Final Project Submission - Developing predictions for house prices in King County

Please fill out:

- Student name: **Group 3** (Vivian Watiri, Wilfred Njagi, Antony Kanai, Sheila Kamanzi, Eunita Nyengo, Bryan Okwach, Cynthia Nasimiyu)
- Student pace: **Part time**
- Scheduled project review date/time:**2/6/2023**

Business Understanding

Brief History and Information on the General Housing Market

The housing market has been shaped by significant events and trends over the years. The Subprime Mortgage Crisis of 2007-2008 triggered a global financial crisis, causing a downturn in the housing market and widespread foreclosures. Post-crisis recovery varied across regions, with some areas rebounding faster than others. Urbanization has driven housing demand, leading to price appreciation and high-density housing development. Governments intervene through policies to promote homeownership and address housing shortages. General trends in the housing market include supply and demand dynamics, price appreciation over the long term, regional disparities, the impact of interest rates and mortgage markets, and housing affordability challenges.

Problem Statement

The project conducted by group 3 focuses on the analysis of the King County House Sales dataset through multiple linear regression. The main goal is to gain insights into the housing market and develop accurate predictions for house prices in King County, Washington, United States. By examining the relationships between various features of a house and its corresponding sale price, the project aims to provide a deeper understanding of the factors that influence housing prices in the area.

Stakeholder

The stakeholder in this project is a real estate agent. The project aims to empower the real estate agent with data-driven information to help them provide expert advice to their clients, whether they are buyers or sellers. With a better understanding of the significant factors influencing house prices in King County, the real estate agent can assist buyers in determining the appropriateness of a property's price and help sellers set competitive prices.

In conclusion, by leveraging the insights gained from this analysis, the stakeholder can make more informed decisions, improve market efficiency, and contribute to the overall development of the real estate sector in King County.

Data Understanding

The data used for this project is in csv format. It is data on feautures that affect house prices in Kingcounty. The data understanding section below explores more on this data set.

In [321]:
```
#importing the relevant libraries
import pandas as pd
import numpy as np
import math
import scipy.stats as stats
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns


import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from statsmodels.stats.diagnostic import het_breuschpagan
from sklearn.metrics import mean_absolute_error, mean_squared_error
from statsmodels.stats.stattools import jarque_bera
from scipy.stats import kstest

import warnings

# importing the dataset
kingcounty_df = pd.read_csv("kc_house_data.csv")
kingcounty_df.sample(3)
```

Out[321]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterf
596	3278600240	7/22/2014	372500.0	2	2.50	1400	2958	2.0	
685	2346200030	1/5/2015	802541.0	5	2.75	2990	6768	2.0	
2784	7631200292	6/26/2014	669000.0	2	1.75	1950	10766	1.0	

3 rows × 21 columns

In [322]:
```
# check for number of rows and columns
kingcounty_df.shape
```

Out[322]:
```
(21597, 21)
```

Our dataset has 21597 rows and 21 columns.

In [323]:
```
#checking the datatypes of the columns
kingcounty_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
```

- Six columns have their datatypes as strings while 15 are numeric. There is a date column which is among the six columns whose data type is an object this will be converted to date object to facilitate feature engineering.
- Sqft_basement has its datatype as strings yet it should be numeric, therefore needs transformation. - The grade column mixes two datatypes.
- The other three, waterfront, view, and condition are have categorical data.

Checking for general Statistics of the data

In [324]:
```
kingcounty_df.describe()
```

Out[324]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	215
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	
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	

A quick sneak peak of some of the columns in our dataset reveals the following statistics:

- That the Average price of a house in King county is USD 540,296.6 with the maximum price being USD 7,700,000 and the minimum price being USD 78,000. The standard deviation of the price is USD 367,368.
- The average number of bedrooms of a house in King County is 3 bedrooms with the maximum number of bedrooms being 33 and the minimum being 1. The standard deviation is 0.93.
- For the number of bathrooms, the average number of bathrooms is 2. The house that has the highest number of bathrooms has 8 while the one that has the least has 0.5. The standard deviation is 0.77.
- The average number of floors in house in King county is 1.49 with the highest having 3.5 floors and the lowest having 1 with a spread of 0.5

Checking proportion of null Values in our dataset

In [325]:
```
kingcounty_df.isna().mean()*100
```

Out[325]:
```
id               0.000000
date             0.000000
price            0.000000
bedrooms         0.000000
bathrooms        0.000000
sqft_living      0.000000
sqft_lot         0.000000
floors           0.000000
waterfront      11.001528
view             0.291707
condition        0.000000
grade            0.000000
sqft_above       0.000000
sqft_basement    0.000000
yr_built         0.000000
yr_renovated    17.789508
zipcode          0.000000
lat              0.000000
long             0.000000
sqft_living15    0.000000
sqft_lot15       0.000000
dtype: float64
```

From the above output:

- 11% of the data in the waterfront column consists of null values
- 0.2% of the data in the view column consist of null values
- 17.78% of the data in the Yr-Renovated consists of missing values

Data Preparation

Dealing with Misssing Data

Currently the following columns in the dataset contain missing data.

- Water front - 2376,
- view - 63
- yr_renovated - 3842

The rows containing missing columns identified above will be dropped based on the following justifications:

- Dropping the rows will not result to the loss of a proportionate percentage of the whole large dataset since the missing values constitute a small percentage
- Replacing the categorical variables with any form of measure of central tendency would result to data that might potentially lead to bias in the model.

In [326]:
```
# droppin the rows missing data
kingcounty_df.dropna(inplace = True)
```

Dealing with duplicates

In [327]:
```
# checking for duplicates using the id column
duplicates = kingcounty_df[kingcounty_df.duplicated(
subset = "id")]
duplicates.head()
```

Out[327]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
94	6021501535	12/23/2014	700000.0	3	1.50	1580	5000	1.0	
314	4139480200	12/9/2014	1400000.0	4	3.25	4290	12103	1.0	
718	8820903380	1/2/2015	730000.0	6	2.25	2660	13579	2.0	
837	8682262400	5/13/2015	419950.0	2	1.75	1350	4003	1.0	
1085	9834200885	4/20/2015	550000.0	4	2.50	2080	4080	1.0	

5 rows × 21 columns

In [328]:
```
duplicates.shape
```

Out[328]:
```
(86, 21)
```

86 rows contain duplicated data. All duplicate rows present in the DataFrame were dropped.

In [329]:
```
# dropping the duplicated data
kingcounty_df.drop_duplicates(subset="id", keep="first", inplace=True)
```

Dealing with Placeholders

In [330]:
```
# changing the sqft_basement column from categorical to numerical
kingcounty_df["sqft_basement"] = pd.to_numeric(kingcounty_df["sqft_basement"],
kingcounty_df["sqft_basement"].dtypes
```

Out[330]:
```
dtype('float64')
```

In [331]:
```
# missing data that was originally in filled by symbols
kingcounty_df["sqft_basement"].isna().sum()
```

Out[331]:
```
332
```

The sqft_basement is missing **332** data points

In [332]:
```
# Drop the missing data in the above column
kingcounty_df.dropna(inplace = True)
```

In [333]:
```
kingcounty_df.shape
```

Out[333]:
```
(15344, 21)
```

Data Transformation

Using ordinal encoding to convert categorical variables into numerical

In [334]:
```
#Checking for the ordinal data used in the "view" column
kingcounty_df["view"].unique()
```

Out[334]:
```
array(['NONE', 'GOOD', 'EXCELLENT', 'AVERAGE', 'FAIR'], dtype=object)
```

In [335]:
```
# Using ordinal encoding to convert the ordinal range into numerical variables
# create a mapping dictionary
mapping_dict = {"NONE": 1, "FAIR": 2, "AVERAGE": 3, "GOOD": 4, "EXCELLENT": 5,

# map the ordinal column to the appropriate numerical values
kingcounty_df['view_num'] = kingcounty_df['view'].map(mapping_dict)
```

In [336]:
```
#Using ordinal encoding to create numerical representation of the "waterfront"
# Map 'yes' to 1 and 'no' to 0
kingcounty_df["waterfront_num"] = kingcounty_df["waterfront"].map({'YES': 1, '

kingcounty_df.head(5)

kingcounty_df.dtypes
```

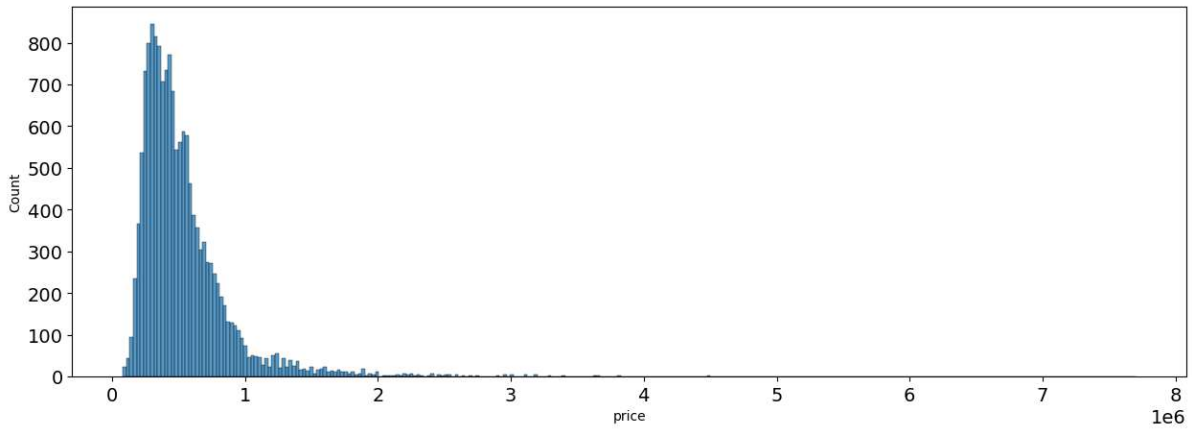
Out[336]:
```
id                 int64
date              object
price            float64
bedrooms           int64
bathrooms        float64
sqft_living        int64
sqft_lot           int64
floors           float64
waterfront        object
view              object
condition         object
grade             object
sqft_above         int64
sqft_basement    float64
yr_built           int64
yr_renovated     float64
zipcode            int64
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
view_num           int64
waterfront_num     int64
dtype: object
```

Exploratory Data Analysis

Distribution of target variable

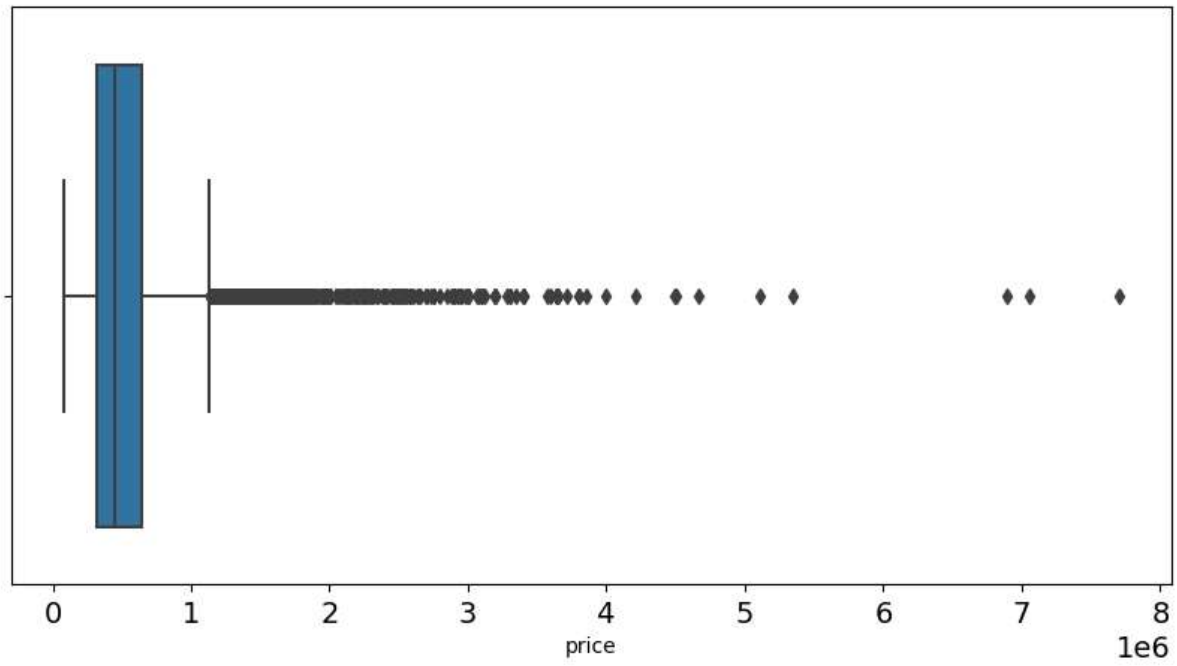
Checking the distribution of the target variable which is the **price** of houses in King County

In [337]:
```
#setting figure size
plt.figure(figsize=(15,5))

#Plotting a histogram to check the distribution of the price
sns.histplot(kingcounty_df['price'])
plt.show()
```



The data distribution is skewed to the right. The boxplot below allows us to investigate whether this is caused by outliers

In [338]:
```
plt.figure(figsize=(10,5))
sns.boxplot(x=kingcounty_df['price'])
plt.show;
```



Linearity assumption check

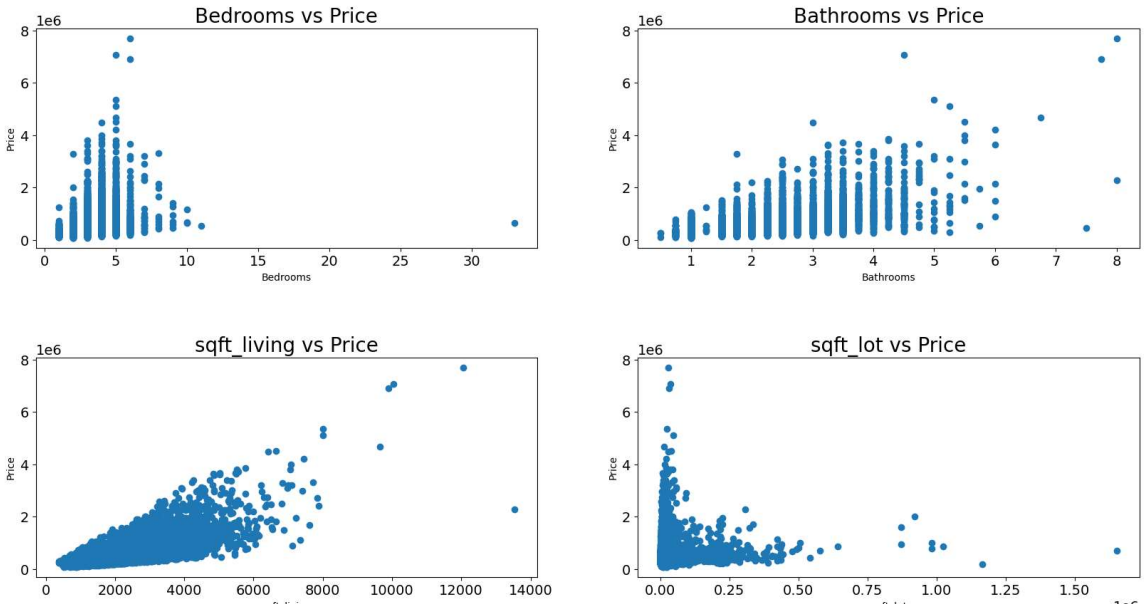
In this section of EDA, we will investigate which features are correlated with the target i.e **price**

