House Sales Regression analysis model

ID	Name
22010110	سلسبيل محمد عسكر
22011629	سلمى طارق عبدالمجيد محمد
22010340	روان كمال فياض
22010380	لينا عصام محمد
22011531	ماريهان عماد الدين محمود
22010082	حبيبة طلعت أحمد منصور
22010193	كريم أحمد خليل
22011942	شهد رضا فراج
22010371	غادة السيد فرج
22011454	روان فايز محمد
22010321	اسراء مصطفى إبراهيم

ID	Name
22010284	نهال خالد محمدصديق علي
22010175	عمرو حسين اسماعيل
22011437	جهاد يسري إبراهيم علي
22010157	عمر أسامة فؤاد سيد قنديل
22010294	ياسين محمود علي
22010172	عمر مهاب إبراهيم

Introduction

This dataset gives us a close look at the houses sold in King County, which covers Seattle in the United States. It provides details on home sales from May 2014 to May 2015. Understanding this dataset is like having a window into the local real estate market during that time. We can use this information to understand what factors influence house prices and how they vary across different properties. Our main goal is to use this data to predict the price of houses based on their characteristics. This predictive analysis can help buyers understand how much they might expect to pay for a house with certain features, and sellers can use it to set prices that reflect the value of their property accurately.

About the Dataset: This dataset is packed with information about various aspects of houses being sold. Here's a simple breakdown of what each column tells us :

1. id: A unique number given to each house sale.
2. date: The date the house was sold.
3. price: The amount of money the house was sold for.
4. bedrooms: How many bedrooms the house has.
5. bathrooms: How many bathrooms the house has.
6. sqft_living: The total area of the house in square feet.
7. sqft_lot: The area of land the house sits on.
8. floors: How many floors the house has.
9. waterfront: Whether the house has a view of the water (yes or no).
10. view: A rating of the quality of the view from the house.
11. condition: The overall condition of the house.
12. grade: A grade given to the house based on its quality and design.
13. sqft_above: The area of the house above ground level.
14. sqft_basement: The area of the house that is a basement.
15. yr_built: The year the house was built.
16. yr_renovated: The year the house was last renovated.
17. zipcode: The postal code of the area where the house is located.
18. lat: The latitude of the house's location.
19. long: The longitude of the house's location.
20. sqft_living15: The area of the living space of the 15 nearest houses.
21. sqft_lot15: The area of the land of the 15 nearest houses.

Target of Linear Regression

Our main aim is to use the information provided in this dataset to predict how much houses are likely to sell for. By analyzing factors like the number of bedrooms, the size of the house, its condition, and other features, we can build a model that estimates the selling price of a house. This prediction can be incredibly useful for buyers who want to know if they're getting a fair deal, sellers who want to set the right price for their property, and real estate professionals who need to understand market trends and property values. Ultimately, by using linear regression, we hope to uncover the key factors that determine house prices in King County, helping everyone involved in the housing market make more informed decisions.

---

first we import the following libraries to help us with cleaning the data , using mathematical functions and analysis. They also help with the visualization and modeling and presenting the data in a clean , organized way.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.formula.api import ols
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from scipy.stats import zscore
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import warnings
warnings.filterwarnings('ignore')
import statsmodels.api as sm
from sklearn import linear_model
from tabulate import tabulate
import scipy.stats as stats
```

```python
# Read the housing data
kchd = pd.read_csv('kc_house_data.csv')
kchd
```

[4] Python

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1955	0	98178	47.5112	-122.257	1340	5650
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	1951	1991	98125	47.7210	-122.319	1690	7639
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	1933	0	98028	47.7379	-122.233	2720	8062
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	1965	0	98136	47.5208	-122.393	1360	5000
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	1987	0	98074	47.6168	-122.045	1800	7503
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21608	263000018	20140521T000000	360000.0	3	2.50	1530	1131	3.0	0	0	...	8	1530	0	2009	0	98103	47.6993	-122.346	1530	1509
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	5813	2.0	0	0	...	8	2310	0	2014	0	98146	47.5107	-122.362	1830	7200
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	1350	2.0	0	0	...	7	1020	0	2009	0	98144	47.5944	-122.299	1020	2007
21611	291310100	20150116T000000	400000.0	3	2.50	1600	2388	2.0	0	0	...	8	1600	0	2004	0	98027	47.5345	-122.069	1410	1287
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	1076	2.0	0	0	...	7	1020	0	2008	0	98144	47.5941	-122.299	1020	1357

21613 rows × 21 columns

This code reads a data file called 'kc_house_data.csv' using the Pandas library and stores the data in a DataFrame called `kchd`. Once the data is read, the DataFrame shows `kchd`.

```
[8] kchd.describe(include='all')
```

Python

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipc
count	2.161300e+04	21613	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	...	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.00(
unique	NaN	372	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
top	NaN	20140623T000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
freq	NaN	142	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
mean	4.580302e+09	NaN	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	...	7.656873	1788.390691	291.509045	1971.005136	84.402258	98077.93
std	2.876566e+09	NaN	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318	...	1.175459	828.090978	442.575043	29.373411	401.679240	53.50
min	1.000102e+06	NaN	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	...	1.000000	290.000000	0.000000	1900.000000	0.000000	98001.00(
25%	2.123049e+09	NaN	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	...	7.000000	1190.000000	0.000000	1951.000000	0.000000	98033.00(
50%	3.904930e+09	NaN	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	...	7.000000	1560.000000	0.000000	1975.000000	0.000000	98065.00(
75%	7.308900e+09	NaN	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	...	8.000000	2210.000000	560.000000	1997.000000	0.000000	98118.00(
max	9.900000e+09	NaN	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	...	13.000000	9410.000000	4820.000000	2015.000000	2015.000000	98199.00(

11 rows × 21 columns

We use `include='all'` to provide a statistical summary of all columns in a DataFrame, so all columns are included in the description regardless of data type.

This statistical description can be useful for understanding the distributions and basic statistics of each variable. The statistical description usually includes the total number of values, the median value, the standard deviation, the minimum and maximum values, and the first and third quartiles of the distribution. This information helps in understanding the data set and determining the next steps in analyzing the data.

```
[5] kchd.head()
```

Python

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1955	0	98178	47.5112	-122.257	1340	5650
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	1951	1991	98125	47.7210	-122.319	1690	7639
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	1933	0	98028	47.7379	-122.233	2720	8062
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	1965	0	98136	47.5208	-122.393	1360	5000
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	1987	0	98074	47.6168	-122.045	1800	7503

5 rows × 21 columns

```
[6] kchd.tail()
```

Python

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
21608	263000018	20140521T000000	360000.0	3	2.50	1530	1131	3.0	0	0	...	8	1530	0	2009	0	98103	47.6993	-122.346	1530	1509
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	5813	2.0	0	0	...	8	2310	0	2014	0	98146	47.5107	-122.362	1830	7200
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	1350	2.0	0	0	...	7	1020	0	2009	0	98144	47.5944	-122.299	1020	2007
21611	291310100	20150116T000000	400000.0	3	2.50	1600	2388	2.0	0	0	...	8	1600	0	2004	0	98027	47.5345	-122.069	1410	1287
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	1076	2.0	0	0	...	7	1020	0	2008	0	98144	47.5941	-122.299	1020	1357

5 rows × 21 columns

• We used kchd.head() to display the first rows of data.
• We used `kchd.tail()` to display the last rows of data.
This allows the data to be inspected and ensured that it has been read correctly and looks as expected. You can use this to check different columns and values in the first and last rows of the data.

```
[7] kchd.sample()
```

Python

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
4984	1443550020	20150506T000000	570000.0	4	2.5	2640	11816	2.0	0	0	...	8	2640	0	1999	0	98019	47.733	-121.968	2400	11816

1 rows × 21 columns

We used `kchd.sample()` to display a random sample of data. This is useful for checking the distribution of data and making sure that the data represents a good variety of values. This function returns a random row of data

```
kchd.describe(include='all')
```

Python

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipc
count	2.161300e+04	21613	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	...	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.00(
unique	NaN	372	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
top	NaN	20140623T000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
freq	NaN	142	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
mean	4.580302e+09	NaN	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	...	7.656873	1788.390691	291.509045	1971.005136	84.402258	98077.93
std	2.876566e+09	NaN	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318	...	1.175459	828.090978	442.575043	29.373411	401.679240	53.50
min	1.000102e+06	NaN	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	...	1.000000	290.000000	0.000000	1900.000000	0.000000	98001.00(
25%	2.123049e+09	NaN	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	...	7.000000	1190.000000	0.000000	1951.000000	0.000000	98033.00(
50%	3.904930e+09	NaN	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	...	7.000000	1560.000000	0.000000	1975.000000	0.000000	98065.00(
75%	7.308900e+09	NaN	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	...	8.000000	2210.000000	560.000000	1997.000000	0.000000	98118.00(
max	9.900000e+09	NaN	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	...	13.000000	9410.000000	4820.000000	2015.000000	2015.000000	98199.00(

11 rows × 21 columns

We use `include='all'` to provide a statistical summary of all columns in a DataFrame, so all columns are included in the description regardless of data type.
This statistical description is useful for understanding the distributions and basic statistics of each variable. The statistical description usually includes the total number of values, themedian value, the standard deviation, the minimum and maximum values, and the first and third quartiles of the distribution. This information helps in understanding the data set and determining the next steps in analyzing the data.

`kchd.info()` gives a summary of the DataFrame data, including the number of rows and columns, the data types in each column, and whether there are missing values (NaN) in each column.

