Final Project Submission

Please fill out:

- Student name: Joel Omondi, Aaron Onserio, James Mbeti, Victoria Nabea, Nyokabi Waiganjo, Kennedy Juma
- Student pace: full time
- Scheduled project review date/time: 20/04/2023
- Instructor name: Lucille Kaleha, Nikita Njoroge, Samuel Karu
- GitHUb repository URL: https://github.com/Ed-Odhiambo/dsc-phase-2-project-v2-3.git (https://github.com/Ed-Odhiambo/dsc-phase-2-project-v2-3.git)
- Trello URL: https://trello.com/b/5UZDcfIc/king-county-house-sales (https://trello.com/b/5UZDcfIc/king-county-house-sales)

VALUE ANALYSIS FOR HOUSE SALE PRICES IN KING COUNTY

Business Overview

Will your home renovations pay off?

Introduction

- In this project, the task is to undertake research on property market in King County and to determine the major factors for the houses within the neighbourhoods for a real estate agency.
- The objective of this project is to utilize multiple linear regression modeling to evaluate house sales data in King County Washington, USA. The aim is to gain insights and make predictions about the factors that affect house sales in the area as well as lucrative neighbourhoods to invest in while using statistical techniques to support relevant recommendations.
- Remodeling certain areas of a property is an excellent way for homeowners to add increased functionality and beauty to a property at someone else's expense. By choosing the right project to enhance your living space, a significant portion of the expense can be passed on to future owners in the form of increased property values

Business Problem / Problem Statement

The real estate agency needs to provide advice to homeowners regarding how home renovations can potentially increase the estimated value of their properties and homes and by what amount. This information will assist the real estate agency in advising their clients about how to make informed decisions regarding home renovations, and in turn, assist homeowners in maximizing the return on their investment when selling their properties.

Proposal

- We will focus on analysing house sales in a northwestern county for a possible stakeholder, a real estate agency helping homeowners buy and/or sell homes. Specifically, we will address the business problem of providing homeowners advice on how home renovations can increase the estimated value of their homes and by what amount.
-

Data Understanding

- This project uses the King County House Sales dataset, which can be found in "kc_house_data.csv"
- One of the challenges that may arise in this project is the incomplete or ambiguous description of the column names in the dataset. However, with thorough research or good judgment, we can understand the data and make informed decisions on which variables to use in our analysis.
- Another challenge is ensuring that the linear regression model we develop adds value to our analysis, rather than simply fulfilling the project's requirement.
- The dataset includes information on house sales in King County such as the prices, the design, the size in square footages, the location etc. Others can be found at 'Property Schema' (https://github.com/learn-co-curriculum/dsc-phase-2-project-v2-3/blob/main/data/column_names.md)

Objectives

- To determine the increase in prices of property due to renovations.
- To define a clear stakeholder and business problem that relates to the King County House Sales dataset, and to use this problem to guide the analysis and modelling process.
- To build and evaluate multiple linear regression models using various combinations of the available features in the dataset, with the aim of identifying the most relevant and impactful features for predicting house sale prices.
- To use appropriate statistical techniques to refine and optimize the regression models, and to clearly explain the rationale behind each technique used.
- To demonstrate an iterative approach to modelling, documenting each stage of the process and providing justification for each model modification .
- To provide a final documentation that includes a detailed analysis of the chosen model, including relevant metrics describing overall model performance and the coefficients of the most impactful features, and a clear explanation of how the model adds value to the stakeholder and the appropriate recommendations.

Metric of Success

The final step in evaluating the quality of the model is cross-validation, which gives us an idea of how the model would perform with new data for the same variables. one that the model will be trained on, and another that it will be tested on. By default, the function takes 75% of the data as the training subset and the other 25% as its test subset.

Data Preparation

In [1]:

```
# Importing relevant libraries

# Pandas for reading the data and analysis
import pandas as pd

# Numpy for numbers
import numpy as np

# Seaborn and Matplotlib for Visualization
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.cm as cm
%matplotlib nbagg
%matplotlib inline

# Statsmodels and Scipy for Statistical Analysis
from statsmodels.stats.power import TTestIndPower, TTestPower
import statsmodels.api as sm
import statsmodels.formula.api as smf
from scipy import stats
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Warnings to remove warnings
import warnings
warnings.filterwarnings("ignore")
```

Reading the data

- The below code reads the CSV file 'kc_house_data.csv' using the pandas library. The data is loaded into a pandas DataFrame object named 'df'. It then views the first few rows.
- The cells that follow check for overview information on the data and then calls for descriptive statistics of the dataset.
- Other important checks include checking the shape of the data, checking the columns and checking for duplicated entries as well as drops them permanently from the dataset.

In [2]:

```
# Viewing the data
def read_data(path):
    """A simple function to read our data"""
    data = pd.read_csv(path)

    return data


df = read_data('kc_house_data.csv')
df
```

Out[2]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_ba
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	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	NO	NONE	...	8 Good	1530	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	NO	NONE	...	8 Good	2310	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	NO	NONE	...	7 Average	1020	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	NaN	NONE	...	8 Good	1600	
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	2.0	NO	NONE	...	7 Average	1020	

21597 rows × 21 columns

In [3]:

```
# Validating the columns
def read_columns(data):
    """A simple function to display the columns"""
    print(f"Columns included in the dataset are: {list(data.columns)}")

read_columns(df)
```

```
Columns included in the dataset are: ['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sq
ft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above', 'sqft_basement', 'yr_buil
t', 'yr_renovated', 'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15']
```

Getting information about the data

In [4]:

```
# Getting the shape and info of the data
def get_info_shape(data):
    """A simple function to get the general information of the data"""

    print(data.info())
    print('++++++++++++++++++++++++++++++++++++++++++++++++++')
    print('++++++++++++++++++++++++++++++++++++++++++++++++++')
    print(f"""
          The shape the dataset is: {data.shape}
          with number of columns as {data.shape[0]}
          and the number of rows as {data.shape[1]}
          """)

get_info_shape(df)
```

```
<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
None
++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++

          The shape the dataset is: (21597, 21)
          with number of columns as 21597
          and the number of rows as 21
```

Checking and dropping duplicated Values

In [5]:

```
# Checking for duplicated entry percentages
def check_duplicates(data):
    """A simple function to check for duplicated entry percentages in the data"""
    freq_dict = {}
    for elem in data:
        if elem in freq_dict:
            freq_dict[elem] += 1
        else:
            freq_dict[elem] = 1

    num_duplicates = 0
    for freq in freq_dict.values():
        if freq > 1:
            num_duplicates += 1

    percentage_duplicates = (num_duplicates / len(data)) * 100

    return percentage_duplicates

print(f"""
      We have {check_duplicates(df)}% of duplicates in our dataset
      """)
print(f"""
      We have a total of {df['id'].duplicated().sum()} duplicates out of {df.shape[0]} entries
      which corresponds to {round(check_duplicates(df['id']),2)}% of duplicates in the house unique identifier 'id'
      """)
```

```
      We have 0.0% of duplicates in our dataset


      We have a total of 177 duplicates out of 21597 entries
      which corresponds to 0.81% of duplicates in the house unique identifier 'id'
```

The column "id" is a unique identifier for a house and thus we drop those that are duplicated and retain the first entry. Inplace= True is added to ensure the change is carried foward when the data is called again

- **This is confirmed using the `.shape` method**

In [6]:

```
# Permanently dropping duplicates from the subset "id"
def drop_duplicates(data, subset):
    """A simple function to drop duplicates in a subset"""
    print(data.drop_duplicates(subset=subset, keep='first', inplace=True))

drop_duplicates(df, "id")
```

```
None
```

In [7]:

```
# confirming dropped rows using the shape
get_info_shape(df)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21420 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21420 non-null  int64  
 1   date           21420 non-null  object 
 2   price          21420 non-null  float64
 3   bedrooms       21420 non-null  int64  
 4   bathrooms      21420 non-null  float64
 5   sqft_living    21420 non-null  int64  
 6   sqft_lot       21420 non-null  int64  
 7   floors         21420 non-null  float64
 8   waterfront     19067 non-null  object 
 9   view           21357 non-null  object 
 10  condition      21420 non-null  object 
 11  grade          21420 non-null  object 
 12  sqft_above     21420 non-null  int64  
 13  sqft_basement  21420 non-null  object 
 14  yr_built       21420 non-null  int64  
 15  yr_renovated   17616 non-null  float64
 16  zipcode        21420 non-null  int64  
 17  lat            21420 non-null  float64
 18  long           21420 non-null  float64
 19  sqft_living15  21420 non-null  int64  
 20  sqft_lot15     21420 non-null  int64  
dtypes: float64(6), int64(9), object(6)
memory usage: 3.6+ MB
None
++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++

          The shape the dataset is: (21420, 21)
          with number of columns as 21420
          and the number of rows as 21
```

Checking and Dropping Null Values

In [8]:

```
# Checking for null values 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 [9]:

```
# Confirming the dropped columns
get_info_shape(df)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15691 entries, 1 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             15691 non-null  int64  
 1   date           15691 non-null  object 
 2   price          15691 non-null  float64
 3   bedrooms       15691 non-null  int64  
 4   bathrooms      15691 non-null  float64
 5   sqft_living    15691 non-null  int64  
 6   sqft_lot       15691 non-null  int64  
 7   floors         15691 non-null  float64
 8   waterfront     15691 non-null  object 
 9   view           15644 non-null  object 
 10  condition      15691 non-null  object 
 11  grade          15691 non-null  object 
 12  sqft_above     15691 non-null  int64  
 13  sqft_basement  15691 non-null  object 
 14  yr_built       15691 non-null  int64  
 15  yr_renovated   15691 non-null  float64
 16  zipcode        15691 non-null  int64  
 17  lat            15691 non-null  float64
 18  long           15691 non-null  float64
 19  sqft_living15  15691 non-null  int64  
 20  sqft_lot15     15691 non-null  int64  
dtypes: float64(6), int64(9), object(6)
memory usage: 2.6+ MB
None
++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++

          The shape the dataset is: (15691, 21)
          with number of columns as 15691
          and the number of rows as 21
```

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

EDA

In [10]:

```
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
7/4/2014        1
```

In [11]:

```
df = df[df["bedrooms"] < 33]
```

In [12]:

```
df.condition.value_counts()
```

Out[12]:

```
Average      10170
Good          4132
Very Good     1244
Fair           125
Poor            19
Name: condition, dtype: int64
```

- *In the cell below we convert the condition ratings into a discrete variable.*
- The changes show from 1 to 5, where the lowest has `1 as Poor` , with a midpoint having `3 as average` , and the highest being `5 as Very Good`

In [13]:

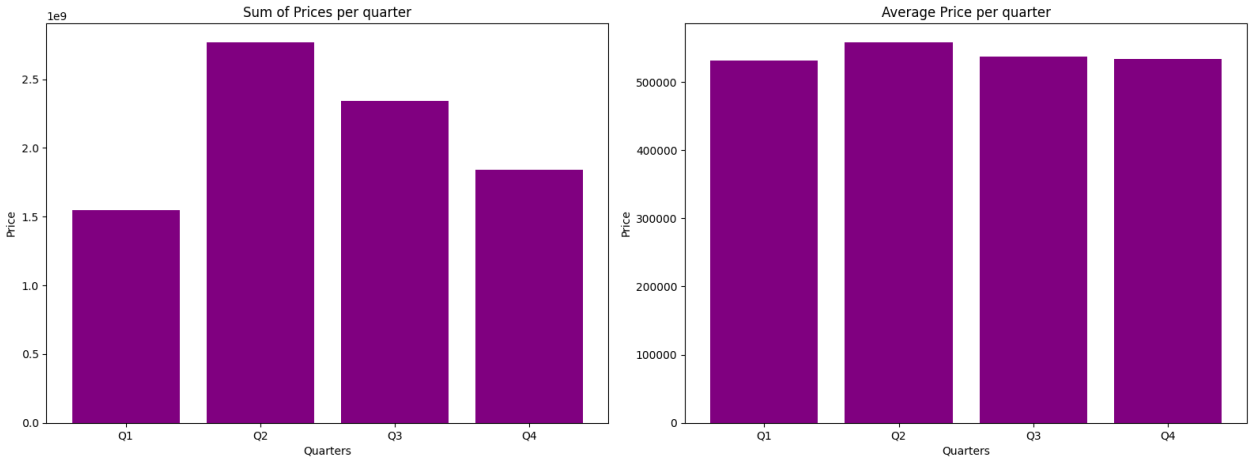
```
to_replace = {"Poor": 1, "Fair": 2,"Average": 3, "Good": 4, "Very Good": 5}
df.condition = df.condition.map(to_replace)
df.condition.value_counts()
```

Out[13]:

```
3    10170
4     4132
5     1244
2      125
1       19
Name: condition, dtype: int64
```

In [14]:

```
# 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 that may not be useful in our evaluation and we thus drop them by first initializing the list called dropped and then dropping them from our dataset***

In [15]:

```
# 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 having `7 as average`, and the highest being `11 as Mansion`***

In [16]:

```
# Here we are checking the grading system used in the dataset
df['grade'].value_counts()
```

Out[16]:

```
7 Average        6503
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 [17]:

```
# 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 [18]:

```
# 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"***

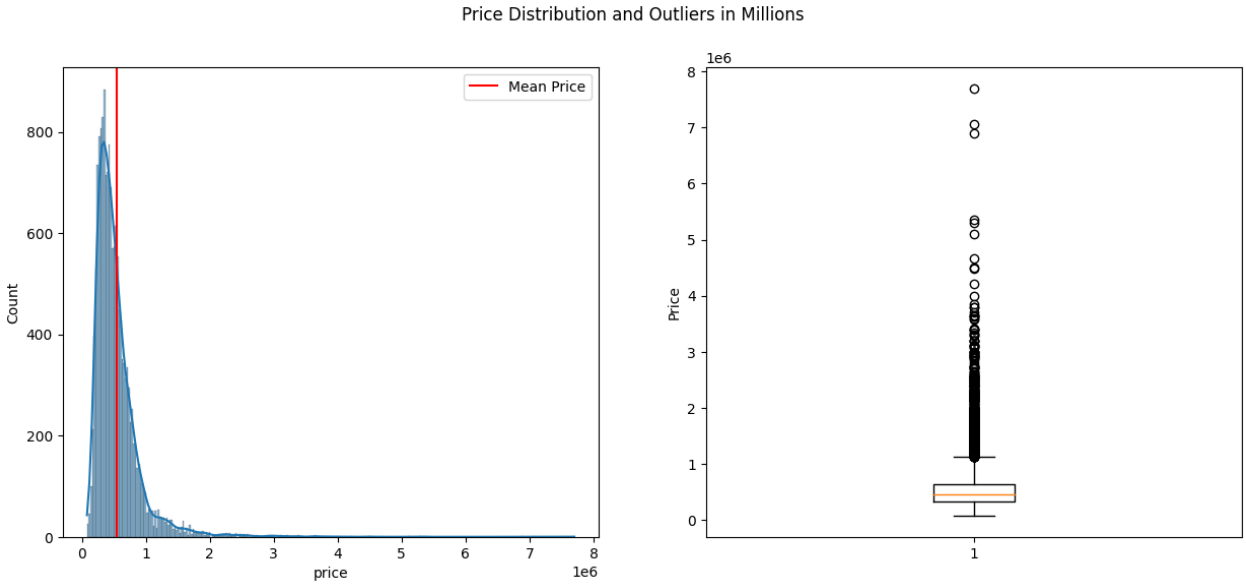
In [19]:

```
# 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 our price column

In [20]:

```
# 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 [21]:

```
# Checking the shape before the change
print(f'Before dropping outliers: {df.shape}')

# Dropping outliers
df = df.loc[df['price'] < 5_000_000]

# Confirming the changes done
print(f'After dropping outliers: {df.shape}')
```

```
Before dropping outliers: (15690, 16)
After dropping outliers: (15684, 16)
```

In [22]:

```
# Getting descriptive statistics on our dataset
df.describe()
```

Out[22]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	condition	grade	sqft_above	sqft_ba
count	1.568400e+04	15684.000000	15684.000000	15684.000000	1.568400e+04	15684.000000	15684.000000	15684.000000	15684.000000	15684
mean	5.396884e+05	3.377200	2.121270	2084.560954	1.530694e+04	1.496685	3.411566	5.666220	1792.999044	285
std	3.569768e+05	0.903297	0.762768	908.912907	4.198147e+04	0.539714	0.651190	1.168288	822.243155	439
min	8.200000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.000000	1.000000	1.000000	370.000000	0
25%	3.213818e+05	3.000000	1.750000	1430.000000	5.046750e+03	1.000000	3.000000	5.000000	1200.000000	0
50%	4.500000e+05	3.000000	2.250000	1920.000000	7.599500e+03	1.500000	3.000000	5.000000	1570.000000	0
75%	6.446250e+05	4.000000	2.500000	2550.000000	1.070125e+04	2.000000	4.000000	6.000000	2220.000000	540
max	4.670000e+06	11.000000	8.000000	13540.000000	1.651359e+06	3.500000	5.000000	11.000000	9410.000000	4820

In [23]:

```
# Sowcasing 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())} respectively
      """)
```

```
      The average price in King County is 539688.37,
      it has the hihgest price and lowest price being 4670000 and 82000 respectively
```

Modelling

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

In [24]:

```
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 [25]:

```
# 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 [26]:

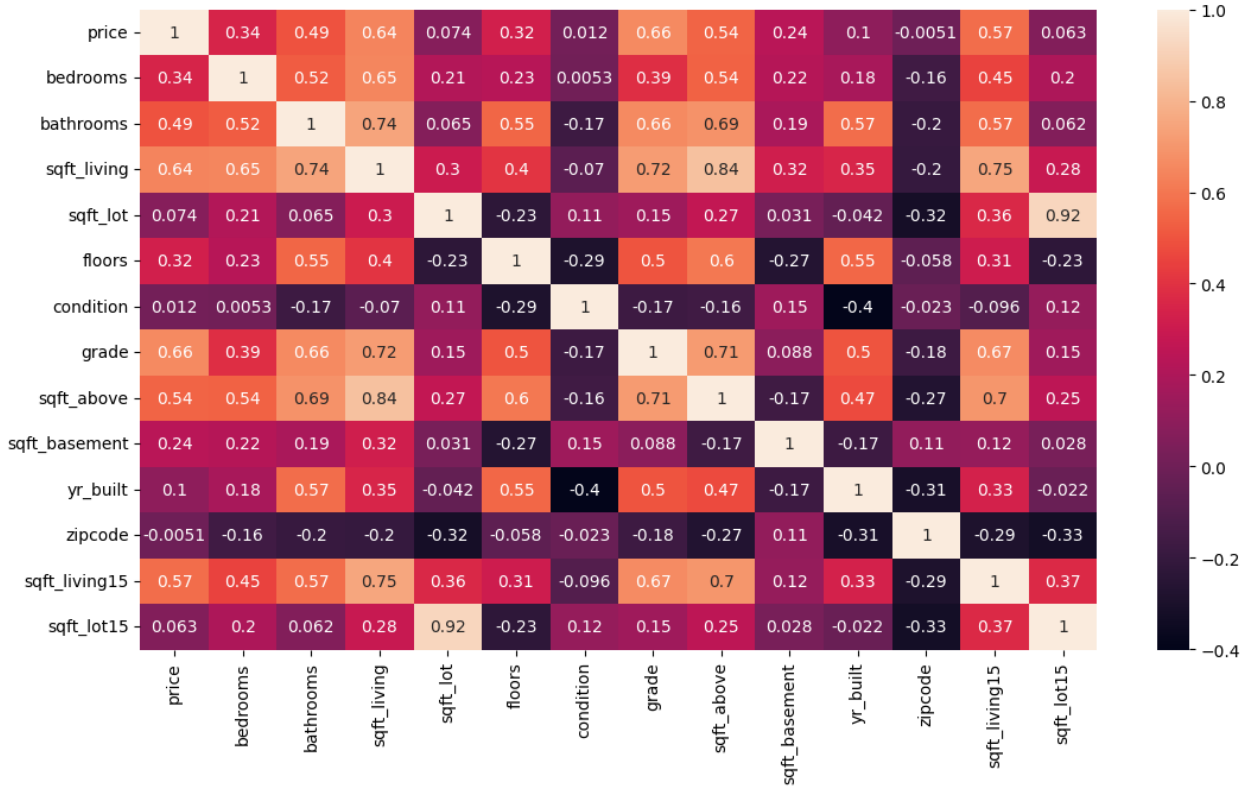
```
# Veiwing the highly correlated predictor pairs
corr_df[(corr_df.corr_coef > 0.75) & (corr_df.corr_coef < 1)]
```