The linearity check will require that their is a linear relationship between the response variable/target and the predictor variable

In [339]:
```
#Plotting the independent variables to the target

fig, ax = plt.subplots(7,2, figsize=(20,40))
y = kingcounty_df["price"]
plt.subplots_adjust(hspace = 0.5)

#adjusting the font sizes in the scatterplots
plt.rcParams['axes.titlesize'] = 20
plt.rcParams['axes.labelsize'] = 18
plt.rcParams['xtick.labelsize'] = 14
plt.rcParams['ytick.labelsize'] = 14


ax[0,0].scatter(kingcounty_df['bedrooms'],y)
ax[0,0].set_title ("Bedrooms vs Price")
ax[0,0].set_xlabel("Bedrooms")
ax[0,0].set_ylabel("Price")

ax[0,1].scatter(kingcounty_df['bathrooms'],y)
ax[0,1].set_title ("Bathrooms vs Price")
ax[0,1].set_xlabel("Bathrooms")
ax[0,1].set_ylabel("Price")

ax[1,0].scatter(kingcounty_df['sqft_living'],y)
ax[1,0].set_title ("sqft_living vs Price")
ax[1,0].set_xlabel("sqft_living")
ax[1,0].set_ylabel("Price")

ax[1,1].scatter(kingcounty_df['sqft_lot'],y)
ax[1,1].set_title ("sqft_lot vs Price")
ax[1,1].set_xlabel("sqft_lot")
ax[1,1].set_ylabel("Price")

ax[2,0].scatter(kingcounty_df['floors'],y)
ax[2,0].set_title ("Floors vs Price")
ax[2,0].set_xlabel("Floors")
ax[2,0].set_ylabel("Price")

ax[2,1].scatter(kingcounty_df['waterfront_num'],y)
ax[2,1].set_title ("Waterfront vs Price")
ax[2,1].set_xlabel("Waterfront")
ax[2,1].set_ylabel("Price")

ax[3,0].scatter(kingcounty_df['view_num'],y)
ax[3,0].set_title ("View vs Price")
ax[3,0].set_xlabel("View")
ax[3,0].set_ylabel("Price")

ax[3,1].scatter(kingcounty_df['sqft_above'],y)
ax[3,1].set_title ("sqft_above vs Price")
ax[3,1].set_xlabel("sqft_above")
ax[3,1].set_ylabel("Price")

ax[4,0].scatter(kingcounty_df['sqft_basement'],y)
ax[4,0].set_title ("sqft_basement vs Price")
ax[4,0].set_xlabel("sqft_basement")
ax[4,0].set_ylabel("Price")
```

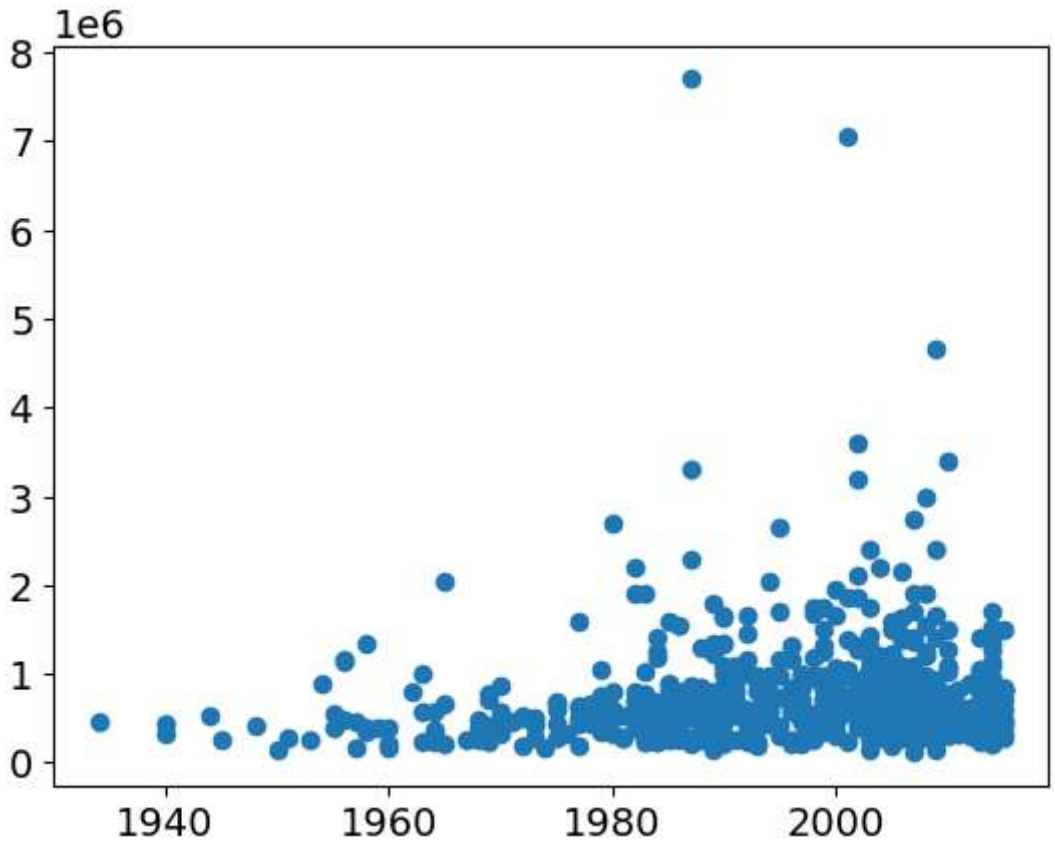
```
ax[4,1].scatter(kingcounty_df['yr_built'],y)
ax[4,1].set_title ("Year Built vs Price")
ax[4,1].set_xlabel("Year Built")
ax[4,1].set_ylabel("Price")

ax[5,0].scatter(kingcounty_df['yr_renovated'],y)
ax[5,0].set_title ("Year Renovated vs Price")
ax[5,0].set_xlabel("Year Renovated")
ax[5,0].set_ylabel("Price")

ax[5,1].scatter(kingcounty_df['sqft_living15'],y)
ax[5,1].set_title ("sqft_living 15 vs Price")
ax[5,1].set_xlabel("sqft_living15")
ax[5,1].set_ylabel("Price")

ax[6,0].scatter(kingcounty_df['sqft_lot15'],y)
ax[6,0].set_title ("sqft_lot15 vs Price")
ax[6,0].set_xlabel("sqft_lot15")
ax[6,0].set_ylabel("Price")
```

Out[339]: Text(0, 0.5, 'Price')



In [340]:
```
# create a mask for only the rows where 'years renovated' is not 0
mask = kingcounty_df['yr_renovated'] != 0

# use the mask to subset the DataFrame
renovated_houses = kingcounty_df[mask]
renovated_houses.dropna()
plt.scatter(renovated_houses["yr_renovated"],renovated_houses["price"])
```

Out[340]:
```
<matplotlib.collections.PathCollection at 0x1ef8079d1f0>
```



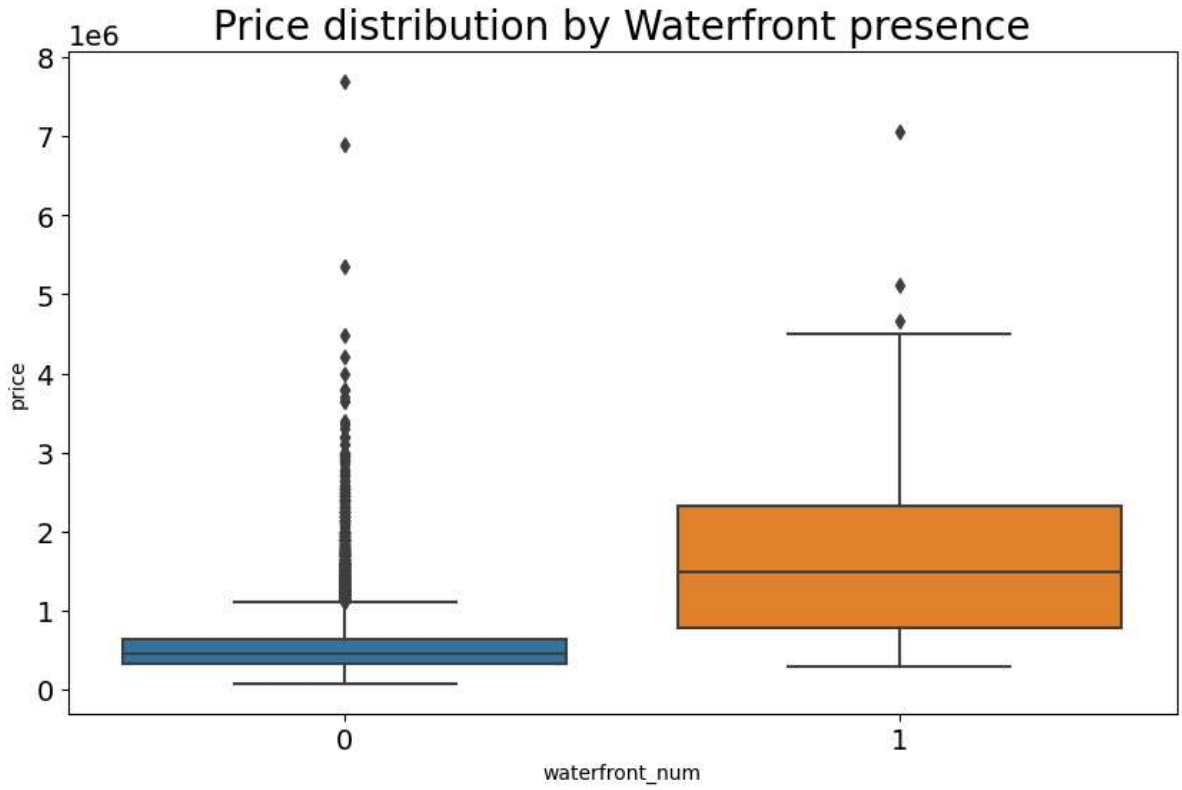
From the scatterplots, the following independent variables show a linear relationship in relationship to **price** which is the target variable:

1. Bathrooms
2. Sqft_living
3. Sqft_above
4. Sq_ft living 15

Bedrooms as an independent variable seems to be affected by an outlier

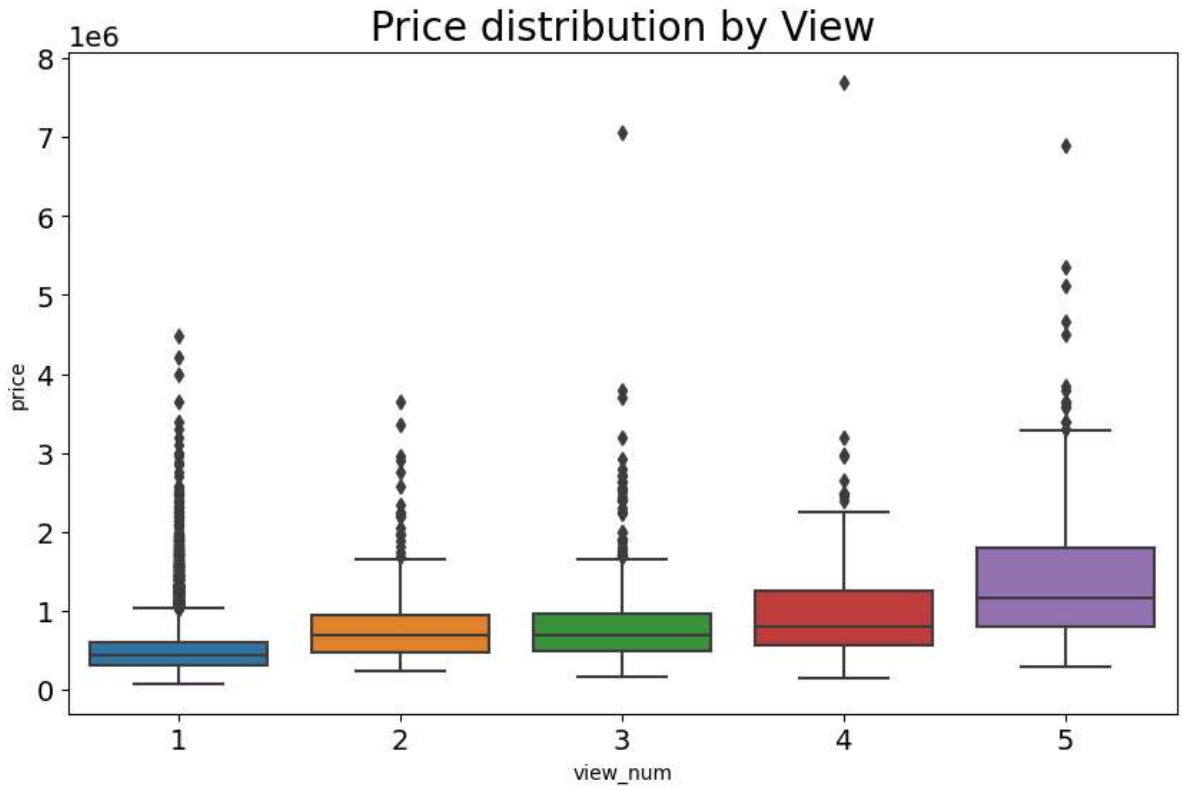
Using boxplots to check for linearity of discrete data

In [341]:
```
# Assuming 'waterfront' is a binary variable indicating presence (Yes) or absen
# If it's not, you'll have to preprocess your data to convert 'waterfront' into

plt.figure(figsize=(10,6))

sns.boxplot(x='waterfront_num', y='price', data=kingcounty_df)

plt.title('Price distribution by Waterfront presence')
plt.show()
```



