King County House Sales Analysis

Author: [Femi Kamau (https://www.github.com/ctrl-Karugu)](https://www.github.com/ctrl-Karugu)

Overview

In our analysis, we explored the data provided by the stakeholder and build a multiple linear
features stipulated in the dataset. From there, we analysed the results and came to a conclu
have a significant impact on the price of a house in King County:

- Have a house by the water
- Increase the number of bedrooms
- Improve the overall grade of the home
- Increase the number of floors
- Increase the size of the basement
- Strive to maintain the house to ensure that it is in good condition

1. Business Understanding

A real estate angency located in King County is looking to advice home owners about how h
the value of their homes. The agency is looking to use the King County house data provided
renovations to make to increase the value of a home.

2. Data Understanding

> This phase is broken down into four tasks together with its projected outcome or outp
>
> - Collect Initial Data
> - Describe Data
> - Explore Data
> - Verify Data Quality

There was no need to collect any data for this project as it was already provided by the stake
house data from King County and is in .csv format.

Load Libraries

In [274]:

```
# data
import numpy as np
import pandas as pd

# visualization
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
import folium
import warnings

# modeling
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

# statistics
import scipy.stats as stats

# styling
plt.style.use('seaborn')
sns.set_style('whitegrid')

warnings.filterwarnings('ignore')
```

Import Data

In [275]:

```python
# King County House Sales dataset is imported and assigned to the variable 'data'
data = pd.read_csv('../data/raw/kc_house_data.csv')

# The shape of the dataframe and the last 5 rows are outputted
print(data.shape)
data.tail()
```

```
(21597, 21)
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	NO
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	NO
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	NO
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	NaN
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	2.0	NO

5 rows × 21 columns

There are 21 columns and 21597 rows in the dataset:

- **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 n

`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, construction and design, and 11-13 have a high quality level of construction and design

  `zipcode` - What zipcode area the house is in

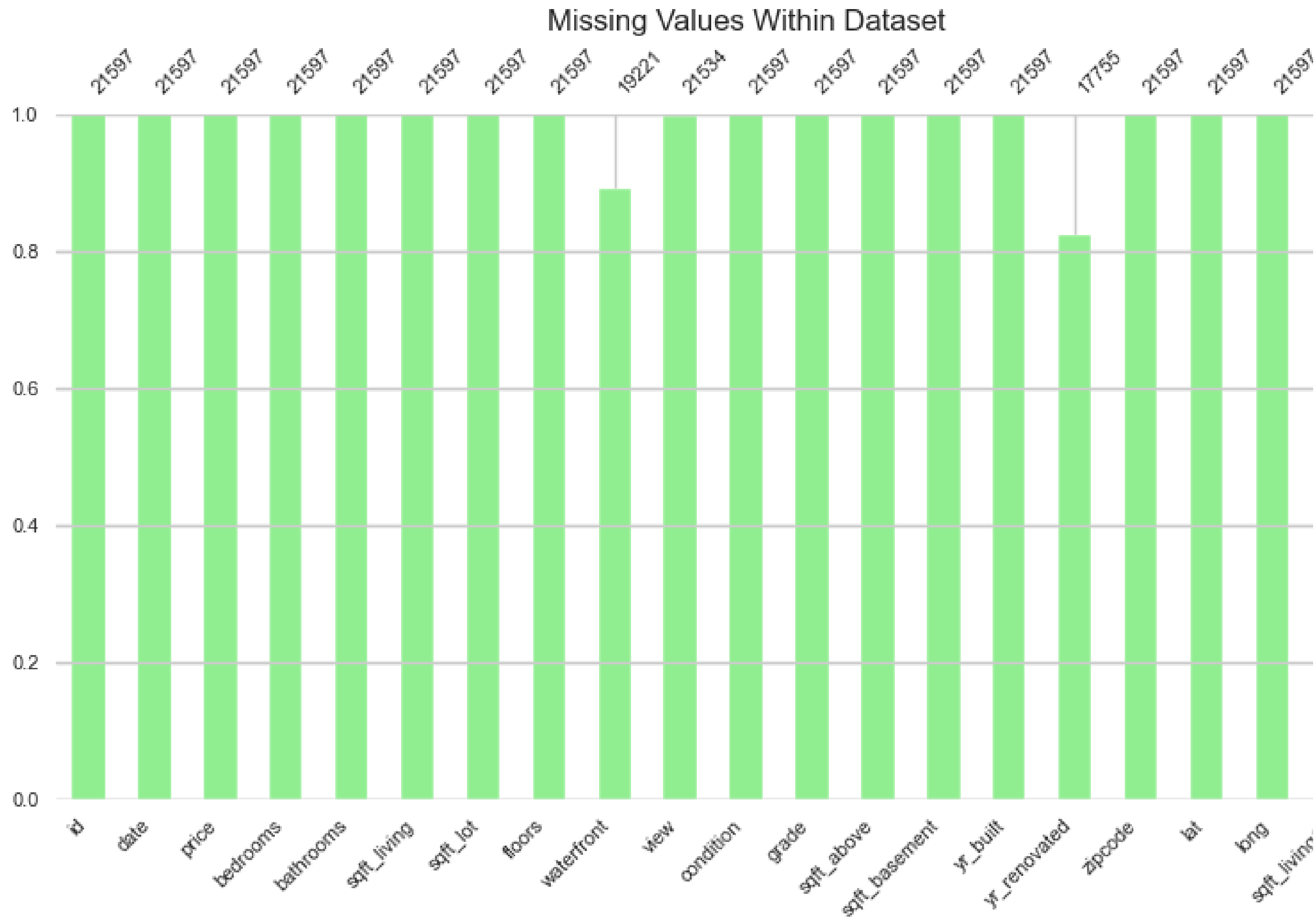
In [276]:

```
# Describe the data
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  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
```

In [277]:

```
# Visualise the missing values in the dataset
msno.bar(data, color='lightgreen', figsize=(10, 5), fontsize=8)
plt.title('Missing Values Within Dataset');
```



From the barplot above, we can see that the columns with missing data are `waterfront` , `vie`
`waterfront` column has 2376 missing values, the `view` column has 63 missing values, and t
3842 missing values. This accounts for 11%, 0.3%, and 18% of the total number of rows in tl

2.1 Univariate Analysis

> In this section, we'll explore each column in the dataset to see the distributions of fea
> 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`

Functions to visualise the data in the categorical columns

In [278]:

```
# Fuction to get the value counts of the data in the columns
def get_value_counts(df, col):
    ''' Returns the value counts of a column in a dataframe '''
    counts = df[col].value_counts(dropna=False)
    return counts

# Function to visualise the the data in the columns
def plot_data(df, col, title):
    ''' Plots the value counts of a column in a dataframe as a bar chart '''
    get_value_counts(df, col).plot(kind='bar', figsize=(10, 5), color='lightgreen', ed
    plt.title(title)
    plt.xticks(rotation=0);
```

2.1.1.1 ID

> The `id` column is a unique identifier for each house sold.

The univariate analysis of the `id` column will be less about identifying the data distribution, b
number of unique values in the column. From the count of the unique values we will be able
any duplicates.

In [279]:

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

We see that there are 177 duplicated ids in the dataset. This could mean that there are some
more than once, or it could also mean that there are some records that have been imputted i
We will have to investigate this further in the data preparation phase.

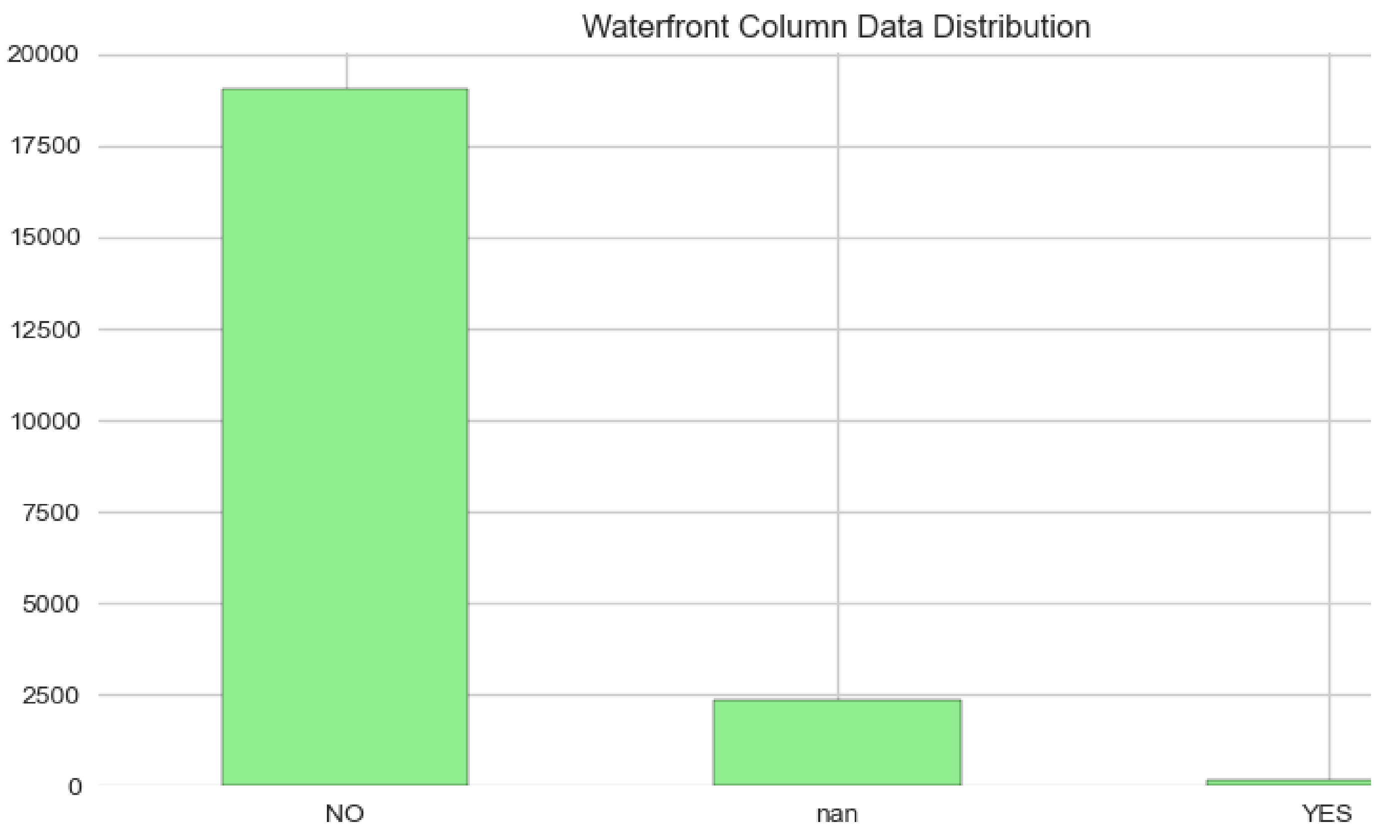
2.1.1.2 Waterfront

> The `waterfront` column identifies whether the house is on a waterfront or not.

In [280]:

```
# Identify the unique values (and counts) in the 'waterfront' column
print(get_value_counts(data, 'waterfront'))

# Visualise the data distribution
plot_data(data, 'waterfront', 'Waterfront Column Data Distribution')
```

```
NO      19075
NaN      2376
YES       146
Name: waterfront, dtype: int64
```



The distribution above shows that most of the houses in the dataset are not on a waterfront.
waterfront is 146, which is 0.7% of the total number of houses in the dataset. The missing va

which is 11% of the total number of rows in the dataset. As this is a categorical column, we w
with the mode of the column.

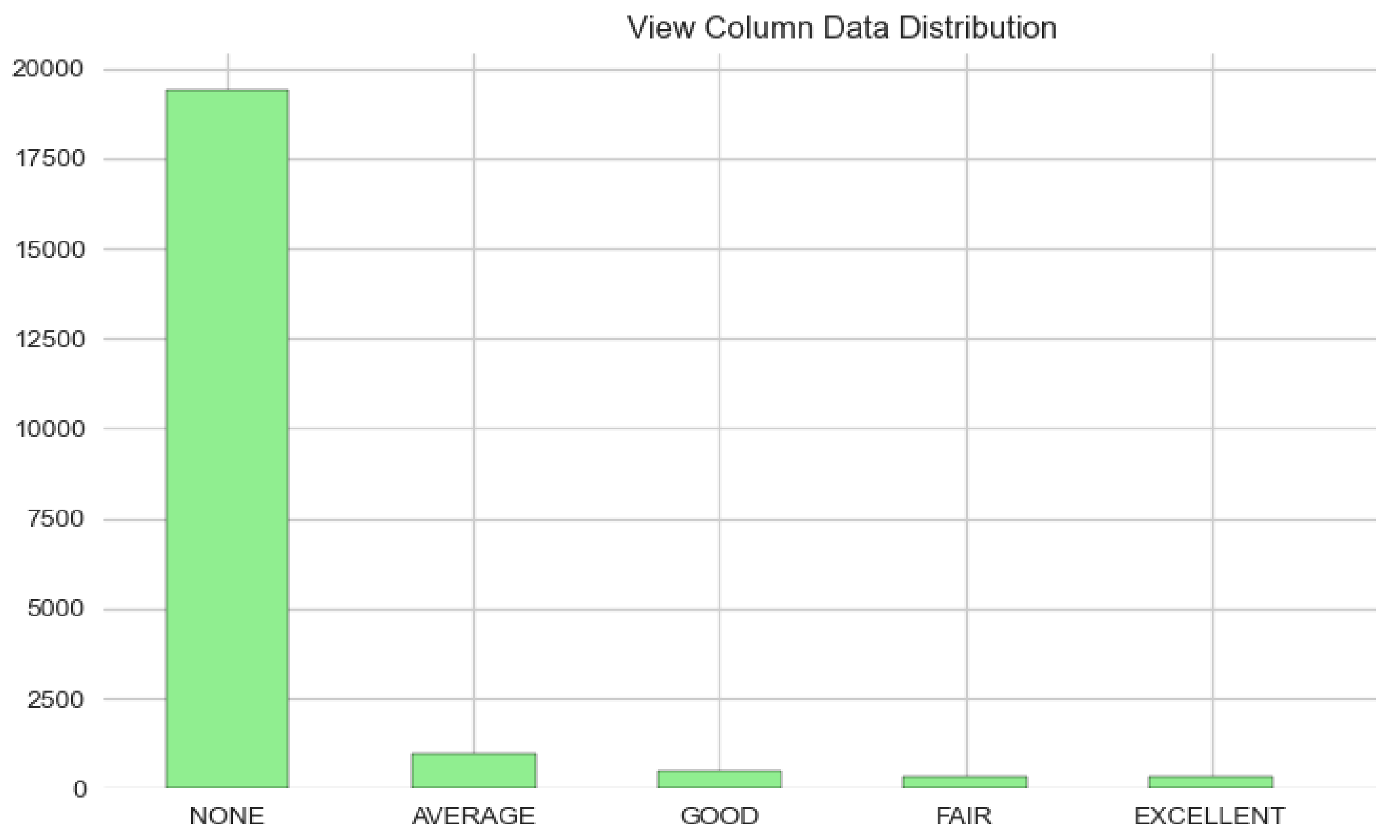
2.1.1.3 View

> The view column identifies the quality of view from the house.

In [281]:

```
# Identify the unique values (and counts) in the 'view' column
print(get_value_counts(data, 'view'))

# Visualise the data distribution
plot_data(data, 'view', 'View Column Data Distribution')
```

```
NONE         19422
AVERAGE        957
GOOD           508
FAIR           330
EXCELLENT      317
NaN             63
Name: view, dtype: int64
```



In the distribution above, we see that majority of the houses in the dataset have a no view. F

in this columns are 63, which is 0.29% of the total number of rows in the dataset. As this is a
number of rows in the dataset, we can drop the rows with missing values in this column.

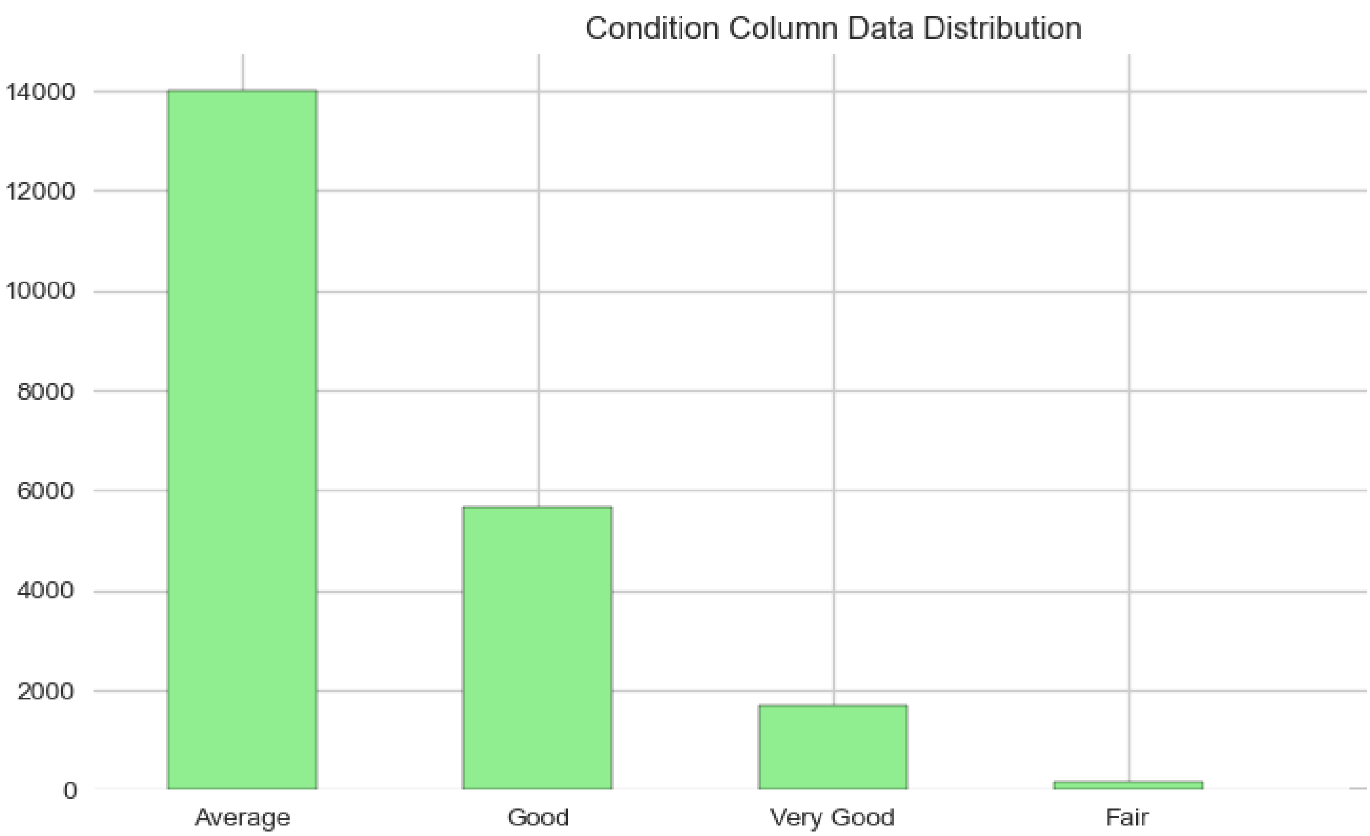
2.1.1.4 Condition

> The condition column identifies the condition of the house.

In [282]:

```
# Identify the unique values (and counts) in the 'condition' column
print(get_value_counts(data, 'condition'))

# Visualise the data distribution
plot_data(data, 'condition', 'Condition Column Data Distribution')
```

```
Average      14020
Good          5677
Very Good     1701
Fair           170
Poor            29
Name: condition, dtype: int64
```



From the distribution above, we can see that most of the houses in the dataset are in averag
houses in average condition is 12437, this accounts for 57.6% of the total number of houses

houses in good condition are 5041, this accounts for 23.3% of the total number of houses in
houses in very good condition are 1509, this accounts for 7% of the total number of houses i
houes in fair condition are 152, this accounts for 0.7% of the total number of houses in the d
poor condition are 25, this accounts for 0.1% of the total number of houses in the dataset. Fu
there is no missing data within this column.

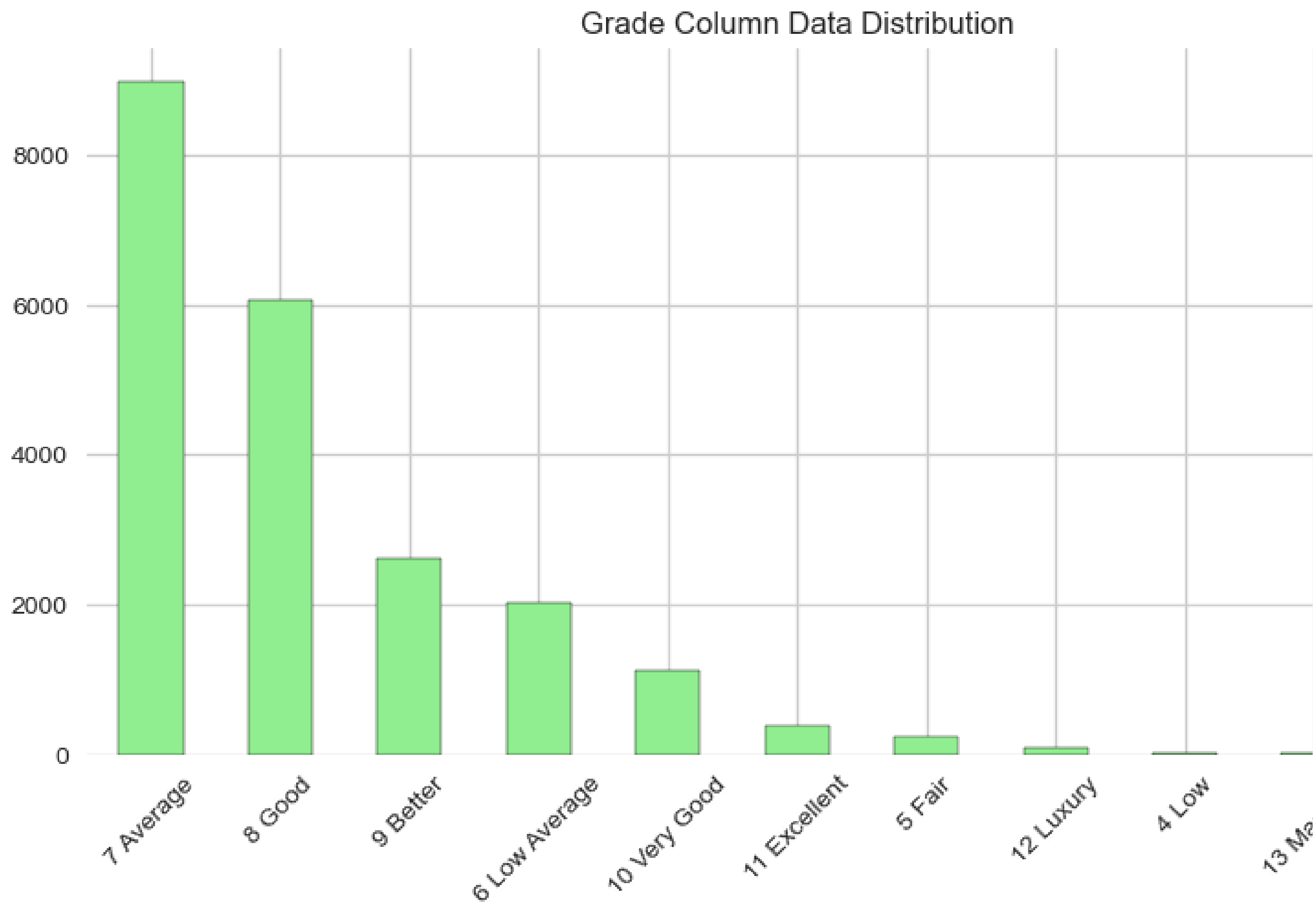
2.1.1.5 Grade

> The grade column identifies the quality of construction and design of the house. The
> construction quality of improvements. Grades run from grade 1 to 13.

In [283]:

```python
# Identify the unique values (and counts) in the 'grade' column
print(get_value_counts(data, 'grade'))

# Visualise the data distribution
plot_data(data, 'grade', 'Grade Column Data Distribution')
plt.xticks(rotation=45);
```

```
7 Average        8974
8 Good           6065
9 Better         2615
6 Low Average    2038
10 Very Good     1134
11 Excellent      399
5 Fair            242
12 Luxury          89
4 Low              27
13 Mansion         13
3 Poor              1
Name: grade, dtype: int64
```



From the distribution above, we see that the houses in this dataset range from grades 3-13.

evenly distributed as we can see majority of the houses with a grade of 7 (representing Aver
Good). Lastly, there are no missing values within this column.

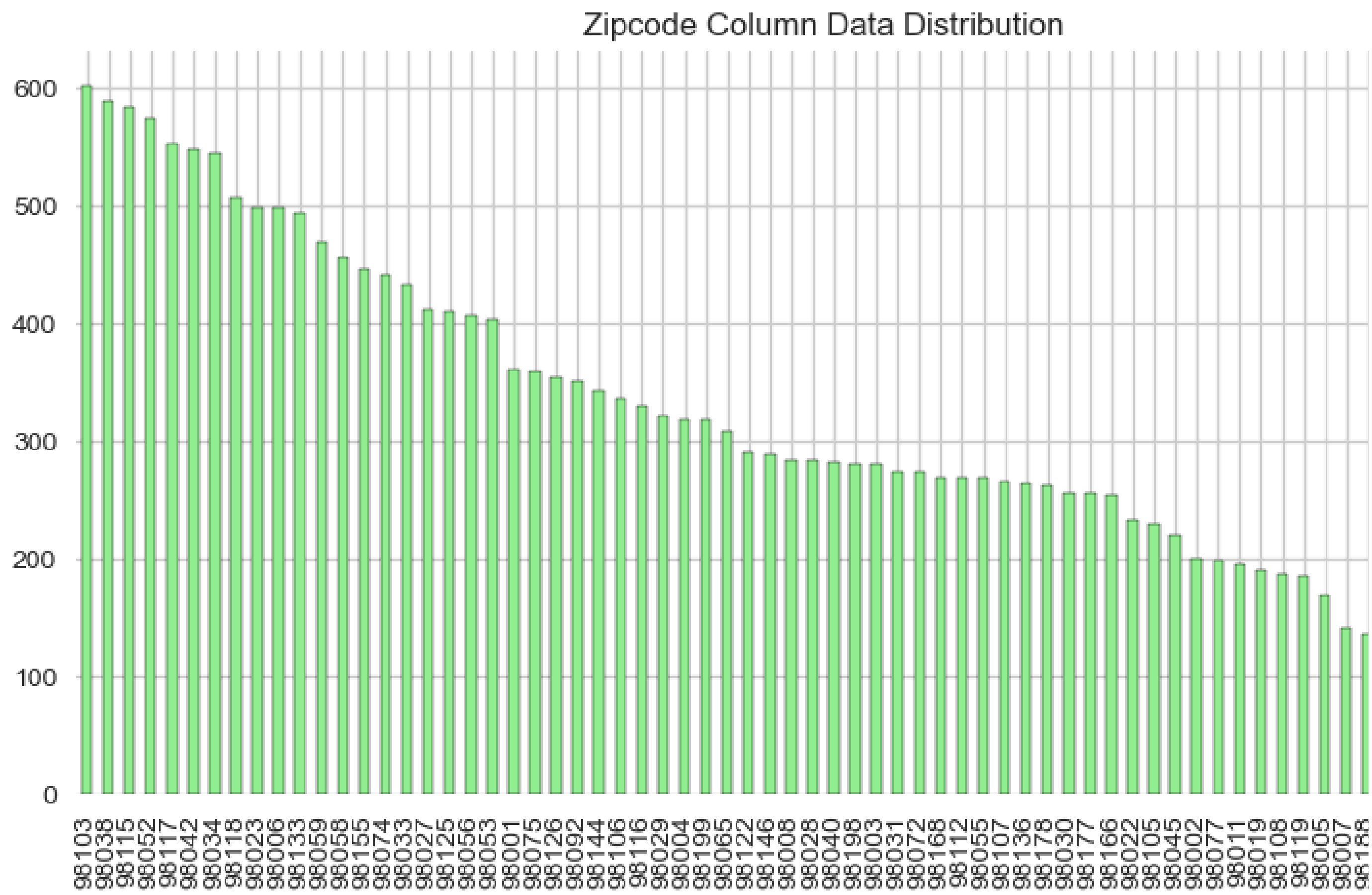
2.1.1.5 Zipcode

> The zipcode column identifies the zipcode area the house is in.

In [284]:

```python
# Identify the unique values (and counts) in the 'zipcode' column
print(get_value_counts(data, 'zipcode'))

# Visualise the data distribution
plot_data(data, 'zipcode', 'Zipcode Column Data Distribution')
plt.xticks(rotation=90);
```

```
98103    602
98038    589
98115    583
98052    574
98117    553
        ...
98102    104
98010    100
98024     80
98148     57
98039     50
Name: zipcode, Length: 70, dtype: int64
```



From the distribution above, we see that the zipcode with the most houses is 98103. The zip
98039. Unlike the other categorical columns, we see more evenly distributed data in this colu

this column

Summary Of The Categorical Columns

- The quality of the data in the categorical columns is fairly good. Other than a few missin
  `view` columns, and duplicated values in the `id` column, the data is good to work with.

2.1.2 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`

Functions to visualise the data in the numerical columns

In [285]:

```
# Fuction that describes the statistics of the data
def describe_data(df, col):
    ''' Returns the statistics of a column in a dataframe '''
    print(df[col].describe())

# Function to plot the histogram, kde and boxplot of the data
def plot_distribution(df, col, title, bins_=10):
    ''' Plots the distribution of a column in a dataframe as a histogram, kde and boxp
    # creating 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_ratios": (

    # 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=bins_,
    plt.suptitle(title)
    plt.tight_layout();
```

2.1.2.1 Date

> The `date` column identifies the date the house was sold.

Date can either be a categorical or numerical column. In this case, we will treat it as a numer

In [286]:

```python
# Print all the unique values in the 'date' column
print(get_value_counts(data, 'date').index.tolist())

# Get the unique values (and counts) in the 'date' column
print(get_value_counts(data, 'date'))
```

```
['6/23/2014', '6/25/2014', '6/26/2014', '7/8/2014', '4/27/2015', '3/25/2015', '7/9/2014', '4/14/2015', '4/28/20
4', '4/21/2015', '8/26/2014', '10/28/2014', '7/14/2014', '5/20/2014', '7/1/2014', '8/20/2014', '6/17/2014', '4,
8/2015', '7/16/2014', '5/28/2014', '4/23/2015', '8/5/2014', '3/27/2015', '8/25/2014', '6/20/2014', '7/23/2014',
'3/26/2015', '6/3/2014', '5/27/2014', '8/22/2014', '9/23/2014', '4/2/2015', '4/24/2015', '7/25/2014', '4/7/201
'8/27/2014', '6/19/2014', '3/24/2015', '6/4/2014', '11/13/2014', '8/12/2014', '3/4/2015', '7/18/2014', '9/24/20
5', '6/10/2014', '7/21/2014', '6/16/2014', '12/2/2014', '8/13/2014', '10/27/2014', '12/1/2014', '2/25/2015', '
9/2014', '7/28/2014', '5/21/2014', '5/5/2015', '7/24/2014', '9/9/2014', '5/7/2014', '8/19/2014', '9/16/2014',
23/2015', '8/14/2014', '10/29/2014', '4/9/2015', '5/22/2014', '10/7/2014', '9/22/2014', '3/30/2015', '8/4/2014
4', '7/2/2014', '7/10/2014', '9/26/2014', '10/21/2014', '6/12/2014', '10/14/2014', '4/13/2015', '5/6/2015', '9,
1/2015', '11/18/2014', '8/21/2014', '11/20/2014', '5/13/2014', '9/10/2014', '9/5/2014', '7/17/2014', '10/1/201
4', '10/15/2014', '8/6/2014', '9/3/2014', '10/30/2014', '8/18/2014', '5/5/2014', '4/6/2015', '10/20/2014', '5/
2015', '9/29/2014', '5/6/2014', '5/19/2014', '4/17/2015', '4/30/2015', '7/31/2014', '11/17/2014', '5/15/2014',
'11/19/2014', '11/21/2014', '5/9/2014', '8/1/2014', '11/10/2014', '5/14/2014', '5/8/2014', '9/2/2014', '5/12/20
14', '12/15/2014', '10/9/2014', '9/4/2014', '11/24/2014', '9/18/2014', '9/25/2014', '6/13/2014', '3/5/2015', '
28/2014', '6/2/2014', '8/8/2014', '4/16/2015', '9/15/2014', '4/10/2015', '6/30/2014', '5/1/2015', '12/11/2014',
'11/5/2014', '5/7/2015', '9/19/2014', '3/12/2015', '12/10/2014', '7/30/2014', '4/3/2015', '2/19/2015', '10/8/20
14', '10/17/2014', '12/8/2014', '9/8/2014', '9/17/2014', '3/10/2015', '7/3/2014', '11/7/2014', '10/2/2014', '3,
3/2014', '2/20/2015', '11/4/2014', '2/23/2015', '11/25/2014', '3/13/2015', '3/19/2015', '10/6/2014', '2/13/201
4', '12/12/2014', '3/9/2015', '2/17/2015', '5/2/2014', '12/18/2014', '10/3/2014', '12/22/2014', '6/6/2014', '1
29/2014', '12/17/2014', '3/20/2015', '10/13/2014', '3/3/2015', '8/7/2014', '12/23/2014', '2/11/2015', '8/15/20
4', '1/28/2015', '2/26/2015', '1/5/2015', '10/24/2014', '3/6/2015', '8/29/2014', '1/27/2015', '2/9/2015', '1/2
015', '2/10/2015', '1/7/2015', '1/16/2015', '2/4/2015', '1/14/2015', '2/6/2015', '2/5/2015', '2/27/2015', '11/
2015', '1/22/2015', '2/12/2015', '1/26/2015', '1/23/2015', '2/2/2015', '11/26/2014', '2/3/2015', '1/20/2015',
'1/15/2015', '1/29/2015', '1/13/2015', '3/2/2015', '12/30/2014', '1/12/2015', '1/6/2015', '12/19/2014', '5/11/
2014', '1/9/2015', '1/30/2015', '12/24/2014', '5/13/2015', '4/25/2015', '3/21/2015', '4/26/2015', '4/12/2015',
'2/22/2015', '6/22/2014', '5/14/2015', '5/24/2014', '6/8/2014', '5/3/2015', '7/12/2014', '1/19/2015', '3/29/20
4', '6/21/2014', '6/14/2014', '3/28/2015', '7/20/2014', '7/26/2014', '8/23/2014', '6/29/2014', '6/15/2014', '5,
1/2014', '4/5/2015', '7/5/2014', '2/16/2015', '3/1/2015', '5/31/2014', '5/2/2015', '4/19/2015', '10/18/2014',
14/2015', '11/1/2014', '9/13/2014', '5/4/2014', '10/25/2014', '5/25/2014', '9/21/2014', '5/10/2014', '9/6/2014
'9/27/2014', '4/18/2015', '2/28/2015', '7/19/2014', '11/8/2014', '3/22/2015', '5/3/2014', '8/17/2014', '12/13/
14', '4/4/2015', '10/19/2014', '6/7/2014', '12/14/2014', '8/16/2014', '9/14/2014', '11/23/2014', '10/26/2014',
'1/25/2015', '9/28/2014', '5/9/2015', '8/10/2014', '11/29/2014', '11/16/2014', '2/14/2015', '10/12/2014', '2/7,
014', '8/31/2014', '10/5/2014', '7/6/2014', '2/21/2015', '7/4/2014', '8/24/2014', '2/1/2015', '10/11/2014', '1
27/2014', '5/11/2014', '12/21/2014', '8/9/2014', '9/7/2014', '11/15/2014', '11/28/2014', '1/10/2015', '5/27/20
4', '2/15/2015', '3/8/2015', '8/30/2014', '5/15/2015', '1/17/2015', '11/2/2014', '1/31/2015', '5/24/2015', '5/
6/23/2014    142
6/25/2014    131
6/26/2014    131
7/8/2014     127
4/27/2015    126
            ...
11/2/2014      1
1/31/2015      1
5/24/2015      1
5/17/2014      1
7/27/2014      1
Name: date, Length: 372, dtype: int64
```

From the output above, we can see that the data has been stored in string format. We will ha
datetime format in the data preparation phase. Futhermore, it seems that most of the houses
2015.

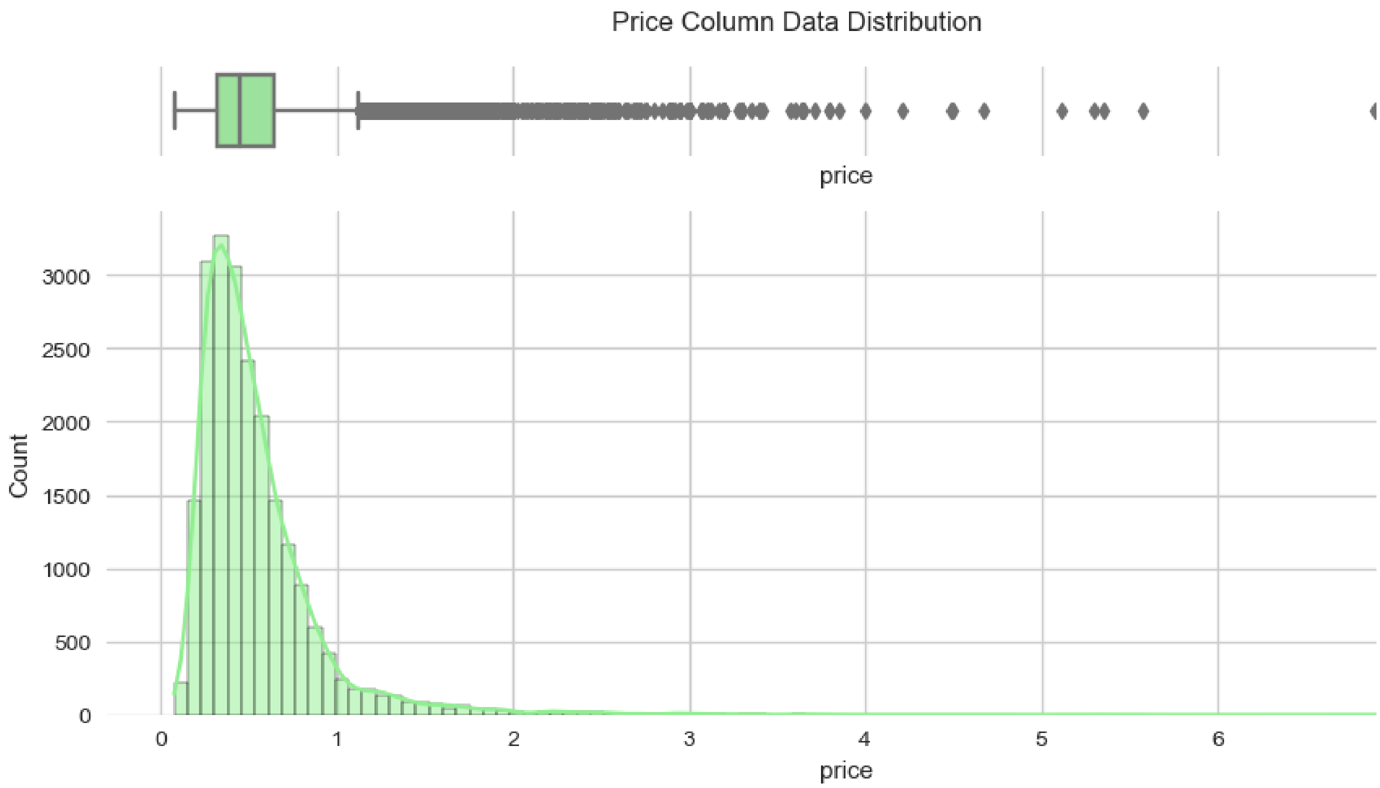
2.1.2.2 Price

> The price column identifies the price of the house.

In [287]:

```
# Describe the 'price' column
describe_data(data, 'price')

# Visualise the data distribution
plot_distribution(data, 'price', 'Price Column Data Distribution', 100)
```

```
count    2.159700e+04
mean     5.402966e+05
std      3.673681e+05
min      7.800000e+04
25%      3.220000e+05
50%      4.500000e+05
75%      6.450000e+05
max      7.700000e+06
Name: price, dtype: float64
```



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

Looking at the kurtosis of the distribution shows that

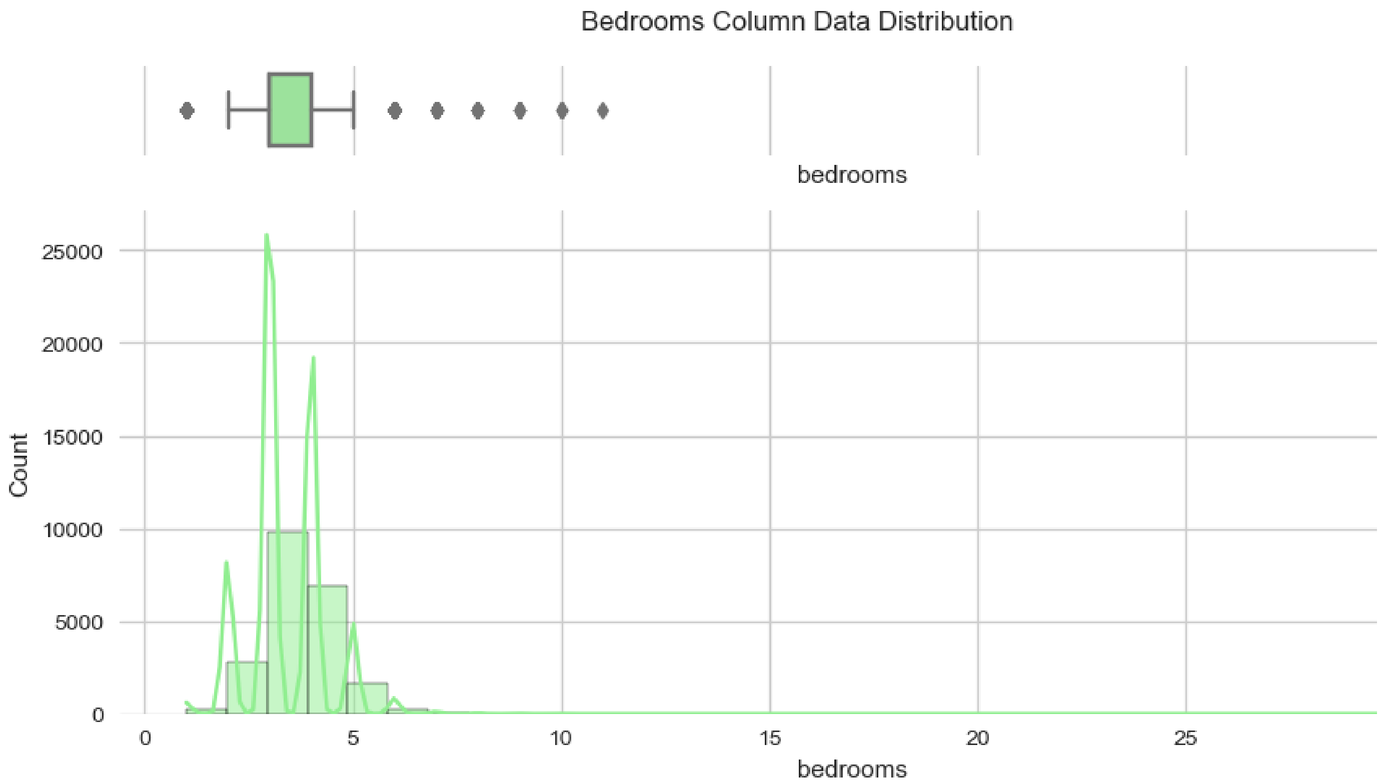
2.1.2.3 Bedrooms

The bedrooms column identifies the number of bedrooms in the house.

In [288]:

```
# Describe the 'bedroom' column
describe_data(data, 'bedrooms')

# Visualise the data distribution
plot_distribution(data, 'bedrooms', 'Bedrooms Column Data Distribution',33)
```

```
count    21597.000000
mean         3.373200
std          0.926299
min          1.000000
25%          3.000000
50%          3.000000
75%          4.000000
max         33.000000
Name: bedrooms, dtype: float64
```



The bedroom column distribution is not skewed as the and is normally distributed. The minin
house in the dataset is 1, and the maximum number of bedrooms in a house in the dataset is
bedrooms in a house in the dataset is 3.37, and the median number of bedrooms in a house
standard deviation of the bedrooms column is 0.93.

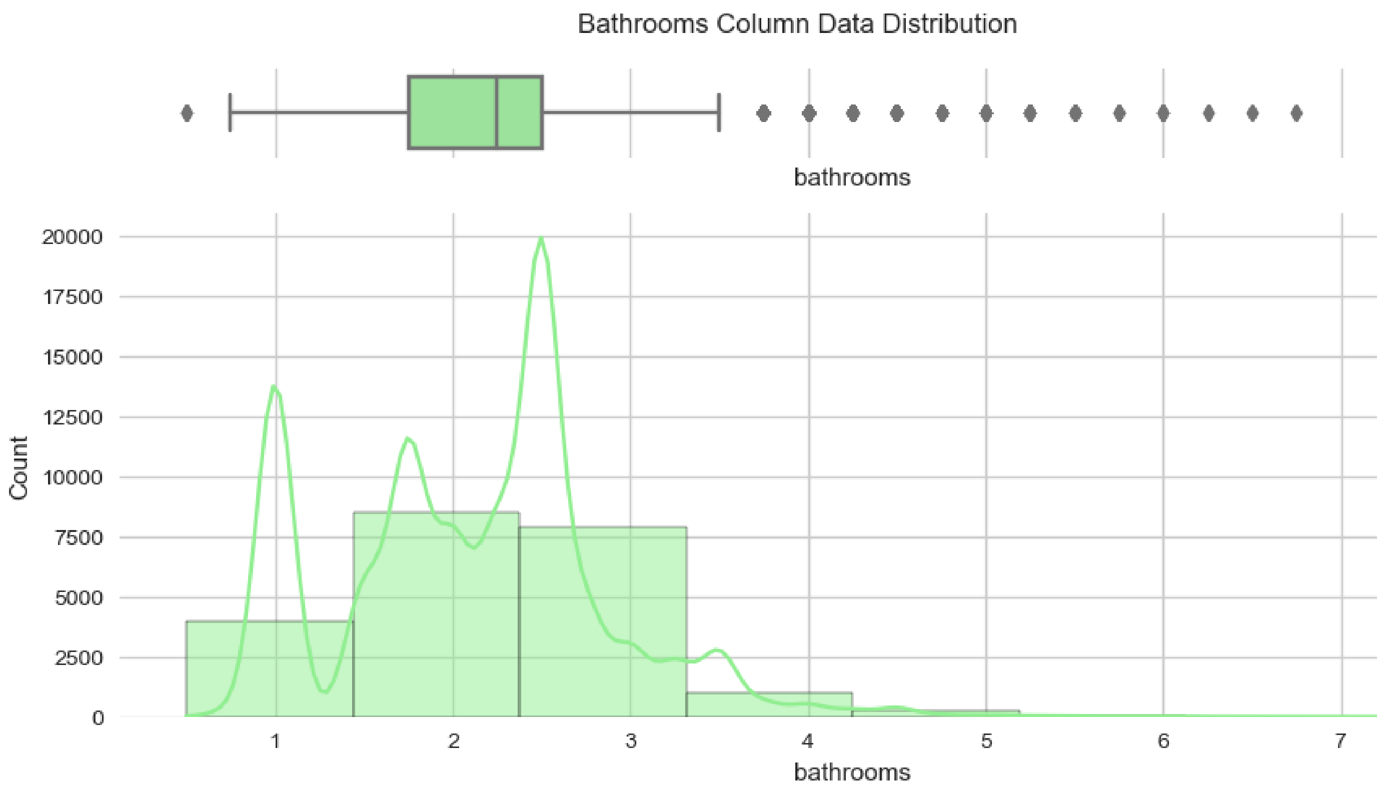
2.1.2.4 Bathrooms

> The bathrooms column identifies the number of bathrooms in the house.

In [289]:

```
# Describe the 'bathrooms' column
describe_data(data, 'bathrooms')

# Visualise the data distribution
plot_distribution(data, 'bathrooms', 'Bathrooms Column Data Distribution', 8)
```

```
count    21597.000000
mean         2.115826
std          0.768984
min          0.500000
25%          1.750000
50%          2.250000
75%          2.500000
max          8.000000
Name: bathrooms, dtype: float64
```



From the distribution above we can see that the bathroom column is not skewed. This is bec almost the same. The minimum number of bathrooms in a house in the dataset is 0.5, and th bathrooms in a house in the dataset is 8. The mean number of bathrooms in a house in the c number of bathrooms in a house in the dataset is 2.25. The standard deviation of the bathroo

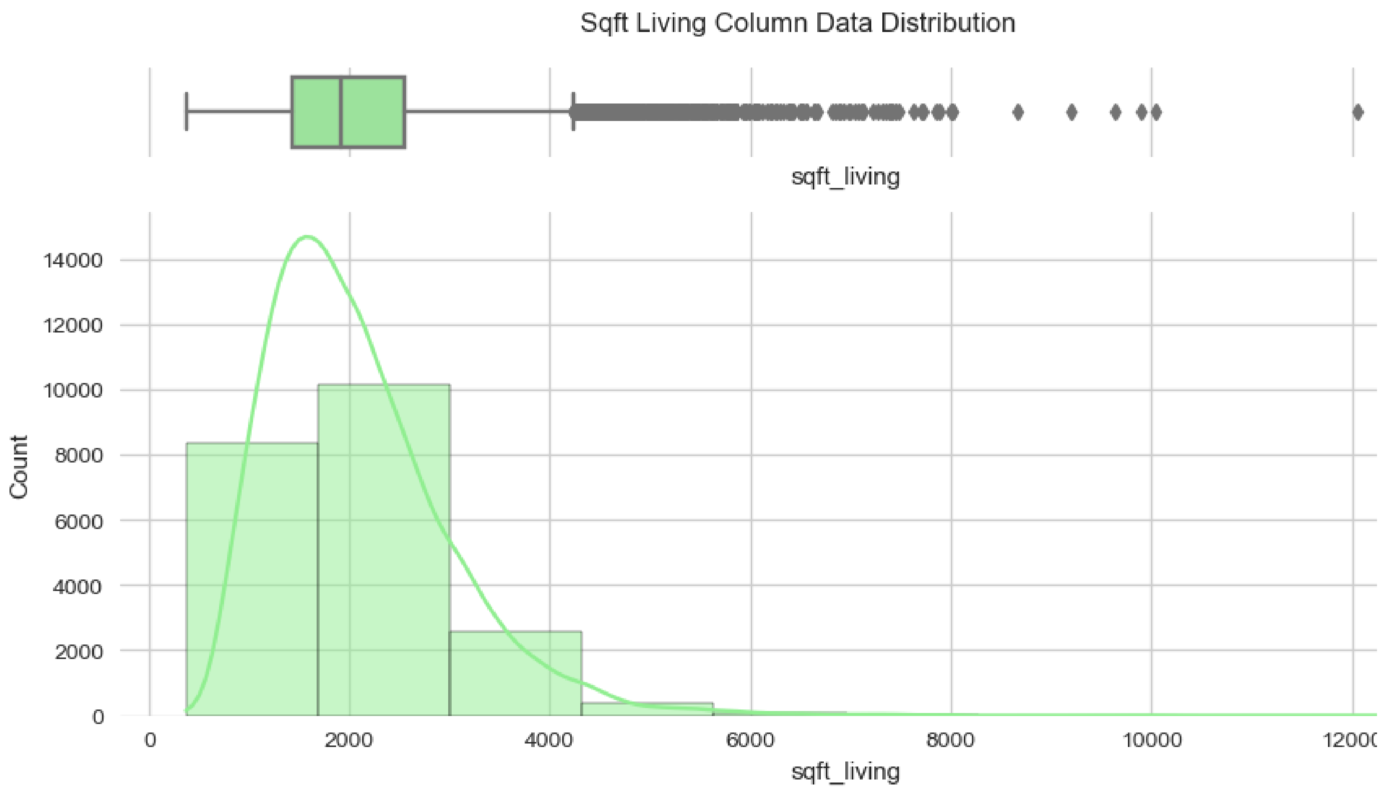
2.1.2.5 Sqft Living

> The sqft living column identifies the square footage of the house.

In [290]:

```
# Describe the 'sqft_living' column
describe_data(data, 'sqft_living')

# Visualise the data distribution
plot_distribution(data, 'sqft_living', 'Sqft Living Column Data Distribution')
```

```
count    21597.000000
mean      2080.321850
std        918.106125
min        370.000000
25%       1430.000000
50%       1910.000000
75%       2550.000000
max      13540.000000
Name: sqft_living, dtype: float64
```



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

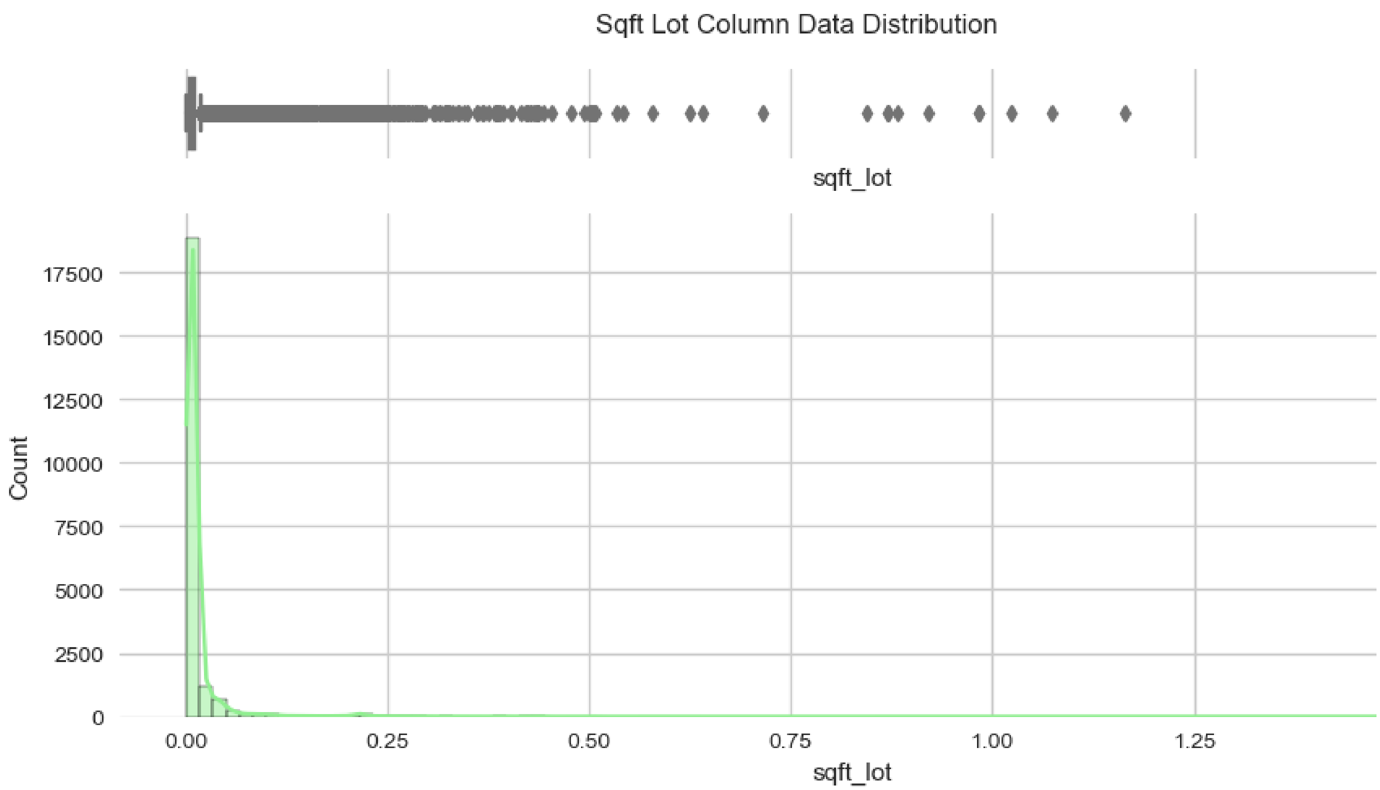
2.1.2.6 Sqft Lot

> The `sqft lot` column identifies the square footage of the lot.

In [291]:

```
# Describe the 'sqft_lot' column
describe_data(data, 'sqft_lot')

