Northwestern County: Real Estate Market Analysis

Project Overview:

Every Door Real Estate, our stakeholder, is facing a business issue related to their clients' home-buying process. The objective is to help clients comprehend more about the house attributes and make informed decisions in their home-buying/selling process. The challenge is to determine a model that can effectively demonstrate how property attributes impact house prices. This model will help Every Door's agents to provide their clients with a clearer insight into how much pre-approval they need for their ideal property.

Audience:

Every Door Real Estate. The agency's mission is to provide clients with comprehensive insights to enhance their understanding on the correlation between house attributes and house prices.

Goal:

This analysis will be guiding Every Door Real Estate on the impact of various property attributes on their pricing structure. The anlysis aims to offer a clear understanding of how property attributes collectively influence the estimated value of homes, enabling strategic

decisions for optimal return on investment.

Data Understanding:Dataset

This analysis uses King County House Sales dataset, found in kc_house_data.csv in the data folder in GitHub repository. The dataset is significant in predicting house prices in relation to property attributes.
The data is imported and uploaded using necessary libraries. Some of the libraries imported for data analysis includes matplotlib, numpy,pandas, seaborn, scipy,and others. The structure of the data is made up of 21597 observations and 21 columns describing the property attributes.

Column Names and Descriptions for King County Data Set

1. **Id** - Unique identifier for a house
2. **Date** - Date house was sold
   3.**Price** - Sale price (prediction target)
   4.**Bedrooms** - Number of bedrooms
   5.**Bathrooms** - Number of bathrooms
   6.**Sqft_living** - Square footage of living space in the home
3. **Sqft_lot** - Square footage of the lot
4. **Floors** - Number of floors (levels) in house
5. **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
6. **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
7. **Condition** - How good the overall condition of the house is. Related to maintenance of house.
   See the King County Assessor Website for further explanation of each condition code
8. **Grade** - Overall grade of the house. Related to the construction and design of the house.
   See the King County Assessor Website for further explanation of each building grade code
9. **Sqft_above** - Square footage of house apart from basement
10. **Sqft_basement** - Square footage of the basement
11. **Yr_built** - Year when house was built
12. **Yr_renovated** - Year when house was renovated
13. **Zipcode** - ZIP Code used by the United States Postal Service
14. **Lat** - Latitude coordinate
15. **Long** - Longitude coordinate
16. **Sqft_living15** - The square footage of interior housing living space for the nearest 15 neighbors
17. **Sqft_lot15** - The square footage of the land lots of the nearest 15 neighbors

The data suggest that prices of houses depends on the features of the property. However data cleaning and wrangling is important to make the data more useful. The process of data cleaning and wrangling includes checking for missing values, duplicates, uniformity,and data formats.Additional cleaning including replacing missing values, dropping insignificant data, filling null values, and dropping duplicates using relevant functions and methods is also done to further make the data more valuable.

Data Loading and Importation of Modules

In [65]:
```
#Import necessary libraries
import scipy.stats as stats
import statsmodels.api as sm
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import pandas as pd
import numpy as np

import warnings
warnings.filterwarnings('ignore')
df=pd.read_csv("kc_house_data.csv")
df
```

Out[65]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
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

Data Cleaning and Wrangling:Data Preparation

The process of data cleaning and wrangling entailed formatting the data to remove duplicates, filled missing values, deleting undesirable symbols, and selecting required property variables for further modelling. From the observations, the data did not have any duplicates.
However, the variables:
a. Waterfront, yr_renovated, and view contained missing values which were filled with appropriate data. The missing values in waterfront were replaced with “NO” to assume the households lacked waterfronts. The missing values in view were replaced with “NONE” to assumed a lack of view. The missing values in yr_renovated were replaced with 0 to mean the houses were never renovated.
b. The values in the variable bathroom were standardized to decimal places because it was observed the values lacked uniformity of decimal places.
c. The date section was standardized to: day, month, and year format for uniformity.
d. The sqft_basement contained undesirable symbols which were removed, replaced with mean, and the data converted to numerical values. The data was also converted to 2 decimal places for uniformity.
e. The values in the latitude and longitude variables were converted to 2 decimal places for effective analysis and modelling.
f. The whole data set was confirmed clean and ready for modelling using the info() method.
g. During wrangling, the columns 'zipcode','id', 'lat', 'long', 'yr_renovated', 'yr_built' were observed to be insignificant and consequently were dropped.

In [66]:
```
#observe data structure
df.shape
```

Out[66]:
```
(21597, 21)
```

In [67]:
```
#check for duplicates
df.duplicated().sum()
```

Out[67]:
```
0
```

In [68]:
```
#confirm duplicates
df[df.duplicated(keep=False)]
```

Out[68]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade

0 rows × 21 columns

In [70]:
```
#Check for nulls
df.isna().sum()
```

Out[70]:
```
id                 0
date               0
price              0
bedrooms           0
bathrooms          0
sqft_living        0
sqft_lot           0
floors             0
waterfront      2376
view              63
condition          0
grade              0
sqft_above         0
sqft_basement      0
yr_built           0
yr_renovated    3842
zipcode            0
lat                0
long               0
sqft_living15      0
sqft_lot15         0
dtype: int64
```

In [71]:
```
#check data type and unique values in waterfront;
df["waterfront"].unique()
```

Out[71]:
```
array([nan, 'NO', 'YES'], dtype=object)
```

In [73]:
```
#check data type and unique values in waterfront
df["waterfront"].value_counts()
```

Out[73]:
```
NO     19075
YES      146
Name: waterfront, dtype: int64
```

In [74]:
```
#replace nan with the mode, that is No value
#Assumption is nan means no waterfront
#Check for successful replacements
df['waterfront'].replace(np.nan, "NO", inplace=True, regex=False)
```

In [75]:
```
#check for successful replacement
df["waterfront"].unique()
```

Out[75]:
```
array(['NO', 'YES'], dtype=object)
```

In [76]:
```
#check data type and unique values in conditions
df["condition"].unique().tolist()
```

Out[76]:
```
['Average', 'Very Good', 'Good', 'Poor', 'Fair']
```

In [77]:
```
#check data type and unique values in conditions
df["condition"].value_counts()
```

Out[77]:
```
Average      14020
Good          5677
Very Good     1701
Fair           170
Poor            29
Name: condition, dtype: int64
```

In [78]:
```
#check data type and unique values in bathrooms
#output shows inconsistent decimals
df["bathrooms"].unique()
```

Out[78]:
```
array([1.  , 2.25, 3.  , 2.  , 4.5 , 1.5 , 2.5 , 1.75, 2.75, 3.25, 4.  ,
       3.5 , 0.75, 4.75, 5.  , 4.25, 3.75, 1.25, 5.25, 6.  , 0.5 , 5.5 ,
       6.75, 5.75, 8.  , 7.5 , 7.75, 6.25, 6.5 ])
```

In [79]:
```
#Round to 2 decimals for uniformity
df["bathrooms"]=df["bathrooms"].round(2)
df["bathrooms"]
```

Out[79]:
```
0        1.00
1        2.25
2        1.00
3        3.00
4        2.00
         ...
21592    2.50
21593    2.50
21594    0.75
21595    2.50
21596    0.75
Name: bathrooms, Length: 21597, dtype: float64
```

In [80]:
```
#Let's assume that missing values in 'yr_renovated'
#mean no renovations were done, so we'll fill the missing values with 0.
# Fill missing values in 'yr_renovated' with 0
df['yr_renovated'].fillna(0, inplace=True)

# Verify the changes
print(df['yr_renovated'].value_counts())
```

```
0.0       20853
2014.0       73
2013.0       31
2003.0       31
2007.0       30
           ...
1951.0        1
1953.0        1
1946.0        1
1976.0        1
1948.0        1
Name: yr_renovated, Length: 70, dtype: int64
```

In [81]:
```
#check data type and unique values in view
df["view"].unique().tolist()
```

Out[81]:
```
['NONE', nan, 'GOOD', 'EXCELLENT', 'AVERAGE', 'FAIR']
```

In [82]:
```
#check data type and unique values in conditions
view_counts = df['view'].value_counts()
view_counts
```

Out[82]:
```
NONE         19422
AVERAGE        957
GOOD           508
FAIR           330
EXCELLENT      317
Name: view, dtype: int64
```

In [83]:
```
# Replacing the NaN values with mode NONE
# Verifying the Changes
df['view'].replace(np.nan, "NONE", inplace=True, regex=False)
df['view'].unique()
```

Out[83]:
```
array(['NONE', 'GOOD', 'EXCELLENT', 'AVERAGE', 'FAIR'], dtype=object)
```

In [84]:
```
# changing my date column for easier readability
df['date'] = pd.to_datetime(df['date'],format='%m/%d/%Y')
```

In [85]:
```
#check for unique values in sqft_basement
df['sqft_basement'].unique()
```

Out[85]:
```
array(['0.0', '400.0', '910.0', '1530.0', '?', '730.0', '1700.0', '300.0',
       '970.0', '760.0', '720.0', '700.0', '820.0', '780.0', '790.0',
       '330.0', '1620.0', '360.0', '588.0', '1510.0', '410.0', '990.0',
       '600.0', '560.0', '550.0', '1000.0', '1600.0', '500.0', '1040.0',
       '880.0', '1010.0', '240.0', '265.0', '290.0', '800.0', '540.0',
       '710.0', '840.0', '380.0', '770.0', '480.0', '570.0', '1490.0',
       '620.0', '1250.0', '1270.0', '120.0', '650.0', '180.0', '1130.0',
       '450.0', '1640.0', '1460.0', '1020.0', '1030.0', '750.0', '640.0',
       '1070.0', '490.0', '1310.0', '630.0', '2000.0', '390.0', '430.0',
       '850.0', '210.0', '1430.0', '1950.0', '440.0', '220.0', '1160.0',
       '860.0', '580.0', '2060.0', '1820.0', '1180.0', '200.0', '1150.0',
       '1200.0', '680.0', '530.0', '1450.0', '1170.0', '1080.0', '960.0',
       '280.0', '870.0', '1100.0', '460.0', '1400.0', '660.0', '1220.0',
       '900.0', '420.0', '1580.0', '1380.0', '475.0', '690.0', '270.0',
       '350.0', '935.0', '1370.0', '980.0', '1470.0', '160.0', '950.0',
       '50.0', '740.0', '1780.0', '1900.0', '340.0', '470.0', '370.0',
       '140.0', '1760.0', '130.0', '520.0', '890.0', '1110.0', '150.0',
       '1720.0', '810.0', '190.0', '1290.0', '670.0', '1800.0', '1120.0',
       '1810.0', '60.0', '1050.0', '940.0', '310.0', '930.0', '1390.0',
       '610.0', '1830.0', '1300.0', '510.0', '1330.0', '1590.0', '920.0',
       '1320.0', '1420.0', '1240.0', '1960.0', '1560.0', '2020.0',
       '1190.0', '2110.0', '1280.0', '250.0', '2390.0', '1230.0', '170.0',
       '830.0', '1260.0', '1410.0', '1340.0', '590.0', '1500.0', '1140.0',
       '260.0', '100.0', '320.0', '1480.0', '1060.0', '1284.0', '1670.0',
       '1350.0', '2570.0', '1090.0', '110.0', '2500.0', '90.0', '1940.0',
       '1550.0', '2350.0', '2490.0', '1481.0', '1360.0', '1135.0',
       '1520.0', '1850.0', '1660.0', '2130.0', '2600.0', '1690.0',
       '243.0', '1210.0', '1024.0', '1798.0', '1610.0', '1440.0',
       '1570.0', '1650.0', '704.0', '1910.0', '1630.0', '2360.0',
       '1852.0', '2090.0', '2400.0', '1790.0', '2150.0', '230.0', '70.0',
       '1680.0', '2100.0', '3000.0', '1870.0', '1710.0', '2030.0',
       '875.0', '1540.0', '2850.0', '2170.0', '506.0', '906.0', '145.0',
       '2040.0', '784.0', '1750.0', '374.0', '518.0', '2720.0', '2730.0',
       '1840.0', '3480.0', '2160.0', '1920.0', '2330.0', '1860.0',
       '2050.0', '4820.0', '1913.0', '80.0', '2010.0', '3260.0', '2200.0',
       '415.0', '1730.0', '652.0', '2196.0', '1930.0', '515.0', '40.0',
       '2080.0', '2580.0', '1548.0', '1740.0', '235.0', '861.0', '1890.0',
       '2220.0', '792.0', '2070.0', '4130.0', '2250.0', '2240.0',
       '1990.0', '768.0', '2550.0', '435.0', '1008.0', '2300.0', '2610.0',
       '666.0', '3500.0', '172.0', '1816.0', '2190.0', '1245.0', '1525.0',
       '1880.0', '862.0', '946.0', '1281.0', '414.0', '2180.0', '276.0',
       '1248.0', '602.0', '516.0', '176.0', '225.0', '1275.0', '266.0',
       '283.0', '65.0', '2310.0', '10.0', '1770.0', '2120.0', '295.0',
       '207.0', '915.0', '556.0', '417.0', '143.0', '508.0', '2810.0',
       '20.0', '274.0', '248.0'], dtype=object)
```

In [86]:
```
# replace '?' with NaN in sqft_basement
df['sqft_basement'].replace('?', np.nan, inplace=True)

# convert the column to a numeric data type
df['sqft_basement'] = pd.to_numeric(df['sqft_basement'])

# fill missing values with the mean value of the column
df['sqft_basement'].fillna(np.mean(df['sqft_basement']), inplace=True)
```

In [87]:
```
#check data type and unique values in 'sqft_basement'
df['sqft_basement'].unique()
```

Out[87]:
```
array([   0.        ,  400.        ,  910.        , 1530.        ,
       291.85172397,  730.        , 1700.        ,  300.        ,
       970.        ,  760.        ,  720.        ,  700.        ,
       820.        ,  780.        ,  790.        ,  330.        ,
      1620.        ,  360.        ,  588.        , 1510.        ,
       410.        ,  990.        ,  600.        ,  560.        ,
       550.        , 1000.        , 1600.        ,  500.        ,
      1040.        ,  880.        , 1010.        ,  240.        ,
       265.        ,  290.        ,  800.        ,  540.        ,
       710.        ,  840.        ,  380.        ,  770.        ,
       480.        ,  570.        , 1490.        ,  620.        ,
      1250.        , 1270.        ,  120.        ,  650.        ,
       180.        , 1130.        ,  450.        , 1640.        ,
      1460.        , 1020.        , 1030.        ,  750.        ,
       640.        , 1070.        ,  490.        , 1310.        ,
       630.        , 2000.        ,  390.        ,  430.        ,
       850.        ,  210.        , 1430.        , 1950.        ,
       440.        ,  220.        , 1160.        ,  860.        ,
       580.        , 2060.        , 1820.        , 1180.        ,
       200.        , 1150.        , 1200.        ,  680.        ,
       530.        , 1450.        , 1170.        , 1080.        ,
       960.        ,  280.        ,  870.        , 1100.        ,
       460.        , 1400.        ,  660.        , 1220.        ,
       900.        ,  420.        , 1580.        , 1380.        ,
       475.        ,  690.        ,  270.        ,  350.        ,
       935.        , 1370.        ,  980.        , 1470.        ,
       160.        ,  950.        ,   50.        ,  740.        ,
      1780.        , 1900.        ,  340.        ,  470.        ,
       370.        ,  140.        , 1760.        ,  130.        ,
       520.        ,  890.        , 1110.        ,  150.        ,
      1720.        ,  810.        ,  190.        , 1290.        ,
       670.        , 1800.        , 1120.        , 1810.        ,
        60.        , 1050.        ,  940.        ,  310.        ,
       930.        , 1390.        ,  610.        , 1830.        ,
      1300.        ,  510.        , 1330.        , 1590.        ,
       920.        , 1320.        , 1420.        , 1240.        ,
      1960.        , 1560.        , 2020.        , 1190.        ,
      2110.        , 1280.        ,  250.        , 2390.        ,
      1230.        ,  170.        ,  830.        , 1260.        ,
      1410.        , 1340.        ,  590.        , 1500.        ,
      1140.        ,  260.        ,  100.        ,  320.        ,
      1480.        , 1060.        , 1284.        , 1670.        ,
      1350.        , 2570.        , 1090.        ,  110.        ,
      2500.        ,   90.        , 1940.        , 1550.        ,
      2350.        , 2490.        , 1481.        , 1360.        ,
      1135.        , 1520.        , 1850.        , 1660.        ,
      2130.        , 2600.        , 1690.        ,  243.        ,
      1210.        , 1024.        , 1798.        , 1610.        ,
      1440.        , 1570.        , 1650.        ,  704.        ,
      1910.        , 1630.        , 2360.        , 1852.        ,
      2090.        , 2400.        , 1790.        , 2150.        ,
       230.        ,   70.        , 1680.        , 2100.        ,
      3000.        , 1870.        , 1710.        , 2030.        ,
       875.        , 1540.        , 2850.        , 2170.        ,
       506.        ,  906.        ,  145.        , 2040.        ,
       784.        , 1750.        ,  374.        ,  518.        ,
      2720.        , 2730.        , 1840.        , 3480.        ,
```

```
       2160.        , 1920.        , 2330.        , 1860.        ,
       2050.        , 4820.        , 1913.        ,   80.        ,
       2010.        , 3260.        , 2200.        ,  415.        ,
       1730.        ,  652.        , 2196.        , 1930.        ,
        515.        ,   40.        , 2080.        , 2580.        ,
       1548.        , 1740.        ,  235.        ,  861.        ,
       1890.        , 2220.        ,  792.        , 2070.        ,
       4130.        , 2250.        , 2240.        , 1990.        ,
        768.        , 2550.        ,  435.        , 1008.        ,
       2300.        , 2610.        ,  666.        , 3500.        ,
        172.        , 1816.        , 2190.        , 1245.        ,
       1525.        , 1880.        ,  862.        ,  946.        ,
       1281.        ,  414.        , 2180.        ,  276.        ,
       1248.        ,  602.        ,  516.        ,  176.        ,
        225.        , 1275.        ,  266.        ,  283.        ,
         65.        , 2310.        ,   10.        , 1770.        ,
       2120.        ,  295.        ,  207.        ,  915.        ,
        556.        ,  417.        ,  143.        ,  508.        ,
       2810.        ,   20.        ,  274.        ,  248.        ])
```