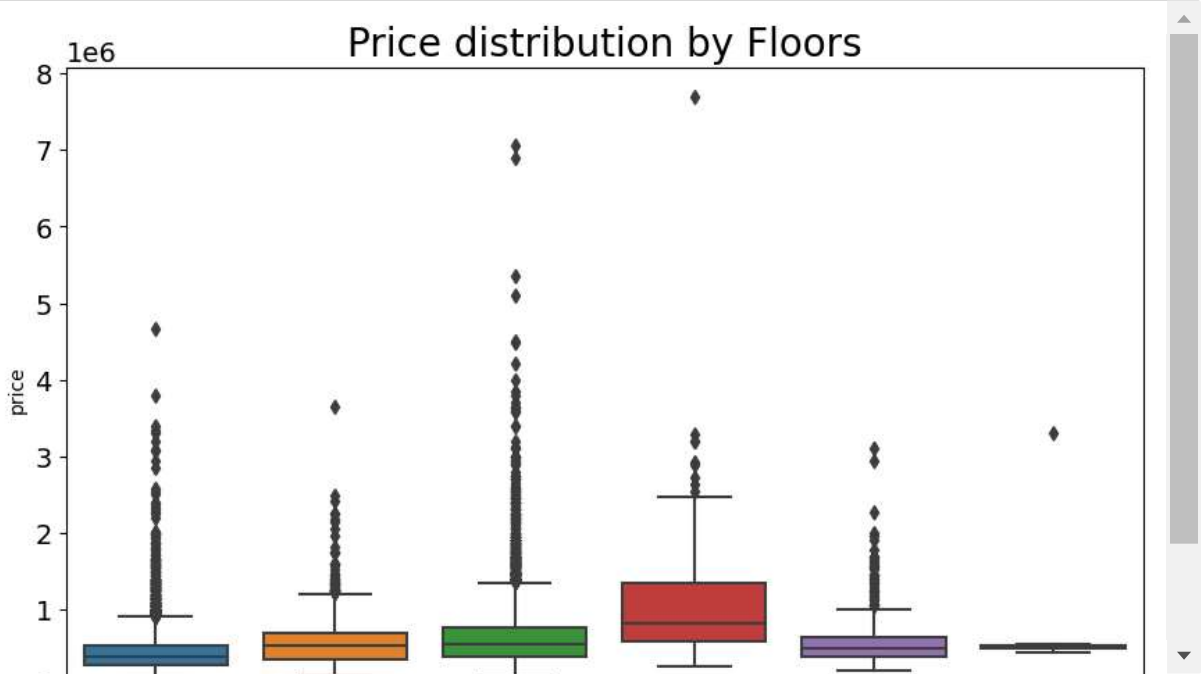
From the box plot above, houses with a waterfront had a higher average pice compared to those without a waterfront

In [342]:
```
plt.figure(figsize=(10,6))

sns.boxplot(x='view_num', y='price', data=kingcounty_df)

plt.title('Price distribution by View')
plt.show()
```



The boxplot above shows that view of a house had a postive relationship with the house price. Excellent view fetched the highest aveage price while houses with no view had the least aveage price

In [343]:
```
plt.figure(figsize=(10,6))

sns.boxplot(x='floors', y='price', data=kingcounty_df)

plt.title('Price distribution by Floors')
plt.show()
```



The price of a house was not affected by the number of floors as shown in the boxplot above

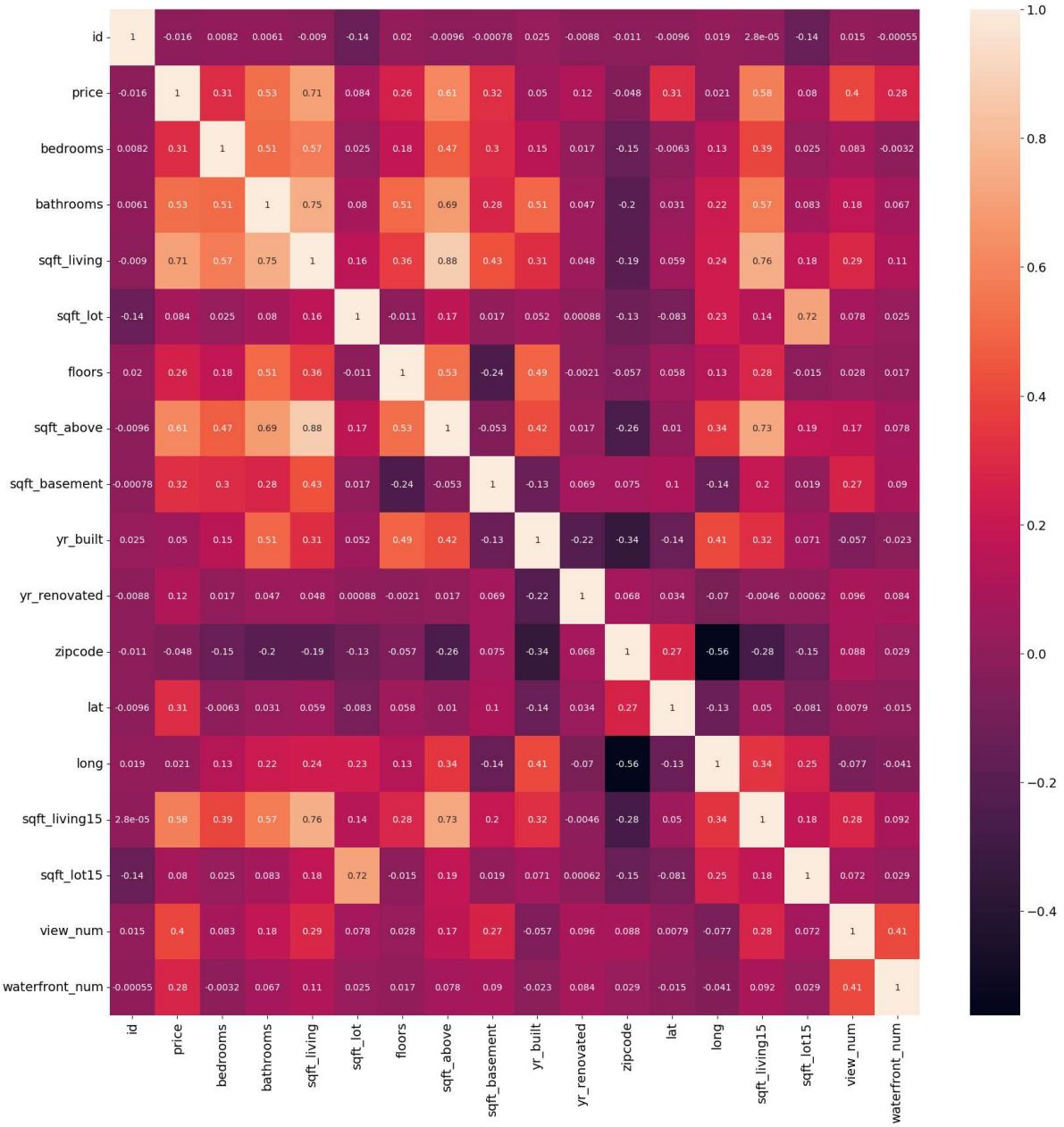
From the box plots, views and waterfront presence seem to have a linear relationship with price however the relationship between floors and price is not quite linear

Multicollinearity check

This section of the EDA focuses on investigating whether there is any perfect linear relationships amongst the variables. A correlation matrix will be used for the multicollinearity check

In [344]:
```
correlation_matrix = kingcounty_df.corr()

# Create a new figure with a specific size (in inches)
plt.figure(figsize=(20, 20))

sns.heatmap(correlation_matrix, annot=True)
plt.show()
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_6756\1316234239.py:1: FutureWarnin
g: The default value of numeric_only in DataFrame.corr is deprecated. In a fu
ture version, it will default to False. Select only valid columns or specify
the value of numeric_only to silence this warning.
  correlation_matrix = kingcounty_df.corr()
```



Based on the results of the correlation matrix, some variables which show a high level of correlation would affect the accuracy of the model.

Feature Engineering

Feature 1: Seasons vs House Prices

Here we find the relationship between seasons and house prices. We converted the 'date' column to a datetime format by extracting the month from the date. Additionally, we mapped the months to corresponding seasons.

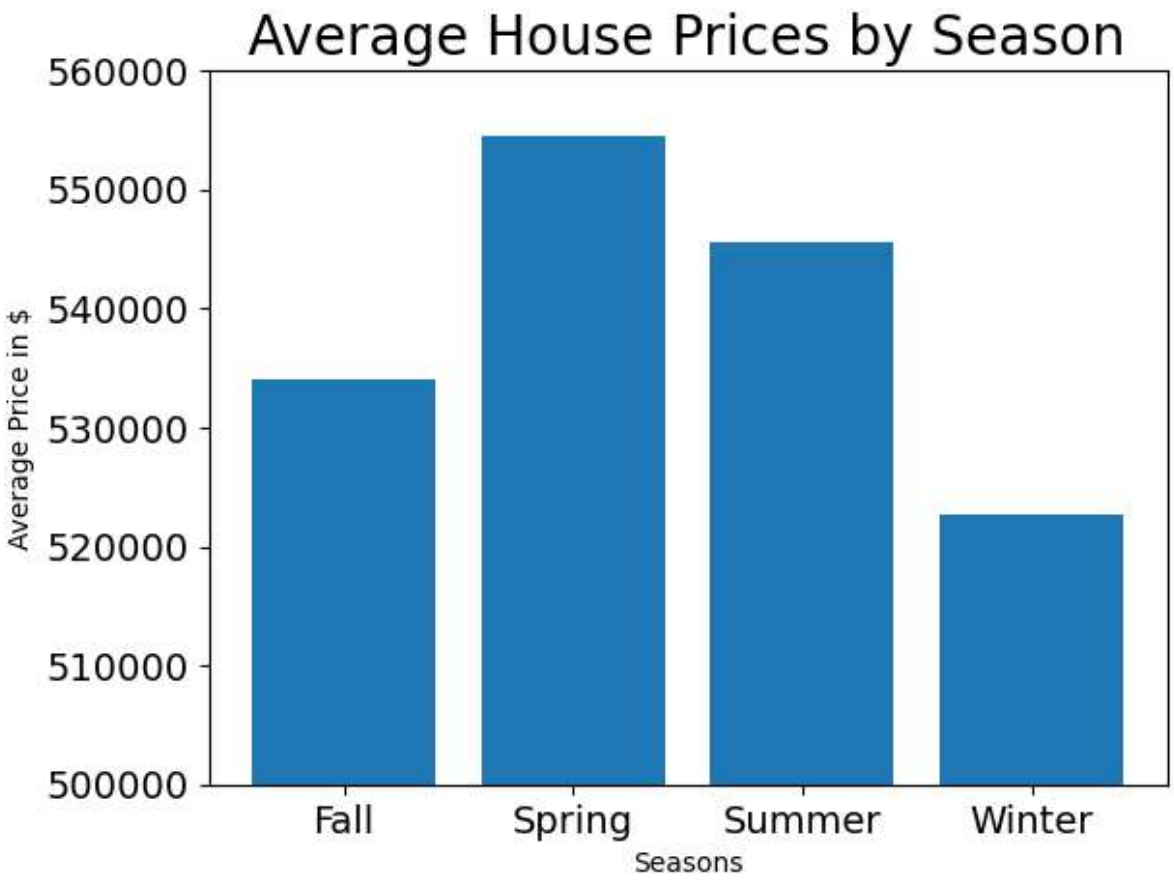
In [345]:
```
# # Convert 'date' column to datetime
kingcounty_df['date'] = pd.to_datetime(kingcounty_df['date'])
#Extract the month from the date column and create a new column
kingcounty_df['month'] = kingcounty_df['date'].dt.month
#Map months to correspondng seasons
season_mapping = {1: 'Winter', 2: 'Winter', 3: 'Spring', 4: 'Spring', 5: 'Spri
                  7: 'Summer', 8: 'Summer', 9: 'Fall', 10: 'Fall', 11: 'Fall',
kingcounty_df['season'] = kingcounty_df['month'].map(season_mapping)
kingcounty_df[["price","season"]]
```

Out[345]:

	price	season
1	538000.0	Winter
3	604000.0	Winter
4	510000.0	Winter
5	1230000.0	Spring
8	229500.0	Spring
...	...	...
21591	475000.0	Winter
21592	360000.0	Spring
21593	400000.0	Winter
21594	402101.0	Summer
21596	325000.0	Fall

15344 rows × 2 columns

In [346]:
```
#Group price by seasons
season_prices = kingcounty_df.groupby('season')['price'].mean()

# plot
seasons = ['Fall', 'Spring', 'Summer', 'Winter']

plt.bar(seasons, season_prices)

# Set x-label and y-label
plt.xlabel('Seasons')
plt.ylabel('Average Price in $')

# Set title
plt.title('Average House Prices by Season')

# Set y-axis limits
plt.ylim(500000, 560000)

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



Conclusion from the graph above; the best time to buy houses is during winter and best time for selling is spring

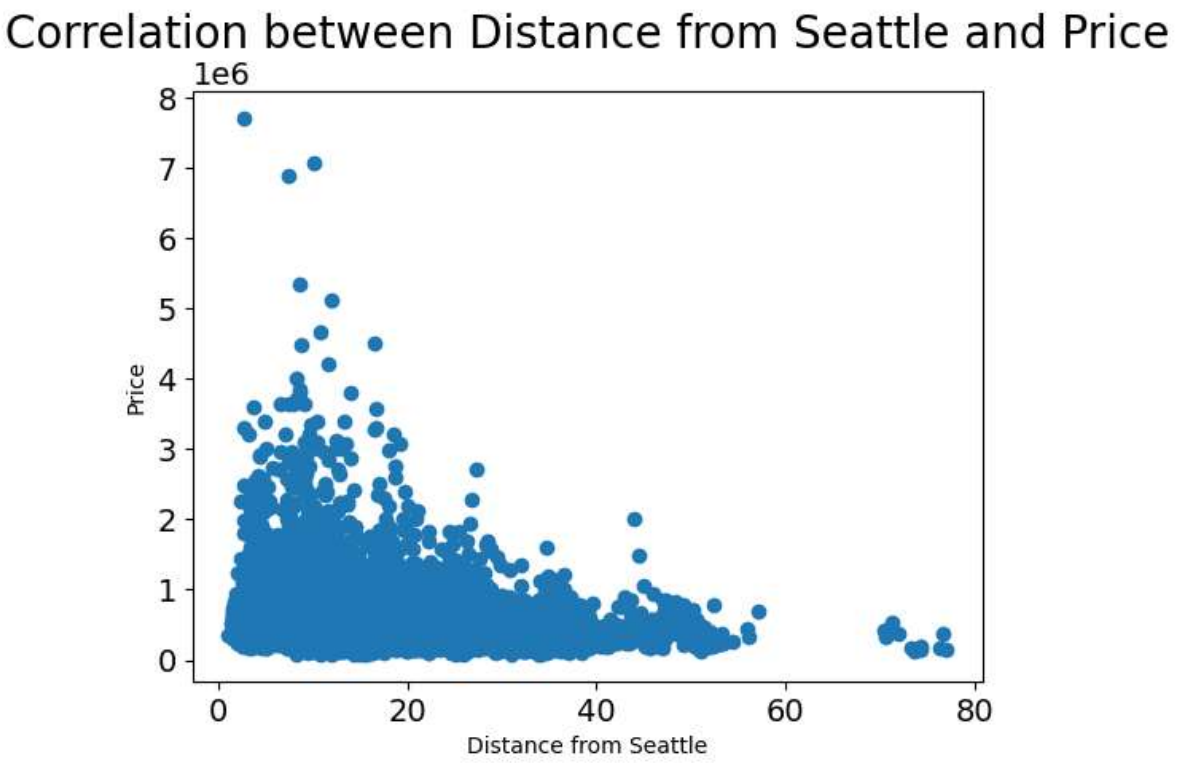
Feature 2: Distance from Seattle and its effect on house price in King

County

Seattle is Washington State's largest city, it's home to a large tech industry, with Microsoft and

In [347]:
```python
#Create a function that calculates distance
import math

def haversine_distance(lat1, lon1, lat2, lon2):
    # Convert latitude and longitude from degrees to radians
    lat1_rad = math.radians(lat1)
    lon1_rad = math.radians(lon1)
    lat2_rad = math.radians(lat2)
    lon2_rad = math.radians(lon2)

