Isaac-Ndirangu-Muturi-749 / **MORINGA-PHASE-2-PROJECT-GROUP-3.3** Public

Code Issues Pull requests 2 Actions Projects Wiki Security Insights

Beta Try the new code view

main

MORINGA-PHASE-2-PROJECT-GROUP-3.3 / **student.ipynb**

Isaac-Ndirangu-Muturi-749 add notebook.pdf **History**

2 contributors

9330 lines (9330 sloc) 1.99 MB

Final Project Submission

Please fill out:

- Student names: Isaac Muturi, Henry Mativo, Nelson Odhiambo, Mark Ngige, Marwa Osman, Fiona Kung'u
- Student pace: full time
- Scheduled project review date/time:
- Instructor name:
- Blog post URL:

BUSINESS UNDERSTANDING

A real estate agency from King County, Seattle hired us for a project to analyse how different factors affect prices of homes.The aim of this analysis is to build a multiple linear regression model that predicts the prices of houses in King County, Seattle.

As a data scientist analyzing the King County housing market, my business understanding is that the real estate industry is a crucial sector that plays a significant role in the economy. The success of a real estate transaction depends on several factors, including the location, the size of the property, the condition of the property, the amenities, and the current market conditions. The housing market is subject to various external factors such as interest rates, economic conditions, and government policies that can impact the demand and supply of properties.

Datasets

The dataset contains information about the houses in King County, Seattle. The dataset has 21 variables including the price, number of bedrooms, bathrooms, square footage of the living area, and other variables. The dataset contains 21,597 observations.

The scope of this analysis is limited to the data provided. We will use feature engineering techniques such as imputation, normalization, and one-hot encoding to preprocess the data. We will use multiple linear regression model. We will evaluate the performance of the model using metrics such as mean squared error, mean absolute error, and R-squared.

To overcome these challenges, we need to use a combination of quantitative and qualitative analysis techniques and incorporate domain knowledge and expertise. By understanding the King County housing market's complexities and using data-driven insights, we can help real estate agents and property owners make informed decisions about pricing, marketing, and selling their properties, ultimately leading to more successful real estate transactions and a more robust housing market.

BUSINESS PROBLEM

By developing a model that can accurately predict the sale price of houses, real estate agents can better advise their clients on pricing strategies, investors can identify potentially undervalued properties, and homeowners can better estimate the value of their own properties. This can ultimately lead to more efficient and profitable real estate transactions in King County.

You are charged with exploring what factors most significantly affect home prices. You must then translate those findings into actionable insights that the real estate agency can use to help decide what factors to consider when advising potential home buyers.

Business Objecives

1. Develop a pricing model: Create a predictive pricing model that incorporates the factors identified in the regression analysis, such as the number of bathrooms, living area, lot size, and condition and grade ratings. This model can help the agency more accurately price their properties, particularly those with unique features such as waterfront views.

2. Refine marketing strategies: Use the insights gained from the regression analysis to refine the agency's marketing strategies. For example, the agency can create targeted marketing campaigns that emphasize the features that have the greatest impact on price, such as the number of bathrooms, living area, and condition and grade ratings.

3. Analyze seasonal trends: Analyze the seasonal trends in home prices and develop strategies for pricing and marketing homes throughout the year. For example, the agency can adjust their pricing and marketing strategies to take advantage of the higher prices in the spring, when homes tend to sell for more.

4. Optimize home renovations: Use the insights gained from the regression analysis to identify which renovations have the greatest impact on a home's price. This information can be used to guide homeowners who are considering renovations and to help the agency market homes that have been recently renovated.

Importing the necessary libraries

In [1]:

```
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import missingno as msno
import numpy as np
import pandas as pd
pd.options.display.max_columns = 30
import statsmodels.api as sm
from scipy import stats
from sklearn.metrics import mean_absolute_error

# Group Libraries
import Functions as fun
```

DATA UNDERSTANDING

In this project, we are analyzing the King County housing market to build a multiple linear regression model that predicts the prices of houses in King County, Seattle.

In [2]:

```
# Your code here - remember to use markdown cells for comments as well!
import pandas as pd
house_df=pd.read_csv('kc_house_data.csv')
house_df.head()
```

Out[2]:

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

In [3]:

```
#Getting data information
house_df.info()
```

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

The dataset provided contains information on 21,597 houses in King County, Seattle.

To better understand the data, we identified the categorical and numerical variables in the dataset that is relevant to our business problema s shown below:

- **Numerical Columns (15)**

  `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

`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

`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

- **Categorical Columns (6)**

`id` - Unique ID for each home sold

`waterfront` - Whether the house has a view to a waterfront

`view` - An index from 0 to 4 of how good the view of the property was

`condition` - An index from 1 to 5 on the condition of the house

`grade` - An index from 1 to 13, where 1-3 falls short of building construction and design, 7 has an average level of construction and design, and 11-13 have a high quality level of construction and design

`zipcode` - What zipcode area the house is in

We chose these columns because they are key features that are often used to determine the value of a residential property in real estate.

The purpose of this exercise is to analyze and comprehend the information contained within the columns of the provided CSV file. Our aim is to carefully examine the data, identify patterns and correlations between the variables, and extract meaningful insights from it.

In [4]:

```
# Calculate the basic statistical summary
house_df.describe()
```

Out[4]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	215
mean	4.580474e+09	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	
std	2.876736e+09	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	
25%	2.123049e+09	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	

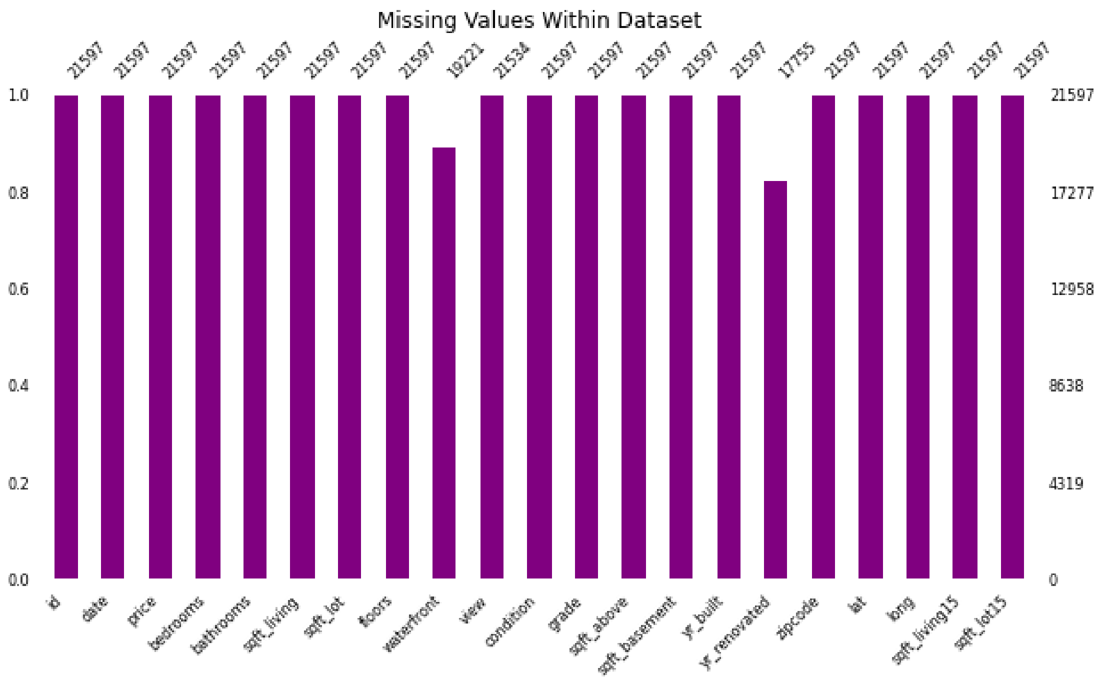
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06

DATA PREPARATION

Detecting missing/null values

We will start by visualizing our missing data.

In [5]:
```
# Visualise the missing values in the dataset
msno.bar(house_df, color='purple', figsize=(10, 5), fontsize=8)
plt.title('Missing Values Within Dataset');
```



In [6]:
```
house_df.isnull().sum()
```

Out[6]:
```
id                  0
date                0
price               0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront       2376
view               63
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_built            0
yr_renovated     3842
zipcode             0
lat                 0
```

```
lat                0
long               0
sqft_living15      0
sqft_lot15         0
dtype: int64
```

From the above bar graph, we can see that there exist missing data in waterfront, view and the year renovated.

We shall replace missing values of waterfront, view with mode and the year renovated with the median.

In [7]:

```
fun.fun_mode_fill_null(house_df, 'waterfront')
fun.fun_mode_fill_null(house_df, 'view')
fun.fun_median_fill_null(house_df, 'yr_renovated')
house_df.isnull().sum()
```

Out[7]:

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

checking for duplicates

In [8]:

```
# Check for duplicates in the 'id' column
house_df.id.duplicated().sum()
```

Out[8]: 177

There are 177 duplivates in the unique column data. We will use this to eliminate all duplicates with the same id.

In [9]:

```
fun.fun_duplicates_drop(house_df, 'id')
house_df.id.duplicated().sum()
```

Out[9]: 0

Detecting outliers

We shall then check for outliers

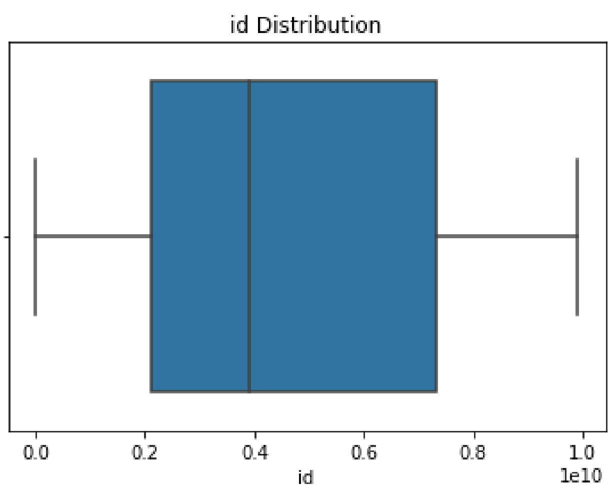
We shall then check for outliers.

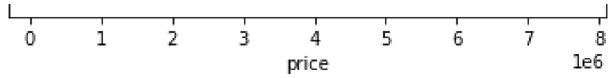
In [10]:
```
# function to check for outliers by plotting
def fun_outlier_plot_box(df, column_name):
    """
    Create a box plot for a specified column of a Pandas DataFrame using Seaborn.

    Parameters:
        df (pandas.DataFrame): The DataFrame containing the column to plot.
        column_name (str): The name of the column to plot.

    Returns:
        None
    """
    sns.boxplot(x=df[column_name])
```

In [11]:
```
house_df.columns
```

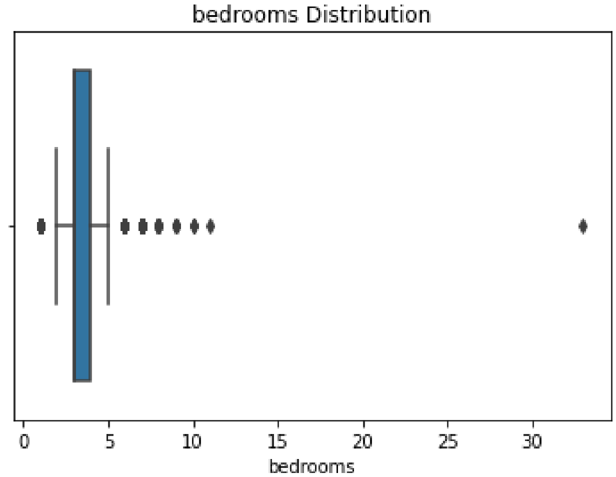
Out[11]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

In [12]:
```
# plot the outliers
cols = ['id','price','bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',

# iterate through the columns of house_df and call fun_outlier_plot_box()
for column in cols:
    fun_outlier_plot_box(house_df, column)
    plt.title(f"{column} Distribution")
    plt.show()
```

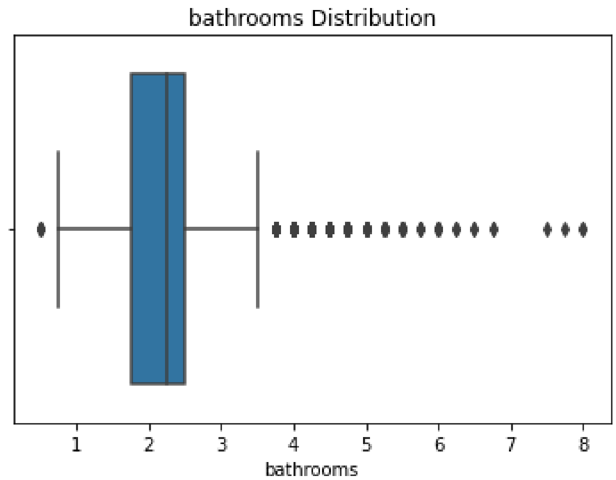




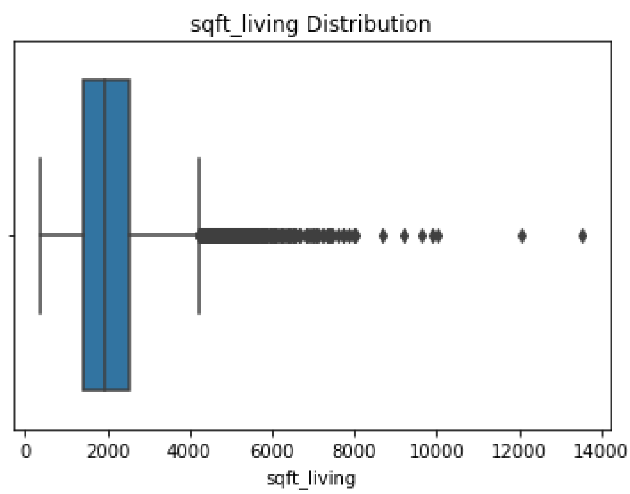
0 1 2 3 4 5 6 7 8
price
1e6



bedrooms Distribution
0 5 10 15 20 25 30
bedrooms

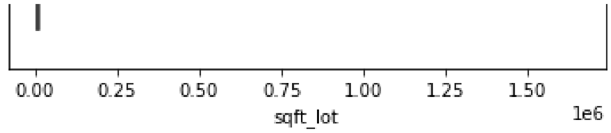


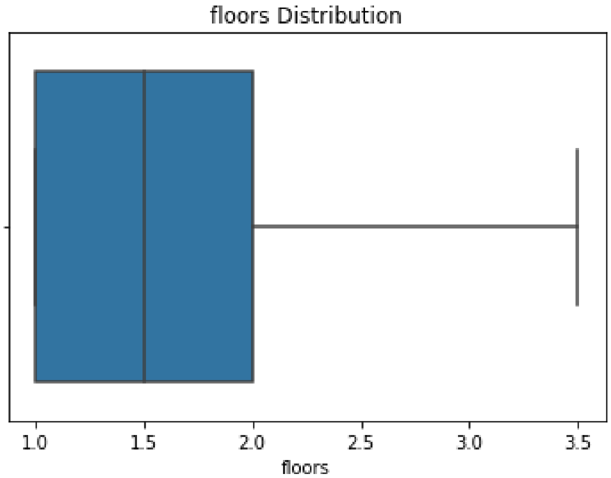
bathrooms Distribution
1 2 3 4 5 6 7 8
bathrooms



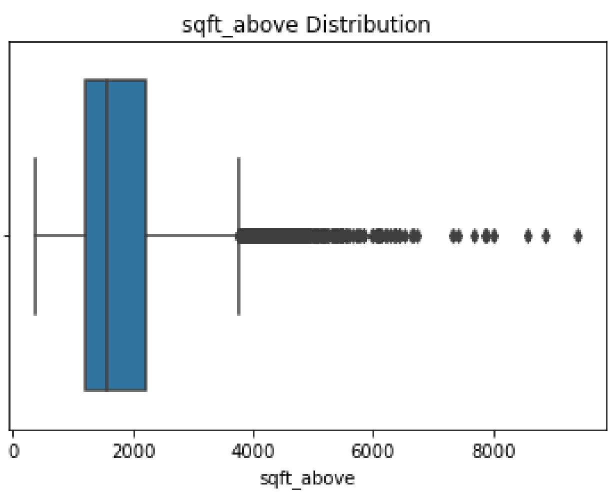
sqft_living Distribution
0 2000 4000 6000 8000 10000 12000 14000
sqft_living

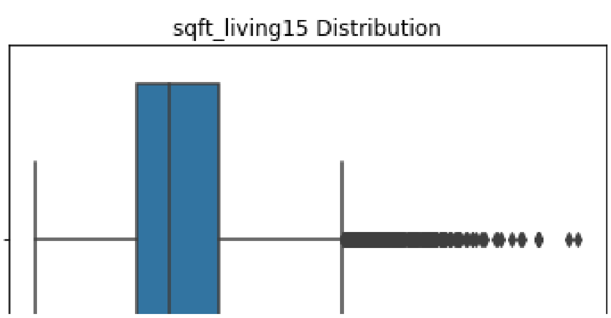
sqft_lot Distribution

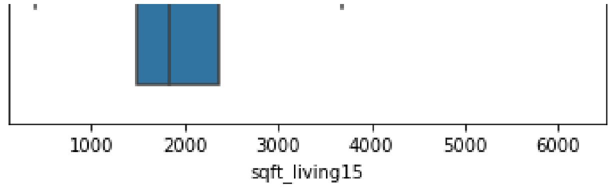

0.00 0.25 0.50 0.75 1.00 1.25 1.50
sqft_lot
1e6


floors Distribution
1.0 1.5 2.0 2.5 3.0 3.5
floors

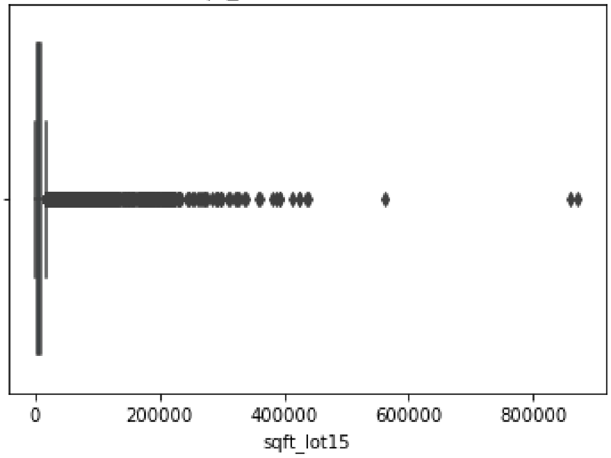

yr_built Distribution
1900 1920 1940 1960 1980 2000 2020
yr_built


sqft_above Distribution
0 2000 4000 6000 8000
sqft_above


sqft_living15 Distribution



sqft_lot15 Distribution



In [13]:

```
#function to remove outliers from the columns

def remove_outliers(data, cols, threshold=3):
    for col in cols:
        z_scores = np.abs(stats.zscore(data[col]))
        data = data[z_scores < threshold]
    return data
```