```
kchd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

```
#clean
#rename cols.
kchd.rename(columns={'sqft_living': 'Area_of_living_room',
                     'sqft_lot': 'Land_area',
                     'sqft_above': 'Area_of_housing_space',
                     'sqft_basement': 'Area_of_housing_space_below_ground_level',
                     'yr_built': 'Year_built',
                     'yr_renovated': 'Year_renovated',
                     'lat': 'Latitude',
                     'long': 'Longitude',
                     'sqft_living15':'Avg Living for closest 15 houeses',
                     'sqft_lot15':'Avg Land for closest 15 houeses'}, inplace=True)
kchd
```

Python

	id	date	price	bedrooms	bathrooms	Area_of_living_room	Land_area	floors	waterfront	view	...	Year_renovated	zipcode	Latitude	Longitude	Avg Living for closest 15 houeses	Avg Land for closest 15 houeses	logPrice	LogPricePerSquareFeet	avgZipCodePrice	log_avgZipCodePrice
0	7129300520	2014-10-13	221900.0	3	1	1180	5650	1	0	0	...	0	98178	47.5112	-122.257	1340	5650	12.309982	5.236712	310612.755725	12.646302
1	6414100192	2014-12-09	538000.0	3	2	2570	7242	2	0	0	...	1991	98125	47.7210	-122.319	1690	7639	13.195614	5.343953	469455.770732	13.059329
2	5631500400	2015-02-25	180000.0	2	1	770	10000	1	0	0	...	0	98028	47.7379	-122.233	2720	8062	12.100712	5.454322	462480.035336	13.044359
3	2487200875	2014-12-09	604000.0	4	3	1960	5000	1	0	0	...	0	98136	47.5208	-122.393	1360	5000	13.311329	5.730630	551688.673004	13.220739
4	1954400510	2015-02-18	510000.0	3	2	1680	8080	1	0	0	...	0	98074	47.6168	-122.045	1800	7503	13.142166	5.715617	685605.775510	13.438058
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21608	263000018	2014-05-21	360000.0	3	2	1530	1131	3	0	0	...	0	98103	47.6993	-122.346	1530	1509	12.793859	5.460836	584919.210963	13.279229
21609	6600060120	2015-02-23	400000.0	4	2	2310	5813	2	0	0	...	0	98146	47.5107	-122.362	1830	7200	12.899220	5.154217	359483.239583	12.792423
21610	1523300141	2014-06-23	402101.0	2	0	1020	1350	2	0	0	...	0	98144	47.5944	-122.299	1020	2007	12.904459	5.976901	594547.650146	13.295556
21611	291310100	2015-01-16	400000.0	3	2	1600	2388	2	0	0	...	0	98027	47.5345	-122.069	1410	1287	12.899220	5.521461	616990.592233	13.332609
21612	1523300157	2014-10-15	325000.0	2	0	1020	1076	2	0	0	...	0	98144	47.5941	-122.299	1020	1357	12.691580	5.764023	594547.650146	13.295556

21613 rows × 25 columns

This code cleans up and changes column labels in the dataset using the `rename()` command, and then applies these changes using `inplace=True`.

Through this code, columns are given names that better reflect their content, making the data set more understandable and analysable.

This code uses the `isnull()` function to check for missing values in the data set, and then uses `sum()` to calculate the sum of the number of missing values in each column. The result shows the number of missing values in each column.

```
[11]
kchd.isnull().sum()
```

```
id                                          0
date                                        0
price                                       0
bedrooms                                    0
bathrooms                                   0
Area_of_living_room                         0
Land_area                                   0
floors                                      0
waterfront                                  0
view                                        0
condition                                   0
grade                                       0
Area_of_housing_space                       0
Area_of_housing_space_below_ground_level    0
Year_built                                  0
Year_renovated                              0
zipcode                                     0
Latitude                                    0
Longitude                                   0
Avg Living for closest 15 houeses           0
Avg Land for closest 15 houeses             0
dtype: int64
```

```
[12]
# Drop rows with missing values
kchd.dropna(inplace=True)
```

This code uses the `dropna()` function to delete rows containing missing values in the dataset. When you pass `inplace=True`, the data is updated directly inside the variable 'kchd' after deleting the rows.

This code uses the `duplicated()` function to find duplicate rows in the dataset, and then `sum()` counts the number of duplicate rows.

```
[13]
kchd.duplicated().sum()
```

```
0
```

```
# Convert date column to datetime format
kchd['date'] = pd.to_datetime(kchd['date'])
```

Convert the columns with a few unique values to categorical columns.
will not convert the following variables to categorical as they seems more like a continuous variable then a categorical variables:grade condition view.

```
# Convert selected columns to categorical:Convert the columns with a few unique values to categorical columns
cols_to_convert = ['floors', 'waterfront', 'view', 'condition', 'grade', 'zipcode']
kchd[cols_to_convert] = kchd[cols_to_convert].astype('category')
```

1. `cols_to_convert`: Specifies which columns will be processed and converted into categorical columns.

2. `kchd[cols_to_convert]`: This selects the selected columns in your DataFrame (the DataFrame is `kchd`).

3. `.astype('category')`: This function converts the selected columns into categorical columns.

4. `kchd[cols_to_convert] = ...`: Returns the converted values to the columns specified in the original DataFrame (`kchd`).

The code converts specific columns in a DataFrame into categorical columns, which helps improve performance and memory consumption when working with data, especially if these columns contain limited and repeated values.

```
kchd['bathrooms'] = kchd['bathrooms'].astype(int)
kchd['floors'] = kchd['floors'].astype(int)
```

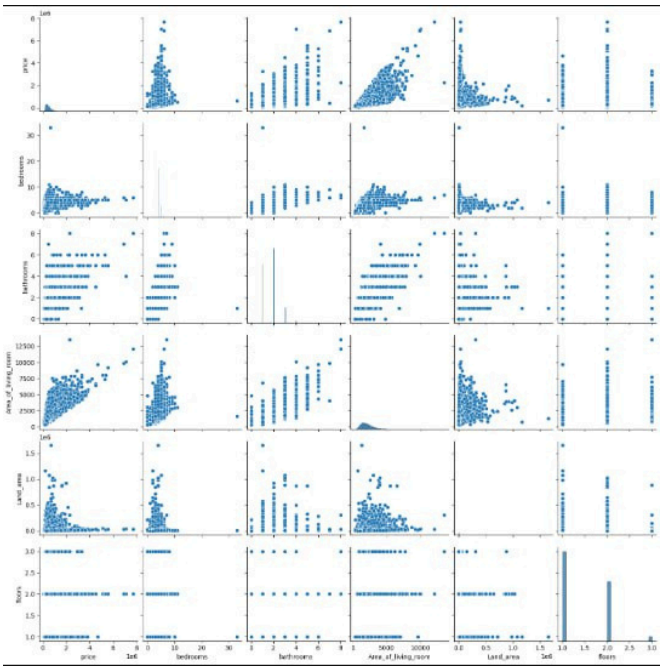
This code converts specific columns in a DataFrame to integer columns.

1. `kchd['bathrooms'] = kchd['bathrooms'].astype(int)`: This line converts the column values in the 'bathrooms' column to integers.

2. `kchd['floors'] = kchd['floors'].astype(int)`: This line performs the same operation for the 'floors' column.

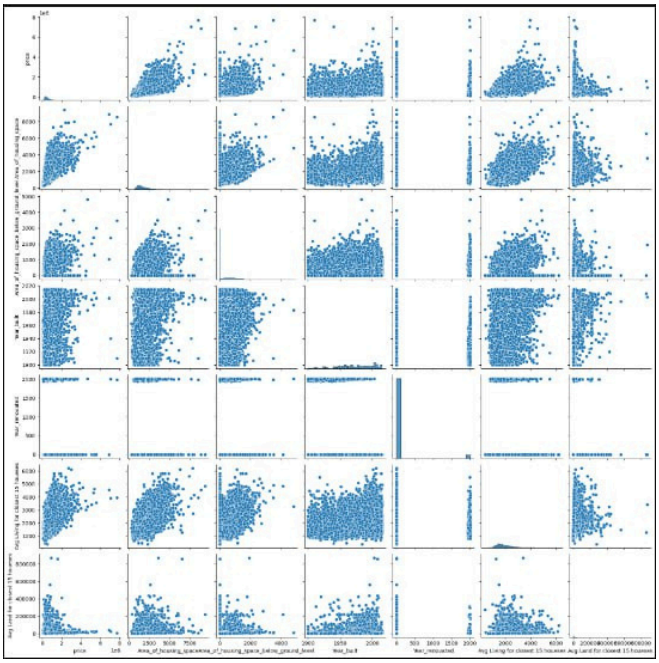
The reason behind this process is to convert values from decimal numbers (such as 1.5 floors) to integers.

```
vars1 = ['price', 'date', 'bedrooms', 'bathrooms', 'Area_of_living_room', 'Land_area', 'floors', 'waterfront', 'view']
sns.pairplot(kchd[vars1])
plt.show()
```