    # Radius of the Earth in kilometers
    radius = 6371

    # Haversine formula
    dlat = lat2_rad - lat1_rad
    dlon = lon2_rad - lon1_rad
    a = math.sin(dlat/2) * math.sin(dlat/2) + math.cos(lat1_rad) * math.cos(la
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    distance = radius * c

    return distance
```

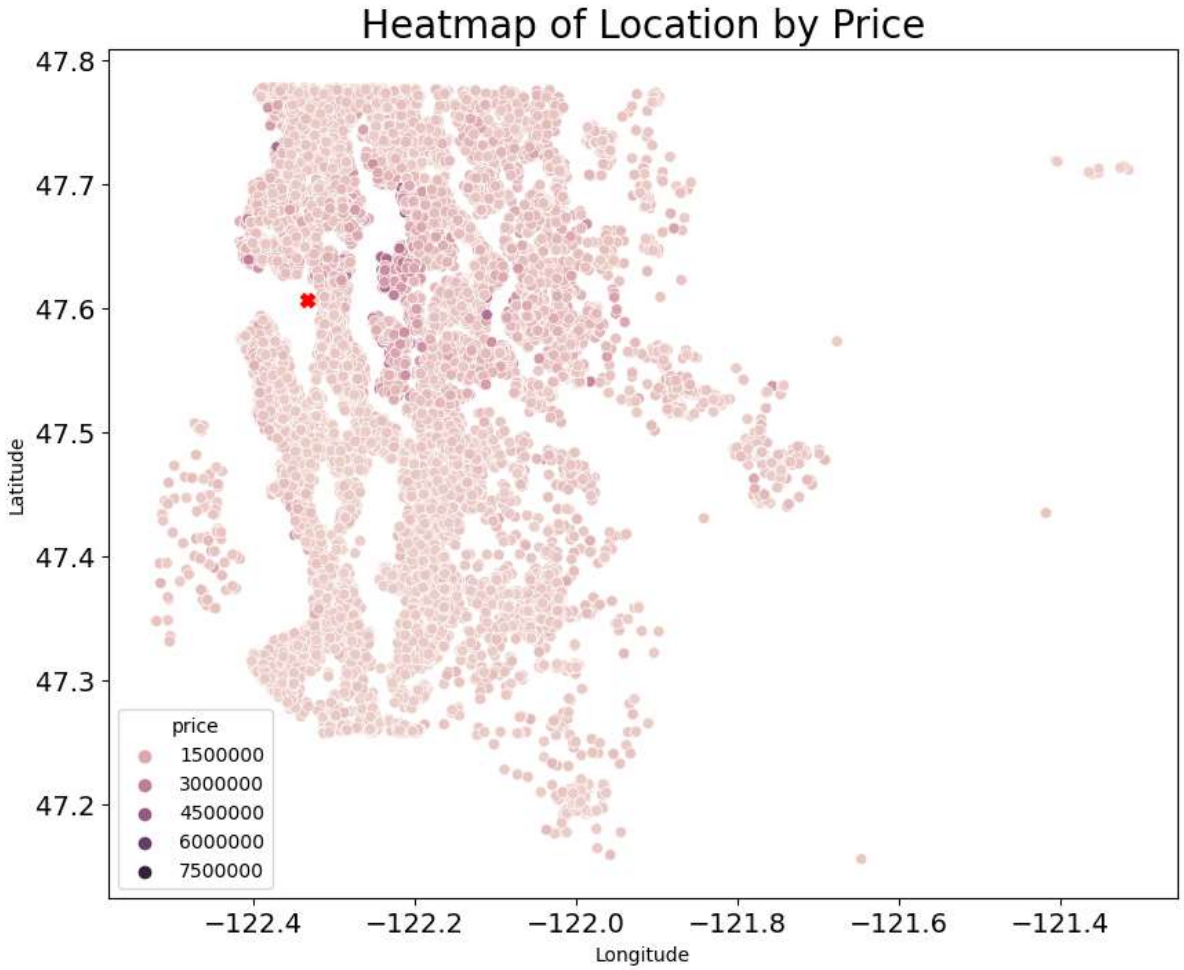
In [348]:
```
#Seattle is 47.6062° N, -122.3321° W </b

kingcounty_df['distance_from_seattle'] = kingcounty_df.apply(lambda row: haver

# Create a scatter plot
plt.scatter(kingcounty_df['distance_from_seattle'], kingcounty_df['price'])
plt.xlabel('Distance from Seattle')
plt.ylabel('Price')
plt.title('Correlation between Distance from Seattle and Price')


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



In [349]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a scatter plot with color-coded price
plt.figure(figsize=(10, 8))
sns.scatterplot(x='long', y='lat', hue='price', data=kingcounty_df, cmap='hsv'

# Set the title and labels
plt.title('Heatmap of Location by Price')
plt.xlabel('Longitude')
plt.ylabel('Latitude')

# Add a marker for Seattle on the scatter plot
seattle_lat = 47.6062
seattle_long = -122.3321
plt.scatter(seattle_long, seattle_lat, color='red', marker='X', s=50, label='S


# Display both the scatter plot and the map
plt.show()
```

```
C:\Users\user\anaconda3\lib\site-packages\seaborn\relational.py:573: UserWarn
ing: No data for colormapping provided via 'c'. Parameters 'cmap' will be ign
ored
  points = ax.scatter(x=x, y=y, **kws)
```



Seattle being the largest City in the County has other amenities in addition to better schools and also offers a better job market with large multinational employers. This makes it a top homeowners choice

Preprocessing

We droppped all the ordinal variables for simplicity of our regression model. We also dropped the ID variable which does not make meaningful contribution to the model

In [350]:
```
data_df = kingcounty_df.drop(["date", "id", "month", "view","yr_renovated",
                              "condition", "grade", "view_num",
                              "waterfront_num","zipcode","lat","long"], axis=1
data_df
```

Out[350]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	sqft_above	sqft
1	538000.0	3	2.25	2570	7242	2.0	NO	2170	
3	604000.0	4	3.00	1960	5000	1.0	NO	1050	
4	510000.0	3	2.00	1680	8080	1.0	NO	1680	
5	1230000.0	4	4.50	5420	101930	1.0	NO	3890	
8	229500.0	3	1.00	1780	7470	1.0	NO	1050	
...	...	...	...	...	...	...	...	...	
21591	475000.0	3	2.50	1310	1294	2.0	NO	1180	
21592	360000.0	3	2.50	1530	1131	3.0	NO	1530	
21593	400000.0	4	2.50	2310	5813	2.0	NO	2310	
21594	402101.0	2	0.75	1020	1350	2.0	NO	1020	
21596	325000.0	2	0.75	1020	1076	2.0	NO	1020	

15344 rows × 14 columns

LINEAR REGRESSION

Baseline Model

Our Baseline is the first model to be run and will include almost all the feature variables. We begin by defining our dependent variable(y) and our independent variable (X)

In [351]:
```
y = data_df["price"]

X_baseline = data_df.drop(data_df.columns[:1], axis=1)

# The Indepndent variable X_baseline is a dataframe that has eliminated Price
```

Converting categrical variables to dummies

To include our categorical variables in our model we code them to numeric data through pandas feature; get dummies

In [352]:
```
X_baseline = pd.get_dummies(X_baseline, columns = ["waterfront", "season"],dro
X_baseline.sample(3)
```

Out[352]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above	sqft_basement	yr_built	s
6127	3	2.50	2580	8154	1.0	2090	490.0	1956	
12799	3	1.00	1790	7055	1.0	1520	270.0	1937	
12772	3	1.75	1550	10050	1.0	1550	0.0	1957	

Running our Baseline Regression Model

In [353]:
```
baseline_model= sm.OLS(y, sm.add_constant(X_baseline))
baseline = baseline_model.fit()

print(baseline.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.68
3
Model:                            OLS   Adj. R-squared:                  0.68
2
Method:                 Least Squares   F-statistic:                     235
5.
Date:                Fri, 02 Jun 2023   Prob (F-statistic):               0.0
0
Time:                        23:42:23   Log-Likelihood:            -2.0984e+0
5
No. Observations:               15344   AIC:                         4.197e+0
5
Df Residuals:                   15329   BIC:                         4.198e+0
5
Df Model:                          14
Covariance Type:            nonrobust
==============================================================================
============
                             coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------------
const                   2.948e+06   1.54e+05     19.159      0.000    2.65e+06
3.25e+06
bedrooms               -4.818e+04   2293.687    -21.006      0.000   -5.27e+04
-4.37e+04
bathrooms                 5.5e+04   3991.248     13.781      0.000    4.72e+04
6.28e+04
sqft_living             150.6442      2.618     57.538      0.000     145.512
155.776
sqft_lot                  0.2843      0.058      4.874      0.000       0.170
0.399
floors                 -1.876e+04   4481.803     -4.185      0.000   -2.75e+04
-9973.379
sqft_above              116.2749      2.644     43.978      0.000     111.093
121.457
sqft_basement            34.3693      3.211     10.702      0.000      28.075
40.664
yr_built               -1438.7712     80.666    -17.836      0.000   -1596.886
-1280.656
sqft_living15             84.0569      3.950     21.281      0.000      76.315
91.799
sqft_lot15                -0.0030      0.089     -0.034      0.973      -0.177
0.172
distance_from_seattle -1.173e+04    192.136    -61.028      0.000   -1.21e+04
-1.13e+04
waterfront_YES         7.777e+05   1.99e+04     39.131      0.000    7.39e+05
8.17e+05
season_Spring          2.677e+04   4683.928      5.716      0.000    1.76e+04
3.6e+04
season_Summer          4403.0079   4706.941      0.935      0.350   -4823.154
1.36e+04
season_Winter          1981.3257   5422.230      0.365      0.715   -8646.888
1.26e+04
```

```
==============================================================================
=
Omnibus:                    11939.809   Durbin-Watson:                   1.98
8
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           912303.90
9
Skew:                           3.160   Prob(JB):                         0.0
0
Kurtosis:                      40.243   Cond. No.                     1.31e+1
7
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The smallest eigenvalue is 2.38e-21. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Checking for RMSE and MAE using SKlearn

In [354]:
```
y_predbaseline = baseline.predict(sm.add_constant(X_baseline))
# use fitted model to generate predictions for y
```

In [355]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
print(mean_absolute_error(y, y_predbaseline))
print(mean_squared_error(y, y_predbaseline, squared=False))
```

```
135360.2369466711
210364.2219768496
```

Reflections on the Baseline Model

- For our baseline model, our Rsquared = 68.3% This implies that 68.3 of the variance in price can be explained by the model.
- The baseline model is significant since the p-Value of the F_statistic is less than the level of significance assuming a level of significance of 0.05.
- All the feature variables are statistically significant except the Sqft-lot15, season_Summer, season_Winter which has a higher P_value than the level of significance(0.05).
- When all feature variables are zero, the price of a house = $ 2,948,000
- There is an inverese relationship between price of a house and the distance the house is from Seattle. Other features that have an inverse relationship with price in the baseline model include, bedrooms, floors and yrs_built.
- The
- Our baseline model has quite high error metrics with an MAE of 135,161 and RMSE of 210,234.
-
- There is need to refine this model to reduce error metrics and enhance rsquared.

Testing for Multicollinearity of feature variables

Check for Multicollinearity

In [356]:
```
df = data_df.corr().abs().stack().reset_index().sort_values(0, ascending=False

# zip the variable name columns (Which were only named level_0 and level_1 by d
df['pairs'] = list(zip(df.level_0, df.level_1))

# set index to pairs
df.set_index(['pairs'], inplace = True)

#d rop level columns
df.drop(columns=['level_1', 'level_0'], inplace = True)

# rename correlation column as cc rather than 0
df.columns = ['cc']

# drop duplicates. This could be dangerous if you have variables perfectly corr
# for the sake of exercise, kept it in.
df.drop_duplicates(inplace=True)
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_6756\645993706.py:1: FutureWarnin
g: The default value of numeric_only in DataFrame.corr is deprecated. In a fu
ture version, it will default to False. Select only valid columns or specify
the value of numeric_only to silence this warning.
  df = data_df.corr().abs().stack().reset_index().sort_values(0, ascending=Fa
lse)
```

In [357]:
```
df[(df.cc>.5) & (df.cc <1)] #  checking for features that have a high multicol
```

Out[357]:

pairs	cc
(sqft_above, sqft_living)	0.876555
(sqft_living15, sqft_living)	0.756479
(bathrooms, sqft_living)	0.752954
(sqft_living15, sqft_above)	0.732382
(sqft_lot15, sqft_lot)	0.716329
(price, sqft_living)	0.706263
(sqft_above, bathrooms)	0.685418
(price, sqft_above)	0.612209
(price, sqft_living15)	0.582762
(sqft_living, bedrooms)	0.573198
(bathrooms, sqft_living15)	0.569161
(sqft_above, floors)	0.528998
(price, bathrooms)	0.525637
(bedrooms, bathrooms)	0.512000
(bathrooms, floors)	0.507153
(bathrooms, yr_built)	0.505536

Setting our threshold for multicollinearity at 0.6. We droppped the following columns which we regarded to be having/causing high multicollineaity

- sqft_above
- sqft_living15
- bathrooms
- sqft_lot

For our next model, we also removed features with high p-values including:

- season_Summer
- season_Winter
- sqft_lot15

Model 2

We removed features with high multicollinearity (>0.60) and the features with high p-values

In [358]:
```
y = data_df["price"]
X_2 =  X_baseline.drop(["sqft_above","bathrooms", "sqft_living15",
                        "sqft_lot","sqft_lot15", "season_Summer", "season_Wint
X_2.sample(3)
```

Out[358]:

	bedrooms	sqft_living	floors	sqft_basement	yr_built	distance_from_seattle	waterfront_Y
9865	3	1570	1.0	490.0	1956	16.178617	
679	4	4370	2.0	0.0	2001	35.200360	
15398	3	2110	1.0	750.0	1973	33.271597	

In [359]:
```
model_2 = sm.OLS(y, sm.add_constant(X_2))
iteration_1 = model_2.fit()

print(iteration_1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.66
8
Model:                            OLS   Adj. R-squared:                  0.66
8
Method:                 Least Squares   F-statistic:                     386
4.
Date:                Fri, 02 Jun 2023   Prob (F-statistic):               0.0
0
Time:                        23:42:23   Log-Likelihood:            -2.1017e+0
5
No. Observations:               15344   AIC:                         4.204e+0
5
Df Residuals:                   15335   BIC:                         4.204e+0
5
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
============
                             coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------------
const                   2.095e+06   1.47e+05     14.254      0.000    1.81e+06
2.38e+06
bedrooms               -4.674e+04   2285.313    -20.454      0.000   -5.12e+04
-4.23e+04
sqft_living              346.9939      2.875    120.695      0.000     341.359
352.629
floors                 -1.459e+04   4375.661     -3.333      0.001   -2.32e+04
-6008.946
sqft_basement            -95.9028      5.202    -18.436      0.000    -106.099
-85.706
yr_built                -948.3192     76.830    -12.343      0.000   -1098.914
-797.724
distance_from_seattle -1.154e+04    189.284    -60.963      0.000   -1.19e+04
-1.12e+04
waterfront_YES           7.89e+05   2.03e+04     38.859      0.000    7.49e+05
8.29e+05
season_Spring          2.492e+04   3784.847      6.585      0.000    1.75e+04
3.23e+04
==============================================================================
=
Omnibus:                    10647.842   Durbin-Watson:                   1.98
4
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           580560.26
6
Skew:                           2.731   Prob(JB):                         0.0
0
Kurtosis:                      32.635   Cond. No.                       2.52e+0
5
==============================================================================
=