In [14]:

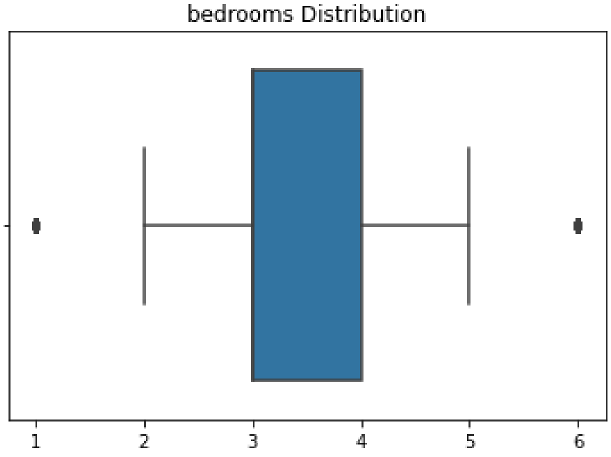
```
# Remove the outliers
columns_to_remove_outliers = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
house_df = remove_outliers(house_df , columns_to_remove_outliers)
```

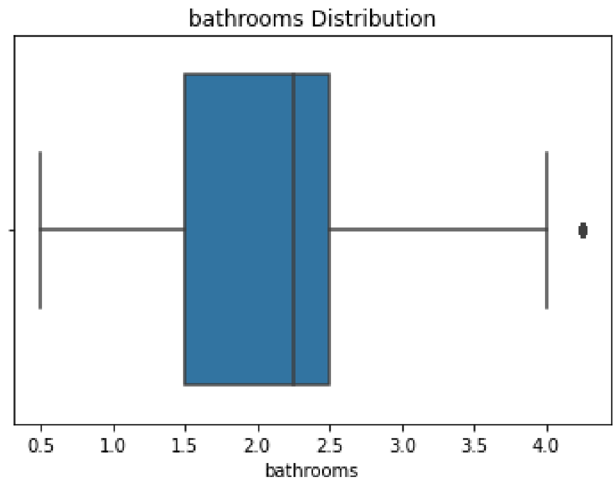
iterate through the columns of house_df and call fun_outlier_plot_box()

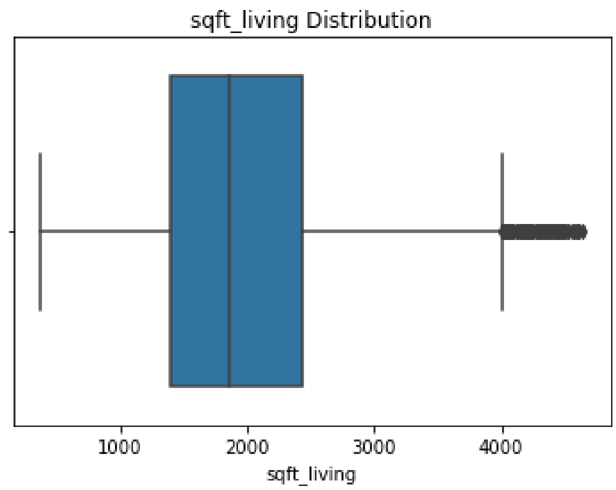
In [15]:

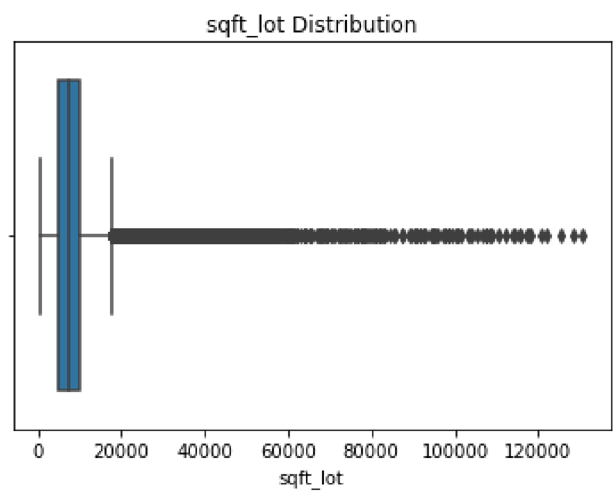
```
# iterate through the columns of house_df and call fun_outlier_plot_box()
for column in columns_to_remove_outliers:
    fun_outlier_plot_box(house_df, column)
    plt.title(f"{column} Distribution")
    plt.show()
```

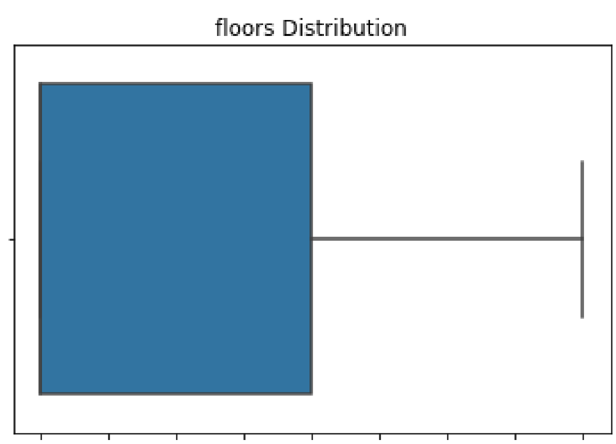
bedrooms Distribution



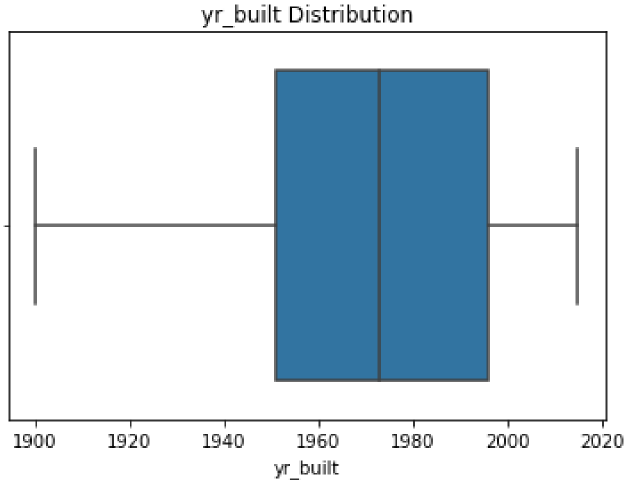

bathrooms Distribution
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
bathrooms

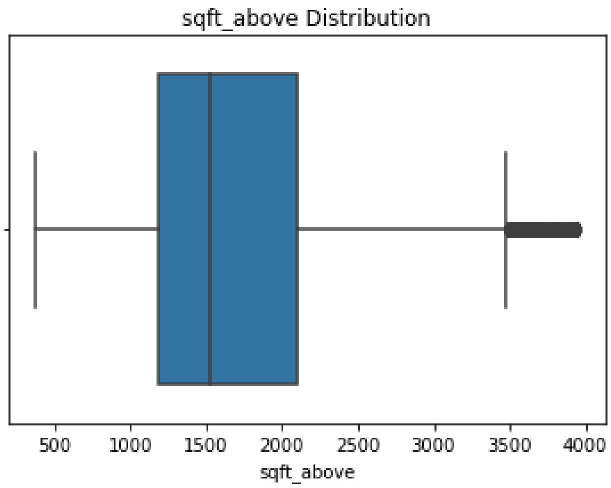

sqft_living Distribution
1000
2000
3000
4000
sqft_living

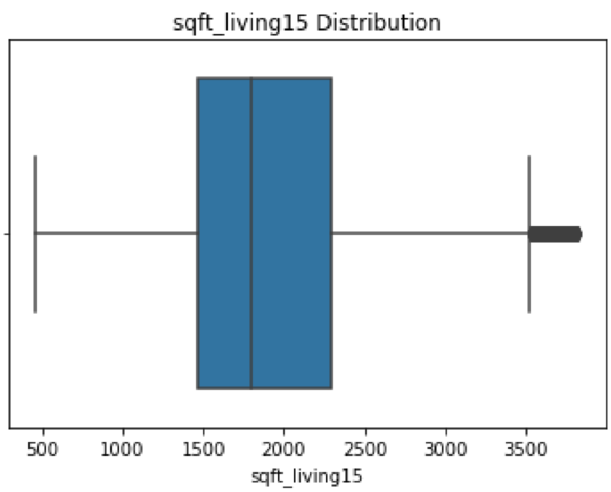

sqft_lot Distribution
0
20000
40000
60000
80000
100000
120000
sqft_lot

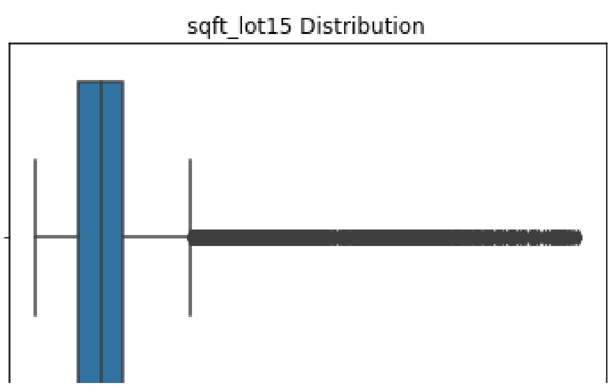

floors Distribution

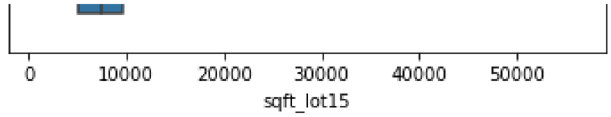

1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00
floors


yr_built Distribution
1900 1920 1940 1960 1980 2000 2020
yr_built


sqft_above Distribution
500 1000 1500 2000 2500 3000 3500 4000
sqft_above


sqft_living15 Distribution
500 1000 1500 2000 2500 3000 3500
sqft_living15


sqft_lot15 Distribution



Exploratory Data Analysis

This section will be the exploratory data analysis question where we will exploring and seeing the relationship that price has with other columns.

Univariate Analysis

In this section, we'll explore each column in the dataset to see the distributions of features and obtain some useful insights. The main two parts in this section are:

> - Categorical Columns
> - Numerical Columns

2.1.1 Categorical Columns

There are 5 Categorical Columns in the dataset that we shall be analysing:

- `id`
- `waterfront`
- `view`
- `condition`
- `grade`
- `zipcode`

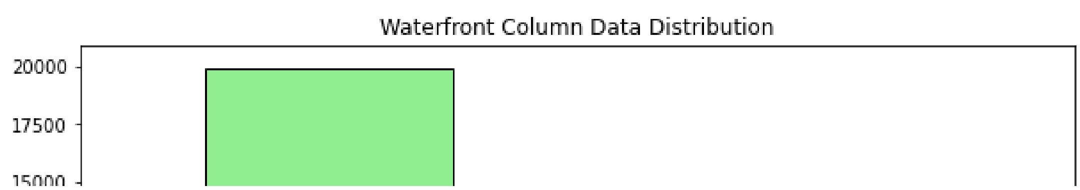
In [16]:

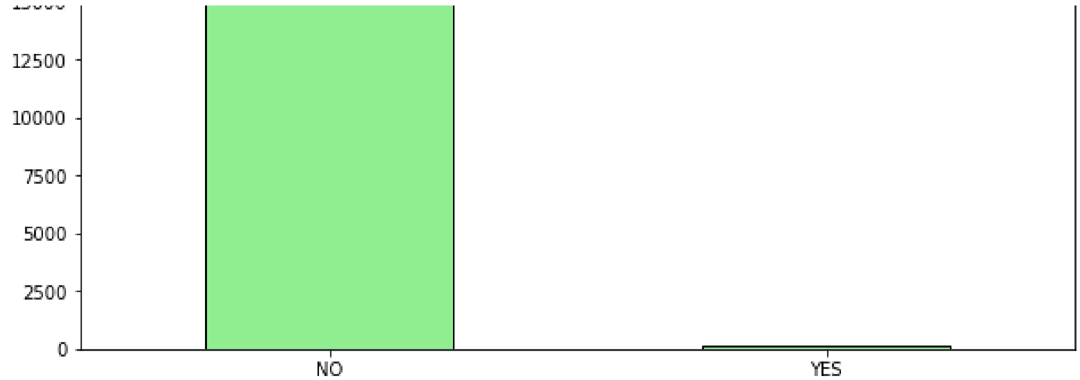
```
def fun_plot_value_counts(df, col, title):
    '''
    Returns the value counts of a column in a dataframe and
    plots the value counts of a column in a dataframe as a bar chart
    '''
    counts = df[col].value_counts(dropna=False)
    print(counts)
    counts.plot(kind='bar', figsize=(10, 5), color='lightgreen', edgecolor='black'
    plt.title(title)
    plt.xticks(rotation=0)
    plt.show()
    return counts
```

In [17]:

```
fun_plot_value_counts(house_df, 'waterfront', 'Waterfront Column Data Distributior
```

```
NO     19928
YES      102
Name: waterfront, dtype: int64
```

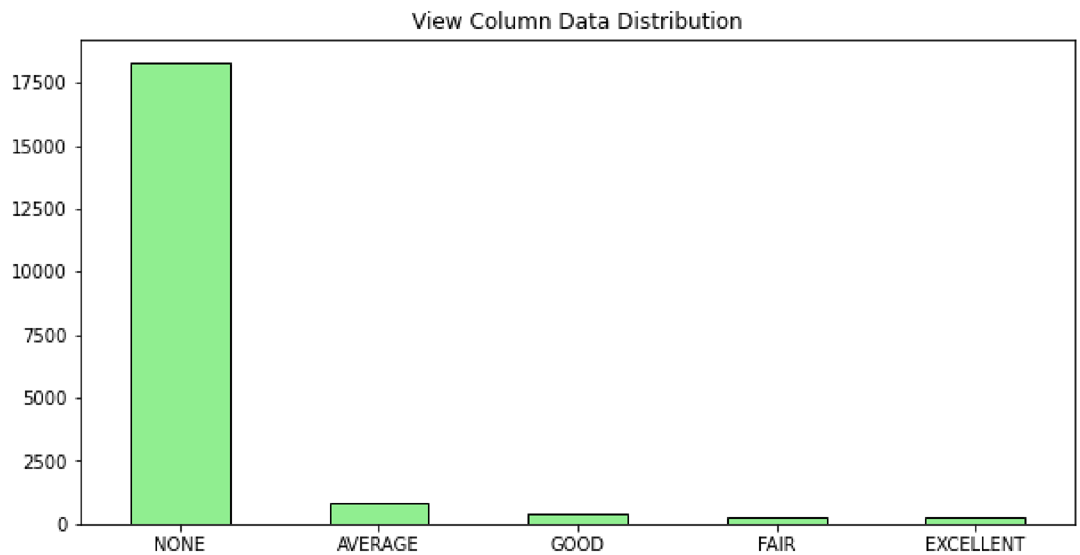




Out[17]:
```
NO     19928
YES      102
Name: waterfront, dtype: int64
```

In [18]:
```python
fun_plot_value_counts(house_df, 'view', 'View Column Data Distribution')
```

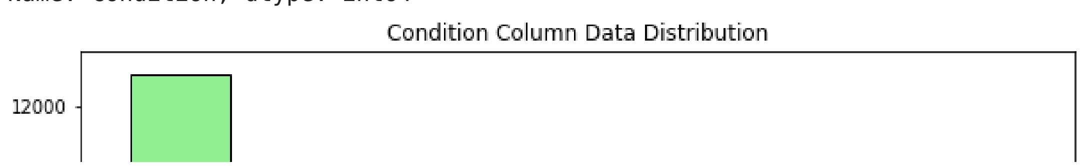
```
NONE         18292
AVERAGE        809
GOOD           388
FAIR           303
EXCELLENT      238
Name: view, dtype: int64
```

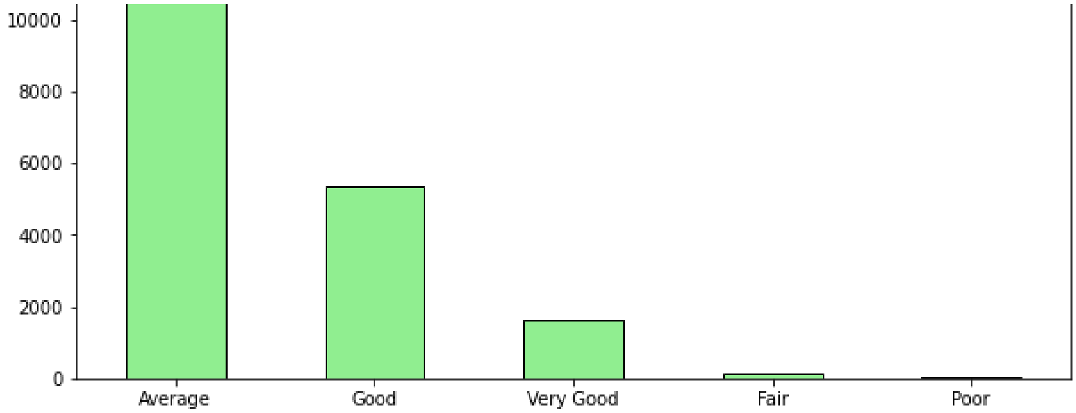


Out[18]:
```
NONE         18292
AVERAGE        809
GOOD           388
FAIR           303
EXCELLENT      238
Name: view, dtype: int64
```

