Final Project Submission

Please fill out:

- Student pace: part time
- Student names:
  - Fredrick Kyeki (https://github.com/FREDRICKKYEKI) (scrum master)
  - Ivy Mwanza (https://github.com/mwanza00)
  - Dennis Kobia (https://github.com/cpakobia)
  - Ben Ochoro (https://github.com/Be-hub-afk)

House Sales in King County, Washington, USA

Overview

This project focuses on analyzing housing sales data in King County, Northwest USA, using multiple linear regression modeling. The goal of this project is to provide valuable insights to homeowners, real estate agencies, and other stakeholders regarding factors that influence house prices and to make data-driven recommendations.

Project Goal

The primary goal of this project is to perform a comprehensive analysis of house sales data in King County, Northwest USA, using multiple linear regression modeling techniques. This analysis aims to provide valuable insights into the factors that influence house prices in the region and make data-driven recommendations for homeowners, real estate agencies, and other stakeholders.

Stake Holders

The intended audience for this project includes:

- **Homeowners**: Individuals looking to enhance the value of their properties through informed decision-making regarding renovations and improvements.
- **Real Estate Agencies**: Companies and agents seeking data-driven insights to assist their clients in buying and selling homes effectively.
- **Data Science Professionals**: Professionals in the field of data science and analytics interested in understanding how regression modeling can be applied to real-world business problems.

1. Business Understanding

The central business problem addressed in this project revolves around the real estate market in King County, Northwest USA. The primary stakeholders include homeowners, real estate agencies, and data science professionals who seek to gain insights into the factors influencing house prices and make data-driven decisions in this dynamic market.

2. Data Understanding

We used data sourced from King County Housing Dataset CSV. The data represents houses with information on price, bedrooms, bathrooms, sqft living, sqft lot, floors, view, aqnd year built. Total data used was from 21597 homes split 80/20 for training and testing. Variables include price, bedrooms, bathrooms, sqft living, sqft lot, floors, view, aqnd year built.

Properties of variables of interest:

1. **Price**: Continuous numeric (float). Represents the sale price of houses in the dataset.
2. **Bedrooms**: Discrete numeric (integer). Represents the number of bedrooms in each house.
3. **Bathrooms**: Discrete numeric (integer).Represents the number of bathrooms in each house.
4. **Sqft living**: Continuous numeric (integer). Represents the total square footage of the living space in each house.
5. **Floors**: Discrete Discrete (integer). Represents the number of floors in each house.
6. **View**: Categorical (object). Represents the view rating of the property.
7. **Year built**: Discrete numeric (integer). Represents the year each house was built.

3. Data Preparation

The following describes the data cleaning process to remove any inconsistencies in the data and prepare it for analysis and modeling.

1. Importing Libraries: Importing the necessary Python libraries, including Pandas, NumPy, Matplotlib, and Seaborn, which are commonly used for data manipulation and visualization.
2. Data Loading: Reading the house data from a CSV file ("kc_house_data.csv") into a Pandas DataFrame using pd.read_csv().
3. Data Cleaning: Used house _data_df.head() to inspect the first few rows of the DataFrame. Checked the shape of the DataFrame using house_data_df.shape to determine the number of rows and columns. Used house_data_df.info() to get information about the data types and missing values in each column. Checked for duplicate rows using house_data_df.duplicated().
4. Data Exploration: Created various visualizations to explore the relationships between variables, such as scatter plots and box plots, to understand how features like square footage, the number of bathrooms, bedrooms, floors, and year built relate to house prices.
5. Investigate polynomial relationships and interactions between variables in greater details.

In [662]:
```
# Your code here - remember to use markdown cells for comments as well!
# Importing the relevant libraries
import pandas as pd
import numpy as mp
import matplotlib.pyplot as plt
import seaborn as sns
```

In [663]:
```
# Reading the house data into a DataFrame
house_data_df = pd.read_csv("data/kc_house_data.csv")
```

In [664]:
```
# Exploring the structure and content of the DataFrame
house_data_df.head()
```

Out[664]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	Na
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	N
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	N
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	N
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	N

5 rows × 21 columns

In [665]:
```
to_drop = [
'date',
'sqft_above',
'sqft_basement',
'yr_renovated',
'zipcode',
'lat',
'long',
'sqft_living15',
'sqft_lot15',
]

house_data_df.drop(columns=to_drop, inplace=True)
```

In [666]:
```
# Looking at the info printout
house_data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   id           21597 non-null  int64  
 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   19221 non-null  object 
 8   view         21534 non-null  object 
 9   condition    21597 non-null  object 
 10  grade        21597 non-null  object 
 11  yr_built     21597 non-null  int64  
dtypes: float64(3), int64(5), object(4)
memory usage: 2.0+ MB
```

In [667]:
```
# Cheking whether there are missing values
house_data_df.isna().sum()
```

Out[667]:
```
id                0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront     2376
view             63
condition         0
grade             0
yr_built          0
dtype: int64
```

In [668]:
```
# Dropping null values in the 'view' column
house_data_df = house_data_df.dropna(subset=['view'])
```

In [669]:
```
# Dropping null values in the 'waterfront' column
house_data_df = house_data_df.dropna(subset=['waterfront'])
```

In [670]:
```python
# Cheking for duplicates
house_data_df.duplicated()
```

Out[670]:
```
1        False
2        False
3        False
4        False
5        False
         ...
21591    False
21592    False
21593    False
21594    False
21596    False
Length: 19164, dtype: bool
```

In [671]:
```python
# Summary statistics for the numerical columns in the DataFrame
summary_statistics = house_data_df.describe()
print(summary_statistics)
```

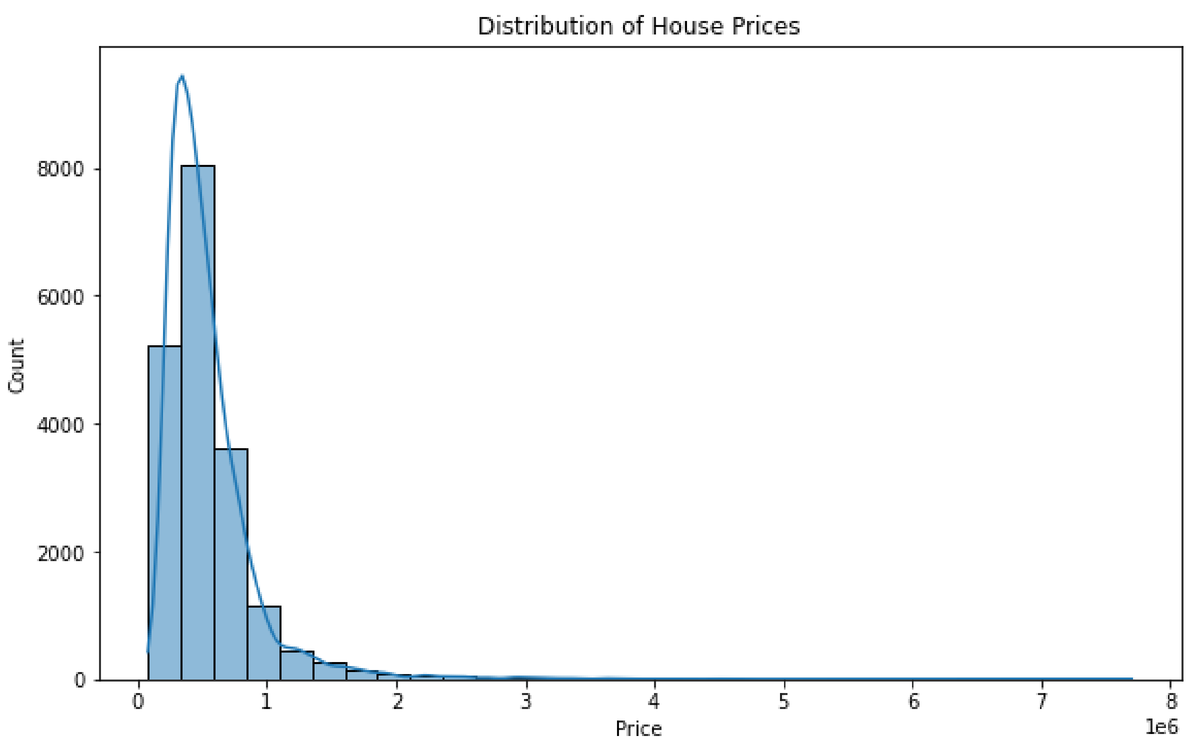
```
                 id         price      bedrooms     bathrooms   sqft_living
\
count  1.916400e+04  1.916400e+04  19164.000000  19164.000000  19164.000000
mean   4.594087e+09  5.414490e+05      3.374452      2.117029   2082.038301
std    2.876912e+09  3.709009e+05      0.928676      0.769241    921.918226
min    1.000102e+06  7.800000e+04      1.000000      0.500000    370.000000
25%    2.124077e+09  3.220000e+05      3.000000      1.750000   1430.000000
50%    3.905082e+09  4.500000e+05      3.000000      2.250000   1920.000000
75%    7.334501e+09  6.439625e+05      4.000000      2.500000   2550.000000
max    9.900000e+09  7.700000e+06     33.000000      8.000000  13540.000000

           sqft_lot        floors      yr_built
count  1.916400e+04  19164.000000  19164.000000
mean   1.506174e+04      1.495173   1971.039553
std    4.077215e+04      0.540308     29.388020
min    5.200000e+02      1.000000   1900.000000
25%    5.040000e+03      1.000000   1951.000000
50%    7.620000e+03      1.500000   1975.000000
75%    1.072000e+04      2.000000   1997.000000
max    1.651359e+06      3.500000   2015.000000
```