In [88]:
```
#replace symbols in sqft_basement
df['sqft_basement']=df['sqft_basement'].replace({',': ''}, regex=True)
df['sqft_basement']
```

Out[88]:
```
0          0.0
1        400.0
2          0.0
3        910.0
4          0.0
         ...
21592      0.0
21593      0.0
21594      0.0
21595      0.0
21596      0.0
Name: sqft_basement, Length: 21597, dtype: float64
```

In [89]:
```
#round values to 2 decimals places for uniformity
df["sqft_basement"]=df["sqft_basement"].round(2)
df["sqft_basement"]
```

Out[89]:
```
0          0.0
1        400.0
2          0.0
3        910.0
4          0.0
         ...
21592      0.0
21593      0.0
21594      0.0
21595      0.0
21596      0.0
Name: sqft_basement, Length: 21597, dtype: float64
```

In [90]:
```
#check for unique values in latitude
df['lat'].unique()
```

Out[90]:
```
array([47.5112, 47.721 , 47.7379, ..., 47.3906, 47.3339, 47.6502])
```

In [91]:
```
#round values to 2 decimals places for uniformity
df["lat"]=df["lat"].round(2)
df["lat"]
```

Out[91]:
```
0        47.51
1        47.72
2        47.74
3        47.52
4        47.62
         ...
21592    47.70
21593    47.51
21594    47.59
21595    47.53
21596    47.59
Name: lat, Length: 21597, dtype: float64
```

In [92]:
```
#check for uniformity in longitude
df['long'].unique()
```

Out[92]:
```
array([-122.257, -122.319, -122.233, -122.393, -122.045, -122.005,
       -122.327, -122.315, -122.337, -122.031, -122.145, -122.292,
       -122.229, -122.394, -122.375, -121.962, -122.343, -122.21 ,
       -122.306, -122.341, -122.169, -122.166, -122.172, -122.218,
       -122.36 , -122.314, -122.304, -122.11 , -122.07 , -122.357,
       -122.368, -122.157, -122.31 , -122.132, -122.362, -122.282,
       -122.18 , -122.027, -122.347, -122.016, -122.364, -122.175,
       -121.977, -122.371, -122.151, -122.301, -122.451, -122.322,
       -122.189, -122.384, -122.369, -122.281, -122.29 , -122.114,
       -122.122, -122.116, -122.149, -122.339, -122.335, -122.344,
       -122.32 , -122.297, -122.192, -122.215, -122.16 , -122.179,
       -122.287, -122.036, -122.073, -121.987, -122.125, -122.34 ,
       -122.025, -122.008, -122.291, -122.365, -122.199, -122.194,
       -122.387, -122.372, -122.391, -122.351, -122.386, -122.249,
       -122.277, -122.378, -121.958, -121.714, -122.08 , -122.196,
       -122.184, -122.133, -122.38 , -122.082, -122.109, -122.053,
       -122.349, -122.295, -122.253, -122.248, -122.303, -122.294,
       -122.226, -122.266, -122.098, -122.212, -122.244, -122.39 ,
       -122.352, -121.85 , -122.152, -122.054, -122.072, -121.998,
```

In [93]:
```
#round values to 2 decimals places for uniformity
df["long"]=df["long"].round(2)
df["long"]
```

Out[93]:
```
0       -122.26
1       -122.32
2       -122.23
3       -122.39
4       -122.04
          ...
21592   -122.35
21593   -122.36
21594   -122.30
21595   -122.07
21596   -122.30
Name: long, Length: 21597, dtype: float64
```

In [94]:
```
#check for unique values in zipcode
df['zipcode'].unique()
```

Out[94]:
```
array([98178, 98125, 98028, 98136, 98074, 98053, 98003, 98198, 98146,
       98038, 98007, 98115, 98107, 98126, 98019, 98103, 98002, 98133,
       98040, 98092, 98030, 98119, 98112, 98052, 98027, 98117, 98058,
       98001, 98056, 98166, 98023, 98070, 98148, 98105, 98042, 98008,
       98059, 98122, 98144, 98004, 98005, 98034, 98075, 98116, 98010,
       98118, 98199, 98032, 98045, 98102, 98077, 98108, 98168, 98177,
       98065, 98029, 98006, 98109, 98022, 98033, 98155, 98024, 98011,
       98031, 98106, 98072, 98188, 98014, 98055, 98039])
```

In [95]:
```
#drop unneccessary columns, less significant
df.drop(['id','zipcode', 'lat', 'long', 'yr_renovated', 'yr_built'],
        axis=1, inplace=True)
```

In [96]:
```
#confirm columns dropped
df.head(4)
```

Out[96]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condi
0	2014-10-13	221900.0	3	1.00	1180	5650	1.0	NO	NONE	Aver
1	2014-12-09	538000.0	3	2.25	2570	7242	2.0	NO	NONE	Aver
2	2015-02-25	180000.0	2	1.00	770	10000	1.0	NO	NONE	Aver
3	2014-12-09	604000.0	4	3.00	1960	5000	1.0	NO	NONE	G

In [97]:
```
#understand structure of dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           21597 non-null  datetime64[ns]
 1   price          21597 non-null  float64
 2   bedrooms       21597 non-null  int64
 3   bathrooms      21597 non-null  float64
 4   sqft_living    21597 non-null  int64
 5   sqft_lot       21597 non-null  int64
 6   floors         21597 non-null  float64
 7   waterfront     21597 non-null  object
 8   view           21597 non-null  object
 9   condition      21597 non-null  object
 10  grade          21597 non-null  object
 11  sqft_above     21597 non-null  int64
 12  sqft_basement  21597 non-null  float64
 13  sqft_living15  21597 non-null  int64
 14  sqft_lot15     21597 non-null  int64
dtypes: datetime64[ns](1), float64(4), int64(6), object(4)
memory usage: 2.5+ MB
```

Data Exploration

In the further exploring the data, the required columns were converted from object to integer data type.This was the case with sqft_basement. A column, total_space, was created to include sqft_living and sqft_lot variables. An additional season column was added to analyse how housing prices across the different seasons. A further analysis was conducted using the describe()method. The data was then check for correlation visually and statistically to determined relevant columns for use in modelling. When checking for correlation, variables than were greater than .5 were considered more correlated to price (with a correlation of 1.0). These variables were retained for machine learning and modelling through a multiple regression analysis. Additionally, the selected variables were checked for skewness. All the variables were skewed and the data was standardized to normal distribution. The outliers were checked and removed to make the regression model for effective. Lastly, variables with categorical data were converted to dummy variables and the data added to the dataframe.