Notes:
```

```
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 2.52e+05. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

Checking for MAE and RMSE

In [360]:
```
y_prediteration_1 = iteration_1.predict(sm.add_constant(X_2))
print(f"MAE for model 2: {mean_absolute_error(y, y_prediteration_1)}")
print(f"RSME for model 2: {mean_squared_error(y, y_prediteration_1, squared=Fa
```

```
MAE for model 2: 139159.98195799987
RSME for model 2: 215013.21606879088
```

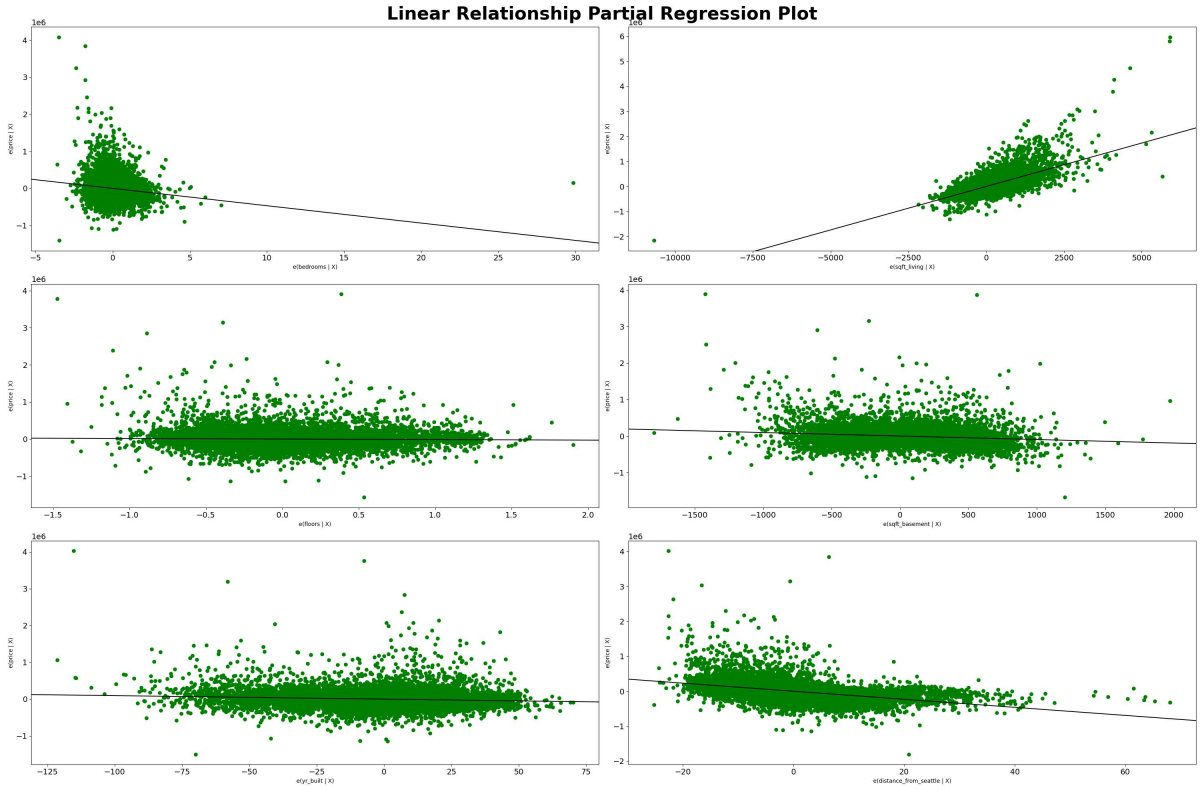
- The second model is statistically significant overall, and explains about 66.9% of the variance in house Price.
- All the Features are statistically significant with low p-values
- The coefficient for Sqft_living is about 345.8, which means that for each additional squarefoot in the living area, we expect the price to increase by about $345.8
- Our RMSE and MAE are still quite high. MAE =138,907 and RMSE = 214,695
- Hence we need to continue refining our model by satisfying other linear regression assumptions.

Checking for Linearity

Since we are using multiple regression, we shall check for linearirty using partial regression plots

In [361]:
```
fig = plt.figure(figsize=(30,20))
sm.graphics.plot_partregress_grid(iteration_1, exog_idx=["bedrooms", "sqft_liv
# Customizing plot appearance; note that the StatsModels code actually uses .p
# with marker 'o', so what looks like a scatter plot is a "line" internally, s
# we access it using .lines rather than .collections
for ax in fig.axes:
    ax.lines[0].set_color("green")
fig.suptitle("Linear Relationship Partial Regression Plot", fontsize=32, fontw
plt.tight_layout()
plt.show();
```

```
eval_env: 1
eval_env: 1
eval_env: 1
eval_env: 1
eval_env: 1
eval_env: 1
```

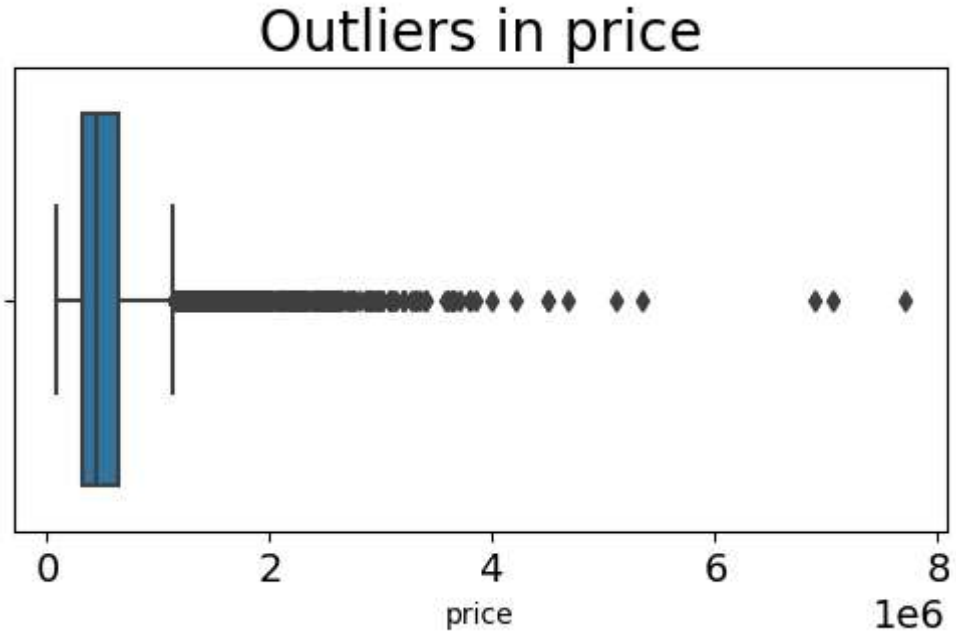


- From the Partial regression plots, its evident that there are no features that exhibit nonlinear relationships.
- The feature variables either show positive linear correlation with price or negative correlation or no correlation.
- Furthermore, the partial regression plot indicate presence of outliers.
- They also show that some of the columns such as yrs_renovated need to be modified since they have placeholders that make it appear as categorical data.

Dealing with Outliers

Outliers in the target variable

In [362]:
```
plt.figure(figsize=(6,3))

sns.boxplot(x=data_df['price'])

plt.title('Outliers in price')
plt.show()
```

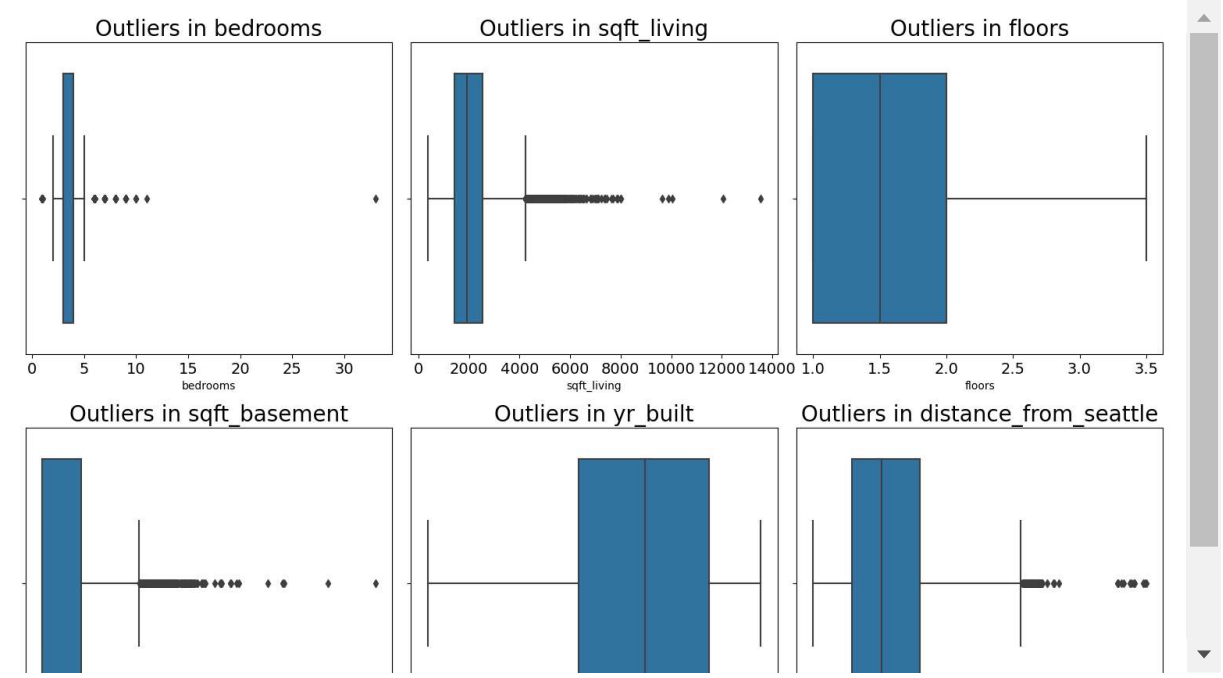


In [363]:
```
# Calculate the Z-score for the data series
# price_z_scores = (data_df['price'] - data_df['price'].mean()) / data_df['pri
# Identify outliers based on a threshold (e.g., 3 or -3)
# price_outliers = data_df['price'][abs(price_z_scores) > 3]
# data_df.drop(price_outliers.index, inplace=True)
```

Outliers in Feature Variables

In [364]:

```
independent_variables = ['bedrooms', 'sqft_living', 'floors', 'sqft_basement',
                         'distance_from_seattle']  # Replace with your actual 

num_plots = len(independent_variables)
num_cols = 3  # Number of columns in the subplot grid
num_rows = (num_plots - 1) // num_cols + 1

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 10))  # Set the figu

for i, variable in enumerate(independent_variables):
    row = i // num_cols
    col = i % num_cols
    ax = axes[row, col]
    sns.boxplot(x=variable, data=data_df, ax=ax)
    ax.set_title(f'Outliers in {variable}')  # Set the title for each subplot

# Remove any empty subplots
if num_plots < num_rows * num_cols:
    for i in range(num_plots, num_rows * num_cols):
        fig.delaxes(axes.flatten()[i])

plt.tight_layout()  # Adjust the spacing between subplots
plt.show()  # Show the plot
```



From the boxplots above we can conclude that the following columns have outliers:

- Bedrooms
- sqft-living
- sqft_basement
- Distance from seattle
- Yrs_renovated

Removing Outliers

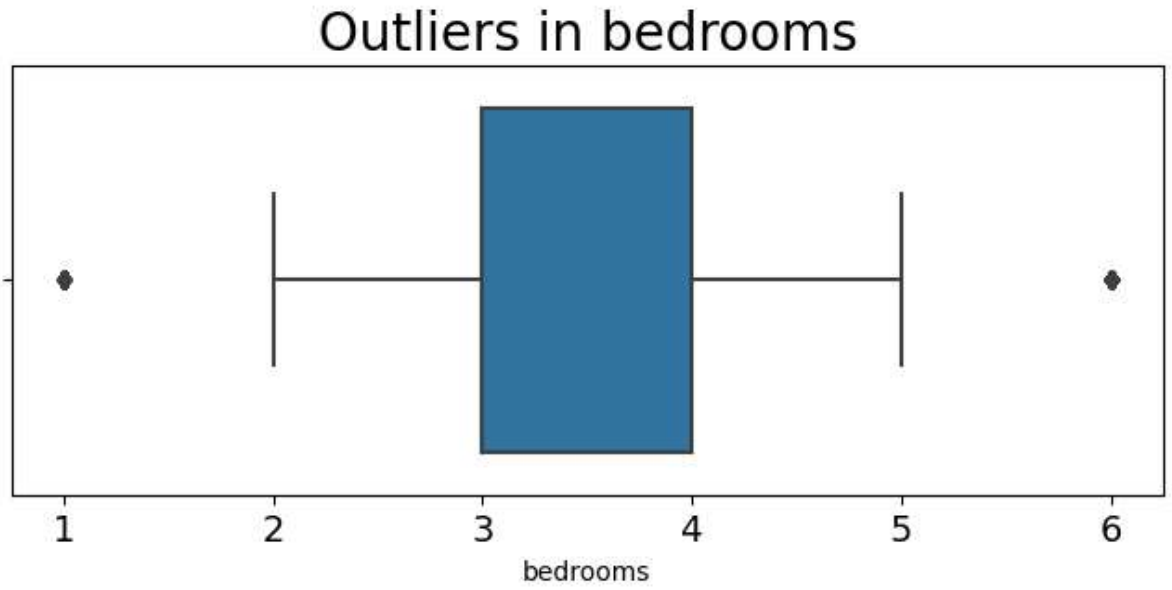
In [365]:
```
# Calculate the Z-score for the data series
bedroom_z_scores = (data_df['bedrooms'] - data_df['bedrooms'].mean()) / data_d
# Identify outliers based on a threshold (e.g., 3 or -3)
bedroom_outliers = data_df['bedrooms'][abs(bedroom_z_scores) > 3]
data_df.drop(bedroom_outliers.index, inplace=True)
```