In [19]:
```python
fun_plot_value_counts(house_df, 'condition', 'Condition Column Data Distribution')
```

```
Average      12881
Good          5368
Very Good     1609
Fair           145
Poor            27
Name: condition, dtype: int64
```

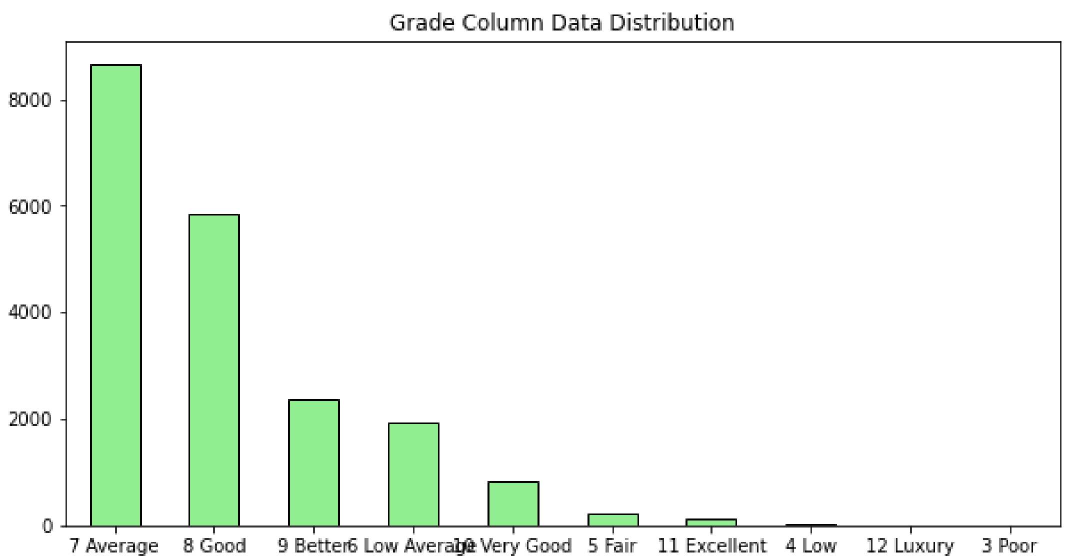




Out[19]:
```
Average      12881
Good          5368
Very Good     1609
Fair           145
Poor            27
Name: condition, dtype: int64
```

In [20]:
```
fun_plot_value_counts(house_df, 'grade', 'Grade Column Data Distribution')
```

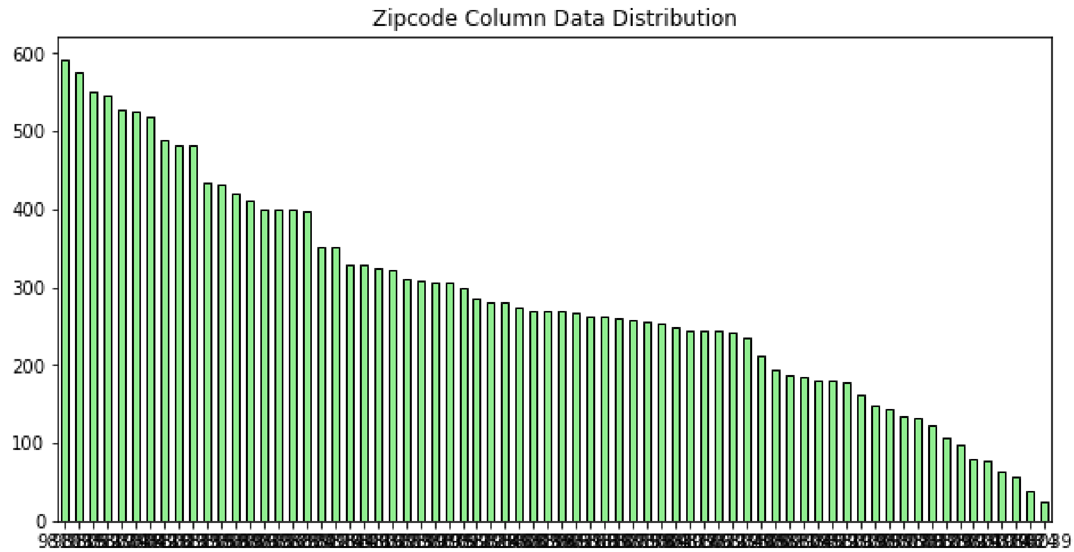
```
7 Average        8656
8 Good           5859
9 Better         2366
6 Low Average    1943
10 Very Good      832
5 Fair            220
11 Excellent      119
4 Low              27
12 Luxury           7
3 Poor              1
Name: grade, dtype: int64
```



Out[20]:
```
7 Average        8656
8 Good           5859
9 Better         2366
6 Low Average    1943
10 Very Good      832
5 Fair            220
11 Excellent      119
4 Low              27
12 Luxury           7
3 Poor              1
Name: grade, dtype: int64
```

In [21]:
```
fun_plot_value_counts(house_df, 'zipcode', 'Zipcode Column Data Distribution')
```

```
98103    592
98115    575
98052    550
98117    545
98034    527
        ...
98010     77
98070     64
98148     56
98024     39
98039     25
Name: zipcode, Length: 70, dtype: int64
```



Out[21]:
```
98103    592
98115    575
98052    550
98117    545
98034    527
        ...
98010     77
98070     64
98148     56
98024     39
98039     25
Name: zipcode, Length: 70, dtype: int64
```

Numerical Columns

There are 15 Numerical Columns in the dataset that we shall be analysing:

- `date`
- `price`
- `bedrooms`
- `bathrooms`
- `sqft_living`
- `sqft_lot`
- `floors`

- sqft_above
- sqft_basement
- yr_built
- yr_renovated
- lat
- long
- sqft_living15
- sqft_lot15

In [22]:

```
def fun_describe_and_plot_distribution(df, col, title):
    '''
    Returns the statistics of a column in a dataframe and
    plots the distribution of a column in a dataframe as a histogram, kde, and box
    '''
    # print the statistics
    print(df[col].describe())

    # create a figure composed of two matplotlib.Axes objects (ax_box and ax_hist,
    f, (ax_box, ax_hist) = plt.subplots(2, sharex=True, gridspec_kw={"height_ratio

    # assign a graph to each ax
    sns.boxplot(df[col], ax=ax_box, color='lightgreen')
    sns.histplot(data=df, x=col, ax=ax_hist, kde=True, color='lightgreen', bins='a

    # set the title and layout
    plt.suptitle(title)
    plt.tight_layout()

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

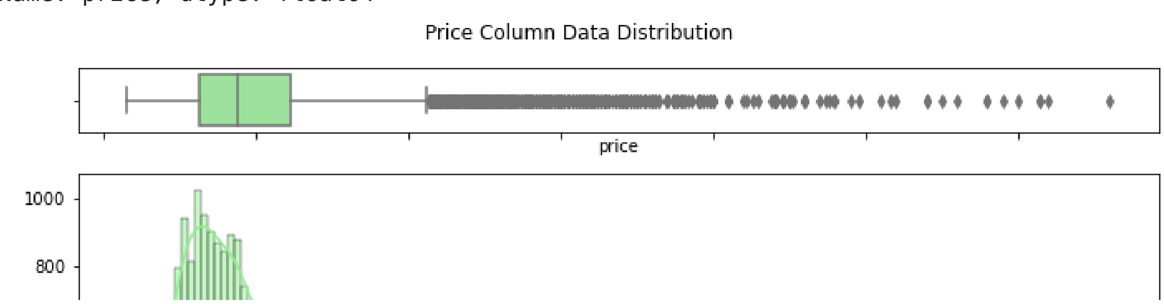
In [23]:

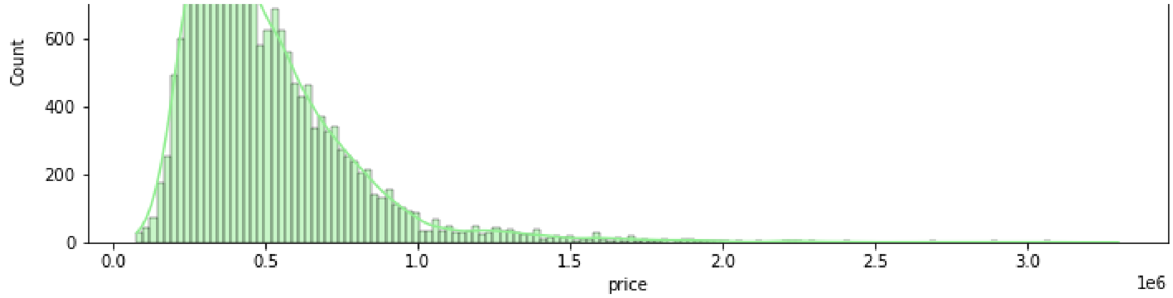
```
# fun_describe_and_plot_distribution(house_df, 'season', 'Seasons Column Data Dist
```

In [24]:

```
fun_describe_and_plot_distribution(house_df, 'price', 'Price Column Data Distribut
```

```
count    2.003000e+04
mean     5.035120e+05
std      2.786262e+05
min      7.800000e+04
25%      3.150000e+05
50%      4.400000e+05
75%      6.150000e+05
max      3.300000e+06
Name: price, dtype: float64
```



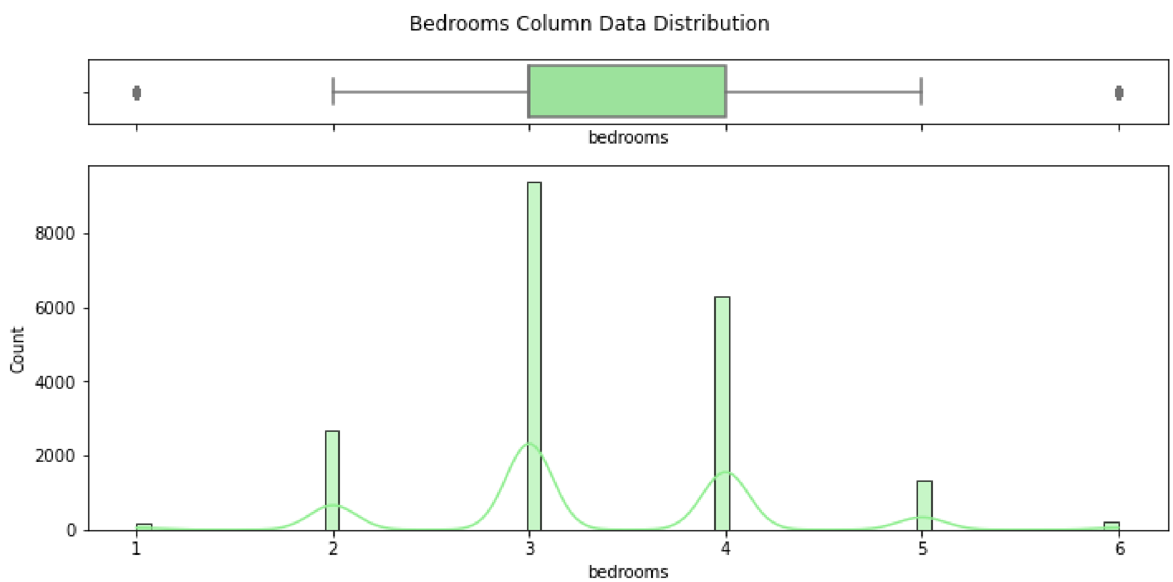


From the distribution above, we see that the price column is skewed to the right. This means that the mean price of the homes in the dataset are skewed too. The minimum price of a house in the dataset is 78,000, and the maximum price of a house in the dataset is 7,700,000. The mean price of a house in the dataset is 540,297, and the median price of a house in the dataset is 450,000. The standard deviation of the price column is 367,368.

In [25]:

```
fun_describe_and_plot_distribution(house_df, 'bedrooms', 'Bedrooms Column Data Dis
```

```
count    20030.000000
mean         3.327559
std          0.858153
min          1.000000
25%          3.000000
50%          3.000000
75%          4.000000
max          6.000000
Name: bedrooms, dtype: float64
```



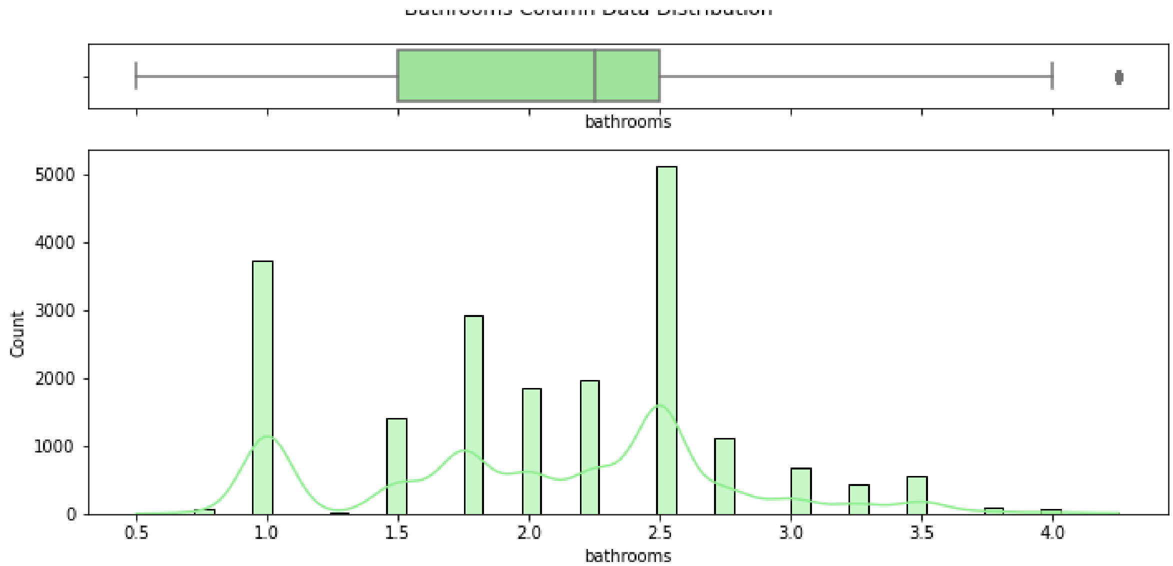
The bedroom column distribution is not skewed as the and is normally distributed.

In [26]:

```
fun_describe_and_plot_distribution(house_df, 'bathrooms', 'Bathrooms Column Data D
```

```
count    20030.000000
mean         2.052322
std          0.694386
min          0.500000
25%          1.500000
50%          2.250000
75%          2.500000
max          4.250000
Name: bathrooms, dtype: float64
```

Bathrooms Column Data Distribution

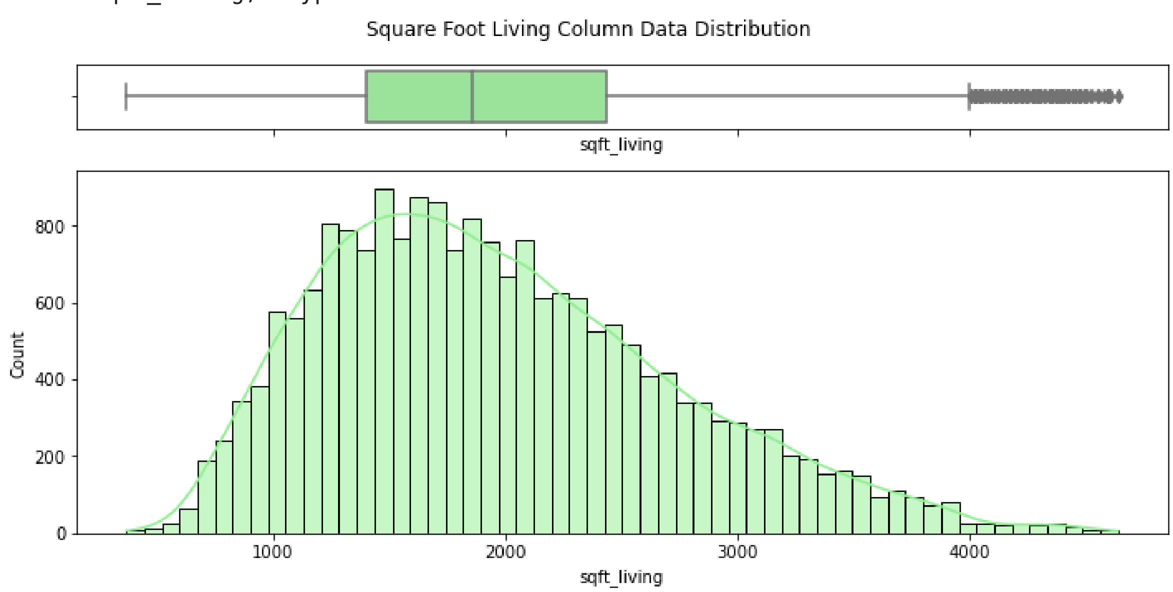


From the distribution above we can see that the bathroom column is not skewed. This is because the mean and median are almost the same. The minimum number of bathrooms in a house in the dataset is 0.5, and the maximum number of bathrooms in a house in the dataset is 8. The mean number of bathrooms in a house in the dataset is 2.12, and the median number of bathrooms in a house in the dataset is 2.25. The standard deviation of the bathrooms column is 0.77.

In [27]:

```
fun_describe_and_plot_distribution(house_df, 'sqft_living', 'Square Foot Living Co
```

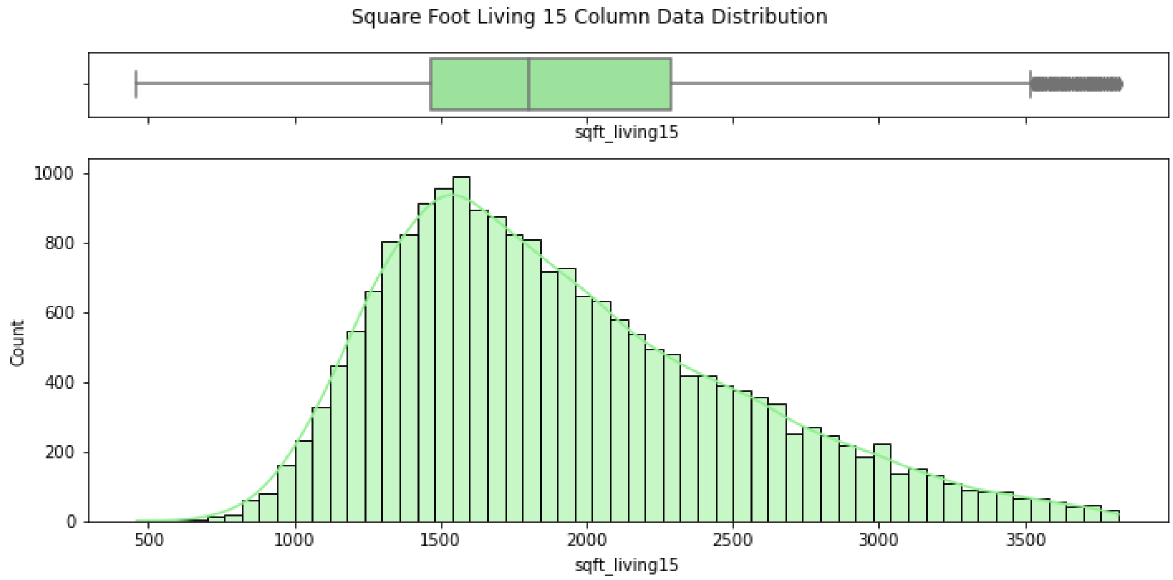
```
count    20030.000000
mean      1973.766700
std        752.717717
min        370.000000
25%       1400.000000
50%       1860.000000
75%       2440.000000
max       4640.000000
Name: sqft_living, dtype: float64
```



From the distribution above, we can see that the sqft living column is skewed to the right. This means that the mean square footage of the homes is greater than the median. The minimum square footage of a house in the dataset is 370, and the maximum square footage of a house in the dataset is 13,540. The mean square footage of a house in the dataset is 2080, and the median square footage of a house in the dataset is 1910. The standard deviation of the sqft living column is 918.

In [28]:
```
fun_describe_and_plot_distribution(house_df, 'sqft_living15', 'Square Foot Living
```

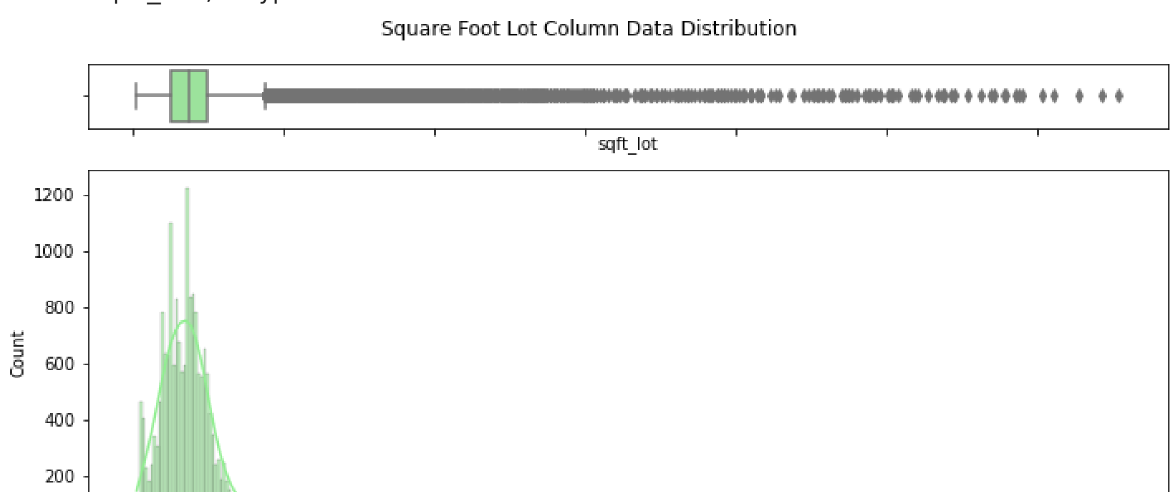
```
count    20030.000000
mean      1919.320569
std        598.045901
min        460.000000
25%       1470.000000
50%       1800.000000
75%       2290.000000
max       3820.000000
Name: sqft_living15, dtype: float64
```



From the distributions above, we can see that the data is skewed to the right. This is as a result of the mean being greater than the median.

In [29]:
```
fun_describe_and_plot_distribution(house_df, 'sqft_lot', 'Square Foot Lot Column D
```

```
count     20030.000000
mean       9641.882177
std       10316.776038
min         520.000000
25%        5000.000000
50%        7385.500000
75%        9996.750000
max      130680.000000
Name: sqft_lot, dtype: float64
```

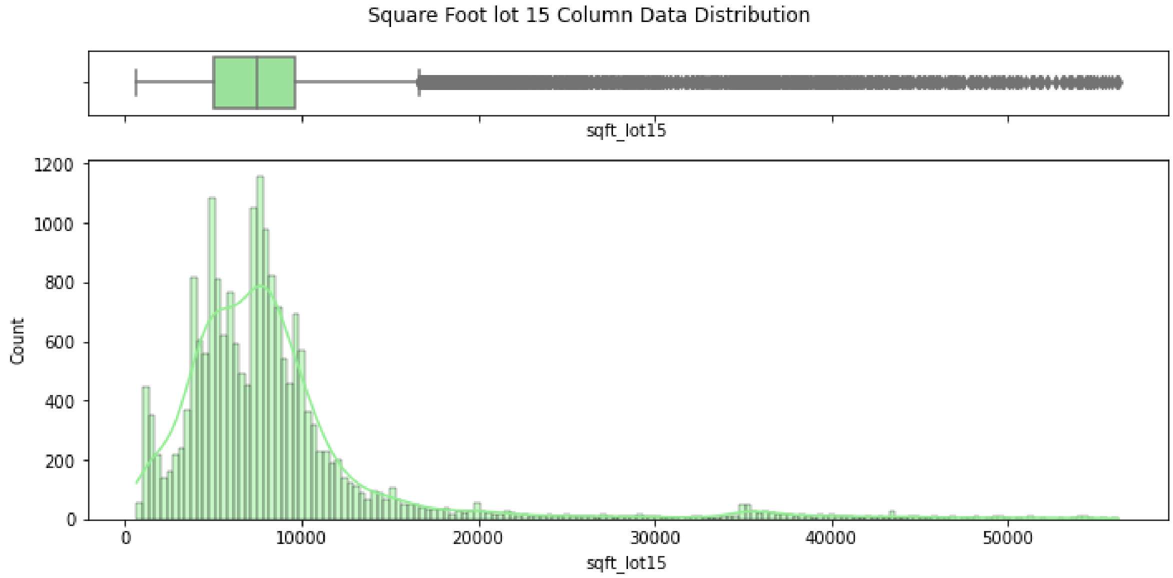


From the distribution above, we can see that the data is skewed to the right. This is because the mean is greater than the median.

In [30]:

```
fun_describe_and_plot_distribution(house_df, 'sqft_lot15', 'Square Foot lot 15 Col
```

```
count    20030.000000
mean      8810.821568
std       7404.445813
min        651.000000
25%       5000.000000
50%       7456.000000
75%       9654.250000
max      56257.000000
Name: sqft_lot15, dtype: float64
```

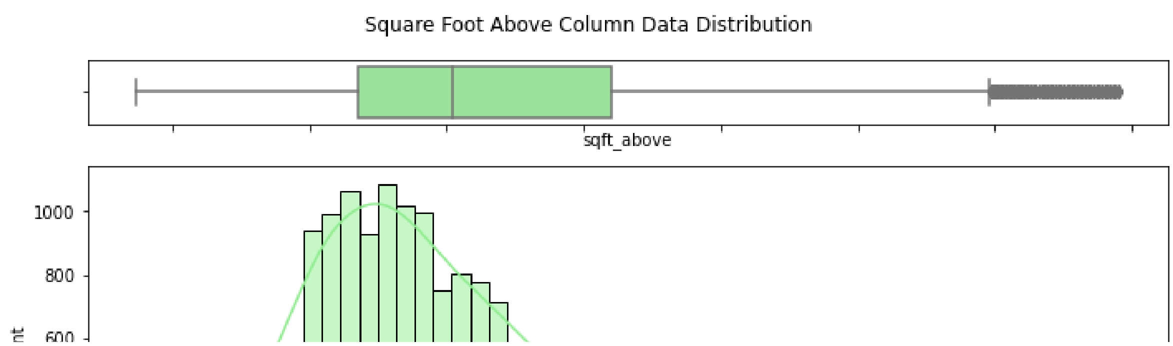


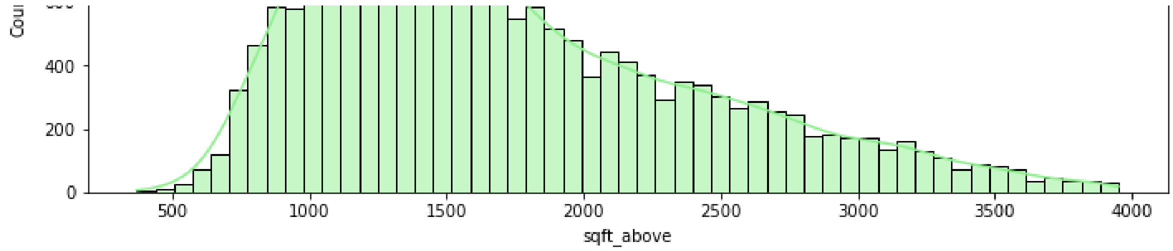
In the distributions above we see a much more skewed to the right column.

In [31]:

```
fun_describe_and_plot_distribution(house_df, 'sqft_above', 'Square Foot Above Colu
```

```
count    20030.000000
mean      1694.759960
std        690.221085
min        370.000000
25%       1180.000000
50%       1520.000000
75%       2100.000000
max       3950.000000
Name: sqft_above, dtype: float64
```

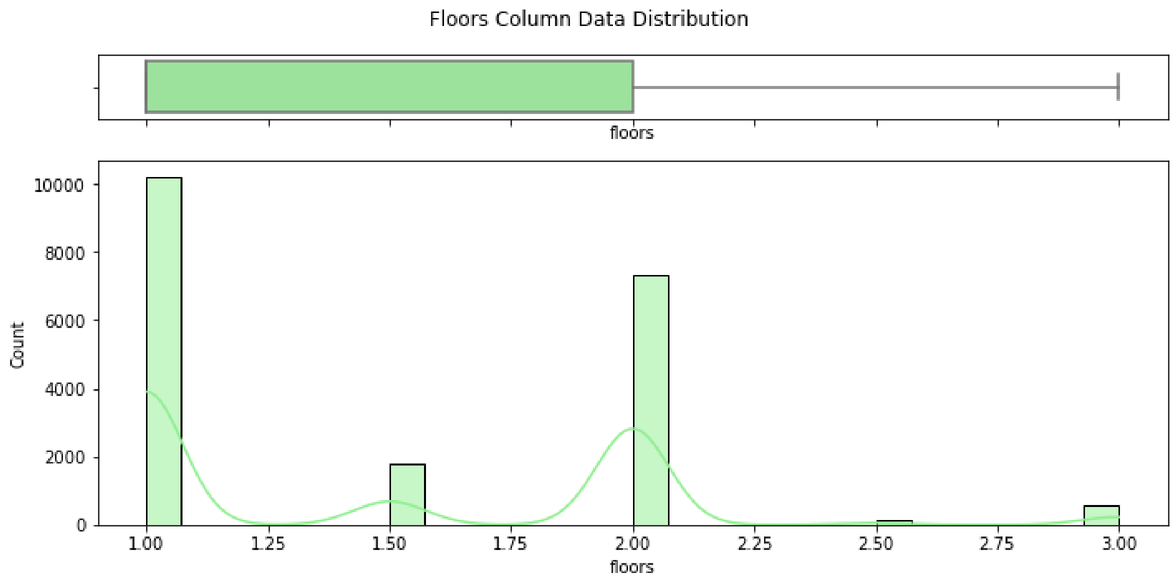




From the distributions above, we see that the square footage above ground of the houses in this dataset is skewed to the right. This is because the mean is greater than the median.

In [32]:
```
fun_describe_and_plot_distribution(house_df, 'floors', 'Floors Column Data Distrib
```

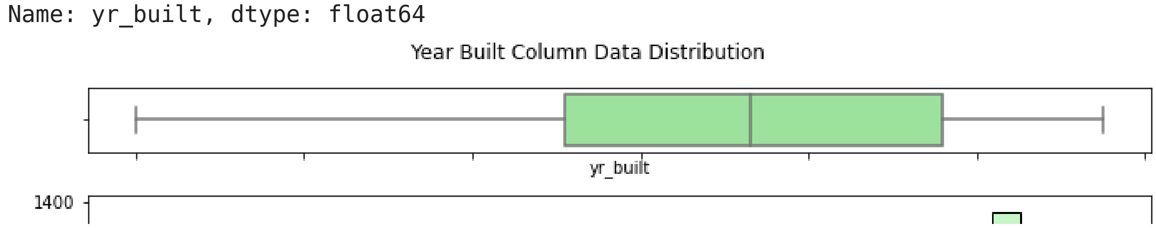
```
count    20030.000000
mean         1.479381
std          0.539099
min          1.000000
25%          1.000000
50%          1.000000
75%          2.000000
max          3.000000
Name: floors, dtype: float64
```

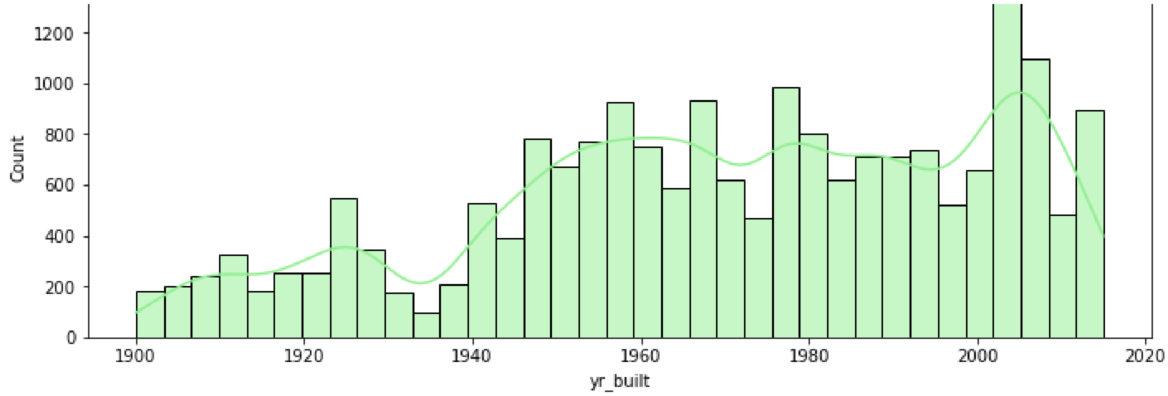


From the distributions above, there is no particular trend in the floors column data.

In [33]:
```
fun_describe_and_plot_distribution(house_df, 'yr_built', 'Year Built Column Data D
```

```
count    20030.000000
mean      1970.181428
std         29.437234
min       1900.000000
25%       1951.000000
50%       1973.000000
75%       1996.000000
max       2015.000000
Name: yr_built, dtype: float64
```



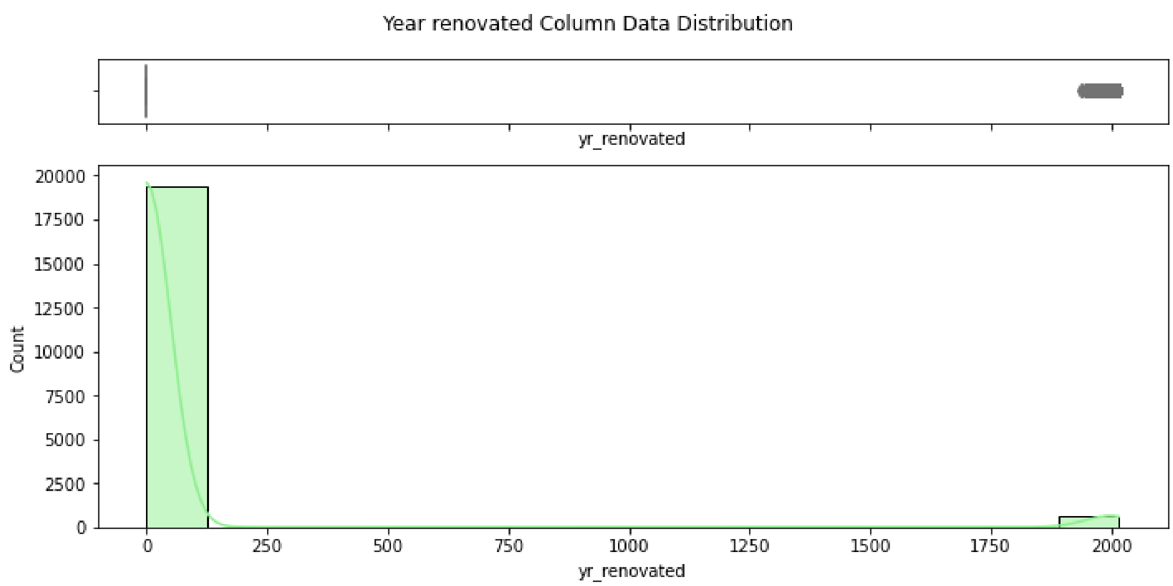


From the distributions above we can see that the data is slightly skewed to the left. This is because the mean is slightly lower than the median. The oldest house in the dataset was built in 1900, and the newest house in the dataset was built in 2015. The mean year the houses in the dataset were built is 1971, and the median year the houses in the dataset were built is 1975. The standard deviation of the yr built column is 29.

In [34]:

```
fun_describe_and_plot_distribution(house_df, 'yr_renovated', 'Year renovated Colum
```

```
count    20030.000000
mean        67.169695
std        359.978163
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max       2015.000000
Name: yr_renovated, dtype: float64
```



From the distribution and value counts above, we can see that the data has a number of zeros. This could either be suggesting that the house has not been renovated, or that the data is missing. Furthermore, there is also some missing data in this column. We shall be analysing the data more indepth in the next phase to see how to deal with the zeros and the missing values in the column.

DATA PREPARATION

Use the date feature to create a new feature called season, which represents whether the home was sold in Spring, Summer, Fall, or Winter.

In [35]:

```
#To create a new feature called "season" in the King County housing dataset,
#we can extract the month information from the "date" column and map it to
#the corresponding season.