Price Distribution with Outliers

In [672]:
```
# Calculating summary statistics for the 'price' column.
price_summary = house_data_df['price'].describe()
print(price_summary)
```

```
count    1.916400e+04
mean     5.414490e+05
std      3.709009e+05
min      7.800000e+04
25%      3.220000e+05
50%      4.500000e+05
75%      6.439625e+05
max      7.700000e+06
Name: price, dtype: float64
```

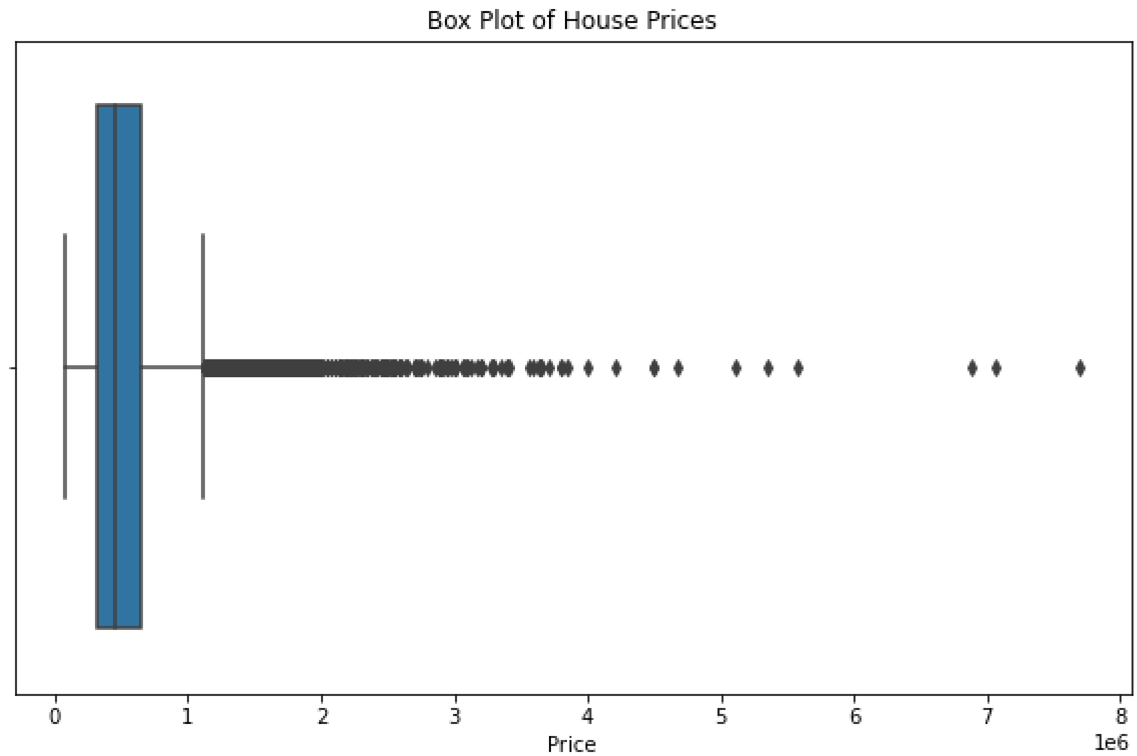
In [673]:
```
# Data Exploration
# Exploring the data by plotting some graphs to visualize the distribution of features.
# Using seaborn to create a histogram of the 'price' column.
plt.figure(figsize=(10, 6))
sns.histplot(house_data_df['price'], bins=30, kde=True)
plt.title('Distribution of House Prices')
plt.xlabel('Price')
plt.ylabel('Count')
plt.show()
```



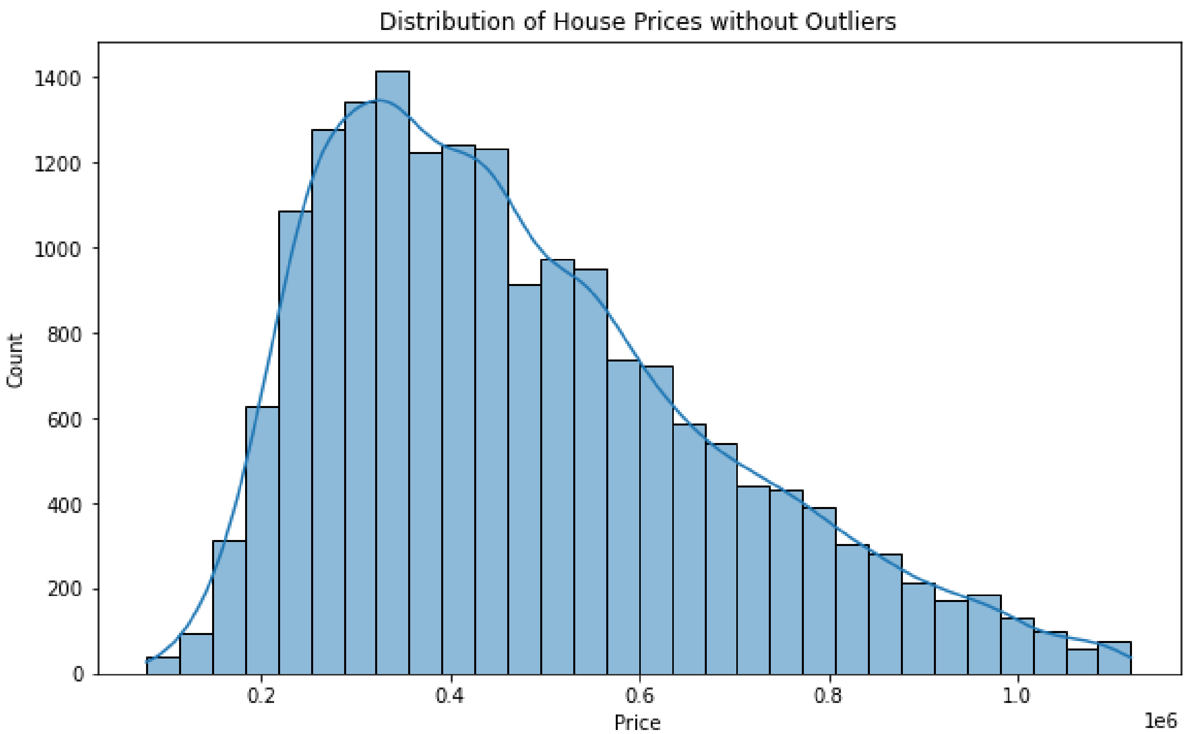
Price Distribution without Outliers

In [674]:

```
# Creating a box plot to identify outliers in the 'price' column
plt.figure(figsize=(10, 6))
sns.boxplot(x=house_data_df['price'])
plt.title('Box Plot of House Prices')
plt.xlabel('Price')
plt.show()
```



In [675]:
```
# Calculating the Interquartile Range (IQR) to identify outliers
Q1 = house_data_df['price'].quantile(0.25)
Q3 = house_data_df['price'].quantile(0.75)
IQR = Q3 - Q1

# Defining the upper and lower bounds for identifying outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filtering the DataFrame to exclude outliers
hsedata_no_outliers = house_data_df[(house_data_df['price'] >= lower_bound) &
(house_data_df['price'] <= upper_bound)]

# Creating a histogram of house prices without outliers
plt.figure(figsize=(10, 6))
sns.histplot(hsedata_no_outliers['price'], bins=30, kde=True)
plt.title('Distribution of House Prices without Outliers')
plt.xlabel('Price')
plt.ylabel('Count')
plt.show()
```



4. Modelling

The following were the steps taken to come up with the model.