In [366]:
```
# Calculate the Z-score for the data series
# sqftliving_z_scores = (data_df['sqft_living'] - data_df['sqft_living'].mean(
# Identify outliers based on a threshold (e.g., 3 or -3)
# sqftliving_outliers = data_df['sqft_living'][abs(sqftliving_z_scores) > 3]
# data_df.drop(sqftliving_outliers.index, inplace=True)
```

In [367]:
```
# Calculate the Z-score for the data series
# sqftbasement_z_scores = (data_df['sqft_basement'] - data_df['sqft_basement']
# Identify outliers based on a threshold (e.g., 3 or -3)
# sqftbasement_outliers = data_df['sqft_basement'][abs(sqftbasement_z_scores) >
# data_df.drop(sqftbasement_outliers.index, inplace=True)
```

In [368]:
```
# Calculate the Z-score for the data series
# distancefromseattle_z_scores = (data_df['distance_from_seattle'] - data_df['
# Identify outliers based on a threshold (e.g., 3 or -3)
# distancefromseattle_outliers = data_df['distance_from_seattle'][abs(distance
# data_df.drop(distancefromseattle_outliers.index, inplace=True)
```

In [369]:
```
plt.figure(figsize=(8,3))

sns.boxplot(x=data_df['bedrooms'])

plt.title('Outliers in bedrooms')
plt.show()
```



In [370]:
```
# Feature Engineering a new variable called No_ of_ years_since Renovation
# Also feature engineeered Age of  House from the columns yrs_built
data_df['Age_of_house'] = 2023 - data_df['yr_built']
data_df = data_df.drop(['yr_built'], axis=1) # dropped yrs_built and yrs_renov
data_df
```

Out[370]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	sqft_above	s
1	538000.0	3	2.25	2570	7242	2.0	NO	2170	
3	604000.0	4	3.00	1960	5000	1.0	NO	1050	
4	510000.0	3	2.00	1680	8080	1.0	NO	1680	
5	1230000.0	4	4.50	5420	101930	1.0	NO	3890	
8	229500.0	3	1.00	1780	7470	1.0	NO	1050	
...	...	...	...	...	...	...	...	...	
21591	475000.0	3	2.50	1310	1294	2.0	NO	1180	
21592	360000.0	3	2.50	1530	1131	3.0	NO	1530	
21593	400000.0	4	2.50	2310	5813	2.0	NO	2310	
21594	402101.0	2	0.75	1020	1350	2.0	NO	1020	
21596	325000.0	2	0.75	1020	1076	2.0	NO	1020	

Model 3

In [371]:
```
y = data_df["price"]

X_3 = data_df.drop(data_df.columns[:1], axis=1)
```

In [372]:
```
X_3 = pd.get_dummies(X_3, columns = ["waterfront", "season"],drop_first =True)
X_3 = X_3.drop(["sqft_above", "sqft_living15","bathrooms","sqft_lot","sqft_lot
                "season_Winter"], axis=1)
X_3.head()
```

Out[372]:

	bedrooms	sqft_living	floors	sqft_basement	distance_from_seattle	Age_of_house	waterfront
1	3	2570	2.0	400.0	12.802819	72	
3	4	1960	1.0	910.0	10.538233	58	
4	3	1680	1.0	0.0	21.553979	36	
5	4	5420	1.0	1530.0	25.131188	22	
8	3	1780	1.0	730.0	10.447676	63	

In [373]:
```
model_3 = sm.OLS(y, sm.add_constant(X_3))
iteration_2 = model_3.fit()

print(iteration_2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.67
0
Model:                            OLS   Adj. R-squared:                  0.67
0
Method:                 Least Squares   F-statistic:                     387
9.
Date:                Fri, 02 Jun 2023   Prob (F-statistic):               0.0
0
Time:                        23:42:28   Log-Likelihood:            -2.0943e+0
5
No. Observations:               15301   AIC:                         4.189e+0
5
Df Residuals:                   15292   BIC:                         4.189e+0
5
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
============
                             coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------------
const                   1.932e+05   1.22e+04     15.790      0.000    1.69e+05
2.17e+05
bedrooms               -5.432e+04   2475.192    -21.948      0.000   -5.92e+04
-4.95e+04
sqft_living              350.9909      2.904    120.868      0.000     345.299
356.683
floors                 -1.517e+04   4345.787     -3.492      0.000   -2.37e+04
-6655.717
sqft_basement            -94.0112      5.175    -18.166      0.000    -104.155
-83.867
distance_from_seattle -1.147e+04    187.730    -61.116      0.000   -1.18e+04
-1.11e+04
Age_of_house             927.3548     76.432     12.133      0.000     777.540
1077.170
waterfront_YES         7.838e+05   2.01e+04     38.965      0.000    7.44e+05
8.23e+05
season_Spring          2.535e+04   3752.654      6.756      0.000     1.8e+04
3.27e+04
==============================================================================
=
Omnibus:                    10667.287   Durbin-Watson:                   1.98
7
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           589616.16
2
Skew:                           2.747   Prob(JB):                         0.0
0
Kurtosis:                      32.911   Cond. No.                       2.69e+0
4
==============================================================================
=

Notes:
```

```
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 2.69e+04. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

Checking for RMSE and MAE

In [374]:
```
y_prediteration_2 = iteration_1.predict(sm.add_constant(X_3))
print(f"MAE for model 3: {mean_absolute_error(y, y_prediteration_2)}")
print(f"RSME for model 3: {mean_squared_error(y, y_prediteration_2, squared=Fa
```

```
MAE for model 3: 1471342.5170185685
RSME for model 3: 1542427.5202168645
```

Interpretation of model 3

- There is a remarkable improvement in our R-squared now at 67.1 from 66.9.
- The RMSE and MSE have also reduced as required
- We continue to refine our model by ensuring other linear assumptions are satisfied

Checking for Homoscedasticity

Breusch pagan test

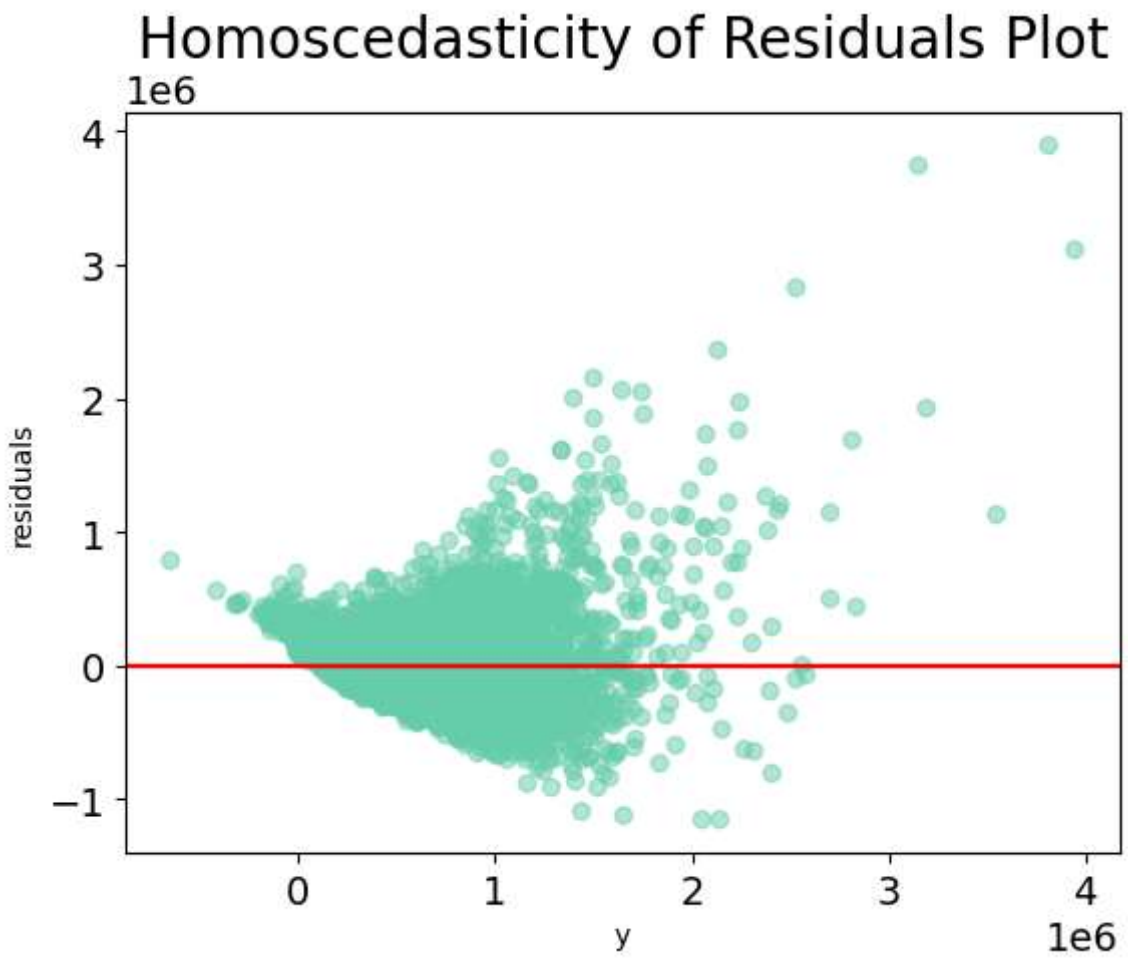
In [375]:
```
from statsmodels.stats.diagnostic import het_breuschpagan
```

In [376]:
```
het_breuschpagan(sm.OLS(y, sm.add_constant(X_3)).fit().resid, sm.add_constant(
```

Out[376]:
```
(2120.4557607575684, 0.0, 307.5177407788934, 0.0)
```

Null Hypothesis = The residuals are homoscedastic Based on the above output(p-values) we reject the null hypothesis and Conclude that our data is not homoscedastic.

In [377]:
```
# Create a LinearRegression model
model = LinearRegression()

# Train the model using your training data
model.fit(X_3, y)

# Make predictions using the trained model
fit_line_generated = model.predict(X_3)

# Calculate residuals
resids_generated = y - fit_line_generated

# Create scatter plot of data
fig, ax = plt.subplots()
ax.scatter(fit_line_generated, resids_generated, color="mediumaquamarine", alp

# Plot horizontal lines
scale = 1.0
ax.axhline(y=0, color="black")
ax.axhline(y=scale*3, color="red")  # Again, we know the position of these
ax.axhline(y=-scale*3, color="red") # lines because we generated the data

# Customize labels
ax.set_xlabel("y")
ax.set_ylabel("residuals")
ax.set_title("Homoscedasticity of Residuals Plot");
```



Checking For Normality of Residuals

To test for normality we used tests and visualization.

Jarque_bera test

To test for normality

In [378]:
```
from statsmodels.stats.stattools import jarque_bera
jarque_bera(iteration_2.resid)
```

Out[378]:
```
(589616.1622366522, 0.0, 2.74687914639752, 32.91063809407521)
```

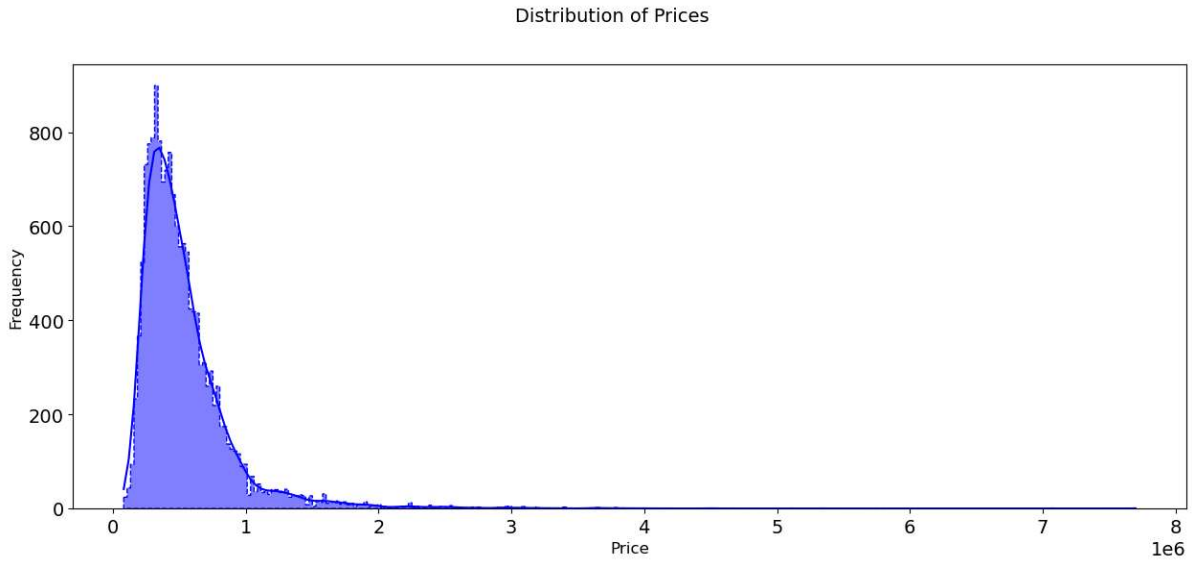
K-test

To test for normality

Dealing with the Depenedent Variable/Target(y)

Checking for Normality of our Target variable(y)

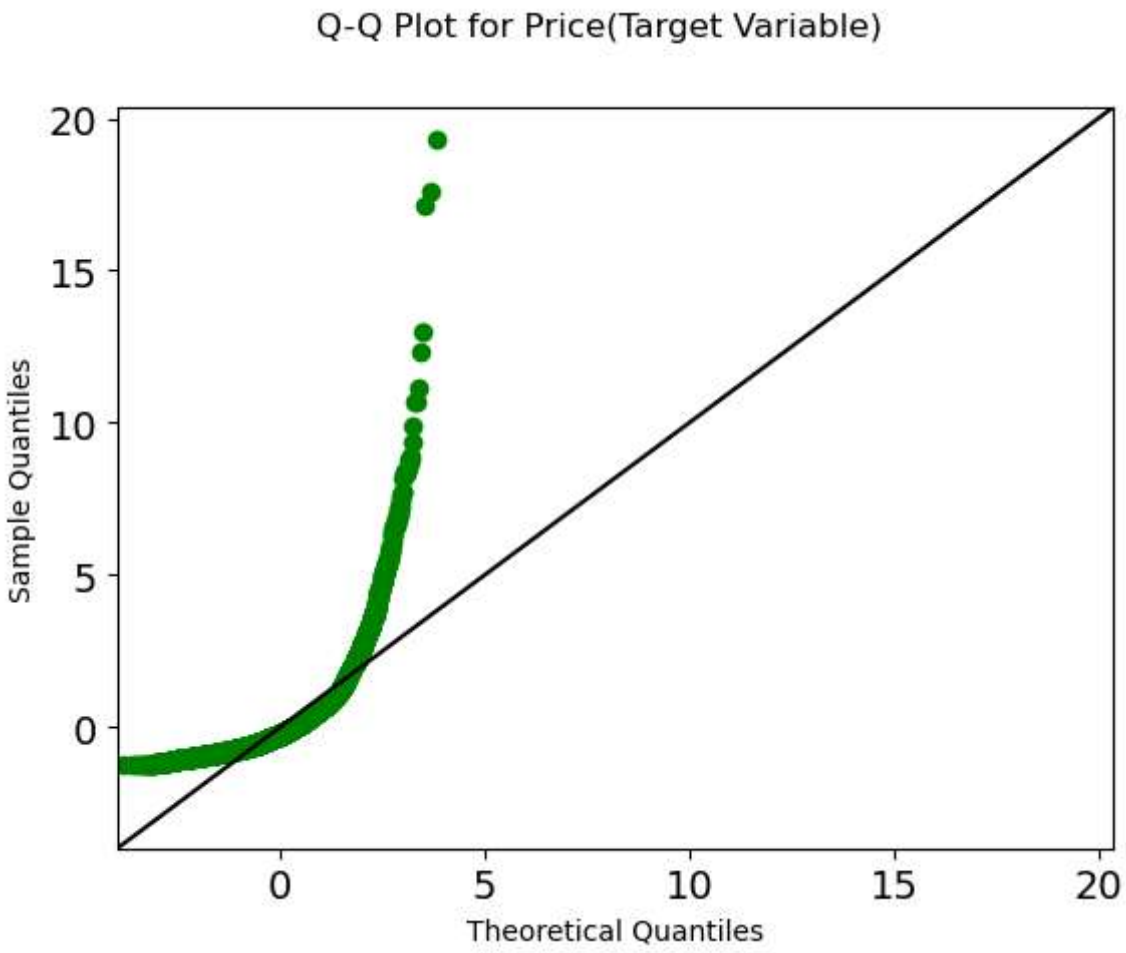
In [379]:
```
fig, ax = plt.subplots(figsize=(15, 6))
sns.histplot(data_df['price'], element="step", kde=True, color="blue", ax=ax,