# Convert the date column to a datetime object
house_df['date'] = pd.to_datetime(house_df['date'])

# Extract the month information and map it to the corresponding season
seasons = {1: 'Winter', 2: 'Winter', 3: 'Spring', 4: 'Spring', 5: 'Spring',
           6: 'Summer', 7: 'Summer', 8: 'Summer', 9: 'Fall', 10: 'Fall',
           11: 'Fall', 12: 'Winter'}
house_df['season'] = house_df['date'].dt.month.map(seasons)
seasons = house_df['season']
house_df
```

Out[35]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfro
0	7129300520	2014-10-13	221900.0	3	1.00	1180	5650	1.0	N
1	6414100192	2014-12-09	538000.0	3	2.25	2570	7242	2.0	N
2	5631500400	2015-02-25	180000.0	2	1.00	770	10000	1.0	N
3	2487200875	2014-12-09	604000.0	4	3.00	1960	5000	1.0	N
4	1954400510	2015-02-18	510000.0	3	2.00	1680	8080	1.0	N
...	...	...	...	...	...	...	...	...	
21592	263000018	2014-05-21	360000.0	3	2.50	1530	1131	3.0	N
21593	6600060120	2015-02-23	400000.0	4	2.50	2310	5813	2.0	N
21594	1523300141	2014-06-23	402101.0	2	0.75	1020	1350	2.0	N
21595	291310100	2015-01-16	400000.0	3	2.50	1600	2388	2.0	N
21596	1523300157	2014-10-15	325000.0	2	0.75	1020	1076	2.0	N

20030 rows × 22 columns

With the new "season" feature, we can now analyze the King County housing dataset with respect to the season and identify any seasonality trends in the housing market. For example, we can investigate whether homes sell for higher prices in certain seasons, or whether certain types of homes are more popular in certain seasons.

adding a column to store the age of the houses

In [36]:

```
# Add house_age column
house_df['age'] = house_df['date'].dt.year - house_df['yr_built']
```

```
house_df
```

Out[36]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfro
0	7129300520	2014-10-13	221900.0	3	1.00	1180	5650	1.0	N
1	6414100192	2014-12-09	538000.0	3	2.25	2570	7242	2.0	N
2	5631500400	2015-02-25	180000.0	2	1.00	770	10000	1.0	N
3	2487200875	2014-12-09	604000.0	4	3.00	1960	5000	1.0	N
4	1954400510	2015-02-18	510000.0	3	2.00	1680	8080	1.0	N
...	...	...	...	...	...	...	...	...	
21592	263000018	2014-05-21	360000.0	3	2.50	1530	1131	3.0	N
21593	6600060120	2015-02-23	400000.0	4	2.50	2310	5813	2.0	N
21594	1523300141	2014-06-23	402101.0	2	0.75	1020	1350	2.0	N
21595	291310100	2015-01-16	400000.0	3	2.50	1600	2388	2.0	N
21596	1523300157	2014-10-15	325000.0	2	0.75	1020	1076	2.0	N

20030 rows × 23 columns

removing null values in the 'yr_built" column and adding the 'renovated' column to show whether the house has been renovated or not

In [37]:

```
house_df.loc[house_df.yr_renovated.isnull(), 'yr_renovated'] = 0
house_df['renovated'] = house_df['yr_renovated'].apply(lambda x: 0 if x == 0 else
house_df
```

Out[37]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfro
0	7129300520	2014-10-13	221900.0	3	1.00	1180	5650	1.0	N
1	6414100192	2014-12-09	538000.0	3	2.25	2570	7242	2.0	N
2	5631500400	2015-02-25	180000.0	2	1.00	770	10000	1.0	N
3	2487200875	2014-12-09	604000.0	4	3.00	1960	5000	1.0	N
4	1954400510	2015-02-18	510000.0	3	2.00	1680	8080	1.0	N
...	...	...	...	...	...	...	...	...	
21592	263000018	2014-05-21	360000.0	3	2.50	1530	1131	3.0	N

21593	6600060120	2015-02-23	400000.0	4	2.50	2310	5813	2.0	N
21594	1523300141	2014-06-23	402101.0	2	0.75	1020	1350	2.0	N
21595	291310100	2015-01-16	400000.0	3	2.50	1600	2388	2.0	N
21596	1523300157	2014-10-15	325000.0	2	0.75	1020	1076	2.0	N

20030 rows × 24 columns

In [38]:
```
lot = house_df['sqft_lot']
```

Add has_basement column that is a binary value

In [39]:
```
house_df['sqft_basement'] = house_df['sqft_basement'].replace('?', '0').astype('fl
house_df['has_basement'] = house_df['sqft_basement'].apply(lambda x: 0 if x == 0 e
house_df['has_basement']
```

Out[39]:
```
0        0
1        1
2        0
3        1
4        0
        ..
21592    0
21593    0
21594    0
21595    0
21596    0
Name: has_basement, Length: 20030, dtype: int64
```

Ordinal Encoding

Ordinal encoding converts each label into integer values and the encoded data represents the sequence of labels

the values in the `condition` and `grade` columns are ordinal, and have been assigned a value based on the quality of the feature. Therefore, we will be ordinal encoding these columns.

Creating a function that maps the ordinal values in a dataframe column to corresponding numerical values based on a provided dictionary.

In [40]:
```
def map_ordinal_values(df, col_name, value_dict):
    # map the ordinal values to numerical values using the provided dictionary
    df[col_name] = df[col_name].map(value_dict)
    return df
```

In [41]:
```
condition_dict = {'Poor': 1, 'Fair': 2, 'Average': 3, 'Good': 4, 'Very Good': 5}
grade_dict = {'3 Poor': 3, '4 Low': 4, '5 Fair': 5, '6 Low Average': 6, '7 Average

df = map_ordinal_values(house_df, 'condition', condition_dict)
```

```
df = map_ordinal_values(house_df, 'grade', grade_dict)

print(house_df[['condition', 'grade']])
```

```
       condition  grade
0              3      7
1              3      7
2              3      6
3              5      7
4              3      8
...          ...    ...
21592          3      8
21593          3      8
21594          3      7
21595          3      8
21596          3      7

[20030 rows x 2 columns]
```

One Hot Encoding

One hot encoding is a process of converting categorical data variables so they can be provided to machine learning algorithms to improve predictions.

We shall be encoding the categorical columns (`waterfront` and `view`) using one hot encoding. Furthermore, in order to avoid the "Dummy Variable Trap" (perfect multicollinearity between the independent variables), we will need to drop one of the columns created.

In [42]:
```
house_df.select_dtypes('object')
```

Out[42]:

	waterfront	view	season
0	NO	NONE	Fall
1	NO	NONE	Winter
2	NO	NONE	Winter
3	NO	NONE	Winter
4	NO	NONE	Winter
...	...	...	...
21592	NO	NONE	Spring
21593	NO	NONE	Winter
21594	NO	NONE	Summer
21595	NO	NONE	Winter
21596	NO	NONE	Fall

20030 rows × 3 columns

creating a function to perform one hot encoding on the specified columns

In [43]:
```
def one_hot_encode(df, columns):
    df = pd.get_dummies(df, columns=columns, drop_first=False)
    return df
```

one hot encoding categorical variable

one hot encoding categorical variable

In [44]:

```python
columns_to_encode = ['waterfront', 'view', 'season']
house_df = one_hot_encode(house_df, columns_to_encode)
# Preview the dataframe
house_df
```

Out[44]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	conditio
0	7129300520	2014-10-13	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	2014-12-09	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	2015-02-25	180000.0	2	1.00	770	10000	1.0	
3	2487200875	2014-12-09	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	2015-02-18	510000.0	3	2.00	1680	8080	1.0	
...	...	...	...	...	...	...	...	...	.
21592	263000018	2014-05-21	360000.0	3	2.50	1530	1131	3.0	
21593	6600060120	2015-02-23	400000.0	4	2.50	2310	5813	2.0	
21594	1523300141	2014-06-23	402101.0	2	0.75	1020	1350	2.0	
21595	291310100	2015-01-16	400000.0	3	2.50	1600	2388	2.0	
21596	1523300157	2014-10-15	325000.0	2	0.75	1020	1076	2.0	

20030 rows × 33 columns

In the `waterfront` column, we shall be dropping the `waterfront_NO` column as the reference column. This will allow us to study the effect of having a house on a waterfront. In the `view` column, we shall be dropping the `view_NONE` column as the reference column. This will allow us to study the effect of having a house with a view. In addition, it is the most common value in the column.In the `season` column, we shall be dropping the `season_Fall` column as the reference column

In [45]:

```python
# Drop the 'waterfront_NO' and 'view_NONE' columns
house_df. drop(['waterfront_NO', 'view_NONE', 'season_Fall'], axis=1, inplace=True
# Preview the dataframe
house_df
```

Out[45]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	conditio
0	7129300520	2014-10-13	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	2014-12-09	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	2015-02-25	180000.0	2	1.00	770	10000	1.0	

3	2487200875	2014-12-09	604000.0	4	3.00	1960	5000	1.0	!
4	1954400510	2015-02-18	510000.0	3	2.00	1680	8080	1.0	:
...	...	...	...	...	...	...	...	...	.
21592	263000018	2014-05-21	360000.0	3	2.50	1530	1131	3.0	:
21593	6600060120	2015-02-23	400000.0	4	2.50	2310	5813	2.0	:
21594	1523300141	2014-06-23	402101.0	2	0.75	1020	1350	2.0	:
21595	291310100	2015-01-16	400000.0	3	2.50	1600	2388	2.0	:
21596	1523300157	2014-10-15	325000.0	2	0.75	1020	1076	2.0	:

20030 rows × 30 columns

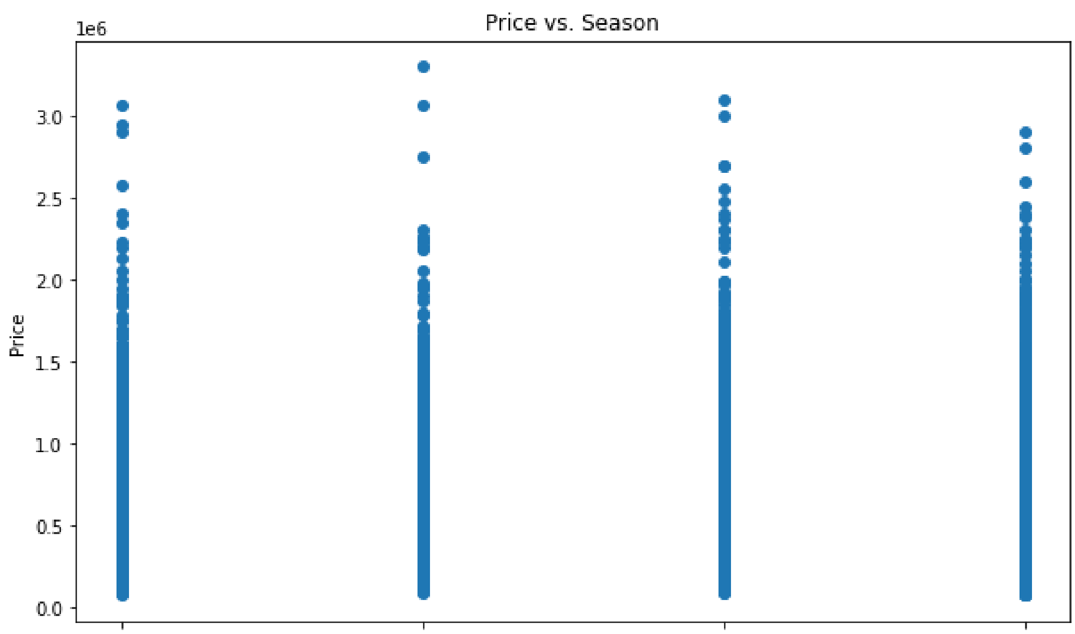
Bi-variate analysis

We will now compare price against 2 variables

What is the relationship between price and season?

In [46]:

```
# create a scatter plot

fig = plt.figure(figsize=(10, 6))
plt.scatter(seasons, house_df['price'])
# add labels and title
plt.xlabel('Season')
plt.ylabel('Price')
plt.title('Price vs. Season')

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



From the plot above, we see that seasons have an influence on price with the fall and summer attracting the highest price.

What is the relationship between price and square foot living?

In [47]:

```
fig, ax = plt.subplots(figsize=(10, 8))

ax.scatter(house_df['sqft_living'], house_df['price'])
ax.set_xlabel('Sqft Living')
ax.set_ylabel('Price')
ax.set_title('Price vs. Sqft Living')

plt.show()
```

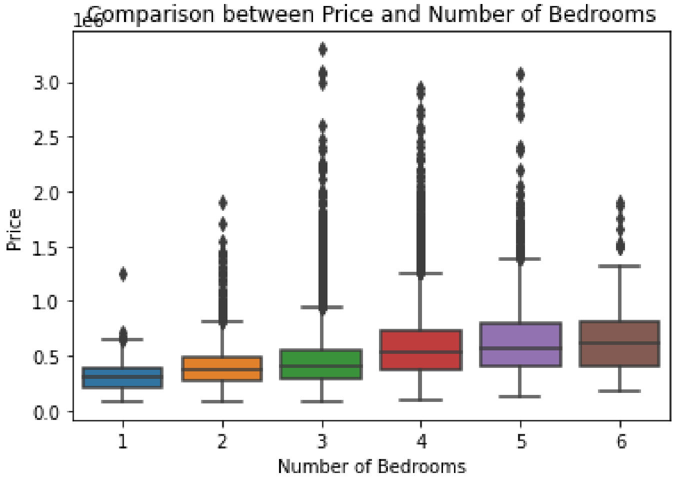
From the above plot, we see that as the square foot living and lot increases, the price of the house increases.

What is the relationship between price and number of bedrooms?

In [48]:

```
# create a box plot
sns.boxplot(x="bedrooms", y="price", data=house_df)

# set the plot title and labels for x and y axes
plt.title("Comparison between Price and Number of Bedrooms")
plt.xlabel("Number of Bedrooms")
plt.ylabel("Price")

# show the plot
```

```
plt.show()
```

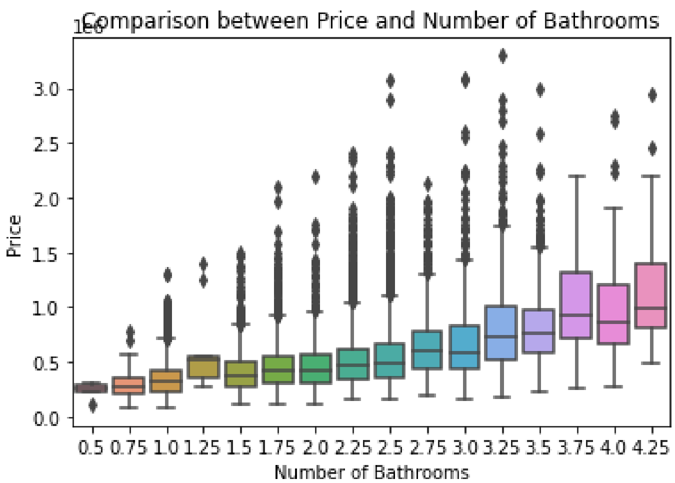


From the plot above we see that the number of bedrooms influence the price af a house. However, the reaches a point where too many bedrooms becomes a negative effect of price.

What is the relationship between price and number of bathrooms?

In [49]:

```
# create a box plot
sns.boxplot(x="bathrooms", y="price", data=house_df)

# set the plot title and labels for x and y axes
plt.title("Comparison between Price and Number of Bathrooms")
plt.xlabel("Number of Bathrooms")
plt.ylabel("Price")

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



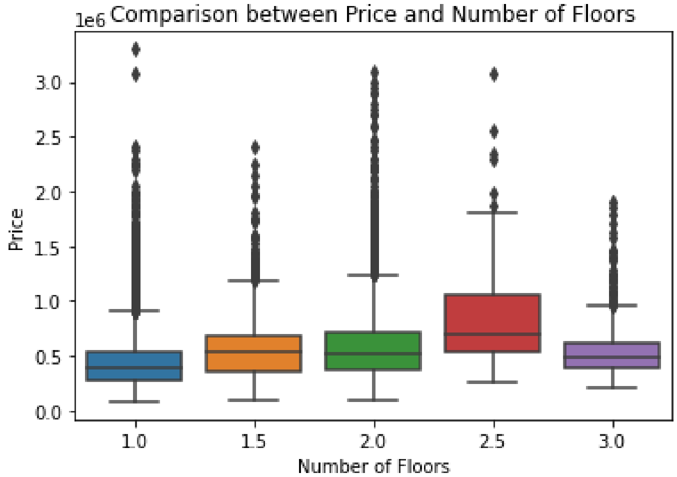
The plot above indicates that the higher number of bathrooms, the higher the price.

What is the relationship between price and number of floors?

In [50]:

```
# create a box plot
sns.boxplot(x="floors", y="price", data=house_df)

# set the plot title and labels for x and y axes
```

```
plt.title("Comparison between Price and Number of Floors")
plt.xlabel("Number of Floors")
plt.ylabel("Price")

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



This plot shows that the number of floors a house may have has average influence in the value of the house. It however shows that houses with 2 to 3 floors are popular among buyers.

