Phase two project

- student name : Julie Chepngeno
- course : DS Full Time

VALUE ANALYSIS FOR HOUSE SALE PRICES IN KING COUNTY

1.Project Overview

a) introduction

At a Real Estate, buying or selling a home is one of the most significant financial decisions you'll make in your life. Our goal is to make the home buying or selling process as smooth and stress-free as possible for our clients. In this project,we will be analyzing house sales in a western county which can be found in kc_house_data.csv

The real estate market serves as a reflection of societal aspirations, investment opportunities, and the pursuit of the elusive "dream home." Whether it's residential, commercial, or industrial, real estate offers a multitude of possibilities for individuals, businesses, and communities.It involves home owners looking to sell their property and home buyers looking to buy a dream home

b)problem statement

The real estate agency needs to develop a comprehensive advisory framework to assist homeowners in understanding the potential value-adding effects of specific home renovations. This framework should include detailed information on the types of renovations, their estimated costs, and the potential increase in the property's value. By addressing this problem, the agency can empower homeowners to make informed decisions, thereby increasing their satisfaction, trust, and the likelihood of engaging the agency's services.

- Main objective

The main objective of the real estate agency's initiative is to provide homeowners with accurate and valuable advice regarding home renovations and their potential impact on the estimated value of their properties

- specific objectives

Develop a Comprehensive Renovation Advisory Framework: The agency aims to create a structured framework that outlines different renovation types and their potential impact on property value.

Track Success Metrics: The agency will track key success metrics, such as the number of homeowners served, the percentage of clients who reported increased property value, and the number of referrals received.

d) Data Understanding

In this project, the data was collected from a CSV (Comma-Separated Values) file. The dataset, kc_house_data.csv, contains information related to real estate properties in King County, Washington. It includes various attributes that describe the properties, such as their location, size, condition, and sale prices.

Analyzing the data set will help us to provide advice to homeowners about how home renovations might increase the estimated value of their homes, and by what amount.

2) Importing libraries

In [2]:
```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
import sklearn
import statsmodels.api as sm
from statsmodels.formula.api import ols
from sklearn.linear_model import LinearRegression
%matplotlib inline
```

Reading and checking the data

In [3]:
```
#loading kc_house_data.csv
df = pd.read_csv("kc_house_data.csv")
# Display the first few rows of the dataset
df.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqf
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN	NONE	...	7 Average	1180	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	NO	NONE	...	7 Average	2170	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	NO	NONE	...	6 Low Average	770	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	NO	NONE	...	7 Average	1050	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	NO	NONE	...	8 Good	1680	

5 rows × 21 columns

In [4]:
```
# Obtaining info for the dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 1   date           21597 non-null  object 
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64  
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64  
 6   sqft_lot       21597 non-null  int64  
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  object 
 9   view           21534 non-null  object 
 10  condition      21597 non-null  object 
 11  grade          21597 non-null  object 
 12  sqft_above     21597 non-null  int64  
 13  sqft_basement  21597 non-null  object 
 14  yr_built       21597 non-null  int64  
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64  
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64  
 20  sqft_lot15     21597 non-null  int64  
dtypes: float64(6), int64(9), object(6)
memory usage: 3.5+ MB
```

checking and dropping duplicated values

In [5]:
```
# Check for duplicated values
df.id.duplicated().sum()
```

Out[5]:
```
177
```

we have a total of 177 duplicates out of 21597 entries in the house unique identifier 'id'

In [6]:
```
# Display duplicates
df.loc[df["id"].duplicated()==True]
```

Out[6]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_abov
94	6021501535	12/23/2014	700000.0	3	1.50	1580	5000	1.0	NO	NONE	...	8 Good	129
314	4139480200	12/9/2014	1400000.0	4	3.25	4290	12103	1.0	NO	GOOD	...	11 Excellent	269
325	7520000520	3/11/2015	240500.0	2	1.00	1240	12092	1.0	NO	NONE	...	6 Low Average	96
346	3969300030	12/29/2014	239900.0	4	1.00	1000	7134	1.0	NO	NONE	...	6 Low Average	100
372	2231500030	3/24/2015	530000.0	4	2.25	2180	10754	1.0	NO	NONE	...	7 Average	110
...	...	...	...	...	...	...	...	...	...	...	...	...	
20165	7853400250	2/19/2015	645000.0	4	3.50	2910	5260	2.0	NO	NONE	...	9 Better	291
20597	2724049222	12/1/2014	220000.0	2	2.50	1000	1092	2.0	NO	NONE	...	7 Average	99
20654	8564860270	3/30/2015	502000.0	4	2.50	2680	5539	2.0	NaN	NONE	...	8 Good	268
20764	6300000226	5/4/2015	380000.0	4	1.00	1200	2171	1.5	NO	NONE	...	7 Average	120
21565	7853420110	5/4/2015	625000.0	3	3.00	2780	6000	2.0	NO	NONE	...	9 Better	278

177 rows × 21 columns

The column "id" is a unique identifier therefore we drop those that are duplicated and retain the first entry implace=True i added to ensure the chane is carried forward when the data is called again

In [7]:
```
# dropping duplicates
df.drop_duplicates(subset="id", inplace=True)
```

Checking and dropping null values

In [8]:
```
# checking for null value in the dataset
def drop_null(data, subset, axis): # Subset should be a list
 """A simple function to find null entries and drop them"""
 print(f"{df.isna().sum()}")
 df.dropna(subset=subset, inplace=True, axis=axis)
drop_null(df, ['yr_renovated', 'waterfront'], 0)
```

```
id                0
date              0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront     2353
view             63
condition         0
grade             0
sqft_above        0
sqft_basement     0
yr_built          0
yr_renovated   3804
zipcode           0
lat               0
long              0
sqft_living15     0
sqft_lot15        0
dtype: int64
```

In the above cell, we dropped rows with null values in the categorical columns as it would not be wise to fill them with biased information.

EDA

In [9]:
```
for column in df.columns:
 print(df[column].value_counts())
```

```
6414100192    1
3333002450    1
1326049170    1
3043200035    1
1424069044    1
             ..
1446400725    1
4030500130    1
3319500299    1
7139800020    1
1523300157    1
Name: id, Length: 15691, dtype: int64
6/25/2014     103
6/23/2014     102
10/28/2014     94
7/8/2014       93
7/14/2014      93
              ...
1/10/2015       1
```

In [10]:
```
df.condition.value_counts()
```

Out[10]:
```
Average      10170
Good          4132
Very Good     1245
Fair           125
Poor            19
Name: condition, dtype: int64
```

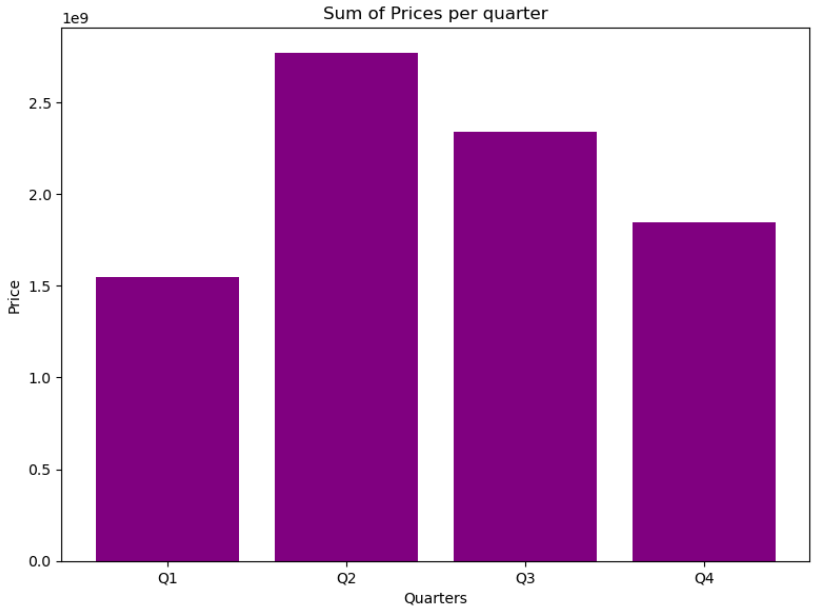
In [11]:
```
to_replace = {"Poor": 1, "Fair": 2,"Average": 3, "Good": 4, "Very Good": 5}
df.condition = df.condition.map(to_replace)
df.condition.value_counts()
```

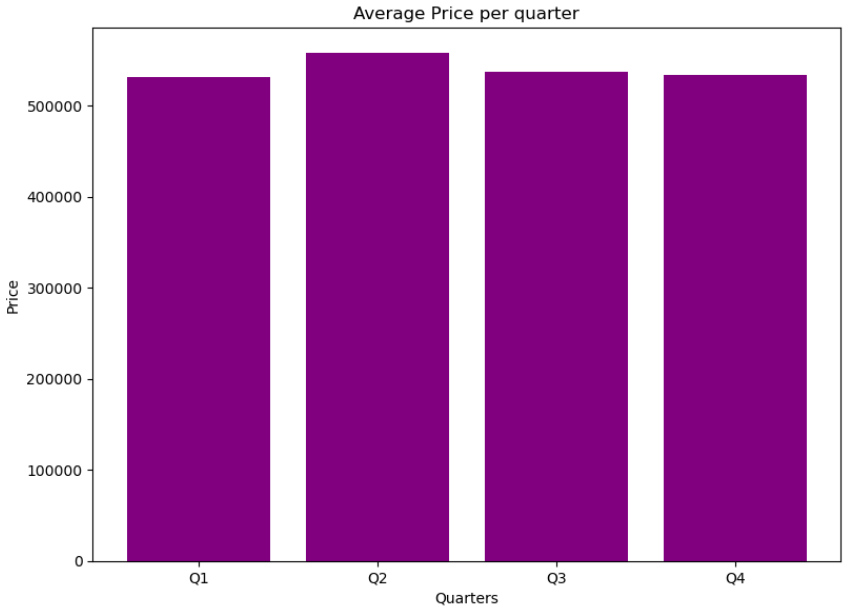
Out[11]:
```
3    10170
4     4132
5     1245
2      125
1       19
Name: condition, dtype: int64
```

- In the above cell,the condition ratings are converted into a discrete variable
- The changes show from 1 to 5,where the lowest has 1 as poor,with the midpoint 3 as average,and the highest 5 as very good

In [12]:
```
# Converting 'date' column to date-time format
df['date'] = pd.to_datetime(df['date'])
df['month'] = df['date'].dt.month.astype(str)
# Converting to quartes
def replace_quarters(x):
 """A bit complex but simple function to demarcate months into quarters"""
 x = int(x)
 if x <= 3:
  str_ = str(x).replace(str(x), "Q1")
 elif 4 <= x <= 6:
  str_ = str(x).replace(str(x), "Q2")
 elif 7 <= x <= 9:
  str_ = str(x).replace(str(x), "Q3")
 elif 10 <= x <= 12:
  str_ = str(x).replace(str(x), "Q4")
 return str_
# Replacing in the column
df["month"] = df['month'].map(replace_quarters)
# Visualizing
fig, ax = plt.subplots(figsize=(16,6), ncols=2)
plot_sum = df.groupby("month")["price"].sum()
plot_mean = df.groupby("month")["price"].mean()
ax[0].bar(plot_sum.index, plot_sum.values, color='purple')
ax[1].bar(plot_mean.index, plot_mean.values, color='purple')
ax[0].set(ylabel='Price', xlabel='Quarters', title='Sum of Prices per quarter')
ax[1].set(ylabel='Price', xlabel='Quarters', title='Average Price per quarter')

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





- There is no much difference between the quarters except that the average price in quarter 2 is slightly higher than the rest
- There are columns in the dataset tat may not be useful in evaluation and we drop them by initializing the list called dropped and then dropping thm from our dataset

In [13]:
```
# Initializing a list for columns to drop
def drop_cols(data, subset, axis):
 """A simple function to drop columns"""
 data.drop(subset, axis=axis, inplace=True)

dropped = ['id', 'date', 'view', 'lat', 'long', 'month']
drop_cols(df, dropped, 1)
```

- In the cells below, the column 'grade' is split for its values and only the first section is singled out to be converted into numeric type and used as a scale for measuring the grade of the houses / properties
- Shifting has been done so that the grading parameters may start from 1.
- From 1 to 11, where the lowest has 1 as poor, with a midpoint of 7 as average and the highest being 11 as mansion

In [14]:
```
# checking the grading system used in the dataset
df['grade'].value_counts()
```

Out[14]:
```
7 Average        6504
8 Good           4429
9 Better         1922
6 Low Average    1459
10 Very Good      832
11 Excellent      288
5 Fair            163
12 Luxury          66
4 Low              16
13 Mansion         11
3 Poor              1
Name: grade, dtype: int64
```

In [15]:
```
# Converting the 'grade' column to numeric dtype and shifting the scale by 2
df['grade'] = df['grade'].map(lambda x: x.split(' ')[0]).astype(int) - 2
```

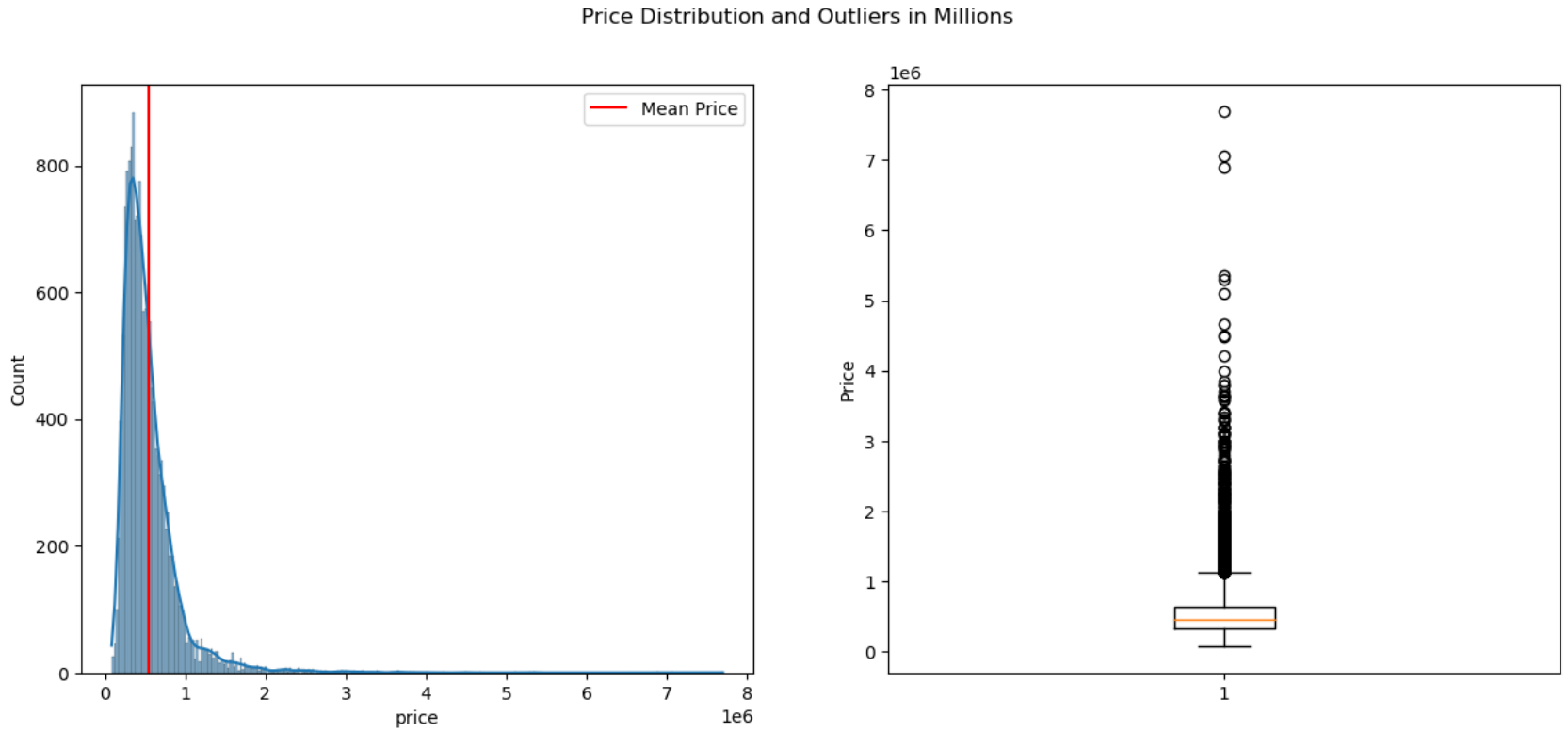
In [16]:
```
# Transforming 'yr_renovated' column into a categorical variable with `renovated` and `not_renovated`
def replace(x):
 """A simple function to categorize the column into renovated and not_renovated"""
 if x > 0:
  str_ = str(x).replace(str(x), "renovated")
 else:
  str_ = str(x).replace(str(x), "not renovated")
 return str_
# Renaming our coulumn and calling our function on the dataset
df['renovated'] = df.yr_renovated.map(replace)
# Viewing the changes
df.renovated.value_counts()
# Dropping the original column
df.drop('yr_renovated', axis=1, inplace=True)
```

- Replacing the '?' in the column 'sqft_basement' with zero values since the entries in those columns showed similar areas for "sqft_living" and "sqft_above'

In [17]:
```
# Replacing the '?' character with "0.0"
df['sqft_basement'].replace({'?': "0.0"}, inplace=True)
# Converting the column 'sqft_basement' to an integer column
df['sqft_basement'] = df.sqft_basement.astype(float)
```

Checking for outliers in the price column

In [18]:
```
# Creating a figure space and visualizing
fig, ax = plt.subplots(figsize=(15,6), ncols=2)
# Histogram
sns.histplot(df.price, kde=True, ax=ax[0])
ax[0].axvline(df['price'].mean(), color='red', label="Mean Price")
# Boxplot
ax[1].boxplot(df['price'])
ax[1].set_ylabel("Price")
ax[0].legend()
# Title and showing
fig.suptitle("Price Distribution and Outliers in Millions")
plt.show()
```



from the above visualizations, we see that a majority of the price distributions lie between 0 and 1.2 million with those beyond this considered as outliers. On the other hand, we consider these prices to be important for our analysis except those above 5 million that we considered as genuine outliers which we will drop

In [19]:
```
# obtaining descriptive statistics on our dataset
df.describe()
```

Out[19]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	condition	grade	sqft_above
count	1.569100e+04	15691.000000	15691.000000	15691.000000	1.569100e+04	15691.000000	15691.000000	15691.000000	15691.000000
mean	5.418726e+05	3.379899	2.122761	2087.263654	1.531265e+04	1.496877	3.411637	5.667899	1794.994392
std	3.743919e+05	0.934517	0.766978	919.928706	4.197367e+04	0.539743	0.651235	1.171426	829.131366
min	8.200000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.000000	1.000000	1.000000	370.000000
25%	3.219750e+05	3.000000	1.750000	1430.000000	5.048000e+03	1.000000	3.000000	5.000000	1200.000000
50%	4.500000e+05	3.000000	2.250000	1920.000000	7.600000e+03	1.500000	3.000000	5.000000	1570.000000
75%	6.450000e+05	4.000000	2.500000	2558.500000	1.071200e+04	2.000000	4.000000	6.000000	2220.000000
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	5.000000	11.000000	9410.000000

In [21]:
```
# Showcasing the maximum, average and minimum market prices
print(f"""
 The average price in King County is {round(df.price.mean(), 2)},
 it has the hihgest price and lowest price being {round(df.price.max())} and {round(df.price.min())} respectiv
 """)
```

```
 The average price in King County is 541872.57,
 it has the hihgest price and lowest price being 7700000 and 82000 respectively
```

Modelling

Checking our dataset for columns with high multicolinearity and dropping them as well due to their redundancy

In [22]:
```
def model_results(y, X):
 """A perfect use of a simple function to fit the models"""
 model = sm.OLS(y, sm.add_constant(X))
 fit = model.fit()
 return fit
```

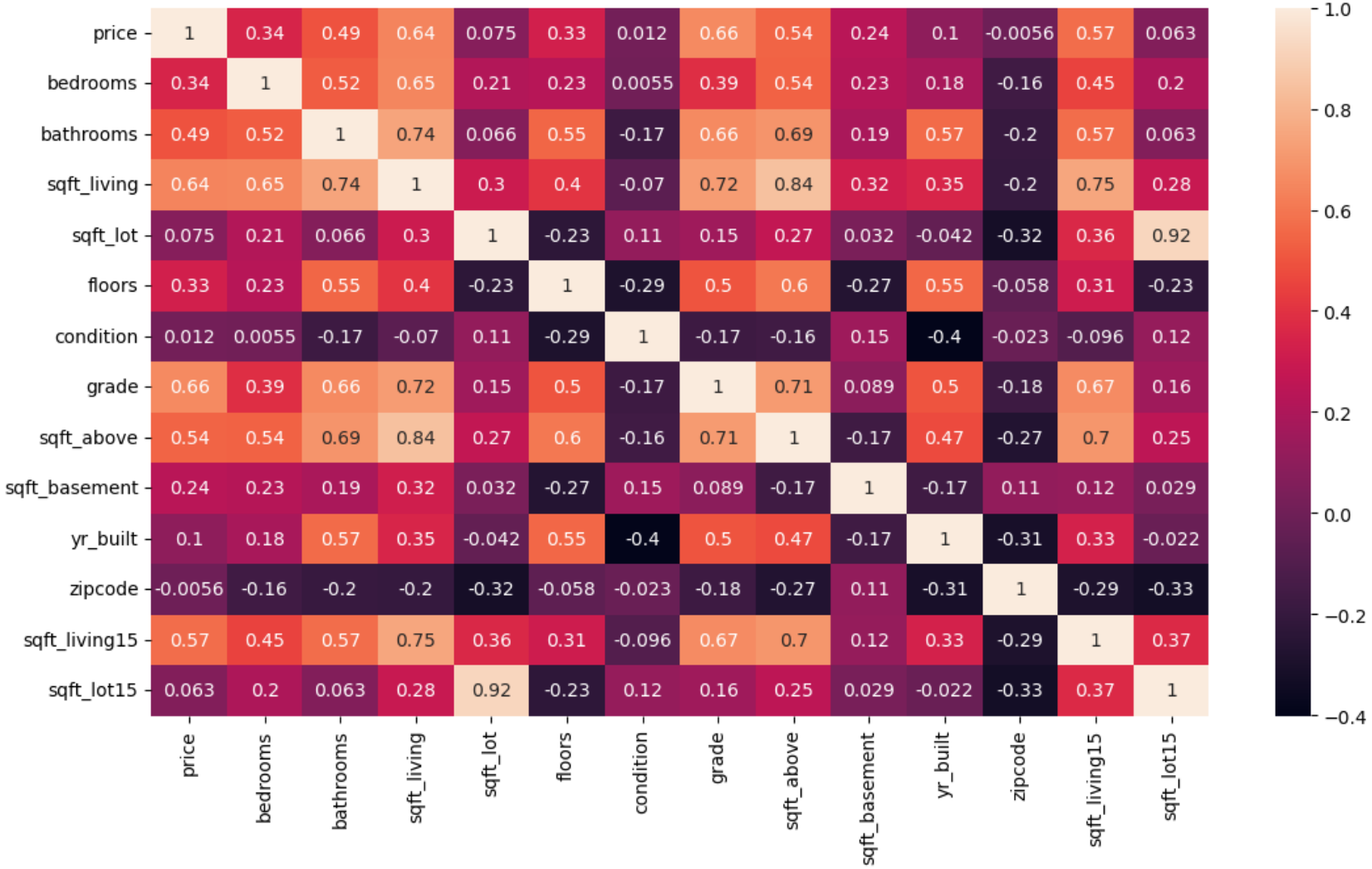
In [23]:
```
# Checking for Multicollinearity in our predictors
corr_df = df.corr(numeric_only=True).abs().stack().reset_index().sort_values(0, ascending=False)
corr_df['pairs'] = list(zip(corr_df.level_0, corr_df.level_1))
# Dropping 'level_0' and 'level_1'
corr_df.set_index(['pairs'], inplace=True)
corr_df.drop(columns=['level_0', 'level_1'], inplace=True)
# Renaming our column
corr_df.columns = ["corr_coef"]
```

In [24]:
```
# Veiwing the highly correlated predictor pairs
corr_df[(corr_df.corr_coef > 0.75) & (corr_df.corr_coef < 1)]
```

Out[24]:

pairs	corr_coef
(sqft_above, sqft_living)	0.876111
(sqft_living, sqft_above)	0.876111
(sqft_living, grade)	0.764588
(grade, sqft_living)	0.764588
(grade, sqft_above)	0.758671
(sqft_above, grade)	0.758671
(sqft_living, sqft_living15)	0.756656
(sqft_living15, sqft_living)	0.756656
(sqft_living, bathrooms)	0.754082
(bathrooms, sqft_living)	0.754082

In [25]:
```
# Visualizing the correlation between predictors
plt.figure(figsize=(13,7))
sns.heatmap(df.corr(method='spearman', numeric_only=True), annot=True);
```



From the observations made above we see that there is a very high correlation between the predictor columns of 'sqft_above', 'sqft_living', 'sqft_living15, 'grade',and 'bathrooms'. Dropping some of these will eliminate multicollinearity features. The columns to drop will be 'sqft_above', 'sqft_living15', 'grade',and 'bathrooms'

In [26]:
```
#  Initializing a list of columns to drop
high_corr_pred = ['sqft_above', 'sqft_living15', 'grade', 'bathrooms']
# Dropping the columns permanently
drop_cols(df, high_corr_pred, 1)
# Viewing the the remaining dataset
print(df.shape)
df.head()
```

```
(15691, 12)
```

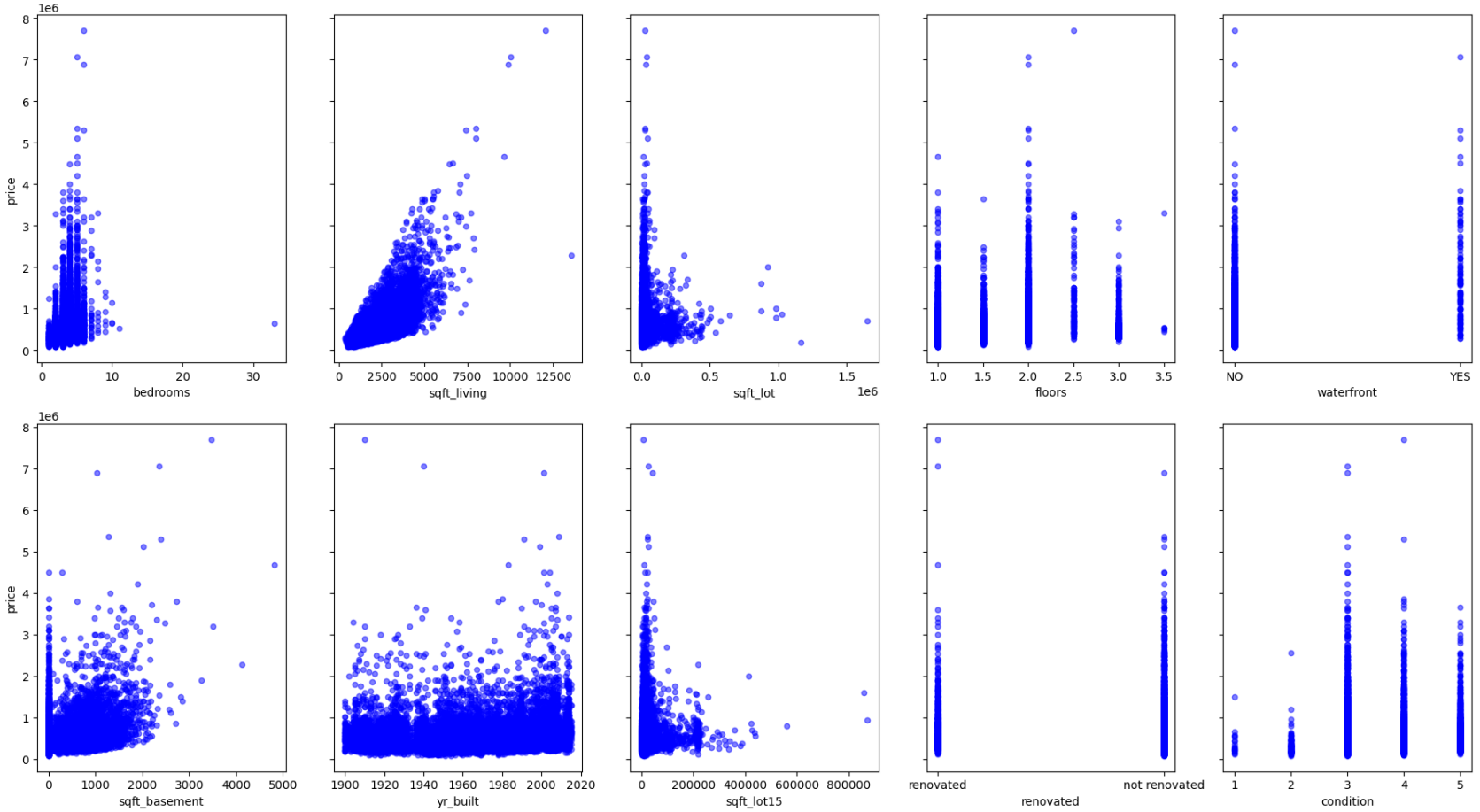
Out[26]:

	price	bedrooms	sqft_living	sqft_lot	floors	waterfront	condition	sqft_basement	yr_built	zipcode	sqft_lot15	renovated
1	538000.0	3	2570	7242	2.0	NO	3	400.0	1951	98125	7639	renovated
3	604000.0	4	1960	5000	1.0	NO	5	910.0	1965	98136	5000	not renovated
4	510000.0	3	1680	8080	1.0	NO	3	0.0	1987	98074	7503	not renovated
5	1230000.0	4	5420	101930	1.0	NO	3	1530.0	2001	98053	101930	not renovated
6	257500.0	3	1715	6819	2.0	NO	3	0.0	1995	98003	6819	not renovated

Fitting the baseline model

In [28]:
```
# Visualization of the predictor relationships with price
fig = plt.figure(figsize=(18,10))
axes = fig.subplots(nrows=2, ncols=5, sharey=True)
for xcol, ax in zip(['bedrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront',\
 'sqft_basement', 'yr_built', 'sqft_lot15', 'renovated', 'condition'], axes.flatten()):
 df.plot(kind='scatter', x=xcol, y='price', ax=ax, alpha=0.5, color='blue')

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



In [29]:
```
# Checking for the highly correlated
df.corr(numeric_only=True)['price']
```

Out[29]:
```
price            1.000000
bedrooms         0.306573
sqft_living      0.706341
sqft_lot         0.082954
floors           0.258315
condition        0.033602
sqft_basement    0.318285
yr_built         0.048950
zipcode         -0.048485
sqft_lot15       0.078430
Name: price, dtype: float64
```

In [30]:
```
# Determining the variables
y = df['price']
X_baseline = df[['sqft_living']]
```

In [32]:
```
# Fitting the model
baseline_results = model_results(y, X_baseline)
# Our summary
print(baseline_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.499
Model:                            OLS   Adj. R-squared:                  0.499
Method:                 Least Squares   F-statistic:                 1.562e+04
Date:                Wed, 05 Jul 2023   Prob (F-statistic):               0.00
Time:                        12:24:53   Log-Likelihood:            -2.1821e+05
No. Observations:               15691   AIC:                         4.364e+05
Df Residuals:                   15689   BIC:                         4.364e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -5.815e+04   5246.275    -11.083      0.000   -6.84e+04   -4.79e+04
sqft_living   287.4664      2.300    124.985      0.000     282.958     291.975
==============================================================================
Omnibus:                    11240.432   Durbin-Watson:                   1.974
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           478061.744
Skew:                           2.961   Prob(JB):                         0.00
Kurtosis:                      29.385   Cond. No.                     5.66e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.66e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

- The model is generally statistically sigmificant at a significance level of with a _value of 0.0, and explains about 49% of the variance in price.
- The constant and coefficient are statistically significant.
- For a unit increase in square-foot living area, we see an associated increase in 275 dollars in selling price of the houses.

In [33]:
```
# Fitting a multiple regression
X_multiple = df.copy().drop(['price', 'zipcode'], axis=1)
X_multiple = pd.get_dummies(X_multiple, columns=['waterfront', 'renovated'], drop_first=True)
X_multiple
```

Out[33]:

	bedrooms	sqft_living	sqft_lot	floors	condition	sqft_basement	yr_built	sqft_lot15	waterfront_YES	renovated_renovated
1	3	2570	7242	2.0	3	400.0	1951	7639	0	1
3	4	1960	5000	1.0	5	910.0	1965	5000	0	0
4	3	1680	8080	1.0	3	0.0	1987	7503	0	0
5	4	5420	101930	1.0	3	1530.0	2001	101930	0	0
6	3	1715	6819	2.0	3	0.0	1995	6819	0	0
...	...	...	...	...	...	...	...	...	...	...
21591	3	1310	1294	2.0	3	130.0	2008	1265	0	0
21592	3	1530	1131	3.0	3	0.0	2009	1509	0	0
21593	4	2310	5813	2.0	3	0.0	2014	7200	0	0
21594	2	1020	1350	2.0	3	0.0	2009	2007	0	0
21596	2	1020	1076	2.0	3	0.0	2008	1357	0	0

15691 rows × 10 columns

In [34]:
```
# Fittng our multiple regression model
multiple_results = model_results(y, X_multiple)
print(multiple_results.summary())
```

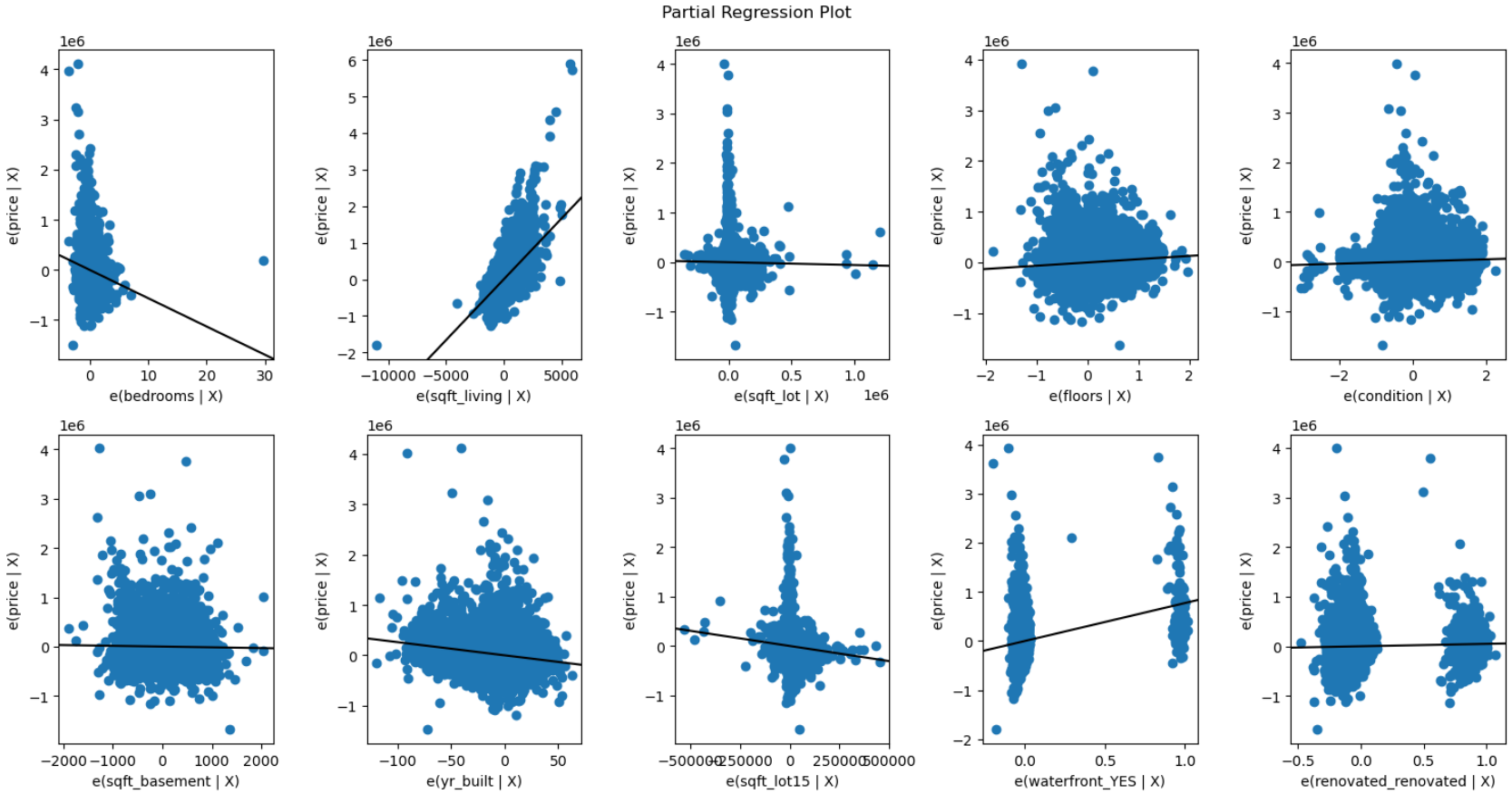
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.592
Model:                            OLS   Adj. R-squared:                  0.592
Method:                 Least Squares   F-statistic:                     2277.
Date:                Wed, 05 Jul 2023   Prob (F-statistic):               0.00
Time:                        12:29:41   Log-Likelihood:            -2.1659e+05
No. Observations:               15691   AIC:                         4.332e+05
Df Residuals:                   15680   BIC:                         4.333e+05
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                           coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const                5.024e+06   1.66e+05     30.294      0.000     4.7e+06    5.35e+06
bedrooms            -5.655e+04   2527.712    -22.374      0.000   -6.15e+04   -5.16e+04
sqft_living            331.9504      3.228    102.835      0.000     325.623     338.278
sqft_lot                -0.0567      0.066     -0.863      0.388      -0.185       0.072
floors               6.439e+04   4619.149     13.940      0.000    5.53e+04    7.34e+04
condition            2.198e+04   3238.782      6.787      0.000    1.56e+04    2.83e+04
sqft_basement         -15.3454      5.531     -2.774      0.006     -26.187      -4.504
yr_built             -2612.6078    83.651    -31.232      0.000   -2776.573   -2448.643
sqft_lot15              -0.6134      0.099     -6.211      0.000      -0.807      -0.420
waterfront_YES       7.723e+05   2.22e+04     34.833      0.000    7.29e+05    8.16e+05
renovated_renovated  4.939e+04   1.01e+04      4.876      0.000    2.95e+04    6.92e+04
==============================================================================
Omnibus:                     9723.971   Durbin-Watson:                   1.978
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           360515.689
Skew:                           2.412   Prob(JB):                         0.00
Kurtosis:                      25.982   Cond. No.                     4.47e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.47e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

- The model overall is statistically significant at standard alpha of 0.05 being 0.0, and it explains 58% percent of the variance in sale price.
- The model coefficients are statistically significant except for "sqft_lot".
- In comparison to our baseline model, our multiple model is an improvement with explained variance in price from 49% to 58%
- The constant here explains that all factors held constant, and a house with no waterfront and not renovated, we would have a sale price of 4,844,000 dollars

In [35]:
```
fig = plt.figure(figsize=(15,8))
sm.graphics.plot_partregress_grid(
 multiple_results,
 exog_idx=list(X_multiple.columns.values),
 grid=(2,5),
 fig=fig)
plt.tight_layout()
plt.show()
```

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



Shifting and centering

In [38]:
```
# Preparing data for shifting and centering
categorical_cols = ['yr_built', 'waterfront_YES', 'renovated_renovated']
numeric_cols = ['bedrooms', 'sqft_living', 'sqft_lot', 'floors', 'sqft_basement', 'sqft_lot15', 'condition']
# Doing the centering
X_centred = X_multiple[numeric_cols]
for col in X_centred.columns:
 X_centred[col] = X_centred[col] - X_centred[col].mean()
# Viewing the centred dataset
X_centred
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_2932\2564494513.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  X_centred[col] = X_centred[col] - X_centred[col].mean()
```

Out[38]:

	bedrooms	sqft_living	sqft_lot	floors	sqft_basement	sqft_lot15	condition
1	-0.379899	482.736346	-8070.645402	0.503123	113.848894	-5285.724046	-0.411637
3	0.620101	-127.263654	-10312.645402	-0.496877	623.848894	-7924.724046	1.588363
4	-0.379899	-407.263654	-7232.645402	-0.496877	-286.151106	-5421.724046	-0.411637
5	0.620101	3332.736346	86617.354598	-0.496877	1243.848894	89005.275954	-0.411637
6	-0.379899	-372.263654	-8493.645402	0.503123	-286.151106	-6105.724046	-0.411637
...	...	...	...	...	...	...	...
21591	-0.379899	-777.263654	-14018.645402	0.503123	-156.151106	-11659.724046	-0.411637
21592	-0.379899	-557.263654	-14181.645402	1.503123	-286.151106	-11415.724046	-0.411637
21593	0.620101	222.736346	-9499.645402	0.503123	-286.151106	-5724.724046	-0.411637
21594	-1.379899	-1067.263654	-13962.645402	0.503123	-286.151106	-10917.724046	-0.411637
21596	-1.379899	-1067.263654	-14236.645402	0.503123	-286.151106	-11567.724046	-0.411637

15691 rows × 7 columns

In [39]:
```
# Concatenating
X_centred = pd.concat([X_centred, X_multiple[categorical_cols]], axis=1)
X_centred['yr_built'] = X_centred['yr_built'] - 1900
X_centred
```

Out[39]:

	bedrooms	sqft_living	sqft_lot	floors	sqft_basement	sqft_lot15	condition	yr_built	waterfront_YES	renovated_
1	-0.379899	482.736346	-8070.645402	0.503123	113.848894	-5285.724046	-0.411637	51	0	
3	0.620101	-127.263654	-10312.645402	-0.496877	623.848894	-7924.724046	1.588363	65	0	
4	-0.379899	-407.263654	-7232.645402	-0.496877	-286.151106	-5421.724046	-0.411637	87	0	
5	0.620101	3332.736346	86617.354598	-0.496877	1243.848894	89005.275954	-0.411637	101	0	
6	-0.379899	-372.263654	-8493.645402	0.503123	-286.151106	-6105.724046	-0.411637	95	0	
...	...	...	...	...	...	...	...	...	...	...
21591	-0.379899	-777.263654	-14018.645402	0.503123	-156.151106	-11659.724046	-0.411637	108	0	
21592	-0.379899	-557.263654	-14181.645402	1.503123	-286.151106	-11415.724046	-0.411637	109	0	
21593	0.620101	222.736346	-9499.645402	0.503123	-286.151106	-5724.724046	-0.411637	114	0	
21594	-1.379899	-1067.263654	-13962.645402	0.503123	-286.151106	-10917.724046	-0.411637	109	0	
21596	-1.379899	-1067.263654	-14236.645402	0.503123	-286.151106	-11567.724046	-0.411637	108	0	

15691 rows × 10 columns

In [40]:
```
# fittimg model
centred_results = model_results(y, X_centred)
print(centred_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.592
Model:                            OLS   Adj. R-squared:                  0.592
Method:                 Least Squares   F-statistic:                     2277.
Date:                Wed, 05 Jul 2023   Prob (F-statistic):               0.00
Time:                        12:40:50   Log-Likelihood:            -2.1659e+05
No. Observations:               15691   AIC:                         4.332e+05
Df Residuals:                   15680   BIC:                         4.333e+05
Df Model:                          10
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
const                7.199e+05   6391.692    112.627      0.000    7.07e+05    7.32e+05
bedrooms            -5.655e+04   2527.712    -22.374      0.000   -6.15e+04   -5.16e+04
sqft_living           331.9504      3.228    102.835      0.000     325.623     338.278
sqft_lot                -0.0567      0.066     -0.863      0.388      -0.185       0.072
floors               6.439e+04   4619.149     13.940      0.000    5.53e+04    7.34e+04
sqft_basement         -15.3454      5.531     -2.774      0.006     -26.187      -4.504
sqft_lot15              -0.6134      0.099     -6.211      0.000      -0.807      -0.420
condition            2.198e+04   3238.782      6.787      0.000    1.56e+04    2.83e+04
yr_built            -2612.6078     83.651    -31.232      0.000   -2776.573   -2448.643
waterfront_YES       7.723e+05   2.22e+04     34.833      0.000    7.29e+05    8.16e+05
renovated_renovated  4.939e+04   1.01e+04      4.876      0.000    2.95e+04    6.92e+04
==============================================================================
Omnibus:                     9723.971   Durbin-Watson:                   1.978
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           360515.689
Skew:                           2.412   Prob(JB):                         0.00
Kurtosis:                      25.982   Cond. No.                     5.51e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.51e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

- The centred model is statistically significant as standard alpha of 0.05 being 0.0, and still explains 58% of the variance in sale price.
- The centred model still explains the same amount of variance as the multiple model
- The model coefficients are statistically significant except for "sqft_lot".
- The constant here explains that for an average house that is not renovated, with no waterfront and built since 1900, we would have a sale price of 711,600 dollars

Transformations

- considering our predictor scatterplots above, we would need to transform them to follow the assumption of linearity in L.I.N.E

In [44]:
```
# Transformations
log_cols = ['sqft_lot', 'sqft_lot15', 'sqft_living']
unlog_cols = ['bedrooms', 'condition', 'floors', 'sqft_basement', 'yr_built', 'waterfront_YES', 'renovated_ren
X_log = X_multiple[log_cols]
for col in X_log.columns:
 X_log[col] = X_log[col] - np.log(X_log[col])
X_log = pd.concat([X_log, X_multiple[unlog_cols]], axis=1)
X_log
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_2932\30814496.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  X_log[col] = X_log[col] - np.log(X_log[col])
```

Out[44]:

	sqft_lot	sqft_lot15	sqft_living	bedrooms	condition	floors	sqft_basement	yr_built	waterfront_YES	renovated_ren
1	7233.112347	7630.058978	2562.148339	3	3	2.0	400.0	1951	0	
3	4991.482807	4991.482807	1952.419300	4	5	1.0	910.0	1965	0	
4	8071.002853	7494.076942	1672.573451	3	3	1.0	0.0	1987	0	
5	101918.467958	101918.467958	5411.402149	4	3	1.0	1530.0	2001	0	
6	6810.172532	6810.172532	1707.552832	3	3	2.0	0.0	1995	0	
...	...	...	...	...	...	...	...	...	...	
21591	1286.834507	1257.857173	1302.822218	3	3	2.0	130.0	2008	0	
21592	1123.969143	1501.680798	1522.666977	3	3	3.0	0.0	2009	0	
21593	5804.332148	7191.118164	2302.254997	4	3	2.0	0.0	2014	0	
21594	1342.792140	1999.395604	1013.072442	2	3	2.0	0.0	2009	0	
21596	1069.018994	1349.786968	1013.072442	2	3	2.0	0.0	2008	0	

15691 rows × 10 columns

In [45]:
```
# Fitting our log model
log_results = model_results(y, X_log)
print(log_results.summary())
```

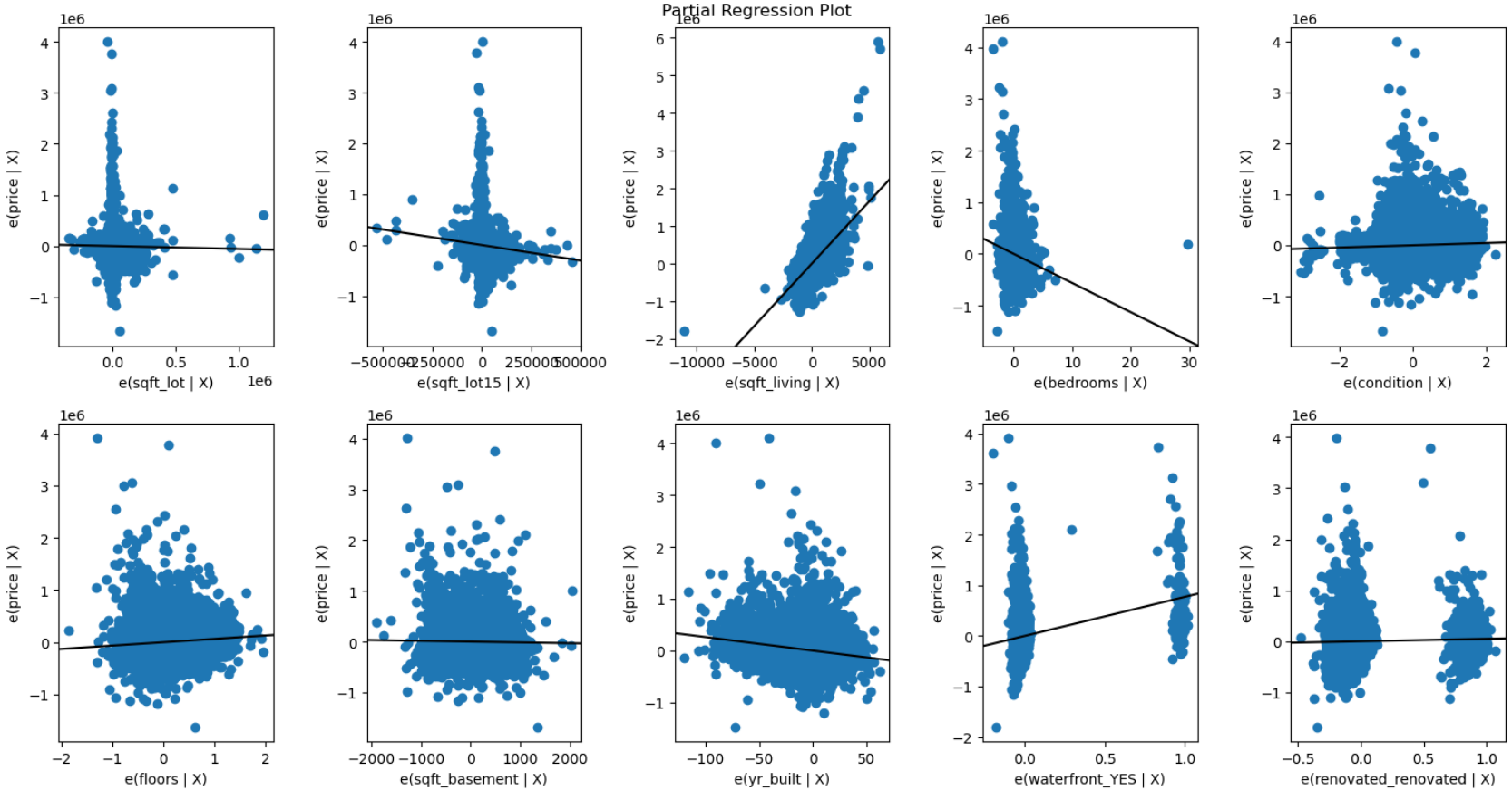
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.592
Model:                            OLS   Adj. R-squared:                  0.592
Method:                 Least Squares   F-statistic:                     2277.
Date:                Wed, 05 Jul 2023   Prob (F-statistic):               0.00
Time:                        12:51:27   Log-Likelihood:            -2.1659e+05
No. Observations:               15691   AIC:                         4.332e+05
Df Residuals:                   15680   BIC:                         4.333e+05
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const                5.026e+06   1.66e+05     30.305      0.000     4.7e+06    5.35e+06
sqft_lot               -0.0567      0.066     -0.863      0.388      -0.185       0.072
sqft_lot15             -0.6134      0.099     -6.212      0.000      -0.807      -0.420
sqft_living           332.0998      3.229    102.845      0.000     325.770     338.429
bedrooms            -5.655e+04   2527.545    -22.372      0.000   -6.15e+04   -5.16e+04
condition            2.199e+04   3238.654      6.789      0.000    1.56e+04    2.83e+04
floors               6.439e+04   4618.935     13.940      0.000    5.53e+04    7.34e+04
sqft_basement         -15.3514      5.531     -2.776      0.006     -26.192      -4.511
yr_built            -2612.3868     83.647    -31.231      0.000   -2776.344   -2448.430
waterfront_YES       7.722e+05   2.22e+04     34.833      0.000    7.29e+05    8.16e+05
renovated_renovated  4.939e+04   1.01e+04      4.877      0.000    2.95e+04    6.92e+04
==============================================================================
Omnibus:                     9721.839   Durbin-Watson:                   1.978
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           360259.128
Skew:                           2.411   Prob(JB):                         0.00
Kurtosis:                      25.973   Cond. No.                     4.47e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.47e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

- In the above cell, log transformations did not assist in improving improving the model. The adjusted r-squared value remained constant. This led to the need for considering other features affecting the variability in selling price

In [47]:
```
fig = plt.figure(figsize=(15,8))
sm.graphics.plot_partregress_grid(
 log_results,
 exog_idx=list(X_log.columns.values),
 grid=(2,5),
 fig=fig)
plt.show()
```

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


Partial Regression Plot
e(price | X)
e(sqft_lot | X)
e(sqft_lot15 | X)
e(sqft_living | X)
e(bedrooms | X)
e(condition | X)
e(floors | X)
e(sqft_basement | X)
e(yr_built | X)
e(waterfront_YES | X)
e(renovated_renovated | X)
1e6

In [48]:
```
# Adding 'zipcode ' column for ohe
X_multiple2 = df.copy().drop('price', axis=1)
X_multiple2 = pd.get_dummies(X_multiple2, columns=['waterfront', 'renovated', 'zipcode'], drop_first=True)
X_multiple2
```

Out[48]:

	bedrooms	sqft_living	sqft_lot	floors	condition	sqft_basement	yr_built	sqft_lot15	waterfront_YES	renovated_renovated	...	z
1	3	2570	7242	2.0	3	400.0	1951	7639	0	1	...	
3	4	1960	5000	1.0	5	910.0	1965	5000	0	0	...	
4	3	1680	8080	1.0	3	0.0	1987	7503	0	0	...	
5	4	5420	101930	1.0	3	1530.0	2001	101930	0	0	...	
6	3	1715	6819	2.0	3	0.0	1995	6819	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	...
21591	3	1310	1294	2.0	3	130.0	2008	1265	0	0	...	
21592	3	1530	1131	3.0	3	0.0	2009	1509	0	0	...	
21593	4	2310	5813	2.0	3	0.0	2014	7200	0	0	...	
21594	2	1020	1350	2.0	3	0.0	2009	2007	0	0	...	
21596	2	1020	1076	2.0	3	0.0	2008	1357	0	0	...	

15691 rows × 79 columns

In [49]:
```
# fitting the model for it
multiple_results2 = model_results(y, X_multiple2)
print(multiple_results2.summary())
```

OLS Regression Results

Dep. Variable:	price	R-squared:	0.785
Model:	OLS	Adj. R-squared:	0.784
Method:	Least Squares	F-statistic:	720.8
Date:	Wed, 05 Jul 2023	Prob (F-statistic):	0.00
Time:	12:55:54	Log-Likelihood:	-2.1157e+05
No. Observations:	15691	AIC:	4.233e+05
Df Residuals:	15611	BIC:	4.239e+05
Df Model:	79		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.817e+05	1.47e+05	-1.240	0.215	-4.69e+05	1.06e+05
bedrooms	-3.637e+04	1872.170	-19.429	0.000	-4e+04	-3.27e+04
sqft_living	297.0622	2.563	115.923	0.000	292.039	302.085
sqft_lot	0.2633	0.048	5.450	0.000	0.169	0.358
floors	-3.562e+04	3895.463	-9.144	0.000	-4.33e+04	-2.8e+04
condition	2.776e+04	2438.989	11.381	0.000	2.3e+04	3.25e+04
sqft_basement	-90.3232	4.356	-20.737	0.000	-98.861	-81.785
yr_built	-4.0584	73.493	-0.055	0.956	-148.113	139.997
sqft_lot15	-0.0643	0.074	-0.865	0.387	-0.210	0.081
waterfront_YES	8.836e+05	1.66e+04	53.333	0.000	8.51e+05	9.16e+05
renovated_renovated	5.143e+04	7445.474	6.908	0.000	3.68e+04	6.6e+04
zipcode_98002	1.524e+04	1.8e+04	0.845	0.398	-2.01e+04	5.06e+04
zipcode_98003	6191.1424	1.61e+04	0.384	0.701	-2.54e+04	3.78e+04
zipcode_98004	8.004e+05	1.6e+04	50.018	0.000	7.69e+05	8.32e+05
zipcode_98005	3.307e+05	1.89e+04	17.480	0.000	2.94e+05	3.68e+05
zipcode_98006	3.218e+05	1.45e+04	22.269	0.000	2.93e+05	3.5e+05
zipcode_98007	2.68e+05	2.05e+04	13.067	0.000	2.28e+05	3.08e+05
zipcode_98008	2.826e+05	1.62e+04	17.437	0.000	2.51e+05	3.14e+05
zipcode_98010	4.154e+04	2.38e+04	1.749	0.080	-5016.755	8.81e+04
zipcode_98011	1.228e+05	1.84e+04	6.681	0.000	8.68e+04	1.59e+05
zipcode_98014	7.625e+04	2.16e+04	3.530	0.000	3.39e+04	1.19e+05
zipcode_98019	6.094e+04	1.9e+04	3.211	0.001	2.37e+04	9.81e+04
zipcode_98022	8406.6537	1.74e+04	0.483	0.629	-2.57e+04	4.26e+04
zipcode_98023	-1.724e+04	1.42e+04	-1.210	0.226	-4.52e+04	1.07e+04
zipcode_98024	1.375e+05	2.55e+04	5.400	0.000	8.76e+04	1.87e+05
zipcode_98027	1.748e+05	1.49e+04	11.762	0.000	1.46e+05	2.04e+05
zipcode_98028	1.261e+05	1.62e+04	7.772	0.000	9.43e+04	1.58e+05
zipcode_98029	2.339e+05	1.56e+04	14.974	0.000	2.03e+05	2.65e+05
zipcode_98030	-2422.1255	1.68e+04	-0.144	0.886	-3.54e+04	3.06e+04

zipcode_98031	1.386e+04	1.64e+04	0.844	0.399	-1.83e+04	4.61e+04
zipcode_98032	1.37e+04	2.07e+04	0.661	0.509	-2.69e+04	5.43e+04
zipcode_98033	3.894e+05	1.47e+04	26.486	0.000	3.61e+05	4.18e+05
zipcode_98034	2.249e+05	1.4e+04	16.051	0.000	1.97e+05	2.52e+05
zipcode_98038	2.264e+04	1.38e+04	1.641	0.101	-4396.382	4.97e+04
zipcode_98039	1.353e+06	3.13e+04	43.228	0.000	1.29e+06	1.41e+06
zipcode_98040	5.692e+05	1.65e+04	34.440	0.000	5.37e+05	6.02e+05
zipcode_98042	-3215.0460	1.39e+04	-0.231	0.818	-3.05e+04	2.41e+04
zipcode_98045	1.033e+05	1.76e+04	5.867	0.000	6.88e+04	1.38e+05
zipcode_98052	2.483e+05	1.39e+04	17.860	0.000	2.21e+05	2.76e+05
zipcode_98053	1.773e+05	1.51e+04	11.766	0.000	1.48e+05	2.07e+05
zipcode_98055	5.254e+04	1.66e+04	3.170	0.002	2.01e+04	8.5e+04
zipcode_98056	9.488e+04	1.48e+04	6.404	0.000	6.58e+04	1.24e+05
zipcode_98058	2.654e+04	1.46e+04	1.816	0.069	-2103.955	5.52e+04
zipcode_98059	7.884e+04	1.46e+04	5.393	0.000	5.02e+04	1.07e+05
zipcode_98065	7.076e+04	1.61e+04	4.399	0.000	3.92e+04	1.02e+05
zipcode_98070	-1.613e+04	2.22e+04	-0.727	0.467	-5.96e+04	2.74e+04
zipcode_98072	1.595e+05	1.64e+04	9.711	0.000	1.27e+05	1.92e+05
zipcode_98074	2.021e+05	1.49e+04	13.580	0.000	1.73e+05	2.31e+05
zipcode_98075	1.893e+05	1.54e+04	12.309	0.000	1.59e+05	2.19e+05
zipcode_98077	1.298e+05	1.8e+04	7.212	0.000	9.45e+04	1.65e+05
zipcode_98092	-2.916e+04	1.54e+04	-1.894	0.058	-5.93e+04	1013.557
zipcode_98102	6.009e+05	2.46e+04	24.441	0.000	5.53e+05	6.49e+05
zipcode_98103	3.744e+05	1.42e+04	26.379	0.000	3.47e+05	4.02e+05
zipcode_98105	5.198e+05	1.75e+04	29.692	0.000	4.86e+05	5.54e+05
zipcode_98106	1.654e+05	1.59e+04	10.412	0.000	1.34e+05	1.96e+05
zipcode_98107	4.036e+05	1.65e+04	24.410	0.000	3.71e+05	4.36e+05
zipcode_98108	1.402e+05	1.87e+04	7.475	0.000	1.03e+05	1.77e+05
zipcode_98109	5.326e+05	2.31e+04	23.091	0.000	4.87e+05	5.78e+05
zipcode_98112	6.96e+05	1.67e+04	41.624	0.000	6.63e+05	7.29e+05
zipcode_98115	3.682e+05	1.42e+04	25.978	0.000	3.4e+05	3.96e+05
zipcode_98116	3.515e+05	1.57e+04	22.373	0.000	3.21e+05	3.82e+05
zipcode_98117	3.529e+05	1.42e+04	24.801	0.000	3.25e+05	3.81e+05
zipcode_98118	1.987e+05	1.44e+04	13.774	0.000	1.7e+05	2.27e+05
zipcode_98119	5.696e+05	1.89e+04	30.092	0.000	5.32e+05	6.07e+05
zipcode_98122	3.873e+05	1.66e+04	23.302	0.000	3.55e+05	4.2e+05
zipcode_98125	2.194e+05	1.5e+04	14.639	0.000	1.9e+05	2.49e+05
zipcode_98126	2.367e+05	1.57e+04	15.111	0.000	2.06e+05	2.67e+05
zipcode_98133	1.744e+05	1.45e+04	12.032	0.000	1.46e+05	2.03e+05
zipcode_98136	3.022e+05	1.67e+04	18.125	0.000	2.69e+05	3.35e+05
zipcode_98144	3.244e+05	1.57e+04	20.709	0.000	2.94e+05	3.55e+05
zipcode_98146	1.156e+05	1.63e+04	7.072	0.000	8.35e+04	1.48e+05
zipcode_98148	6.342e+04	2.9e+04	2.184	0.029	6513.079	1.2e+05

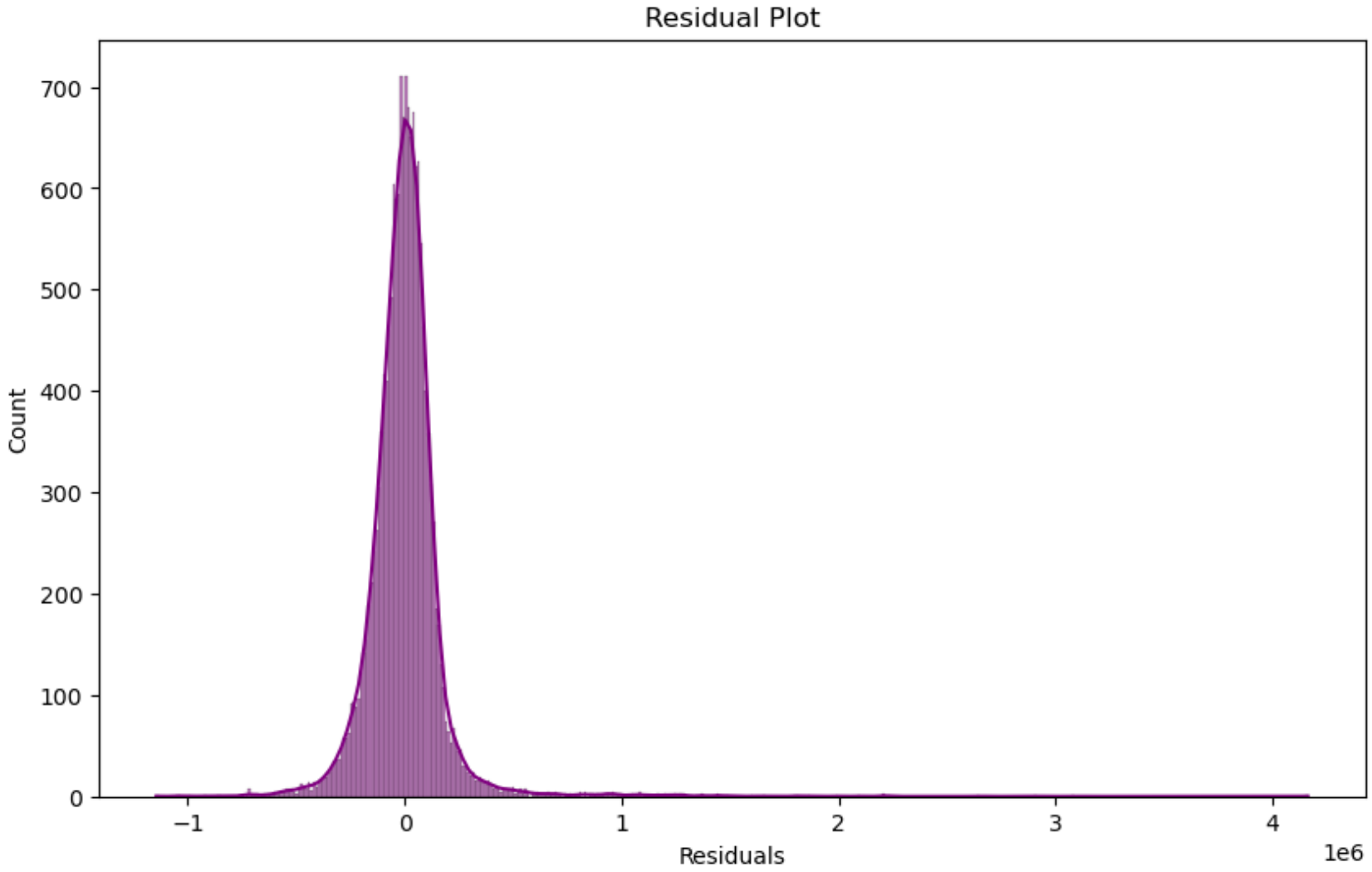
zipcode_98155	1.615e+05	1.48e+04	10.918	0.000	1.33e+05	1.91e+05
zipcode_98166	8.001e+04	1.7e+04	4.706	0.000	4.67e+04	1.13e+05
zipcode_98168	6.857e+04	1.68e+04	4.081	0.000	3.56e+04	1.01e+05
zipcode_98177	2.891e+05	1.68e+04	17.168	0.000	2.56e+05	3.22e+05
zipcode_98178	5.962e+04	1.69e+04	3.524	0.000	2.65e+04	9.28e+04
zipcode_98188	4.079e+04	2.08e+04	1.960	0.050	-6.217	8.16e+04
zipcode_98198	2.941e+04	1.64e+04	1.788	0.074	-2827.183	6.17e+04
zipcode_98199	4.558e+05	1.62e+04	28.089	0.000	4.24e+05	4.88e+05

Omnibus:	14197.463	Durbin-Watson:	1.985
Prob(Omnibus):	0.000	Jarque-Bera (JB):	2051666.653
Skew:	3.831	Prob(JB):	0.00
Kurtosis:	58.492	Cond. No.	5.47e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.47e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

In [50]:
```
fig, ax = plt.subplots(figsize=(10,6))
sns.histplot(bins='auto', x=multiple_results2.resid, kde=True, color='purple');
plt.xlabel('Residuals')
plt.title("Residual Plot")
```

Out[50]:
```
Text(0.5, 1.0, 'Residual Plot')
```



In [52]:
```
multiple_results2_df = pd.concat([multiple_results2.params, multiple_results2.pvalues], axis=1)
multiple_results2_df.columns = ["coefficient", "pvalue"]
multiple_results2_df
```

Out[52]:

	coefficient	pvalue
const	-181738.183574	2.151542e-01
bedrooms	-36373.972664	4.214033e-83
sqft_living	297.062217	0.000000e+00
sqft_lot	0.263302	5.124160e-08
floors	-35620.938932	6.733407e-20
...	...	...
zipcode_98177	289092.833436	1.850873e-65
zipcode_98178	59622.570580	4.267632e-04
zipcode_98188	40786.194894	5.003492e-02
zipcode_98198	29413.406801	7.375780e-02
zipcode_98199	455752.956232	2.100359e-169

80 rows × 2 columns

In [53]:
```
# Checking those that are not statistically significant
multiple_results2_df[multiple_results2_df["pvalue"] > 0.05]
```

Out[53]:

	coefficient	pvalue
const	-181738.183574	0.215154
yr_built	-4.058373	0.955963
sqft_lot15	-0.064297	0.387147
zipcode_98002	15243.424317	0.398111
zipcode_98003	6191.142442	0.701274
zipcode_98010	41543.916947	0.080324
zipcode_98022	8406.653651	0.629386
zipcode_98023	-17236.943719	0.226217
zipcode_98030	-2422.125527	0.885589
zipcode_98031	13862.952382	0.398584
zipcode_98032	13700.824353	0.508725
zipcode_98038	22636.105518	0.100748
zipcode_98042	-3215.046030	0.817604
zipcode_98058	26536.207194	0.069371
zipcode_98070	-16126.953241	0.467311
zipcode_98092	-29158.379313	0.058208
zipcode_98188	40786.194894	0.050035
zipcode_98198	29413.406801	0.073758

Our reference category is 98001 Auburn, Washington hence these zipcodes means that there is no significant differences in these areas compared to Auburn.

Findings

- The multiple linear regression model has an R-squared value of 0.79, which indicates that the model can explain 79% of the variance of the market house sale prices which is a good sign that the model is effective in predicting the prices
- The waterfront view was seen to be most impactful with houses having a waterfront view having value increase of about 834,000 dollars more than those without a waterfront.This can be inferred for the other variables as well
- For an average house that is not renovated, with no waterfront and built since 1900, we would have a sale price of 711,600 dollars
- The combination of square footage of the living area, bedrooms, floors, and whether the property had a waterfront view or was renovated are the most reliable predictors of a house's price in King County.

Recommendations

- We recommend for customers on a budget should look out for houses situated on higher property floors as we expect a price drop on said houses of about 30,000 dollars for every floor increase
- Development of a comprehensive database the agency can use to track the renovation projects that would improve property value.
- For customers who would like to sell their properties/houses, it is recommended that they should renovate them first as we see these would be an increase in value by about 48,000 dollars

CONCLUSION

- The combination of square footage of the living area, bedrooms, floors, and whether the property had a waterfront view or was renovated are the most reliable predictors of a house's price in King County.
- There were some limitations to the model. To meet regression assumptions, we had to try out log-transformation on certain variables. Therefore,any new data used with the model would require similar preprocessing. Additionally, since housing prices vary regionally, the model's usefulness for data from other counties may be restricted.
- If you are seeking affordable housing, it may be advisable to compromise on square footage and have no waterfront view. But, given that many urban residents already do this, it may not be a viable solution for everyone

Next steps

- Using datasets from other counties to be able to better advice our customers from comparing the dataset results.

- The agency may prepare questionnaires to identify their strengths, weaknesses, opportunities and threats and use this information to prioritize recommendations that would help address their weaknesses and take advantge of their opportunities and strengths.
- It is also important for the agency to continuously monitor the effectiveness of the strategies they implement and make adjustments as necessary. This could involve tracking metrics like website traffic, this model, social media engagement, and lead generation to assess the impact of their efforts and identify areas for improvement.

Type *Markdown* and LaTeX: α^2