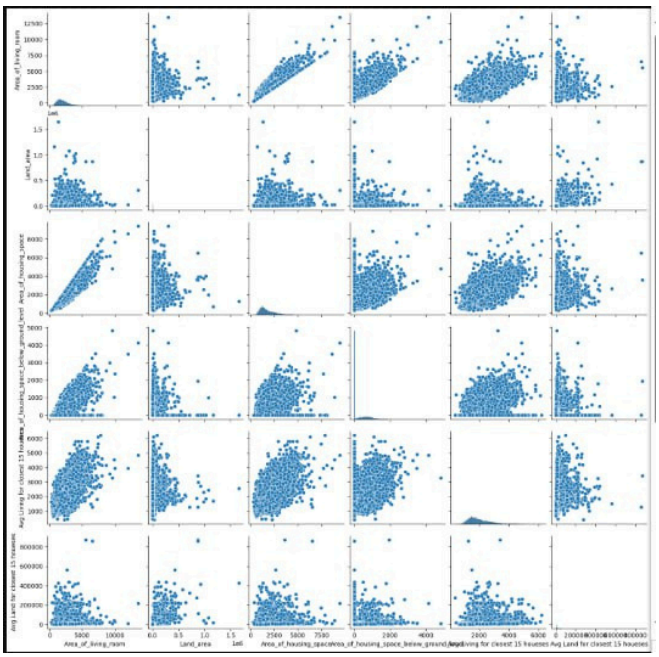


identify high level correlations among the various variables, particularly those with correlations with price. Since the features zipcode, latitude, and longitude do not work with scatter plot well, we will deal with them separately.

```
vars2 = ['price', 'condition', 'grade', 'Area_of_housing_space', 'Area_of_housing_space_below_ground_level', 'Year_built',
         'Year_renovated', 'Avg Living for closest 15 houeses', 'Avg Land for closest 15 houeses']
# Plot the second subset of variables
sns.pairplot(kchd[vars2])
plt.show()
```



```
# Select the variables for plotting before fit
vars_to_plot = ["Area_of_living_room", "Land_area", "Area_of_housing_space", "Area_of_housing_space_below_ground_level",
                "Avg Living for closest 15 houeses", "Avg Land for closest 15 houeses"]

# Create pairplot for selected variables
sns.pairplot(kchd[vars_to_plot])
plt.show()
```



Based on the scatter plots abovethere are some corrlations among the following variables:

Price and bathrooms Price and sqft_living Bedrooms and bathrooms Bedrooms and sqft_living Price and grade Price and sqft_above Price and sqft_living15 Grade and sqft_above Grade and sqft_basement Grade and sqft_living 15 sqft_above and sqft_basement

#there might be some correlations between all of the features related to sqft.

```python
vars_for_correlation = ["price", "bedrooms", "bathrooms", 'Area_of_living_room', "Land_area", "Year_built", "Year_renovated",
                        "Area_of_housing_space", "Area_of_housing_space_below_ground_level", "Avg Living for closest 15 houeses",
                        "Avg Land for closest 15 houeses"]
# Calculate correlation matrix
correlation_matrix = kchd[vars_for_correlation].corr()
# Display correlation matrix
print(correlation_matrix)
```

	price	bedrooms	bathrooms
price	1.000000	0.308350	0.510072
bedrooms	0.308350	1.000000	0.467452
bathrooms	0.510072	0.467452	1.000000
Area_of_living_room	0.702035	0.576671	0.697875
Land_area	0.089661	0.031703	0.085310
Year_built	0.054012	0.154178	0.433647
Year_renovated	0.126434	0.018841	0.060436
Area_of_housing_space	0.605567	0.477600	0.639933
Area_of_housing_space_below_ground_level	0.323816	0.303093	0.250880
Avg Living for closest 15 houeses	0.585379	0.391638	0.510049
Avg Land for closest 15 houeses	0.082447	0.029244	0.080779

	Area_of_living_room	Land_area
price	0.702035	0.089661
bedrooms	0.576671	0.031703
bathrooms	0.697875	0.085310
Area_of_living_room	1.000000	0.172826
Land_area	0.172826	1.000000
Year_built	0.318049	0.053080
Year_renovated	0.055363	0.007644
Area_of_housing_space	0.876597	0.183512
Area_of_housing_space_below_ground_level	0.435043	0.015286
Avg Living for closest 15 houeses	0.756420	0.144608
Avg Land for closest 15 houeses	0.183286	0.718557

	Year_built	Year_renovated
price	0.054012	0.126434
bedrooms	0.154178	0.018841
bathrooms	0.433647	0.060436
Area_of_living_room	0.318049	0.055363
Land_area	0.053080	0.007644
Year_built	1.000000	-0.224874
Year_renovated	-0.224874	1.000000
Area_of_housing_space	0.423898	0.023285
Area_of_housing_space_below_ground_level	-0.133124	0.071323
Avg Living for closest 15 houeses	0.326229	-0.002673
Avg Land for closest 15 houeses	0.070958	0.007854

	Area_of_housing_space
price	0.605567
bedrooms	0.477600
bathrooms	0.639933
Area_of_living_room	0.876597
Land_area	0.183512
Year_built	0.423898
Year_renovated	0.023285
Area_of_housing_space	1.000000
Area_of_housing_space_below_ground_level	-0.051943
Avg Living for closest 15 houeses	0.731870
Avg Land for closest 15 houeses	0.194050

	Area_of_housing_space_below_ground_level
price	0.323816
bedrooms	0.303093
bathrooms	0.250880
Area_of_living_room	0.435043
Land_area	0.015286
Year_built	-0.133124
Year_renovated	0.071323
Area_of_housing_space	-0.051943
Area_of_housing_space_below_ground_level	1.000000
Avg Living for closest 15 houeses	0.200355
Avg Land for closest 15 houeses	0.017276

	Avg Living for closest 15 houeses
price	0.585379
bedrooms	0.391638
bathrooms	0.510049
Area_of_living_room	0.756420
Land_area	0.144608
Year_built	0.326229
Year_renovated	-0.002673
Area_of_housing_space	0.731870
Area_of_housing_space_below_ground_level	0.200355
Avg Living for closest 15 houeses	1.000000
Avg Land for closest 15 houeses	0.183192

	Avg Land for closest 15 houeses
price	0.082447
bedrooms	0.029244
bathrooms	0.080779
Area_of_living_room	0.183286
Land_area	0.718557
Year_built	0.070958
Year_renovated	0.007854
Area_of_housing_space	0.194050
Area_of_housing_space_below_ground_level	0.017276
Avg Living for closest 15 houeses	0.183192
Avg Land for closest 15 houeses	1.000000

from correlation table the correlations:Price and bathrooms Price and sqft_living Price and sqft_above Price and sqft_living15 Bedrooms and bathrooms Bedrooms and

sqft_living Bathrooms and sqft_living Bathrooms and yr_built Bathrooms and sqft_above Bathrooms and sqft_living15 sqft_living and sqft_above sqft_living and sqft_living15 sqft_lot and sqft_lot15 sqft_above and sqft_living15

```
#it might help to predict price per square feet ('Area_of_living_room').

# Create additional variables
kchd['logPrice'] = np.log(kchd['price'])
kchd['LogPricePerSquareFeet'] = np.log(kchd['price'] / kchd['Area_of_living_room'])
```

```
#after adding "LogPricePerSquareFeet","LogPrice"
vars_for_correlation = ["price","logPrice","LogPricePerSquareFeet", "bedrooms", "bathrooms", 'Area_of_living_room', "Land_area", "Year_built",
                        "Year_renovated", "Area_of_housing_space", "Area_of_housing_space_below_ground_level", "Avg Living for closest 15 houeses",
                        "Avg Land for closest 15 houeses"]
# Calculate correlation matrix
correlation_matrix = kchd[vars_for_correlation].corr()
# Display correlation matrix
print(correlation_matrix)
```

	price	logPrice
price	1.000000	0.891654
logPrice	0.891654	1.000000
LogPricePerSquareFeet	0.531349	0.607880
bedrooms	0.308350	0.343561
bathrooms	0.510072	0.511119
Area_of_living_room	0.702035	0.695341
Land_area	0.089661	0.099622
Year_built	0.054012	0.080655
Year_renovated	0.126434	0.114498
Area_of_housing_space	0.605567	0.601801
Area_of_housing_space_below_ground_level	0.323816	0.316970
Avg Living for closest 15 houeses	0.585379	0.619312
Avg Land for closest 15 houeses	0.082447	0.091592

	LogPricePerSquareFeet	bedrooms
price	0.531349	0.308350
logPrice	0.607880	0.343561
LogPricePerSquareFeet	1.000000	-0.209736
bedrooms	-0.209736	1.000000
bathrooms	-0.046184	0.467452
Area_of_living_room	-0.099240	0.576671
Land_area	-0.027273	0.031703
Year_built	-0.268263	0.154178
Year_renovated	0.097781	0.018841
Area_of_housing_space	-0.092721	0.477600
Area_of_housing_space_below_ground_level	-0.032455	0.303093
Avg Living for closest 15 houeses	0.033686	0.391638
Avg Land for closest 15 houeses	-0.050926	0.029244

	bathrooms	Area_of_living_room
price	0.510072	0.702035
logPrice	0.511119	0.695341
LogPricePerSquareFeet	-0.046184	-0.099240
bedrooms	0.467452	0.576671
bathrooms	1.000000	0.697875
Area_of_living_room	0.697875	1.000000
Land_area	0.085310	0.172826
Year_built	0.433647	0.318049
Year_renovated	0.060436	0.055363
Area_of_housing_space	0.639933	0.876597
Area_of_housing_space_below_ground_level	0.250880	0.435043
Avg Living for closest 15 houeses	0.510049	0.756420
Avg Land for closest 15 houeses	0.080779	0.183286