In [98]:
```
#check for columns
df.columns
```

Out[98]:
```
Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

In [99]:
```
#check data types
df.dtypes
```

Out[99]:
```
date             datetime64[ns]
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
sqft_living15             int64
sqft_lot15                int64
dtype: object
```

In [100]:
```
#convert to numeric from object data type
df['sqft_basement']=pd.to_numeric(df['sqft_basement'].replace('?', '0')).asty
df['sqft_basement']
```

Out[100]:
```
0          0
1        400
2          0
3        910
4          0
        ...
21592      0
21593      0
21594      0
21595      0
21596      0
Name: sqft_basement, Length: 21597, dtype: int64
```

Feature Engineering

In [101]:
```
#merge 'sqft_living' and 'sqft_lot' columns
df['total_space'] = df['sqft_living'] + df['sqft_lot']
df['total_space']
```

Out[101]:
```
0         6830
1         9812
2        10770
3         6960
4         9760
         ...
21592     2661
21593     8123
21594     2370
21595     3988
21596     2096
Name: total_space, Length: 21597, dtype: int64
```

In [102]:
```
#Create column season to check house prices in different seasons
def get_season(month):
    if 3 <= month <= 5:
        return 'Spring'
    elif 6 <= month <= 8:
        return 'Summer'
    elif 9 <= month <= 11:
        return 'Fall'
    else:
        return 'Winter'
```

In [103]:
```
# Create a new 'season' column by applying the function to the 'date' column
df['season'] = df['date'].dt.month.apply(get_season)
```

In [104]:
```
#check dataframe
df
```

Out[104]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	c
0	2014-10-13	221900.0	3	1.00	1180	5650	1.0	NO	NONE	
1	2014-12-09	538000.0	3	2.25	2570	7242	2.0	NO	NONE	
2	2015-02-25	180000.0	2	1.00	770	10000	1.0	NO	NONE	
3	2014-12-09	604000.0	4	3.00	1960	5000	1.0	NO	NONE	
4	2015-02-18	510000.0	3	2.00	1680	8080	1.0	NO	NONE	
...	...	...	...	...	...	...	...	...	...	...
21592	2014-05-21	360000.0	3	2.50	1530	1131	3.0	NO	NONE	
21593	2015-02-23	400000.0	4	2.50	2310	5813	2.0	NO	NONE	
21594	2014-06-23	402101.0	2	0.75	1020	1350	2.0	NO	NONE	
21595	2015-01-16	400000.0	3	2.50	1600	2388	2.0	NO	NONE	
21596	2014-10-15	325000.0	2	0.75	1020	1076	2.0	NO	NONE	

21597 rows × 17 columns

Data Analysis

In [105]:
```
# Check data summary
df.describe()
```

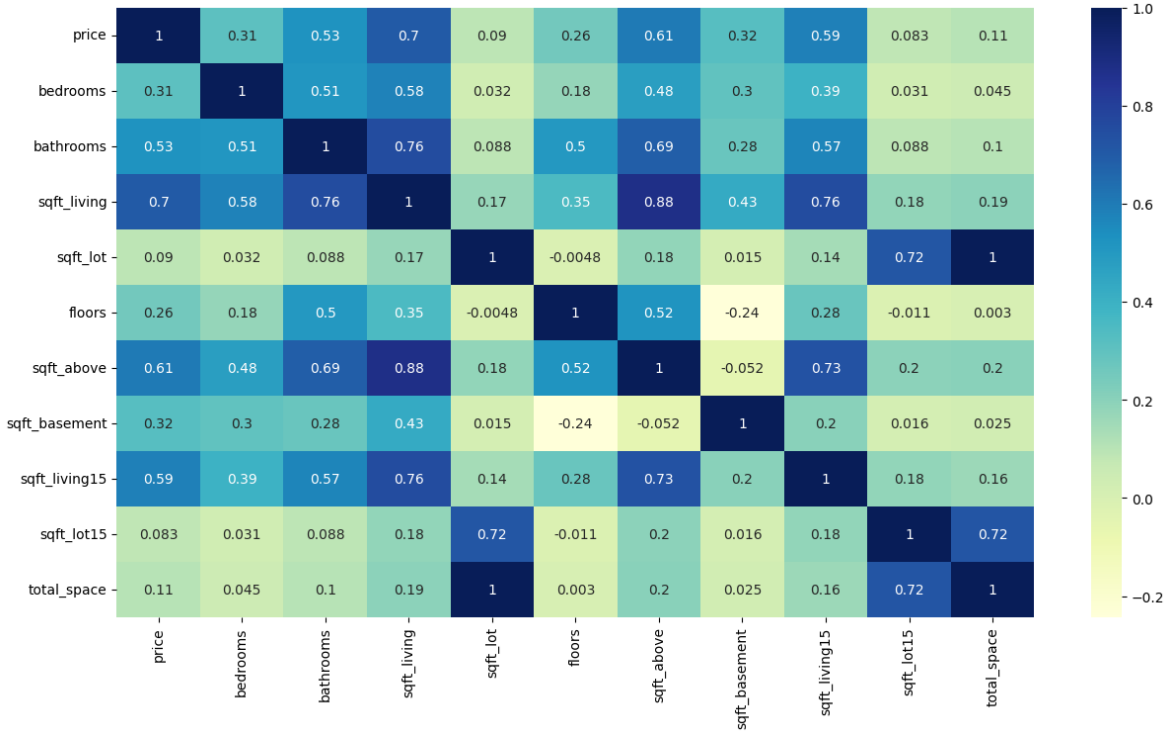
Out[105]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	
count	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	21597.000000	21
mean	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	1.494096	1
std	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	0.539683	
min	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.000000	
25%	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1.000000	1
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	1
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	2.000000	2
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	9

In [106]:
```
#check for correlations
df.corr()["price"].sort_values(ascending=False)
```

Out[106]:
```
price            1.000000
sqft_living      0.701917
sqft_above       0.605368
sqft_living15    0.585241
bathrooms        0.525906
sqft_basement    0.322193
bedrooms         0.308787
floors           0.256804
total_space      0.105009
sqft_lot         0.089876
sqft_lot15       0.082845
Name: price, dtype: float64
```

In [107]:
```
#Visualize correlations
plt.figure(figsize=(15,8))
sns.heatmap(df.corr(),annot=True, cmap="YlGnBu");
```



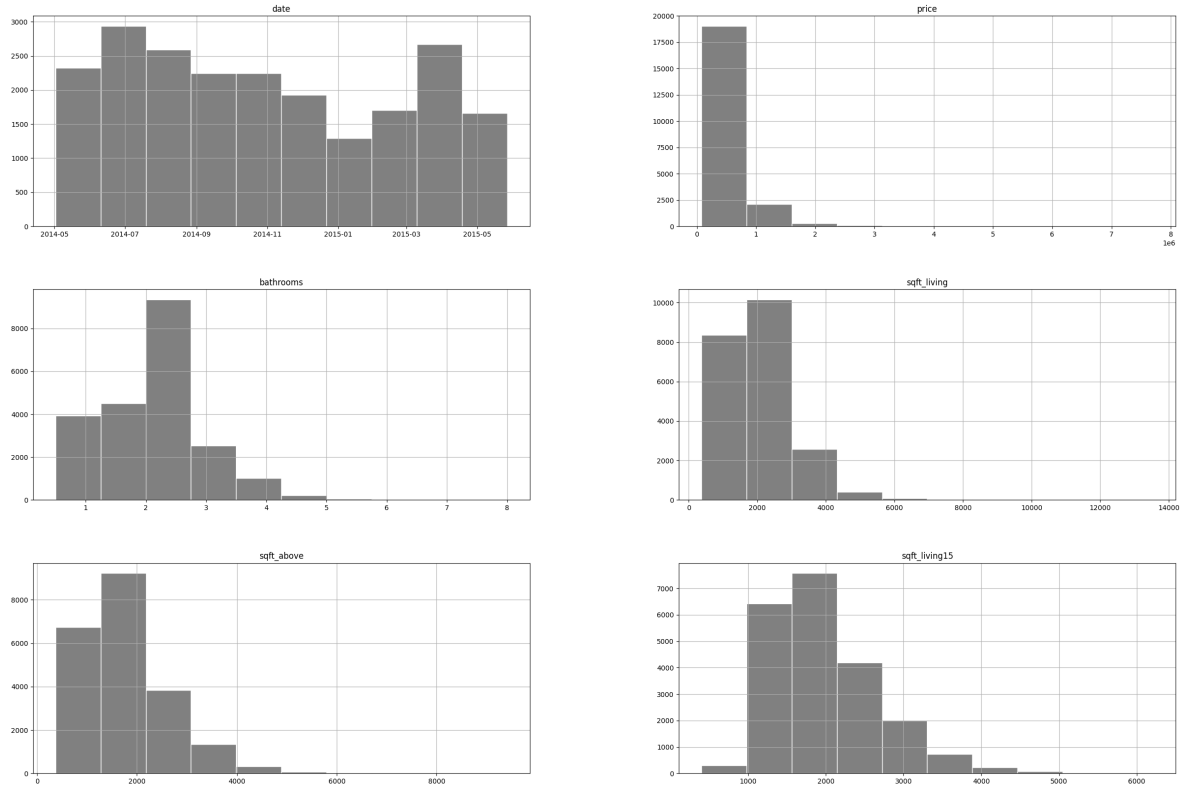
In [108]:
```
#drop less correlated variables
df.drop(['bedrooms','sqft_lot', 'floors', 'sqft_basement', 'sqft_lot15', 'tota
        axis=1, inplace=True)
```

In [109]:
```
#confirm changes in the dataframe
df.head(2)
```

Out[109]:

	date	price	bathrooms	sqft_living	waterfront	view	condition	grade	sqft_above	s
0	2014-10-13	221900.0	1.00	1180	NO	NONE	Average	7 Average	1180	
1	2014-12-09	538000.0	2.25	2570	NO	NONE	Average	7 Average	2170	

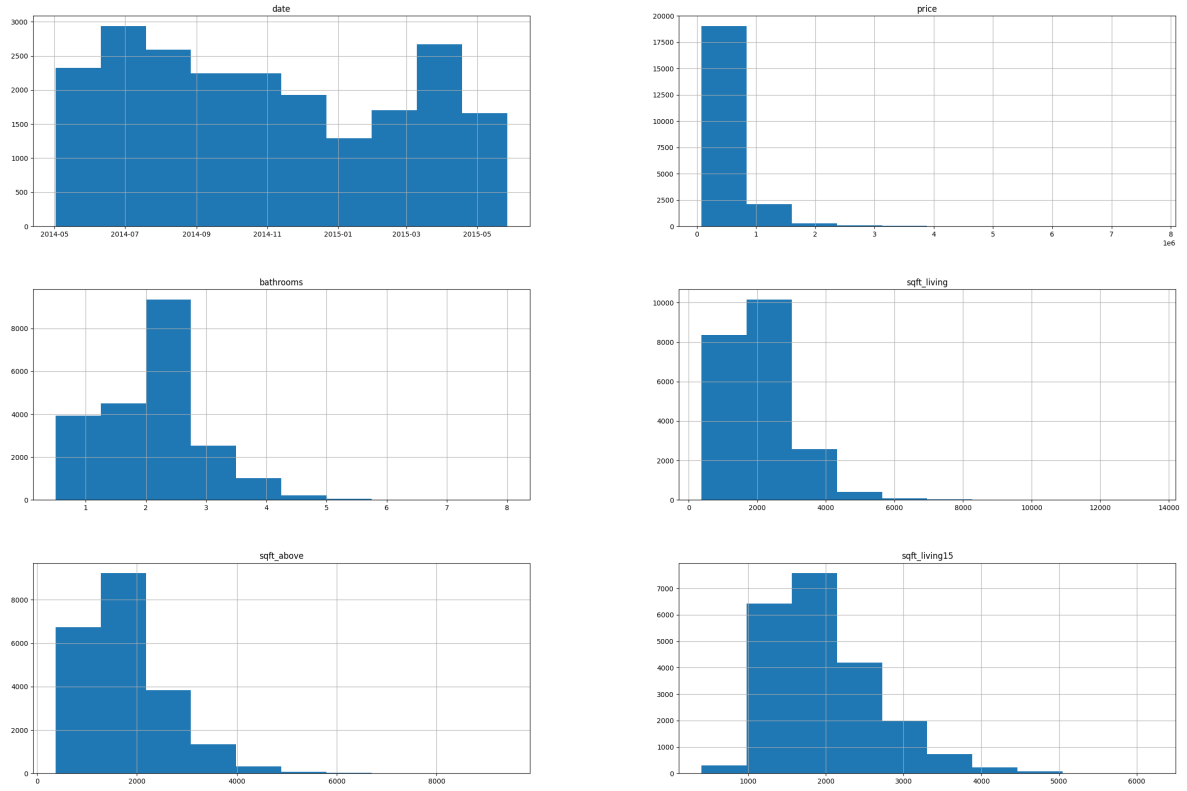
In [110]:
```
from sklearn.model_selection import train_test_split
#check for basic correlations in the data
df.hist(figsize=(30,20), color='grey', edgecolor='white');
```



Deep Processing

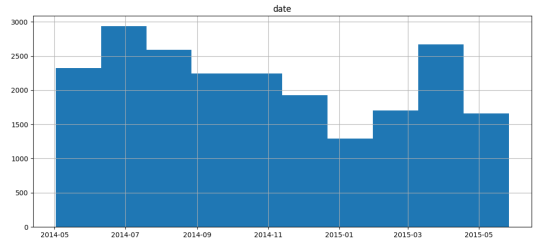
Checking and Removing Skewness

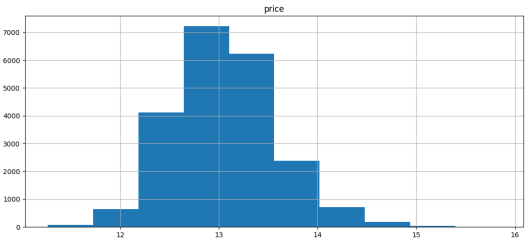
In [111]:
```
#Visualize skewness using a histogram
df.hist(figsize=(30,20));
```

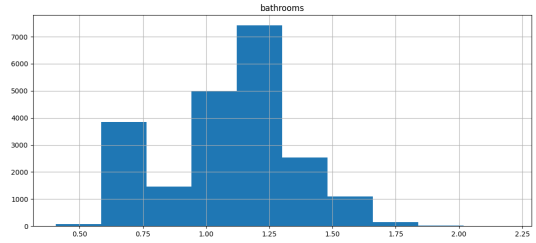


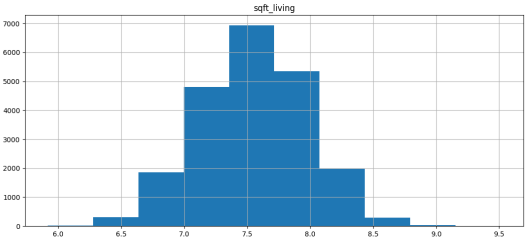
In [112]:
```
#standardize data to normal distribution
df['price']=np.log(df['price']+1)
df['bathrooms']=np.log(df['bathrooms']+1)
df['sqft_living']=np.log(df['sqft_living']+1)
df['sqft_above']=np.log(df['sqft_above']+1)
df['sqft_living15']=np.log(df['sqft_living15']+1)
```

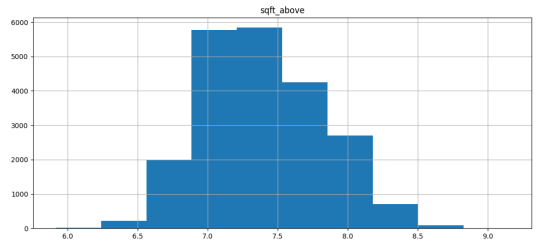
In [113]:
```
#confirm successful data standardization
df.hist(figsize=(30,20));
```

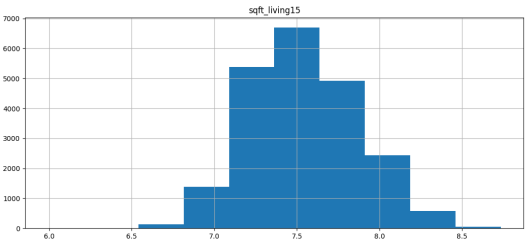




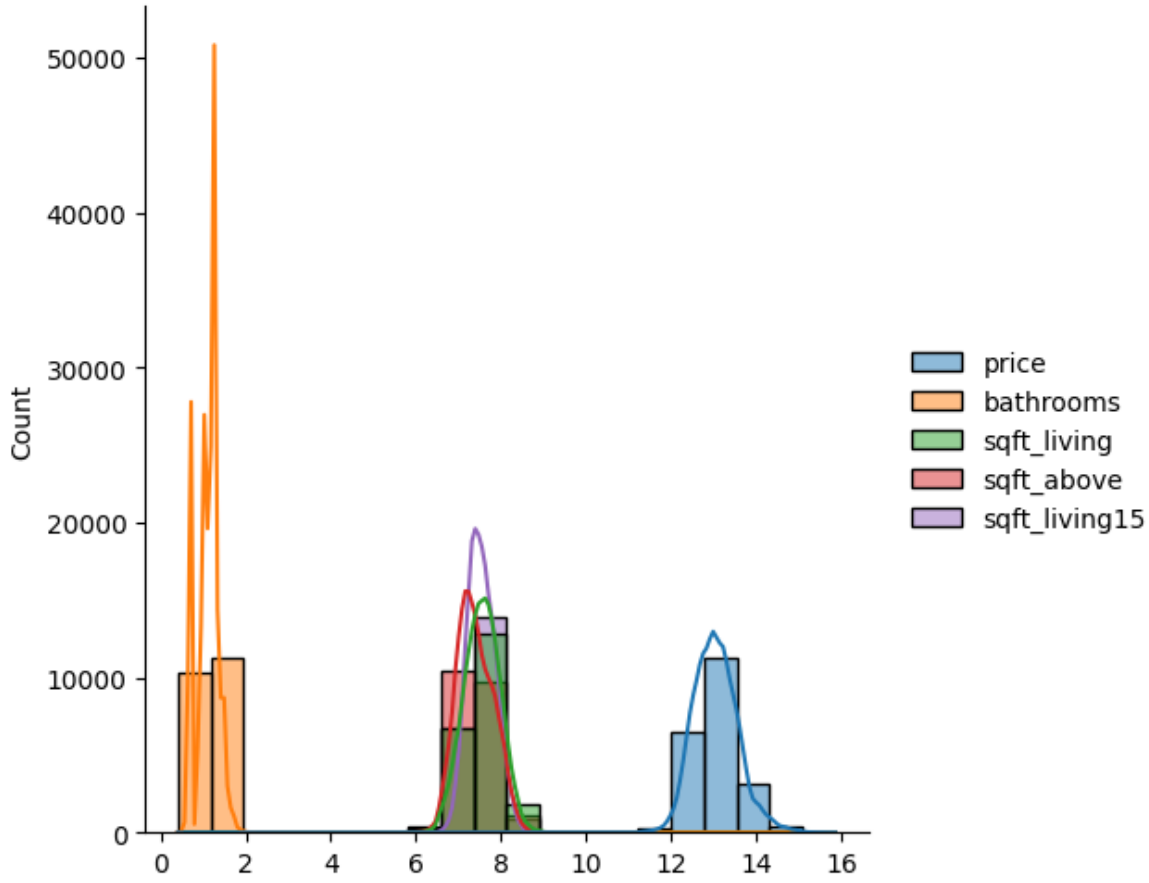








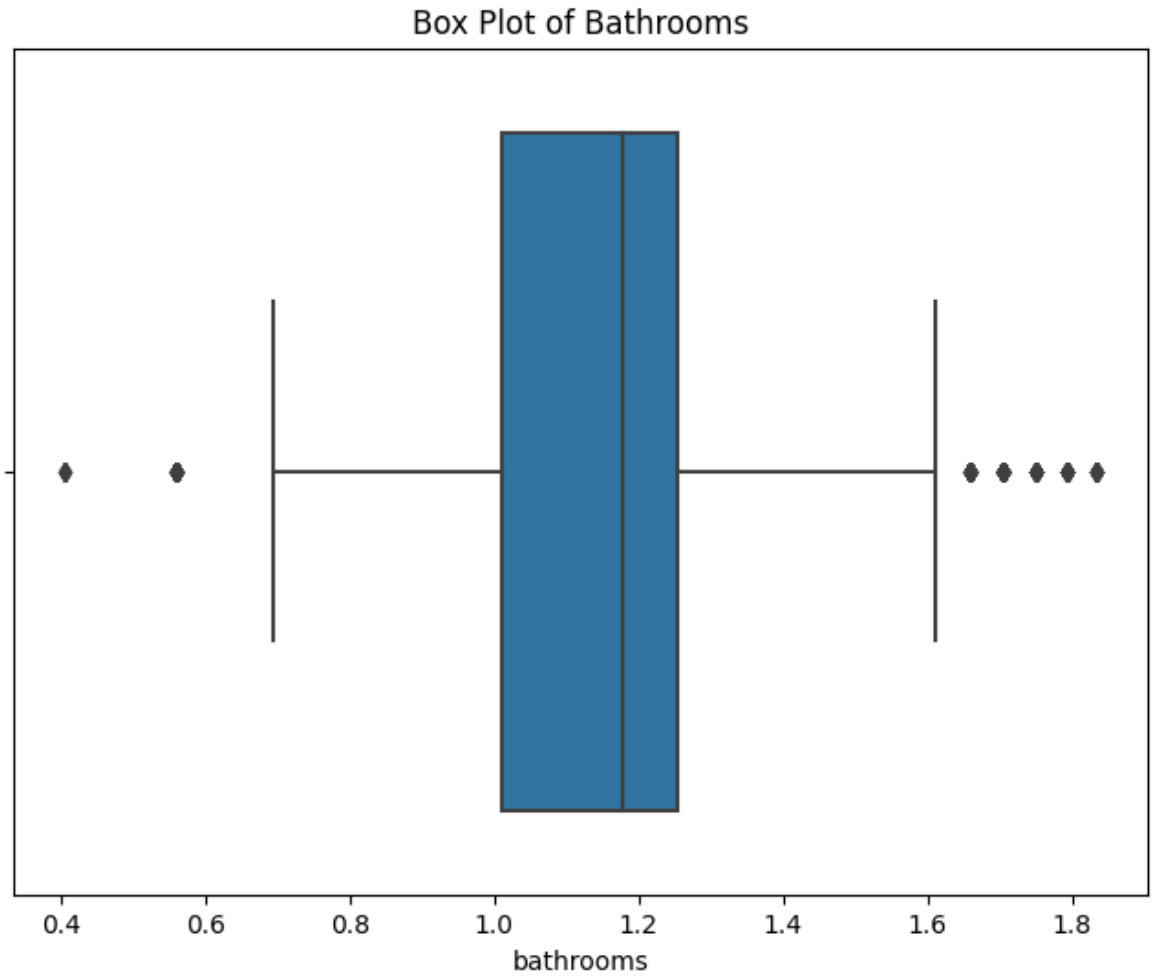
In [114]:
```
#Check for distribution using displot
sns.displot(df,bins=20,kde=True);
```



In [115]:
```
#Bathroom variable shows abnormal distribution
#Remove for outliers in the bathroom column
import numpy as np
count = 0
bathroom_outliers = []
mean = np.mean(df['bathrooms'])
max_distance = np.std(df['bathrooms']) * 3