4.1 Steps taken while modelling

1. Start with the data with outliers and create the baseline model.
2. Add one predictor (independent) variable.
3. Check the performance.
4. Add a categorical variable.
5. Repeat steps 2 - 4 until adequate performance is reached.
6. Repeat these steps for the data without outliers and choose the best model.

I. First Iteration (Base model)

In [676]:
```
# find the correlation matrix of the data
def highest_corr(df, resp_var, to_drop=[]):
    """finds highest corr
    Parameters:
    -----------
    df:
        DataFrame
    resp_var:
        Response variable
    to_drop:
        Columns to drop before finding the correlation matrix
    """
    df = df.drop(columns=to_drop)
    data_corr = df.corr()[resp_var]
    del data_corr[resp_var]

    max_corr = max(data_corr.values)
    print(data_corr, "\n")

    for k, v in data_corr.items():
        if v == max_corr:
            print('highest correlation: ', {k : v})

# highest correlation
highest_corr(house_data_df, 'price')
```

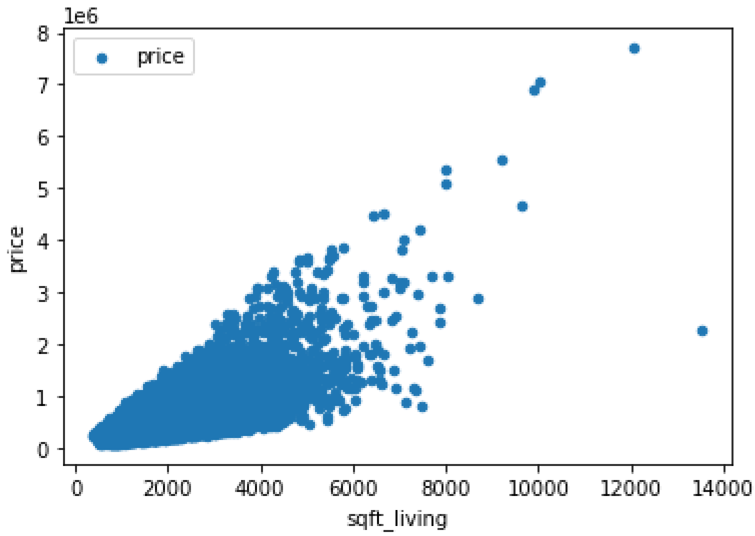
```
id            -0.018107
bedrooms       0.309057
bathrooms      0.526609
sqft_living    0.704428
sqft_lot       0.087430
floors         0.258797
yr_built       0.053433
Name: price, dtype: float64

highest correlation:  {'sqft_living': 0.7044283177851761}
```

From the above result, **sqft_living** (which represents the square foot of living area) has the highest correlation with the price (0.7044283177851761), and is perfect for the baseline model, which is a simple linear regression.

> $$\hat{y} = \hat{\beta_0} + \hat{\beta_1}x$$
>
> \ 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 $\hat{\beta_1}$ (the slope) and $\hat{\beta_0}$ (the intercept).

In [677]:
```
house_data_df.plot.scatter(x="sqft_living", y="price", label="price");
```



In [678]:
```
from statsmodels.api import OLS
import statsmodels.api as sm

# construct the exogenous and endogenous variables
# i.e dependent (price) and independent variables (sqft_living)
y = house_data_df['price']
X_baseline = house_data_df[['sqft_living']]
```

In [679]:
```
# construct the baseline model
baseline_model = sm.OLS(endog=y, exog=sm.add_constant(X_baseline))
baseline_results = sm.OLS(endog=y, exog=sm.add_constant(X_baseline)).fit()
baseline_results_summary = baseline_results.summary()
```

In [680]:
```python
# The summary of the model
print(baseline_results_summary)
```

```
                            OLS Regression Results
=============================================================================
=
Dep. Variable:                  price   R-squared:                       0.49
6
Model:                            OLS   Adj. R-squared:                  0.49
6
Method:                 Least Squares   F-statistic:                 1.887e+0
4
Date:                Sun, 10 Sep 2023   Prob (F-statistic):              0.0
0
Time:                        13:43:01   Log-Likelihood:           -2.6638e+0
5
No. Observations:               19164   AIC:                         5.328e+0
5
Df Residuals:                   19162   BIC:                         5.328e+0
5
Df Model:                           1
Covariance Type:            nonrobust
=============================================================================
==
                  coef    std err          t      P>|t|      [0.025      0.97
5]
-----------------------------------------------------------------------------
--
const        -4.86e+04   4697.110    -10.348      0.000   -5.78e+04   -3.94e+
04
sqft_living   283.4016      2.063    137.384      0.000     279.358     287.4
45
=============================================================================
=
Omnibus:                    13130.357   Durbin-Watson:                   1.98
7
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           481938.43
8
Skew:                           2.817   Prob(JB):                        0.0
0
Kurtosis:                      26.913   Cond. No.                    5.62e+0
3
=============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 5.62e+03. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

Interpretation:

Simple Linear Regression Results

Looking at the summary above, we can see that the regression line we found was

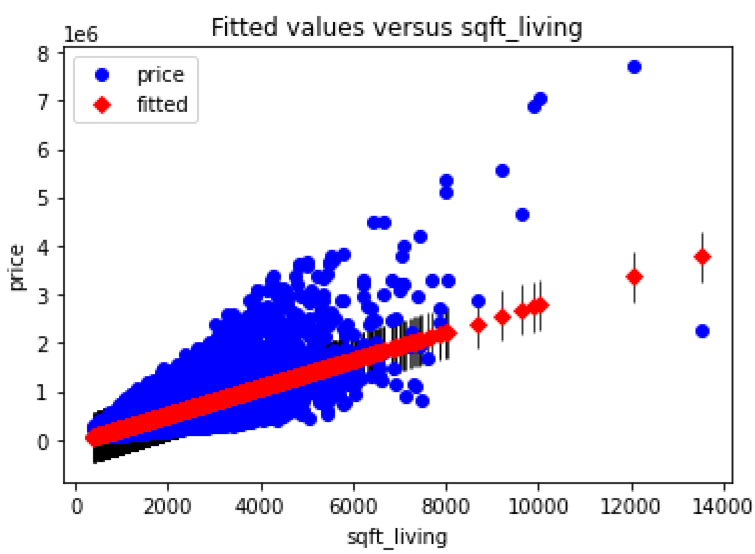
$$\hat{price} = -48,600 + 283.4016 sqft\_living$$

- The model **is statistically significant** overall, with an F-statistic p-value well below 0.05
- The model explains about 49.6% of the variance in price.
  - indicating that approximately **49.6% of the variance in house prices is explained by the square footage of living space ("sqft_living").**
- The model coefficients (`const` and `sqft_living`) are both statistically significant, with t-statistic p-values well below 0.05
- If the sqft_living is $0 ft^2$, we would expect price to be about $\$-48,600$
- For each increase of 1 square foot of living area, we see an associated increase in price of about $\$283.4016$

Simple Linear Regression Visualization

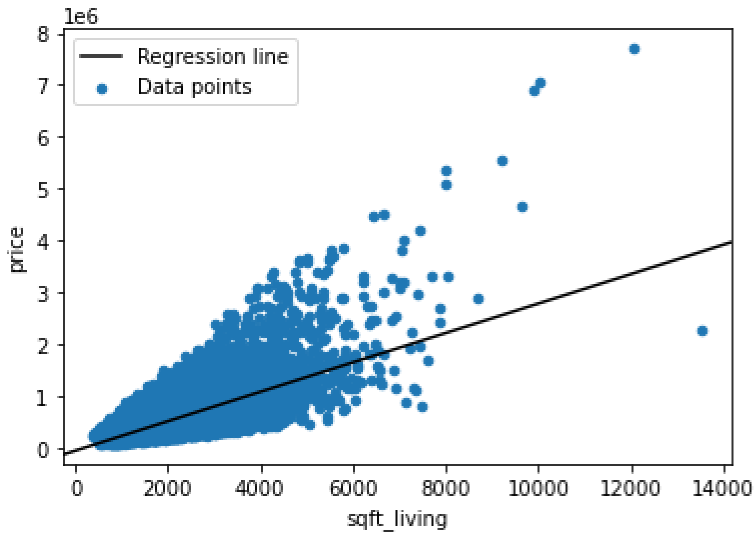
We'll also plot the actual vs. predicted values:

In [681]:
```
sm.graphics.plot_fit(baseline_results, "sqft_living")
plt.show()
```

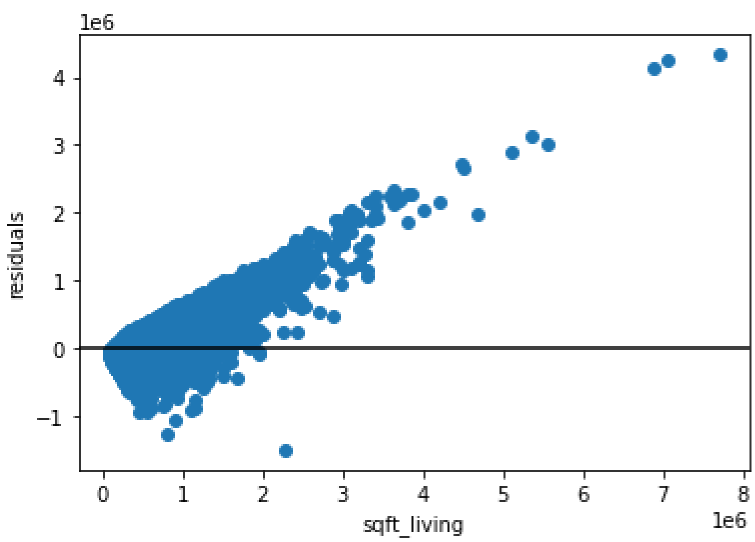


The regression line:

In [682]:
```
fig, ax = plt.subplots()
house_data_df.plot.scatter(x="sqft_living", y="price", label="Data points", ax
=ax)
sm.graphics.abline_plot(model_results=baseline_results, label="Regression lin
e", ax=ax, color="black")
ax.legend();
```



In [683]:
```
fig, ax = plt.subplots()