	Area_of_housing_space_below_ground_level
price	0.323816
logPrice	0.316970
LogPricePerSquareFeet	-0.032455
bedrooms	0.303093
bathrooms	0.250880
Area_of_living_room	0.435043
Land_area	0.015286
Year_built	-0.133124
Year_renovated	0.071323
Area_of_housing_space	-0.051943
Area_of_housing_space_below_ground_level	1.000000
Avg Living for closest 15 houeses	0.200355
Avg Land for closest 15 houeses	0.017276

	Avg Living for closest 15 houeses
price	0.585379
logPrice	0.619312
LogPricePerSquareFeet	0.033686
bedrooms	0.391638
bathrooms	0.510049
Area_of_living_room	0.756420
Land_area	0.144608
Year_built	0.326229
Year_renovated	-0.002673
Area_of_housing_space	0.731870
Area_of_housing_space_below_ground_level	0.200355
Avg Living for closest 15 houeses	1.000000
Avg Land for closest 15 houeses	0.183192

	Avg Land for closest 15 houeses
price	0.082447
logPrice	0.091592
LogPricePerSquareFeet	-0.050926
bedrooms	0.029244
bathrooms	0.080779
Area_of_living_room	0.183286
Land_area	0.718557
Year_built	0.070958
Year_renovated	0.007854
Area_of_housing_space	0.194050
Area_of_housing_space_below_ground_level	0.017276
Avg Living for closest 15 houeses	0.183192
Avg Land for closest 15 houeses	1.000000

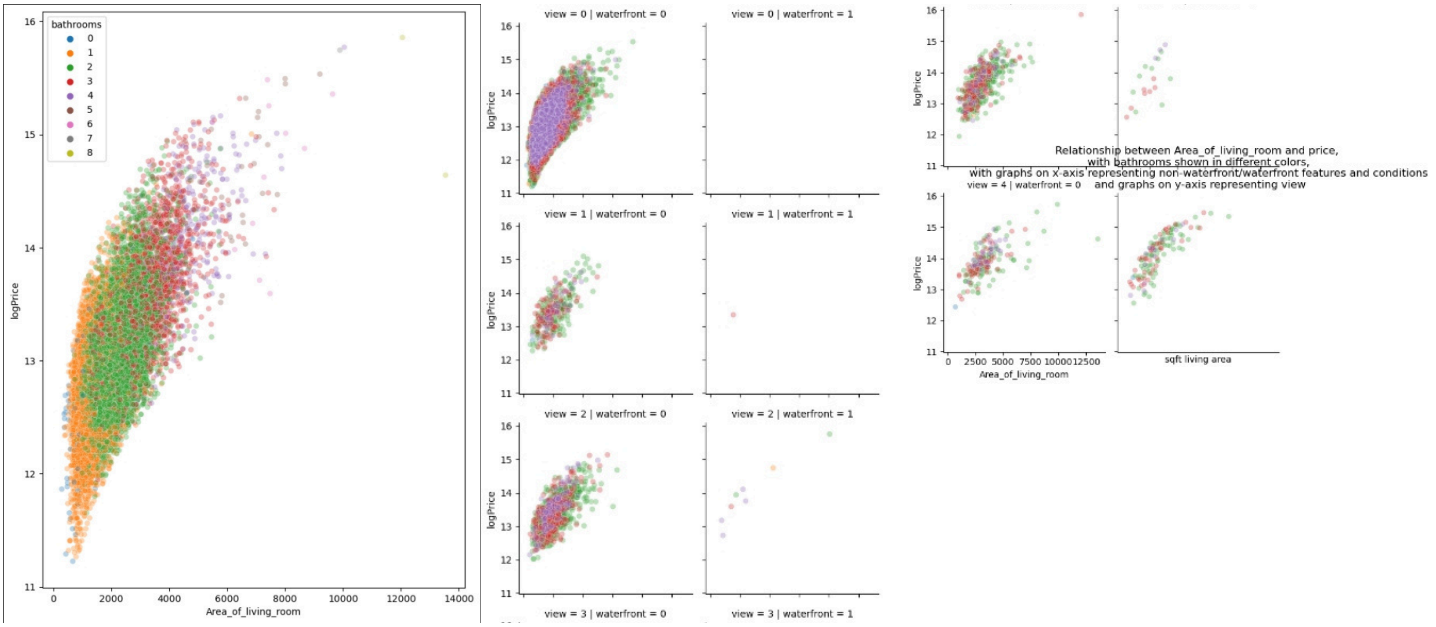
	Land_area	Year_built
price	0.089661	0.054012
logPrice	0.099622	0.080655
LogPricePerSquareFeet	-0.027273	-0.268263
bedrooms	0.031703	0.154178
bathrooms	0.085310	0.433647
Area_of_living_room	0.172826	0.318049
Land_area	1.000000	0.053080
Year_built	0.053080	1.000000
Year_renovated	0.007644	-0.224874
Area_of_housing_space	0.183512	0.423898
Area_of_housing_space_below_ground_level	0.015286	-0.133124
Avg Living for closest 15 houeses	0.144608	0.326229
Avg Land for closest 15 houeses	0.718557	0.070958

	Year_renovated
price	0.126434
logPrice	0.114498
LogPricePerSquareFeet	0.097781
bedrooms	0.018841
bathrooms	0.060436
Area_of_living_room	0.055363
Land_area	0.007644
Year_built	-0.224874
Year_renovated	1.000000
Area_of_housing_space	0.023285
Area_of_housing_space_below_ground_level	0.071323
Avg Living for closest 15 houeses	-0.002673
Avg Land for closest 15 houeses	0.007854

	Area_of_housing_space
price	0.605567
logPrice	0.601801
LogPricePerSquareFeet	-0.092721
bedrooms	0.477600
bathrooms	0.639933
Area_of_living_room	0.876597
Land_area	0.183512
Year_built	0.423898
Year_renovated	0.023285
Area_of_housing_space	1.000000
Area_of_housing_space_below_ground_level	-0.051943
Avg Living for closest 15 houeses	0.731870
Avg Land for closest 15 houeses	0.194050

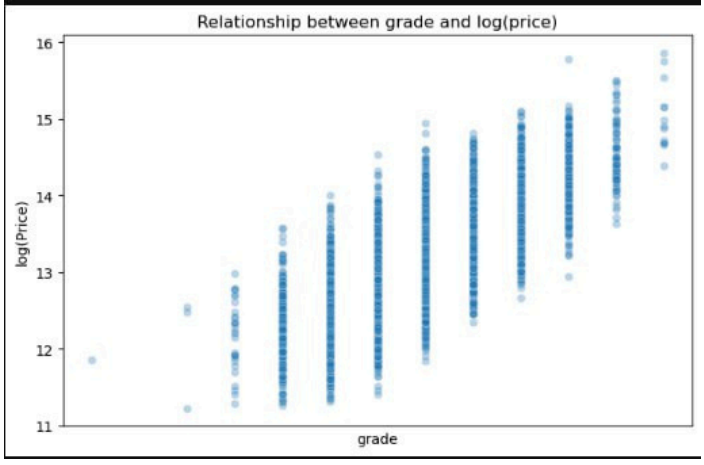
The correlation table indicates that the feature variables are (in general) better correlated to logPrice than the variable price. The correlation between log PricePerSquareFeet seems be worse than the correlation with price will see.

```python
# Set the figure size
plt.figure(figsize=(8, 11))
# Create the scatter plot
sns.scatterplot(data=kchd, x='Area_of_living_room', y='logPrice', hue='bathrooms', palette='tab10', alpha=0.3)
# Facet the plot
g = sns.FacetGrid(kchd, row='view', col='waterfront', hue='condition')
g.map(sns.scatterplot, 'Area_of_living_room', 'logPrice', alpha=0.3)
# Set labels and title
plt.xlabel('sqft living area')
plt.ylabel('log(Price)')
plt.title('Relationship between Area_of_living_room and price, \n with bathrooms shown in different colors, \n with graphs on x-axis representing non-waterfront/waterfront features and conditions \n and graphs on y-axis representing view')
# Hide x-axis ticks and labels
plt.tick_params(axis='x', which='both', bottom=False, labelbottom=False)
plt.show()
```