for idx, row in df['bathrooms'].T.items():
    if abs(row-mean) >= max_distance:
        count += 1
        df.drop(idx, inplace=True)
count
```

Out[115]: 30

In [116]:
```
# Assuming df is your DataFrame with cleaned data
#Confirm outliers in boxplot
min_values, max_values = df.bathrooms.quantile([0.010, 0.95])
min_values, max_values
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['bathrooms'])
plt.title('Box Plot of Bathrooms')
plt.show()
```



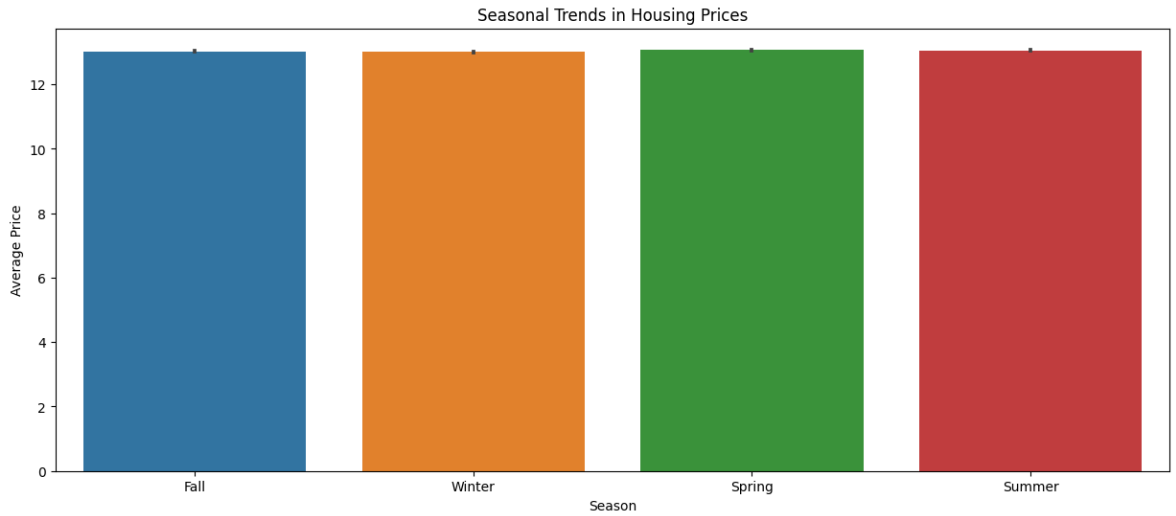
In [117]:
```
# creating summary statistics of pricing of each season
# Group by 'season' and calculate summary statistics
df.groupby('season')['price'].agg(["count","mean"])
```

Out[117]:

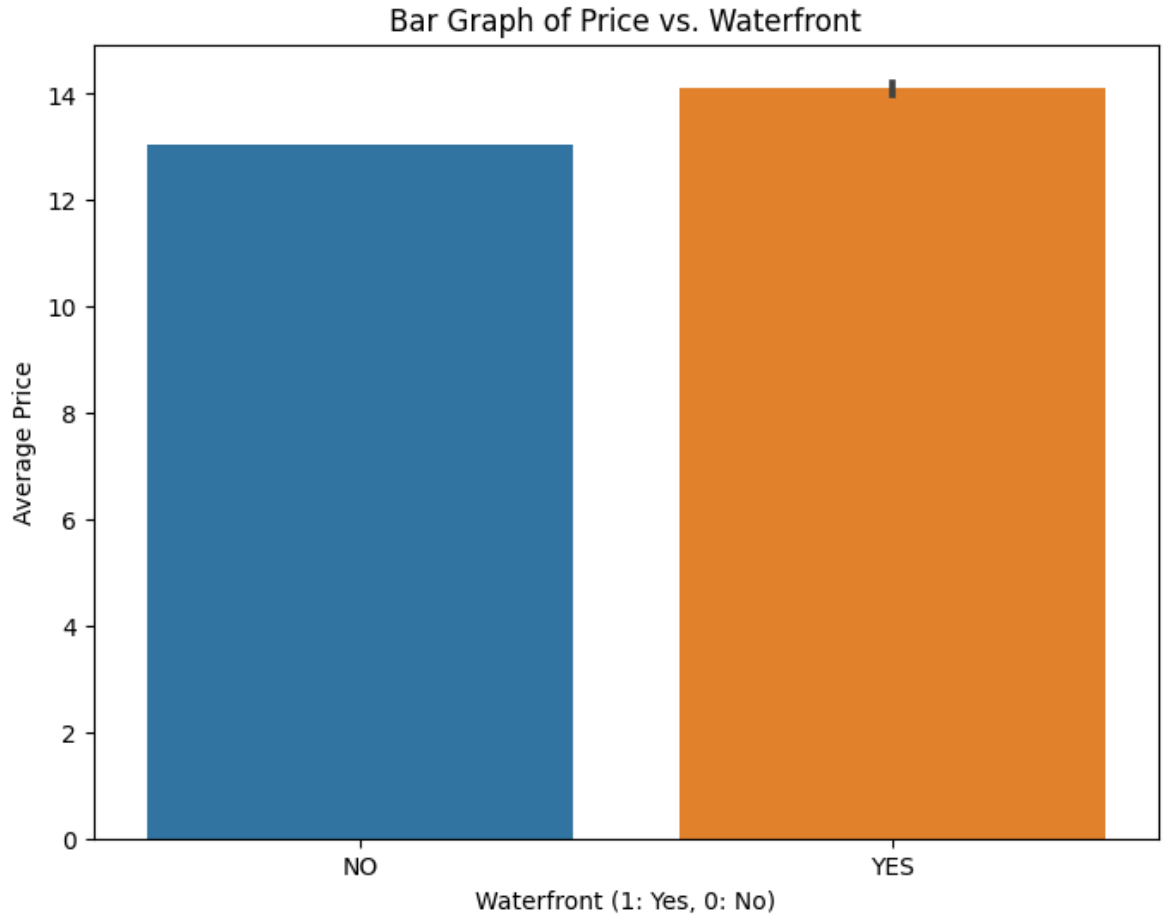
season	count	mean
Fall	5049	13.027457
Spring	6510	13.070408
Summer	6317	13.059062
Winter	3691	13.006037

From the summary, we have more average sales during spring evidently seen in value counts and mean.

In [118]:
```
#Visualization: relationship between price and seasons
# Plotting seasonal trends using a bar chart
plt.figure(figsize=(15, 6))
sns.barplot(x='season', y='price', data=df)
plt.title('Seasonal Trends in Housing Prices')
plt.xlabel('Season')
plt.ylabel('Average Price')
plt.show()
```

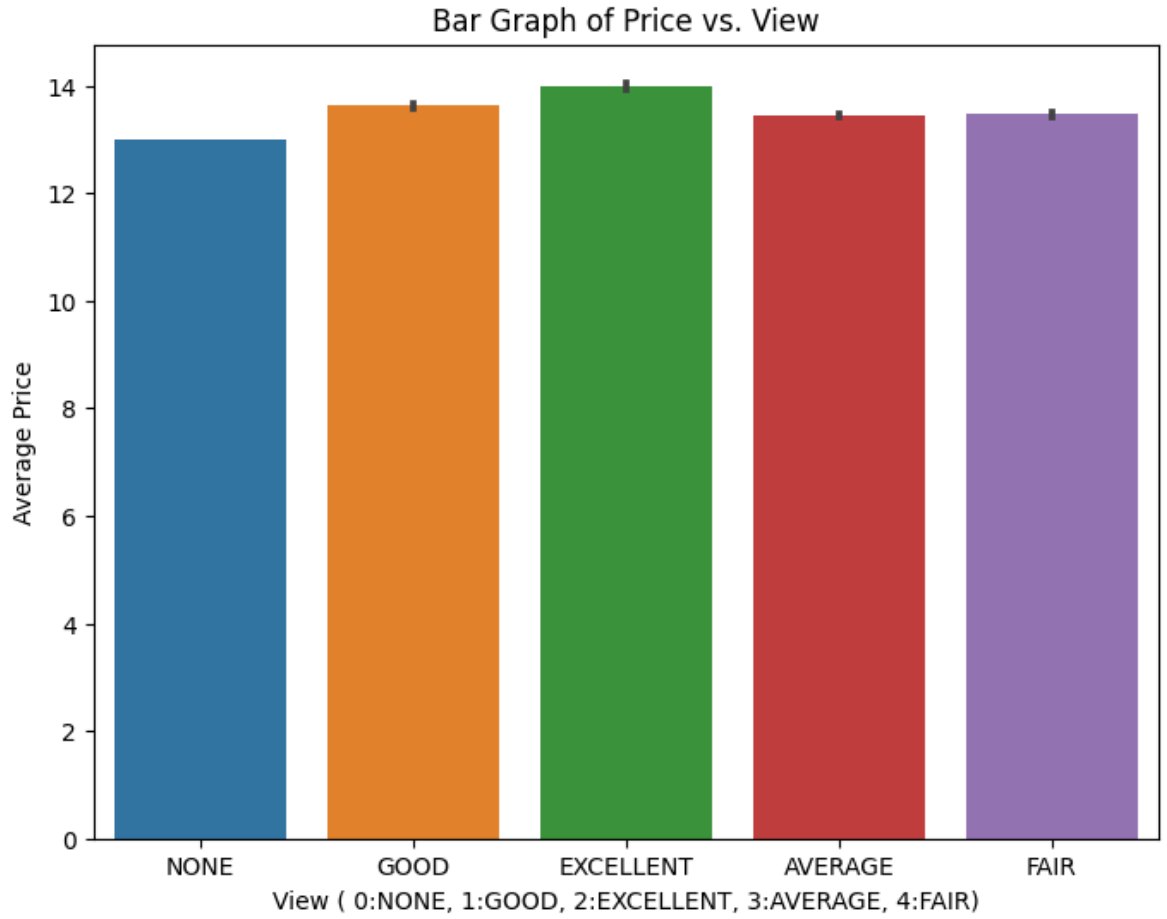


In [119]:
```
plt.figure(figsize=(8, 6))
sns.barplot(x='waterfront', y='price', data=df)
plt.title('Bar Graph of Price vs. Waterfront')
plt.xlabel('Waterfront (1: Yes, 0: No)')
plt.ylabel('Average Price')
plt.show()
```



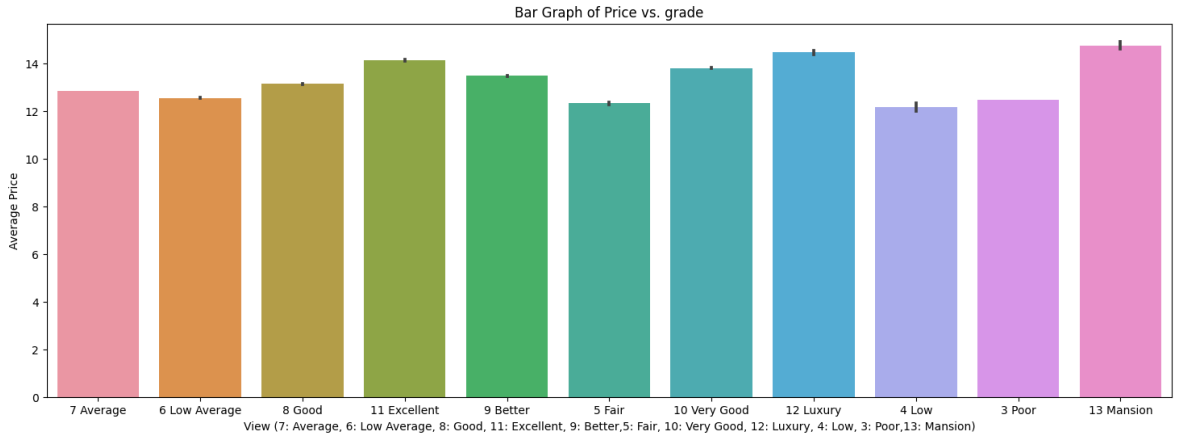
We see that properties on a waterfront typically have higher sale prices than properties that are inland

In [120]:
```
plt.figure(figsize=(8, 6))
sns.barplot(x='view', y='price', data=df)
plt.title('Bar Graph of Price vs. View')
plt.xlabel('View ( 0:NONE, 1:GOOD, 2:EXCELLENT, 3:AVERAGE, 4:FAIR)')
plt.ylabel('Average Price')
plt.show()
```



We see that properties with an excellent view typically have higher sale prices

In [121]:
```
plt.figure(figsize=(18, 6))
sns.barplot(x='grade', y='price', data=df)
plt.title('Bar Graph of Price vs. grade')
plt.xlabel('View (7: Average, 6: Low Average, 8: Good, 11: Excellent, 9: Bette
plt.ylabel('Average Price')
plt.show()
```



We see that properties graded mansion, luxury,excellent, and very good typically have higher sale prices

In [122]:
```
plt.figure(figsize=(12, 6))
sns.barplot(x='condition', y='price', data=df)
plt.title('Bar Graph of Price vs. condition')
plt.xlabel('Condition (1: Average, 2: Very Good, 3: Good, 4:Poor, 5:Fair)')
plt.ylabel('Average Price')
plt.show()
```

We see that properties with very good condition typically have higher sale prices than properties that are average, good, poor, and fair condition

Analysing and Combined Modelling

Regression Models

Here we begin to examine potential relationships between different combinations of variables. We are looking to see if we can build a model that shows strong relationships between our variables of interest.

Now that the categorical variables are removed, let us check for correlation of independent vs dependent variables.
In this case, our dependent variable being "price"

In [123]:
```
#Correlations
df.corr()["price"]
```

Out[123]:
```
price            1.000000
bathrooms        0.526878
sqft_living      0.670892
sqft_above       0.581305
sqft_living15    0.605553
Name: price, dtype: float64
```

Correlation is a measure of causal relationship rather than causation
In this summary above, the variables show low-to-medium-to-strong correlations with price
The property value with a highest correlation with price is the sqft_living followed by sqft_living15.

Modelling for a simple Linear Regression
Regression models are evaluated against a "baseline".
For simple linear regression, this baseline is an "intercept-only" model that just predicts the mean of the dependent variable every time.
For multiple linear regression, build a simple linear regression to be that baseline.

In this case, the simple linear regression will be based price vs sqft_living.
sqft_living showed the highest and strongest correlation hence fit to create a baseline model for the multiple linear regression

In other words, set up the formula

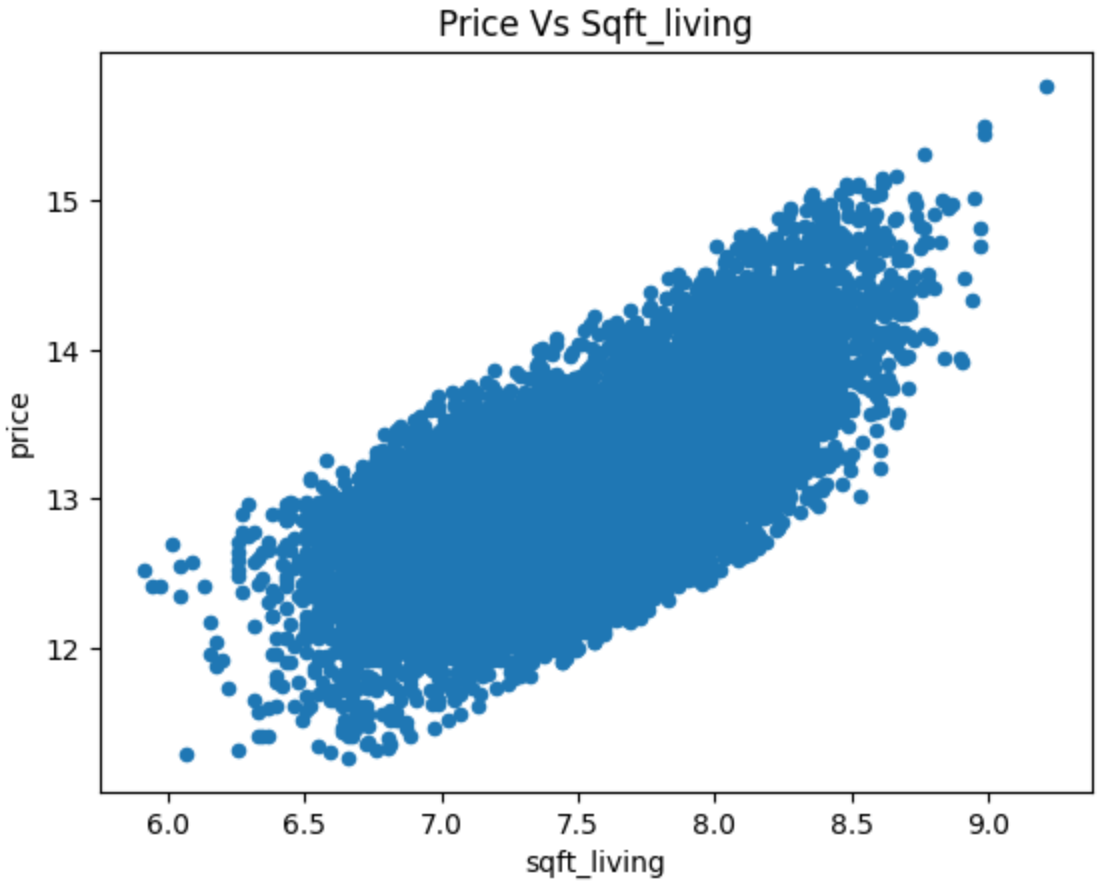
$\hat{y}$ =β0^+β1^x

Where $\hat{y}$ is price, the dependent (endogenous) variable, and x is sqft_living, the independent (exogenous) variable. When we fit our model, we are looking for β1^ (the slope) and β0^ (the intercept).

Baseline_model

Model One

In [124]:
```
#Price vs Sqft_living
X="sqft_living"
y="price"
#plot a scatter plot to show sqft_living vs price
df.plot(x="sqft_living", y="price", kind="scatter", title="Price Vs Sqft_livi
```



In [125]:
```
#fit the baseline_model since the scatter shows correlation
X=df["sqft_living"]
y=df["price"]
baseline_model = sm.OLS(endog=y, exog=sm.add_constant(X))
baseline_model
```

Out[125]:
```
<statsmodels.regression.linear_model.OLS at 0x7dd55d170e80>
```

In [126]:
```
baseline_modelresults = baseline_model.fit()
baseline_modelresults
```

Out[126]:
```
<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x7dd557e1d
ed0>
```

In [127]:
```
#Inferential analysis using summary for baseline_modelresults
baseline_modelresults.summary()
```

Out[127]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.450
Model:	OLS	**Adj. R-squared:**	0.450
Method:	Least Squares	**F-statistic:**	1.765e+04
Date:	Thu, 04 Jan 2024	**Prob (F-statistic):**	0.00
Time:	06:51:42	**Log-Likelihood:**	-10170.
No. Observations:	21567	**AIC:**	2.034e+04
Df Residuals:	21565	**BIC:**	2.036e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	6.7597	0.047	142.640	0.000	6.667	6.853
sqft_living	0.8327	0.006	132.857	0.000	0.820	0.845

Omnibus:	122.212	**Durbin-Watson:**	1.979
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	110.977
Skew:	0.136	**Prob(JB):**	7.97e-25
Kurtosis:	2.777	**Cond. No.**	138.

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

Simple Linear Regression Results and Interpretation

Looking at the summary above,

a. Constant/y-intercept/C=6.7597
b. slope/m/x-intercept=0.8327
c. therefore, regression line is given by:

$$\hat{price} = 6.7597 + 0.8327 sqft\_living$$

- The model is statistically significant overall, with an F-statistic p-value well below 0.05
- The model explains (R-squared value= 0.493) 50% of the variance in Price explained by sqft_living
- The model coefficients (`const` and `sqft_Living`) are both statistically significant, with t-statistic p-values well below 0.05
- If a house had sqft_living of 0 sqft, we would expect the price to be about 6.7597K (thousands of dollars)

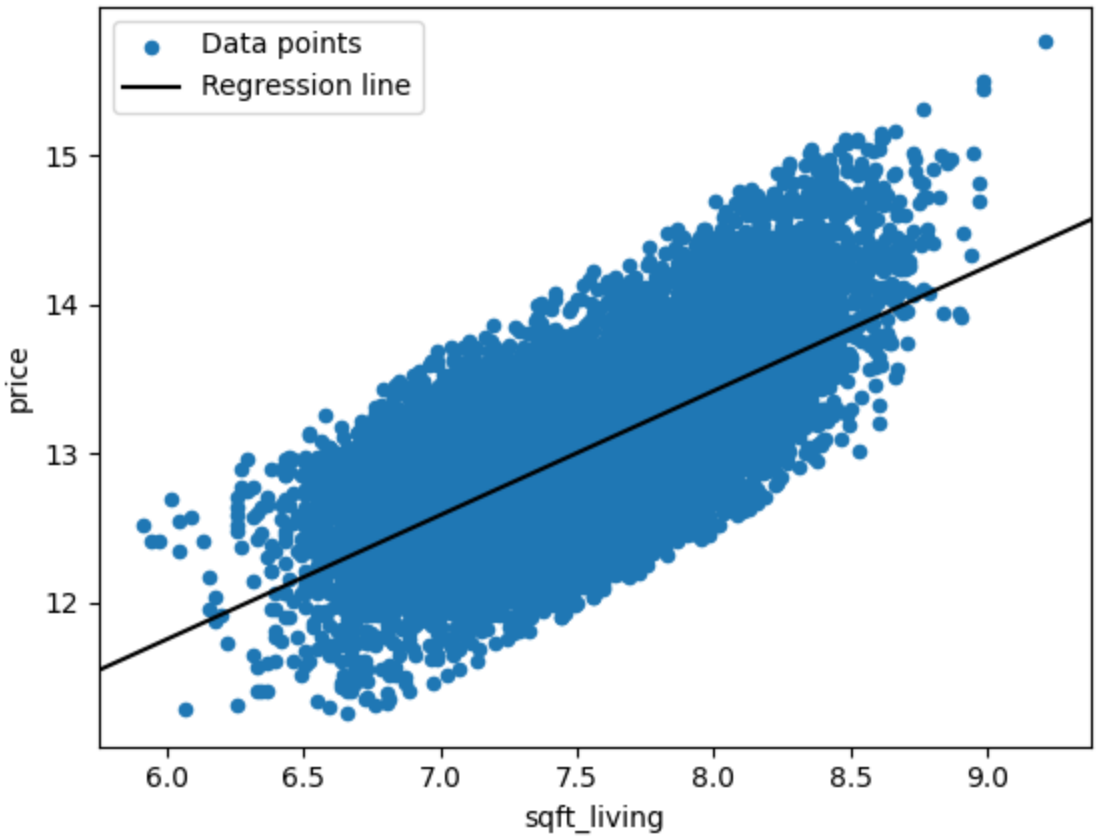
- For each additional 1sqft living of a property, we see an associated increase in price of about 0.8327K (thousands of dollars)

Note that all of these coefficients represent associations rather than causation.

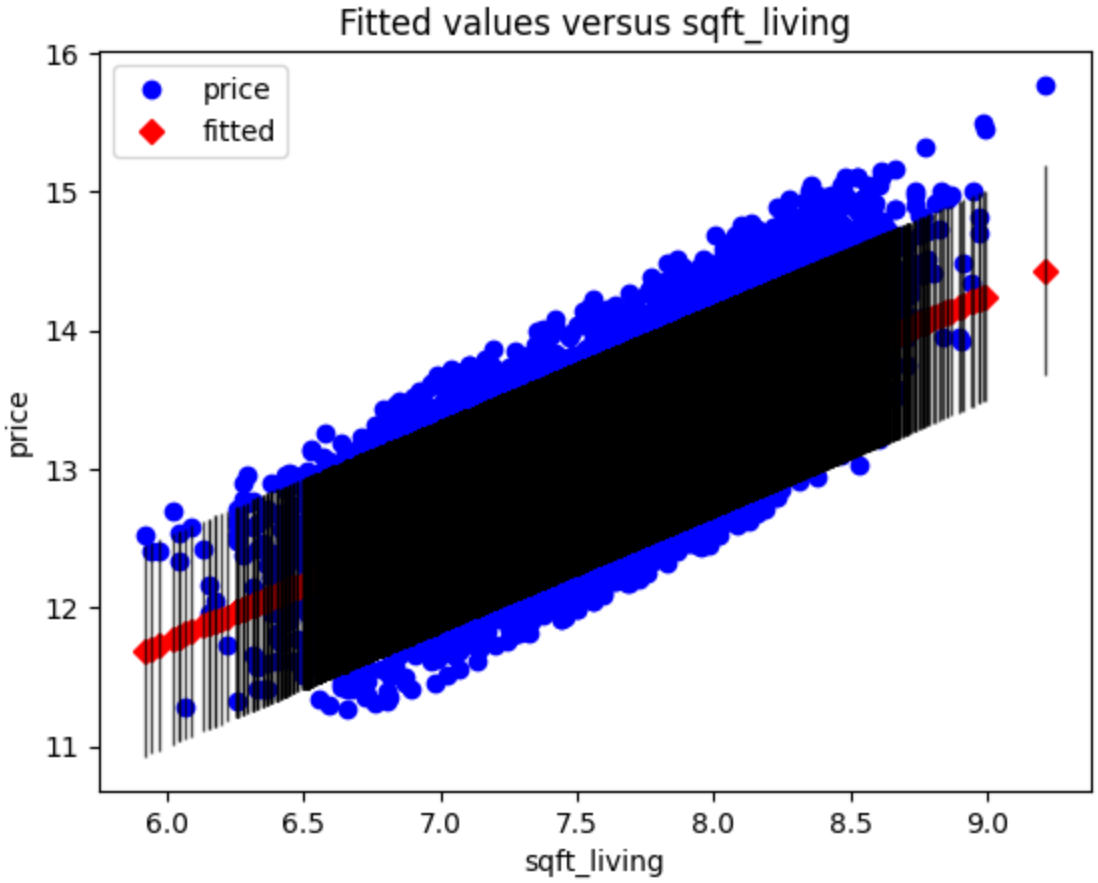
Regression and Visualization¶

We'll also plot the actual vs. predicted values:

In [128]:
```
#Show the model
#For Regression line
fig, ax = plt.subplots()
df.plot.scatter(x="sqft_living", y="price", label="Data points", ax=ax)
sm.graphics.abline_plot(model_results=baseline_modelresults,
                        label="Regression line",
                        color="black",
                        ax=ax)
ax.legend();
```



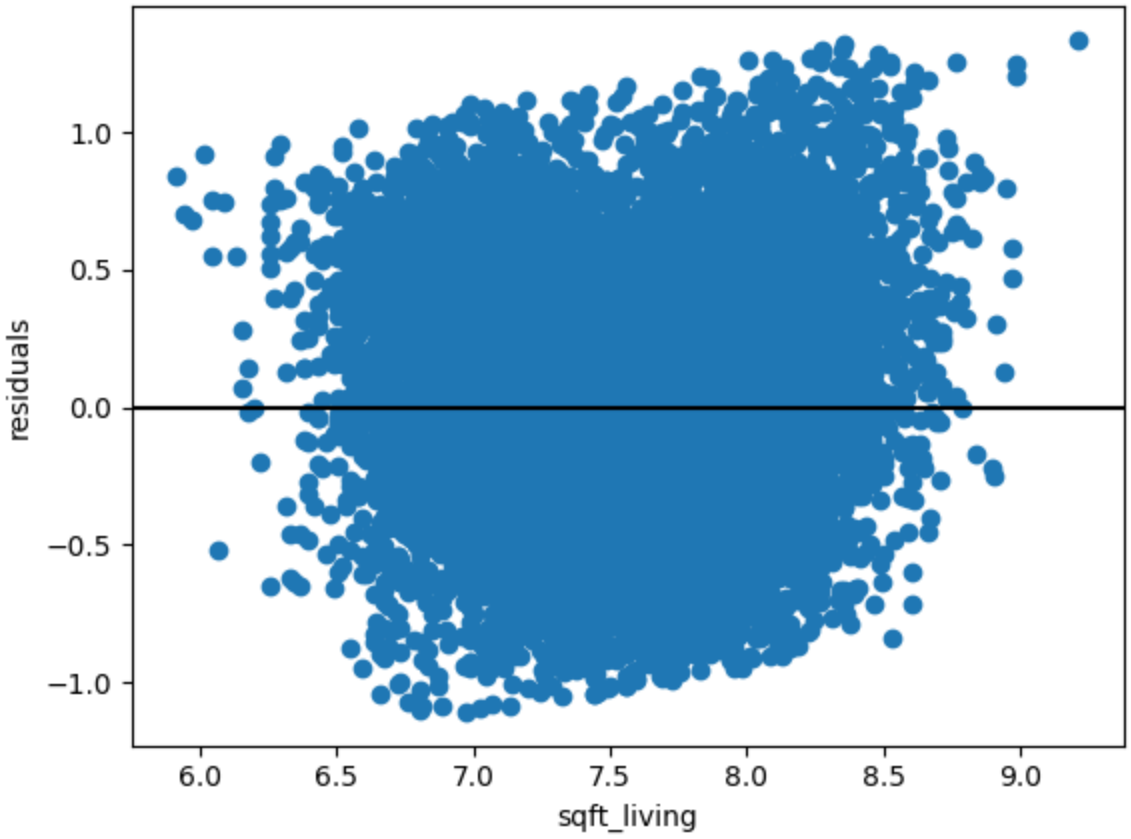
In [ ]:
```
sm.graphics.plot_fit(baseline_modelresults, "sqft_living")
plt.show()
```



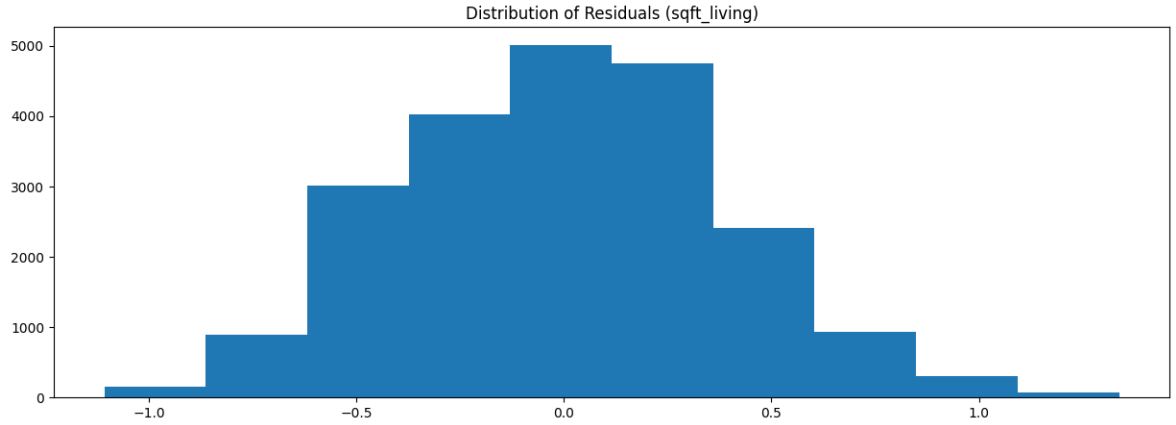
In [129]:
```
#Calculate residues
baseline_modelresults.resid
```

Out[129]:
```
0        -0.340199
1        -0.102339
2        -0.194390
3         0.238900
4         0.198024
             ...
21592    -0.072453
21593    -0.309956
21594     0.375496
21595    -0.004320
21596     0.162619
Length: 21567, dtype: float64
```

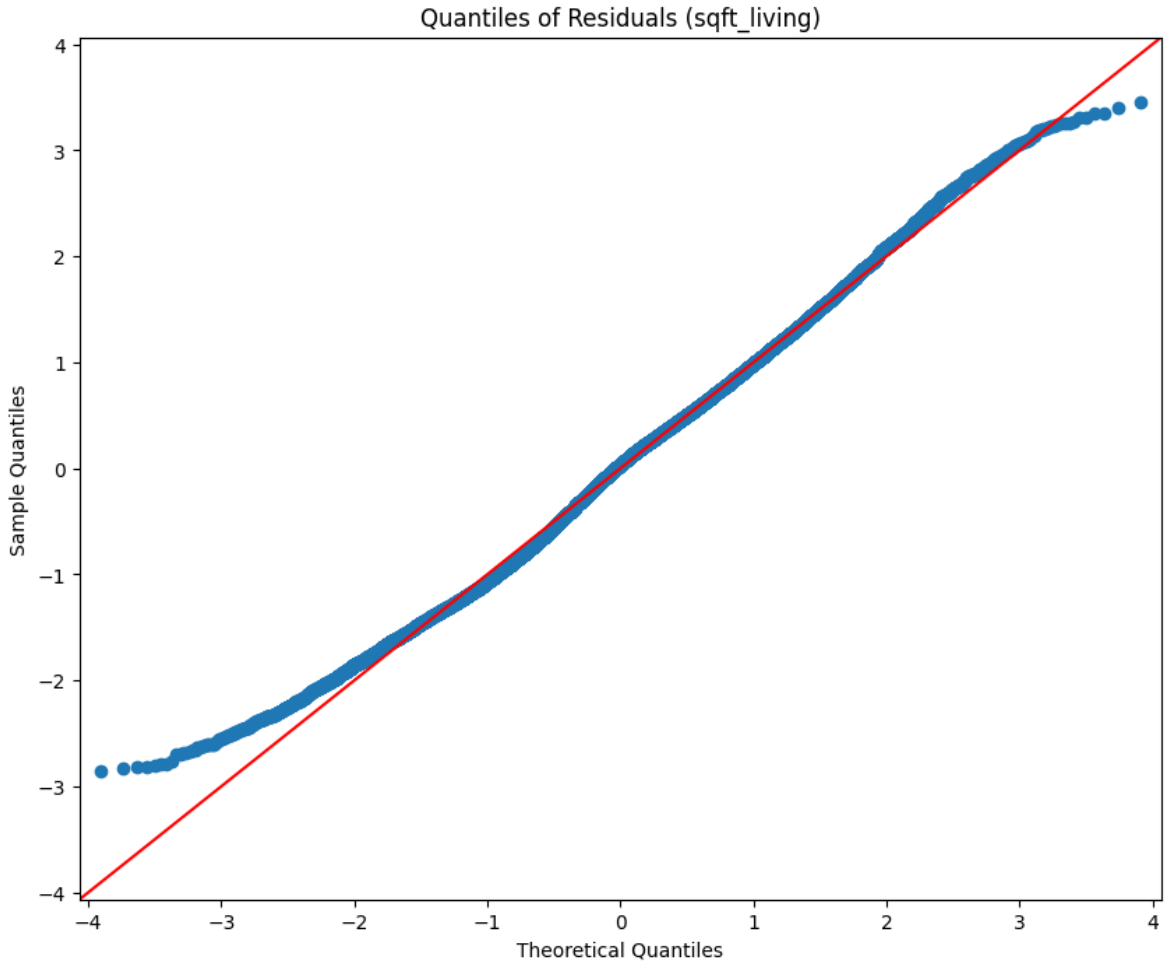
In [130]:
```
#Visualize residues
fig, ax = plt.subplots()