# Visualise the data distribution
plot_distribution(data, 'sqft_lot', 'Sqft Lot Column Data Distribution', 100)
```

```
count    2.159700e+04
mean     1.509941e+04
std      4.141264e+04
min      5.200000e+02
25%      5.040000e+03
50%      7.618000e+03
75%      1.068500e+04
max      1.651359e+06
Name: sqft_lot, dtype: float64
```



From the distribution above, we can see that the data is skewed to the right. This is because median. The minimum lot square footage is 520, the maximium lot square footage is 1,651,3

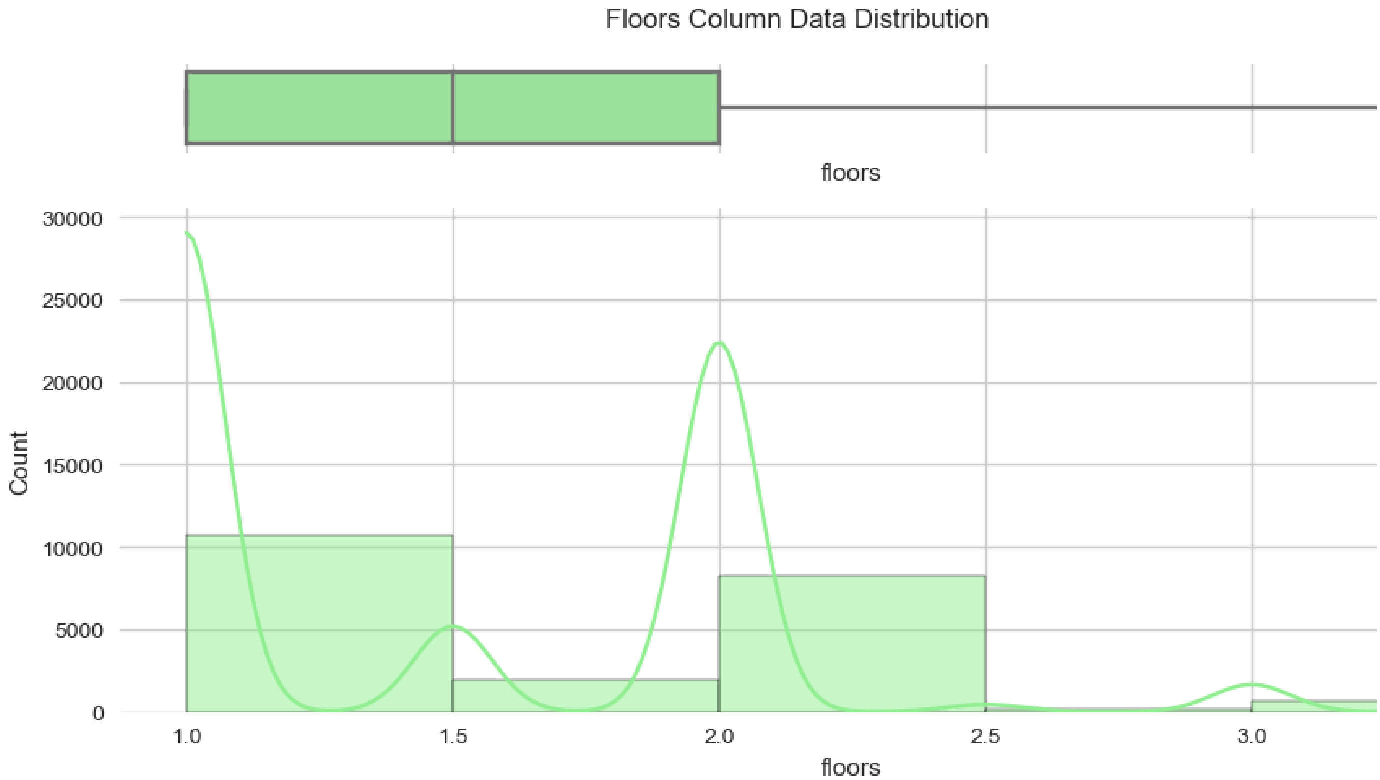
2.1.2.7 Floors

> `floors` column identifies the number of floors in the house.

In [292]:

```
# Describe the 'floors' column
describe_data(data, 'floors')

# Visualise the data distribution
plot_distribution(data, 'floors', 'Floors Column Data Distribution', 5)
```

```
count    21597.000000
mean         1.494096
std          0.539683
min          1.000000
25%          1.000000
50%          1.500000
75%          2.000000
max          3.500000
Name: floors, dtype: float64
```



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

1 floors. The minimum number of floors in a house is 1, and the maximum number of floors i
number of floors in this dataset is 1.5, and the mean number of floors in this dataset is appro

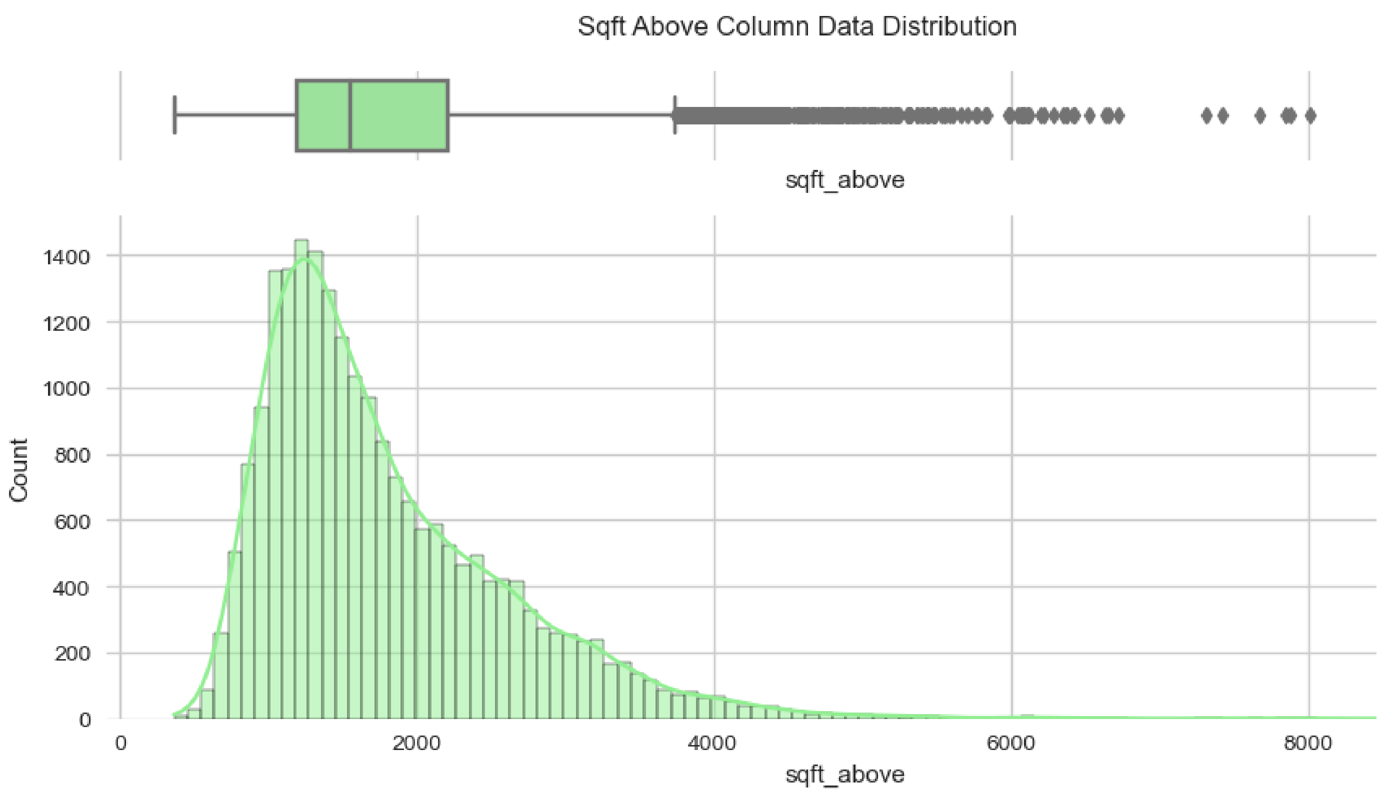
2.1.2.8 Sqft Above

> The `sqft above` column identifies the square footage of the house above the ground.

In [293]:

```
# Describe the 'sqft_above' column
describe_data(data, 'sqft_above')

# Visualise the data distribution
plot_distribution(data, 'sqft_above', 'Sqft Above Column Data Distribution',100)
```

```
count    21597.000000
mean      1788.596842
std        827.759761
min        370.000000
25%       1190.000000
50%       1560.000000
75%       2210.000000
max       9410.000000
Name: sqft_above, dtype: float64
```



From the distributions above, we see that the square footage above ground of the houses in
right. This is because the mean is greater than the median. The minimum square footage ab
maximum square footage of a house above ground is 9,410. The mean square footage abov
median square footage above ground is 1,560. The standard deviation of the sqft above colu

2.1.2.9 Sqft Basement

> The `sqft basement` column identifies the square footage of the basement of the hous

As this column is of the type `object` , we cannot do a distribution like the other numerical colu
viewing the contents of the column using the same technique as the categorical columns.

In [294]:

```
# Print all the unique values in the 'sqft_basement' column
print(get_value_counts(data, 'sqft_basement').index.tolist())

# Get the unique values (and counts) in the 'sqft_basement' column
print(get_value_counts(data, 'sqft_basement'))
```

```
['0.0', '?', '600.0', '500.0', '700.0', '800.0', '400.0', '1000.0', '900.0', '300.0', '200.0', '750.0', '450.0
0', '620.0', '580.0', '840.0', '420.0', '860.0', '1100.0', '670.0', '780.0', '550.0', '650.0', '240.0', '380.0
0', '770.0', '940.0', '910.0', '440.0', '880.0', '290.0', '1200.0', '350.0', '520.0', '920.0', '630.0', '730.0
0', '1010.0', '760.0', '640.0', '280.0', '340.0', '950.0', '820.0', '570.0', '560.0', '460.0', '790.0', '1060.
0', '540.0', '810.0', '1040.0', '250.0', '140.0', '120.0', '890.0', '990.0', '1020.0', '470.0', '1070.0', '125
0.0', '330.0', '390.0', '690.0', '610.0', '1030.0', '270.0', '150.0', '970.0', '1120.0', '220.0', '100.0', '26
0.0', '1050.0', '1300.0', '320.0', '710.0', '1400.0', '180.0', '1110.0', '190.0', '1080.0', '1090.0', '1220.0',
0', '1170.0', '1500.0', '160.0', '1140.0', '170.0', '490.0', '1180.0', '1150.0', '210.0', '1160.0', '130.0', '
'1280.0', '1320.0', '90.0', '1260.0', '1380.0', '1240.0', '1330.0', '80.0', '1360.0', '1340.0', '1290.0', '142
50.0', '1390.0', '1600.0', '1350.0', '1460.0', '1310.0', '1590.0', '1430.0', '1580.0', '1440.0', '1510.0', '15
00.0', '230.0', '60.0', '1480.0', '1490.0', '1650.0', '1780.0', '1690.0', '1760.0', '1570.0', '1720.0', '1520.
0.0', '1620.0', '1870.0', '1530.0', '1790.0', '1680.0', '70.0', '1850.0', '1940.0', '1550.0', '1470.0', '1710.
0.0', '2020.0', '1640.0', '1830.0', '1900.0', '1630.0', '1950.0', '40.0', '1610.0', '1860.0', '2160.0', '1750.
0.0', '2170.0', '2070.0', '2150.0', '265.0', '414.0', '1810.0', '2330.0', '1840.0', '2000.0', '2010.0', '2040.
0', '515.0', '2100.0', '2030.0', '2080.0', '1820.0', '2550.0', '435.0', '1890.0', '235.0', '2090.0', '2110.0',
0', '2190.0', '2610.0', '1008.0', '946.0', '666.0', '1245.0', '1525.0', '1880.0', '862.0', '2300.0', '768.0',
0', '274.0', '20.0', '2810.0', '508.0', '143.0', '417.0', '556.0', '915.0', '207.0', '295.0', '2120.0', '2310.
0', '266.0', '1275.0', '225.0', '176.0', '516.0', '602.0', '1248.0', '276.0', '2180.0', '1990.0', '1548.0', '2
'506.0', '588.0', '2850.0', '1284.0', '875.0', '2570.0', '2500.0', '3000.0', '2490.0', '4130.0', '1481.0', '11
'1852.0', '2360.0', '2600.0', '243.0', '704.0', '784.0', '2390.0', '374.0', '518.0', '935.0', '792.0', '475.0'
0', '1930.0', '2196.0', '652.0', '415.0', '3260.0', '1913.0', '4820.0', '2050.0', '1960.0', '1920.0', '3480.0'
0']
0.0       12826
?           454
600.0       217
500.0       209
700.0       208
          ...
1920.0        1
3480.0        1
2730.0        1
2720.0        1
248.0         1
Name: sqft_basement, Length: 304, dtype: int64
```

From the output above, we can see that this is numeric data that has been converted to a str
done because of the presence of the '?' character. In order to make this data usable, we sha
a float. Furthermore, we will need to deal with the missing values ('?') in this column. The mis
records in this dataset. This comes up to 2.1% of the total records in the dataset. As this is a
total records, we shall be dropping the records missing values.

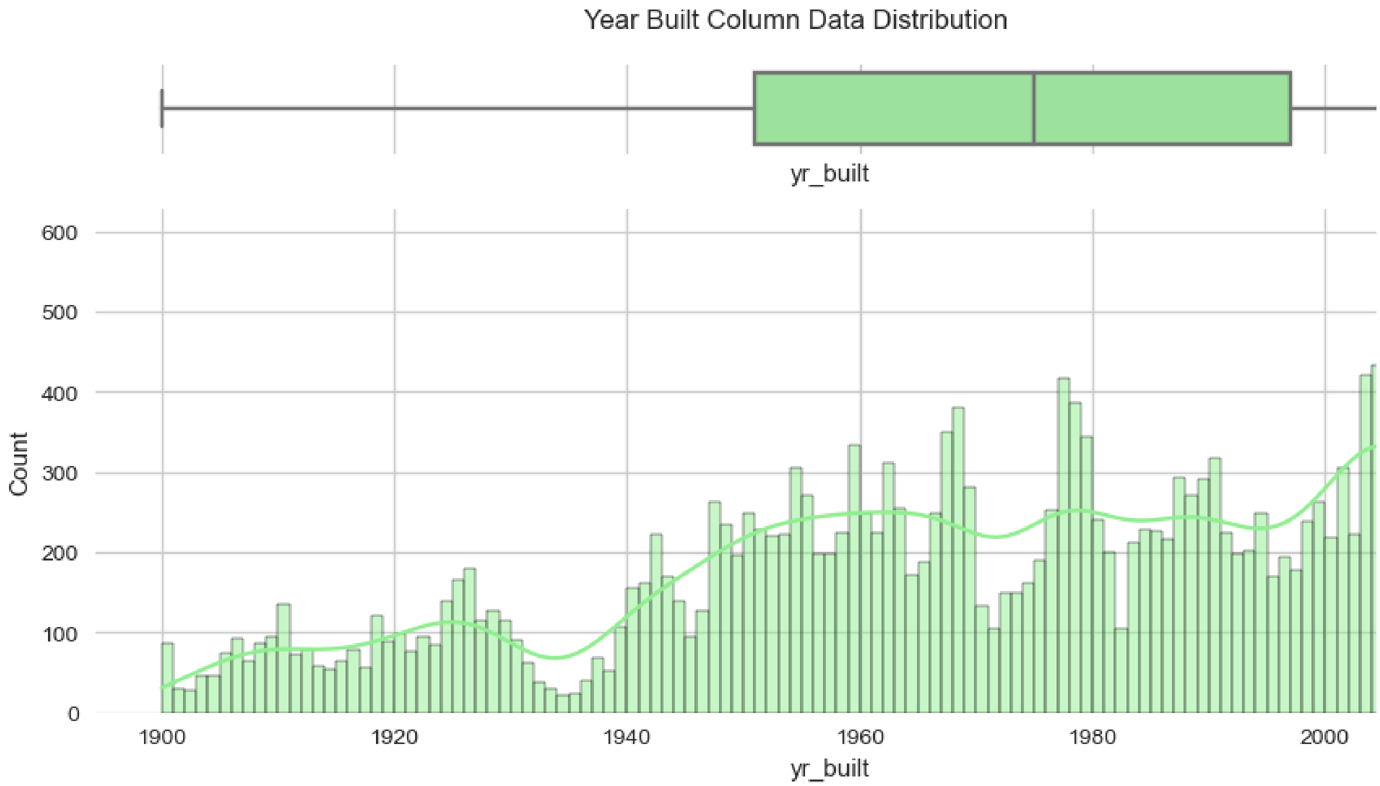
2.1.2.10 Yr Built

> The `yr_built` column identifies the year the house was built.

In [295]:

```python
# Describe the 'yr_built' column
describe_data(data, 'yr_built')

# Visualise the data distribution
plot_distribution(data, 'yr_built', 'Year Built Column Data Distribution', 115)
```

```
count    21597.000000
mean      1970.999676
std         29.375234
min       1900.000000
25%       1951.000000
50%       1975.000000
75%       1997.000000
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 be
than the median. The oldest house in the dataset was built in 1900, and the newest house in
The mean year the houses in the dataset were built is 1971, and the median year the houses
1975. The standard deviation of the yr built column is 29.

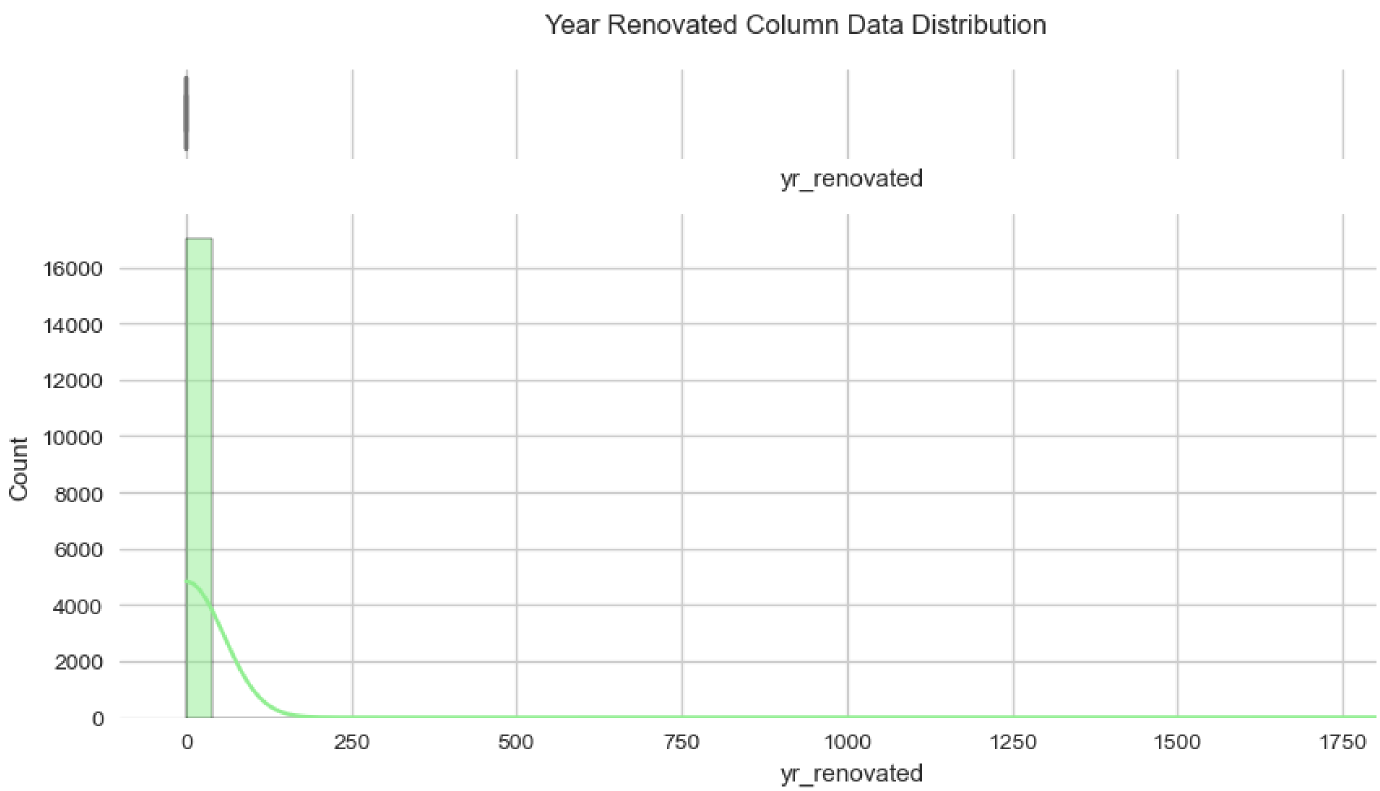
2.1.2.11 Yr Renovated

The `yr renovated` column identifies the year the house was renovated.