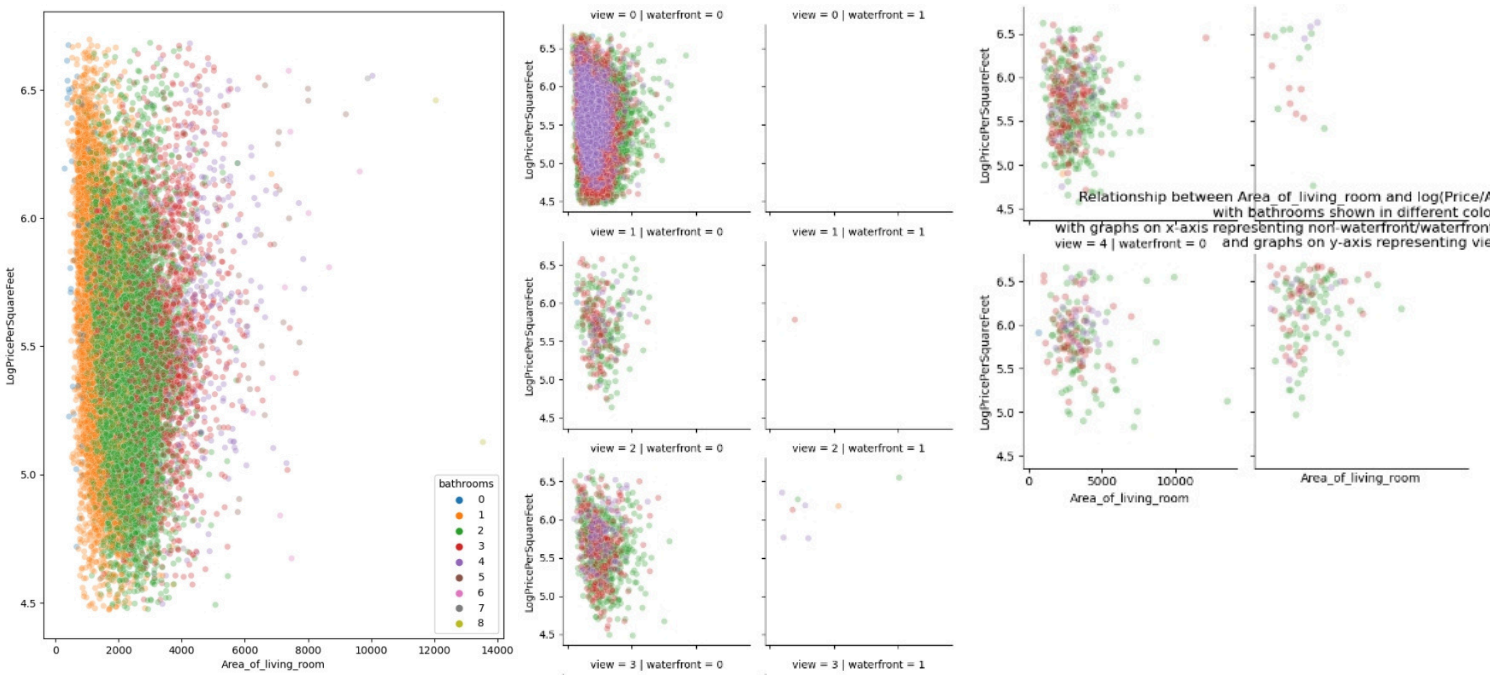
All waterfront homes have been viewed at least once, suggesting higher interest in these properties compared to those without waterfront access. Despite their desirable location, waterfront properties do not command significantly higher prices but exhibit pricing trends similar to non-waterfront properties. Among properties that haven't been viewed, those in better condition tend to have slightly higher prices. However, this association is not observed among viewed properties. There is a positive correlation between the square footage of living area and the logarithm of price, indicating that larger homes tend to be priced higher. The number of bathrooms positively correlates with price, suggesting that properties with more bathrooms tend to be more expensive. Properties that have been viewed more frequently tend to have higher logarithmic prices, indicating that popularity or visibility influences pricing.

```python
# Set the figure size
plt.figure(figsize=(8, 5))
# Create the scatter plot
sns.scatterplot(data=kchd, x='grade', y='logPrice', alpha=0.3)
# Set labels and title
plt.xlabel('grade')
plt.ylabel('log(Price)')
plt.title('Relationship between grade and log(price)')
# Hide x-axis ticks and labels
plt.tick_params(axis='x', which='both', bottom=False, labelbottom=False)

plt.show()
```



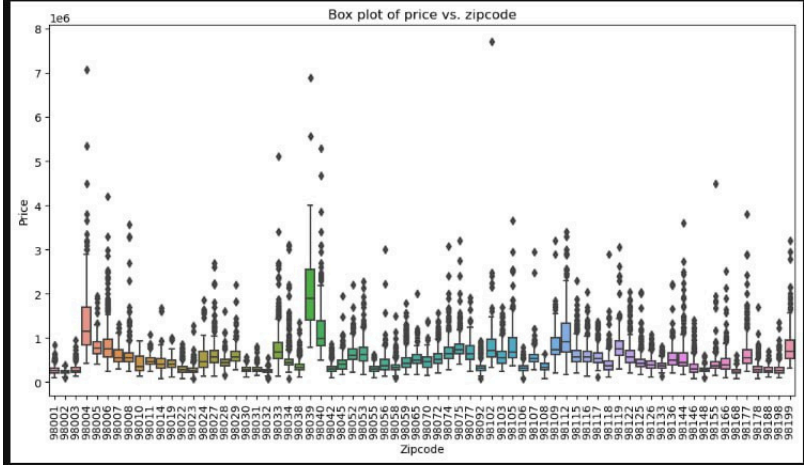
The graph above confirms that log(price) and grade have a positive correlation.

```
# Set the figure size
plt.figure(figsize=(8, 11))
# Create the scatter plot
sns.scatterplot(data=kchd, x='Area_of_living_room', y='LogPricePerSquareFeet', hue='bathrooms', palette='tab10', alpha=0.3)
# Facet the plot
g = sns.FacetGrid(kchd, row='view', col='waterfront', hue='condition')
g.map(sns.scatterplot, 'Area_of_living_room', 'LogPricePerSquareFeet', alpha=0.3)
# Set labels and title
plt.xlabel('Area_of_living_room')
plt.ylabel('log(Price/sqft_living)')
plt.title('Relationship between Area_of_living_room and log(Price/Area_of_living_room), \n with bathrooms shown in different colors, \n with graphs on x-axis representing non-waterfront/waterfront features and conditions \n and graphs on y-axis representing view')
# Hide x-axis ticks and labels
plt.tick_params(axis='x', which='both', bottom=False, labelbottom=False)

plt.show()
```



The graph doesn't suggest any strong connections between the price per square foot and the size of the living area. However, it does hint at a possible link between the price per square foot and the number of bedrooms. Therefore, we won't consider the price per square foot for further analysis.

```
#Analyze the impact of zip code
# Plot boxplot of price vs. zipcode
plt.figure(figsize=(12, 6))
sns.boxplot(x=kchd['zipcode'], y=kchd['price'])
plt.xlabel('Zipcode')
plt.ylabel('Price')
plt.title('Box plot of price vs. zipcode')
plt.xticks(rotation=90)
plt.show()
```



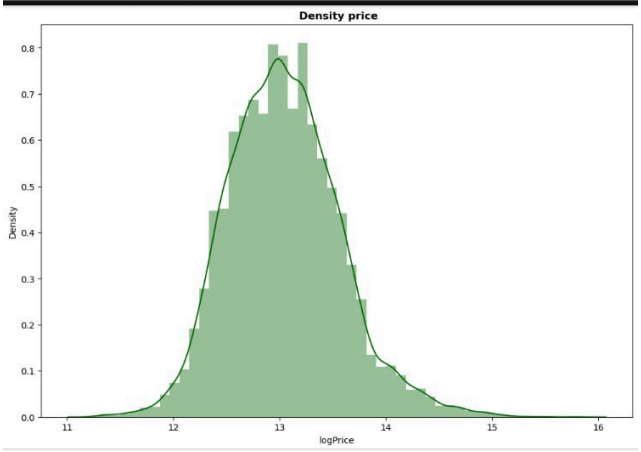
Displays the distinctive Zipcode

```
# Plot scatter plot of Latitude and Longitude
plt.figure(figsize=(12,8))
sns.scatterplot(x='Longitude',y='Latitude',data=kchd,hue='price')
```

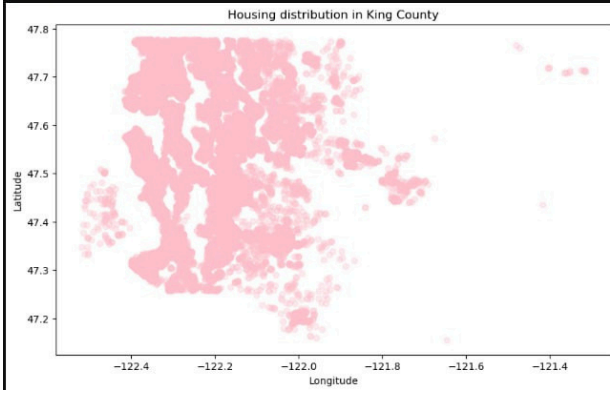
```
<Axes: xlabel='Longitude', ylabel='Latitude'>
```

The graph shows that home prices are strongly influenced by zip codes. Some zip codes have significantly lower housing prices than others. Any predictions about housing prices need to consider this variation in prices across different zip codes.

```
plt.figure(figsize=(12,8))
sns.distplot(kchd['logPrice'],color='darkgreen')
plt.title(' Density price', fontweight='bold')
```




```
# Plot scatter plot of Latitude and Longitude
plt.figure(figsize=(10, 6))
plt.scatter(kchd['Longitude'], kchd['Latitude'], alpha=0.3,color='pink')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Housing distribution in King County')
plt.show()
```



```
# Perform linear regression
X = kchd[['bathrooms', 'Area_of_living_room', 'Area_of_housing_space', 'Avg Living for closest 15 houeses', 'grade', 'view', 'condition']]
y = kchd['price']
```

```
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)

# Calculate R-squared
r_squared = r2_score(y, y_pred)
print("R-squared:", r_squared)
```

```
R-squared: 0.5827249745180376
```

