrooms         -0.0545      0.005    -11.575      0.000      -0.064      -
0.045
bathrooms         0.0552      0.005     11.066      0.000       0.045       
0.065
sqft_living       0.4423      0.008     58.880      0.000       0.428       
0.457
sqft_lot          0.1140      0.010     11.398      0.000       0.094       
0.134
floors            0.0086      0.003      2.830      0.005       0.003       
0.015
sqft_living15     0.1799      0.006     29.385      0.000       0.168       
0.192
sqft_lot15        0.0042      0.009      0.459      0.646      -0.014       
0.022
basement         -0.0154      0.001    -10.806      0.000      -0.018      -
0.013
zipcode_98002     0.0119      0.007      1.763      0.078      -0.001       
0.025
zipcode_98003     0.0189      0.006      3.136      0.002       0.007       
0.031
zipcode_98004     0.3834      0.007     58.608      0.000       0.371       
0.396
zipcode_98005     0.2705      0.007     37.036      0.000       0.256       
0.285
zipcode_98006     0.2437      0.006     43.357      0.000       0.233       
0.255
```

```
zipcode_98007     0.2402      0.008     31.230      0.000       0.225
0.255
zipcode_98008     0.2428      0.006     39.884      0.000       0.231
0.255
zipcode_98010     0.0875      0.010      9.029      0.000       0.069
0.107
zipcode_98011     0.1522      0.007     22.060      0.000       0.139
0.166
zipcode_98014     0.1025      0.009     10.797      0.000       0.084
0.121
zipcode_98019     0.0964      0.007     13.001      0.000       0.082
0.111
zipcode_98022     0.0250      0.007      3.599      0.000       0.011
0.039
zipcode_98023     0.0075      0.005      1.410      0.158      -0.003
0.018
zipcode_98024     0.1459      0.013     11.346      0.000       0.121
0.171
zipcode_98027     0.1944      0.006     33.116      0.000       0.183
0.206
zipcode_98028     0.1465      0.006     24.215      0.000       0.135
0.158
zipcode_98029     0.2203      0.006     37.343      0.000       0.209
0.232
zipcode_98030     0.0226      0.006      3.588      0.000       0.010
0.035
zipcode_98031     0.0305      0.006      4.971      0.000       0.018
0.042
zipcode_98032     0.0124      0.008      1.613      0.107      -0.003
0.028
zipcode_98033     0.2797      0.006     50.254      0.000       0.269
0.291
zipcode_98034     0.1965      0.005     37.396      0.000       0.186
0.207
zipcode_98038     0.0516      0.005      9.826      0.000       0.041
0.062
zipcode_98039     0.4344      0.019     22.777      0.000       0.397
0.472
zipcode_98040     0.3286      0.006     50.565      0.000       0.316
0.341
zipcode_98042     0.0289      0.005      5.512      0.000       0.019
0.039
zipcode_98045     0.1170      0.007     16.992      0.000       0.103
0.130
zipcode_98052     0.2313      0.005     44.223      0.000       0.221
0.242
zipcode_98053     0.2062      0.006     35.192      0.000       0.195
0.218
zipcode_98055     0.0582      0.006      9.434      0.000       0.046
0.070
zipcode_98056     0.1253      0.006     22.611      0.000       0.114
0.136
zipcode_98058     0.0624      0.005     11.394      0.000       0.052
0.073
zipcode_98059     0.1194      0.006     21.593      0.000       0.109
0.130
zipcode_98065     0.1376      0.006     22.090      0.000       0.125
```