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



II. Second iteration

Adding Another Independent Variable

Now, we expand from our simple linear regression to a multiple linear regression, in bid to improve the overall model performance. Let's try find the next highly correlated variable to price after sqft_living to use in our next iteration.

But before that, we create a python **class** to avoid unnecessary repetitions of code so as to make the model iterations efficient and readable:

In [684]:

```
class RegressionAnalysis:
    """Regression class"""
    def __init__(self, df, y, X):
        """
        constructor function
        parameters:
        -----------
        df(pandas.DataFrame):
            DataFrame containing our data
        y (pandas.Series):
            endogenous (dependent) variable
        X (pandas.DataFrame):
            exogenous (independent) variable(s)
        """
        self.df = df
        self.X = X
        self.y = y

        self.model = sm.OLS(endog=y, exog=sm.add_constant(X))
        self.results = sm.OLS(endog=y, exog=sm.add_constant(X)).fit()

    def summary(self):
        """returns model summary"""
        return self.results.summary()

    def plot_fit(self, var=None):
        """Plot fit against one regressor."""
        if not var:
            var = self.X.columns[0]
        sm.graphics.plot_fit(self.results, var)
        plt.show()

    def plot_partial_reg(self, figsize=(15,8)):
        """Plot partial regression for a set of regressors."""
        fig = plt.figure(figsize=figsize)
        sm.graphics.plot_partregress_grid(self.results, exog_idx=list(self.X.c
olumns), fig=fig)
        plt.tight_layout()
        plt.show()

    def plot_ccpr(self):
        """Generate CCPR plots against a set of regressors, plot in a grid."""
        fig = plt.figure(figsize=(15,5))
        sm.graphics.plot_ccpr_grid(self.results, exog_idx=list(self.X.column
s), grid=(1,2), fig=fig)
        plt.tight_layout()
        plt.show()
```

In [685]:
```
# next highly correlated variable
highest_corr(house_data_df, 'price', ['sqft_living'])
```

```
id           -0.018107
bedrooms      0.309057
bathrooms     0.526609
sqft_lot      0.087430
floors        0.258797
yr_built      0.053433
Name: price, dtype: float64

highest correlation:  {'bathrooms': 0.5266090477103713}
```

It looks like the number of `bathrooms` is the next most strongly *positively* correlated predictor, so let's use that.

In [686]:
```
indep_vars = ['sqft_living', 'bathrooms']
X_second = house_data_df[indep_vars]
X_second.head()
```

Out[686]:

	sqft_living	bathrooms
1	2570	2.25
2	770	1.00
3	1960	3.00
4	1680	2.00
5	5420	4.50

In [687]:
```
# check bathroom values before analysis
house_data_df['bathrooms'].unique()
```

Out[687]:
```
array([2.25, 1.  , 3.  , 2.  , 4.5 , 2.5 , 1.75, 2.75, 1.5 , 3.25, 4.  ,
       3.5 , 0.75, 4.75, 5.  , 4.25, 3.75, 1.25, 5.25, 0.5 , 5.5 , 6.75,
       6.  , 5.75, 8.  , 7.5 , 7.75, 6.25, 6.5 ])
```

Since the number of bathrooms should be discrete numerial value (i.e. we cannot have 2.75 bathrooms), we have to floor the values to a discrete value before any sort of analysis on the variable.

In [688]:
```
house_data_df['bathrooms'] = house_data_df['bathrooms'].apply(int)
house_data_df['bathrooms'].apply(int).unique()
```