Multi-variate analysis

We will compare price against more that 3 variables

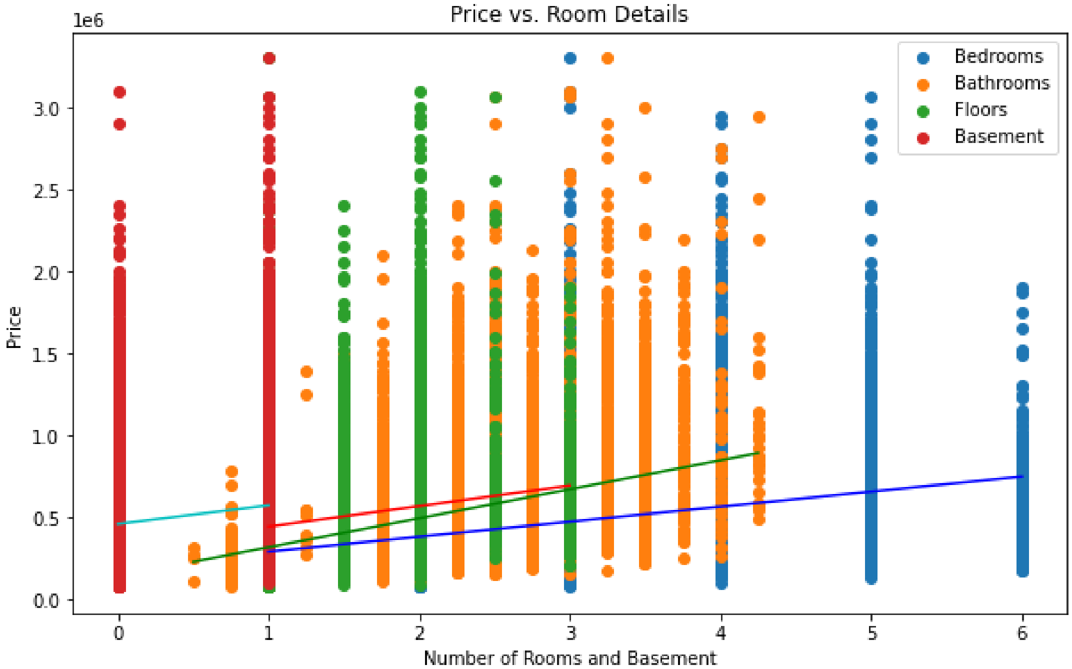
What is the relationship between the price and number of bedrooms, bathrooms and floors ?

We shall be doing this exploration to compare how much the number of bedrooms, bathrooms and florr compare against each other in relation to price.

In [51]:

```
# from mpl_toolkits.mplot3d import Axes3D

# Create the figure and axis objects
fig, ax = plt.subplots(figsize=(10, 6))

# Plot the scatter plot for each column
ax.scatter(house_df['bedrooms'], house_df['price'], label='Bedrooms')
ax.scatter(house_df['bathrooms'], house_df['price'], label='Bathrooms')
ax.scatter(house_df['floors'], house_df['price'], label='Floors')
ax.scatter(house_df['has_basement'], house_df['price'], label='Basement')

# Add labels and legend
ax.set_xlabel('Number of Rooms and Basement')
ax.set_ylabel('Price')
ax.set_title('Price vs. Room Details')
ax.legend()

# Add trend lines
for col, color in zip(['bedrooms', 'bathrooms', 'floors', 'has_basement'], ['b', '
    # Fit a polynomial function of degree 1 to the data
    z = np.polyfit(house_df[col], house_df['price'], 1)
    p = np.poly1d(z)

    # Create an array of x values
    x_vals = np.array([house_df[col].min(), house_df[col].max()])
```

```
    # Calculate the corresponding y values and plot the line
    y_vals = p(x_vals)
    ax.plot(x_vals, y_vals, '-', label=f'{col} Trend', color=color)

# Display the plot
plt.show()
```



From the scatter plot above, we see that the slopes and thus the rate of price increase for bedrooms, bathrooms, floors and basements increase according to their rarity.

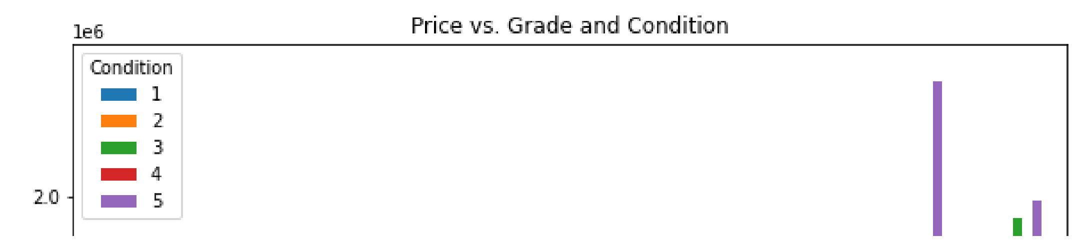
We can thus assume that the more rare and few these are, the more the price will increase.

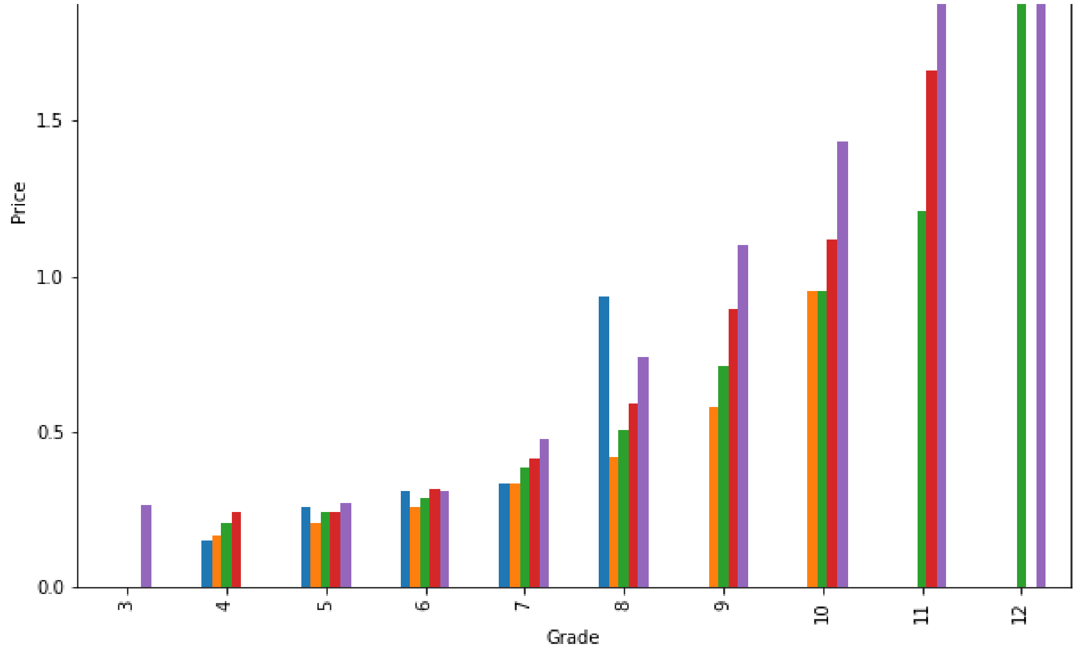
What is the relationship between price, grade and condition?

We shall now compare price against the grade and condition.

In [52]:

```
# Create a pivot table to calculate mean price for each grade and condition combi
pivot_table = pd.pivot_table(house_df, values='price', index='grade', columns='con

# Plot the pivot table as a bar plot
ax = pivot_table.plot(kind='bar', figsize=(10, 8))

# Add labels and legend
ax.set_xlabel('Grade')
ax.set_ylabel('Price')
ax.set_title('Price vs. Grade and Condition')
ax.legend(title='Condition')

# Display the plot
plt.show()
```



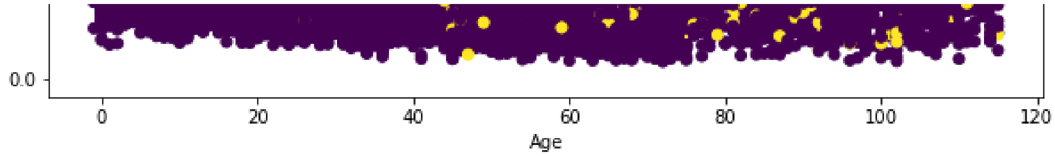


From the above plot, we see that as grade and condition increase, the higher the price, especially for those with a condition of 5. The worse the condition and grade the lower the price.

What is the relationship between price and age, renovated?

In [53]:
```
fig, ax = plt.subplots(figsize=(10, 8))

ax.scatter(house_df['age'], house_df['price'], c=house_df['renovated'])
ax.set_xlabel('Age')
ax.set_ylabel('Price')
ax.legend()
ax.set_title('Price vs. Age vs. Renovated')

plt.show()
```

```
No handles with labels found to put in legend.
```



From the plot, we can see that renovations occur more frequently among older building and doing such renovations has a positive impact on their price.

Checking correlations and multicollinearity

In [54]:
```
house_df.corr()
```

Out[54]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
id	1.000000	-0.008270	0.005718	0.024677	0.012803	-0.107635	0.024023
price	-0.008270	1.000000	0.282174	0.440603	0.623043	0.092917	0.241889
bedrooms	0.005718	0.282174	1.000000	0.496057	0.607494	0.111114	0.157110
bathrooms	0.024677	0.440603	0.496057	1.000000	0.712198	0.050901	0.503035
sqft_living	0.012803	0.623043	0.607494	0.712198	1.000000	0.218753	0.334110
sqft_lot	-0.107635	0.092917	0.111114	0.050901	0.218753	1.000000	-0.119389
floors	0.024023	0.241889	0.157110	0.503035	0.334110	-0.119389	1.000000
condition	-0.027916	0.070289	0.036232	-0.120546	-0.039438	0.043904	-0.267990
grade	0.029128	0.626410	0.340348	0.619258	0.710863	0.129745	0.451666
sqft_above	0.019265	0.502771	0.484544	0.629012	0.837322	0.203195	0.537598
sqft_basement	-0.008557	0.289836	0.289118	0.240747	0.413808	0.058493	-0.283909
yr_built	0.034235	0.002622	0.159174	0.524902	0.311543	0.019074	0.492443
yr_renovated	-0.012569	0.126178	0.010032	0.040745	0.051556	0.015250	0.001823
zipcode	-0.026690	-0.003531	-0.161549	-0.195782	-0.184091	-0.182094	-0.053585
lat	-0.011465	0.377709	-0.033740	0.004767	0.038784	-0.067197	0.038623
long	0.054700	-0.012057	0.142438	0.218518	0.225209	0.258683	0.121466
sqft_living15	0.016121	0.546453	0.399472	0.528372	0.736968	0.266462	0.255098
sqft_lot15	-0.093999	0.097050	0.124251	0.060161	0.238195	0.809582	-0.128148
age	-0.034080	-0.002422	-0.159277	-0.525261	-0.311906	-0.019272	-0.492705
renovated	-0.012558	0.125836	0.009752	0.040265	0.051298	0.015457	0.001745
has_basement	-0.004550	0.198082	0.155917	0.161110	0.217696	-0.022211	-0.266301
waterfront_YES	-0.002750	0.222289	-0.016683	0.031479	0.061412	0.064959	0.011844
view_AVERAGE	0.020403	0.151342	0.039010	0.068175	0.117444	0.017642	-0.004619
view_EXCELLENT	0.021739	0.272935	0.012371	0.063733	0.117978	0.057543	0.010604
view_FAIR	-0.002327	0.102233	0.017040	0.033663	0.072610	0.014573	-0.026747
view_GOOD	-0.008608	0.174653	0.036263	0.077964	0.131662	0.042586	0.003696
season_Spring	0.014103	0.030879	-0.003778	-0.013663	-0.013591	-0.014140	-0.008687
season_Summer	0.002016	0.010271	0.011326	0.024646	0.026018	0.010224	0.017215

season_Winter	-0.001923	-0.028869	0.001965	-0.014340	-0.010489	0.006902	-0.014947

creating a function that takes in our dataframe and returns corellations between column in pair in descending order

In [55]:

```
def get_correlation_df(df):
    corr_df = df.corr().abs().stack().reset_index().sort_values(0, ascending=False
    corr_df['pairs'] = list(zip(corr_df.level_0, corr_df.level_1))
    corr_df.set_index(['pairs'], inplace=True)
    corr_df.drop(columns=['level_1', 'level_0'], inplace=True)
    corr_df.columns = ['cc']
    corr_df = corr_df.drop_duplicates()
    return corr_df.head(50)
```

let call our function on our dataframe

In [56]:

```
get_correlation_df(house_df)
```

Out[56]:

pairs	cc
(id, id)	1.000000
(renovated, yr_renovated)	0.999968
(age, yr_built)	0.999874
(sqft_living, sqft_above)	0.837322
(sqft_basement, has_basement)	0.835333
(sqft_lot15, sqft_lot)	0.809582
(sqft_living15, sqft_living)	0.736968
(bathrooms, sqft_living)	0.712198
(sqft_living, grade)	0.710863
(sqft_living15, sqft_above)	0.703387
(sqft_above, grade)	0.702474
(sqft_living15, grade)	0.665941
(bathrooms, sqft_above)	0.629012
(grade, price)	0.626410
(price, sqft_living)	0.623043
(bathrooms, grade)	0.619258
(sqft_living, bedrooms)	0.607494
(view_EXCELLENT, waterfront_YES)	0.581211
(zipcode, long)	0.568047
(sqft_living15, price)	0.546453
(sqft_above, floors)	0.537598
(bathrooms, sqft_living15)	0.528372
(age, bathrooms)	0.525261

(yr_built, bathrooms)	0.524902
(floors, bathrooms)	0.503035
(sqft_above, price)	0.502771
(bedrooms, bathrooms)	0.496057
(age, floors)	0.492705
(floors, yr_built)	0.492443
(sqft_above, bedrooms)	0.484544
(floors, grade)	0.451666
(age, grade)	0.447949
(yr_built, grade)	0.447534
(bathrooms, price)	0.440603
(sqft_above, age)	0.434932
(yr_built, sqft_above)	0.434654
(season_Spring, season_Summer)	0.422424
(sqft_living, sqft_basement)	0.413808
(yr_built, long)	0.406052
(long, age)	0.406045
(sqft_living15, bedrooms)	0.399472
(lat, price)	0.377709
(condition, yr_built)	0.358682
(age, condition)	0.357964
(long, sqft_above)	0.348177
(bedrooms, grade)	0.340348
(yr_built, zipcode)	0.338290
(zipcode, age)	0.338264
(floors, sqft_living)	0.334110
(sqft_living15, long)	0.322773

we will drop columns that have strong multicollinearity or provide no use to the model

In [57]:

```
house_df = house_df.drop(columns=['id', 'yr_renovated', 'sqft_lot', 'sqft_above',
house_df
```

Out[57]:

	price	bedrooms	bathrooms	sqft_living	floors	condition	grade	age	renovated	ha
0	221900.0	3	1.00	1180	1.0	3	7	59	0	
1	538000.0	3	2.25	2570	2.0	3	7	63	1	
2	180000.0	2	1.00	770	1.0	3	6	82	0	
3	604000.0	4	3.00	1960	1.0	5	7	49	0	
4	510000.0	3	2.00	1680	1.0	3	8	28	0	
...	...	...	...	...	...	...	...	...	...	

21592	360000.0	3	2.50	1530	3.0	3	8	5	0
21593	400000.0	4	2.50	2310	2.0	3	8	1	0
21594	402101.0	2	0.75	1020	2.0	3	7	5	0
21595	400000.0	3	2.50	1600	2.0	3	8	11	0
21596	325000.0	2	0.75	1020	2.0	3	7	6	0

20030 rows × 18 columns

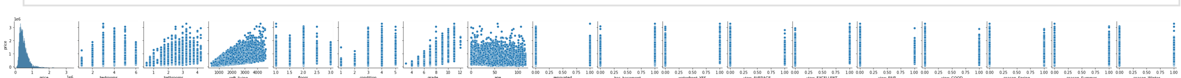
There is still some multicollinearity between predictor variable, but not strong enough to initially drop on our models.