ax.set_xlabel("Price", fontsize=12)
ax.set_ylabel("Frequency", fontsize=12)
fig.suptitle("Distribution of Prices", fontsize=14)

plt.show()
```



The target variable shows positively skewed distribution, also known as a right-skewed distribution. The tail on the right side of the distribution is longer and more spread out than the tail on the left side. As a result, the majority of the data points or values have clustered toward the left side of the distribution, while a few extreme values have extended the right tail.

Using QQplot

In [380]:
```
import statsmodels.api as sm
import matplotlib.pyplot as plt
from scipy import stats

fig, ax = plt.subplots()
sm.graphics.qqplot(data_df['price'], dist=stats.norm, line='45', fit=True, ax=
# Customize plot appearance
scatter = ax.lines[0]
line = ax.lines[1]
scatter.set_markeredgecolor("green")
scatter.set_markerfacecolor("green")
line.set_color("black")
fig.suptitle("Q-Q Plot for Price(Target Variable)");
```



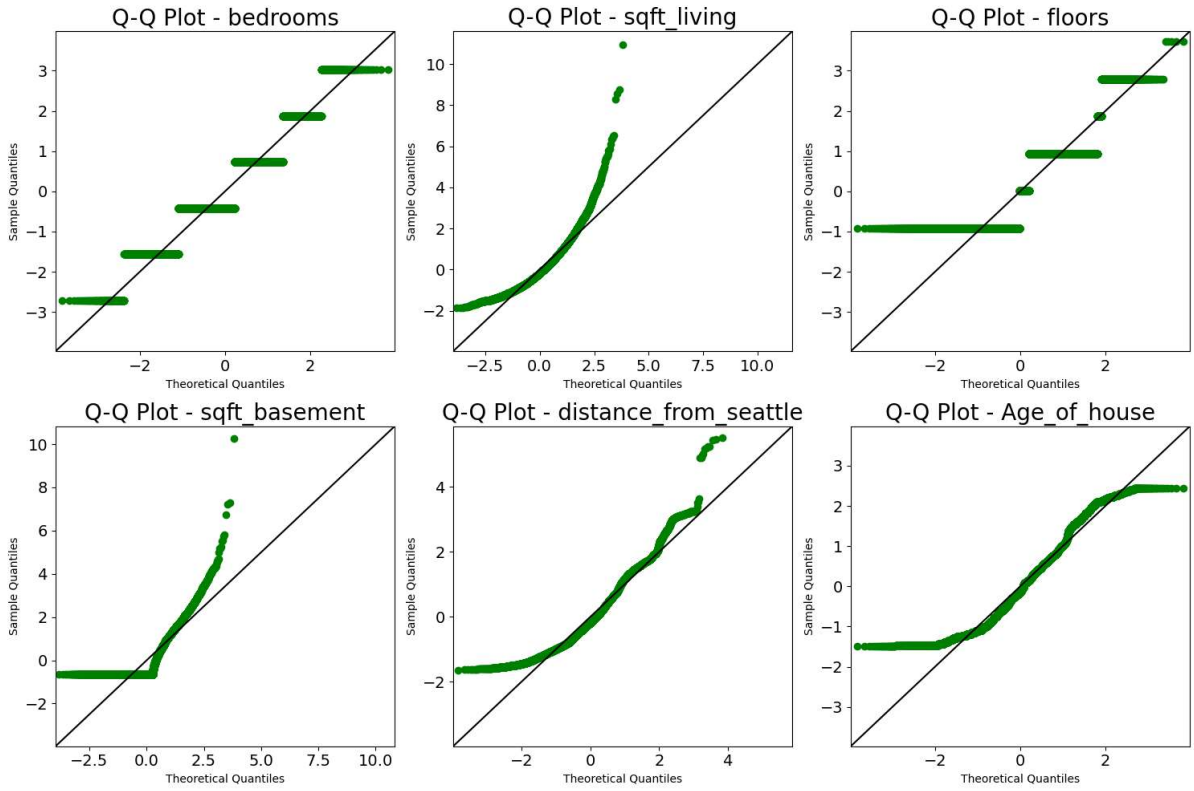
In [381]:
```
stats.kstest(data_df['price'], "norm")
```

Out[381]:
```
KstestResult(statistic=1.0, pvalue=0.0, statistic_location=82000.0, statistic
_sign=-1)
```

The result for ktest and the Q Q plot suggests that the distribution of the target variable(y) does not follow a normal distribution. The p-value of 0.0 indicates that the probability of obtaining such extreme test statistic value (or even more extreme) under the assumption of normality is extremely low

Checking for Normality of Feature Variables(X)

In [382]:
```
import statsmodels.api as sm
import matplotlib.pyplot as plt
from scipy import stats

variables = ['bedrooms', 'sqft_living', 'floors', 'sqft_basement', 'distance_f
             'Age_of_house']

num_rows = 2
num_cols = 3
fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(15, 10))

for i, variable in enumerate(variables):
    row = i // num_cols
    col = i % num_cols
    ax = axes[row, col]
    sm.graphics.qqplot(data_df[variable], dist=stats.norm, line='45', fit=True

    scatter = ax.lines[0]
    line = ax.lines[1]
    scatter.set_markeredgecolor("green")
    scatter.set_markerfacecolor("green")
    line.set_color("black")

    ax.set_xlabel("Theoretical Quantiles")
    ax.set_ylabel("Sample Quantiles")
    ax.set_title(f"Q-Q Plot - {variable}")

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



In [383]:
```
from scipy.stats import kstest

variables = ['bedrooms', 'sqft_living', 'floors', 'sqft_basement', 'distance_f

for variable in variables:
    data = data_df[variable]  # Assuming 'data_df' is the DataFrame containing

    statistic, p_value = kstest(data, 'norm')

    alpha = 0.05  # significance level

    if p_value > alpha:
        print(f"The distribution of '{variable}' appears to be normally distril
    else:
        print(f"The distribution of '{variable}' does not appear to be normall
```

```
The distribution of 'bedrooms' does not appear to be normally distributed (re
ject H0)
The distribution of 'sqft_living' does not appear to be normally distributed
(reject H0)
The distribution of 'floors' does not appear to be normally distributed (reje
ct H0)
The distribution of 'sqft_basement' does not appear to be normally distribute
d (reject H0)
The distribution of 'distance_from_seattle' does not appear to be normally di
stributed (reject H0)
The distribution of 'Age_of_house' does not appear to be normally distributed
(reject H0)
```

The Q-Q plots the ktest suggests both the target variable and the feature variables do not follow a normal distribution thus the need for log transformation

Log Transformation

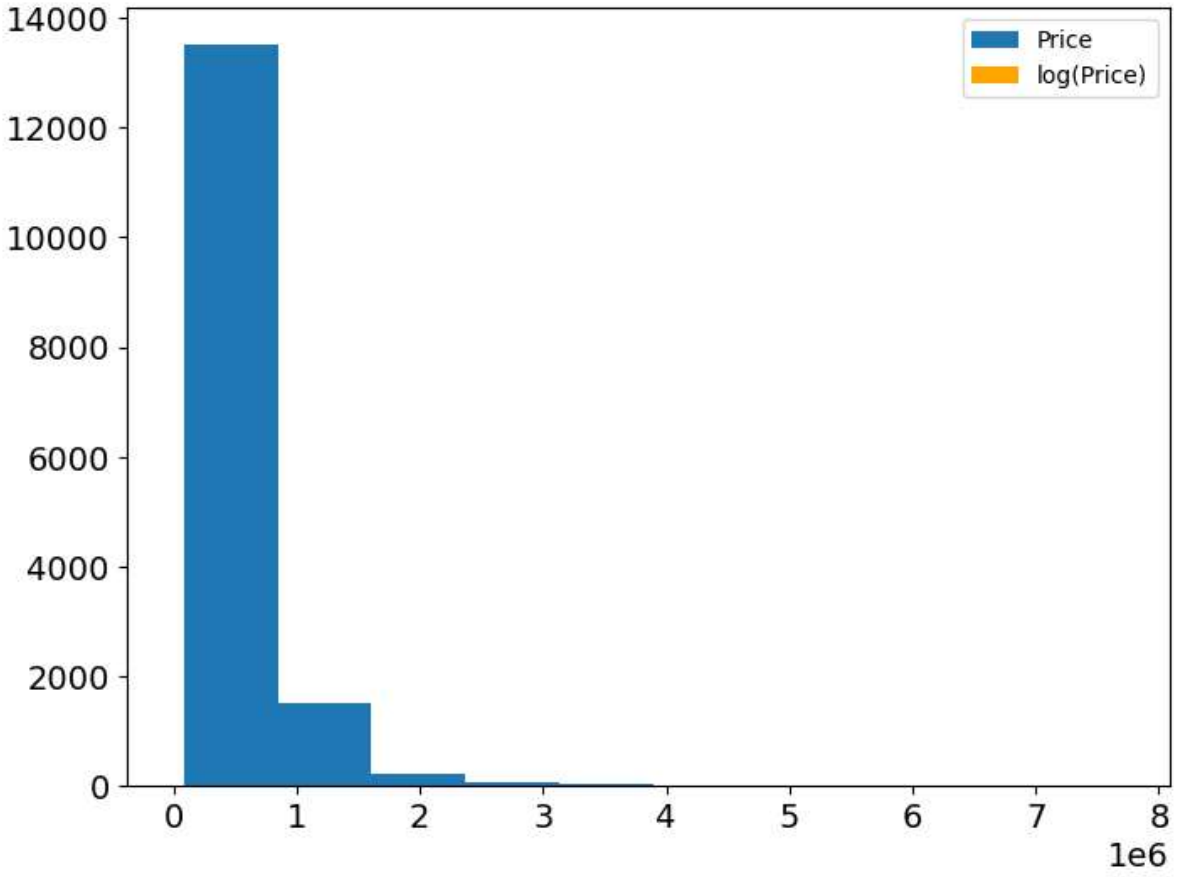
Log Transformation of the Target Variable (Y)

In [384]:
```
y = data_df['price']
```

In [385]:
```
y = data_df['price']
y_log = np.log(y)
y_log.name = "log(price)"
y_log
```

Out[385]:
```
1        13.195614
3        13.311329
4        13.142166
5        14.022525
8        12.343658
            ...
21591    13.071070
21592    12.793859
21593    12.899220
21594    12.904459
21596    12.691580
Name: log(price), Length: 15301, dtype: float64
```

In [386]:
```
fig, ax = plt.subplots(figsize=(8,6))
ax.hist(y, label="Price")
ax.hist(y_log, color="orange", label="log(Price)")
ax.legend();
```



In [387]:
```
X_4 = data_df.drop(data_df.columns[:1], axis=1)
```

In [388]:
```
X_4 = pd.get_dummies(X_4, columns = ["waterfront", "season"],drop_first =True)
X_4 = X_4.drop(["sqft_above", "sqft_living15","bathrooms","sqft_lot","sqft_lot
                "sqft_basement",], axis=1)
X_4.head()
```

Out[388]:

	bedrooms	sqft_living	floors	distance_from_seattle	Age_of_house	waterfront_YES	season_S
1	3	2570	2.0	12.802819	72	0	
3	4	1960	1.0	10.538233	58	0	
4	3	1680	1.0	21.553979	36	0	
5	4	5420	1.0	25.131188	22	0	
8	3	1780	1.0	10.447676	63	0	

Model 4

In [389]:
```
model_4 = sm.OLS(y_log, sm.add_constant(X_4))
iteration_3 = model_4.fit()

print(iteration_3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             log(price)   R-squared:                       0.
669
Model:                            OLS   Adj. R-squared:                  0.
669
Method:                 Least Squares   F-statistic:                     44
17.
Date:                Fri, 02 Jun 2023   Prob (F-statistic):
0.00
Time:                        23:42:33   Log-Likelihood:                -340
0.3
No. Observations:               15301   AIC:                             68
17.
Df Residuals:                   15293   BIC:                             68
78.
Df Model:                           7
Covariance Type:            nonrobust
```

In [390]:
```
y_prediteration_3 = iteration_3.predict(sm.add_constant(X_4))
print(f"MAE for model 4: {mean_absolute_error(y, y_prediteration_3)}")
print(f"RSME for model 4: {mean_squared_error(y, y_prediteration_3, squared=Fa
```

```
MAE for model 4: 540176.40030696
RSME for model 4: 655009.7835539838
```

Log Transformation of Feature Variables(X)