ax.scatter(df["sqft_living"], baseline_modelresults.resid)
ax.axhline(y=0, color="black")
ax.set_xlabel("sqft_living")
ax.set_ylabel("residuals");
```



In [131]:
```
#show residue using histogram
fig, ax = plt.subplots(figsize=(15,5))
ax.hist(baseline_modelresults.resid)
ax.set_ylabel=("Price")
ax.set_xlabel=("sqft_living")
ax.set_title("Distribution of Residuals (sqft_living)");
```



In [ ]:
```
#Residue plot using qq-plot
from scipy.stats import norm
fig, ax = plt.subplots(figsize=(10,8))
sm.graphics.qqplot(baseline_modelresults.resid, dist=norm, line="45", fit=Tru
ax.set_title("Quantiles of Residuals (sqft_living)")
plt.show()
```



Model evaluation

In [132]:
```
mae = baseline_modelresults.resid.abs().sum() / len(y)
mae
```

Out[132]:
```
0.3141413132521112
```

model is off by 0.31K in any given prediction

Model Two :With additional independent variables

The second model employs variables with strong causal relationship with price in reference to the correlation summary. The variables added in the second model include bathrooms, sqft_above, and sqft_living15.

In [133]:
```
X_secondvariable=df[["sqft_living","bathrooms","sqft_above","sqft_living15"]]
X_secondvariable.head(3)
```

Out[133]:

	sqft_living	bathrooms	sqft_above	sqft_living15
0	7.074117	0.693147	7.074117	7.201171
1	7.852050	1.178655	7.682943	7.433075
2	6.647688	0.693147	6.647688	7.908755

In [134]:
```
#for second model and Summary
second_model = sm.OLS(y, sm.add_constant(X_secondvariable))
second_modelresults = second_model.fit()

print(second_modelresults.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.476
Model:                            OLS   Adj. R-squared:                  0.476
Method:                 Least Squares   F-statistic:                     4903.
Date:                Thu, 04 Jan 2024   Prob (F-statistic):               0.00
Time:                        06:52:26   Log-Likelihood:                -9644.0
No. Observations:               21567   AIC:                         1.930e+04
Df Residuals:                   21562   BIC:                         1.934e+04
Df Model:                           4
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const            5.7587      0.070     81.685      0.000       5.621       5.897
sqft_living      0.6355      0.014     43.829      0.000       0.607       0.664
bathrooms        0.0799      0.016      4.972      0.000       0.048       0.111
sqft_above      -0.0764      0.012     -6.194      0.000      -0.101      -0.052
sqft_living15    0.3935      0.012     32.495      0.000       0.370       0.417
==============================================================================
Omnibus:                      142.903   Durbin-Watson:                   1.978
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              121.225
Skew:                           0.125   Prob(JB):                     4.75e-27
Kurtosis:                       2.731   Cond. No.                         362.
==============================================================================

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

Results and Interpretations

Model Regression equation:

$$\hat{Price} = 5.7587 + 0.6355sqft\_living + 0.0799bathrooms - 0.0764sqft\_above + 0.393$$

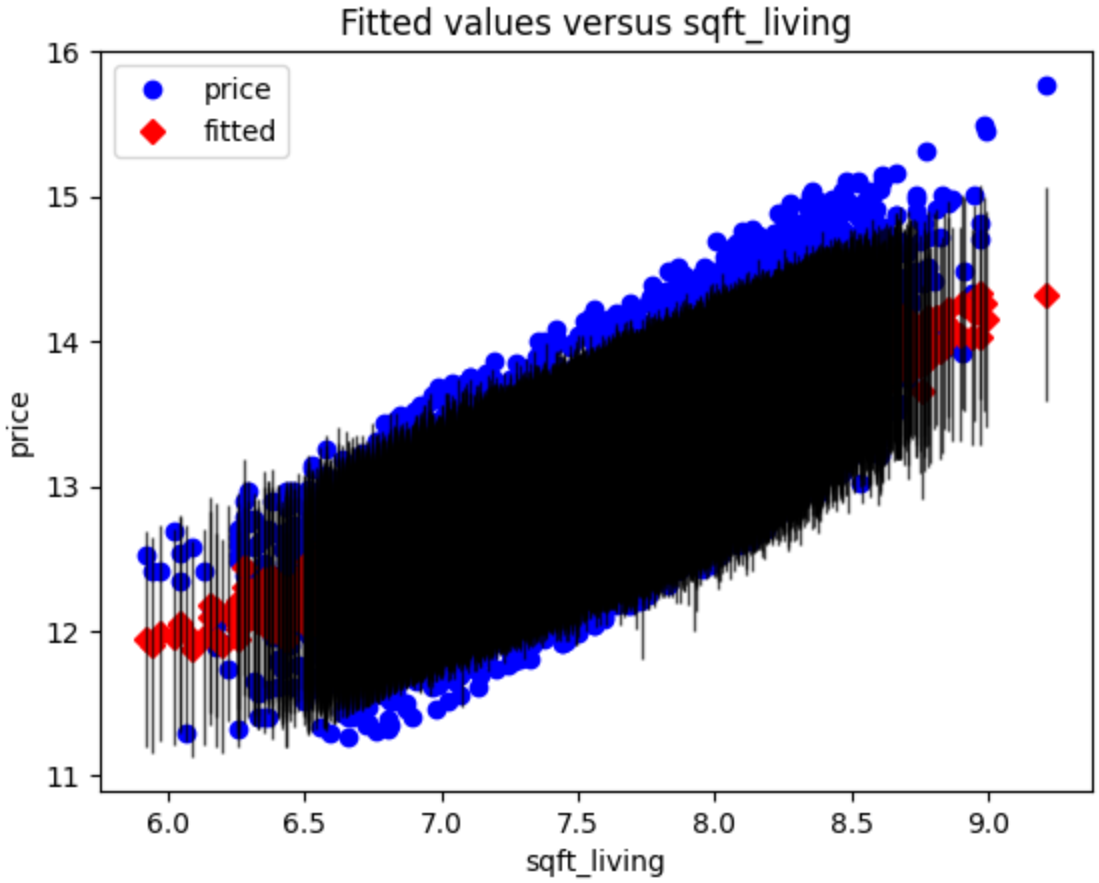
The coeffecient has decreased from 6.7597 in the first model to 5.7587 in the second model. This is because of the intercept is now with respect to bathrooms, sqft_living15, and sqft_above

- The model is statistically significant overall, with an F-statistic p-value well below 0.05
- The model explains about 43% of the variance in price
- The model coefficients (`const` , `bathrooms` , `sqft_above` and `sqft_living15`) are all statistically significant, with t-statistic p-values well below 0.05
- If a house had sqft_above of 0 sqft, or sqft_living15 of 0, or no bathroom, we would expect the price to be about 5.7587K (thousands of dollars)
- For each additional 1sqft_living15, we see an associated increase in price of about 0.3935K
- For each additional 1sqft_above, we see an associated decrease in price of about 0.0764K
- For each additional 1bathroom, we see an associated increase in price of about 0.0799 (thousands of dollars)
- For each additional 1sqft_living, we see an associated increase in price of about 0.6355 (thousands of dollars)
  This is a little bit smaller of a decrease than we saw with the simple model, but not a big change. This means that second model was not meaningfully confounding in the relationship between price and sqft_living
  Note that all of these coefficients represent associations rather than causation.

Visualization for Model two

a. **Model fit**

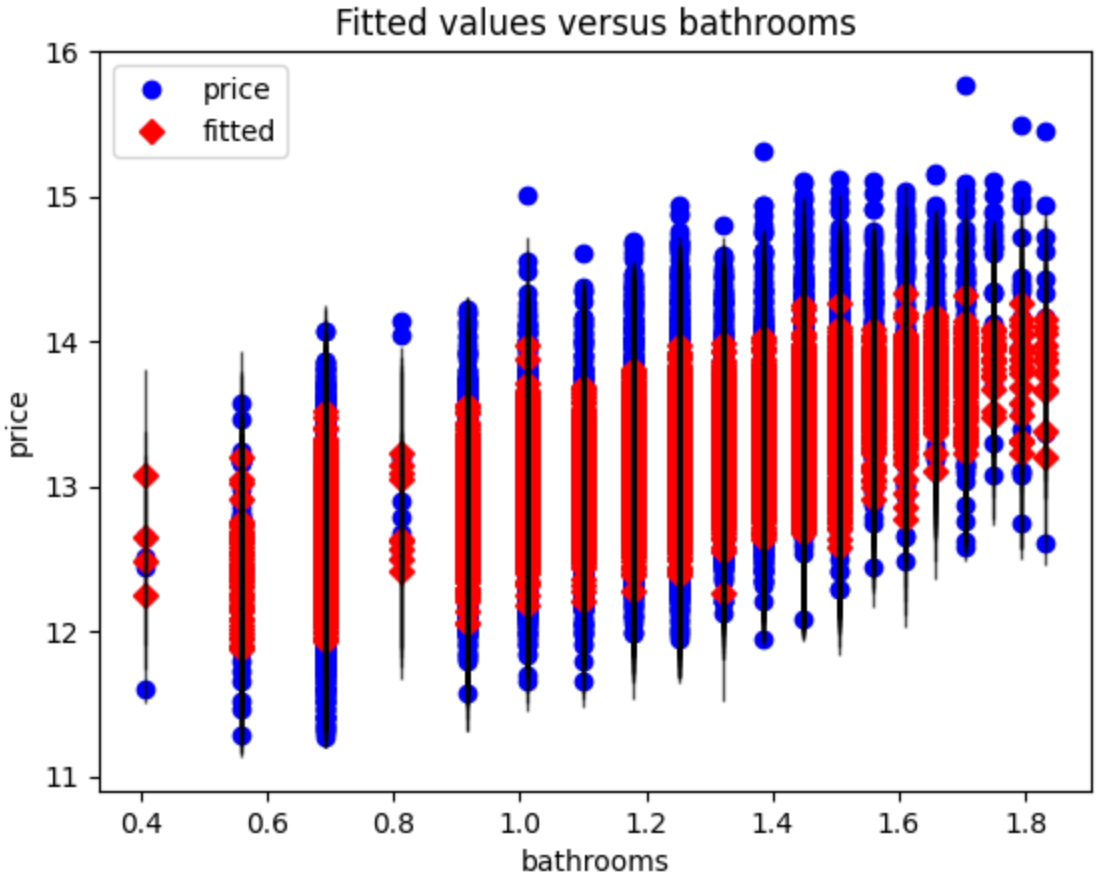
In [135]:
```
sm.graphics.plot_fit(second_modelresults, "sqft_living")
plt.show()
```



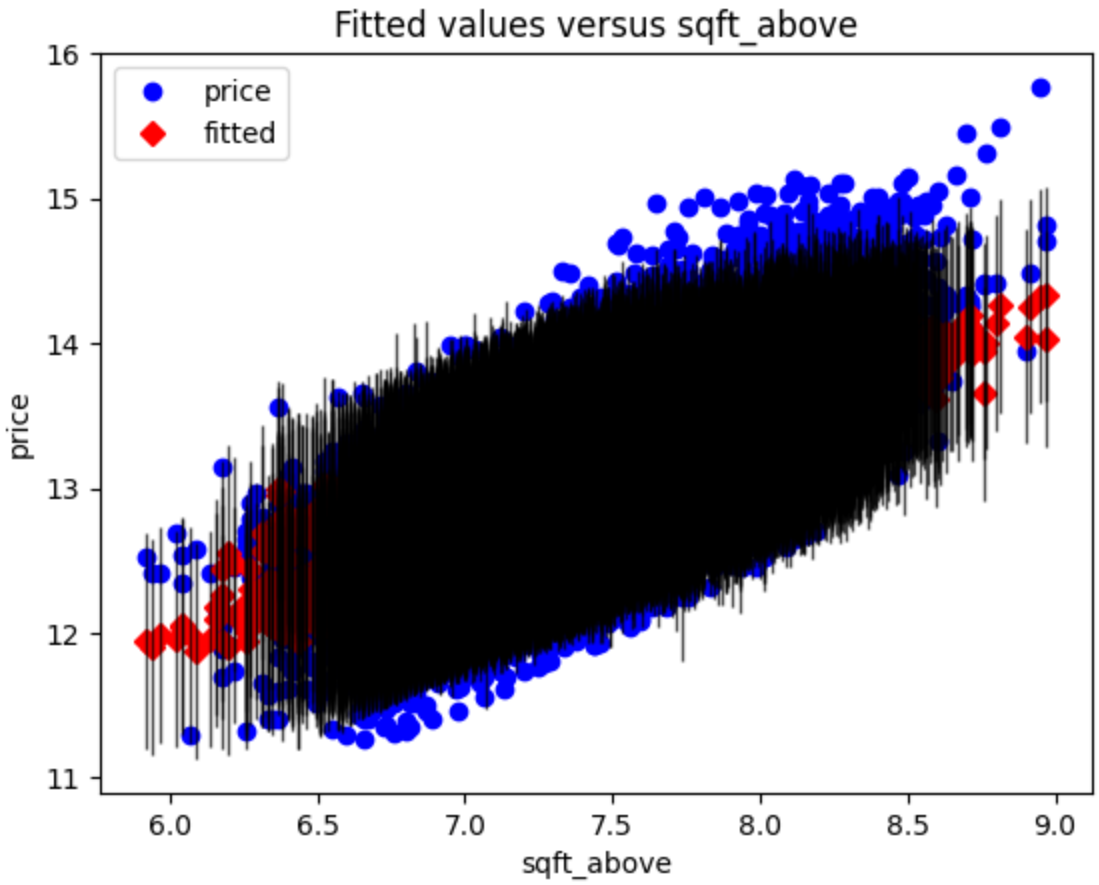
This shows the true (blue) vs. predicted (red) values, with the particular predictor (in this case, sqft_living) along the x-axis. Note that unlike with a simple regression, the red dots don't appear in a perfectly straight line. This is because their predictions are made based on the entire model, not just this predictor.

b.**Fit Model for Bathrooms, Sqft_above, and Sqft_living15**

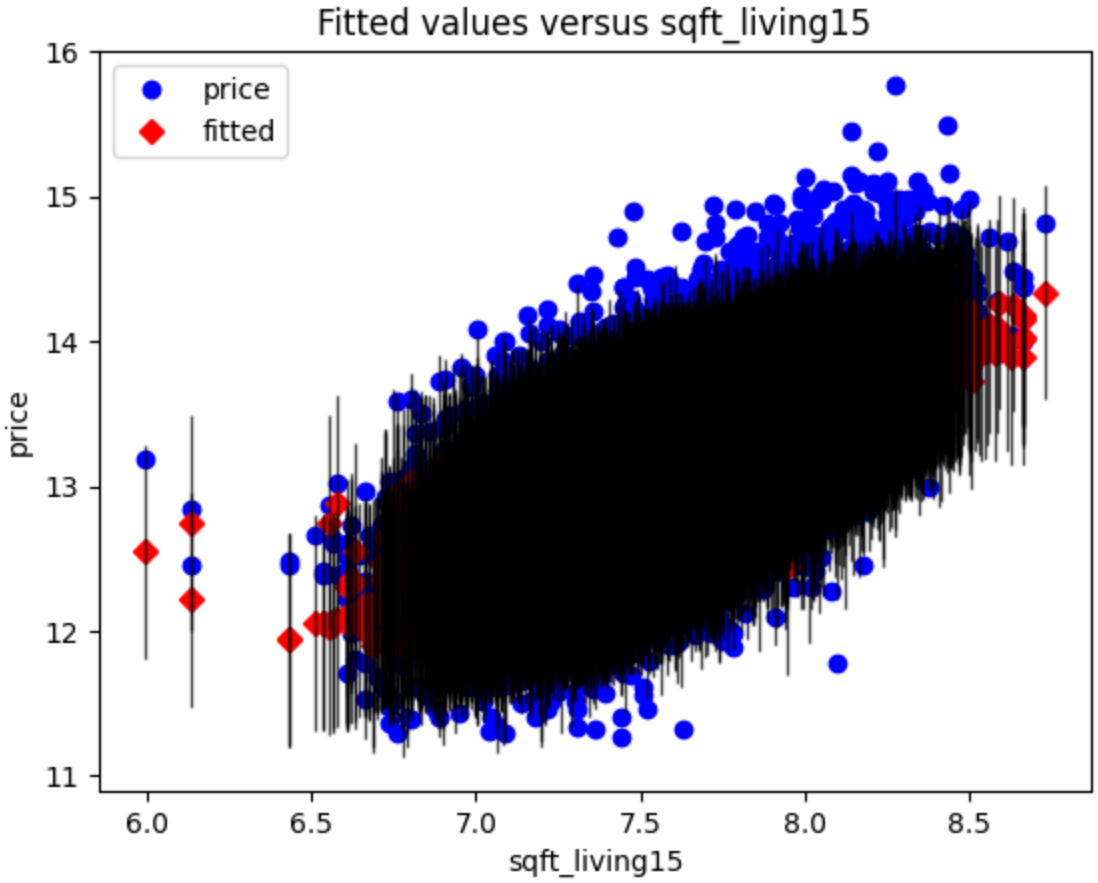
In [136]:
```
#Bathroom
sm.graphics.plot_fit(second_modelresults, "bathrooms")
plt.show()
```



In [137]:
```
#For Sqft_above
sm.graphics.plot_fit(second_modelresults, "sqft_above")
plt.show()
```



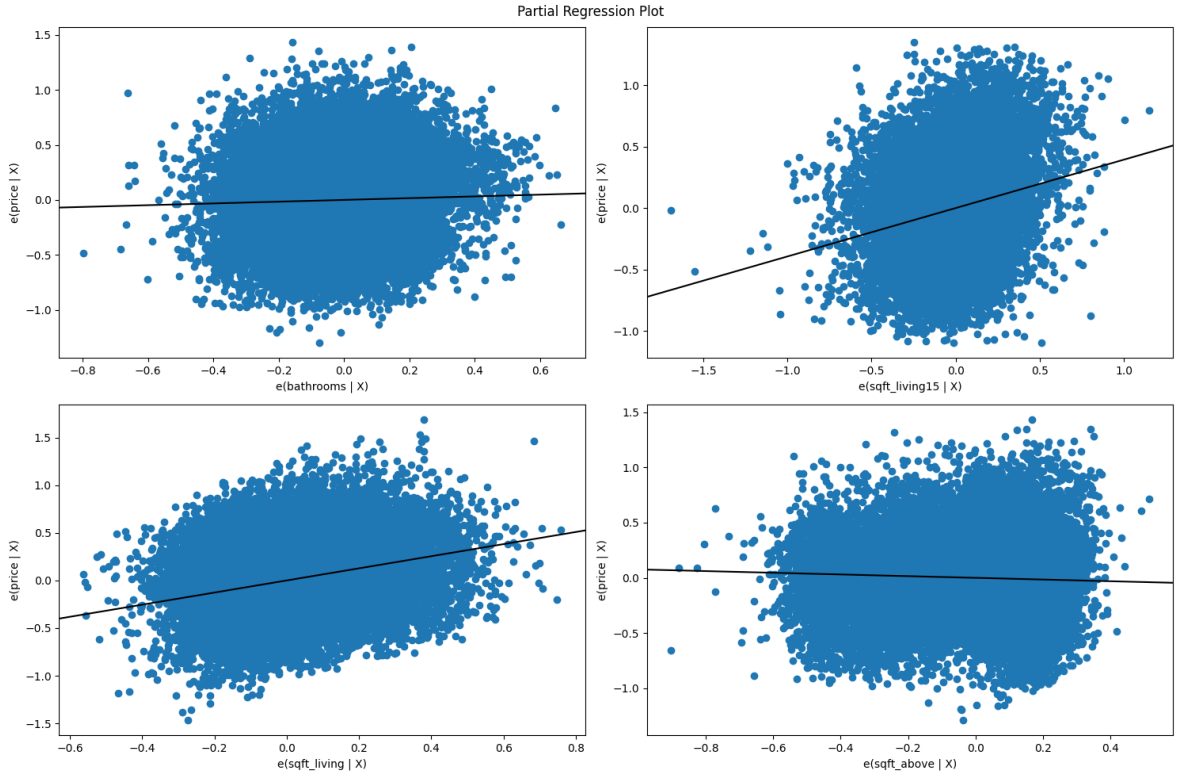
In [138]:
```
#For sqft_living15
sm.graphics.plot_fit(second_modelresults, "sqft_living15")
plt.show()
```



C. **Partial Regression Plot**

Then, instead of a basic scatter plot with a best-fit line (since our model is now higher-dimensional), we'll use two partial regression plots, one for each of our predictors.

In [139]:
```
fig = plt.figure(figsize=(15,10))
sm.graphics.plot_partregress_grid(second_modelresults,
                                  exog_idx=["bathrooms", "sqft_living15","sqf
                                  fig=fig)
plt.tight_layout()
plt.show()
```

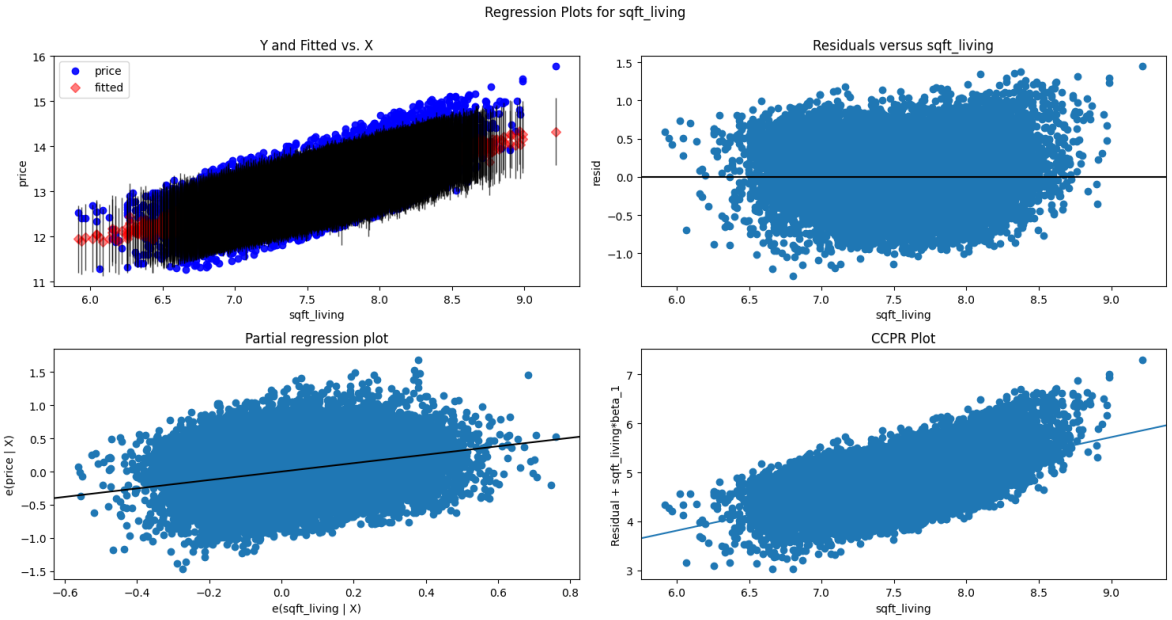


Plot shows a linear relationship with a non-zero slope, it is therefore beneficial to add sqft_living to the model, vs. having a model without sqft_living (i.e. a model with just an intercept and the other variables).

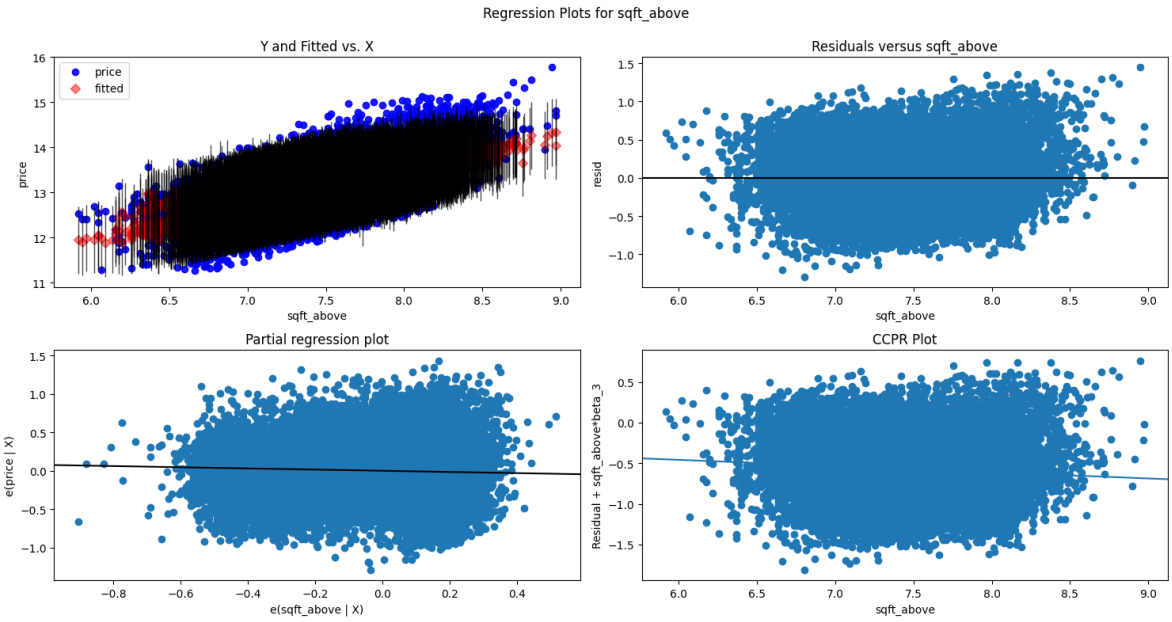
It can be deduced that these predictors are useful and should be included in the model. However, bathroom and sqft_living can be ignored since their slopes are nearly zeros slopes

d.**Residual Plots combination other plots**

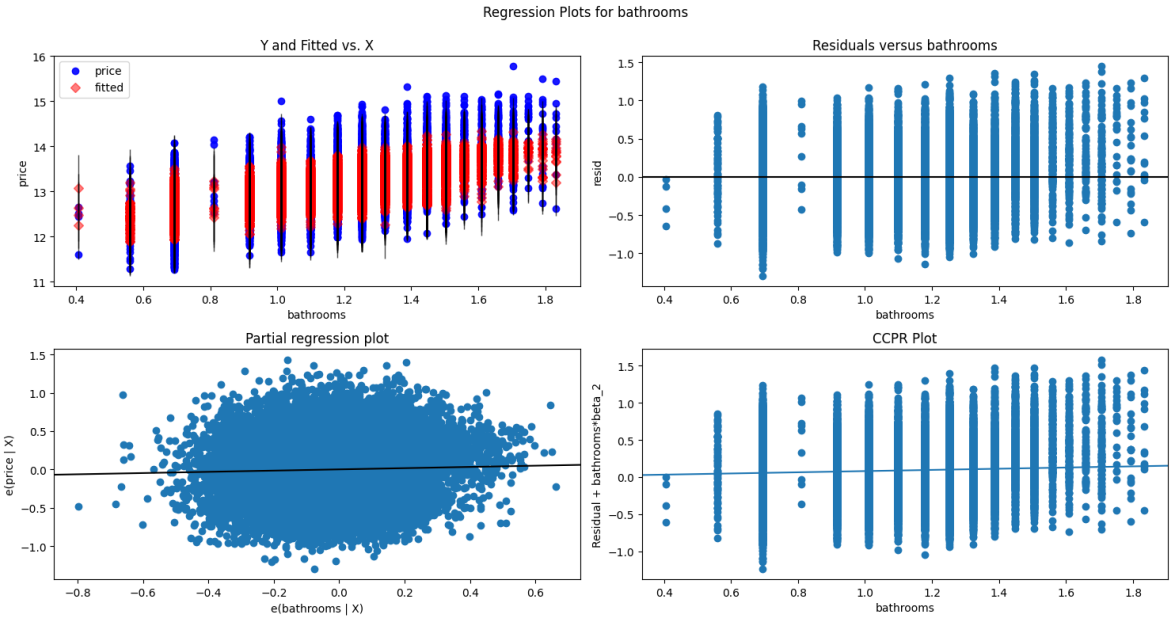
In [140]:
```
fig = plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(second_modelresults, "sqft_living", fig=fig)
plt.show()
```



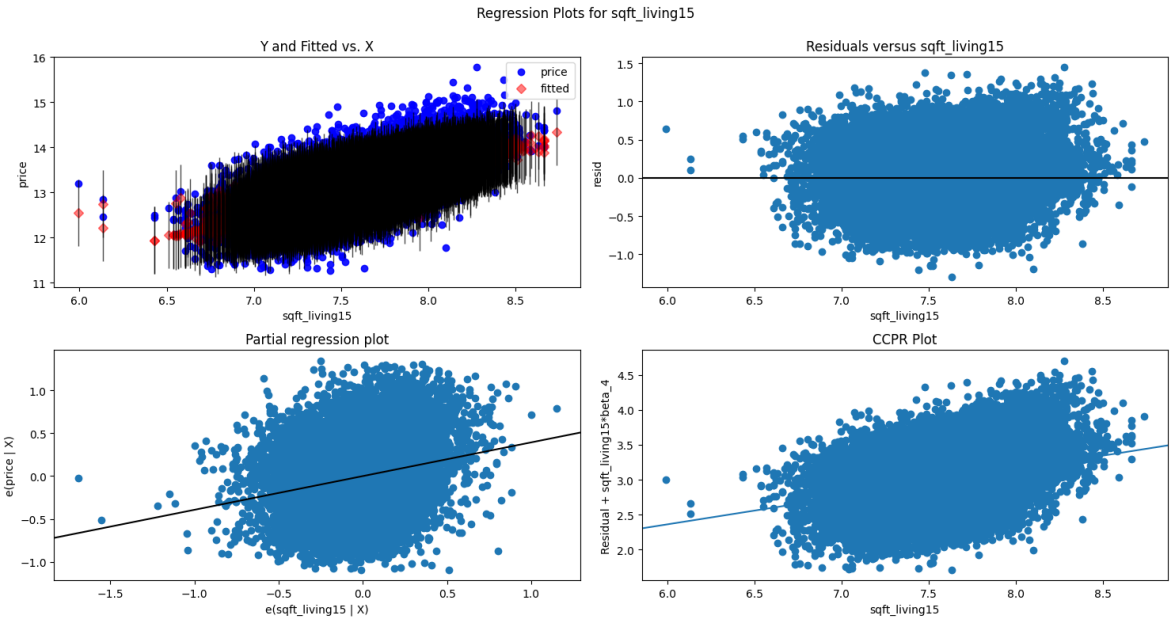
In [141]:
```
fig = plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(second_modelresults, "sqft_above", fig=fig)
plt.show()
```



In [142]:
```
fig = plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(second_modelresults, "bathrooms", fig=fig)
plt.show()
```



In [143]:
```
fig = plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(second_modelresults, "sqft_living15", fig=fig)
plt.show()
```



Model Evalution

In [144]:
```
mae = second_modelresults.resid.abs().sum() / len(y)
mae
```

Out[144]: 0.3067996070529389

Model is off by 0.306k price for a given prediction

Model Three: Multiple Regression with Many Features

In [145]:
```
# dropping price because this is our target,
#then only selecting numeric features
X_all = df.drop(["price"], axis=1).select_dtypes("number")
X_all.head(5)
```

Out[145]:

	bathrooms	sqft_living	sqft_above	sqft_living15
0	0.693147	7.074117	7.074117	7.201171
1	1.178655	7.852050	7.682943	7.433075
2	0.693147	6.647688	6.647688	7.908755
3	1.386294	7.581210	6.957497	7.215975
4	1.098612	7.427144	7.427144	7.496097