Out[688]:
```
array([2, 1, 3, 4, 0, 5, 6, 8, 7], dtype=int64)
```

In [689]:
```
# reselect the variables
indep_vars = ['sqft_living', 'bathrooms']
X_second = house_data_df[indep_vars]

second_iteration = RegressionAnalysis(house_data_df, y, X_second)

# let's see the summary
print(second_iteration.summary())
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  price   R-squared:                       0.49
7
Model:                            OLS   Adj. R-squared:                  0.49
7
Method:                 Least Squares   F-statistic:                     946
4.
Date:                Sun, 10 Sep 2023   Prob (F-statistic):               0.0
0
Time:                        13:43:06   Log-Likelihood:           -2.6636e+0
5
No. Observations:               19164   AIC:                       5.327e+0
5
Df Residuals:                   19161   BIC:                       5.328e+0
5
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
==
                  coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
const       -5.989e+04   5157.920    -11.611      0.000      -7e+04   -4.98e+
04
sqft_living   272.7382      2.887     94.482      0.000     267.080     278.3
96
bathrooms    1.912e+04   3624.227      5.277      0.000     1.2e+04    2.62e+
04
==============================================================================
=
Omnibus:                    13143.114   Durbin-Watson:                   1.98
7
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           484508.14
7
Skew:                           2.819   Prob(JB):                         0.0
0
Kurtosis:                      26.979   Cond. No.                     6.58e+0
3
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 6.58e+03. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

Interpretation:

Second Iteration Regression Results

Looking at the summary above, we can see that the regression line we found was

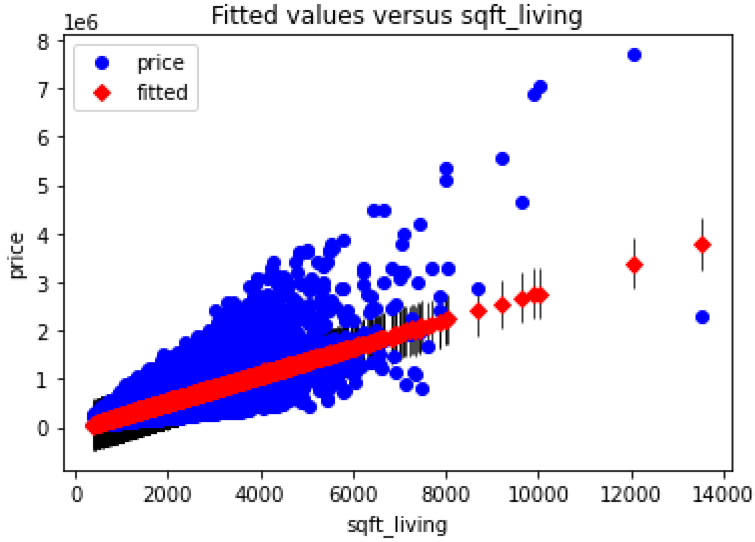
$$\hat{price} = -59,890 + 272.7382 sqft\_living + 19,120 bathrooms$$

- The model **is statistically significant** overall, with an **F-statistic p-value** well below 0.05
- The model exhibits an **R-squared value of 49.7%**:
  - indicating that approximately **49.7% of the variance in house prices is explained by the square footage of living space ("sqft_living") and the number of bathrooms in the houses**
  - slight improvement of 0.1%.
- The model coefficients (`const` , `sqft_living and bathrooms`) are statistically significant, with t-statistic p-values well below 0.05
- For each increase of 1 square foot of living area, we see an associated increase in price of about $\$272.7382$
  - this here is an decrease of -10.66 from the last model, which means that number of bathrooms was not meaningfully confounding in the relationship between sqft_living and price.
- For each increase of 1 bathroom, we see an associated decrease in price of about $\$19,120$
- The model predicts a price of $\$-59,890$ when `sqft_living` and number of `bathrooms` are 0.

Second Iteration Regression Visualization

i) Model Fit

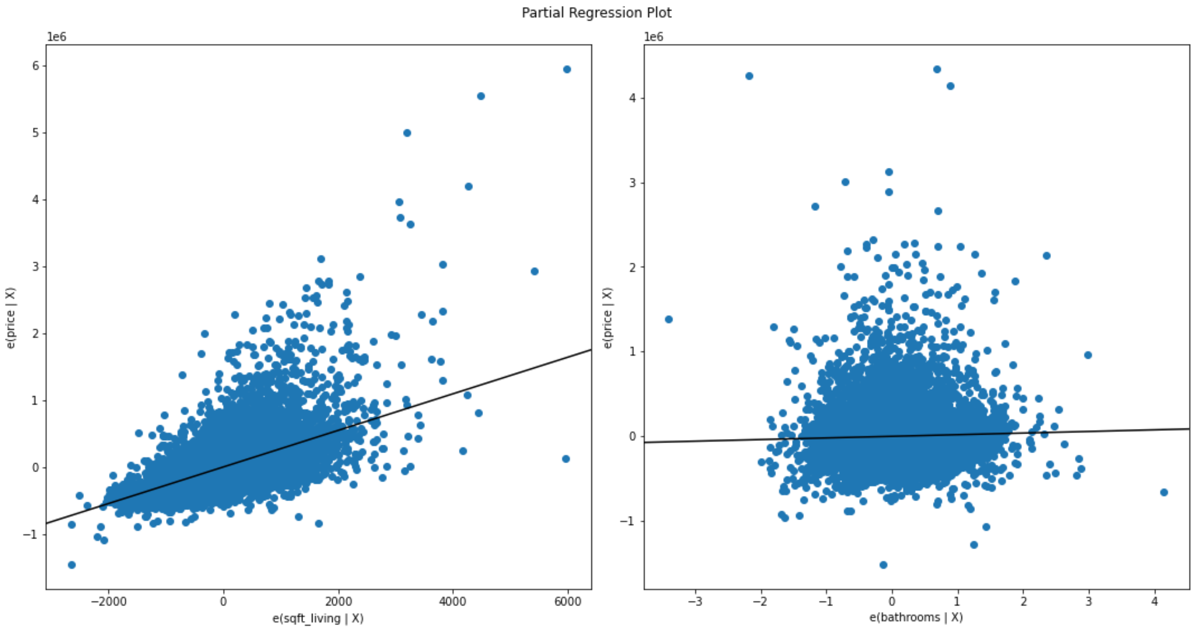
In [690]:
```
# plot model fit for sqft_living and bathrooms
second_iteration.plot_fit('sqft_living')
second_iteration.plot_fit('bathrooms')
```



From the above model fit plot of the bathrooms variables appears a bit different from the sqft_living due to it's discrete nature.

ii.) Partial Regression Plot

In [691]:
```
# plot partial regression plot
second_iteration.plot_partial_reg()
```



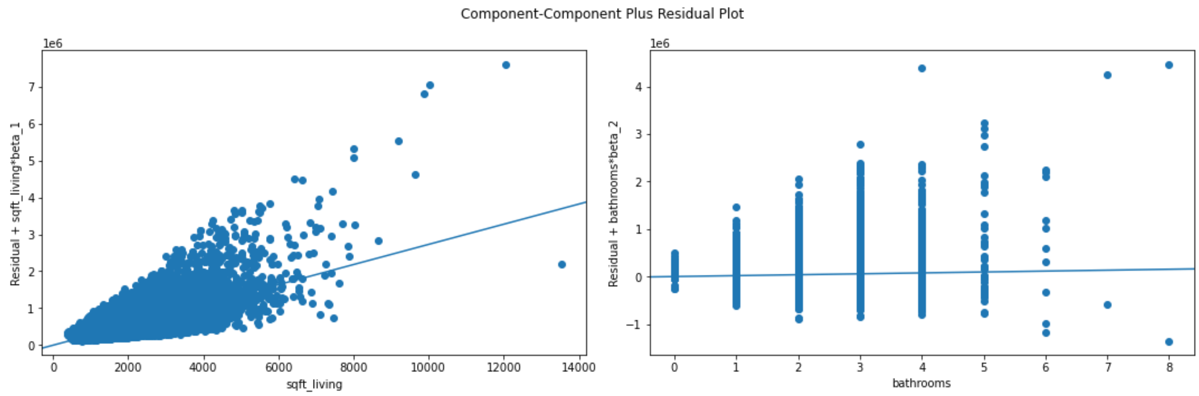
This plot explains the unique contribution of each of the independent variables.

- From left:
  - the `sqft_living` regression plot shows a linear relationship with a non-zero slope, and that means that it is beneficial to add *sqft_living* to the model, vs. having a model without *sqft_living* (i.e. a model with just an intercept and *bathrooms*)
  - The partial regression plot for `bathrooms` is similarly showing the marginal contribution of `bathrooms` compared to a model with just *sqft_living*, albeit, a small one (seen by the near zero slope of regression line).

A reasonable conclusion to reach, looking at these plots, is that both predictors are useful and should be included in the model (given the slight increase of the R^2 value), **BUT** the improvement is very little and this justifies the next model iteration.

iii. component and component-plus-residual plot

In [692]:
```
second_iteration.plot_ccpr()
```



III. Third iteration

Adding all features

Given the small improvement shown in the last iteration, we switch gears on this iteration and add all other independent variables and then check the performance. If the performance does not improve significantly or we see some redundancy, we choose a variable to drop and check the performance again. If the performance is positively impactly, we go to the next step.

In [693]:
```
# evaluate the correlations
highest_corr(house_data_df, 'price', ['sqft_living', 'bathrooms', 'id'])
```

```
bedrooms     0.309057
sqft_lot     0.087430
floors       0.258797
yr_built     0.053433
Name: price, dtype: float64

highest correlation:  {'bedrooms': 0.30905739979220165}
```

The number of floors should be discrete values so we floor the values to discrete values, before adding them to our model for analysis.

In [694]:
```
# we see the floors are floats, which doesn't make sense
house_data_df['floors'].unique()
```

Out[694]:
```
array([2. , 1. , 1.5, 3. , 2.5, 3.5])
```

In [695]:
```
# we floor (ignore what comes after the decimal) the values
house_data_df['floors'] = house_data_df['floors'].map(int)
```

In [713]:
```
# 'floor' is all good now
house_data_df['floors'].unique()
```

Out[713]:
```
array([2, 1, 3], dtype=int64)
```

In [696]:

```
X_all = house_data_df[['sqft_living', 'bathrooms', 'bedrooms', 'floors', 'sq
_lot', 'yr_built']]

third_iteration = RegressionAnalysis(house_data_df, y, X_all)

print(third_iteration.summary())
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  price   R-squared:                       0.56
1
Model:                            OLS   Adj. R-squared:                  0.56
1
Method:                 Least Squares   F-statistic:                     407
9.
Date:                Sun, 10 Sep 2023   Prob (F-statistic):               0.0
0
Time:                        13:43:14   Log-Likelihood:          -2.6506e+0
5
No. Observations:               19164   AIC:                      5.301e+0
5
Df Residuals:                   19157   BIC:                      5.302e+0
5
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
==
                  coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
const        6.577e+06   1.46e+05     45.000      0.000    6.29e+06    6.86e+
06
sqft_living   311.3285      2.996    103.903      0.000     305.455     317.2
02
bathrooms    6.452e+04   3713.698     17.373      0.000    5.72e+04    7.18e+
04
bedrooms    -6.723e+04   2379.882    -28.249      0.000   -7.19e+04   -6.26e+
04
floors        5.59e+04   4199.788     13.310      0.000    4.77e+04    6.41e+
04
sqft_lot       -0.3350      0.045     -7.509      0.000      -0.422      -0.2
48
yr_built    -3371.6483     75.973    -44.379      0.000   -3520.563   -3222.7
34
==============================================================================
=
Omnibus:                    12447.716   Durbin-Watson:                   1.98
5
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           438147.50
4
Skew:                           2.610   Prob(JB):                         0.0
0
Kurtosis:                      25.835   Cond. No.                     3.58e+0
6
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 3.58e+06. This might indicate that there a
re
```

```
strong multicollinearity or other numerical problems.
```

Interpretation:

Third Iteration Regression Results

Looking at the summary above, we can see that the regression line we found was

$$\hat{price} = 6,577,000 + 311.3285 \times \text{sqft\_living} + 64,520 \times \text{bathrooms} - 67,230 \times \text{bedrooms} + 55,9($$
$$371.6483 \times \text{yr\_built}$$

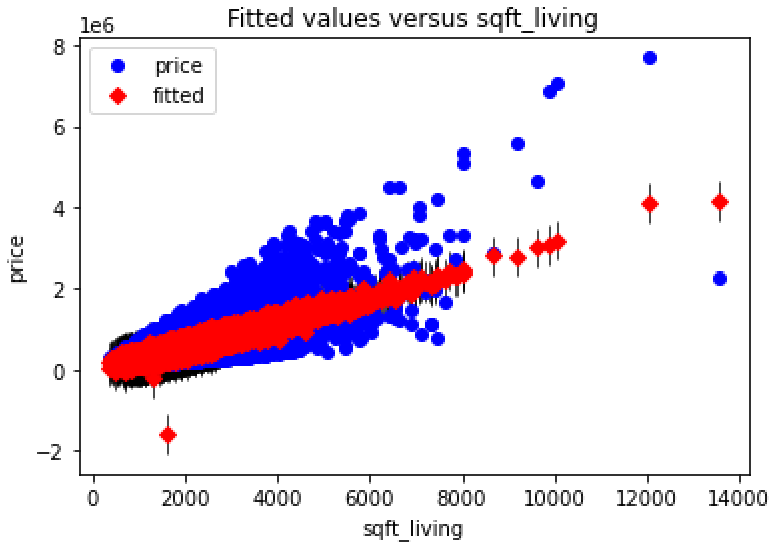
- The model **is statistically significant** overall, with an **F-statistic p-value** well below 0.05.
- The model exhibits an **R-squared value of 56.1%** of the variance in price.
  - Indicating that approximately **56.1% of the variance in house prices is explained by the model in this iteration and its predictor variables.**
  - An overall improvement of 6.4% from the last model.
- The model coefficients are statistically significant, with t-statistic p-values well below 0.05.
- For each increase of 1 square foot of living area, we see an associated increase in price of about 311.33.
  - This increase from the previous model ($+38.89$) suggests that the number of bathrooms was confounding the relationship of the other variables.
- For each increase of 1 bedroom, we see an associated decrease in price of about $-67,230$.
- For each increase of 1 floor, we see an associated increase in price of about $55,900$.
- For each increase of 1 square foot of the lot, we see an associated decrease in price of about -0.3350.
- For each increase of 1 year, we see an associated decrease in price of about $-3,371.65$.

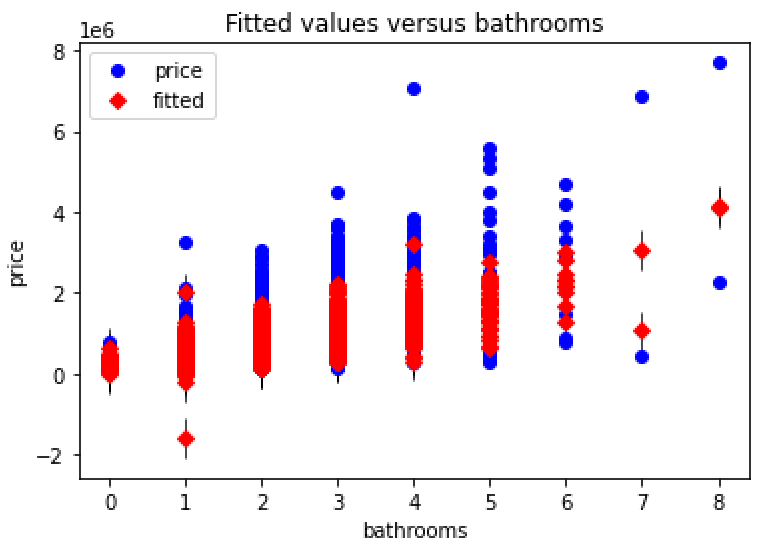
Model with All Features Visualization

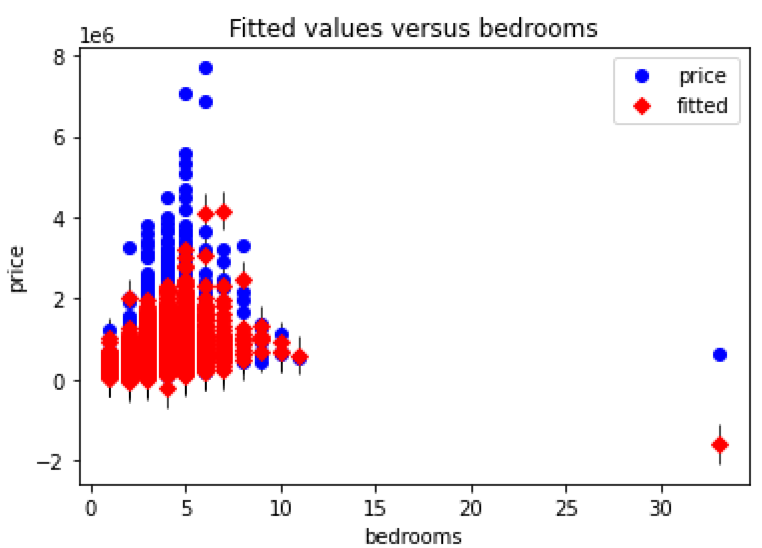
i) Plot fits

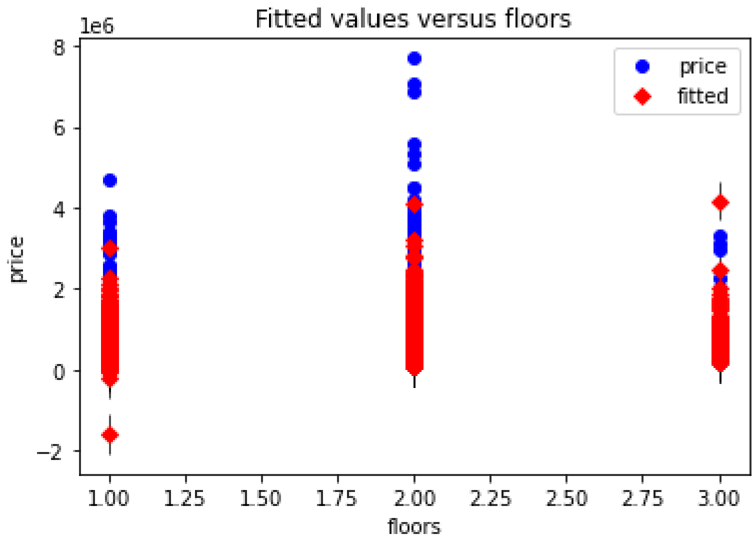
In [697]:

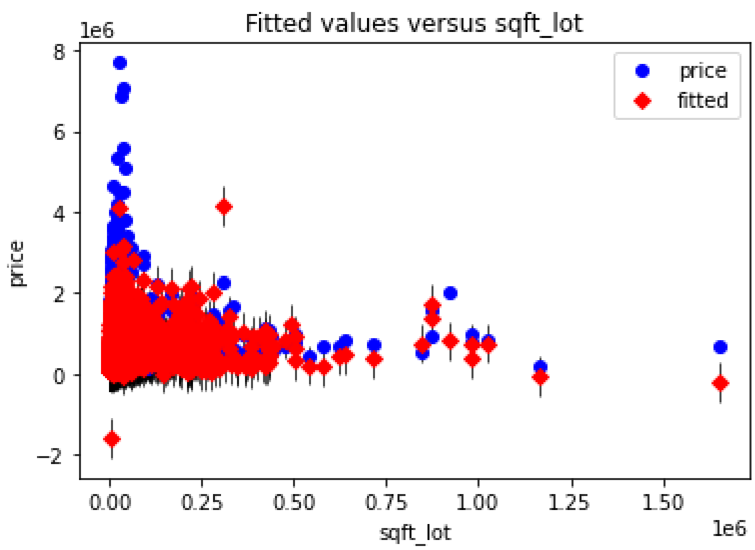
```
for var in X_all.columns:
    third_iteration.plot_fit(var)
```

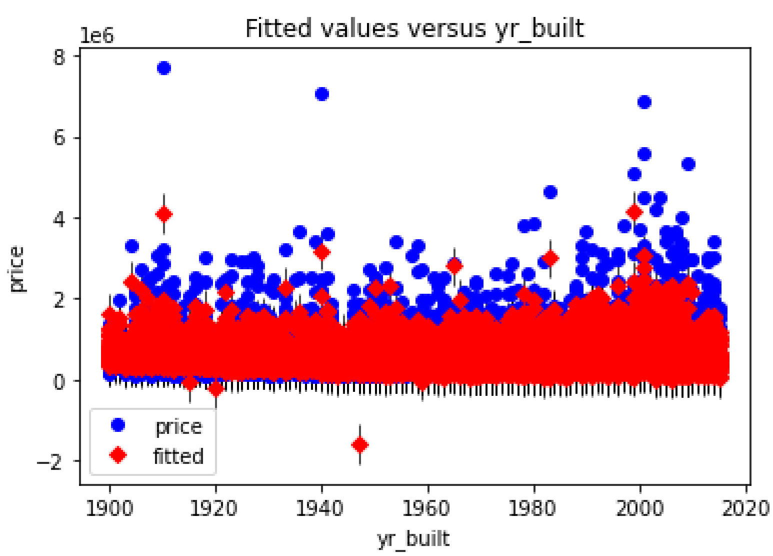

Fitted values versus sqft_living
1e6
8
6
4
2
0
−2
price
fitted
price
0
2000
4000
6000
8000
10000
12000
14000
sqft_living


Fitted values versus bathrooms
1e6
8
6
4
2
0
−2
price
fitted
price
0
1
2
3
4
5
6
7
8
bathrooms


Fitted values versus bedrooms
1e6
8
6
4
2
0
−2
price
fitted
price
0
5
10
15
20
25
30
bedrooms



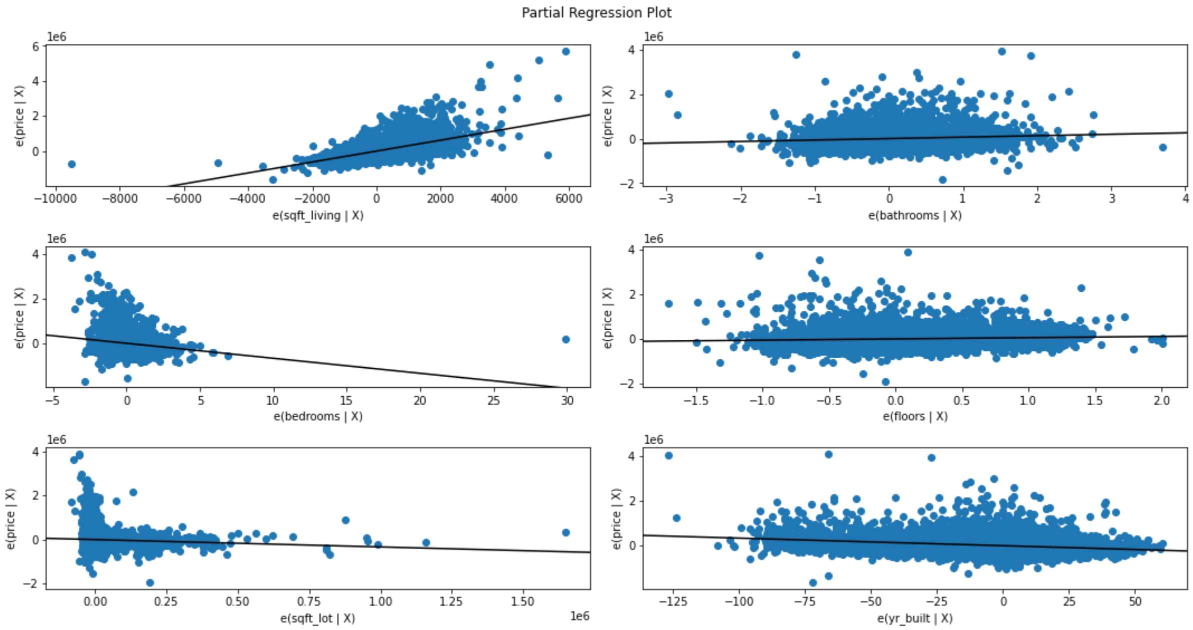




ii) Partial regression plot

We use a partial regression plot to display the contribution of each feature in the performance.