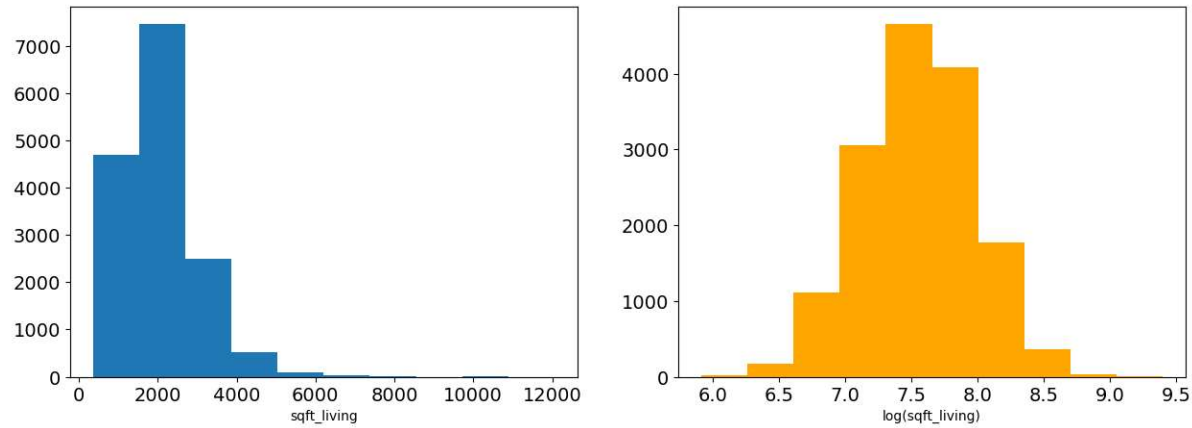
In [391]:
```
# Log transforming the feature variables sqft living
X_sqftliving_log = X_4.copy()

X_sqftliving_log["log(sqft_living)"] = np.log(X_sqftliving_log["sqft_living"])

# Visually inspect raw vs. transformed values
X_sqftliving_log[["sqft_living", "log(sqft_living)"]]
```

Out[391]:

	sqft_living	log(sqft_living)
1	2570	7.851661
3	1960	7.580700
4	1680	7.426549
5	5420	8.597851
8	1780	7.484369
...	...	...
21591	1310	7.177782
21592	1530	7.333023
21593	2310	7.745003
21594	1020	6.927558
21596	1020	6.927558

15301 rows × 2 columns

In [392]:
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(15,5))
ax1.hist(X_sqftliving_log["sqft_living"])
ax1.set_xlabel("sqft_living")
ax2.hist(X_sqftliving_log["log(sqft_living)"], color="orange")
ax2.set_xlabel("log(sqft_living)");
```



Model 5

In [393]:
```
model_5 = sm.OLS(y_log, sm.add_constant(X_sqftliving_log))
iteration_4 = model_5.fit()

print(iteration_4.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             log(price)   R-squared:                       0.67
9
Model:                            OLS   Adj. R-squared:                  0.67
9
Method:                 Least Squares   F-statistic:                     403
7.
Date:                Fri, 02 Jun 2023   Prob (F-statistic):               0.0
0
Time:                        23:42:34   Log-Likelihood:                -3175.
3
No. Observations:               15301   AIC:                             636
9.
Df Residuals:                   15292   BIC:                             643
7.
Df Model:                           8
Covariance Type:            nonrobust
================================================================================
============
                               coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
------------
const                        9.5919      0.137     69.760      0.000       9.322
9.861
bedrooms                    -0.0696      0.004    -19.089      0.000      -0.077
-0.062
sqft_living                  0.0002   9.15e-06     27.131      0.000       0.000
0.000
floors                       0.0661      0.005     12.151      0.000       0.055
0.077
distance_from_seattle       -0.0195      0.000    -77.228      0.000      -0.020
-0.019
Age_of_house                 0.0008      0.000      7.093      0.000       0.001
0.001
waterfront_YES               0.5541      0.028     19.674      0.000       0.499
0.609
season_Spring                0.0454      0.005      8.647      0.000       0.035
0.056
log(sqft_living)             0.4474      0.021     21.366      0.000       0.406
0.488
==============================================================================
=
Omnibus:                      166.089   Durbin-Watson:                   1.99
7
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              208.46
2
Skew:                          -0.177   Prob(JB):                     5.41e-4
6
Kurtosis:                       3.450   Cond. No.                     1.31e+0
5
==============================================================================
=

Notes:
```

```
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 1.31e+05. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

In [394]:
```
y_prediteration_4 = iteration_4.predict(sm.add_constant(X_sqftliving_log))
print(f"MAE for model 4: {mean_absolute_error(y, y_prediteration_4)}")
print(f"RSME for model 4: {mean_squared_error(y, y_prediteration_4, squared=Fa
```

```
MAE for model 4: 540176.4003069601
RSME for model 4: 655009.7874754862
```

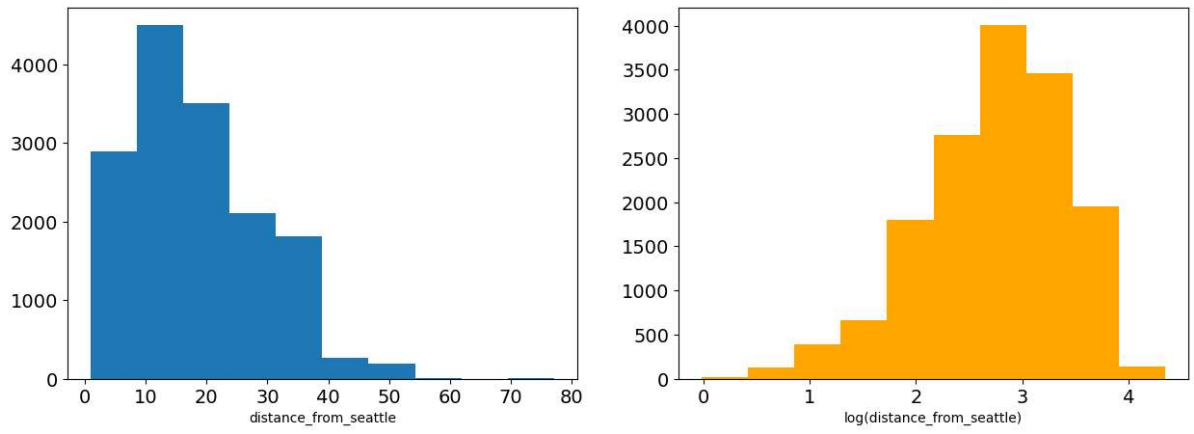
Log transformation of distance from seattle

In [395]:
```
X_Distance_log = X_sqftliving_log.copy()

X_Distance_log["log(distance_from_seattle)"] = np.log(X_Distance_log["distance_

# Visually inspect raw vs. transformed values
X_Distance_log[["distance_from_seattle", "log(distance_from_seattle)"]].sample
```

Out[395]:

	distance_from_seattle	log(distance_from_seattle)
16191	14.908152	2.701908
13626	34.376077	3.537361
8040	4.829348	1.574711

In [396]:
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(15,5))
ax1.hist(X_Distance_log["distance_from_seattle"])
ax1.set_xlabel("distance_from_seattle")
ax2.hist(X_Distance_log["log(distance_from_seattle)"], color="orange")
ax2.set_xlabel("log(distance_from_seattle)");
```

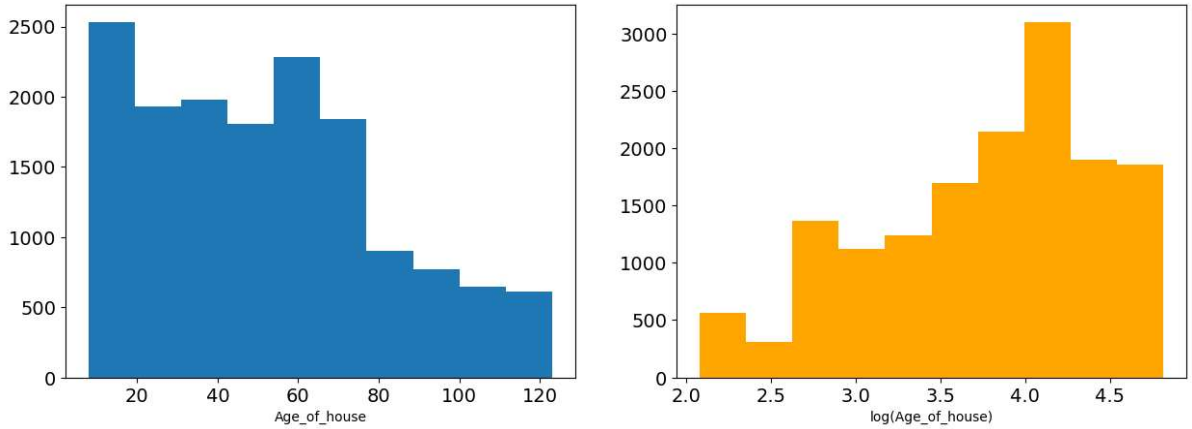


In [397]:
```
X_HouseAge_log = X_sqftliving_log.copy()

X_HouseAge_log["log(Age_of_house)"] = np.log(X_HouseAge_log["Age_of_house"])

# Visually inspect raw vs. transformed values
X_HouseAge_log[["Age_of_house", "log(Age_of_house)"]].sample(3)
```

Out[397]:

	Age_of_house	log(Age_of_house)
13	46	3.828641
4770	25	3.218876
12314	22	3.091042

In [398]:
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(15,5))
ax1.hist(X_HouseAge_log["Age_of_house"])
ax1.set_xlabel("Age_of_house")
ax2.hist(X_HouseAge_log["log(Age_of_house)"], color="orange")
ax2.set_xlabel("log(Age_of_house)");
```



In [399]:
```
X_HouseAge_log.drop(['sqft_living', 'Age_of_house'], axis = 1)
```

Out[399]:

	bedrooms	floors	distance_from_seattle	waterfront_YES	season_Spring	log(sqft_living)
1	3	2.0	12.802819	0	0	7.851661
3	4	1.0	10.538233	0	0	7.580700
4	3	1.0	21.553979	0	0	7.426549
5	4	1.0	25.131188	0	1	8.597851
8	3	1.0	10.447676	0	1	7.484369
...	...	...	...	...	...	...
21591	3	2.0	6.601720	0	0	7.177782
21592	3	3.0	10.404472	0	1	7.333023
21593	4	2.0	10.853552	0	0	7.745003
21594	2	2.0	2.807293	0	0	6.927558
21596	2	2.0	2.823044	0	0	6.927558

15301 rows × 7 columns

Model 6

In [400]:
```
model_6 = sm.OLS(y_log, sm.add_constant(X_HouseAge_log))
iteration_5 = model_6.fit()

print(iteration_5.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             log(price)   R-squared:                       0.67
9
Model:                            OLS   Adj. R-squared:                  0.67
9
Method:                 Least Squares   F-statistic:                     360
1.
Date:                Fri, 02 Jun 2023   Prob (F-statistic):               0.0
0
Time:                        23:42:35   Log-Likelihood:                -3157.
7
No. Observations:               15301   AIC:                             633
5.
Df Residuals:                   15291   BIC:                             641
2.
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
============
                             coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------------
const                      9.7495      0.140     69.692      0.000       9.475
10.024
bedrooms                  -0.0689      0.004    -18.906      0.000      -0.076
-0.062
sqft_living                0.0002   9.16e-06     26.697      0.000       0.000
0.000
floors                     0.0510      0.006      8.493      0.000       0.039
0.063
distance_from_seattle     -0.0193      0.000    -75.776      0.000      -0.020
-0.019
Age_of_house               0.0023      0.000      8.181      0.000       0.002
0.003
waterfront_YES             0.5604      0.028     19.903      0.000       0.505
0.616
season_Spring              0.0455      0.005      8.684      0.000       0.035
0.056
log(sqft_living)           0.4569      0.021     21.781      0.000       0.416
0.498
log(Age_of_house)         -0.0761      0.013     -5.927      0.000      -0.101
-0.051
==============================================================================
=
Omnibus:                      168.115   Durbin-Watson:                   1.99
3
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              211.37
4
Skew:                          -0.178   Prob(JB):                     1.26e-4
6
Kurtosis:                       3.453   Cond. No.                     1.34e+0
5
==============================================================================
=
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 1.34e+05. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

In [401]:
```
y_prediteration_5 = iteration_5.predict(sm.add_constant(X_Distance_log))
print(f"MAE for model 5: {mean_absolute_error(y, y_prediteration_5)}")
print(f"RSME for model 5: {mean_squared_error(y, y_prediteration_5, squared=Fa
```

```
MAE for model 5: 540176.3223288098
RSME for model 5: 655009.7178297067
```

Interpretation of the model

It has a relatively high Rsquared of 67.9% value of 0.679 indicates that approximately 67.9% of the variance in the logarithm of housing prices can be explained by the independent variables included in the model.

- Adj. R-squared: The adjusted R-squared value of 0.679
- The F-statistic of 3601 indicates a significant relationship between the independent variables and the dependent variable.The associated probability (Prob (F-statistic)) is very close to 0, suggesting strong evidence against the null hypothesis that all the coefficients are zero.

Coefficients

- Const: when all other independent variables are zero. In this case, it is approximately $ 9.7495.
- bedrooms: A one-unit increase in the number of bedrooms is associated with a 6.08% decrease in the price of the house
- Distance_from_Seattle: An increase in one kilometer from seatttle is associated with 1.94% decrease in price of a house.
- Floors: An increase in one floor is associated with a 5.02% increase in the price of a house.
- log(sqft_living): A 1% increase in sqft_liviing is a associated with a 46.48% increase in the price of a house, holding other variables constant
- log(Age_of_house): A 1% increase in Age_of_house is associated with a decrease in housing price by 7.96%, holding other variables constant.
- waterfront_YES: If the house has a waterfront (waterfront_YES), the housing price is expected to increase by 56.04%, holding other variables constant.
- season_Spring: If the season is spring (season_Spring), the housing price is expected to increase by 4.55% , holding other variables constant.

Observation

Model 6 is our peferred model beacuse: From the evaluation metrics, we can see that the models have similar performance in terms of MAE and RMSE. However, Model 6, which includes log transformations of the distance from Seattle and the age of the house, has the highest R-squared value and the lowest AIC and BIC values among the models. Therefore, Model 6 might be the best model among the ones considered.

Conclusion

- The further you are from seattle the cheaper the houses
- The more the bedrooms the more expensive the house
- The more space/land a house occupies,the more expensive it is
- Houses are most expenive in spring
- Spring and summer are the best times to sell houses while fall and winter are the best times to buy
- Square Footage of Living Space: The square footage of living space has a positive impact on house prices. As the size of the living space increases, the estimated price of the house also increases. This indicates that larger houses are generally priced higher.
- As the age of the house increases, the estimated price also increases. This could be due to factors such as historical significance or architectural value associated with older houses.

In [ ]: