PREDICTING HOUSE PRICES IN KING COUNTY: A MULTIPLE LINEAR REGRESSION APPROACH.

PROJECT BY:

- Kelvin Rotich
- Grace Mutuku
- Joy Ogutu
- Peter Otieno
- Shuaib Mahamud

INTRODUCTION

Welcome to this presentation on predicting house prices in King County, Washington. In this project, we leverage a Multiple Linear Regression model to gain insights into the dynamic real estate market of this thriving region. Our analysis is based on a comprehensive dataset from King County, which encompasses a multitude of factors influencing property prices.

The primary purpose of our project is to develop a predictive model that accurately estimates house prices in King County. We aim to provide valuable insights to various stakeholders, including homeowners, real estate agents, investors, and developers, to help them make informed decisions regarding property pricing

In the following sections, we will delve into our project's methodology, key findings, and the model's performance in predicting house prices. Thank you for joining us in this exploration of King County's real estate landscape.

PROJECT OVERVIEW

King County’s Real Estate Background

The real estate market in King County, Washington, is known for its dynamism and diversity. This thriving area boasts a robust economy driven by tech giants like Amazon, Microsoft, and Boeing, which continually attract a large workforce. However, King County's real estate market, while vibrant, is not without its complexities and challenges. One defining characteristic of King County's real estate market is the ever-present demand for homes. Thus, there is a growing interest in sustainable housing options. This robust demand has contributed to an ongoing conundrum: affordability. As more individuals seek housing, the supply-demand balance skews, leaving many residents struggling to find reasonably priced homes. As King County experiences urban expansion, the competition extends beyond traditional housing. The development of sustainable, modern, and environmentally friendly housing solutions is another front where stakeholders compete.

The King County real estate market is highly competitive, with various stakeholders such as developers, online platforms, and established real estate companies vying for their share of the market. To navigate the complexities of this environment effectively, it's crucial for local stakeholders to understand the dynamics of the real estate market.

To comprehend this intricate landscape, stakeholders must understand the factors influencing property prices, most of which align with the columns provided in our dataset:

- Property-specific attributes, including location, size, condition, and amenities.
- Market dynamics, which encompass supply and demand, interest rates, and broader economic conditions.
- External factors, such as neighborhood characteristics and government policies.

In response to these challenges, our project seeks to develop a predictive multilinear regression model, utilizing the dataset at hand. Real estate agents can provide more accurate pricing guidance and develop effective marketing strategies. Homeowners can make informed decisions when pricing their properties, and investors and developers can identify promising opportunities to maximize their returns.

As we delve into the details of our predictive model, we will explore the methodology, key findings, and the model's performance. We will uncover how specific features and factors, such as the number of bedrooms, quality of views, and location, influence house prices in King County.

Problem Statement

In the dynamic real estate market of King County, Washington, where economic conditions, housing demand, and external influences drive property prices, the importance of accurate pricing cannot be overstated. The ever-present demand for homes, fueled by the presence of major tech companies and a stream of workers, creates a competitive environment. However, the challenge of accurately pricing properties in this competitive landscape can sometimes lead to overpricing. Sellers, eager to maximize their returns in a high-demand market, may set initial prices that are higher than what the market can sustain. This overpricing can, in turn, deter potential buyers and extend the time properties spend on the market. Therefore, the need for accurate pricing models that consider all relevant factors, including property conditions becomes paramount in ensuring that homes in King County are competitively priced, facilitating smoother transactions for both buyers and sellers.

Objectives

1. Objective: To explore and analyze the impact of numeric attributes on house prices in King County. Identify which numeric features have the most significant influence on pricing.Provide insights into how each unit increase or decrease in these attributes affects the final sale price. Generate recommendations for homeowners, buyers, and investors to optimize property attributes and investments based on numeric data.

2. Objective: To investigate the influence of categorical attributes on house prices. Determine which categorical features, such as being on a waterfront or having a high-grade rating, command premium prices. Provide recommendations on how to leverage these categorical attributes to maximize property values. Assist stakeholders in making informed decisions based on categorical data.
3. Objective: To create a precise property valuation model that calculates the cost of homes depending on a range of characteristics. This model will utilize a property's characteristics, including but not limited to bedrooms, square footage, and more. By carefully selecting and incorporating these features, we intend to build a model that accurately reflects the diverse attributes influencing house prices.

Importing libraries.

In [1]:
```
# Importing the necessary libraries
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np

import statsmodels.api as sm
import scipy.stats as stats

import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('seaborn')
%matplotlib inline
```

Data Understanding

In [2]:
```python
# Function to load and examine the data
def load_and_examine_data(file_path):
    try:
        # Load the data from the specified file path
        data = pd.read_csv(file_path)

        # Display the shape, columns and the first few rows of the dataset
        print("----------------------------Details about the data---------------------------- ")
        print("----------------------------Shape of the dataset---------------------------- ")
        display(data.shape)
        print("----------------------------Columns of the dataset---------------------------- ")
        display(data.columns)
        print("----------------------------Head of the dataset---------------------------- ")
        display(data.head())

        # Display information about the dataset
        print("\n----------------------------Data information ----------------------------")
        display(data.info())

        return data

    except FileNotFoundError:
        print(f"File '{file_path}' not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

# Replace with your data file path
file_path = "data\kc_house_data.csv"
data = load_and_examine_data(file_path)
```

```
----------------------------Details about the data----------------------------
----------------------------Shape of the dataset----------------------------

(21597, 21)

----------------------------Columns of the dataset----------------------------

Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')

----------------------------Head of the dataset----------------------------
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN	NONE	...	7 Average	1180	0.0	1955
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	NO	NONE	...	7 Average	2170	400.0	1951
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	NO	NONE	...	6 Low Average	770	0.0	1933
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	NO	NONE	...	7 Average	1050	910.0	1965
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	NO	NONE	...	8 Good	1680	0.0	1987

5 rows × 21 columns

```
----------------------------Data information ----------------------------
<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 dataset contains the sale prices and details of houses sold from 2nd May 2014 to 27th May 2015. The dataset has 21 columns with 21,597 entries.

Additional information of the columns:

- `id` - Unique identifier for a house
- `date` - Date house was sold
- `price` - Sale price (prediction target)
- `bedrooms` - Number of bedrooms
- `bathrooms` - Number of bathrooms
- `sqft_living` - Square footage of living space in the home
- `sqft_lot` - Square footage of the lot
- `floors` - Number of floors (levels) in house
- `waterfront` - Whether the house is on a waterfront
- `view` - Quality of view from house
- `condition` - How good the overall condition of the house is. Related to maintenance of house.
- `grade` - Overall grade of the house. Related to the construction and design of the house.
- `sqft_above` - Square footage of house apart from basement
- `sqft_basement` - Square footage of the basement
- `yr_built` - Year when house was built
- `yr_renovated` - Year when house was renovated
- `zipcode` - ZIP Code used by the United States Postal Service
- `lat` - Latitude coordinate
- `long` - Longitude coordinate
- `sqft_living15` - The square footage of interior housing living space for the nearest 15 neighbors
- `sqft_lot15` - The square footage of the land lots of the nearest 15 neighbors

Data Preparation

Null Values

All columns apart from `waterfront` , `view` , `yr_renovated` and `sqft_basement` have no null values

In [3]:
```
#Checking for null value counts and their percentages
columns_with_missing_values = ['waterfront', 'view', 'yr_renovated']
missing_values_table = pd.DataFrame([
    {
        'Column': column,
        'Missing Count': data[column].isnull().sum(),
        'Missing Percentage': (data[column].isnull().sum() / len(data[column])) * 100
    }
    for column in columns_with_missing_values])
print(missing_values_table)
```

```
         Column  Missing Count  Missing Percentage
0    waterfront           2376           11.001528
1          view             63            0.291707
2  yr_renovated           3842           17.789508
```

In [4]:
```
# Replace the null values in yr_renovated with the most most common value '0'
data['yr_renovated'].fillna(0, inplace = True)
```

In [5]:
```
# Replace the null values in waterfront and view with 'unknown'
data.fillna('unknown', inplace = True)
```

In [6]:
```
# Checking for null value counts and their percentages
columns_with_missing_values = ['waterfront', 'view', 'yr_renovated']
missing_values_table = pd.DataFrame([
    {
        'Column': column,
        'Missing Count': data[column].isnull().sum(),
        'Missing Percentage': (data[column].isnull().sum() / len(data[column])) * 100
    }
    for column in columns_with_missing_values])
print(missing_values_table)
```

```
         Column  Missing Count  Missing Percentage
0    waterfront              0                 0.0
1          view              0                 0.0
2  yr_renovated              0                 0.0
```

Duplicates

177 houses flagged as duplicates according to the `id` , are not duplicates but houses sold more than one times.

In [7]:
```
# Checking for duplicate entries
data.duplicated(subset='id').sum()
```

Out[7]:
```
177
```

In [8]:
```
# Identifying the duplicate entries
duplicate_rows = data[data.duplicated(subset=['id'], keep=False)].sort_values(by='id')
duplicate_rows['id'].value_counts()
```

Out[8]:
```
795000620     3
8651402750    2
5536100020    2
7387500235    2
9238500040    2
             ..
2143700830    2
3271300955    2
1901600090    2
3323059027    2
2023049218    2
Name: id, Length: 176, dtype: int64
```

Datatype Conversion

The `date` datatypes wa converted from object to datetime.

In [9]:
```
# Convert the datatype of date from object to datetime
data['date'] = pd.to_datetime(data['date'])
```

Outliers

The presence of outliers, representing distinctive property attributes, is retained because they are genuine events that provide valuable information for predicting house prices.

In [10]:
```
# Creating a function that checks for outliers in all the columns
def check_outliers(data, columns):
    for column in columns:
        # Calculate IQR (Interquartile Range)
        iqr = data[column].quantile(0.75) - data[column].quantile(0.25)

        # Define lower and upper thresholds
        lower_threshold = data[column].quantile(0.25) - 1.5 * iqr
        upper_threshold = data[column].quantile(0.75) + 1.5 * iqr

        # Find outliers
        outliers = data[(data[column] < lower_threshold) | (data[column] > upper_threshold)]

        # Print the count of outliers
        print(f"{column}\nNumber of outliers: {len(outliers)}\n")

columns_to_check = data.select_dtypes(include = ['number'])
check_outliers(data, columns_to_check)
```

```
id
Number of outliers: 0

price
Number of outliers: 1158

bedrooms
Number of outliers: 530

bathrooms
Number of outliers: 561

sqft_living
Number of outliers: 571

sqft_lot
Number of outliers: 2419

floors
Number of outliers: 0

sqft_above
Number of outliers: 610

yr_built
Number of outliers: 0

yr_renovated
Number of outliers: 744

zipcode
Number of outliers: 0

lat
Number of outliers: 2

long
Number of outliers: 255

sqft_living15
Number of outliers: 543

sqft_lot15
Number of outliers: 2188
```

In [11]:
```
# Checking for placeholders in each column
for column in data.columns:
    unique_values = data[column].unique()
    placeholders = [value for value in unique_values if str(value).strip().lower() in ['placeholder', 'na', 'n/a',
    placeholder_count = len(placeholders)

    print(f"Column: '{column}'")
    print(f"Placeholders found: {placeholders}")
    print(f"Count of placeholders: {placeholder_count}\n")
```

```
Column: 'price'
Placeholders found: []
Count of placeholders: 0

Column: 'bedrooms'
Placeholders found: []
Count of placeholders: 0

Column: 'bathrooms'
Placeholders found: []
Count of placeholders: 0

Column: 'sqft_living'
Placeholders found: []
Count of placeholders: 0

Column: 'sqft_lot'
Placeholders found: []
Count of placeholders: 0
```

In [12]:
```
# Replace the ? placeholder with '0.0'
data['sqft_basement'].replace('?', '0.0', inplace = True)
```

In [13]:
```
# Changing the datatype of sqft_basement from object to float after replacing the placeholder
data['sqft_basement'] = data['sqft_basement'].astype('float')
```

In [14]:
```
# Checking info
data.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  datetime64[ns]
 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     21597 non-null  object
 9   view           21597 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  float64
 14  yr_built       21597 non-null  int64
 15  yr_renovated   21597 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: datetime64[ns](1), float64(7), int64(9), object(4)
memory usage: 3.5+ MB
```

Feature Engineering

Two new features, `season` and `house_age_lv` are developed from `date` and `yr_built` respectively.

In [15]:
```python
# Create a function to map months to seasons
def get_season(date):
    if date.month in [3,4,5]:
        return 'Spring'
    elif date.month in [6,7,8]:
        return 'Summer'
    elif date.month in [9,10,11]:
        return 'Autumn'
    else:
        return 'Winter'

# Apply the function to the 'date' column to create a 'season' column
data['season'] = data['date'].apply(get_season)
data[['date', 'season']]
```

Out[15]:

	date	season
0	2014-10-13	Autumn
1	2014-12-09	Winter
2	2015-02-25	Winter
3	2014-12-09	Winter
4	2015-02-18	Winter
...	...	...
21592	2014-05-21	Spring
21593	2015-02-23	Winter
21594	2014-06-23	Summer
21595	2015-01-16	Winter
21596	2014-10-15	Autumn

21597 rows × 2 columns

In [16]:
```python
# Feature engineering to create a new column called house_age_lv

data['house_age'] = 2015 - data.yr_built
def houseage(house_age):
    if house_age >= 50:
        return 'Old'
    elif house_age >= 25:
        return 'Mid-age'
    else:
        return 'New'

# Apply the function to the 'date' column to c
data['house_age_lv'] = data['house_age'].apply(houseage)
data.drop(['house_age'], axis = 1, inplace = True)
```

EXPLORATORY DATA ANALYSIS

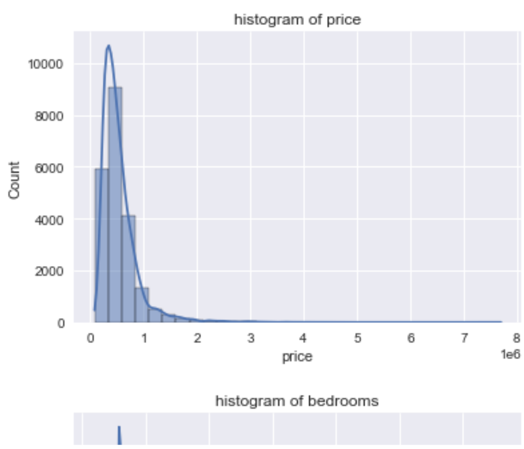
In [17]:
```python
# Getting the statistic summary of columns
data.describe()
```

Out[17]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above	sqft_basement	yr_built	yr_r
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	21597.000000	21597.000000	21597.000000	21597.000000	2159
mean	4.580474e+09	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	1.494096	1788.596842	285.716581	1970.999676	6
std	2.876736e+09	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	0.539683	827.759761	439.819830	29.375234	36
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.000000	370.000000	0.000000	1900.000000	
25%	2.123049e+09	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1.000000	1190.000000	0.000000	1951.000000	
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	1560.000000	0.000000	1975.000000	
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	2.000000	2210.000000	550.000000	1997.000000	
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	9410.000000	4820.000000	2015.000000	201

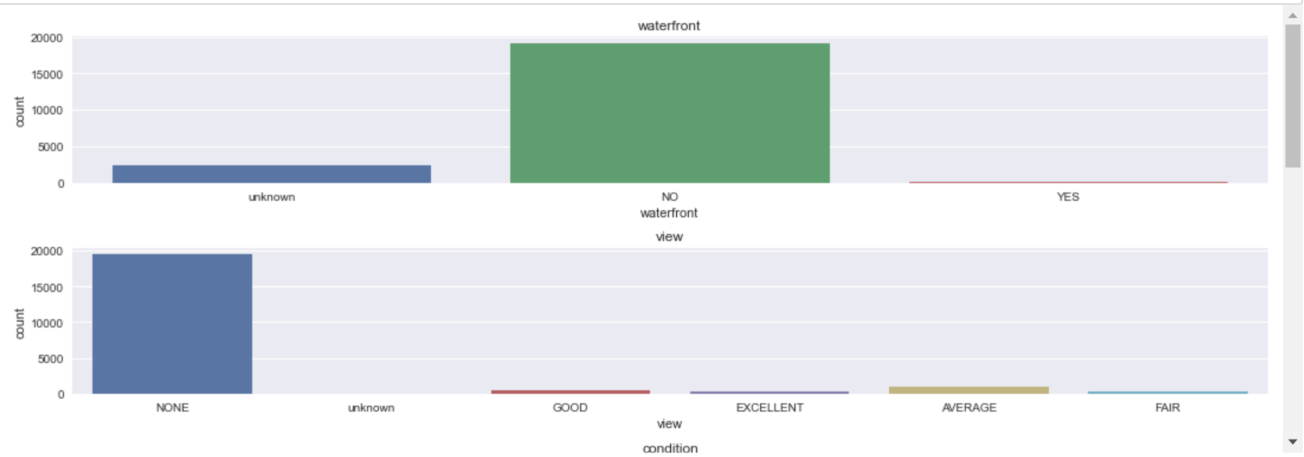
Histogram Summary

In [18]:
```
# Creating histograms for selected columns

# Identify numerical columns
numeric_columns = data[['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'sqft_above', 'sqft_l

# Iterate over numerical columns and create histograms
for column in numeric_columns.columns:
    plt.figure(figsize=(6, 4))
    sns.histplot(data=numeric_columns, x=column,bins = 30 ,common_norm = False, kde = True)
    plt.title(f"histogram of {column}")
    plt.show()
```



Count Plots

In [19]:
```
# creating count plots for selected columns

# Identify categorical columns
categorical_columns = data[['waterfront', 'view', 'condition', 'grade', 'season', 'house_age_lv']].columns

# Create a figure with a grid of subplots
fig, axes = plt.subplots(len(categorical_columns), 1, figsize=(15, 15))

# Iterate over categorical columns and create countplots
for i, column in enumerate(categorical_columns):
    sns.countplot(data=data, x=column, ax=axes[i])
    axes[i].set_title(column)

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



In [20]:
```
categories = ['waterfront', 'view', 'condition', 'grade', 'season', 'house_age_lv']

# Create subplots
fig, axes = plt.subplots(3, 2, figsize=(20, 15))
fig.suptitle('Mean Price Distribution by Category', fontsize=16)

# Loop through the categories and create bar plots
for i, category in enumerate(categories):
    row, col = i // 2, i % 2
    grouped_data = data.groupby(category)['price'].mean()
    sns.barplot(x=grouped_data.index, y=grouped_data.values, ax=axes[row, col])
    axes[row, col].set_title(f'Mean Price by {category}')
    axes[row, col].set_xlabel(category)
    axes[row, col].set_ylabel('Price')

# Remove any extra empty subplot if the number of categories is odd
if len(categories) % 2 == 1:
    fig.delaxes(axes[2, 1])

# Adjust layout
plt.tight_layout()
plt.subplots_adjust(top=0.9)

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

1. The houses with waterfronts nearby were sold at high prices compared to those without waterfronts nearby
2. The houses with an excellent view had higher prices compared with the other houses
3. The houses with very good condition sold at higher prices compared to the other houses
4. The houses given a mansion grade were sold at higher prices compared to the other houses
5. The houses sold during spring were sold at higher prices compared to the other seasons

Scatter Plots

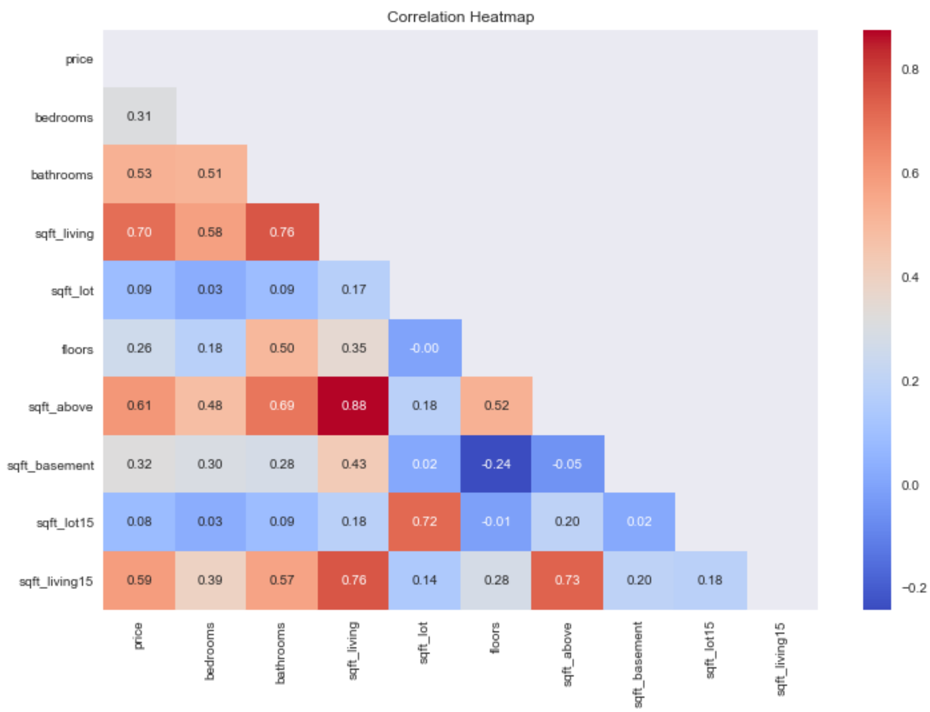
In [21]:
```
# setting the target column
target_column = "price"

# List of columns to create scatterplots for
columns_to_plot = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'sqft_above',
                   'sqft_basement']

# Create subplots
fig, axes = plt.subplots(3, 3, figsize=(20, 15))
fig.suptitle('Scatter Plots', fontsize=16)

# Create scatterplots for each selected column against the target using Seaborn
for i, column in enumerate(columns_to_plot):
    row, col = i // 3, i % 3
    sns.scatterplot(x=column, y=target_column, data = data, ax=axes[row, col])
    axes[row, col].set_xlabel(column)
    axes[row, col].set_ylabel('Price')

# Remove any extra empty subplot if the number of categories is odd
if len(column) % 3 == 1:
    fig.delaxes(axes[2, 2])
    fig.delaxes(axes[2, 1])

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

In [22]:
```
# Comparing correlation between price and selected features
numeric_columns = data[['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'sqft_above', 'sqft_l
correlation_matrix = numeric_columns.corr()['price']
correlation_matrix
```

Out[22]:
```
price            1.000000
bedrooms         0.308787
bathrooms        0.525906
sqft_living      0.701917
sqft_lot         0.089876
floors           0.256804
sqft_above       0.605368
sqft_basement    0.321108
sqft_lot15       0.082845
sqft_living15    0.585241
Name: price, dtype: float64
```

HeatMap

In [23]:
```
# Creating a heatmap
correlation_matrix = numeric_columns.corr()
# Create a mask to hide the upper triangle
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", mask=mask,)
plt.title('Correlation Heatmap')
plt.show()
```



MODELLING

1. Baseline Model

Based on the correlation matrix above, we observed that the `sqft_living` column has the highest correlation with `price` . This column will be our predictor variable for our baseline model

In [24]:
```
# Setting the target and predictor variables for our baseline model
y = data['price']
X_baseline = data[['sqft_living']]

# Creating the model
baseline_model = sm.OLS(y, sm.add_constant(X_baseline))
# Fitting the model
baseline_results = baseline_model.fit()
# Printing the summary of the model
print(baseline_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.493
Model:                            OLS   Adj. R-squared:                  0.493
Method:                 Least Squares   F-statistic:                 2.097e+04
Date:                Thu, 26 Oct 2023   Prob (F-statistic):               0.00
Time:                        03:14:46   Log-Likelihood:            -3.0006e+05
No. Observations:               21597   AIC:                         6.001e+05
Df Residuals:                   21595   BIC:                         6.001e+05
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const       -4.399e+04   4410.023     -9.975      0.000   -5.26e+04   -3.53e+04
sqft_living   280.8630      1.939    144.819      0.000     277.062     284.664
==============================================================================
Omnibus:                    14801.942   Durbin-Watson:                   1.982
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           542662.604
Skew:                           2.820   Prob(JB):                         0.00
Kurtosis:                      26.901   Cond. No.                     5.63e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.63e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [25]:
```
# Getting the coefficients of the model
baseline_results.params
```

Out[25]:
```
const         -43988.892194
sqft_living      280.863014
dtype: float64
```

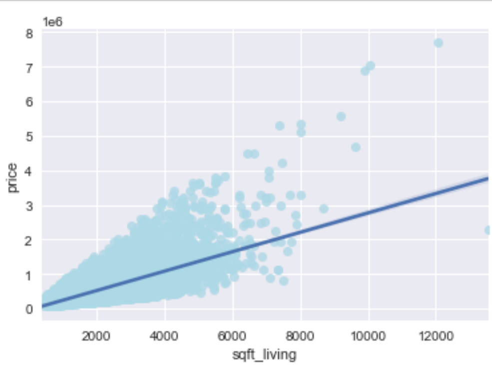
In [26]:
```
# Getting the p-value of the f-statistic
baseline_results.f_pvalue
```

Out[26]:
```
0.0
```

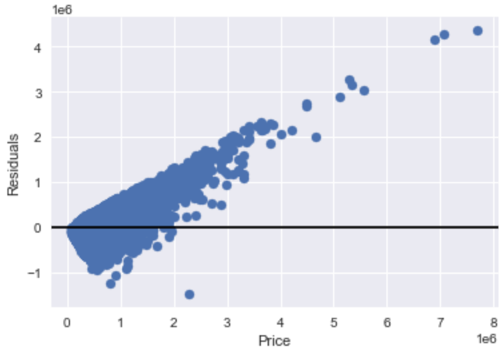
In [27]:
```
# Getting the R-squared of the model
baseline_results.rsquared
```

Out[27]:
```
0.49268789904035104
```

In [28]:
```
# Plotting the fit
sns.regplot(x = 'sqft_living', y = 'price', data = data, scatter_kws={'color': 'lightblue'})
plt.show()
```



In [29]:
```
# Plotting residuals
fig, ax = plt.subplots()
ax.scatter(data['price'], baseline_results.resid)
ax.axhline(y = 0, color = 'black')
ax.set_xlabel('Price')
ax.set_ylabel('Residuals');
```



In [30]:
```
# Create a function that calculates mae and rmse
def metrics(results):
    mae = (results.resid.abs().sum()) / len(results.resid)
    rmse = (((results.resid ** 2).sum()) / len(results.resid)) ** 0.5
    print(f'MAE: {mae} \nRMSE: {rmse}')

metrics(baseline_results)
```

```
MAE: 173824.8874961748
RMSE: 261655.00451904477
```

Baseline Model Results

The results of the analysis indicate the following key findings:

`sqft_living` explains approximately 49.3% of the variation in house prices, suggesting that the size of the living area significantly influences house prices.

The overall model is statistically significant, as indicated by an F-statistic p-value of 0.00, demonstrating the model's validity and suggesting that at least one predictor is associated with house prices.

Both the model and the coefficients are statistically significant, with p-values below the set alpha of 0.05, confirming the model's reliability in explaining variations in house prices.

For a house with no living area (0 square feet), the estimated price is approximately - USD 43,988.89. Additionally, for each additional square foot of living area, the house's price increases by about USD 280.86.

It is important to acknowledge that this baseline model has limitations, as suggested by residual plots showing a notable deviation of actual values from the regression line. This indicates that the model may not capture the full complexity of house price determinants, suggesting the need for further refinement and the inclusion of additional relevant factors for more accurate predictions.

2. Second Model

The next step we will take is to create a model with the columns we selected for the correlation matrix in the previous step.

In [31]:
```
# Creating the second model
X_1 = numeric_columns.copy()
X_1 = X_1.drop(columns = 'price', axis = 1)
model_1 = sm.OLS(y, sm.add_constant(X_1))
# Fitting the model
results_1 = model_1.fit()
print(results_1.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                  price   R-squared:                       0.519
Model:                            OLS   Adj. R-squared:                  0.518
Method:                 Least Squares   F-statistic:                     2584.
Date:                Thu, 26 Oct 2023   Prob (F-statistic):               0.00
Time:                        03:14:59   Log-Likelihood:            -2.9950e+05
No. Observations:               21597   AIC:                         5.990e+05
Df Residuals:                   21587   BIC:                         5.991e+05
Df Model:                           9                                         
Covariance Type:            nonrobust                                         
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
const          1.242e+04   8612.601      1.442      0.149   -4460.892    2.93e+04
bedrooms      -5.847e+04   2348.775    -24.893      0.000   -6.31e+04   -5.39e+04
bathrooms       988.7075   3817.954      0.259      0.796   -6494.765    8472.180
sqft_living     269.9686     22.777     11.852      0.000     225.323     314.614
sqft_lot          0.0514      0.060      0.851      0.395      -0.067       0.170
floors         1.751e+04   4313.382      4.060      0.000    9055.648     2.6e+04
sqft_above       -6.8371     22.800     -0.300      0.764     -51.527      37.853
sqft_basement    45.4632     22.676      2.005      0.045       1.016      89.910
sqft_lot15       -0.8602      0.092     -9.316      0.000      -1.041      -0.679
sqft_living15    72.8181      4.005     18.181      0.000      64.968      80.668
==============================================================================
Omnibus:                    15120.072   Durbin-Watson:                   1.984
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           619129.899
Skew:                           2.874   Prob(JB):                         0.00
Kurtosis:                      28.593   Cond. No.                     2.61e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.61e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

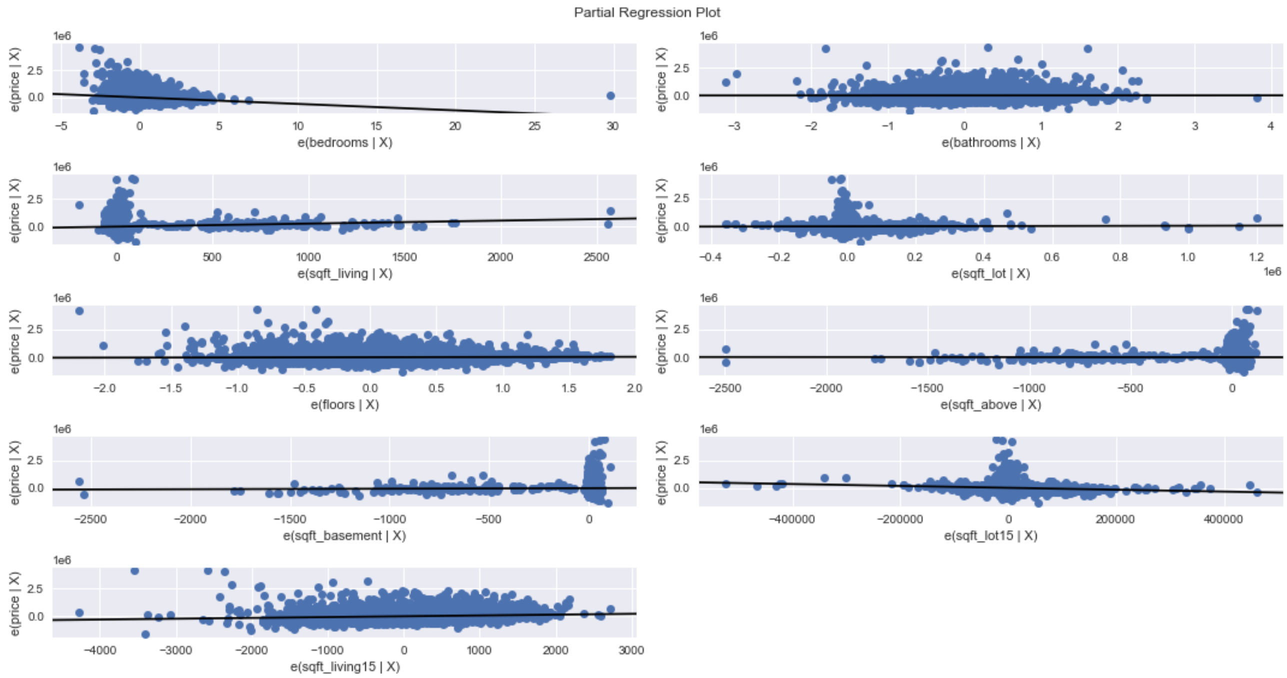
In [32]:
```
# Creating a function that returns the p-values of the F-statistic, coefficients and adjusted R-squared of the mode
def model_info(results):
    # Finding the p-value of the F-statistic
    p_value = results.f_pvalue
    # Finding the coefficients
    coefficients = results.params
    # Finding the adjusted R-squared
    adj_r_squared = (round(results.rsquared_adj, 4)) * 100
    # Returns the values
    print(f'p-value: {p_value},\n---------------coefficients--------------\n{coefficients}, \n------------- Adjuste
# Using the function for the model
model_info(results_1)
```

```
p-value: 0.0,
---------------coefficients--------------
const            12420.441945
bedrooms        -58468.043349
bathrooms          988.707508
sqft_living        269.968592
sqft_lot             0.051448
floors           17510.194445
sqft_above          -6.837082
sqft_basement       45.463170
sqft_lot15          -0.860178
sqft_living15       72.818082
dtype: float64,
------------- Adjusted R-Squared:------------
 51.839999999999996
```

In [33]:
```
# Finding metrics of the second model
metrics(results_1)
```

```
MAE: 167781.30532381844
RMSE: 254879.74279573272
```

In [34]:
```
# Creating a function that plots residuals
def plot_residuals(results, X):
    # Plotting the residuals
    fig = plt.figure(figsize = (15,8))
    sm.graphics.plot_partregress_grid(
        results,
        exog_idx = list(X.columns),
        fig = fig)
    plt.tight_layout()
    plt.show();

# Plotting for the model
plot_residuals(results_1, X_1)
```



Second Model Results

The model is statistically significant (F-statistic p-value < 0.05) and can explain 51.84% of the variance in sales.

For each additional bedroom, there's a decrease of about USD 58,468 in sales, while each additional bathroom increases sales by about USD 988.70.

An additional square foot of living space adds approximately USD 269 to the sale price, while each square foot of lot size increases it by around $0.051.

Adding more floors is associated with an increase of about USD17,510 in sale price.

For each additional square foot of the house apart from the basement, there is an associated decrease in sale price by about USD 6.84. For each additional square foot of the basement, there is an associated increase in sale price by about USD 45.46.

For each additional square footage of interior housing living space for the nearest 15 neighbors, there is an associated increase in sale price by about USD 72.81.

Based on the p-values of the coefficients, model refinement is needed. Consider standardization and the inclusion of categorical columns for better performance.

3. Third Model

In [35]:
```
# To standardize the numerical columns
X_standardized = X_1.copy()

for col in X_standardized:
    X_standardized[col] = (X_standardized[col] - X_standardized[col].mean()) \
                        / X_standardized[col].std()

# Creating the dummy variables for the categorical columns we will add and dropping the others
X_cat = pd.get_dummies(data, columns=["waterfront", 'grade', 'house_age_lv'], drop_first=True, dtype=int)
X_cat = X_cat.drop(['id', 'date', 'price', 'sqft_living', 'sqft_lot', 'bedrooms', 'bathrooms', 'floors', 'sqft_above

#Combining the two dataframes
X_2 = pd.concat([X_standardized, X_cat], axis = 1)

# Creating the third model
model_2 = sm.OLS(y, sm.add_constant(X_2))
results_2 = model_2.fit()
print(results_2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.667
Model:                            OLS   Adj. R-squared:                  0.667
Method:                 Least Squares   F-statistic:                     1963.
Date:                Thu, 26 Oct 2023   Prob (F-statistic):               0.00
Time:                        03:15:07   Log-Likelihood:            -2.9552e+05
No. Observations:               21597   AIC:                         5.911e+05
Df Residuals:                   21574   BIC:                         5.913e+05
Df Model:                          22
Covariance Type:            nonrobust
=========================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
const                  8.176e+05   8045.270    101.630      0.000    8.02e+05    8.33e+05
bedrooms              -2.563e+04   1892.443    -13.545      0.000   -2.93e+04   -2.19e+04
bathrooms                3.8e+04   2649.589     14.341      0.000    3.28e+04    4.32e+04
sqft_living            1.269e+05    1.75e+04     7.268      0.000    9.27e+04    1.61e+05
sqft_lot               1616.5738   2084.954      0.775      0.438   -2470.090    5703.238
floors                 2.928e+04   2163.744     13.534      0.000     2.5e+04    3.35e+04
sqft_above            -2.004e+04    1.57e+04    -1.274      0.203   -5.09e+04    1.08e+04
sqft_basement          1.425e+04   8300.771      1.717      0.086   -2021.708    3.05e+04
sqft_lot15             -1.48e+04   2102.566     -7.038      0.000   -1.89e+04   -1.07e+04
sqft_living15          2.719e+04   2428.606     11.195      0.000    2.24e+04    3.19e+04
waterfront_YES         7.194e+05    1.79e+04    40.229      0.000    6.84e+05    7.54e+05
grade_11 Excellent     2.735e+05    1.26e+04    21.694      0.000    2.49e+05    2.98e+05
grade_12 Luxury        7.644e+05     2.4e+04    31.788      0.000    7.17e+05    8.12e+05
grade_13 Mansion       1.993e+06    6.03e+04    33.058      0.000    1.87e+06    2.11e+06
grade_3 Poor          -4.693e+05    2.12e+05    -2.209      0.027   -8.86e+05   -5.29e+04
grade_4 Low            -5.31e+05    4.21e+04   -12.612      0.000   -6.13e+05   -4.48e+05
grade_5 Fair          -5.263e+05    1.69e+04   -31.190      0.000   -5.59e+05   -4.93e+05
grade_6 Low Average   -4.802e+05    1.07e+04   -44.825      0.000   -5.01e+05   -4.59e+05
grade_7 Average        -4.12e+05   8875.155    -46.426      0.000   -4.29e+05   -3.95e+05
grade_8 Good          -3.264e+05   8010.986    -40.749      0.000   -3.42e+05   -3.11e+05
grade_9 Better        -1.859e+05   7798.272    -23.842      0.000   -2.01e+05   -1.71e+05
house_age_lv_New      -6.527e+04   4483.825    -14.558      0.000   -7.41e+04   -5.65e+04
house_age_lv_Old       1.619e+05   3866.784     41.871      0.000    1.54e+05    1.69e+05
==============================================================================
Omnibus:                    12096.083   Durbin-Watson:                   1.985
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           394180.056
Skew:                           2.116   Prob(JB):                         0.00
Kurtosis:                      23.497   Cond. No.                         299.
==============================================================================

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

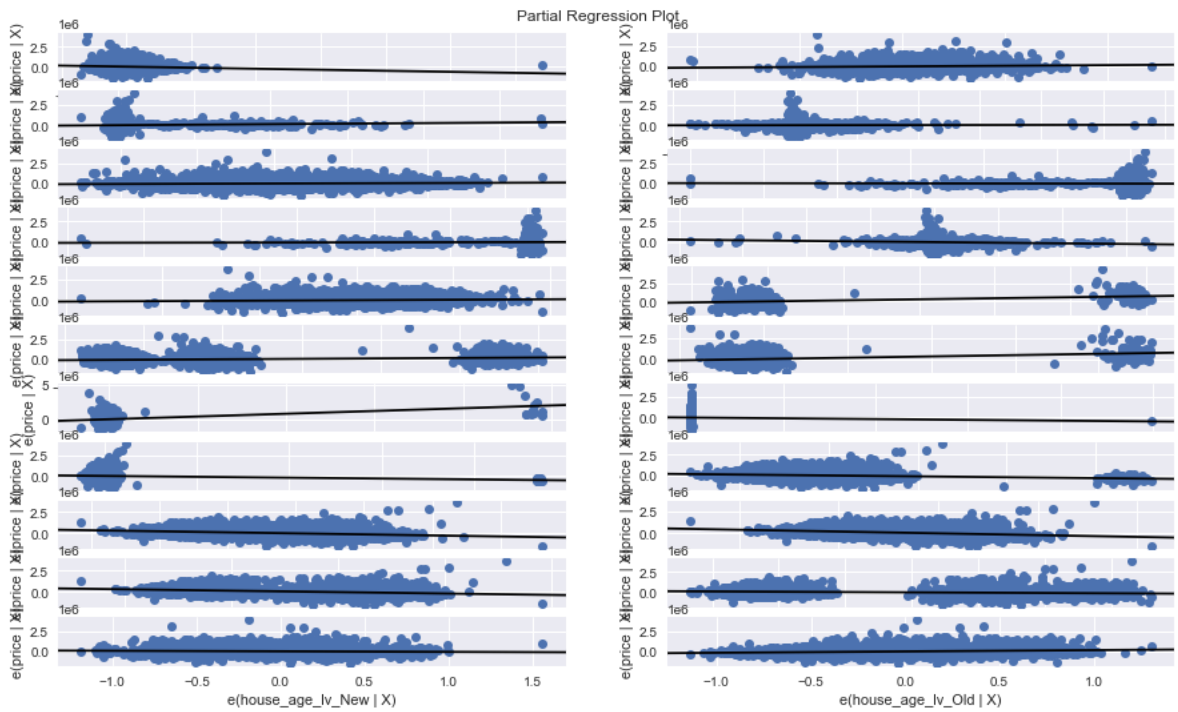
In [36]:
```
# Call the model_info function that outputs the model pvalue, coefficients and Adjusted R-squared

model_info(results_2)
```

```
p-value: 0.0,
---------------coefficients-------------
const                  8.176434e+05
bedrooms              -2.563408e+04
bathrooms              3.799906e+04
sqft_living            1.268989e+05
sqft_lot               1.616574e+03
floors                 2.928341e+04
sqft_above            -2.004390e+04
sqft_basement          1.424842e+04
sqft_lot15            -1.479854e+04
sqft_living15          2.718895e+04
waterfront_YES         7.193868e+05
grade_11 Excellent     2.734823e+05
grade_12 Luxury        7.644263e+05
grade_13 Mansion       1.992768e+06
grade_3 Poor          -4.692654e+05
grade_4 Low           -5.309623e+05
grade_5 Fair          -5.263303e+05
grade_6 Low Average   -4.802120e+05
grade_7 Average       -4.120355e+05
grade_8 Good          -3.264435e+05
grade_9 Better        -1.859231e+05
house_age_lv_New      -6.527357e+04
house_age_lv_Old       1.619054e+05
dtype: float64,
------------- Adjusted R-Squared:------------
 66.66
```

In [37]:
```
# Getting the metrics for the model
metrics(results_2)
```

```
MAE: 139993.9354728652
RMSE: 212020.519748943
```

In [38]:
```
# Calling th plot_residuals function to plot the partial regression plots

plot_residuals(results_2, X_2)
```



Third Model results

High Statistical Significance: The third model is statistically significant with an F-statistic p-value of 0.0.

Explained Variance: The model can explain approximately 66.7% of the variation in house prices.

Key Coefficients: Important predictor variables include `bedrooms` , `bathrooms` , `sqft_living` , `floors` , `waterfront` , `grade` categories, and `house_age` levels.

Model Metrics: The model's performance metrics are MAE ≈ USD 139,994 and RMSE ≈ USD 212,021.

Addressing Multicollinearity: The model deals with potential multicollinearity by refining predictor variable selection.

Potential for Improvement: Despite its significance, the model may benefit from further refinement to better capture the complexity of house price determinants.

In [39]:

```
# This code will help us get the numeric variables that are correlated with each other
#  It saves absolute value of correlation matrix as a data frame
# sort values. 0 is the column automatically generated by the stacking
df=X_standardized.corr().abs().stack().reset_index().sort_values(0, ascending=False)

# zip the variable name columns (Which were only named level_0 and level_1 by default) in a new column named "pairs'
df['pairs'] = list(zip(df.level_0, df.level_1))

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

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

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

# drop duplicates.
df.drop_duplicates(inplace=True)
# We look for values which have a correlation greater than 0.75 and less than 1
df[(df.cc>.75) & (df.cc <1)]
```

Out[39]:

pairs	cc
(sqft_above, sqft_living)	0.876448
(sqft_living15, sqft_living)	0.756402
(bathrooms, sqft_living)	0.755758

4. Fourth Model

Based on the above solution, we can drop `sqft_above` and `sqft_living15` columns and try to model again.

In [40]:
```
X_3 = X_2.copy()
# Drop the correlated columns sqft_above and sqft_living15
X_3.drop(['sqft_above', 'sqft_living15'], axis=1, inplace=True)

# Create the fourth model
model_3 = sm.OLS(y, sm.add_constant(X_3))

# Fit the model
results_3 = model_3.fit()
print(results_3.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                  price   R-squared:                       0.665
Model:                            OLS   Adj. R-squared:                  0.665
Method:                 Least Squares   F-statistic:                     2141.
Date:                Thu, 26 Oct 2023   Prob (F-statistic):               0.00
Time:                        03:15:23   Log-Likelihood:            -2.9558e+05
No. Observations:               21597   AIC:                         5.912e+05
Df Residuals:                   21576   BIC:                         5.914e+05
Df Model:                          20                                         
Covariance Type:            nonrobust                                         
=======================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------
const                8.372e+05   7868.900    106.388      0.000    8.22e+05    8.53e+05
bedrooms            -2.578e+04   1897.780    -13.582      0.000   -2.95e+04   -2.21e+04
bathrooms            3.716e+04   2653.174     14.004      0.000     3.2e+04    4.24e+04
sqft_living           1.22e+05   3345.536     36.458      0.000    1.15e+05    1.29e+05
sqft_lot              393.1212   2087.905      0.188      0.851   -3699.327    4485.570
floors                2.55e+04   2136.923     11.933      0.000    2.13e+04    2.97e+04
sqft_basement        2.106e+04   1927.354     10.929      0.000    1.73e+04    2.48e+04
sqft_lot15          -1.324e+04   2103.978     -6.295      0.000   -1.74e+04   -9120.644
waterfront_YES        7.22e+05   1.79e+04     40.263      0.000    6.87e+05    7.57e+05
grade_11 Excellent   2.787e+05   1.26e+04     22.062      0.000    2.54e+05    3.03e+05
grade_12 Luxury      7.685e+05   2.41e+04     31.883      0.000    7.21e+05    8.16e+05
grade_13 Mansion     1.968e+06   6.04e+04     32.587      0.000    1.85e+06    2.09e+06
grade_3 Poor        -4.905e+05   2.13e+05     -2.303      0.021   -9.08e+05   -7.29e+04
grade_4 Low         -5.488e+05   4.22e+04    -13.009      0.000   -6.32e+05   -4.66e+05
grade_5 Fair        -5.531e+05   1.67e+04    -33.026      0.000   -5.86e+05    -5.2e+05
grade_6 Low Average -5.097e+05   1.04e+04    -48.984      0.000    -5.3e+05   -4.89e+05
grade_7 Average     -4.368e+05   8608.545    -50.745      0.000   -4.54e+05    -4.2e+05
grade_8 Good        -3.424e+05   7897.284    -43.354      0.000   -3.58e+05   -3.27e+05
grade_9 Better      -1.918e+05   7801.129    -24.591      0.000   -2.07e+05   -1.77e+05
house_age_lv_New     -6.52e+04   4495.442    -14.503      0.000    -7.4e+04   -5.64e+04
house_age_lv_Old     1.593e+05   3868.993     41.182      0.000    1.52e+05    1.67e+05
==============================================================================
Omnibus:                    11830.288   Durbin-Watson:                   1.985
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           360588.138
Skew:                           2.069   Prob(JB):                         0.00
Kurtosis:                      22.585   Cond. No.                         247.
==============================================================================

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

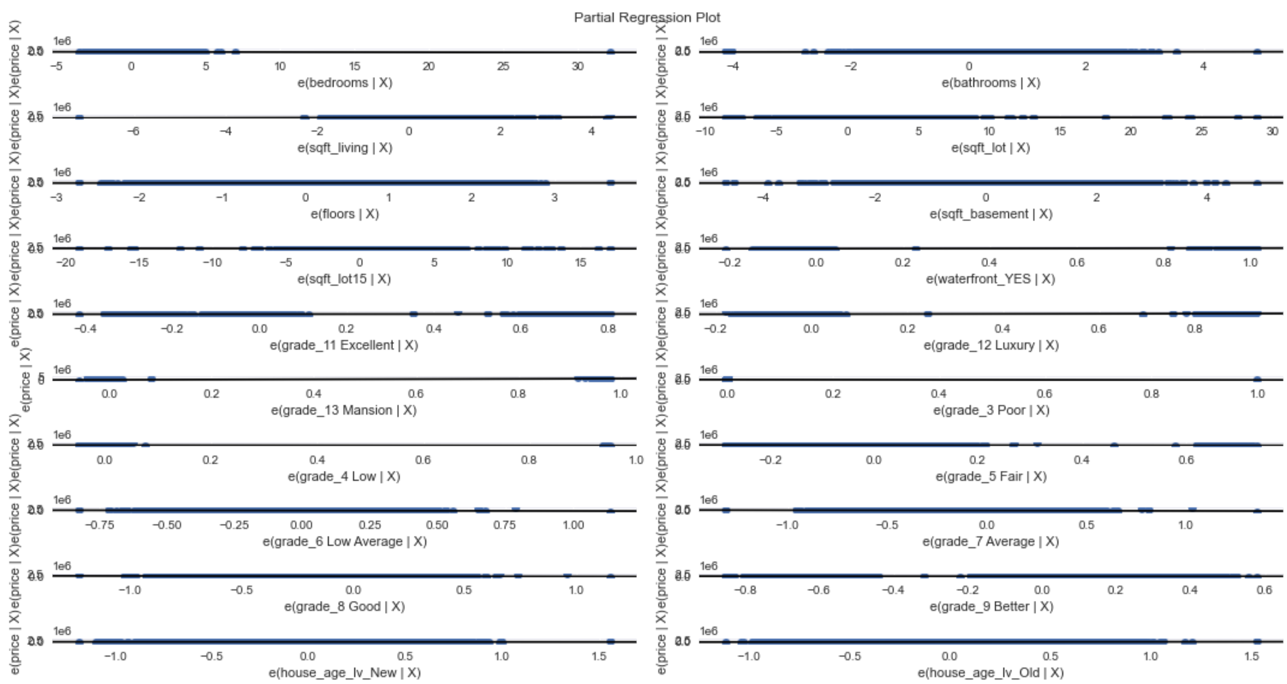
In [41]:
```
# Call the model_info function that outputs the model pvalue, coefficients and Adjusted R-squared
model_info(results_3)
```

```
p-value: 0.0,
---------------coefficients-------------
const                   8.371587e+05
bedrooms               -2.577649e+04
bathrooms               3.715529e+04
sqft_living             1.219729e+05
sqft_lot                3.931212e+02
floors                  2.550042e+04
sqft_basement           2.106454e+04
sqft_lot15             -1.324460e+04
waterfront_YES          7.220015e+05
grade_11 Excellent      2.786822e+05
grade_12 Luxury         7.685192e+05
grade_13 Mansion        1.968461e+06
grade_3 Poor           -4.904516e+05
grade_4 Low            -5.488261e+05
grade_5 Fair           -5.531277e+05
grade_6 Low Average    -5.097316e+05
grade_7 Average        -4.368436e+05
grade_8 Good           -3.423826e+05
grade_9 Better         -1.918366e+05
house_age_lv_New       -6.519620e+04
house_age_lv_Old        1.593345e+05
dtype: float64,
------------- Adjusted R-Squared:------------
 66.46
```

In [42]:
```
# Getting the metrics of the model
metrics(results_3)
```

```
MAE: 140536.28751924628
RMSE: 212637.44161924344
```

In [43]:
```
# Calling the plot_residuals function to plot the partial regression plots
plot_residuals(results_3, X_3)
```



Fourth Model Results

While the fourth model demonstrates strong explanatory power, it still has some limitations.

The removal of certain columns mitigated potential multicollinearity issues, resulting in an reduced R-squared value.

However, to further enhance the model's accuracy, additional refinements and explorations is be necessary.

5. Fifth Model

To improve the model, we will drop `sqft_lot` column because it is statistically insignificant with a p-value of 0.851.

In [44]:
```
# Drop sqft_lot column to improve the model
X_4 = X_3.copy()
X_4.drop(['sqft_lot'], axis=1, inplace=True)

# Create the fifth model
model_4 = sm.OLS(y, sm.add_constant(X_4))

# Fit the model
results_4 = model_4.fit()

print(results_4.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.665
Model:                            OLS   Adj. R-squared:                  0.665
Method:                 Least Squares   F-statistic:                     2254.
Date:                Thu, 26 Oct 2023   Prob (F-statistic):               0.00
Time:                        03:15:33   Log-Likelihood:            -2.9558e+05
No. Observations:               21597   AIC:                         5.912e+05
Df Residuals:                   21577   BIC:                         5.914e+05
Df Model:                          19
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const              8.372e+05   7868.723    106.391      0.000    8.22e+05    8.53e+05
bedrooms          -2.578e+04   1897.278    -13.590      0.000   -2.95e+04   -2.21e+04
bathrooms          3.716e+04   2653.088     14.005      0.000     3.2e+04    4.24e+04
sqft_living         1.22e+05   3341.024     36.517      0.000    1.15e+05    1.29e+05
floors             2.549e+04   2136.141     11.933      0.000    2.13e+04    2.97e+04
sqft_basement      2.105e+04   1926.199     10.929      0.000    1.73e+04    2.48e+04
sqft_lot15        -1.297e+04   1505.023     -8.616      0.000   -1.59e+04      -1e+04
waterfront_YES      7.22e+05   1.79e+04     40.264      0.000    6.87e+05    7.57e+05
grade_11 Excellent 2.787e+05   1.26e+04     22.063      0.000    2.54e+05    3.03e+05
grade_12 Luxury    7.685e+05   2.41e+04     31.885      0.000    7.21e+05    8.16e+05
grade_13 Mansion   1.968e+06   6.04e+04     32.588      0.000    1.85e+06    2.09e+06
grade_3 Poor      -4.904e+05   2.13e+05     -2.303      0.021   -9.08e+05    -7.3e+04
grade_4 Low       -5.488e+05   4.22e+04    -13.009      0.000   -6.31e+05   -4.66e+05
grade_5 Fair      -5.531e+05   1.67e+04    -33.030      0.000   -5.86e+05    -5.2e+05
grade_6 Low Average -5.097e+05   1.04e+04  -48.985      0.000    -5.3e+05   -4.89e+05
grade_7 Average   -4.368e+05   8608.347    -50.747      0.000   -4.54e+05    -4.2e+05
grade_8 Good      -3.424e+05   7897.106    -43.356      0.000   -3.58e+05   -3.27e+05
grade_9 Better    -1.918e+05   7800.648    -24.594      0.000   -2.07e+05   -1.77e+05
house_age_lv_New   -6.52e+04   4495.339    -14.503      0.000    -7.4e+04   -5.64e+04
house_age_lv_Old   1.593e+05   3868.834     41.185      0.000    1.52e+05    1.67e+05
==============================================================================
Omnibus:                    11829.255   Durbin-Watson:                   1.985
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           360481.572
Skew:                           2.069   Prob(JB):                         0.00
Kurtosis:                      22.582   Cond. No.                         245.
==============================================================================

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

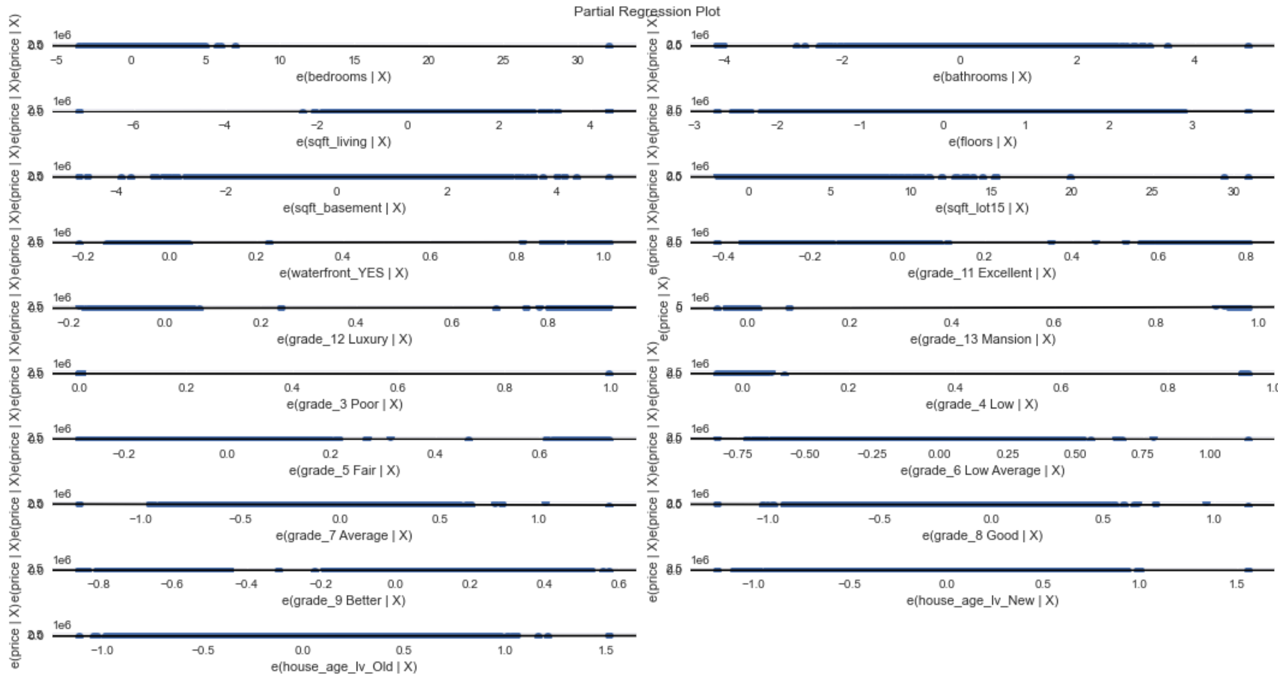
In [45]:
```
# Call the model_info function that outputs the model pvalue, coefficients and Adjusted R-squared
model_info(results_4)
```

```
p-value: 0.0,
---------------coefficients-------------
const                  8.371578e+05
bedrooms              -2.578436e+04
bathrooms              3.715751e+04
sqft_living            1.220053e+05
floors                 2.548987e+04
sqft_basement          2.105222e+04
sqft_lot15            -1.296778e+04
waterfront_YES         7.219689e+05
grade_11 Excellent     2.786895e+05
grade_12 Luxury        7.685384e+05
grade_13 Mansion       1.968309e+06
grade_3 Poor          -4.904488e+05
grade_4 Low           -5.488049e+05
grade_5 Fair          -5.530617e+05
grade_6 Low Average   -5.097242e+05
grade_7 Average       -4.368455e+05
grade_8 Good          -3.423833e+05
grade_9 Better        -1.918497e+05
house_age_lv_New      -6.519534e+04
house_age_lv_Old       1.593390e+05
dtype: float64,
------------- Adjusted R-Squared:------------
 66.47
```

In [46]:
```
# Getting the MAE and the RMSE
metrics(results_4)
```

```
MAE: 140537.59614097438
RMSE: 212637.61630999626
```

In [47]:
```
# Calling th plot_residuals function to plot the partial regression plots
plot_residuals(results_4, X_4)
```



Summary of the Fifth and Final Model

Performance: The model explains approximately 66.5% of the variance in house prices.

Key Predictors: It identifies essential predictors, such as the number of bedrooms, bathrooms, living space, floors, basement space, and waterfront view, which significantly impact house prices. Factors like grade and house age also play crucial roles.

Waterfront View: Having a waterfront view notably increases house prices by about USD 722,000.

Grade and House Age: Higher grades and newer houses are associated with higher prices, with Grade 13 (Mansion) having the most substantial impact.

Model Evaluation: The model's predictions are reasonably accurate, with a Mean Absolute Error (MAE) of approximately USD 140,537 and a Root Mean Square Error (RMSE) of around 212,637.

CONCLUSION

Waterfront: A waterfront view has the most significant positive impact on house prices, followed by high-quality house grades and spacious living areas.

House Grade: Higher house grades, such as "Mansion" and "Luxury," significantly increase prices, emphasizing the importance of property quality.

Square Footage: More living space, including basements, positively contributes to house prices, with larger homes commanding higher values.

Bathrooms and Floors: Additional bathrooms and floors enhance the price of a property.

Lot Size: Larger lot sizes, particularly Lot 15, have a negative effect on prices, indicating that smaller, more manageable lots are valued.

House Age: Older houses are generally more expensive than newer ones, possibly due to historical or architectural significance.

Bedrooms: An increase in the number of bedrooms is associated with lower house prices.

RECOMMENDATIONS

Prioritize Waterfront Properties: Promote homes with waterfront views to maximize pricing potential.

Enhance House Quality: Invest in property quality improvements, as higher-grade homes command better prices.

Highlight Square Footage: Emphasize living space square footage in property listings.

Consider Additional Bathrooms and Floors: Add more bathrooms or floors where feasible to increase property value.

Optimize Lot Sizes: Smaller, manageable lots, especially Lot 15, are preferred by buyers. Subdivide larger lots if possible.

Value Older Homes: Highlight the historical and architectural charm of older properties, positioning them as valuable assets.

Optimize Bedroom Layouts: Balance the bedroom count with effective layout design to maintain property appeal.

LIMITATIONS

Data Constraints: The analysis relies on available data, potentially missing critical variables.

External Variables: Economic shifts and government policies were excluded, which can affect the real estate market.

Simplified Model: The model assumes linear relationships, neglecting potential nonlinear interactions.

NEXT STEPS ¶

Incorporate Economic Indicators: Integrate economic factors into the model for better market trend predictions.

Advanced Predictive Models: Explore advanced machine learning techniques like gradient boosting and neural networks for more precise price forecasts.