```
In [698]: third_iteration.plot_partial_reg()
```



From the above partial regression plots we see that for some variables like `floors` and `sqft_lot` have slopes of near zero, meaning they do not contribute that much to the model.

But we retain them because they keep the performance high while still having a low p-value, way below our α of 0.05.

From here the model is at it's optimum state given all the predictors, now we add a categorical variable to it.

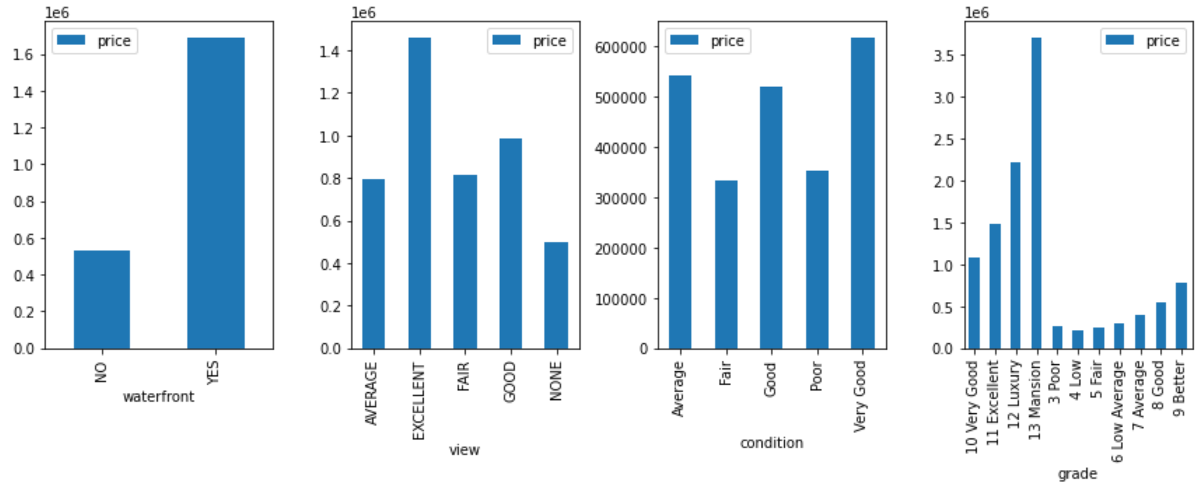
Add a categorical variable

We add atleast one categorical value from our dataset in the model. There are 4 categorical variables in our dataset, namely:

- `waterfront`
- `view`
- `condition`
- `grade`

Let's see their bar graphs below:

In [699]:
```
# Lets see the bar graphs of the categorical features

categorical_features = house_data_df.select_dtypes("object").columns
fig, axes = plt.subplots(ncols=len(categorical_features), figsize=(12,5))

for index, feature in enumerate(categorical_features):
    house_data_df.groupby(feature).mean().plot.bar(
        y="price", ax=axes[index])
plt.tight_layout()
```



We choose a categorical predictor that will be interpretable in our model below. We go with `view`, as this categorizes the condition of the house from **NONE** to **EXCELLENT**.

Since typically for categorical features the data type is a `string`, we have to encode it numerically to allow for regression modelling.

So we create dummy variables (0's and 1's) representing True or False (**One-Hot Encoding**):

In [700]:
```
# select all needed features
X_all = house_data_df[['sqft_living', 'bathrooms', 'bedrooms', 'floors', 'sqft_lot', 'yr_built', 'view']]

# create the dummy variables
X_with_categ = pd.get_dummies(X_all, columns=["view"], dtype=int)

X_with_categ.head()
```

Out[700]:

	sqft_living	bathrooms	bedrooms	floors	sqft_lot	yr_built	view_AVERAGE	view_EXCELLENT
1	2570	2	3	2	7242	1951	0	0
2	770	1	2	1	10000	1933	0	0
3	1960	3	4	1	5000	1965	0	0
4	1680	2	3	1	8080	1987	0	0
5	5420	4	4	1	101930	2001	0	0

To avoid the **Dummy variable Trap** brought by when you can perfectly predict what one variable will be using some combination of the other variables, also known as **Multicollinearity**, we have to drop one of the dummy variables to break the collinearity.

The dummy variable to drop is the `view_NONE`, since it is the lowest category of the views.

This becomes our reference variable.

In [701]:
```
# drop the dummy variable view_NONE
X_with_categ = X_with_categ.drop(columns='view_NONE')