```
0.150
zipcode_98070      0.1627      0.010     16.443      0.000       0.143
0.182
zipcode_98072      0.1648      0.006     26.083      0.000       0.152
0.177
zipcode_98074      0.2082      0.006     36.642      0.000       0.197
0.219
zipcode_98075      0.2095      0.006     35.390      0.000       0.198
0.221
zipcode_98077      0.1607      0.007     22.127      0.000       0.146
0.175
zipcode_98092      0.0135      0.006      2.263      0.024       0.002
0.025
zipcode_98102      0.3627      0.010     38.009      0.000       0.344
0.381
zipcode_98103      0.3123      0.005     58.738      0.000       0.302
0.323
zipcode_98105      0.3547      0.007     53.284      0.000       0.342
0.368
zipcode_98106      0.1464      0.006     24.548      0.000       0.135
0.158
zipcode_98107      0.3255      0.006     52.372      0.000       0.313
0.338
zipcode_98108      0.1429      0.007     20.479      0.000       0.129
0.157
zipcode_98109      0.3697      0.009     42.620      0.000       0.353
0.387
zipcode_98112      0.3825      0.007     58.276      0.000       0.370
0.395
zipcode_98115      0.3066      0.005     58.043      0.000       0.296
0.317
zipcode_98116      0.3017      0.006     51.243      0.000       0.290
0.313
zipcode_98117      0.3097      0.005     58.388      0.000       0.299
0.320
zipcode_98118      0.1845      0.005     34.233      0.000       0.174
0.195
zipcode_98119      0.3771      0.007     52.277      0.000       0.363
0.391
zipcode_98122      0.3087      0.006     49.448      0.000       0.296
0.321
zipcode_98125      0.2109      0.006     37.772      0.000       0.200
0.222
zipcode_98126      0.2306      0.006     39.374      0.000       0.219
0.242
zipcode_98133      0.1740      0.005     32.208      0.000       0.163
0.185
zipcode_98136      0.2746      0.006     43.947      0.000       0.262
0.287
zipcode_98144      0.2600      0.006     43.693      0.000       0.248
0.272
zipcode_98146      0.1162      0.006     19.196      0.000       0.104
0.128
zipcode_98148      0.0495      0.011      4.554      0.000       0.028
0.071
zipcode_98155      0.1584      0.006     28.685      0.000       0.148
0.169
```

```
zipcode_98166     0.1393      0.006     21.996      0.000       0.127
0.152
zipcode_98168     0.0378      0.006      6.052      0.000       0.026
0.050
zipcode_98177     0.2331      0.006     36.573      0.000       0.221
0.246
zipcode_98178     0.0762      0.006     12.109      0.000       0.064
0.089
zipcode_98188     0.0381      0.008      4.926      0.000       0.023
0.053
zipcode_98198     0.0475      0.006      7.728      0.000       0.035
0.060
zipcode_98199     0.3261      0.006     53.011      0.000       0.314
0.338
=============================================================================
=
Omnibus:                      981.961   Durbin-Watson:                   1.99
0
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4604.77
8
Skew:                          -0.124   Prob(JB):                         0.0
0
Kurtosis:                       5.742   Cond. No.                         12
1.
=============================================================================
=

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





In [63]:
```
def get_coefficients_categorical(scaled_coefs, features):
    for i, feat in enumerate(features):
        maximum = df_log['price'].max()
        minimum = df_log['price'].min()
        unscale = abs(scaled_coefs[i])*(maximum-minimum)+minimum
        unlog = math.exp(unscale)
        if scaled_coefs[i] >= 0:
            print('Coefficient for {} is ${}'.format(feat, unlog))
        else:
            print('Coefficient for {} is ${}'.format(feat, unlog*-1))
```

In [64]:
```
def get_coefficients_continuous(scaled_coefs, features):
    for i, feat in enumerate(features):
        maximum = df_log['price'].max()
        minimum = df_log['price'].min()
        range_feat = df_no_outlier[feat].max() - df_no_outlier[feat].min()
        unscale = abs(scaled_coefs[i])*(maximum-minimum)+minimum
        unlog = math.exp(unscale)

        slope_actual = unlog/range_feat

        if scaled_coefs[i] >= 0:
            print('Coefficient for {} is ${}'.format(feat, slope_actual))
        else:
            print('Coefficient for {} is ${}'.format(feat, slope_actual*-1))
```