Out[26]:

pairs	corr_coef
(sqft_above, sqft_living)	0.874010
(sqft_living, sqft_above)	0.874010
(sqft_living, grade)	0.764519
(grade, sqft_living)	0.764519
(sqft_living, sqft_living15)	0.758122
(sqft_living15, sqft_living)	0.758122
(grade, sqft_above)	0.757643
(sqft_above, grade)	0.757643
(sqft_living, bathrooms)	0.751433
(bathrooms, sqft_living)	0.751433

In [27]:

```
# 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 [28]:

```
# 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()
```

```
(15684, 12)
```

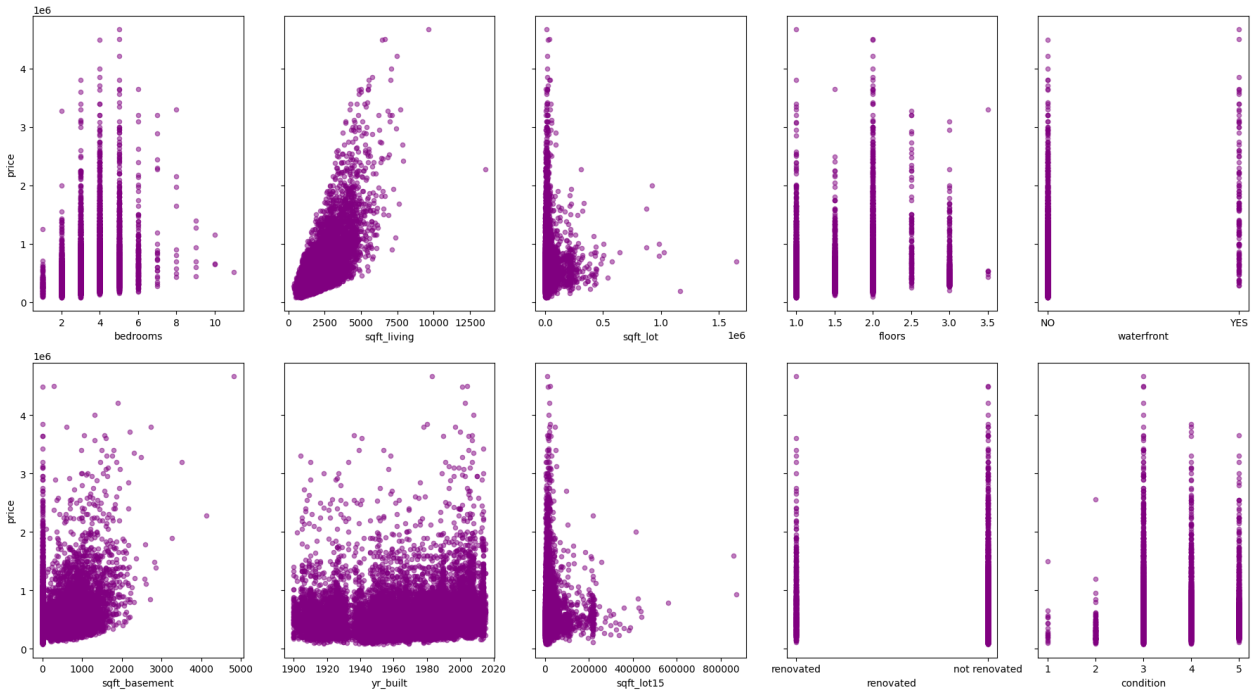
Out[28]:

	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 our baseline model

In [29]:

```
# 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='purple')

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



In [30]:

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

Out[30]:

```
price            1.000000
bedrooms         0.317642
sqft_living      0.700306
sqft_lot         0.084651
floors           0.264193
condition        0.035804
sqft_basement    0.309243
yr_built         0.051680
zipcode         -0.046585
sqft_lot15       0.079721
Name: price, dtype: float64
```

In [31]:

```
# 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:                       0.490
Model:                            OLS   Adj. R-squared:                  0.490
Method:                 Least Squares   F-statistic:                 1.509e+04
Date:                Thu, 20 Apr 2023   Prob (F-statistic):               0.00
Time:                        08:25:40   Log-Likelihood:            -2.1749e+05
No. Observations:               15684   AIC:                         4.350e+05
Df Residuals:                   15682   BIC:                         4.350e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       -3.366e+04   5091.274     -6.612      0.000   -4.36e+04   -2.37e+04
sqft_living   275.0460      2.239    122.853      0.000     270.658     279.434
==============================================================================
Omnibus:                     8750.591   Durbin-Watson:                   1.974
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           144878.111
Skew:                           2.330   Prob(JB):                         0.00
Kurtosis:                      17.141   Cond. No.                      5.69e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.69e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

- ***The model is generally statistically sigmificant at a significance level of $\alpha = 0.05$ with a p_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

15684 rows × 10 columns

In [34]:

```
# Fittng our multiple regression model
multiple_results = model_results(y, X_multiple)

print(multiple_results.summary())
```

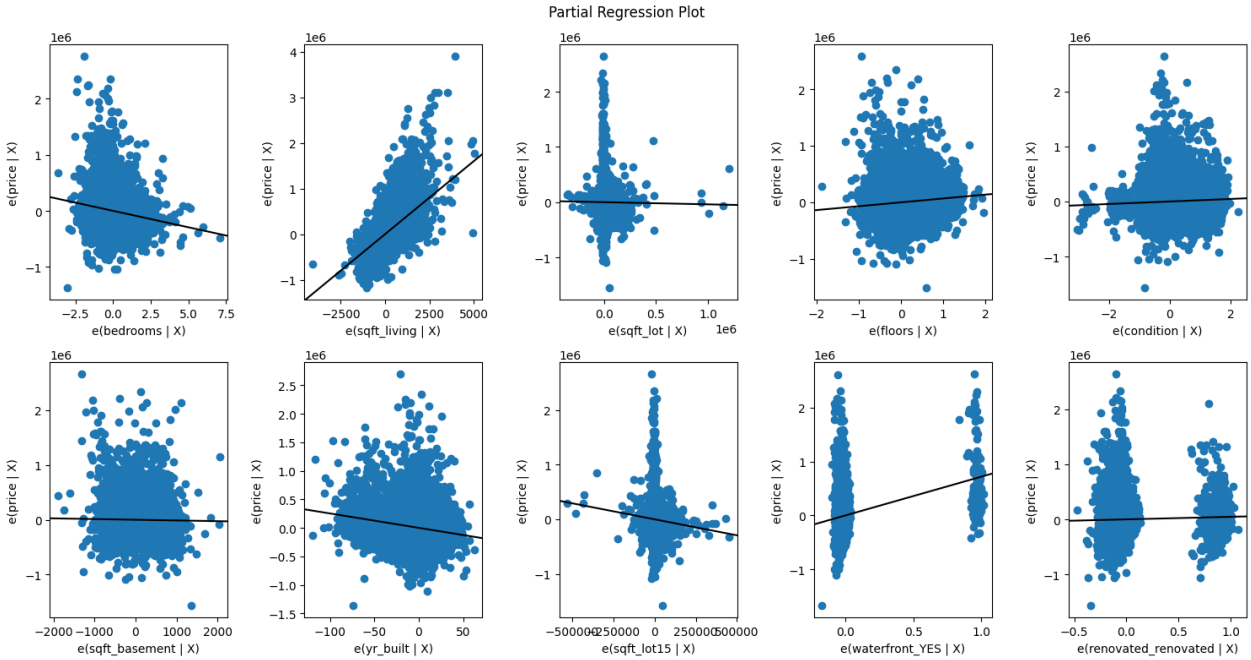
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.583
Model:                            OLS   Adj. R-squared:                  0.583
Method:                 Least Squares   F-statistic:                     2195.
Date:                Thu, 20 Apr 2023   Prob (F-statistic):               0.00
Time:                        08:25:40   Log-Likelihood:            -2.1591e+05
No. Observations:               15684   AIC:                             4.318e+05
Df Residuals:                   15673   BIC:                             4.319e+05
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
const                  4.863e+06    1.6e+05     30.405      0.000    4.55e+06    5.18e+06
bedrooms              -5.868e+04   2570.303    -22.830      0.000   -6.37e+04   -5.36e+04
sqft_living             320.7917      3.182    100.800      0.000     314.554     327.030
sqft_lot                 -0.0395      0.063     -0.623      0.533      -0.164       0.085
floors                 6.815e+04   4453.470     15.302      0.000    5.94e+04    7.69e+04
condition              2.295e+04   3121.272      7.353      0.000    1.68e+04    2.91e+04
sqft_basement           -12.8009      5.334     -2.400      0.016     -23.255      -2.347
yr_built              -2521.1812     80.673    -31.252      0.000   -2679.309   -2363.054
sqft_lot15               -0.5849      0.095     -6.144      0.000      -0.771      -0.398
waterfront_YES         7.203e+05   2.16e+04     33.346      0.000    6.78e+05    7.63e+05
renovated_renovated    4.694e+04   9766.331      4.806      0.000    2.78e+04    6.61e+04
==============================================================================
Omnibus:                     7285.348   Durbin-Watson:                   1.978
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           107832.085
Skew:                           1.850   Prob(JB):                         0.00
Kurtosis:                      15.301   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 [36]:

```
# 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
```

Out[36]:

	bedrooms	sqft_living	sqft_lot	floors	sqft_basement	sqft_lot15	condition
1	-0.3772	485.439046	-8064.94268	0.503315	114.558977	-5281.620314	-0.411566
3	0.6228	-124.560954	-10306.94268	-0.496685	624.558977	-7920.620314	1.588434
4	-0.3772	-404.560954	-7226.94268	-0.496685	-285.441023	-5417.620314	-0.411566
5	0.6228	3335.439046	86623.05732	-0.496685	1244.558977	89009.379686	-0.411566
6	-0.3772	-369.560954	-8487.94268	0.503315	-285.441023	-6101.620314	-0.411566
...	...	...	...	...	...	...	...
21591	-0.3772	-774.560954	-14012.94268	0.503315	-155.441023	-11655.620314	-0.411566
21592	-0.3772	-554.560954	-14175.94268	1.503315	-285.441023	-11411.620314	-0.411566
21593	0.6228	225.439046	-9493.94268	0.503315	-285.441023	-5720.620314	-0.411566
21594	-1.3772	-1064.560954	-13956.94268	0.503315	-285.441023	-10913.620314	-0.411566
21596	-1.3772	-1064.560954	-14230.94268	0.503315	-285.441023	-11563.620314	-0.411566

15684 rows × 7 columns

In [37]:

```
# 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[37]:

	bedrooms	sqft_living	sqft_lot	floors	sqft_basement	sqft_lot15	condition	yr_built	waterfront_YES	renovated_renovated
1	-0.3772	485.439046	-8064.94268	0.503315	114.558977	-5281.620314	-0.411566	51	0	1
3	0.6228	-124.560954	-10306.94268	-0.496685	624.558977	-7920.620314	1.588434	65	0	0
4	-0.3772	-404.560954	-7226.94268	-0.496685	-285.441023	-5417.620314	-0.411566	87	0	0
5	0.6228	3335.439046	86623.05732	-0.496685	1244.558977	89009.379686	-0.411566	101	0	0
6	-0.3772	-369.560954	-8487.94268	0.503315	-285.441023	-6101.620314	-0.411566	95	0	0
...	...	...	...	...	...	...	...	...	...	...
21591	-0.3772	-774.560954	-14012.94268	0.503315	-155.441023	-11655.620314	-0.411566	108	0	0
21592	-0.3772	-554.560954	-14175.94268	1.503315	-285.441023	-11411.620314	-0.411566	109	0	0
21593	0.6228	225.439046	-9493.94268	0.503315	-285.441023	-5720.620314	-0.411566	114	0	0
21594	-1.3772	-1064.560954	-13956.94268	0.503315	-285.441023	-10913.620314	-0.411566	109	0	0
21596	-1.3772	-1064.560954	-14230.94268	0.503315	-285.441023	-11563.620314	-0.411566	108	0	0

15684 rows × 10 columns

In [38]:

```
# fittimg model
centred_results = model_results(y, X_centred)

print(centred_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.583
Model:                            OLS   Adj. R-squared:                  0.583
Method:                 Least Squares   F-statistic:                     2195.
Date:                Thu, 20 Apr 2023   Prob (F-statistic):               0.00
Time:                        08:25:44   Log-Likelihood:            -2.1591e+05
No. Observations:               15684   AIC:                         4.318e+05
Df Residuals:                   15673   BIC:                         4.319e+05
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
const                 7.118e+05   6162.609    115.508      0.000       7e+05    7.24e+05
bedrooms             -5.868e+04   2570.303    -22.830      0.000   -6.37e+04   -5.36e+04
sqft_living            320.7917      3.182    100.800      0.000     314.554     327.030
sqft_lot                -0.0395      0.063     -0.623      0.533      -0.164       0.085
floors                6.815e+04   4453.470     15.302      0.000    5.94e+04    7.69e+04
sqft_basement           -12.8009      5.334     -2.400      0.016     -23.255      -2.347
sqft_lot15              -0.5849      0.095     -6.144      0.000      -0.771      -0.398
condition             2.295e+04   3121.272      7.353      0.000    1.68e+04    2.91e+04
yr_built             -2521.1812     80.673    -31.252      0.000   -2679.309   -2363.054
waterfront_YES        7.203e+05   2.16e+04     33.346      0.000    6.78e+05    7.63e+05
renovated_renovated   4.694e+04   9766.331      4.806      0.000    2.78e+04    6.61e+04
==============================================================================
Omnibus:                     7285.348   Durbin-Watson:                   1.978
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           107832.085
Skew:                           1.850   Prob(JB):                         0.00
Kurtosis:                      15.301   Cond. No.                     5.58e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.58e+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***

In [39]:

```
# Standardizion of our model
X_std = X_multiple[numeric_cols]

scale = StandardScaler().fit(X_std)
X_stand= scale.transform(X_std)

X_stand = pd.DataFrame(X_stand, columns=X_std.columns, index=X_std.index)
X_stand = pd.concat([X_stand, X_multiple[categorical_cols]], axis=1)


model_stand_results = model_results(y, X_stand)

model_stand_results.params.abs().sort_values(ascending=False)
```

Out[39]:

```
const                 5.502072e+06
waterfront_YES        7.202706e+05
sqft_living           2.915624e+05
bedrooms              5.300277e+04
renovated_renovated   4.694103e+04
floors                3.677852e+04
sqft_lot15            1.641785e+04
condition             1.494524e+04
sqft_basement         5.629306e+03
yr_built              2.521181e+03
sqft_lot              1.656430e+03
dtype: float64
```

Transformations

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

In [40]:

```
# Transformations
log_cols = ['sqft_lot', 'sqft_lot15', 'sqft_living']
unlog_cols = ['bedrooms', 'condition', 'floors', 'sqft_basement', 'yr_built', 'waterfront_YES', 'renovated_renovated']
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
```

Out[40]:

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

15684 rows × 10 columns

In [41]:

```
# Fitting our log model
log_results = model_results(y, X_log)

print(log_results.summary())
```

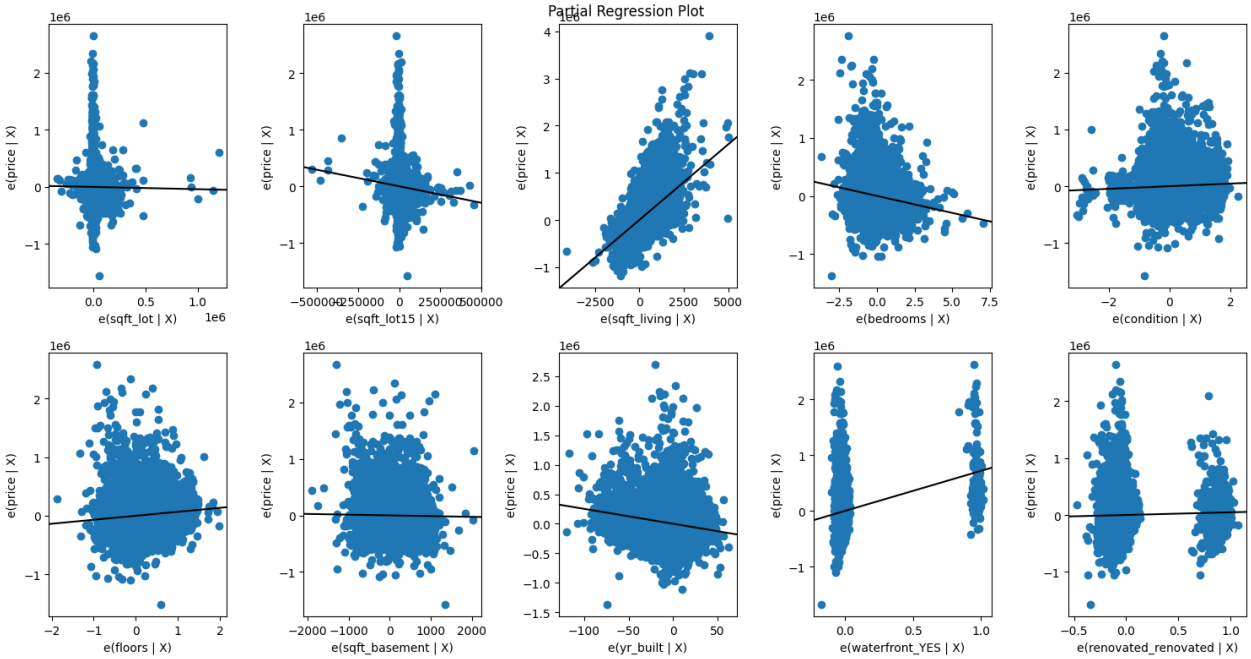
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.583
Model:                            OLS   Adj. R-squared:                  0.583
Method:                 Least Squares   F-statistic:                     2196.
Date:                Thu, 20 Apr 2023   Prob (F-statistic):               0.00
Time:                        08:25:45   Log-Likelihood:            -2.1591e+05
No. Observations:               15684   AIC:                         4.318e+05
Df Residuals:                   15673   BIC:                         4.319e+05
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
const                  4.865e+06    1.6e+05     30.416      0.000    4.55e+06    5.18e+06
sqft_lot                 -0.0395      0.063     -0.624      0.533      -0.164       0.085
sqft_lot15               -0.5849      0.095     -6.145      0.000      -0.771      -0.398
sqft_living             320.9332      3.184    100.808      0.000     314.693     327.173
bedrooms              -5.867e+04   2570.153    -22.827      0.000   -6.37e+04   -5.36e+04
condition              2.296e+04   3121.181      7.355      0.000    1.68e+04    2.91e+04
floors                 6.815e+04   4453.310     15.302      0.000    5.94e+04    7.69e+04
sqft_basement           -12.8030      5.333     -2.401      0.016     -23.257      -2.349
yr_built              -2520.9598     80.670    -31.250      0.000   -2679.082   -2362.838
waterfront_YES         7.202e+05   2.16e+04     33.346      0.000    6.78e+05    7.63e+05
renovated_renovated    4.695e+04   9766.040      4.807      0.000    2.78e+04    6.61e+04
==============================================================================
Omnibus:                     7284.383   Durbin-Watson:                   1.978
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           107806.947
Skew:                           1.850   Prob(JB):                         0.00
Kurtosis:                      15.299   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 [42]:

```
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
```



In [43]:

```
# 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[43]:

	bedrooms	sqft_living	sqft_lot	floors	condition	sqft_basement	yr_built	sqft_lot15	waterfront_YES	renovated_renovated	...	zipcode_
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	...	

15684 rows × 79 columns

In [44]:

```
# fitting the model for it
multiple_results2 = model_results(y, X_multiple2)

print(multiple_results2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.791
Model:                            OLS   Adj. R-squared:                  0.790
Method:                 Least Squares   F-statistic:                     746.1
Date:                Thu, 20 Apr 2023   Prob (F-statistic):               0.00
Time:                        08:25:49   Log-Likelihood:            -2.1052e+05
No. Observations:               15684   AIC:                         4.212e+05
Df Residuals:                   15604   BIC:                         4.218e+05
Df Model:                          79
Covariance Type:            nonrobust
==============================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const                -2.584e+05   1.38e+05     -1.872      0.061   -5.29e+05    1.22e+04
bedrooms             -3.614e+04   1862.855    -19.401      0.000   -3.98e+04   -3.25e+04
sqft_living            284.0456      2.459    115.516      0.000     279.226     288.865
sqft_lot                 0.2787      0.045      6.134      0.000       0.190       0.368
floors               -3.019e+04   3667.173     -8.231      0.000   -3.74e+04    -2.3e+04
condition             2.813e+04   2294.053     12.264      0.000    2.36e+04    3.26e+04
sqft_basement          -87.0066      4.100    -21.220      0.000     -95.044     -78.970
yr_built                41.4886     69.173      0.600      0.549     -94.099     177.076
sqft_lot15              -0.0386      0.070     -0.552      0.581      -0.176       0.099
waterfront_YES        8.337e+05   1.58e+04     52.899      0.000    8.03e+05    8.65e+05
renovated_renovated    4.81e+04   7008.294      6.863      0.000    3.44e+04    6.18e+04
zipcode_98002         1.322e+04    1.7e+04      0.779      0.436      -2e+04    4.65e+04
zipcode_98003         7159.1813   1.52e+04      0.472      0.637   -2.26e+04    3.69e+04
zipcode_98004         7.889e+05   1.51e+04     52.330      0.000    7.59e+05    8.18e+05
zipcode_98005         3.411e+05   1.78e+04     19.170      0.000    3.06e+05    3.76e+05
zipcode_98006         3.339e+05   1.36e+04     24.566      0.000    3.07e+05    3.61e+05
zipcode_98007         2.723e+05   1.93e+04     14.115      0.000    2.34e+05     3.1e+05
zipcode_98008         2.885e+05   1.52e+04     18.927      0.000    2.59e+05    3.18e+05
zipcode_98010         4.407e+04   2.23e+04      1.973      0.049     283.894    8.79e+04
zipcode_98011         1.273e+05   1.73e+04      7.362      0.000    9.34e+04    1.61e+05
zipcode_98014         7.738e+04   2.03e+04      3.809      0.000    3.76e+04    1.17e+05
zipcode_98019         6.244e+04   1.78e+04      3.499      0.000    2.75e+04    9.74e+04
zipcode_98022         7092.2983   1.64e+04      0.433      0.665    -2.5e+04    3.92e+04
zipcode_98023        -1.501e+04   1.34e+04     -1.120      0.263   -4.13e+04    1.12e+04
zipcode_98024         1.414e+05    2.4e+04      5.904      0.000    9.45e+04    1.88e+05
zipcode_98027         1.803e+05    1.4e+04     12.896      0.000    1.53e+05    2.08e+05
zipcode_98028         1.293e+05   1.53e+04      8.473      0.000    9.94e+04    1.59e+05
zipcode_98029          2.36e+05   1.47e+04     16.066      0.000    2.07e+05    2.65e+05
zipcode_98030       -1296.0782   1.58e+04     -0.082      0.935   -3.23e+04    2.97e+04
zipcode_98031         1.435e+04   1.54e+04      0.929      0.353   -1.59e+04    4.46e+04
zipcode_98032         1.331e+04   1.95e+04      0.682      0.495   -2.49e+04    5.15e+04
zipcode_98033          3.89e+05   1.38e+04     28.119      0.000    3.62e+05    4.16e+05
zipcode_98034         2.268e+05   1.32e+04     17.210      0.000    2.01e+05    2.53e+05
zipcode_98038         2.359e+04    1.3e+04      1.819      0.069   -1835.904     4.9e+04
zipcode_98039          1.29e+06   2.98e+04     43.338      0.000    1.23e+06    1.35e+06
zipcode_98040         5.748e+05   1.56e+04     36.936      0.000    5.44e+05    6.05e+05
zipcode_98042       -2284.7063   1.31e+04     -0.174      0.862    -2.8e+04    2.34e+04
zipcode_98045         1.038e+05   1.66e+04      6.274      0.000    7.14e+04    1.36e+05
zipcode_98052         2.537e+05   1.31e+04     19.403      0.000    2.28e+05    2.79e+05
zipcode_98053         1.841e+05   1.42e+04     12.989      0.000    1.56e+05    2.12e+05
zipcode_98055         5.204e+04   1.56e+04      3.339      0.001    2.15e+04    8.26e+04
zipcode_98056         9.721e+04   1.39e+04      6.976      0.000    6.99e+04    1.25e+05
zipcode_98058         2.878e+04   1.37e+04      2.094      0.036    1841.899    5.57e+04
zipcode_98059         8.409e+04   1.37e+04      6.116      0.000    5.71e+04    1.11e+05
zipcode_98065         7.693e+04   1.51e+04      5.085      0.000    4.73e+04    1.07e+05
zipcode_98070       -8064.3892   2.09e+04     -0.386      0.699    -4.9e+04    3.28e+04
zipcode_98072          1.65e+05   1.55e+04     10.678      0.000    1.35e+05    1.95e+05
zipcode_98074         2.111e+05    1.4e+04     15.077      0.000    1.84e+05    2.39e+05
zipcode_98075         2.022e+05   1.45e+04     13.975      0.000    1.74e+05    2.31e+05
zipcode_98077          1.39e+05   1.69e+04      8.213      0.000    1.06e+05    1.72e+05
zipcode_98092       -2.706e+04   1.45e+04     -1.869      0.062   -5.54e+04    1314.426
zipcode_98102         5.379e+05   2.32e+04     23.136      0.000    4.92e+05    5.83e+05
zipcode_98103         3.682e+05   1.34e+04     27.556      0.000    3.42e+05    3.94e+05
zipcode_98105         5.247e+05   1.65e+04     31.864      0.000    4.92e+05    5.57e+05
zipcode_98106         1.613e+05   1.49e+04     10.798      0.000    1.32e+05    1.91e+05
zipcode_98107         3.989e+05   1.56e+04     25.651      0.000    3.68e+05    4.29e+05
zipcode_98108         1.391e+05   1.76e+04      7.889      0.000    1.05e+05    1.74e+05
zipcode_98109         5.348e+05   2.17e+04     24.651      0.000    4.92e+05    5.77e+05
zipcode_98112         7.035e+05   1.57e+04     44.726      0.000    6.73e+05    7.34e+05
zipcode_98115         3.689e+05   1.33e+04     27.668      0.000    3.43e+05    3.95e+05
zipcode_98116         3.515e+05   1.48e+04     23.790      0.000    3.23e+05    3.81e+05
zipcode_98117         3.517e+05   1.34e+04     26.278      0.000    3.25e+05    3.78e+05
zipcode_98118         1.986e+05   1.36e+04     14.638      0.000    1.72e+05    2.25e+05
zipcode_98119         5.713e+05   1.78e+04     32.090      0.000    5.36e+05    6.06e+05
zipcode_98122         3.859e+05   1.56e+04     24.686      0.000    3.55e+05    4.17e+05
zipcode_98125         2.193e+05   1.41e+04     15.558      0.000    1.92e+05    2.47e+05
zipcode_98126         2.342e+05   1.47e+04     15.892      0.000    2.05e+05    2.63e+05
zipcode_98133         1.721e+05   1.36e+04     12.628      0.000    1.45e+05    1.99e+05
zipcode_98136         3.026e+05   1.57e+04     19.296      0.000    2.72e+05    3.33e+05
zipcode_98144         3.249e+05   1.47e+04     22.054      0.000    2.96e+05    3.54e+05
zipcode_98146         1.164e+05   1.54e+04      7.572      0.000    8.63e+04    1.47e+05
zipcode_98148         6.293e+04   2.73e+04      2.305      0.021    9412.283    1.16e+05
```

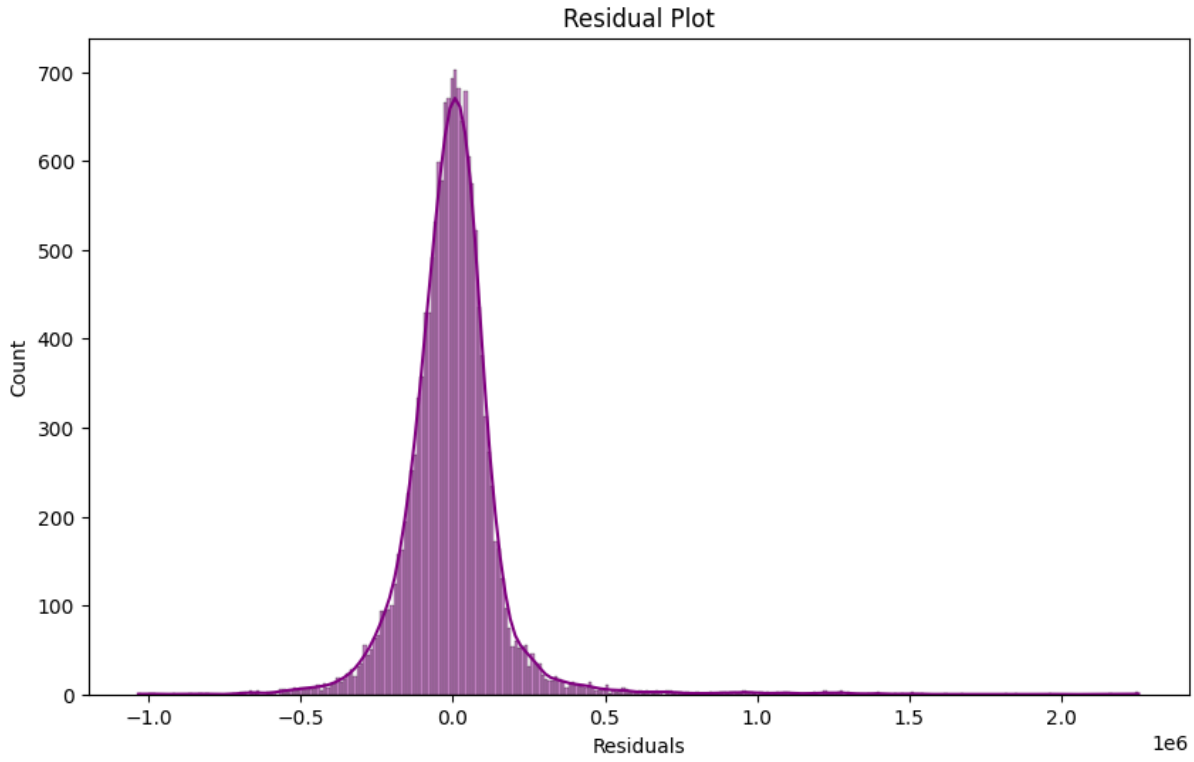
```
zipcode_98155        1.623e+05   1.39e+04     11.664      0.000    1.35e+05     1.9e+05
zipcode_98166        8.753e+04    1.6e+04      5.474      0.000    5.62e+04    1.19e+05
zipcode_98168        6.648e+04   1.58e+04      4.208      0.000    3.55e+04    9.75e+04
zipcode_98177        2.961e+05   1.58e+04     18.693      0.000    2.65e+05    3.27e+05
zipcode_98178        6.196e+04   1.59e+04      3.894      0.000    3.08e+04    9.32e+04
zipcode_98188        4.161e+04   1.96e+04      2.126      0.034    3241.627       8e+04
zipcode_98198        3.085e+04   1.55e+04      1.995      0.046     532.677    6.12e+04
zipcode_98199        4.596e+05   1.53e+04     30.113      0.000     4.3e+05     4.9e+05
==============================================================================
Omnibus:                    10331.164   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           421210.575
Skew:                           2.613   Prob(JB):                         0.00
Kurtosis:                      27.844   Cond. No.                     5.48e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.48e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [45]:

```
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[45]:

```
Text(0.5, 1.0, 'Residual Plot')
```



In [46]:

```
# MAE
y_pred = multiple_results.predict(sm.add_constant(X_multiple))
print(mean_absolute_error(y, y_pred))

y_pred2 = multiple_results2.predict(sm.add_constant(X_multiple2))
print(mean_absolute_error(y, y_pred2))
```

```
155674.0560874512
100952.09687719482
```

- We are using MAE over RMSE because of the propensity for RMSE to inflate values due to the square feature
- Our final model has reduced our mean absolute error by about 55,000 dollars

In [47]:

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

Out[47]:

	coefficient	pvalue
const	-258372.256479	6.125832e-02
bedrooms	-36141.977981	7.099173e-83
sqft_living	284.045576	0.000000e+00
sqft_lot	0.278746	8.782333e-10
floors	-30185.742175	1.996512e-16
...	...	...
zipcode_98177	296081.235338	3.908621e-77
zipcode_98178	61960.484676	9.917193e-05
zipcode_98188	41605.375322	3.354055e-02
zipcode_98198	30854.329085	4.610930e-02
zipcode_98199	459607.334099	1.093305e-193

80 rows × 2 columns

In [48]:

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

Out[48]:

	coefficient	pvalue
const	-258372.256479	0.061258
yr_built	41.488611	0.548662
sqft_lot15	-0.038572	0.581269
zipcode_98002	13221.143686	0.435806
zipcode_98003	7159.181303	0.637168
zipcode_98022	7092.298293	0.665078
zipcode_98023	-15008.959781	0.262521
zipcode_98030	-1296.078179	0.934751
zipcode_98031	14350.776858	0.352797
zipcode_98032	13307.033418	0.494944
zipcode_98038	23587.116765	0.068997
zipcode_98042	-2284.706316	0.861656
zipcode_98070	-8064.389201	0.699225
zipcode_98092	-27061.171013	0.061597

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

Findings/ Results

- The multiple linear regression model built 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.
- 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 most impactful features for inferring and predicting house sale prices were found to be the square footage of the living area, bedrooms, floors, and whether the property had a waterfront view or was renovated.
- The waterfront view proved 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.
- Our model is off by about 100,000 dollars for each prediction of the price

Challenges in the project

- More research is needed to be done to ascertain the integrity of the data. Specifically the bedrooms data which showed that as bedrooms increase the price decreases. Houses with more than 8 bedrooms being about 10 total, having even 1 with 33 bedrooms
- We had many predictor variables that were highly correlated that necessitated dropping of those columns
- We had to include more predictors in our model to realise a higher explanation for the variability in price that meets our metric of success expectation of 75%.

Recommendations

- 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 see an increase in value by about 48,000 dollars.
- 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

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.
- However, there are 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.
- In summary, 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

- More research is required to have a more integrated and informative dataset finding more factors that influence the price.
- More time would be required to fine_tune our findings and model results.
- Using datasets from other counties to be able to better advice our customers from comparing the dataset results.
- The agency may have a questionnaire 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 evaluate 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.