R-squared value is very low at 0.583

```
# Get the coefficients (θ)
coefficients = model.coef_
intercept = model.intercept_
print("Coefficients:", coefficients)
print("Intercept:", intercept)
```

```
Coefficients: [ 9.67426968e+03  1.73195626e+02 -2.74995111e+01  8.23098967e+00
  1.02142241e+05  9.10725479e+04  5.32811321e+04]
Intercept: -789327.2772103335
```

```
# Calculate mean squared error (MSE), sum of squared errors (SSE), and total sum of squares (SST)
mse = mean_squared_error(y, y_pred)
sse = np.sum((y - y_pred) ** 2)
sst = np.sum((y - np.mean(y)) ** 2)
print("Mean Squared Error (MSE):", mse)
print('')

print("Sum of Squared Errors (SSE):", sse)
print('')

print("Total Sum of Squares (SST):", sst)
```

```
Mean Squared Error (MSE): 56238718181.53637

Sum of Squared Errors (SSE): 1215487416057545.5

Total Sum of Squares (SST): 2912916761921299.5
```

The R-squared ups to 0.79 from 0.583 if we use the mean price in the zipcode

Calculate mean squared error (MSE), sum of squared errors (SSE), and total sum of squares (SST)

```
# Create new variable: average price per zipcode
kchd['avgZipCodePrice'] = kchd.groupby('zipcode')['price'].transform('mean')
```

```
# Perform linear regression to predict log(price)
X_log = kchd[['bathrooms', 'Area_of_living_room', 'grade', 'view', 'condition', 'avgZipCodePrice']]
y_log = np.log(kchd['price'])
```

```
model_log = LinearRegression()
model_log.fit(X_log, y_log)
y_pred_log = model_log.predict(X_log)
```

```
# Calculate R-squared for log(price) prediction
r_squared_log = r2_score(y_log, y_pred_log)
print("R-squared for log(price):", r_squared_log)
```

```
R-squared for log(price): 0.7916165504596662
```

We have to be precise.

```
# Create the 'log(avgZipCodePrice)' column
kchd['log_avgZipCodePrice'] = np.log(kchd['avgZipCodePrice'])

# Remove unnecessary features and repeat linear regression for log(price)
X_log_simplified = kchd[['Area_of_living_room', 'grade', 'view', 'log_avgZipCodePrice']]
y_log = np.log(kchd['price'])
```

```
model_log_simplified = LinearRegression()
model_log_simplified.fit(X_log_simplified, y_log)
y_pred_log_simplified = model_log_simplified.predict(X_log_simplified)
```


```
model_log_simplified = LinearRegression()
model_log_simplified.fit(X_log_simplified, y_log)
y_pred_log_simplified = model_log_simplified.predict(X_log_simplified)
```


```
# Calculate R-squared for simplified log(price) prediction
r_squared_log_simplified = r2_score(y_log, y_pred_log_simplified)
print("R-squared for simplified log(price):", r_squared_log_simplified)
```

```
R-squared for simplified log(price): 0.8242190892159609
```

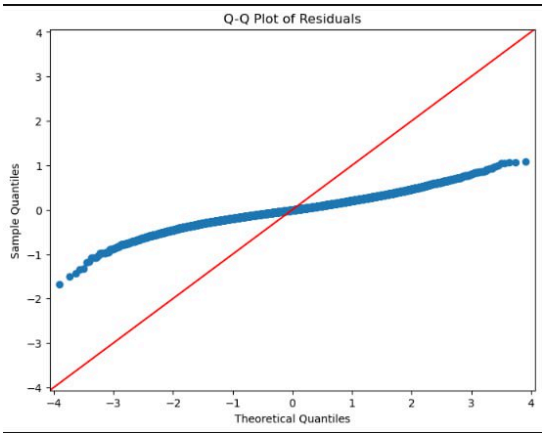
The R-squared jumps up to 0.828 from 0.79 if we predict log(avgZipCodePrice)

```
# Get the coefficients (θ) and intercept
coefficients = model_log_simplified.coef_
intercept = model_log_simplified.intercept_
print("Coefficients:", coefficients)
print('')
print("Intercept:", intercept)
print('')

# Calculate mean squared error (MSE), sum of squared errors (SSE), and total sum of squares (SST)
mse = mean_squared_error(y_log, y_pred_log_simplified)
sse = np.sum((y_log - y_pred_log_simplified) ** 2)
sst = np.sum((y_log - np.mean(y_log)) ** 2)

print("Mean Squared Error (MSE):", mse)
print('')
print("Sum of Squared Errors (SSE):", sse)
print('')
print("Total Sum of Squares (SST):", sst)
print('')
```

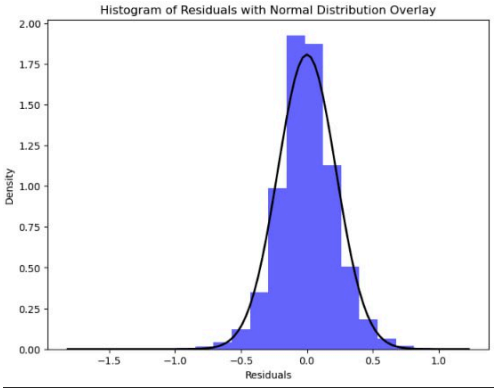
```
Coefficients: [1.99295979e-04 9.10298165e-02 9.74886621e-02 7.16784681e-01]

Intercept: 2.510201543967243

Mean Squared Error (MSE): 0.048758767838872154

Sum of Squared Errors (SSE): 1053.823249301544

Total Sum of Squares (SST): 5995.094942910211
```

Calculate mean squared error (MSE), sum of squared errors (SSE), and total sum of squares (SST)

```python
# Assuming you have your data in a pandas DataFrame df
X = X_log_simplified
y = y_log

# Fit the linear regression model
model = sm.OLS(y, sm.add_constant(X)).fit()

# Calculate residuals
residuals = model.resid

# Q-Q plot
fig, ax = plt.subplots(figsize=(8, 6))
sm.qqplot(residuals, line='45', ax=ax)
ax.set_title('Q-Q Plot of Residuals')
```

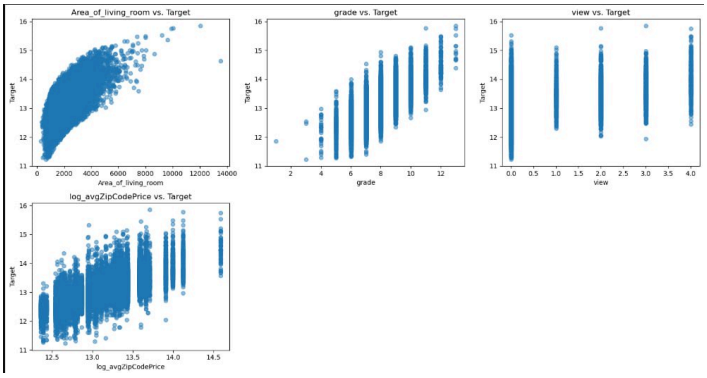



```python
# Histogram
fig, ax = plt.subplots(figsize=(8, 6))
ax.hist(residuals, bins=20, density=True, alpha=0.6, color='b')
# Overlay a normal distribution curve
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = stats.norm.pdf(x, residuals.mean(), residuals.std())
ax.plot(x, p, 'k', linewidth=2)
ax.set_title('Histogram of Residuals with Normal Distribution Overlay')
ax.set_xlabel('Residuals')
ax.set_ylabel('Density')

# Show plots
plt.show()
```



our line is in middle where
(we expected on average that there is a linear relationship + some errors)
the residual(error) take the shape of normal distribution

```python
X = X_log_simplified
y = y_log

num_cols = X.shape[1]  # Number of independent variables

# Set the number of columns and rows for subplots
cols_per_row = 3
rows = (num_cols - 1) // cols_per_row + 1

# Create a figure and subplots
fig, axes = plt.subplots(rows, cols_per_row, figsize=(15, 4 * rows))
axes = axes.flatten()

# Iterate over each independent variable
for i, (col, ax) in enumerate(zip(X.columns, axes)):
    # Scatter plot
    ax.scatter(X[col], y, alpha=0.5)
    ax.set_title(f'{col} vs. Target')
    ax.set_xlabel(col)
    ax.set_ylabel('Target')

# Remove any unused subplots
for ax in axes[num_cols:]:
    ax.remove()

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



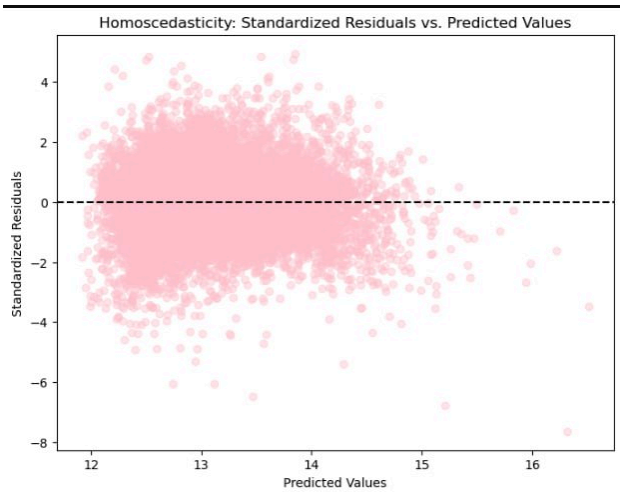
-when plotting a figures between our target and feature we get that (there is a linear reggression between each feature&the target)