In [296]:

```python
# Describe the 'yr_renovated' column
describe_data(data, 'yr_renovated')

# Visualise the data distribution
plot_distribution(data, 'yr_renovated', 'Year Renovated Column Data Distribution', 50)
```

```
count    17755.000000
mean        83.636778
std        399.946414
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 zer
suggesting that the house has not been renovated, or that the data is missing. Furthermore,
in this column. We shall be analysing the data more indepth in the next phase to see how to
missing values in the column.

2.1.2.12 Lat & Long

> The `lat` column identifies the latitude of the house. The `long` column identifies the l

In [297]:

```
latlon = list(zip(data.lat, data.long))

base_map = folium.Map([data.lat.mean(), data.long.mean()], zoom_start=13)
base_map

for coord in latlon:
    folium.Marker( location=[ coord[0], coord[1]], fill_color='#43d9de', radius=8 ).ad

# export the map as HTML file
base_map.save('../images/map.html')
```

From the exported map above, we can see that the houses in the dataset are located in the importantly, we see that the houses are roughly within the same area therefore we do not ne data in the lat and long columns.

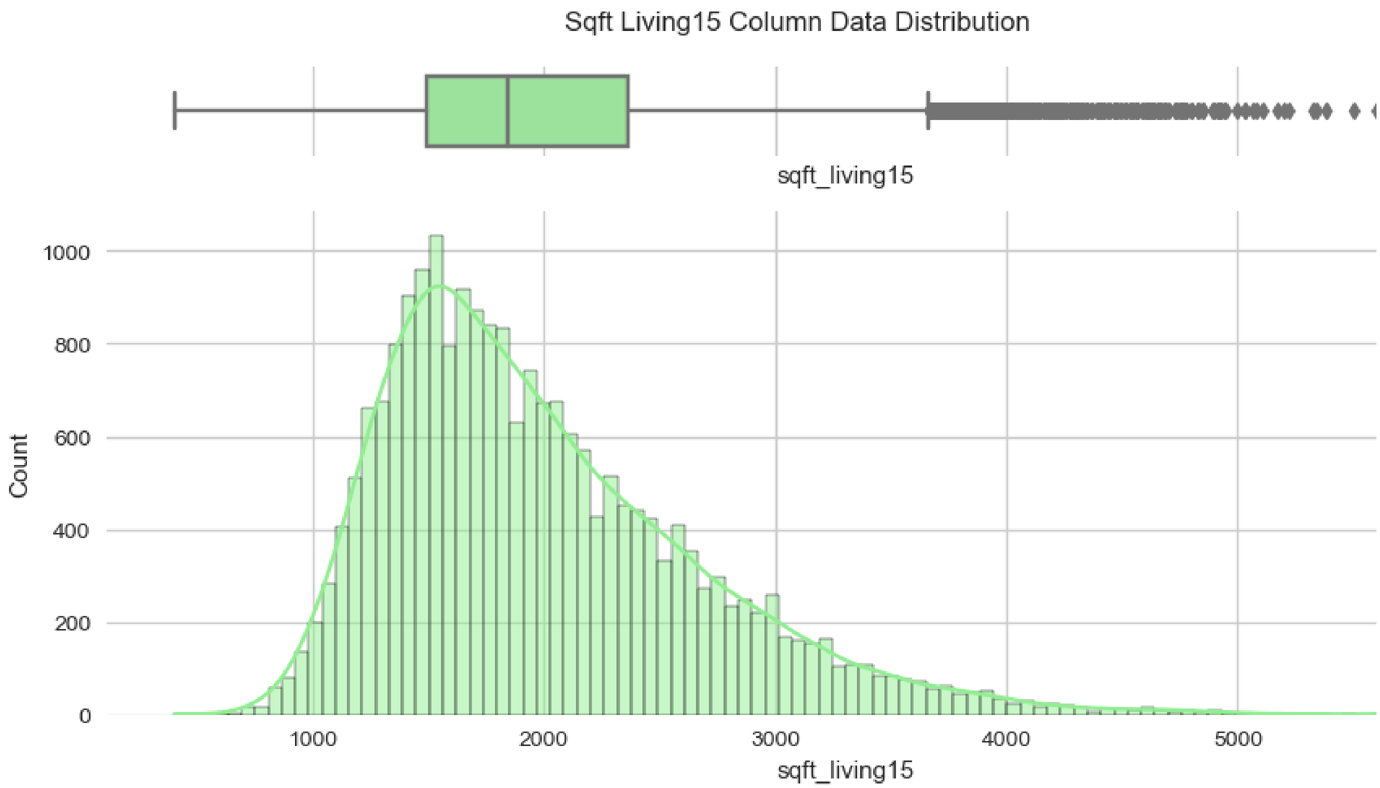
2.1.2.13 Sqft Living15

> The `sqft living15` square footage of interior housing living space for the nearest 15

In [298]:

```python
# Describe the 'sqft_living15' column
describe_data(data, 'sqft_living15')

# Visualise the data distribution
plot_distribution(data, 'sqft_living15', 'Sqft Living15 Column Data Distribution', 100)
```

```
count    21597.000000
mean      1986.620318
std        685.230472
min        399.000000
25%       1490.000000
50%       1840.000000
75%       2360.000000
max       6210.000000
Name: sqft_living15, dtype: float64
```



From the distributions above, we can see that the data is skewed to the right. This is as a res
than the median. The minimum square footage of the nearest 15 neighbors is 399, and the n
nearest 15 neighbors is 6,210. The mean square footage of the nearest 15 neighbors is 198
footage of the nearest 15 neighbors is 1840. The standard deviation of the sqft living15 colur

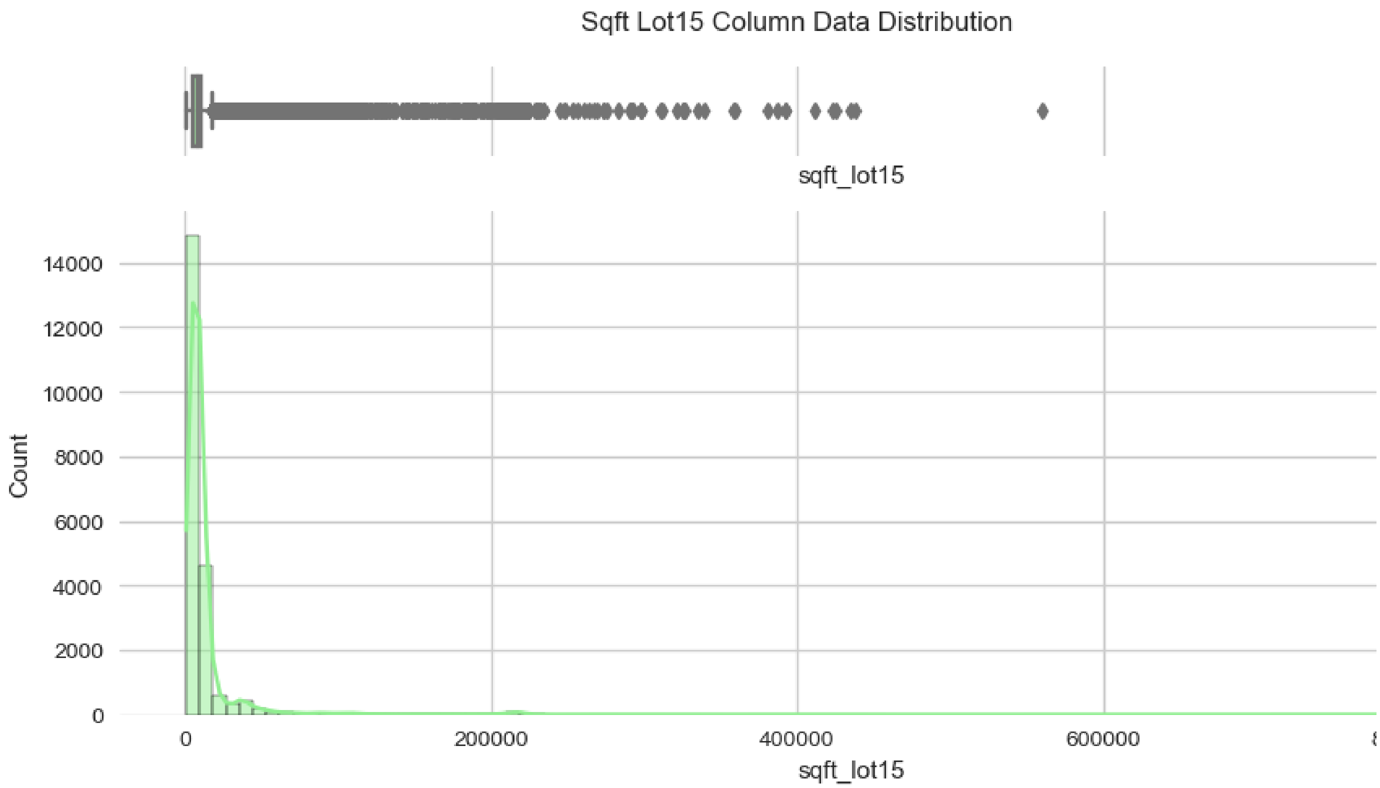
2.1.2.14 Sqft Lot15

> The `sqft lot15` column represents the square footage of the land lots for the neares

In [299]:

```
# Describe the 'sqft_lot15' column
describe_data(data, 'sqft_lot15')

# Visualise the data distribution
plot_distribution(data, 'sqft_lot15', 'Sqft Lot15 Column Data Distribution', 100)
```

```
count     21597.000000
mean      12758.283512
std       27274.441950
min         651.000000
25%        5100.000000
50%        7620.000000
75%       10083.000000
max      871200.000000
Name: sqft_lot15, dtype: float64
```



In the distributions above we see a much more skewed to the right column. The minimum sq neighbors is 651, and the maximum square footage of the nearest 15 neighbors is 871,200. nearest 15 neighbors is 12758, and the median square footage of the nearest 15 neighbors i of the sqft lot15 column is 27274.

Summary Of Numerical Columns

- The data in the numerical columns is also of fairly decent quality. Other than a few missi column and datatype corrections that need to be made to the date and sqft basement c

There are quite a number of outliers in the data, however, I do not think that will affect th

3. Data Processing

> This phase, which is often referred to as "data munging", prepares the final data set(s
> tasks:
>
> - Select Data
> - Clean Data
> - Construct Data
> - Integrate Data
> - Format Data

3.1 Clean Data

> In this section we will be looking at the missing values in the dataset as well as the du
> dataset.

The columns that were identified to be having missing data and duplicates were:

- `id`
- `waterfront`
- `yr renovated`
- `view`

3.1.1 Duplicate Records

The `id` column was identified to have duplicate records. However, we did not know if the du duplicates or if they were different records with the same id. In order to find out, we shall be records in the id column.

In [300]:

```
# Create a new dataframe that contains the ids that have been duplicated in the dataset
duplicates = data[data.duplicated(['id'], keep=False)]

# Preview the duplicates dataframe
duplicates
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfron
93	6021501535	7/25/2014	430000.0	3	1.50	1580	5000	1.0	NO
94	6021501535	12/23/2014	700000.0	3	1.50	1580	5000	1.0	NO
313	4139480200	6/18/2014	1380000.0	4	3.25	4290	12103	1.0	NO
314	4139480200	12/9/2014	1400000.0	4	3.25	4290	12103	1.0	NO
324	7520000520	9/5/2014	232000.0	2	1.00	1240	12092	1.0	NaN
...	...	...	...	...	...	...	...	...	...
20654	8564860270	3/30/2015	502000.0	4	2.50	2680	5539	2.0	NaN
20763	6300000226	6/26/2014	240000.0	4	1.00	1200	2171	1.5	NO
20764	6300000226	5/4/2015	380000.0	4	1.00	1200	2171	1.5	NO
21564	7853420110	10/3/2014	594866.0	3	3.00	2780	6000	2.0	NO
21565	7853420110	5/4/2015	625000.0	3	3.00	2780	6000	2.0	NO

353 rows × 21 columns

Looking at the duplicated id records, we can see that the records are not erroneous. The ids same house was sold multiple times. Therefore, we shall be keeping the records. In order to duplicate records, we shall be checking the `date` column along with the `id` to see if the ther multiple times on the same day. That would be an erroneous record.

In [301]:

```
# Check for duplicate records that have both the same id and date
duplicates[duplicates.duplicated(['id', 'date'], keep=False)]
```

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	s

0 rows × 21 columns

We see that we have no records indicating that the same house has been sold multiple time: we shall be keeping the records with duplicated ids.

3.1.2 Missing Values

The columns that were identified to be having missing data were `waterfront` , `yr renovated` ; with the missing values in these columns. Furthermore, using the insights that were identifie shall be using the type of data along with the data distribution to determine the best way to d

3.1.2.1 Waterfront

The `waterfront` column is a categorical column. The column has 2 unique values, 'YES' and this accounted for 11% of the total records in the dataset. As this is a fairly large percentage replacing the missing values with the mode of the column. The mode of the column is 'NO'. T the missing values with 'NO'.

In [302]:

```
# Fill the missing values with the mode of the column
data['waterfront'] = data['waterfront'].fillna(data['waterfront'].mode()[0])
```

3.1.2.2 Year Renovated

The `yr renovated` column is a numerical column. With 3842 missing values, this accounted the dataset. Futhermore, majority of the data in the records were zero. This could either be s never been renovated or that the data is erroneous. As there is no ideal way of daling with th drop the entire column.

In [303]:

```
# Drop the 'yr_renovated' column
data.drop('yr_renovated', axis=1, inplace=True)

# Preview the first five rows of the dataframe
data.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	vi
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NO	NC
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	NO	NC
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	NO	NC
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	NO	NC
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	NO	NC

3.1.2.3 View

The `view` column is a categorical column. With 63 missing values, this accounted for 0.3% c
As this is a small percentage of the total records, we shall be dropping the records with miss

In [304]:

```
# Drop the missing records in the 'view' column
data = data[data.view.notnull()]

get_value_counts(data, 'view')
```

```
NONE         19422
AVERAGE        957
GOOD           508
FAIR           330
EXCELLENT      317
Name: view, dtype: int64
```

3.2 Construct Data

> In this section, we shall be deriving new attributes that will be helpful in our analysis.

We shall be creating new columns that will be useful in the analysis. The columns that we sh
and `price` columns which represents the date that the homes were sold and their price resp
from the date column and creating a new column called `yr_sold` . The `yr_sold` column will b
to adjust the price of the homes for inflation, if so, the column will once again be used to calc

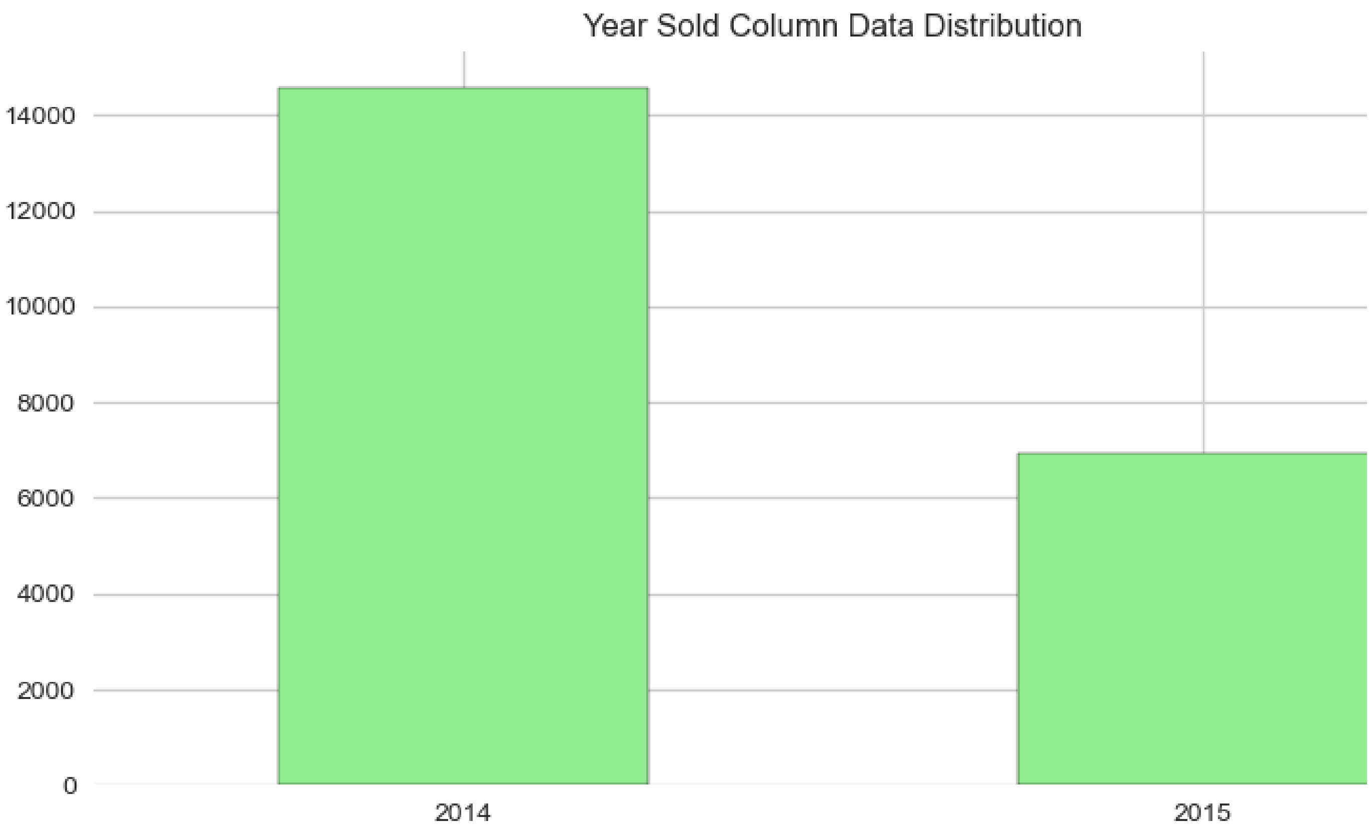
3.2.1 Year Sold

> The `yr sold` column represents the year that the homes were sold.

In [305]:

```
# Create a new column, 'yr_sold', from the 'date' column
data['yr_sold'] = data['date'].apply(lambda x: int(x.split('/')[-1]))