In [146]:
```
third_model = sm.OLS(y, sm.add_constant(X_all))
third_results = third_model.fit()

print(third_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.476
Model:                            OLS   Adj. R-squared:                  0.476
Method:                 Least Squares   F-statistic:                     4903.
Date:                Thu, 04 Jan 2024   Prob (F-statistic):               0.00
Time:                        06:53:25   Log-Likelihood:                -96404.
No. Observations:               21567   AIC:                         1.930e+05
Df Residuals:                   21562   BIC:                         1.934e+05
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          5.7587      0.070     81.685      0.000       5.621       5.897
bathrooms      0.0799      0.016      4.972      0.000       0.048       0.111
sqft_living    0.6355      0.014     43.829      0.000       0.607       0.664
sqft_above    -0.0764      0.012     -6.194      0.000      -0.101      -0.052
sqft_living15  0.3935      0.012     32.495      0.000       0.370       0.417
==============================================================================
Omnibus:                      142.903   Durbin-Watson:                   1.978
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              121.225
Skew:                           0.125   Prob(JB):                     4.75e-27
Kurtosis:                       2.731   Cond. No.                         362.
==============================================================================

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

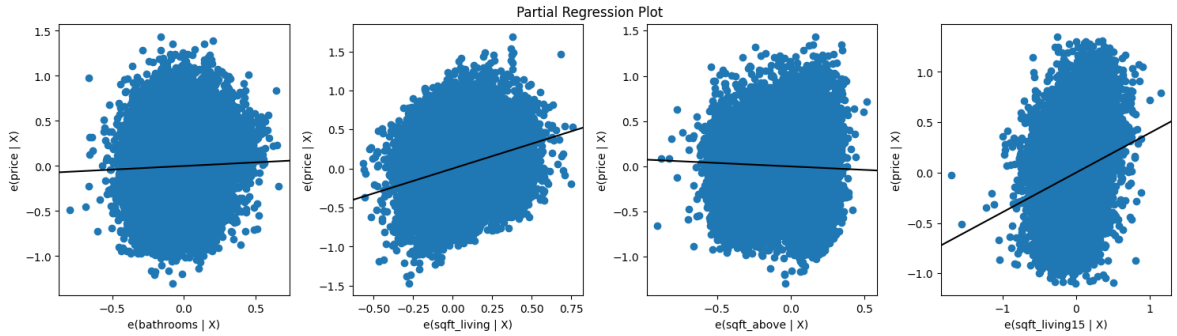
Results and Interpretations

The model is:

price = 9.9275 − 0.0510bedrooms - 0.0022bathrooms + 0.0771sqft_living - 0.0003sqft_lot + 0.1141floors - 0.1283sqft_above + 0.00007104sqft_basement + 0.3745sqft_living15- 0.000001045sqft_lot15 + 0.003total_space
*The model is statistically significant overall, with an F-statistic p-value well below 0.05
*The model explains about 51.8% of the variance in price
*Using multiple predictors increased the value of R-Squared by 15.11%, this makes this model fit and suitable for use
*Only some of the model coefficients are statistically significant while others statistically insignificant
*Bedrooms, sqft_lot, floors, sqft_above, sqft_basement, sqft_living15, sqft_lot15, and total_space p-values below 0.05 and are therefore statistically significant
*Bathrooms and sqft_living have p-values above 0.05. This means that there is greater than a 1 in 20 chance that their true coefficients are 0 (i.e. they have no effect on price), and are thus not statistically significant at an alpha of 0.05

Visualization using Partial regression plots

In [147]:
```
fig = plt.figure(figsize=(15,8))
sm.graphics.plot_partregress_grid(third_results,
                                  exog_idx=list(X_all.columns.values),
                                  grid=(2,4),
                                  fig=fig)

plt.show()
```



Model Evaluation

In [154]:
```
mae = third_results.resid.abs().sum() / len(y)
mae
```

Out[154]:
```
0.3067996070529391
```

model is off by 0.30K dollars in a given prediction

model value is smaller and near zero making it the best model

Modeling with categorical variables

Model Four

Model with multiple variables and waterfront being categorical variable

In [149]:
```
# Model with multiple independent  variables and waterfront being categorical
df['waterfront'].value_counts(ascending=True)
```

Out[149]:
```
YES      143
NO     21424
Name: waterfront, dtype: int64
```

In [150]:
```
# choosing my refence category based on the model to avoid dummy variable trap
X_waterfront_model = df[['sqft_living','bathrooms','sqft_above','sqft_living15
# origin is categorical and needs to be numeric to run regression
X_waterfront_model = pd.get_dummies(X_waterfront_model, columns=['waterfront'

waterfront_model = sm.OLS(y, sm.add_constant(X_waterfront_model))
waterfront_results = waterfront_model.fit()

print(waterfront_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.489
Model:                            OLS   Adj. R-squared:                  0.489
Method:                 Least Squares   F-statistic:                     4123.
Date:                Thu, 04 Jan 2024   Prob (F-statistic):               0.00
Time:                        06:53:57   Log-Likelihood:                -9383.2
No. Observations:               21567   AIC:                         1.878e+04
Df Residuals:                   21561   BIC:                         1.883e+04
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          5.8660      0.070     84.029      0.000       5.729       6.003
sqft_living    0.6225      0.014     43.423      0.000       0.594       0.651
bathrooms      0.0845      0.016      5.321      0.000       0.053       0.116
sqft_above    -0.0700      0.012     -5.742      0.000      -0.094      -0.046
sqft_living15  0.3846      0.012     32.133      0.000       0.361       0.408
waterfront_YES 0.7234      0.031     22.976      0.000       0.662       0.785
==============================================================================
Omnibus:                      154.296   Durbin-Watson:                   1.975
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              116.243
Skew:                           0.085   Prob(JB):                     5.73e-26
Kurtosis:                       2.683   Cond. No.                         36.3
==============================================================================

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

Results and Interpretations

sqft_living: For each additional square foot of living space, the estimated house price increases by 0.6225 thousand dollars.

bathrooms: For each additional bathroom, the estimated house price increases by 0.0845 thousand dollars.

sqft_above: For each additional square foot above ground, the estimated house price decreases by 0.0700 thousand dollars.

sqft_living15: For each additional square foot15 space, the estimated house price increases by 0.3846 thousand dollars.

waterfront_YES: If a house has a waterfront (compared to not having a waterfront), the estimated house price increases by 0.7234 thousand dollars.

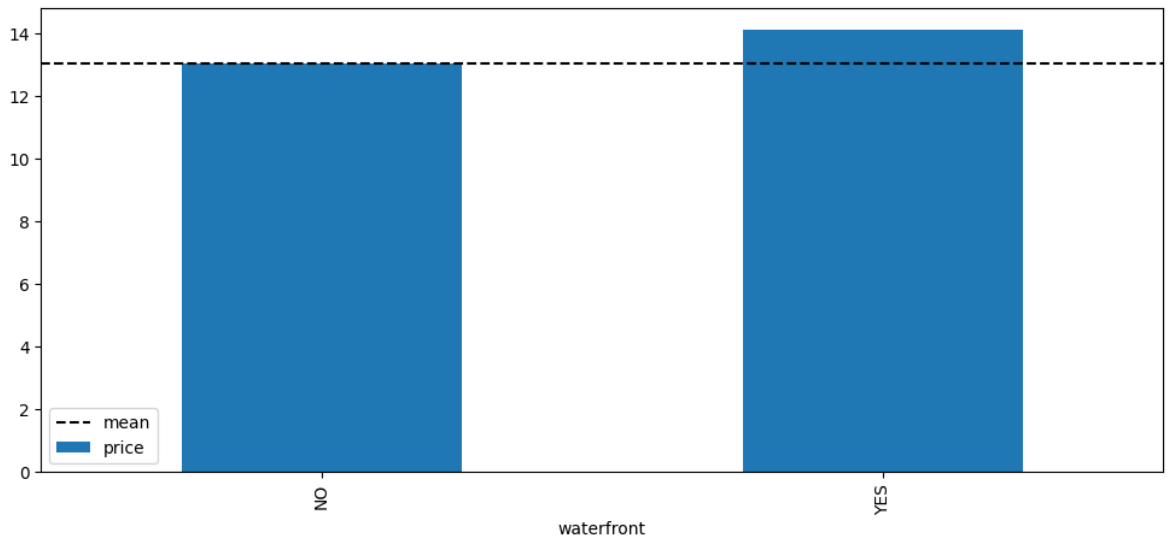
Overall Model:

The overall model has an R-squared value of 0.489, indicating that approximately 48.9% of the variability in house prices is explained by the model. Statistical Significance:

All coefficients have p-values less than 0.05, indicating that they are statistically significant.

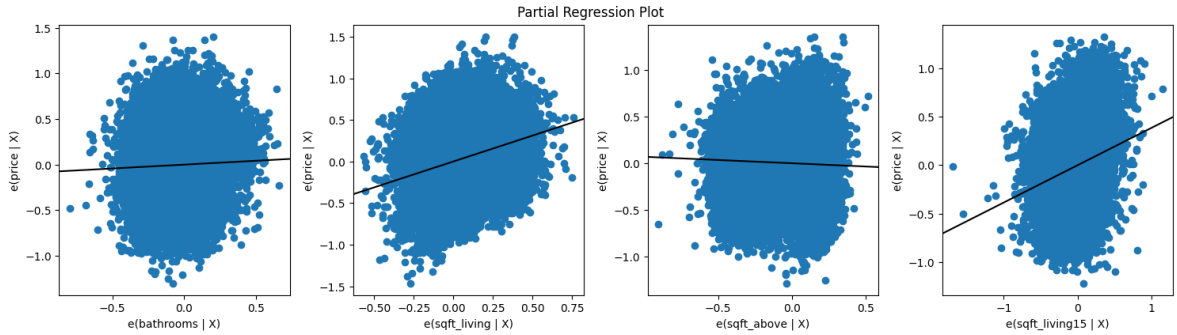
Visualization

In [151]:
```
# visualizing to check effect of waterfront on house price
fig, ax = plt.subplots(figsize=(12,5))
df.groupby("waterfront").mean('price').sort_values(by="price").plot.bar(y="pr
ax.axhline(y=df["price"].mean(), label="mean", color="black", linestyle="--")
ax.legend();
```



Visualization using Partial regression plot

In [152]:
```
fig = plt.figure(figsize=(15,8))
sm.graphics.plot_partregress_grid(waterfront_results,
                                  exog_idx=list(X_all.columns.values),
                                  grid=(2,4),
                                  fig=fig)
plt.show()
```



**Model Evaluation using mean absolute error

In [153]:
```
mae = waterfront_results.resid.abs().sum() / len(y)
mae
```

Out[153]:
```
0.3039494264861998
```

model is off by 0.30 thousand dollars in a given prediction

Model Five

Model with multiple independent variables and view being: categorical variable

In [155]:
```
# Choosing my reference category based on the one which is most common
df['view'].value_counts(ascending=True)
```

Out[155]:
```
EXCELLENT      310
FAIR           329
GOOD           505
AVERAGE        953
NONE         19470
Name: view, dtype: int64
```

In [156]:

```
X_view_model = df[['sqft_living', 'bathrooms', 'sqft_above', 'sqft_living15',

X_view_model = pd.get_dummies(X_view_model, columns=['view'], dtype=int)
#Drop one of the dummy variable columns ('view_NONE')

X_view_model = X_view_model.drop("view_NONE", axis=1)

X_view_model = sm.add_constant(X_view_model)

y = df['price']

View_model = sm.OLS(y, X_view_model)

View_results = View_model.fit()

print(View_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.506
Model:                            OLS   Adj. R-squared:                  0.506
Method:                 Least Squares   F-statistic:                     2762.
Date:                Thu, 04 Jan 2024   Prob (F-statistic):               0.00
Time:                        06:55:29   Log-Likelihood:                -9010.2
No. Observations:               21567   AIC:                         1.804e+04
Df Residuals:                   21558   BIC:                         1.811e+04
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          6.3165      0.070     89.833      0.000       6.179       6.454
sqft_living    0.5606      0.014     39.339      0.000       0.533       0.589
bathrooms      0.0949      0.016      6.080      0.000       0.064       0.126
sqft_above    -0.0152      0.012     -1.256      0.209      -0.039       0.009
sqft_living15  0.3286      0.012     27.579      0.000       0.305       0.352
view_AVERAGE   0.2063      0.012     16.634      0.000       0.182       0.231
view_EXCELLENT 0.5959      0.021     27.914      0.000       0.554       0.638
view_FAIR      0.2479      0.021     12.034      0.000       0.207       0.288
view_GOOD      0.2797      0.017     16.581      0.000       0.247       0.313
==============================================================================
Omnibus:                      156.856   Durbin-Watson:                   1.971
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              112.248
Skew:                           0.061   Prob(JB):                     4.22e-25
Kurtosis:                       2.668   Cond. No.                         373.
==============================================================================

Notes:
```

```
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
```

Results and Interpretation

sqft_living: For each additional square foot of living space, the estimated house price increases by 0.5606 thousand dollars.

bathrooms: For each additional bathroom, the estimated house price increases by 0.0949 thousand dollars.

sqft_above: For each additional square foot above ground, the estimated house price decreases by 0.0152 thousand dollars.

sqft_living15: For each additional square foot15, the estimated house price increases by 0.3286 thousand dollars.

Excellent_view: If a house has an Excellent view (compared to not having one, which is represented by the dropped category 'view_None'), the estimated house price increases by 0.5959 thousand dollars.

Average_view: If a house has an Average view (compared to not having one), the estimated house price increases by 0.2063 thousand dollars.

Fair_view: If a house has a Fair view (compared to not having one), the estimated house price increases by 0.2479 thousand dollars.

Good_view: If a house has an Good view (compared to not having one), the estimated house price increases by 0.2797 thousand dollars.

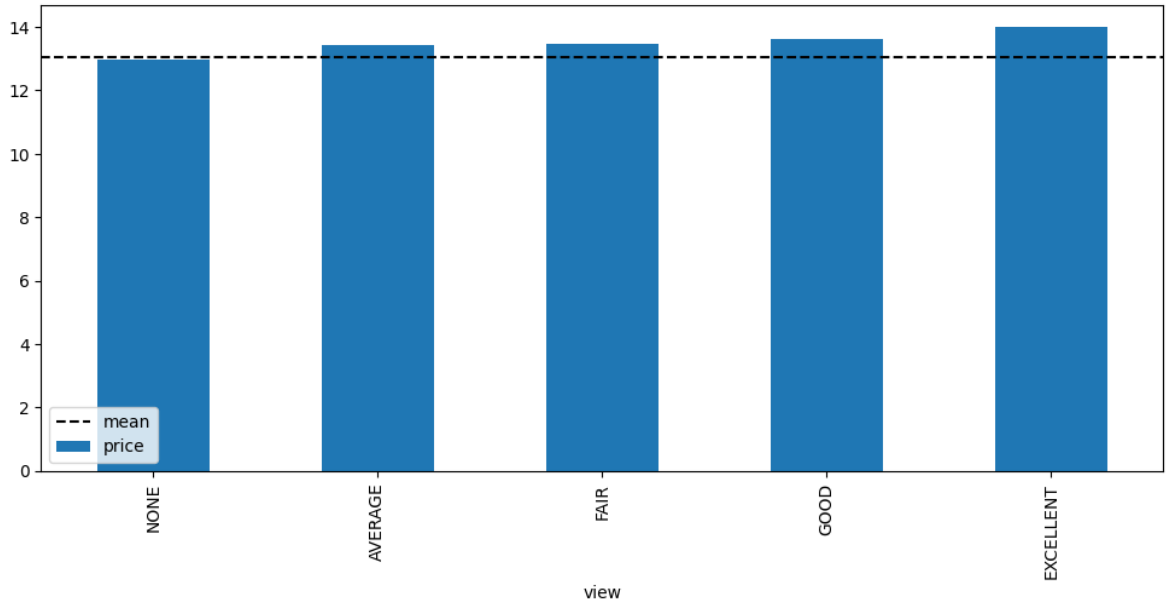
Overall Model:

The overall model has an R-squared value of 0.506, indicating that approximately 50.6% of the variability in house prices is explained by the model. Statistical Significance:

All coefficients have p-values less than 0.05, indicating that they are statistically significant expect for sqft_above

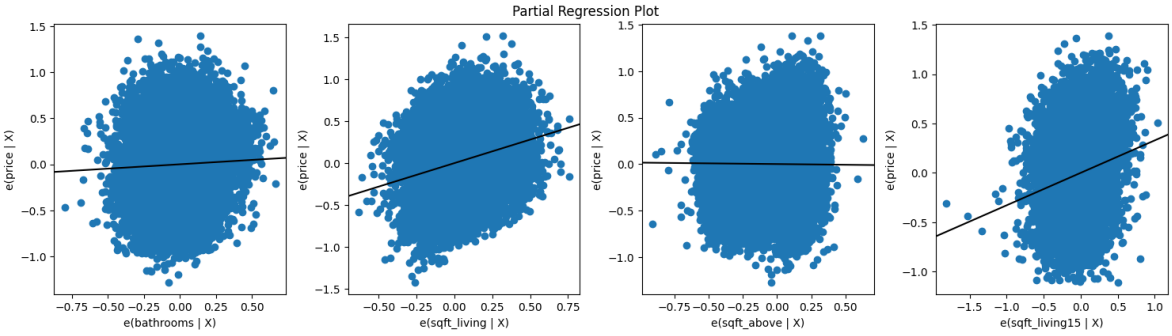
Visualization

In [ ]:
```
# visualizing to check relationship between different view and house price
fig, ax = plt.subplots(figsize=(12,5))
df.groupby("view").mean('price').sort_values(by="price").plot.bar(y="price", a
ax.axhline(y=df["price"].mean(), label="mean", color="black", linestyle="--")
ax.legend();
```



Visualization using Partial regression plot

In [157]:
```
fig = plt.figure(figsize=(15,8))
sm.graphics.plot_partregress_grid(View_results,
                                  exog_idx=list(X_all.columns.values),
                                  grid=(2,4),
                                  fig=fig)
plt.show()
```



Model Evaluation using mean absolute error

In [158]:
```
mae = View_results.resid.abs().sum() / len(y)
mae
```

Out[158]:
```
0.29949147888322947
```

The model is off by 0.30 thousand dollars in a given prediction

Conclusion

On model Performance: The regression models do exhibit moderate to good performance, with R-squared values ranging from 0.45 to 0.50. This indicates the ability of the models to explain a significant portion of the variance in house prices.

Sqft_living : In all five models, 'sqft_living' consistently emerges as a vital predictor with a strong positive impact on house prices. All the five models do show a significant and consistent relationship, suggesting that, on average, an increase in the living area ('sqft_living') corresponds to a higher property price. This clearly explains the crucial role of living space in determining house values, providing valuable guidance for Every Door Real Estate agents to help clients understand the influential factor of living area on property prices.

Recommendations

Feature Importance: Emphasize to Every Door's agents the importance of 'sqft_living' and other significant predictors in influencing house prices. Provide guidance on how to use these insights to better inform clients.

Client Communication: Train agents to effectively communicate the model's findings to clients, emphasizing the key factors influencing house prices and helping clients gauge the financial commitment required for their desired properties