In [58]:
```
house_df.corr()['price'].sort_values(ascending=False)
```

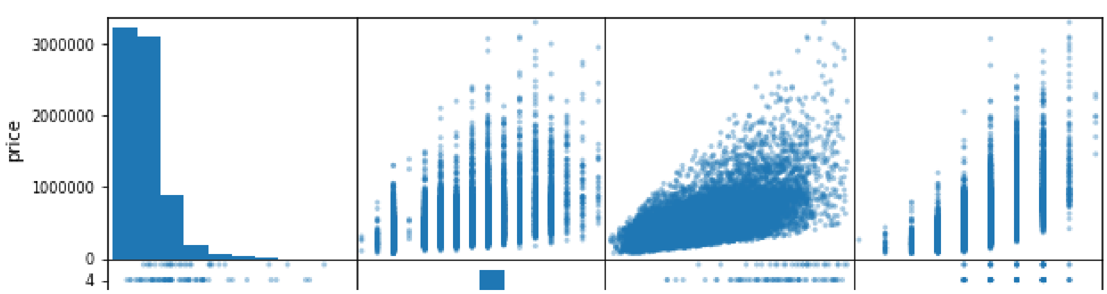
Out[58]:
```
price             1.000000
grade             0.626410
sqft_living       0.623043
bathrooms         0.440603
bedrooms          0.282174
view_EXCELLENT    0.272935
floors            0.241889
waterfront_YES    0.222289
has_basement      0.198082
view_GOOD         0.174653
view_AVERAGE      0.151342
renovated         0.125836
view_FAIR         0.102233
condition         0.070289
season_Spring     0.030879
season_Summer     0.010271
age              -0.002422
season_Winter    -0.028869
Name: price, dtype: float64
```

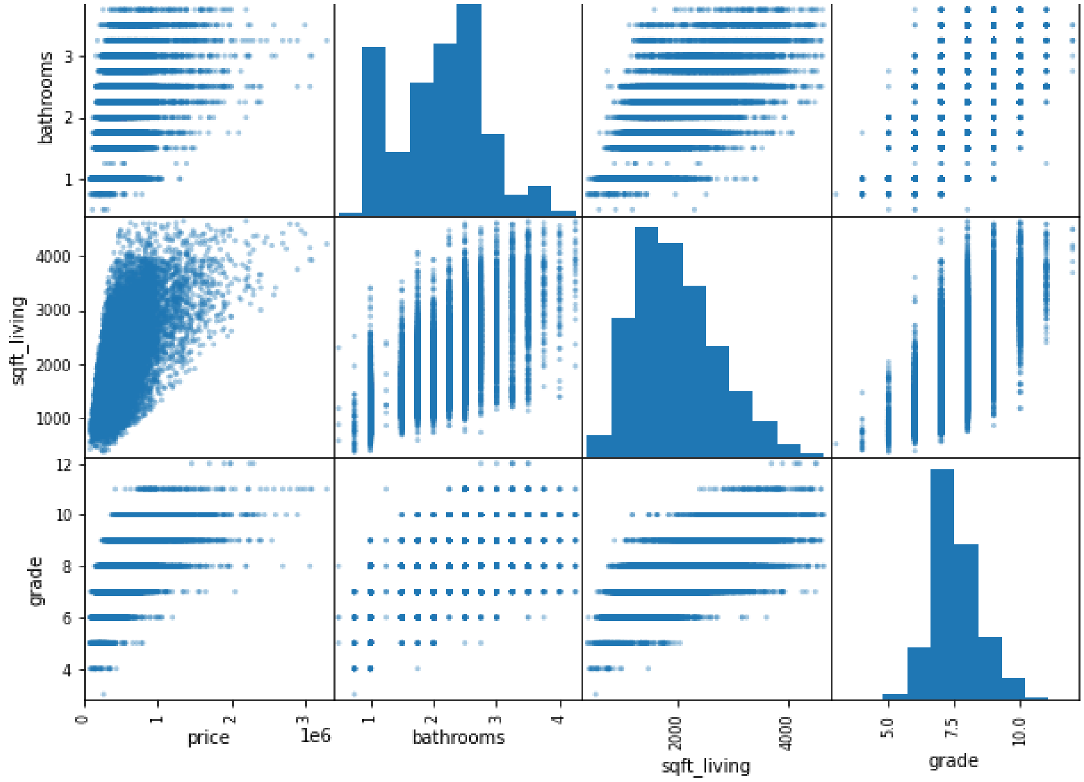
we want to visualize the collinearity using plots

In [59]:
```
# Visualizing how each variable distributes? with price
sns.pairplot(house_df, y_vars='price');
```



In [60]:
```
# A further look at certain attributes
attributes = ['price', 'bathrooms', 'sqft_living', 'grade']

pd.plotting.scatter_matrix(house_df[attributes], figsize = [10, 10], alpha=0.4);
plt.show()
```





we dont notice any problems with multicollinearity

In [61]:

```
house_df.head(5)
```

Out[61]:

	price	bedrooms	bathrooms	sqft_living	floors	condition	grade	age	renovated	has_ba
0	221900.0	3	1.00	1180	1.0	3	7	59	0	
1	538000.0	3	2.25	2570	2.0	3	7	63	1	
2	180000.0	2	1.00	770	1.0	3	6	82	0	
3	604000.0	4	3.00	1960	1.0	5	7	49	0	
4	510000.0	3	2.00	1680	1.0	3	8	28	0	

REGRESSION MODELLING

Regression is, in my opinion, the finest algorithm to try in this experiment. Based on the values of the independent variables, regression is a supervised learning process used to forecast the value of a dependent variable. In this instance, we're attempting to estimate the impact that various property characteristics have on our dependent variable, the homes' prices. As a result, we will be able to offer our stakeholders a model that can foretell the key characteristics of homes that will have the most effects on their prices.

We will also use multiple linear regression because we are working with numerous features. Contrary to linear regression, which only employs one independent variable, multiple linear regression uses the values of many independent variables to predict the value of a dependent variable.

Building a Baseline Model

We will first start by building a baseline model. The baseline model will be used to compare the performance of the other models that we will be building. After that, we will build our multiple linear regression model.

The target variable is price. Therefore, we look at the correlation coefficients for all of the predictor variables to find the one with the highest correlation with price.

In [62]:

```
corr = house_df.corr()['price'].sort_values(ascending=False)
corr
```

Out[62]:

```
price            1.000000
grade            0.626410
sqft_living      0.623043
bathrooms        0.440603
bedrooms         0.282174
view_EXCELLENT   0.272935
floors           0.241889
waterfront_YES   0.222289
has_basement     0.198082
view_GOOD        0.174653
view_AVERAGE     0.151342
renovated        0.125836
view_FAIR        0.102233
condition        0.070289
season_Spring    0.030879
season_Summer    0.010271
age             -0.002422
season_Winter   -0.028869
Name: price, dtype: float64
```

We can see that the 'price' column and the'sqft_living' column have the strongest association. This is understandable given that a large portion of a house's price is determined by its size. In order to see the relationship between "sqft_living" and "price," we will also make a scatter plot.

In [63]:

```
# Plot a scatter plot of the 'price' column against the 'sqft_living' column
plt.figure(figsize=(10, 5))
plt.scatter(house_df['sqft_living'], house_df['price'], color='b', alpha=0.7, s=1(
plt.title('Price vs Living Space')
plt.xlabel('Living Space (sqft)')
plt.ylabel('Price');
```

We may now declare the variables y and X_baseline, where y is a Series with pricing data and X_baseline is a DataFrame with the column with the highest correlation ('sqft_living').

In [64]:

```
y = house_df['price']

X_baseline = house_df[['sqft_living']]
```

we'll use our variables to build and fit a simple linear regression model

In [65]:

```
baseline_model = sm.OLS(y, sm.add_constant(X_baseline))
baseline_results = baseline_model.fit()
```

lets evaluate the model

In [66]:

```
print(baseline_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.388
Model:                            OLS   Adj. R-squared:                  0.388
Method:                 Least Squares   F-statistic:                 1.271e+04
Date:                Thu, 20 Apr 2023   Prob (F-statistic):               0.00
Time:                        23:22:37   Log-Likelihood:            -2.7463e+05
No. Observations:               20030   AIC:                         5.493e+05
Df Residuals:                   20028   BIC:                         5.493e+05
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const        4.831e+04   4321.770     11.178      0.000    3.98e+04    5.68e+04
sqft_living   230.6258      2.046    112.727      0.000     226.616     234.636
==============================================================================
Omnibus:                     8778.566   Durbin-Watson:                   1.981
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            80420.940
Skew:                           1.875   Prob(JB):                         0.00
Kurtosis:                      12.072   Cond. No.                     5.93e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly sp
ecified.
[2] The condition number is large, 5.93e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

lets interpret the results

In [67]:

```
baseline_results.rsquared
```

Out[67]: 0.3881828773301257

R-squared: This represents the proportion of the variance in the target variable (price) that can be explained by the independent variable (sqft_living). Here, R-squared is 0.443, which means that approximately 44.3% of the variance in housing prices can be explained by the square footage of the living area.

In [68]:
```
baseline_results.f_pvalue
```

Out[68]:
```
0.0
```

The p-value of the f-statistic is extremely small (p < 0.001), indicating that the regression model is significant overall and that the independent variable (sqft_living) is a good predictor of the dependent variable (price).

In [69]:
```
baseline_results.pvalues
```

Out[69]:
```
const          6.337036e-29
sqft_living    0.000000e+00
dtype: float64
```

the p-value for the sqft_living and const coefficients is 4.237554e-13 and 0.000000e+00 respectively are well below the significance level, indicating that they are both statistically significant.

In [70]:
```
baseline_results.params
```

Out[70]:
```
const          48310.407091
sqft_living      230.625835
dtype: float64
```

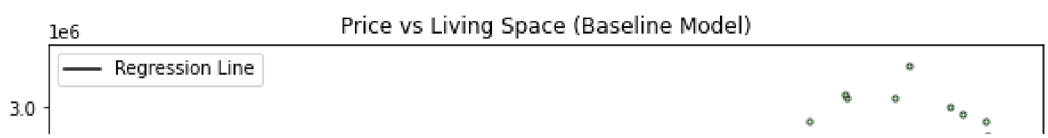
In this case, we have one independent variable, so we have one coefficient, which is 240.9939. This means that for each one-unit increase in square footage of the living area, the housing price increases by $240.99, holding other variables constant.

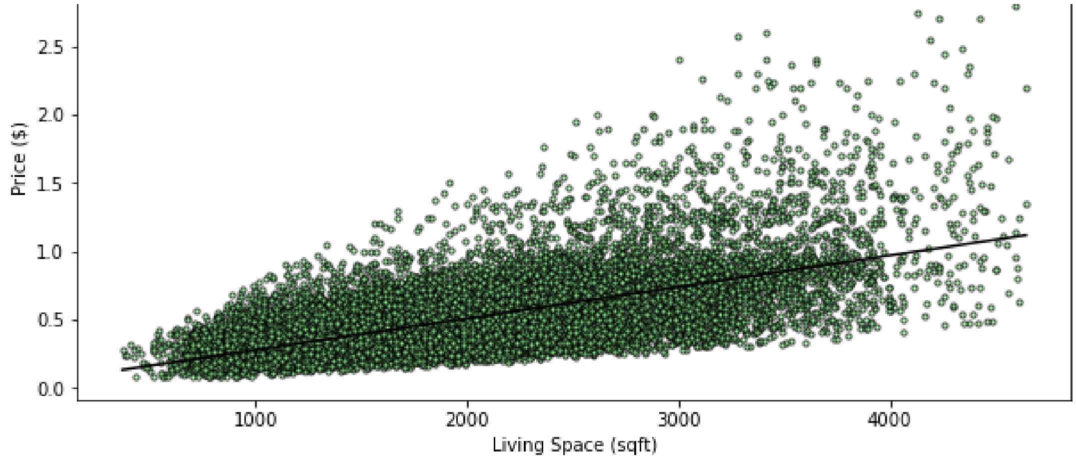
the estimated intercept value is $29,880. However, since in the context of the problem, the independent variable (sqft_living) cannot be zero, this interpretation is not particularly useful.

Confidence Intervals: These show the range within which we can be 95% confident that the true coefficient lies. Here, we can be 95% confident that the true coefficient for sqft_living is between 237.351 and 244.637.

We can plot the regression line on top of the scatter plot earlier to see how well the model fits the data.

In [71]:
```
# Plot a scatter plot of the 'price' column against the 'sqft_living' column
plt.figure(figsize=(10, 5))

# Plot the regression line of the baseline model
x = np.linspace(house_df.sqft_living.min(), house_df.sqft_living.max(), 100)
Y_predicted = baseline_results.params[0] + baseline_results.params[1] * x

plt.plot(x, Y_predicted, color='black', label='Regression Line')

plt.scatter(X_baseline, y, color='lightgreen', alpha=0.7, s=10, edgecolors='black'
plt.title('Price vs Living Space (Baseline Model)')
plt.xlabel('Living Space (sqft)')
plt.ylabel('Price (\$)')
plt.legend();
```





Calculate the mean absolute error of the baseline model

In [72]:
```
baseline_mae = mean_absolute_error(y, baseline_results.predict(sm.add_constant(X_b
baseline_mae
```

Out[72]: 154780.12328975898

This means that on average, the model's predictions for the price of a house are off by about $159,750.

This is a relatively large error, considering that the average price of a house in the dataset is around $540,000. Therefore, the model's predictions may not be very accurate and may need to be improved by either selecting additional features or by trying a different type of model.

Build Iterated Multiple Linear Regression Model

We will now iterate the baseline model by building a multiple linear regression model that will have more than one independent variable.

We will start by creating a new dataframe that will contain all of the features that we want to have in our model.

In [73]:
```
house_df.columns
```

Out[73]:
```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'floors', 'condition',
       'grade', 'age', 'renovated', 'has_basement', 'waterfront_YES',
       'view_AVERAGE', 'view_EXCELLENT', 'view_FAIR', 'view_GOOD',
       'season_Spring', 'season_Summer', 'season_Winter'],
      dtype='object')
```

In [74]:
```
X_iterated = house_df.drop(columns='price')
X_iterated.columns
```

Out[74]:
```
Index(['bedrooms', 'bathrooms', 'sqft_living', 'floors', 'condition', 'grade',
       'age', 'renovated', 'has_basement', 'waterfront_YES', 'view_AVERAGE',
       'view_EXCELLENT', 'view_FAIR', 'view_GOOD', 'season_Spring',
       'season_Summer', 'season_Winter'],
      dtype='object')
```

We will now build our multiple linear regression model.

In [75]:
```
X_iterated
```

Out[75]:

	bedrooms	bathrooms	sqft_living	floors	condition	grade	age	renovated	has_basemen
0	3	1.00	1180	1.0	3	7	59	0	
1	3	2.25	2570	2.0	3	7	63	1	
2	2	1.00	770	1.0	3	6	82	0	
3	4	3.00	1960	1.0	5	7	49	0	
4	3	2.00	1680	1.0	3	8	28	0	
...	...	...	...	...	...	...	...	...	
21592	3	2.50	1530	3.0	3	8	5	0	
21593	4	2.50	2310	2.0	3	8	1	0	
21594	2	0.75	1020	2.0	3	7	5	0	
21595	3	2.50	1600	2.0	3	8	11	0	
21596	2	0.75	1020	2.0	3	7	6	0	

20030 rows × 17 columns

In [76]:
```
iterated_model = sm.OLS(y, sm.add_constant(X_iterated))
iterated_results = iterated_model.fit()
```

lets evaluate the model

In [77]:
```
print(iterated_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.610
Model:                            OLS   Adj. R-squared:                  0.610
Method:                 Least Squares   F-statistic:                     1840.
Date:                Thu, 20 Apr 2023   Prob (F-statistic):               0.00
Time:                        23:22:41   Log-Likelihood:            -2.7012e+05
No. Observations:               20030   AIC:                         5.403e+05
Df Residuals:                   20012   BIC:                         5.404e+05
Df Model:                          17
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const          -9.588e+05    1.5e+04    -63.777      0.000   -9.88e+05   -9.29e+05
bedrooms       -2.676e+04   1875.114    -14.273      0.000   -3.04e+04   -2.31e+04
bathrooms       3.069e+04   3099.521      9.901      0.000    2.46e+04    3.68e+04
sqft_living      118.6509      3.101     38.261      0.000     112.573     124.729
floors          4.644e+04   3042.791     15.261      0.000    4.05e+04    5.24e+04
condition       2.353e+04   2078.200     11.323      0.000    1.95e+04    2.76e+04
grade           1.237e+05   1860.965     66.457      0.000     1.2e+05    1.27e+05
age             3153.2996     59.354     53.127      0.000    3036.962    3269.638
renovated       2.596e+04   7190.639      3.610      0.000    1.19e+04    4.01e+04
has_basement    2.502e+04   2905.048      8.613      0.000    1.93e+04    3.07e+04
waterfront_YES   3.74e+05    2.13e+04    17.561      0.000    3.32e+05    4.16e+05
view_AVERAGE    6.256e+04   6391.712      9.788      0.000       5e+04    7.51e+04
view_EXCELLENT  2.815e+05    1.42e+04    19.882      0.000    2.54e+05    3.09e+05
view_FAIR       1.024e+05    1.02e+04    10.059      0.000    8.25e+04    1.22e+05
view_GOOD       1.205e+05   9128.821     13.204      0.000    1.03e+05    1.38e+05
```

```
season_Spring   2.544e+04   3389.372      7.506      0.000    1.88e+04    3.21e+04
season_Summer   4949.8669   3402.068      1.455      0.146   -1718.468    1.16e+04
season_Winter   1071.2560   3914.469      0.274      0.784   -6601.427    8743.938
==============================================================================
Omnibus:                     7036.061   Durbin-Watson:                   1.966
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            60185.770
Skew:                           1.449   Prob(JB):                         0.00
Kurtosis:                      10.983   Cond. No.                     4.00e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly sp
ecified.
[2] The condition number is large,  4e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

lets interpret the results

First, we can see that the R-squared value for this model is 0.634, which means that the model explains about 63.4% of the variance in the target variable (price). This is a significant improvement over the previous model which had an R-squared value of 0.443.

The F-statistic of 2036 and the corresponding p-value of 0.00 indicate that the overall model is statistically significant, meaning that at least one of the independent variables in the model is significantly related to the target variable.

The constant term (const) in this model is -994,600. This represents the predicted price of a house with all independent variables set to zero (which is not realistic for most variables).

Now let's examine the feature coefficients.

Each coefficient represents the change in the target variable associated with a one-unit change in the corresponding independent variable, holding all other variables constant.

The coefficient for the bedrooms variable is -26,120, which means that for each additional bedroom, the predicted price of the house decreases by $26,120, holding all other variables constant.

The coefficient for the bathrooms variable is $36,100, which means that for each additional bathroom, the predicted price of the house increases by $36,100, holding all other variables constant.

The coefficient for the square footage of the living area (sqft_living) is 123.8, which means that for each additional square foot of living area, the predicted price of the house increases by $123.80, holding all other variables constant.

The coefficient for the square footage of the lot (sqft_lot) is -0.0885, which means that for each additional square foot of lot size, the predicted price of the house decreases by $0.0885, holding all other variables constant.

The coefficient for the floors variable is $39,150, which means that for each additional floor, the predicted price of the house increases by $39,150, holding all other variables constant.

The coefficient for the condition variable is $21,030, which means that for each unit increase in the condition rating (on a scale of 1-5), the predicted price of the house increases by $21,030, holding all other variables constant.

The coefficient for the grade variable is $128,000, which means that for each unit increase in the grade rating (on a scale of 1-13), the predicted price of the house increases by $128,000, holding all other variables constant.

The coefficient for the age variable is $3,264.43, which means that for each additional year of age of the house, the predicted price of the house increases by $3,264.43, holding all other variables constant.

The coefficient for the renovated variable is $25,880, which means that if the house has been renovated, the predicted price of the house increases by $25,880, holding all other variables constant.