In [79]:
```
categorical_coef = [0.2428, 0.2082, 0.3097, 0.2600, 0.2331]
categorical_features = ['zipcode_98008', 'zipcode_98074', 'zipcode_98117', 'zi

continuous_coef = [0.4423, 0.1799, -0.0545, 0.0086, 0.0552, 0.1140]
continuous_features = ['sqft_living', 'sqft_living15', 'bedrooms', 'floors', '

get_coefficients_categorical(categorical_coef, categorical_features)
get_coefficients_continuous(continuous_coef, continuous_features)
```

```
Coefficient for zipcode_98008 is $169959.30663666
Coefficient for zipcode_98074 is $153192.66292287616
Coefficient for zipcode_98117 is $207759.30309087687
Coefficient for zipcode_98144 is $178964.988254935
Coefficient for zipcode_98177 is $165081.8589739885
Coefficient for sqft_living is $123487.74911877913
Coefficient for sqft_living15 is $66394.5581188671
Coefficient for bedrooms is $-53899.42895874723
Coefficient for floors is $42072.219101705305
Coefficient for bathrooms is $46540.35864547536
Coefficient for sqft_lot is $20891.42559555274
```

Train Test Split

In [65]:
```
# Getting a copy of our df
df_tts = df.copy()
x = df_tts.drop('price', axis=1)
y = df_tts['price']
```

Split original data into training data (80%) and testing data (20%).

In [66]:
```
# Split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20)
```

Concat x with y to remove outliers

In [67]:
```
# Concat x with y to remove outliers
train = pd.concat([x_train, y_train], axis=1)
test = pd.concat([x_test, y_test], axis=1)
len(train)
```

Out[67]: 12540

Remove outliers separately

In [68]:
```
# Remove outliers separately
train1 = outliers(continuous, train)
test1 = outliers(continuous, test)
len(train1)
```

Out[68]: 11646

Log transform train and test splits

In [69]:
```
# Log transform train and test splits
train2 = log_transform(normalize, train1)
test2 = log_transform(normalize, test1)
```

Scale and OHE training and testing data separately

In [70]:
```
# Scale and OHE training and testing data separately
train_preprocessed = scale_ohe('zipcode', train2)

test_preprocessed = scale_ohe('zipcode', test2)
```

Drop features determined by our final model

In [71]:
```
# Drop features determined by our final model
train_preprocessed.drop(['sqft_lot15', 'zipcode_98002', 'zipcode_98023', 'zipc
test_preprocessed.drop(['sqft_lot15', 'zipcode_98002', 'zipcode_98023', 'zipco
```

Apply interactions determmined by our final model

In [72]:
```
# Apply interactions determmined by our final model
train_preprocessed['sqft_living*floors'] = train_preprocessed['sqft_living']*t
test_preprocessed['sqft_living*floors'] = test_preprocessed['sqft_living']*tes
```

Check to see that the training and testing sets are split correctly

In [80]:
```
# Check to see that the training and testing sets are split correctly
x_train_preprocessed = train_preprocessed.drop('price', axis=1)
y_train_preprocessed = train_preprocessed['price']

x_test_preprocessed = test_preprocessed.drop('price', axis=1)
y_test_preprocessed = test_preprocessed['price']

print(len(x_train_preprocessed), len(x_test_preprocessed), len(y_train_preproc
```

```
11646 2902 11646 2902
```

Run testing data through training model

In [84]:
```
# Run testing data through training model
linreg = LinearRegression()
linreg.fit(x_train_preprocessed, y_train_preprocessed)
y_hat_test = linreg.predict(x_test_preprocessed)

test_rmse = mean_squared_error(y_test_preprocessed, y_hat_test, squared=False)
test_rmse
```

Out[84]:
```
0.06463831997659172
```

In [83]:
```
# Calculate evaluation metrics on the original scale
y_pred_original = np.exp(y_hat_test)  # Transform predicted values back to the
y_test_original = np.exp(y_test_preprocessed)  # Transform actual values back t

rmse_original = mean_squared_error(y_test_original, y_pred_original, squared=Fa

print("RMSE in original scale:", rmse_original)
```

```
RMSE in original scale: 0.11351287323753316
```

CONCLUSIONS

Interpretation of results from the Final Model

1. The model is generally statistically significant with an F-statistic p_value of 0.0 at a significance level of 0.05

2. The R-squared value is 0.833, indicating that approximately 83.3% of the variation in the price can be explained by the model. This value indicates a great improvement from the previous model.