# View the values (and counts) in the 'yr_sold' column
print(get_value_counts(data, 'yr_sold'))

# Visualise the data distribution
plot_data(data, 'yr_sold', 'Year Sold Column Data Distribution')
```

```
2014    14588
2015     6946
Name: yr_sold, dtype: int64
```



We see that the `yr sold` column has been created and populated with the year that the hom
the homes in the dataset were sold in 2014. Ultimately, the data is fairly clean and good to w

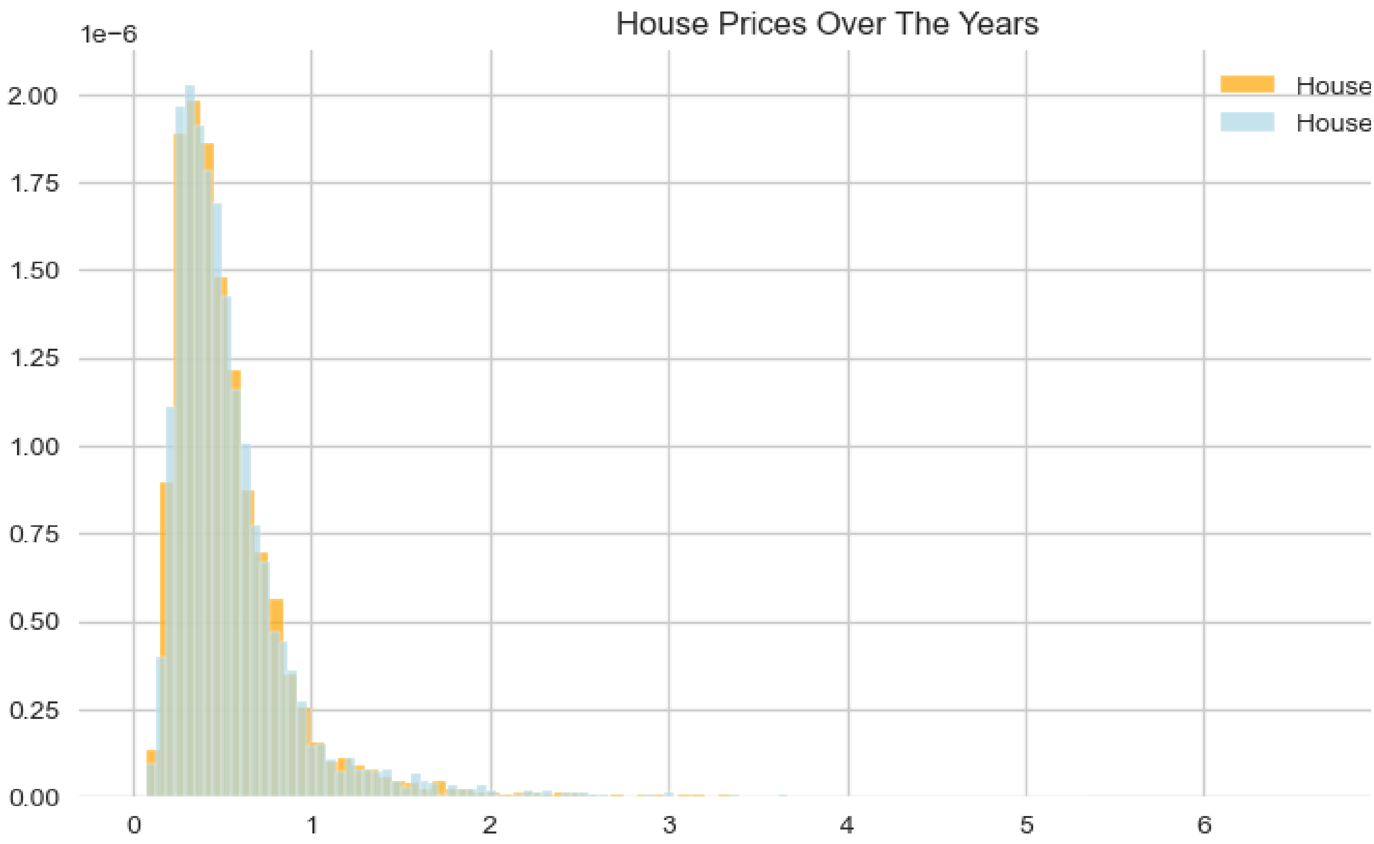
3.2.2 Current Price

> The `current price` column represents the price of the homes adjusted for inflation.

The `current price` column could be created since there are different years involved in the sa
prices of the homes may be different due to the different market conditions. Therefore, by cre
current price of the homes, we can fairly compare the prices of the homes in different years.
column, we first have to establish that the prices of the homes are indeed different due to the
shall be doing this by looking at the distributions of the prices of the homes sold in the differe

In [306]:

```
# Create different dataframes for each year (2014 and 2015)
df_2014 = data[data['yr_sold'] == 2014]
df_2015 = data[data['yr_sold'] == 2015]


# Plot the distribution of the 'price' column for each year
fig, ax = plt.subplots(figsize=(10, 5))

plt.hist(df_2014['price'], bins=100, color='orange', alpha=0.7, label='House Prices in
plt.hist(df_2015['price'], bins=100, color='lightblue', alpha=0.7, label='House Prices
plt.title('House Prices Over The Years')
plt.legend();
```



From the distribution above, we can see that there is not much difference in the prices of the years. Therefore, there is no need to create a new column with the current price of the home sold column as it is no longer needed.

In [307]:

```
# Drop the 'yr_sold' column
data.drop('yr_sold', axis=1, inplace=True)
```

3.3 Format Data

> In this section, we shall be re-formatting data as necessary.

The specific columns that we shall be looking at in this section are:

- date
- sqft_basement

3.3.1 Date

We shall be converting the date column to a datetime object.

In [308]:

```
# Convert the 'date' column to datetime format
data['date'] = pd.to_datetime(data['date'], format='%m/%d/%Y')

# Preview the first five rows of the dataframe
data.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
0	7129300520	2014-10-13	221900.0	3	1.00	1180	5650	1.0	NO	NONE
1	6414100192	2014-12-09	538000.0	3	2.25	2570	7242	2.0	NO	NONE
2	5631500400	2015-02-25	180000.0	2	1.00	770	10000	1.0	NO	NONE
3	2487200875	2014-12-09	604000.0	4	3.00	1960	5000	1.0	NO	NONE
4	1954400510	2015-02-18	510000.0	3	2.00	1680	8080	1.0	NO	NONE

3.3.2 Basement Square Footage

We shall be converting the `sqft_basement` column to a numerical column. However, we first r
'?' value that we identified in the previous phase. In the previous phase we saw that it accour
in the dataset. As this is a fairly small percentage of the total records, we shall be dropping th
value. This will ensure that we are not introducing any bias into the dataset. Once we have d
erroneous '?' value, we shall be converting the remaining `sqft_basement` column values to a

In [309]:

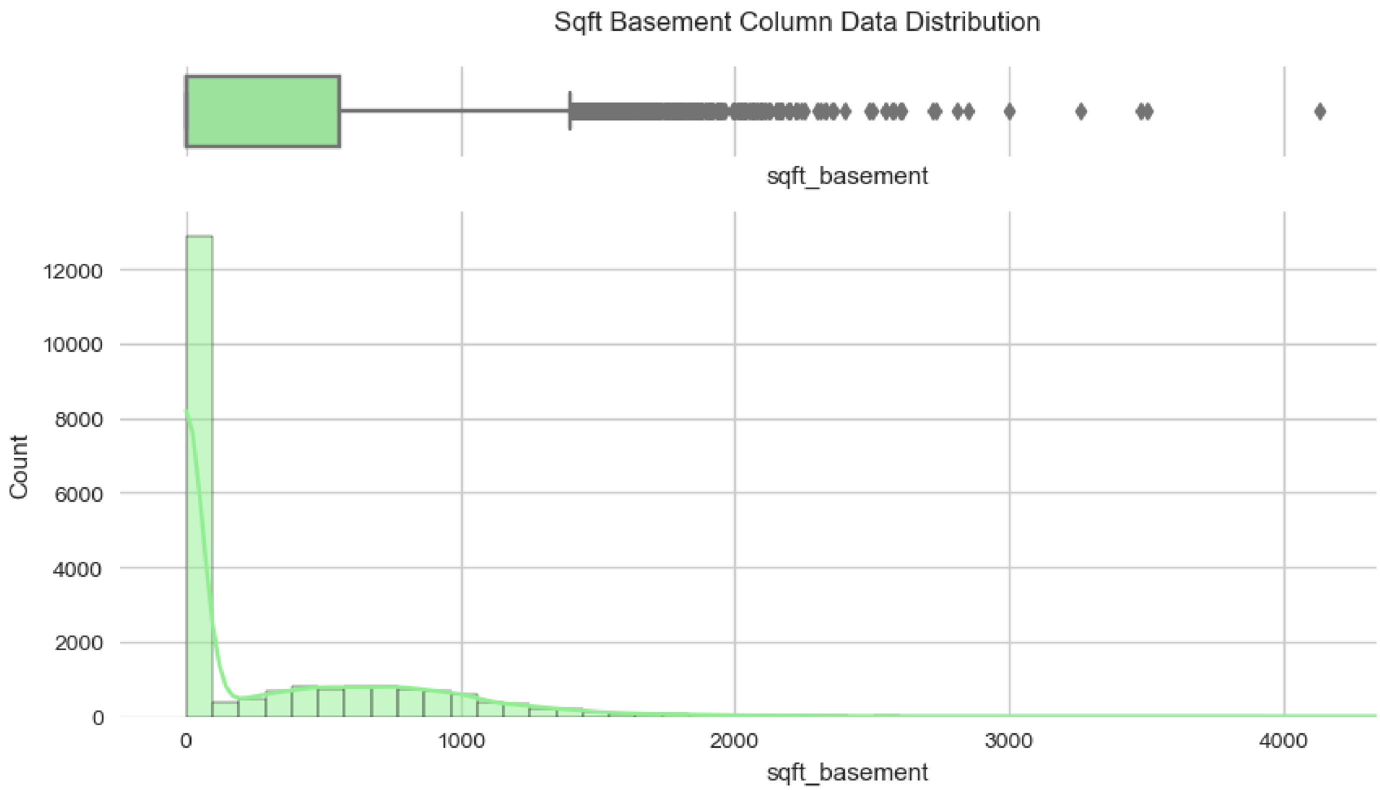
```
# Drop the records with a '?' in the 'sqft_basement' column
data = data[data['sqft_basement'] != '?']

# Convert the 'sqft_basement' column to float
data['sqft_basement'] = data['sqft_basement'].astype(float)

# Preview the first five rows of the dataframe
data.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
0	7129300520	2014-10-13	221900.0	3	1.00	1180	5650	1.0	NO	NONE
1	6414100192	2014-12-09	538000.0	3	2.25	2570	7242	2.0	NO	NONE
2	5631500400	2015-02-25	180000.0	2	1.00	770	10000	1.0	NO	NONE
3	2487200875	2014-12-09	604000.0	4	3.00	1960	5000	1.0	NO	NONE
4	1954400510	2015-02-18	510000.0	3	2.00	1680	8080	1.0	NO	NONE

In [310]:

```
# We can also visualize the data distribution of the 'sqft_basement' column
describe_data(data, 'sqft_basement')

# Plot the visualisation
plot_distribution(data, 'sqft_basement', 'Sqft Basement Column Data Distribution', 50)
```

```
count    21082.000000
mean       291.359975
std        442.007858
min          0.000000
25%          0.000000
50%          0.000000
75%        560.000000
max       4820.000000
Name: sqft_basement, dtype: float64
```



From our distributions above, we can see that the 'sqft_basement' column is highly positively the mean is higher than the median. This is as a result of the outliers in the data. Furthermor maxiumum basement size is 4820 square feet. This is quite a large basement size. However as it is as it is not erroneous.

Now that we have completely cleaned our data, we can export the cleaned data to a csv file.

In [311]:

```
# Export the dataframe to a csv file
data.to_csv('../data/processed/cleaned_kc_house_data.csv', index=False)
```

4. Modeling

> In this phase, we'll likely build and assess various models based on several different
> This phase has four tasks:
>
> - Select Modeling Techniques
> - Generate Test Design
> - Build Models
> - Assess Models

4.1 Select Modeling Techniques

> In this section, we shall be determining which algorithms to try

I believe that the best algorithm to try for this experiment is regression. Regression is a supe used to predict the value of a dependent variable based on the value of the independent vari to estimate the effect that the different features of the homes has on our dependent variable, result, we will be able to provide our stakeholder with a model that will be able to predict the homes that will have the most impact on the price of the homes.

Furthermore, as we are working with multiple features, we will be using multiple linear regres is a regression algorithm that is used to predict the value of a dependent variable based on t variables (unlike linear regression which only uses one independent variable).

4.2 Build Models

> In this section, we shall be building the models.

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

4.2.1 Build Baseline Simple Linear Regression Model

> A baseline model is essentially a simple model that acts as a reference in a machine function is to contextualize the results of trained models.

The target variable is price. Therefore, we look at the correlation coefficients for all of the pre with the highest correlation with price.

In [312]:

```
# Create a correlation matrix for the dataset
corr = data.corr()['price'].sort_values(ascending=False)
corr
```

```
price            1.000000
sqft_living      0.702004
sqft_above       0.605481
sqft_living15    0.586495
bathrooms        0.525029
sqft_basement    0.323018
bedrooms         0.308454
lat              0.307667
floors           0.256603
sqft_lot         0.088400
sqft_lot15       0.083530
yr_built         0.054849
long             0.022512
id              -0.016413
zipcode         -0.053429
Name: price, dtype: float64
```

We see that the `sqft_living` column has the highest correlation with the `price` column. This the house is a major factor in determining the price of the house. We shall also create a scat between the `sqft_living` and `price` .

In [313]:

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

We can now declare y and X_baseline variables, where y is a Series containing price data a
containing the column with the strongest correlation (`sqft_living`).

In [314]:

```python
# Declare y and X_baseline variables
y = data['price']
X_baseline = data[['sqft_living']]
```

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

In [315]:

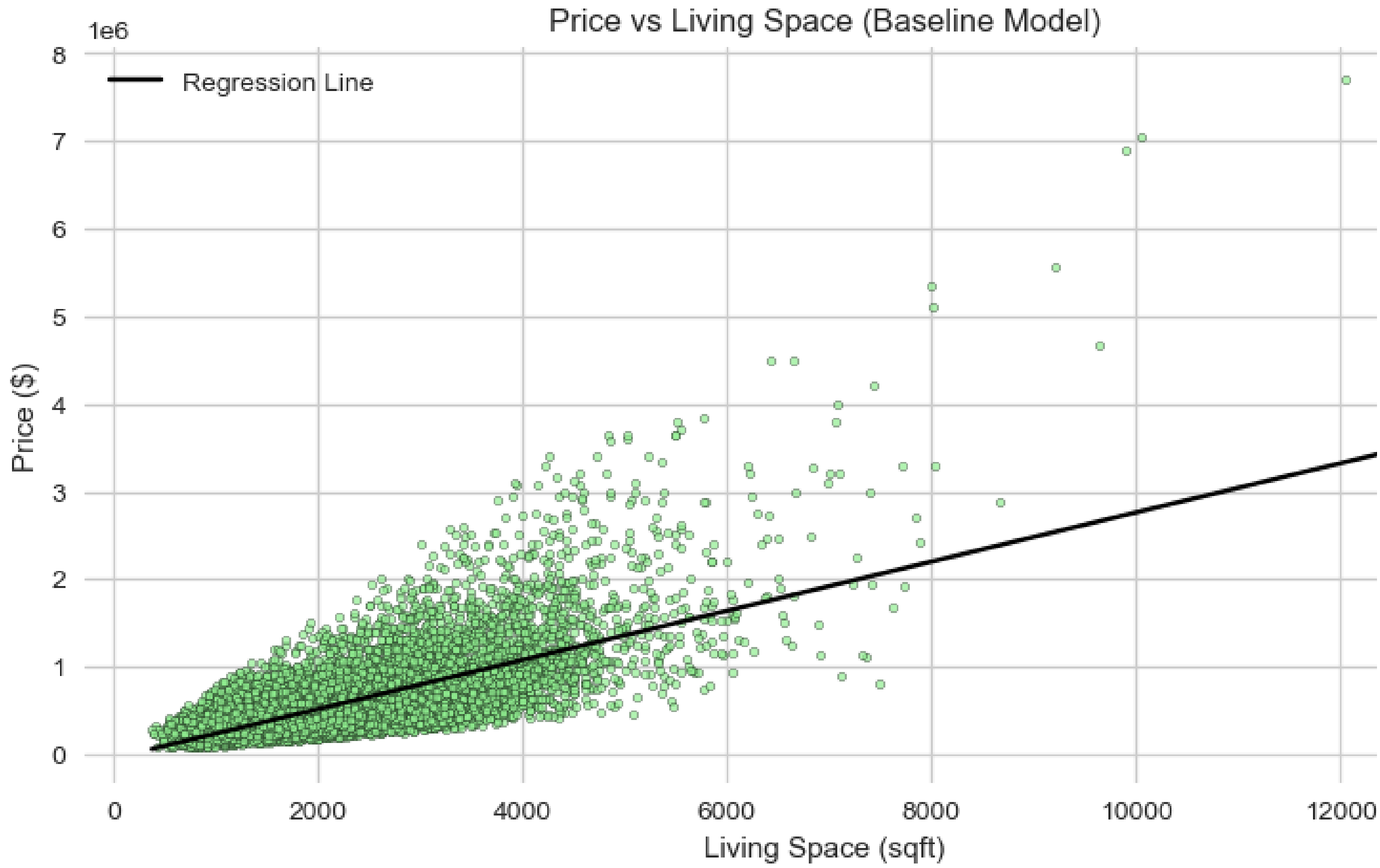
```python
# Create a baseline model
baseline_model = sm.OLS(y, sm.add_constant(X_baseline))
baseline_results = baseline_model.fit()

# Print the summary results of the baseline 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:                 2.048e+04
Date:                Sat, 01 Oct 2022   Prob (F-statistic):               0.00
Time:                        11:55:31   Log-Likelihood:            -2.9287e+05
No. Observations:               21082   AIC:                         5.857e+05
Df Residuals:                   21080   BIC:                         5.858e+05
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const       -4.327e+04   4456.393     -9.709      0.000    -5.2e+04   -3.45e+04
sqft_living   280.4877      1.960    143.116      0.000     276.646     284.329
==============================================================================
Omnibus:                    14303.984   Durbin-Watson:                   1.986
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           509767.330
Skew:                           2.786   Prob(JB):                         0.00
Kurtosis:                      26.437   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.
```

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

In [316]:

```
# 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(data.sqft_living.min(), data.sqft_living.max(), 100)
Y = baseline_results.params[0] + baseline_results.params[1] * x

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

plt.scatter(data['sqft_living'], data['price'], color='lightgreen', alpha=0.7, s=10, ed
plt.title('Price vs Living Space (Baseline Model)')
plt.xlabel('Living Space (sqft)')
plt.ylabel('Price (\$)')
plt.legend();
```



In [317]:

```
# Calculate the mean absolute error of the baseline model
baseline_mae = mean_absolute_error(y, baseline_results.predict(sm.add_constant(X_basel:
baseline_mae
```

```
173713.2378046139
```

Our most strongly correlated variable with `price` is `sqft_living`

The model is statistically significant as it explains only 50% of the variance in the data. Howe our analysis. In a typical prediction, the model is off by about $173992.

- The intercept is about -$45130. Thismeansthatthatazerosquarefoothousewouldbeu
- The coefficient of `sqft_living` is about $281. Thismeansthatforeverysquarefootincreaseinthehouse, thepriceofthehous

4.2.2 Build Iterated Multiple Linear Regression Model

> We will now iterate the baseline model by building a multiple linear regression model one independent variable.

We will start by creating a new dataframe that will contain all of the features that we want to will encode the categorical columns. In order to know which variables to keep in our model, matrix. This is done in order to reduce multicollinearity. Multicollinearity is a situation in which variables are highly correlated. This can cause problems in the model as it can lead to unsta coefficients. Therefore, we will be removing the variables that are highly correlated with each

In [318]:

```
# Declare X_iter variables
X_iter = data[['bedrooms', 'bathrooms','sqft_living', 'sqft_lot', 'floors', 'waterfront

# Preview the X_iter dataframe
X_iter
```

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft
0	3	1.00	1180	5650	1.0	NO	NONE	Average	7 Average	118(
1	3	2.25	2570	7242	2.0	NO	NONE	Average	7 Average	217(
2	2	1.00	770	10000	1.0	NO	NONE	Average	6 Low Average	770
3	4	3.00	1960	5000	1.0	NO	NONE	Very Good	7 Average	105(
4	3	2.00	1680	8080	1.0	NO	NONE	Average	8 Good	168(
...	...	...	...	...	...	...	...	...	...	...
21592	3	2.50	1530	1131	3.0	NO	NONE	Average	8 Good	153(
21593	4	2.50	2310	5813	2.0	NO	NONE	Average	8 Good	231(
21594	2	0.75	1020	1350	2.0	NO	NONE	Average	7 Average	102(
21595	3	2.50	1600	2388	2.0	NO	NONE	Average	8 Good	160(
21596	2	0.75	1020	1076	2.0	NO	NONE	Average	7 Average	102(

21082 rows × 14 columns

We have 4 categorical columns in our dataset. As a result, we will need to encode them in or our model. We will be ordinal encoding the `condition` and `grade` columns and one-hot enco columns.

4.2.2.1 Encode Categorical Columns

> We will now encode the categorical columns in the dataset.

4.2.2.1.1 Ordinal Encoding

> Ordinal encoding converts each label into integer values and the encoded data repre labels

Using the official [King County Assessor Website (https://info.kingcounty.gov/assessor/esales](https://info.kingcounty.gov/assessor/esales) were able to understand that the values in the `condition` and `grade` columns are ordinal, an based on the quality of the feature. Therefore, we will be ordinal encoding these columns.

In [319]:

```
# Create dictionaries for mapping the ordinal numberical value
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': 7,

# Map the ordinal numerical values to the 'condition' and 'grade' columns
X_iter['condition'] = X_iter['condition'].map(condition_dict)
X_iter['grade'] = X_iter['grade'].map(grade_dict)

# Preview the dataframe
X_iter
```

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_
0	3	1.00	1180	5650	1.0	NO	NONE	3	7	1180
1	3	2.25	2570	7242	2.0	NO	NONE	3	7	2170
2	2	1.00	770	10000	1.0	NO	NONE	3	6	770
3	4	3.00	1960	5000	1.0	NO	NONE	5	7	1050
4	3	2.00	1680	8080	1.0	NO	NONE	3	8	1680
...	...	...	...	...	...	...	...	...	...	...
21592	3	2.50	1530	1131	3.0	NO	NONE	3	8	1530
21593	4	2.50	2310	5813	2.0	NO	NONE	3	8	2310
21594	2	0.75	1020	1350	2.0	NO	NONE	3	7	1020
21595	3	2.50	1600	2388	2.0	NO	NONE	3	8	1600
21596	2	0.75	1020	1076	2.0	NO	NONE	3	7	1020

21082 rows × 14 columns

4.2.2.1.2 One Hot Encoding

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

We shall be encoding the remaining categorical columns (`waterfront` and `view`) using one h
order to avoid the "Dummy Variable Trap" (perfect multicollinearity between the independent

In [320]:

```
# Encode the categorical variables
X_iter = pd.get_dummies(X_iter, columns=['waterfront', 'view'], drop_first=False)

# Preview the dataframe
X_iter
```

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	condition	grade	sqft_above	sqft_basem
0	3	1.00	1180	5650	1.0	3	7	1180	0.0
1	3	2.25	2570	7242	2.0	3	7	2170	400.0
2	2	1.00	770	10000	1.0	3	6	770	0.0
3	4	3.00	1960	5000	1.0	5	7	1050	910.0
4	3	2.00	1680	8080	1.0	3	8	1680	0.0
...	...	...	...	...	...	...	...	...	...
21592	3	2.50	1530	1131	3.0	3	8	1530	0.0
21593	4	2.50	2310	5813	2.0	3	8	2310	0.0
21594	2	0.75	1020	1350	2.0	3	7	1020	0.0
21595	3	2.50	1600	2388	2.0	3	8	1600	0.0
21596	2	0.75	1020	1076	2.0	3	7	1020	0.0

21082 rows × 19 columns

In the `waterfront` column, we shall be dropping the `waterfront_NO` column as the reference
study the effect of having a house on a waterfront. In the `view` column, we shall be dropping
reference column. This will allow us to study the effect of having a house with a view. In addi
value in the column.

In [321]:

```
# Drop the 'waterfront_NO' and 'view_NONE' columns
X_iter. drop(['waterfront_NO', 'view_NONE'], axis=1, inplace=True)

# Preview the dataframe
X_iter
```

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	condition	grade	sqft_above	sqft_basem
0	3	1.00	1180	5650	1.0	3	7	1180	0.0
1	3	2.25	2570	7242	2.0	3	7	2170	400.0
2	2	1.00	770	10000	1.0	3	6	770	0.0
3	4	3.00	1960	5000	1.0	5	7	1050	910.0
4	3	2.00	1680	8080	1.0	3	8	1680	0.0
...	...	...	...	...	...	...	...	...	...
21592	3	2.50	1530	1131	3.0	3	8	1530	0.0
21593	4	2.50	2310	5813	2.0	3	8	2310	0.0
21594	2	0.75	1020	1350	2.0	3	7	1020	0.0
21595	3	2.50	1600	2388	2.0	3	8	1600	0.0
21596	2	0.75	1020	1076	2.0	3	7	1020	0.0

21082 rows × 17 columns

4.2.2.2 Correlation Matrix

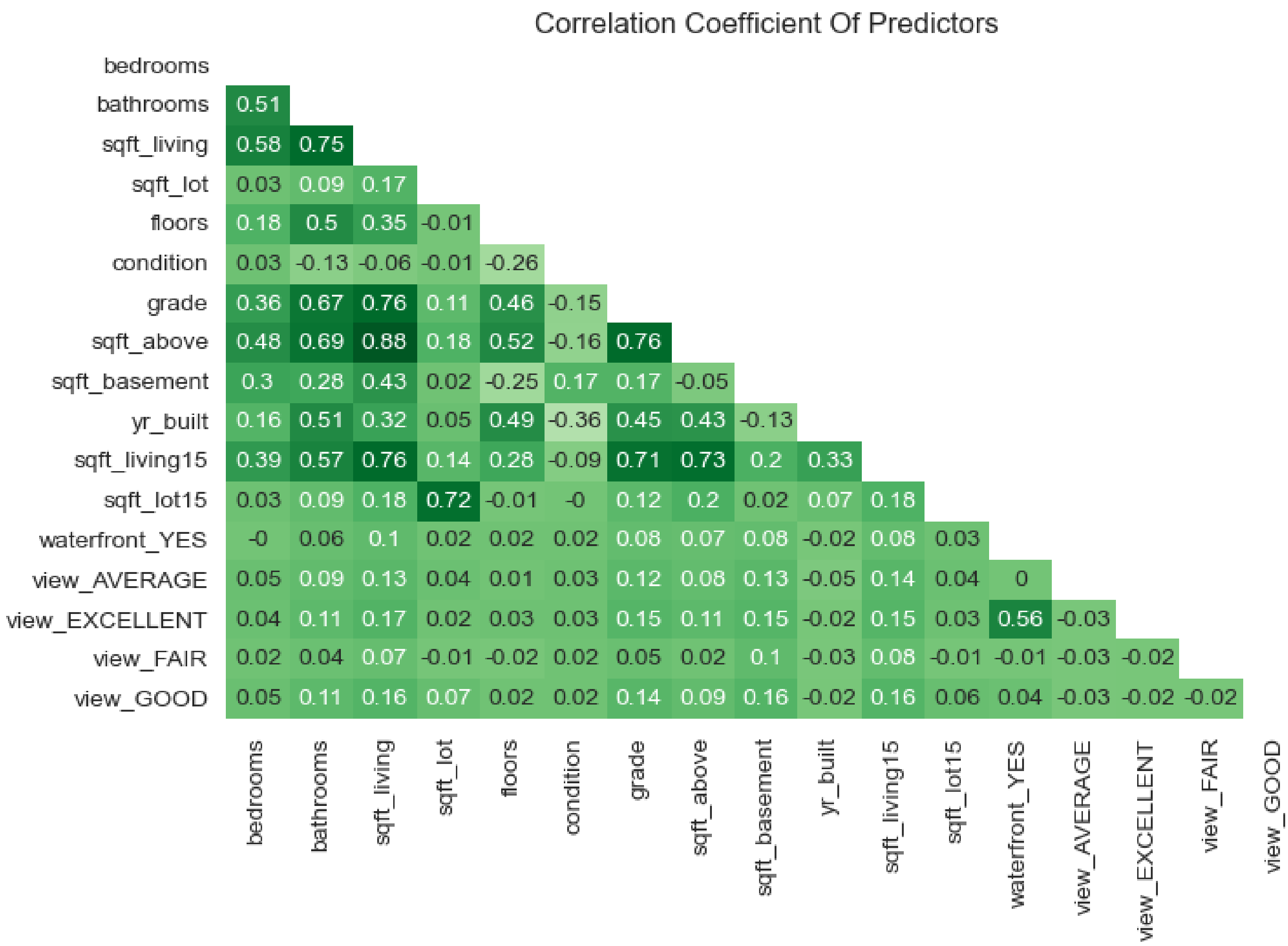
> A correlation matrix is a table showing correlation coefficients between variables

We will now analyse the correlation matrix to determine which variables to keep in our mode matrix, we will also be looking at the VIF (Variance Inflation Factor) of each variable. The VIF variance of an estimated regression coefficient increases if the independent variables are co

We are aiming to ensure that a correlation coefficient is less than 0.6 and a VIF is less than ! coefficient of 0.6 or higher indicates that the variables are highly correlated. A VIF of 5 or hig are highly correlated.

In [322]:

```
# Define function to plot the correlation matrix
def corrmatrix(df):
    ''' This function plots a correlation matrix for a given dataframe '''
    plt.figure(figsize=(10, 5))

    corr = df.corr()

    # Generate a mask to only show the bottom triangle
    mask = np.triu(np.ones_like(corr, dtype=bool))

    # generate heatmap
    sns.heatmap(round(corr,2), annot=True, mask=mask, vmin=-1, vmax=1, cmap='Greens')
    plt.title('Correlation Coefficient Of Predictors')
    plt.show()

# Define function to print the VIF values of the predictors
def vif_df(df):
    ''' This function prints the VIF values of the predictors in a given dataframe '''
    vif_data = pd.DataFrame()

    # Add a constant to the dataframe
    X = df.assign(const=1)

    vif_data["feature"] = X.columns

    # calculating VIF for each feature
    vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1]

    print(vif_data.sort_values(by='VIF', ascending=False))



# Plot the correlation matrix
corrmatrix(X_iter)

# Print the VIF values of the predictors
vif_df(X_iter)
```

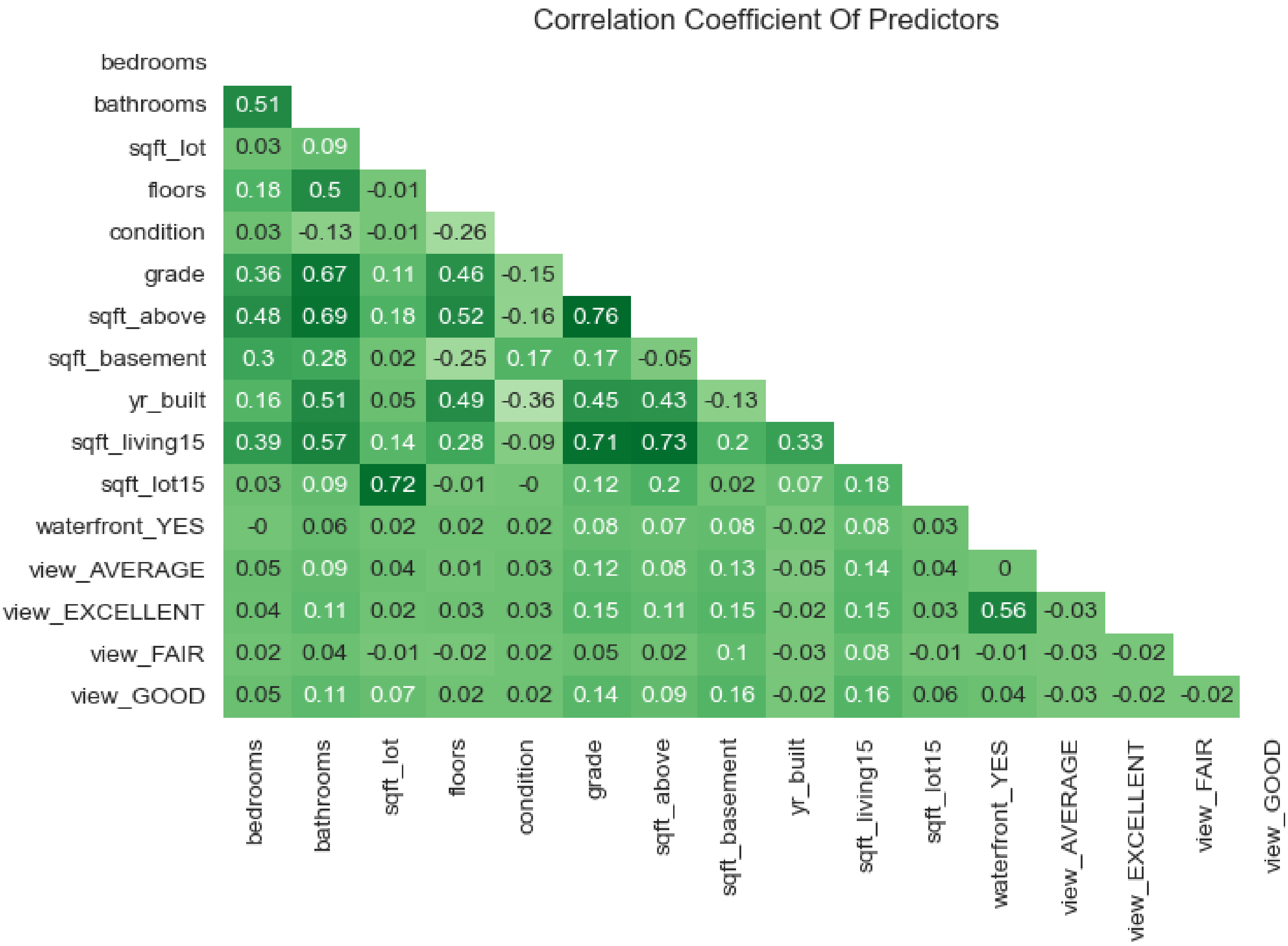


	feature	VIF
2	sqft_living	inf
7	sqft_above	inf
8	sqft_basement	inf
17	const	7991.177201
1	bathrooms	3.302484
6	grade	3.240348
10	sqft_living15	2.812048
11	sqft_lot15	2.123483
3	sqft_lot	2.094685
4	floors	1.934228
9	yr_built	1.816720
0	bedrooms	1.641691
14	view_EXCELLENT	1.546534
12	waterfront_YES	1.478656
5	condition	1.187949
16	view_GOOD	1.079328
13	view_AVERAGE	1.063058
15	view_FAIR	1.025156

We can see a very high VIF value for the `sqft_living` , `sqft_above` , and `sqft_basement` colu dropping `sqft_living` from our model.

In [323]:

```python
# Drop the 'sqft_living' column
X_iter.drop(['sqft_living'], axis=1, inplace=True)

# Visualize the correlation matrix and the VIF dataframe
corrmatrix(X_iter)
vif_df(X_iter)
```



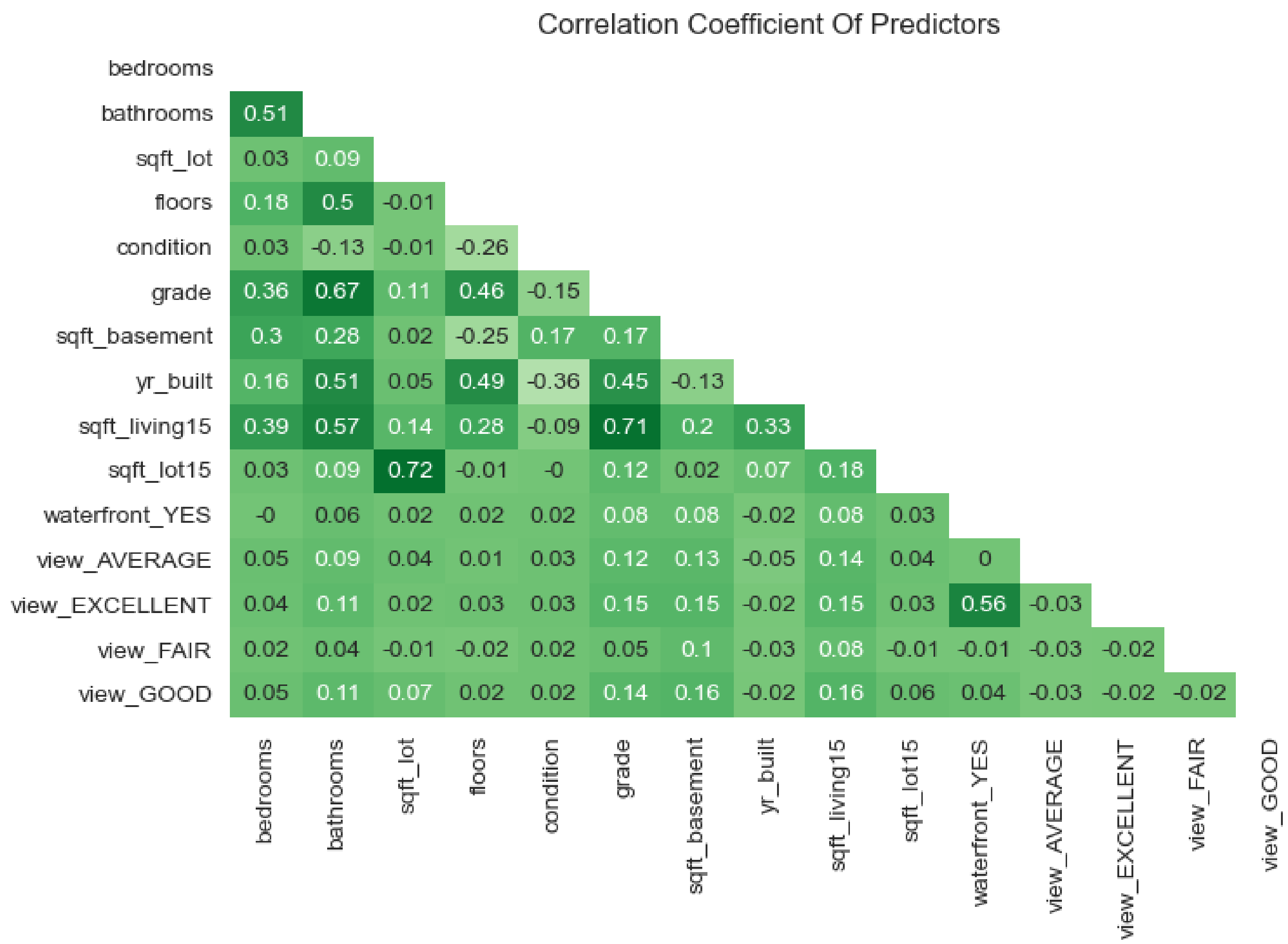
	feature	VIF
16	const	7991.177201
6	sqft_above	4.834813
1	bathrooms	3.302484
5	grade	3.240348
9	sqft_living15	2.812048
10	sqft_lot15	2.123483
2	sqft_lot	2.094685
7	sqft_basement	1.981861
3	floors	1.934228
8	yr_built	1.816720
0	bedrooms	1.641691
13	view_EXCELLENT	1.546534
11	waterfront_YES	1.478656

4	condition	1.187949
15	view_GOOD	1.079328
12	view_AVERAGE	1.063058
14	view_FAIR	1.025156

The correlation matrix shows that the `sqft_above` column still has a high correlation. Therefo
`sqft_above` from our model.

In [324]:

```python
# Drop the 'sqft_above' column
X_iter.drop(['sqft_above'], axis=1, inplace=True)

# Visualize the correlation matrix and the VIF dataframe
corrmatrix(X_iter)
vif_df(X_iter)
```



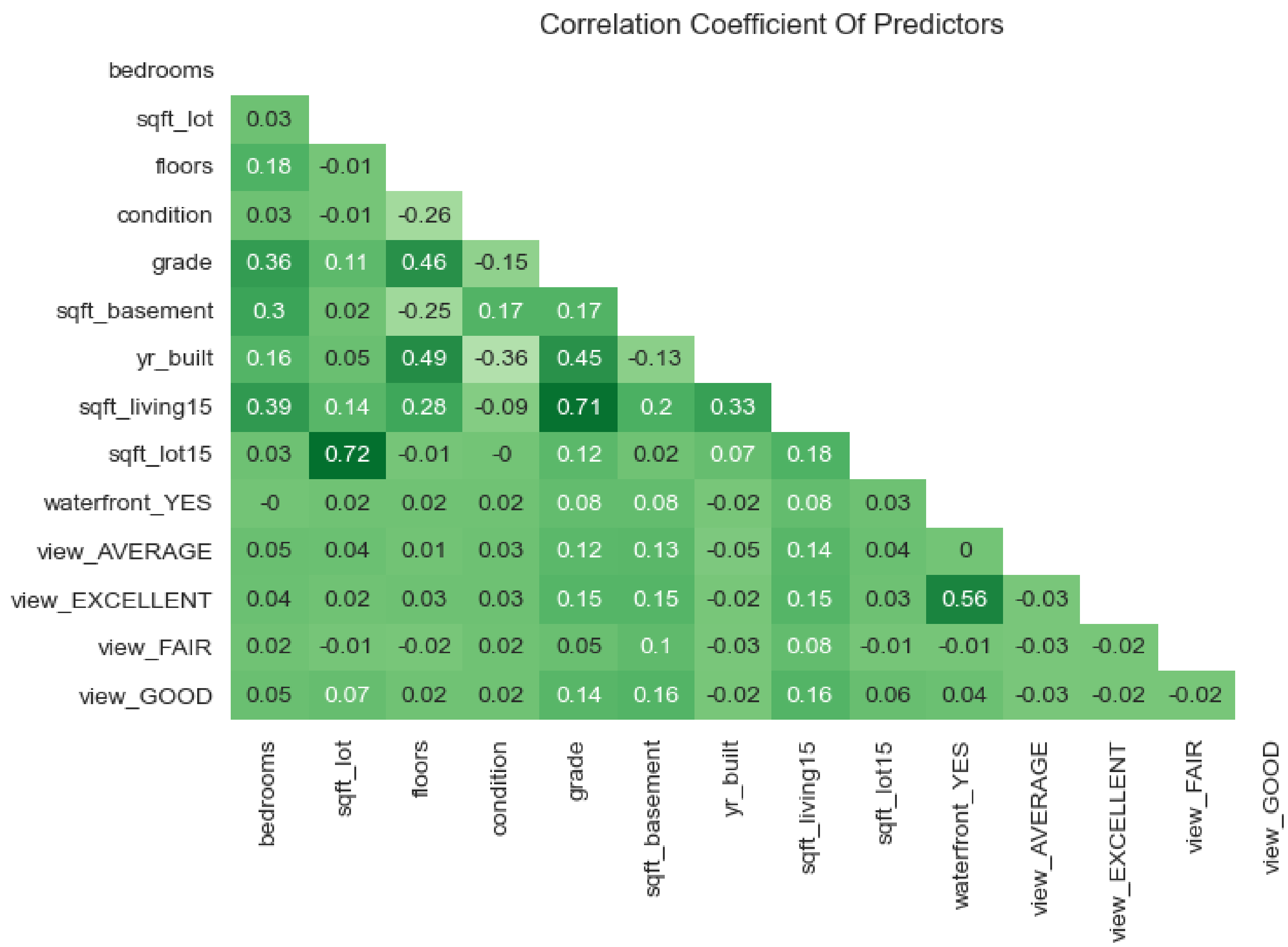

```
              feature          VIF
15              const  7956.618704
1           bathrooms     2.969652
5               grade     2.780924
8       sqft_living15     2.282297
9          sqft_lot15     2.114176
2            sqft_lot     2.082816
3              floors     1.896715
7            yr_built     1.790623
6       sqft_basement     1.578408
12     view_EXCELLENT     1.546500
10     waterfront_YES     1.477030
0            bedrooms     1.475156
4           condition     1.187719
```

```
14       view_GOOD     1.078949
11    view_AVERAGE     1.062833
13       view_FAIR     1.025087
```

The correlation coefficients for the `bathrooms` column is still higher than our threshold. There
`bathrooms` column from our model.

In [325]:

```python
# Drop the 'bathrooms' column
X_iter.drop(['bathrooms'], axis=1, inplace=True)

# Visualize the correlation matrix and the VIF dataframe
corrmatrix(X_iter)
vif_df(X_iter)
```



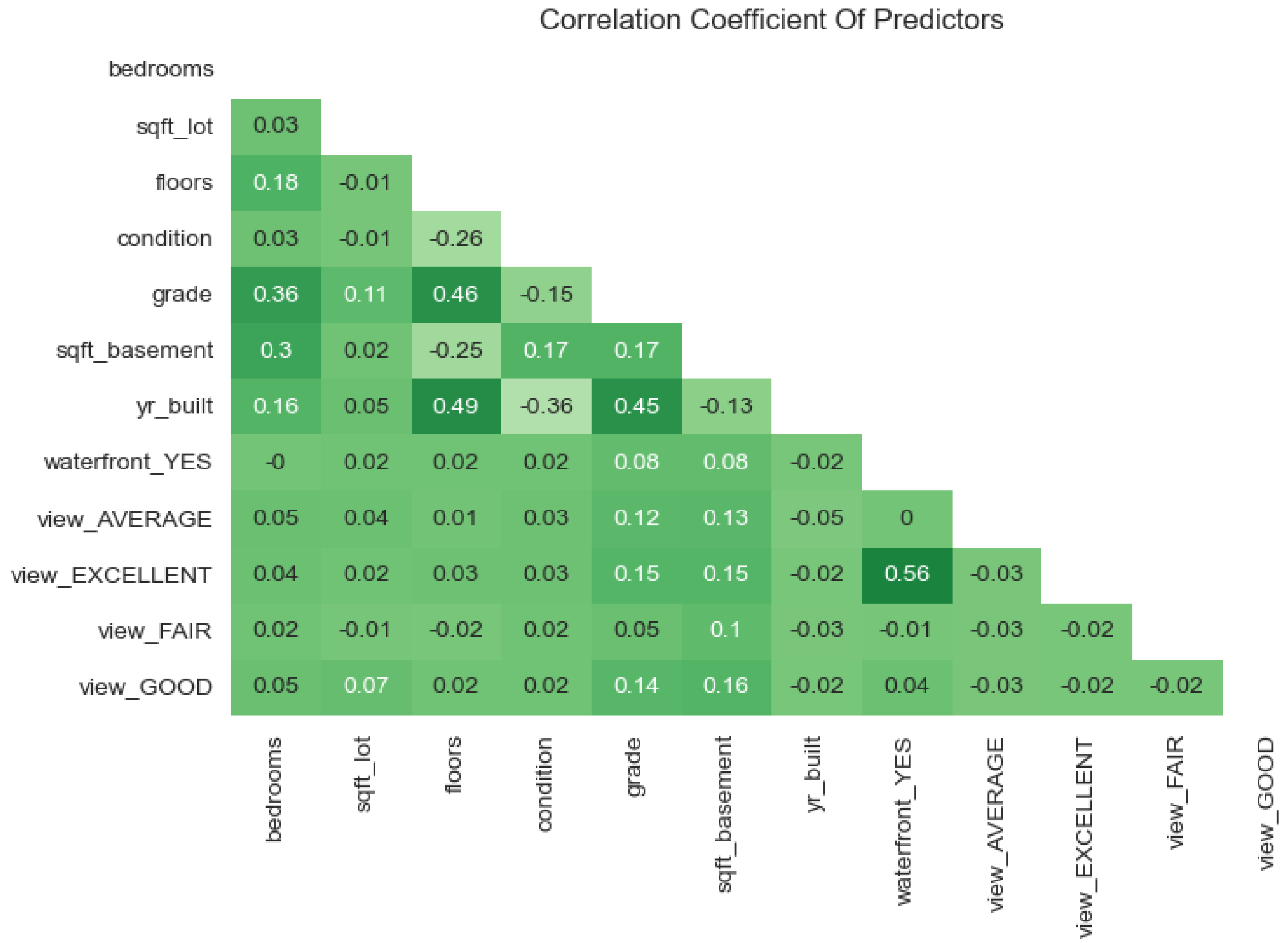
	feature	VIF
14	const	7066.323721
4	grade	2.621639
7	sqft_living15	2.249374
8	sqft_lot15	2.114127
1	sqft_lot	2.080635
2	floors	1.670595
6	yr_built	1.615203
11	view_EXCELLENT	1.546476
9	waterfront_YES	1.476606
5	sqft_basement	1.401570
0	bedrooms	1.323164
3	condition	1.185202
13	view_GOOD	1.078677

```
10    view_AVERAGE     1.062599
12       view_FAIR     1.025008
```

Dropping the `bathrooms` column has further reduced the overall correlation in the dataset. Ho
`sqft_lot15` columns still have a high correlation. Therefore, we will be dropping them both fr

In [326]:

```python
# Drop the 'sqft_lot15' and 'sqft_living15' column
X_iter.drop(['sqft_lot15', 'sqft_living15'], axis=1, inplace=True)

# Visualize the correlation matrix and the VIF dataframe
corrmatrix(X_iter)
vif_df(X_iter)
```



	feature	VIF
12	const	6995.603649
4	grade	1.753385
2	floors	1.658031
6	yr_built	1.606184
9	view_EXCELLENT	1.537416
7	waterfront_YES	1.476263
5	sqft_basement	1.400994
0	bedrooms	1.266129
3	condition	1.184633
11	view_GOOD	1.068646
8	view_AVERAGE	1.054079
1	sqft_lot	1.024057
10	view_FAIR	1.019902

Now that we have our VIF and correlation matrix, below the threshold, we can now build our model.

4.2.2.3 Build Model

> We will now build our multiple linear regression model.

In [327]:

```python
# Create a model
iterated_model = sm.OLS(y, sm.add_constant(X_iter))
iterated_results = iterated_model.fit()

# Print the summary results of the baseline model
print(iterated_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.601
Model:                            OLS   Adj. R-squared:                  0.601
Method:                 Least Squares   F-statistic:                     2647.
Date:                Sat, 01 Oct 2022   Prob (F-statistic):               0.00
Time:                        11:55:37   Log-Likelihood:            -2.9033e+05
No. Observations:               21082   AIC:                         5.807e+05
Df Residuals:                   21069   BIC:                         5.808e+05
Df Model:                          12
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const           5.137e+06   1.33e+05     38.501      0.000    4.88e+06     5.4e+06
bedrooms        1.624e+04   1940.787      8.368      0.000    1.24e+04       2e+04
sqft_lot           0.1365      0.039      3.482      0.000       0.060       0.213
floors          7.826e+04   3808.798     20.548      0.000    7.08e+04    8.57e+04
condition       1.944e+04   2669.062      7.284      0.000    1.42e+04    2.47e+04
grade           2.051e+05   1799.964    113.942      0.000    2.02e+05    2.09e+05
sqft_basement    118.8650      4.272     27.822      0.000     110.491     127.239
yr_built       -3277.1706     68.955    -47.526      0.000   -3412.328   -3142.013
waterfront_YES  5.409e+05   2.38e+04     22.745      0.000    4.94e+05    5.88e+05
view_AVERAGE    6.994e+04   7976.568      8.768      0.000    5.43e+04    8.56e+04
view_EXCELLENT  3.409e+05   1.64e+04     20.776      0.000    3.09e+05    3.73e+05
view_FAIR       1.311e+05    1.3e+04     10.052      0.000    1.06e+05    1.57e+05
view_GOOD       1.371e+05   1.09e+04     12.595      0.000    1.16e+05    1.58e+05
==============================================================================
Omnibus:                    18594.679   Durbin-Watson:                   1.974
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          2185504.878
Skew:                           3.741   Prob(JB):                         0.00
Kurtosis:                      52.315   Cond. No.                     3.67e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.67e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Now we can compare the baseline and iterated model statistics.

In [328]:

```
# Compare the baseline and iterated model mean absolute errors
iterated_mae = mean_absolute_error(y, iterated_results.predict(sm.add_constant(X_iter)
print("Baseline Model Mean Absolute Error: ", baseline_mae)
print("Iterated Model Mean Absolute Error: ", iterated_mae)

# Compare the adjusted R-squared values of the baseline and model
print("Baseline Model Adjusted R-squared: ", baseline_results.rsquared_adj)
print("Iterated Model Adjusted R-squared: ", iterated_results.rsquared_adj)
```

```
Baseline Model Mean Absolute Error:  173713.2378046139
Iterated Model Mean Absolute Error:  147296.625677394
Baseline Model Adjusted R-squared:  0.49278505895823355
Iterated Model Adjusted R-squared:  0.6010210227347927
```

From the model results, we can see that the model is statistically significant as it explains 60 compared to the 49% in the baseline model. Furthermore, the model is off by about $147, 29$ baseline model. This is a significant improvement.

We will now do an analysis of the coefficients of the model.

In [329]:

```
# Create a dataframe of the coefficients of the iterated model along with their p-valu
results_df = pd.concat([round(iterated_results.params,3), round(iterated_results.pvalu
results_df.columns = ["coefficient", "p-value"]
results_df
```

	coefficient	p-value
const	5137454.742	0.000000
bedrooms	16239.726	0.000000
sqft_lot	0.137	0.000498
floors	78262.473	0.000000
condition	19442.583	0.000000
grade	205091.669	0.000000
sqft_basement	118.865	0.000000
yr_built	-3277.171	0.000000
waterfront_YES	540924.431	0.000000
view_AVERAGE	69940.897	0.000000
view_EXCELLENT	340899.246	0.000000
view_FAIR	131058.754	0.000000
view_GOOD	137050.033	0.000000

4.2.2.4 Model Results Analysis

We can see that all of the variables in the model are statistically significant.

- We can see that constant value is about $5,137,455. Thismeansthatahousewithnofeatureswouldbeworthabout$5,137,455.
- The coefficient of `bedrooms` is $16,240 whichmeansthatforeverybedroomincreaseinthehouse, thepriceofthehouseincreasesbyabout$16,240.
- The coefficient of `ssqft_lot` is $0.14 whichmeansthatforeverysquarefootincreaseinthelot, thepriceofthehousei$
- The coefficient of `floors` is $78,262 whichmeansthatforeveryfloorincreaseinthehouse, thepriceofthehousei$
- The coefficient of `condition` is $19,443 whichmeansthatforeveryconditionratingincreaseinthehouse, thepriceofthehouseincreasesbyabout$19,443.

- The coefficient of `grade` is

  $205,092 whichmeansthatforeverygraderatingincreaseinthehouse, thepriceofthehouseincreasesbyabout$ 205,092.
- The coefficient of `sqft_basement` is

  $119 whichmeansthatforeverysquarefootincreaseinthebasement, thepriceofthehouseincreasesbyabout$ 119.
- The coefficient of `yr_built` is -

  $3,277 whichmeansthatforeveryyearincreaseintheyearthehousewasbuilt, thepriceofthehousedecreasesbyabout$ 3,277.
- The coefficient of `waterfront_YES` is

  $540,924 thismeansthatifahouseisonawaterfront, thepriceofthehouseincreases$
- The coefficients for `view` range from $69,941 to$ 340,899
  - `view_AVERAGE` is

    $69,941 whichmeansthatforanaverageviewcomparedtonoview, thepriceofthehouseincreasesbyabout$ 69,941.
  - `view_FAIR` is

    $131,058 whichmeansthatforafairviewcomparedtonoview, thepriceofthehouseincreasesbyabout$ 131,058.
  - `view_GOOD` is

    $137,050 whichmeansthatforagoodviewcomparedtonoview, thepriceofthehouseincreasesbyabout$ 137,050.
  - `view_EXCELLENT` is

    $340,899 whichmeansthatforanexcellentviewcomparedtonoview, thepriceofthehouseincreasesbyabout$ 340,899.
- This `view` outcome is surprising since we would expect that the effect of having an aver having a fair view. However, the model shows that the effect of having a fair view is bett This could perhaps suggest that the homes in the dataset with an average view are not view, or that that homes with an average value have been undervalued.

5. Conclusion

> In this phase we will be interpreting the model results and limitations in the context of
> and giving reccomendations to the stakeholder based on our modeling results.

5.1 Recommendations

Taking this analysis back to the original business problem, the aim was to help a real estate
the best possible potential renovations to make to increase the the value. After modelling the
renovations are as follows:

- Moving the house closer to the water. This will increase the value of the house by about
  $540, 924. Asaresultthismostlikelymeansthatitmaketheviewexcellentasthetwofeatu. Inturn, bymakingtheviewexcellent, thevalueofthehousewillincreasebyabout$
  340,899. However, this renovation can only be made if land is close water.
- The second best renovation to make is to improve the grade of the house. This will incre
  about $205,092 for every grade.
- The third best renovation to make is to increase the number of floors in the house. This
  house by about $78,262 for every floor. However, it is worth mentioning that our data on
  maximum. Therefore, it is unlikely that this statistic would apply to a house with more tha
- Increasing the number of bedrooms in the house will increase the value of the house by
  bedroom. However, it is worth mentioning that our data only had 10 bedrooms as the ma
  that this statistic would apply to a house with more than 10 bedrooms.
- Lastly, increasing the size of the basement will increase the value of the house by about
  However, it is worth mentioning that our data only had 4,820 square feet as the maximu
  this statistic would apply to a house with more than 4,820 square feet.

5.2 Limitations

Though our model did show a significant increase in the accuracy of the model, there are stil
These limitations are as follows:

- The data in the dataset is from 2014 and 2015. Therefore, it may not be able to account
  market since then. As a result the model may not be able to predict the value of a house
- In order to improve the value of a house, we would need to understand the market (i.e.
  Therefore, by not having this information, we are unable to advise our clients on the bes
  possible to build the most expensive house in the world, but if it is not what buyers are l
  sold. There is no value in that.
- By using a correlation threshold of 0.6, we may have ignored dropping some freatures w
  have led to multicollinearity in the model. As a result, the model may not be able to pred
  accurately.