X_with_categ.head()
```

Out[701]:

	sqft_living	bathrooms	bedrooms	floors	sqft_lot	yr_built	view_AVERAGE	view_EXCELLENT
1	2570	2	3	2	7242	1951	0	0
2	770	1	2	1	10000	1933	0	0
3	1960	3	4	1	5000	1965	0	0
4	1680	2	3	1	8080	1987	0	0
5	5420	4	4	1	101930	2001	0	0

Now we check the impact of the categorical variable on our model:

In [702]:
```
# Create new model with the categorical variable (view)
model_with_categ = RegressionAnalysis(house_data_df, y, X_with_categ)

print(model_with_categ.summary())
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  price   R-squared:                       0.59
5
Model:                            OLS   Adj. R-squared:                  0.59
5
Method:                 Least Squares   F-statistic:                     281
4.
Date:                Sun, 10 Sep 2023   Prob (F-statistic):               0.0
0
Time:                        13:43:36   Log-Likelihood:           -2.6428e+0
5
No. Observations:               19164   AIC:                        5.286e+0
5
Df Residuals:                   19153   BIC:                        5.287e+0
5
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
=====
                     coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const           5.894e+06   1.42e+05     41.525      0.000    5.62e+06    6.1
7e+06
sqft_living      283.1530      2.980     95.022      0.000     277.312     28
8.994
bathrooms       6.105e+04   3569.602     17.104      0.000    5.41e+04    6.8
1e+04
bedrooms       -5.703e+04   2301.563    -24.778      0.000   -6.15e+04   -5.2
5e+04
floors           5.82e+04   4035.557     14.422      0.000    5.03e+04    6.6
1e+04
sqft_lot          -0.3236      0.043     -7.542      0.000      -0.408      -
0.239
yr_built       -3020.6393     73.768    -40.948      0.000   -3165.232   -287
6.047
view_AVERAGE    8.464e+04   8520.550      9.934      0.000    6.79e+04    1.0
1e+05
view_EXCELLENT  5.344e+05   1.43e+04     37.246      0.000    5.06e+05    5.6
3e+05
view_FAIR       1.266e+05    1.4e+04      9.013      0.000    9.91e+04    1.5
4e+05
view_GOOD       1.629e+05   1.17e+04     13.907      0.000     1.4e+05    1.8
6e+05
==============================================================================
=
Omnibus:                    11738.083   Durbin-Watson:                   1.98
1
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           441560.81
3
Skew:                           2.367   Prob(JB):                         0.0
0
Kurtosis:                      26.034   Cond. No.                    3.62e+0
6
```

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

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 3.62e+06. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

In [709]:

```
params = (dict(model_with_categ.results.params))
params
```

Out[709]:

```
{'const': 5894483.184484629,
 'sqft_living': 283.1529731421636,
 'bathrooms': 61053.88447506365,
 'bedrooms': -57028.99579869452,
 'floors': 58199.17371256267,
 'sqft_lot': -0.3235921344488202,
 'yr_built': -3020.6392742562716,
 'view_AVERAGE': 84639.9814288361,
 'view_EXCELLENT': 534425.989702586,
 'view_FAIR': 126599.94072644773,
 'view_GOOD': 162932.45880673744}
```

Final Iteration Regression Results:

Looking at the summary above, we can see that the regression line we found was

$$\hat{price} = 5,894,000 + (283.1530 * sqft\_living) + (61053.884 * bathrooms) - (57028.996 * bedroom$$
$$- (0.32359 * sqft\_lot) - (3020.639 * yr\_built) + view\_COEFF. * view\_TY$$

> **NB**: Due to the categorical nature of the `view` variable, the **view_COEFF.** and **view_TYPE** above are placeholders for the regression coefficients for the view types, \ i.e
>
> | Variable (view_TYPE) | Coefficient (view_COEFF.) |
> |---:|---:|
> | **view_AVERAGE** | 84639.9814 |
> | **view_EXCELLENT** | 534425.9897 |
> | **view_FAIR** | 126599.9407 |
> | **view_GOOD** | 162932.4588 |
>
> This because a house only has one view quality so the model requires one view type to work correctly.

Model Interpretation:

- The model **is statistically significant** overall, with an **F-statistic p-value** well below 0.05
- The model exhibits an **R-squared value of 59.5%** of the variance in price.
  - indicating that approximately **59.5% of the variance in house prices is explained by the model in this iteration and its predictor variables**
  - An overall improvement of +3.4% from the last model.
- The model coefficients are statistically significant, with t-statistic p-values well below 0.05
- For each increase of 1 square foot of living area, we see an associated increase in price of about $\$283.1530$
- For each increase of 1 bathroom, we see an associated increase in price of about $\$ + 61,053.88$
- For each increase of 1 bedroom, we see an associated decrease in price of about $\$ - 57,029.00$
- For each increase of 1 floor, we see an associated increase in price of about $\$ + 58,199.17$
- For each increase of 1 Square footage of the lot, we see an associated decrease in price of about $\$ - 0.32359$
- For each increase of 1 year built, we see an associated decrease in price of about $\$ - 3020.64$

- The intercept (const) implies that when all other variables are 0, and `view` is `NONE` , the price is $\$5,894,000$ #### Interpretation of categorical variables:
- Since our **`view_NONE`** is our reference category, we interpret the model summary in reference to it:\ i.e
  - A shift from a house with **no view** to one with a **FAIR view** impacts the price positively by + $\$126,599.94$
  - A shift from a house with **no view** to one with a **AVERAGE view** impacts the price positively by + $\$84,639.98$
  - A shift from a house with **no view** to one with a **GOOD view** impacts the price positively by + $\$162,932.45$

- A shift from a house with **no view** to one with a **EXCELLENT view** impacts the price positively by + $\$534,425.99$

Observation:

> A take-away from this interpretation is that houses with an AVERAGE **quality of** view **might be under-valued since our understanding seems to indicate that it should be better than one with** FAIR **view (+$\$126,599.94$). More investigation of other features is needed to understand whether this can be explained by other variables, or if "**AVERAGE**" is actually undervalued.

Recommendations:

The data analysis and regression modeling have provided valuable insights into the factors influencing house prices in the King County area. To maximize the estimated value of your homes, consider the following strategies:

1. **Leverage Living Space**: Increasing the square footage of the living area has a substantial positive impact on house prices. Consider renovation or expansion projects that can add more living space to your property. Each additional square foot of living area can potentially increase the estimated price by approximately $311.33.
2. **Bathroom Upgrades**: Investing in bathroom upgrades can significantly boost your home's value. Each additional bathroom can increase the estimated price by approximately **$61,053.88. Consider modernizing and expanding your bathrooms to attract potential buyers.
3. **Bedroom Considerations**: While bedrooms are essential, be mindful of overinvesting in them. Each additional bedroom may decrease the estimated price by approximately $57,028. Ensure that the number of bedrooms aligns with the needs of potential buyers.
4. **Flooring and Layout**: Houses with more floors tend to command higher prices. Consider optimizing your home's layout to make the most of the available space. Each additional floor can increase the estimated price by about $58,199.
5. **Lot Size Management**: Pay attention to your property's lot size. Smaller lots may deter some buyers, and each square foot reduction in lot size can potentially decrease the estimated price by approximately $0.32. Evaluate your landscaping and lot usage to make the most of your property.
6. **Year Built**: Keep up with yearly maintenance and consider renovations to modernize your home. Older homes tend to have lower estimated prices, with each year of age potentially decreasing the estimated price by around $3,020.
7. **View Quality Matters**: If your property has a view, consider it a valuable asset. Homes with "excellent" views can command significantly higher prices, with each category upgrade potentially adding substantial value. Invest in maintaining or enhancing the quality of the view if possible.
8. **Market Trends**: Keep an eye on local real estate market trends. Factors like location, neighborhood, and market demand can also influence house prices. Stay informed about the King County housing market to make informed decisions.
9. **Consult a Real Estate Expert**: To get a precise estimate of your home's value and tailor your strategy to your specific situation, consider consulting a local real estate expert or appraiser. They can provide personalized recommendations based on your property's unique characteristics.

Conclusion:

The final regression model achieved an R-squared value of approximately 59.5%, indicating that 59.5% of the variance in house prices is explained by the selected predictor variables. The performance was not that good given that most of the variables were dropped or not used at all. It is worthy noting that the recommendations are data-driven and based on the statistical analysis. However, individual factors and market conditions can vary, so it's essential to assess your property's specific situation before making any significant decisions.