3. Also, of great importance to note is that the mean RMSE is approximately 0.06465. Then the RMSE in original scale is 0.1135. This means that our model is off by about 0.1135 when making an average prediction, indicating that it is a good model.

4. These coefficients represent the expected change in the price for a one-unit change in the corresponding predictor variable, assuming other variables are held constant.

- ZIPCODE--is a strong predictor of a homes value, the saying "Location, Location, Location" holds true, as even in a similar area the location plays a huge factor in the value of a home.

Based on the coefficients of different localities, moving from zip code 98002 to 98039 shows that the prices changes by USD 228,087 and USD 298,174 respectively, as compared to our reference categorical variable which is zipcode 98001. This is a clear indication that locality of the house has high influence on the price.

- Coefficient for `sqft_living` is $123487.74911877913
  - For a one-unit increase in square-foot living area, we see an associated increase in around $123487.74 in selling price of the houses.

- Coefficient for `sqft_living15` is $66394.5581188671
  - For a one-unit increase in square-foot living area15, we see an associated increase in around $66394.55 in selling price of the houses.

- Coefficient for `floors` is $42072.219101705305
  - For a one-unit increase in number of floors of the house, we see an associated increase in around $42072.21 in selling price of the houses.

- Coefficient for `bathrooms` is $46540.35864547536
  - For a one-unit increase in the number of bathrooms, we see an associated increase in around $46540.35 in selling price of the houses.

- Coefficient for `sqft_lot` is $20891.42559555274
  - For a one-unit increase in square-foot of the lot area, we see an associated increase in around $20891.42 in selling price of the houses.
- Coefficient for `bedrooms` is $-53899.42895874723
  - For a one-unit increase in the number of bedrooms, we see an associated decrease in around $53899.42 in selling price of the houses. This particular finding caught our attention as this is not the case in the real world, whereby as you increase the number of bedrooms in a house, the price of the house tends to increase too.

5. The plot to test for homoscedasticity reveals that the residuals are now homoscedastic because they are converging and appear to be having an equal variance. So this assumption remains satisfied.
6. The QQ-plot is used to test for normality of residuals. In this case, the residuals appear to be almost normal as they are following along the line almost neatly, except for the ends where it indicates there could be some skewness in the data.

RECOMMENDATIONS

1. The real estate agency should explore properties that occupy a large square foot of the lot area since, for a one-unit increase in square-foot of the lot area, we see an associated increase in around $ 20891.42 in selling price of the houses.
2. The real estate agency should explore properties that have more bathrooms since, for a one-unit increase in the number of bathrooms, we see an associated increase in around $ 46540.35 in selling price of the houses.
3. The real estate agency should explore properties that occupy a large square foot of living area since, for a one-unit increase in square-foot living area, we see an associated increase in around $ 123487.74 in selling price of the houses.
4. The real estate agency should explore properties with more floors since, for a one-unit increase in number of floors of the house, we see an associated increase in around $ 42072.21 in selling price of the houses.

NEXT STEPS

1. More research is required to have a more integrated and informative dataset for finding more factors that influence the price. Also, use of more complex and robust regression models that will help to deal with the outliers.
2. Using datasets from other counties to be able to better advice our customers from comparing the dataset results.
3. It is also important for the agency to continuously evaluate the effectiveness of the strategies they implement and make adjustments as necessary. This could involve tracking

metrics like, this model, social media engagement/reviews, and lead generation to assess the impact of their efforts and identify areas for improvement.