```python
# Assuming you have your data in a pandas DataFrame df
X =X_log_simplified

correlation_matrix = X.corr()

# Display the correlation matrix
print("Pearson's Correlation Matrix:")
print(correlation_matrix)
```

```
Pearson's Correlation Matrix:
                     Area_of_living_room     grade      view
Area_of_living_room             1.000000  0.762704  0.284611
grade                           0.762704  1.000000  0.251321
view                            0.284611  0.251321  1.000000
log_avgZipCodePrice             0.283582  0.371502  0.101741

                     log_avgZipCodePrice
Area_of_living_room             0.283582
grade                           0.371502
view                            0.101741
log_avgZipCodePrice             1.000000
```

Compute Pearson's correlation matrix

```python
# Assuming you have your data in a pandas DataFrame df
X = X_log_simplified
y =y_log

# Fit the linear regression model
model = sm.OLS(y, sm.add_constant(X)).fit()

# Get the predicted values and standardized residuals
predicted_values = model.predict()
standardized_residuals = model.get_influence().resid_studentized_internal

# Create a scatter plot of standardized residuals vs. predicted values
plt.figure(figsize=(8, 6))
plt.scatter(predicted_values, standardized_residuals, alpha=0.4,color='pink')
plt.axhline(y=0, color='black', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Standardized Residuals')
plt.title('Homoscedasticity: Standardized Residuals vs. Predicted Values')
plt.show()
```



"The homoscedasticity assumes that the variance of the residual errors is similar across the value of each independent variable.
One way of checking that is through a plot of
the predicted values against the standardized residual values
to see if the points are equally distributed across all the values of the independent variables."

```python
# mse before fit
print("Mean Squared Error (MSE) for this model:", mean_squared_error(y, y_pred))
```

```
Mean Squared Error (MSE) for this model: 370218305667.1767
```

```python
#second accuracy
model = ols('price ~ Area_of_living_room + grade + view + avgZipCodePrice', data=kchd).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(tabulate(anova_table, headers=["sum_sq" , "df" ,"F", "PR(>F)"], tablefmt="grid"))
```

	sum_sq	df	F	PR(>F)
grade	1.24438e+14	11	392.84	0
view	1.14955e+14	4	997.981	0
Area_of_living_room	1.37464e+14	1	4773.57	0
avgZipCodePrice	4.82447e+14	1	16753.5	0
Residual	6.21867e+14	21595	nan	nan

Displays the Anova table for the second accuracy : 0.75

```
#third accuracy
model = ols('logPrice ~ Area_of_living_room + grade + view + log_avgZipCodePrice', data=kchd).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(tabulate(anova_table, headers=["sum_sq" , "df" ,"F", "PR(>F)"], tablefmt="grid"))
```

```
+----------------------+----------+-------+-----------+----------+
|                      |   sum_sq |    df |         F |   PR(>F) |
+======================+==========+=======+===========+==========+
| grade                |  103.837 |    11 |   196.171 |        0 |
+----------------------+----------+-------+-----------+----------+
| view                 |  122.592 |     4 |   636.907 |        0 |
+----------------------+----------+-------+-----------+----------+
| Area_of_living_room  |  297.22  |     1 |  6176.67  |        0 |
+----------------------+----------+-------+-----------+----------+
| log_avgZipCodePrice  | 1488.67  |     1 | 30936.7   |        0 |
+----------------------+----------+-------+-----------+----------+
| Residual             | 1039.15  | 21595 |   nan     |      nan |
+----------------------+----------+-------+-----------+----------+
```

Displays the Anova table for the third accuracy : 0.82

```
fter fit
Mean Squared Error (MSE) for this model:", mean_squared_error(y_log, y_pred_log_simpl
```

```
uared Error (MSE) for this model: 0.048758767838872154
```

decrease for mean square error from 370218305667.2 into 0.0488

```
# Using statsmodels
X = sm.add_constant(x)
model = sm.OLS(y, X).fit()
predictions_statsmodels = model.predict(X)
summary = model.summary()
print(summary)
```

```
# partial f_ test (second accuracy)
x= kchd[['Area_of_living_room', 'grade', 'view', 'avgZipCodePrice']]
y = kchd['price']
# Using sklearn
regression = linear_model.LinearRegression()
regression.fit(x, y)
predictions_sklearn = regression.predict(x)
print("Intercept: \n", regression.intercept_)
print('')
print("Coefficients: \n", regression.coef_)
```

```
Intercept:
 -540723.1364849595

Coefficients:
 [1.60154692e+02 4.55780440e+04 9.68774173e+04 6.96222588e-01]
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.743
Model:                            OLS   Adj. R-squared:                  0.743
Method:                 Least Squares   F-statistic:                 1.559e+04
Date:                Mon, 13 May 2024   Prob (F-statistic):               0.00
Time:                        16:45:57   Log-Likelihood:            -2.9294e+05
No. Observations:               21613   AIC:                         5.859e+05
Df Residuals:                   21608   BIC:                         5.859e+05
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -5.407e+05   9935.150    -54.425      0.000   -5.6e+05   -5.21e+05
Area_of_living_room   160.1547      2.158     74.232      0.000     155.926     164.384
grade       4.558e+04   1716.335     26.555      0.000    4.22e+04    4.89e+04
view        9.688e+04   1727.325     56.085      0.000    9.35e+04       1e+05
avgZipCodePrice     0.6962      0.006    119.510      0.000       0.685       0.708
==============================================================================
Omnibus:                    21949.878   Durbin-Watson:                   1.980
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          4187780.549
Skew:                           4.606   Prob(JB):                         0.00
Kurtosis:                      70.568   Cond. No.                     4.68e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.68e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Displays the partial f-test for the second accuracy : 0.75

```python
# Using statsmodels
x = sm.add_constant(x)
model = sm.OLS(y, x).fit()
predictions_statsmodels = model.predict(x)
summary = model.summary()
print(summary)
```

```python
# partial f_ test (third accuracy)
x= kchd[['Area_of_living_room', 'grade', 'view', 'log_avgZipCodePrice']]
y = kchd['logPrice']
# Using sklearn
regression = linear_model.LinearRegression()
regression.fit(x, y)
predictions_sklearn = regression.predict(x)
print("Intercept: \n", regression.intercept_)
print('')
print("Coefficients: \n", regression.coef_)
```

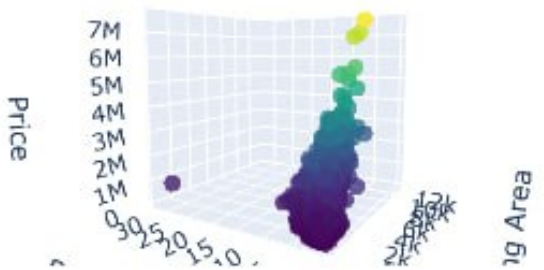
```
Intercept:
 2.510201543967243

Coefficients:
 [1.99295979e-04 9.10298165e-02 9.74886621e-02 7.16784681e-01]
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 logPrice   R-squared:                       0.824
Model:                            OLS   Adj. R-squared:                  0.824
Method:                 Least Squares   F-statistic:                 2.533e+04
Date:                Mon, 13 May 2024   Prob (F-statistic):               0.00
Time:                        16:45:57   Log-Likelihood:                 1977.5
No. Observations:               21613   AIC:                            -3945.
Df Residuals:                   21608   BIC:                            -3905.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.5102      0.051     49.030      0.000       2.410       2.611
Area_of_living_room     0.0002   2.56e-06     77.932      0.000       0.000       0.000
grade          0.0910      0.002     44.543      0.000       0.087       0.095
view           0.0975      0.002     47.600      0.000       0.093       0.102
log_avgZipCodePrice     0.7168      0.004    175.023      0.000       0.709       0.725
==============================================================================
Omnibus:                      848.418   Durbin-Watson:                   1.995
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2660.828
Skew:                          -0.048   Prob(JB):                         0.00
Kurtosis:                       4.716   Cond. No.                     7.77e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.77e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Displays the partial f-test for the third accuracy : 0.82

Living Area vs Bedrooms vs Price

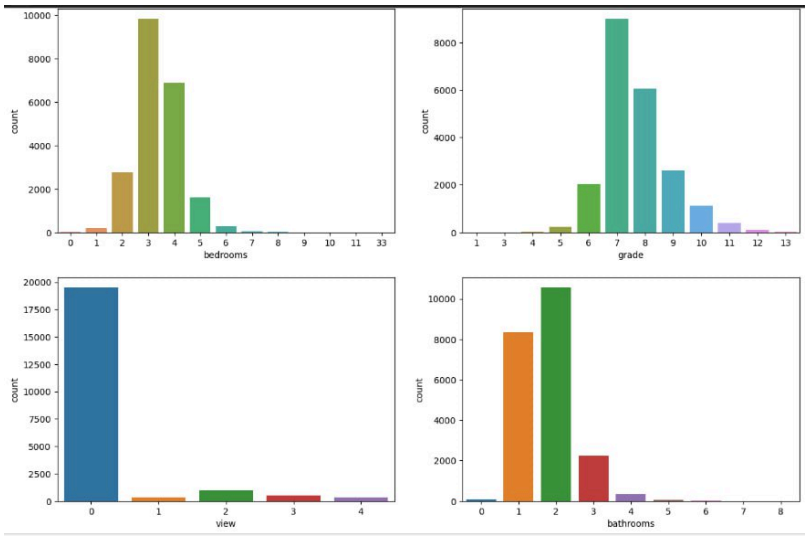
```
import plotly.graph_objects as go
fig2 = go.Figure(data=[go.Scatter3d(
    x=kchd['Area_of_living_room'],
    y=kchd['bedrooms'],
    z=kchd['price'],
    mode='markers',
    marker=dict(
        size=5,
        color=kchd['price'],
        colorscale='Viridis',
        opacity=0.8 ))])

fig2.update_layout(scene=dict(
                    xaxis_title='Living Area',
                    yaxis_title='Bedrooms',
                    zaxis_title='Price'),
                    title='Living Area vs Bedrooms vs Price')
fig2.show()
```



3D graph (showing relationship between Living Area , Bedrooms and Price)