The coefficient for the has_basement variable is $16,810, which means that if the house has a basement, the predicted price of the house increases by $16,810, holding all other variables constant.

The coefficient for the waterfront_YES variable is $236,700, which means that if the house has a waterfront view, the predicted price of the house increases by $236,700 compared to waterfront_NO (which was the reference waterfront).

The coefficients for the view variables (view_AVERAGE, view_EXCELLENT, view_FAIR, and view_GOOD) represent the additional price associated with each respective view rating, holding all other variables constant.The coefficients for all 'view' categories are positive, indicating that homes better view ratings tend to have higher prices compared to view_NONE (which was the reference view)

The coefficient for 'season_Spring' is also positive, indicating that homes tend to sell for higher prices during spring compared to fall (which was the reference season). On the other hand, the coefficients for 'season_Summer' and 'season_Winter' are not statistically significant, indicating that there is no evidence that homes sell for higher prices in summer or winter compared to fall.

We can also see that some variables have a stronger effect than others. For example, the coefficient for the waterfront view variable is much larger than the coefficients for the other variables, indicating that having a waterfront view is a very significant factor in determining the price of a house.

Overall, this model provides a more comprehensive understanding of the factors that affect the price of a house, and can be used to make more accurate predictions of house prices based on the characteristics of the house.

RMSE measure of how well the model is able to predict the outcome variable

In [78]:
```
rmse = ((iterated_results.resid ** 2).sum() / len(y)) ** 0.5
rmse
```

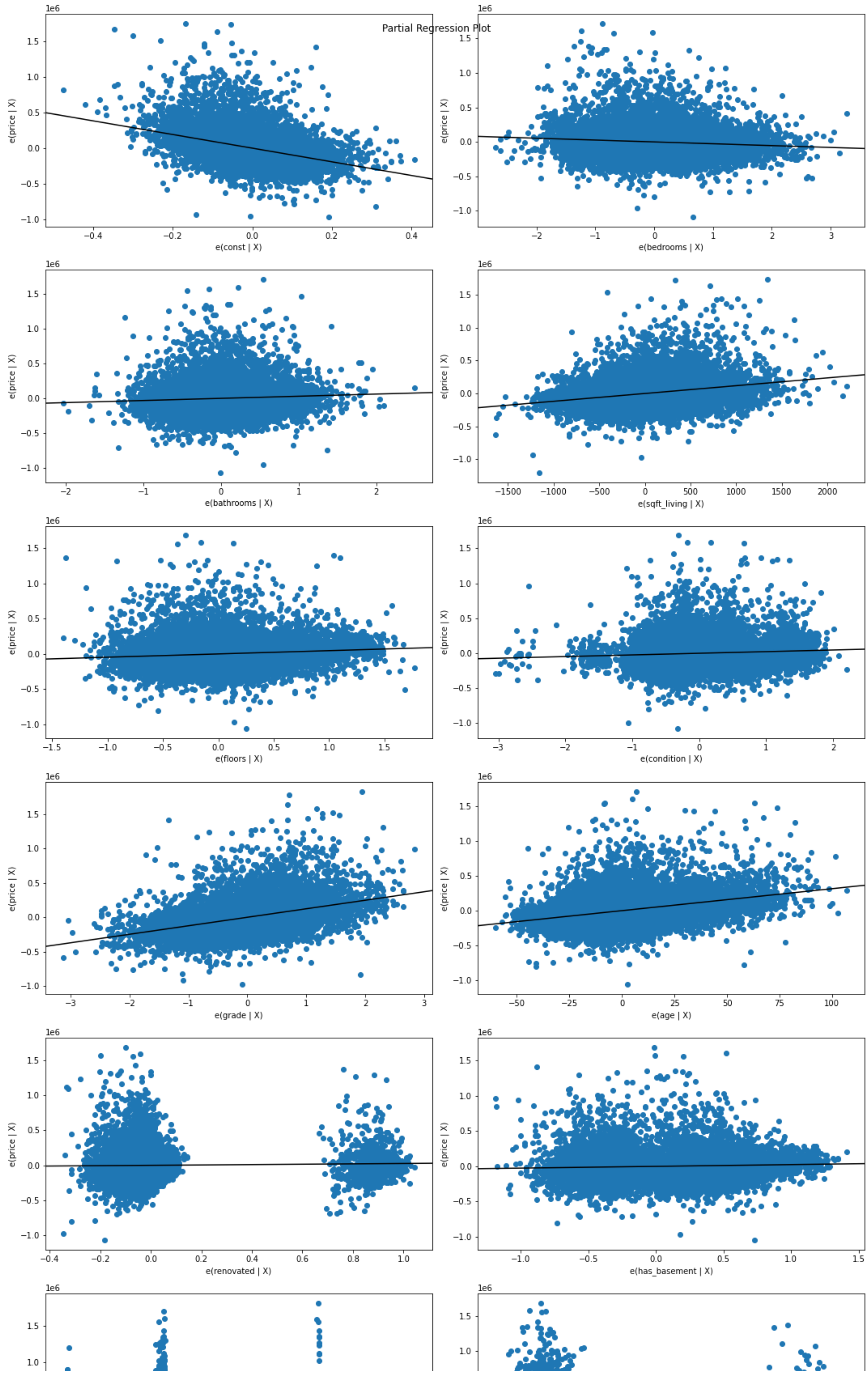
Out[78]:
```
174029.06043026305
```

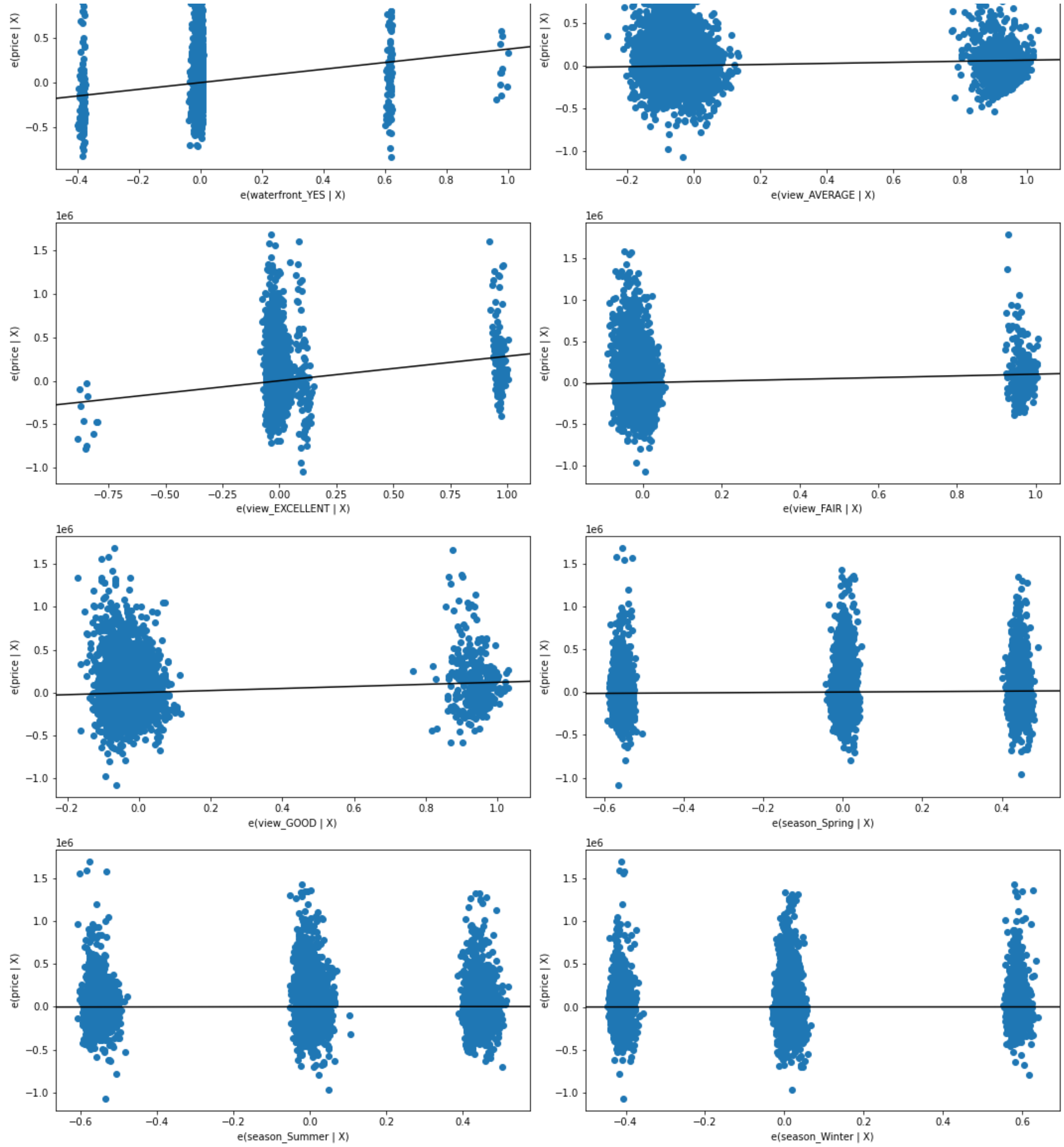
For this specific RMSE value, it means that our model is off by about 181k us dollars in a given prediction.

plotting a partial regression plot for our model for each predictor variable

In [79]:
```
# create partial regression plots for each predictor variable
fig = plt.figure(figsize=(15,40))
sm.graphics.plot_partregress_grid(iterated_results, fig=fig)
```

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





statistical test for homoscedasticity

the Goldfeld-Quandt test, which divides the dataset into two groups, then finds the MSE of the residuals for each group. The ratio of the second group's mse_resid divided by the first group's mse_resid becomes a statistic that can be compared to the f-distribution to find a p-value

In [80]:
```
from statsmodels.stats.diagnostic import het_goldfeldquandt
```

In [81]:
```
het_goldfeldquandt(y, X_iterated.values, alternative='two-sided')
```

Out[81]:
```
(1.076485483886226, 0.00022942190496135338, 'two-sided')
```

For the auto MPG data, we have a p-value of about 0.0032560392244241248, so we reject the null hypothesis at an alpha of 0.05. This means we consider the King County House data to be heteroscedastic.

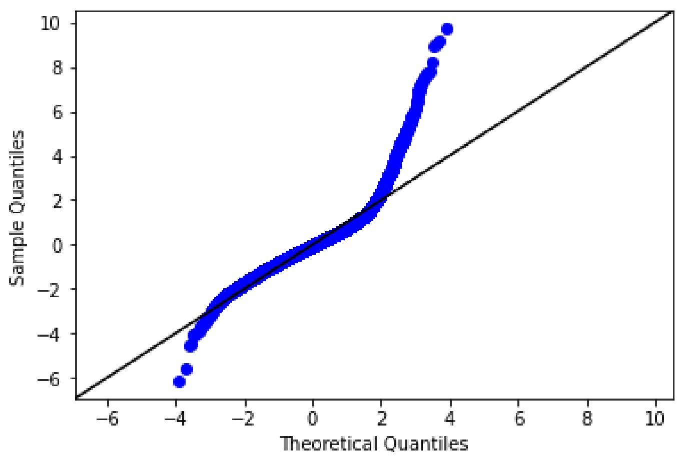
test for the Normality Assumption

Q-Q (quantile-quantile) plot is a probability plot, which is a graphical method for comparing two probability distributions by plotting their quantiles against each other

In [82]:
```
import scipy.stats as stats
```

In [83]:
```
# Use qqplot function from StatsModels
fig, ax = plt.subplots()
sm.graphics.qqplot(iterated_results.resid, dist=stats.norm, line='45', fit=True, a

# Customize plot appearance
line = ax.lines[1]
line.set_color("black")
fig.suptitle("our distribution compared to the normal Distribution");
```

our distribution compared to the normal Distribution



We see that the middle looks ok, but the ends, especially the higher end, are diverging from a normal distribution.

Building an Iterated Log-Transformed Model

We will use a non_linear transformation technique Log transformations are one of several different techniques that fundamentally reshape the modeled relationship between the variables

The reason to apply this kind of transformation is that we believe that the underlying relationship is not linear. Then by applying these techniques, we may be able to model a linear relationship between the transformed variables

Log Transforming the numerical Features

Let's try building a model that uses the log of numerical features rather than the raw values

In [84]:
```
house_df.head()
```

Out[84]:

	price	bedrooms	bathrooms	sqft_living	floors	condition	grade	age	renovated	has_ba
0	221900.0	3	1.00	1180	1.0	3	7	59	0	
1	538000.0	3	2.25	2570	2.0	3	7	63	1	

2	180000.0	2	1.00	770	1.0	3	6	82	0
3	604000.0	4	3.00	1960	1.0	5	7	49	0
4	510000.0	3	2.00	1680	1.0	3	8	28	0

our numerical_features are ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'age']

log-transform the columns with a small constant added to avoid negative or 0 values

In [85]:

```
house_df[['bedrooms', 'bathrooms', 'sqft_living', 'floors', 'age']] = house_df[['b
#preview
house_df.head()
```

Out[85]:

	price	bedrooms	bathrooms	sqft_living	floors	condition	grade	age	renova
0	221900.0	1.098612	1.000000e-08	7.073270	1.000000e-08	3	7	4.077537	
1	538000.0	1.098612	8.109302e-01	7.851661	6.931472e-01	3	7	4.143135	
2	180000.0	0.693147	1.000000e-08	6.646391	1.000000e-08	3	6	4.406719	
3	604000.0	1.386294	1.098612e+00	7.580700	1.000000e-08	5	7	3.891820	
4	510000.0	1.098612	6.931472e-01	7.426549	1.000000e-08	3	8	3.332205	

In [86]:

```
house_df.isnull().sum()
```

Out[86]:

```
price             0
bedrooms          0
bathrooms         0
sqft_living       0
floors            0
condition         0
grade             0
age              12
renovated         0
has_basement      0
waterfront_YES    0
view_AVERAGE      0
view_EXCELLENT    0
view_FAIR         0
view_GOOD         0
season_Spring     0
season_Summer     0
season_Winter     0
dtype: int64
```

mean imputation for the missing values in the age column

In [87]:

```
house_df['age'] = house_df['age'].fillna(house_df['age'].mean())
```

declare the variable X_log_iterated

In [88]:

```
X_log_iterated= house_df.drop(columns='price')
```

```
X_log_iterated= house_df.drop(columns='price')
X_log_iterated.head()
```

Out[88]:

	bedrooms	bathrooms	sqft_living	floors	condition	grade	age	renovated	has_l
0	1.098612	1.000000e-08	7.073270	1.000000e-08	3	7	4.077537	0	
1	1.098612	8.109302e-01	7.851661	6.931472e-01	3	7	4.143135	1	
2	0.693147	1.000000e-08	6.646391	1.000000e-08	3	6	4.406719	0	
3	1.386294	1.098612e+00	7.580700	1.000000e-08	5	7	3.891820	0	
4	1.098612	6.931472e-01	7.426549	1.000000e-08	3	8	3.332205	0	

Model with a Log Transformed Features

In [89]:

```
X_log_iterated_model = sm.OLS(y, sm.add_constant(X_log_iterated))
X_log_iterated_results = X_log_iterated_model.fit()
```

lets evaluate the model

In [90]:

```
print(X_log_iterated_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.539
Model:                            OLS   Adj. R-squared:                  0.539
Method:                 Least Squares   F-statistic:                     1378.
Date:                Thu, 20 Apr 2023   Prob (F-statistic):               0.00
Time:                        23:22:59   Log-Likelihood:            -2.7179e+05
No. Observations:               20030   AIC:                         5.436e+05
Df Residuals:                   20012   BIC:                         5.438e+05
Df Model:                          17
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const          -2.111e+06   4.04e+04    -52.247      0.000   -2.19e+06   -2.03e+06
bedrooms       -5.938e+04   6542.941     -9.076      0.000   -7.22e+04   -4.66e+04
bathrooms      -7.886e+04   5771.056    -13.665      0.000   -9.02e+04   -6.75e+04
sqft_living     2.052e+05   6757.425     30.366      0.000    1.92e+05    2.18e+05
floors          5.303e+04   5098.361     10.401      0.000     4.3e+04     6.3e+04
condition       5.731e+04   2168.531     26.429      0.000    5.31e+04    6.16e+04
grade           1.227e+05   1955.990     62.707      0.000    1.19e+05    1.26e+05
age             1638.5203    452.487      3.621      0.000     751.608    2525.432
renovated       1.381e+05   7508.018     18.397      0.000    1.23e+05    1.53e+05
has_basement    5.474e+04   3171.090     17.263      0.000    4.85e+04     6.1e+04
waterfront_YES  3.727e+05   2.31e+04     16.101      0.000    3.27e+05    4.18e+05
view_AVERAGE    1.024e+05   6914.073     14.814      0.000    8.89e+04    1.16e+05
view_EXCELLENT  3.329e+05   1.54e+04     21.664      0.000    3.03e+05    3.63e+05
view_FAIR       1.374e+05    1.1e+04     12.434      0.000    1.16e+05    1.59e+05
view_GOOD       1.696e+05   9888.102     17.157      0.000     1.5e+05    1.89e+05
season_Spring    2.44e+04   3690.250      6.611      0.000    1.72e+04    3.16e+04
season_Summer   2944.5059   3697.303      0.796      0.426   -4302.513    1.02e+04
season_Winter  -1678.6285   4257.537     -0.394      0.693       -1e+04    6666.495
==============================================================================
Omnibus:                     7333.985   Durbin-Watson:                   1.972
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            55553.627
```

```
Skew:                           1.563   Prob(JB):                         0.00
Kurtosis:                      10.536   Cond. No.                         363.
==============================================================================

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

lets interpret the results

The log-transformed features have improved the R-squared value of the model, indicating that the model is better at explaining the variability of the response variable.

The coefficients of the log-transformed features can be interpreted as follows:

bedrooms: For each increase of 1% in the number of bedrooms, we see a decrease of $478.2 in the price (coefficient is negative).

bathrooms: For each increase of 1% in the number of bathrooms, we see a decrease of $837.6 in