```
fig, ax = plt.subplots(2, 2, figsize=(15, 10))
sns.countplot(x='bedrooms', data=kchd, ax=ax[0, 0])
sns.countplot(x='bathrooms', data=kchd, ax=ax[1, 1])
sns.countplot(x='view', data=kchd, ax=ax[1, 0])
sns.countplot(x='grade', data=kchd, ax=ax[0, 1])
plt.show()
```

frequency for (bedrooms - bathrooms - view - grade)



GUI

```
def countplot() :
    fig, ax = plt.subplots(2, 2, figsize=(15, 10))
    sns.countplot(x='bedrooms', data=kchd, ax=ax[0, 0])
    sns.countplot(x='bathrooms', data=kchd, ax=ax[1, 1])
    sns.countplot(x='view', data=kchd, ax=ax[1, 0])
    sns.countplot(x='grade', data=kchd, ax=ax[0, 1])
    plt.show()

def plot_boxplot():
    # Plot boxplot of price vs. zipcode
    plt.figure(figsize=(10, 6))
    sns.boxplot(x=kchd['zipcode'], y=kchd['price'])
    plt.xlabel('Zipcode')
    plt.ylabel('Price')
    plt.title('Box plot of price vs. zipcode')
    plt.xticks(rotation=90)
    plt.show()

def plot_scatter():
    # Plot scatter plot of latitude and longitude
    plt.figure(figsize=(10, 6))
    plt.scatter(kchd['long'], kchd['lat'], alpha=0.3)
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.title('Housing distribution in King County')
    plt.show()

def d_graph() :
    fig2 = go.Figure(data=[go.Scatter3d(
    x=kchd['sqft_living'],
    y=kchd['bedrooms'],
    z=kchd['price'],
    mode='markers',
    marker=dict(
        size=5,
        color=kchd['price'],
        colorscale='Viridis',
        opacity=0.8
    )
    )])

    fig2.update_layout(scene=dict(
                    xaxis_title='Living Area',
                    yaxis_title='Bedrooms',
                    zaxis_title='Price'),
                    title='Living Area vs Bedrooms vs Price')
    fig2.show()

def qqplot_histo():
    X = X_log_simplified
    y = y_log

    # Fit the linear regression model
    model = sm.OLS(y, sm.add_constant(X)).fit()

   # Calculate residuals
    residuals = model.resid

   # Q-Q plot
    fig, ax = plt.subplots(figsize=(8, 6))
    sm.qqplot(residuals, line='45', ax=ax)
    ax.set_title('Q-Q Plot of Residuals')

   # Histogram
    fig, ax = plt.subplots(figsize=(8, 6))
    ax.hist(residuals, bins=20, density=True, alpha=0.6, color='b')
# Overlay a normal distribution curve
    xmin, xmax = plt.xlim()
    x = np.linspace(xmin, xmax, 100)
    p = stats.norm.pdf(x, residuals.mean(), residuals.std())
    ax.plot(x, p, 'k', linewidth=2)
    ax.set_title('Histogram of Residuals with Normal Distribution Overlay')
    ax.set_xlabel('Residuals')
    ax.set_ylabel('Density')

    # Show plots
    plt.show()
```

```python
def scatter_plot():
    X = X_log_simplified
    y =y_log

    # Fit the linear regression model
    model = sm.OLS(y, sm.add_constant(X)).fit()

    # Get the predicted values and standardized residuals
    predicted_values = model.predict()
    standardized_residuals = model.get_influence().resid_studentized_internal

    # Create a scatter plot of standardized residuals vs. predicted values
    plt.figure(figsize=(8, 6))
    plt.scatter(predicted_values, standardized_residuals, alpha=0.4,color='pink')
    plt.axhline(y=0, color='black', linestyle='--')
    plt.xlabel('Predicted Values')
    plt.ylabel('Standardized Residuals')
    plt.title('Homoscedasticity: Standardized Residuals vs. Predicted Values')
    plt.show()

def scatter():
    plt.figure(figsize=(8, 5))

    # Create the scatter plot
    sns.scatterplot(data=kchd, x='grade', y='logPrice', alpha=0.3)

    # Set labels and title
    plt.xlabel('grade')
    plt.ylabel('log(Price)')
    plt.title('Relationship between grade and log(price)')

    # Hide x-axis ticks and labels
    plt.tick_params(axis='x', which='both', bottom=False, labelbottom=False)

    plt.show()


def scatt():
    plt.figure(figsize=(12,8))
    sns.scatterplot(x='long',y='lat',data=kchd,hue='price')
    plt.show()

def target():
    X = X_log_simplified
    y = y_log

    num_cols = X.shape[1]  # Number of independent variables

    # Set the number of columns and rows for subplots
    cols_per_row = 3
    rows = (num_cols - 1) // cols_per_row + 1

    # Create a figure and subplots
    fig, axes = plt.subplots(rows, cols_per_row, figsize=(15, 4 * rows))
    axes = axes.flatten()

    # Iterate over each independent variable
    for i, (col, ax) in enumerate(zip(X.columns, axes)):
        # Scatter plot
        ax.scatter(X[col], y, alpha=0.5)
        ax.set_title(f'{col} vs. Target')
        ax.set_xlabel(col)
        ax.set_ylabel('Target')

    # Remove any unused subplots
    for ax in axes[num_cols:]:
        ax.remove()

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

```python
def density():
    plt.figure(figsize=(12,8))
    sns.distplot(kchd['logPrice'],color='darkgreen')
    plt.title(' Density price', fontweight='bold')
    plt.show()

def pairplot_1():
    vars1 = ['price', 'date', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view']


    sns.pairplot(kchd[vars1])
    plt.show()

def pairplot_2():
    vars2 = ['price', 'condition', 'grade', 'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'sqft_living15', 'sqft_lot15']

    # Plot the second subset of variables
    sns.pairplot(kchd[vars2])
    plt.show()

def pairplot_3():
    # Select the variables for plotting
    vars_to_plot = ["sqft_living", "sqft_lot", "sqft_above", "sqft_basement", "sqft_living15", "sqft_lot15"]

    # Create pairplot for selected variables
    sns.pairplot(kchd[vars_to_plot])
    plt.show()


def perform_linear_regression():
    # Perform linear regression
    X = kchd[['bathrooms', 'sqft_living', 'sqft_above', 'sqft_living15', 'grade', 'view', 'condition']]
    y = kchd['price']
    model = LinearRegression()
    model.fit(X, y)
    y_pred = model.predict(X)

    # Calculate R-squared
    r_squared = r2_score(y, y_pred)
    messagebox.showinfo("Linear Regression", f"R-squared: {r_squared}")
```

```python
def calculate_second_accuracy():
    # Perform OLS regression
    model = ols('price ~ sqft_living + grade + view + avgZipCodePrice', data=kchd).fit()
    # Compute ANOVA table
    anova_table = sm.stats.anova_lm(model, typ=2)
    # Display ANOVA table in a messagebox
    result_text = tabulate(anova_table, headers=["sum_sq", "df", "F", "PR(>F)"], tablefmt="grid")
    messagebox.showinfo("Second Accuracy Result", result_text)

def calculate_third_accuracy():
    # Perform OLS regression
    model = ols('logPrice ~ sqft_living + grade + view + log_avgZipCodePrice', data=kchd).fit()
    # Compute ANOVA table
    anova_table = sm.stats.anova_lm(model, typ=2)
    # Display ANOVA table in a messagebox
    result_text = tabulate(anova_table, headers=["sum_sq", "df", "F", "PR(>F)"], tablefmt="grid")
    messagebox.showinfo("Third Accuracy Result", result_text)

def calculate_mse_before_fit():
    # Perform linear regression
    X = kchd[['bathrooms', 'sqft_living', 'sqft_above', 'sqft_living15', 'grade', 'view', 'condition']]
    y = kchd['price']
    model = LinearRegression()
    model.fit(X, y)
    y_pred = model.predict(X)

    # Calculate mean squared error
    mse = mean_squared_error(y, y_pred)
    messagebox.showinfo("Mean Squared Error (MSE) before Fit", f"MSE: {mse}")

def calculate_mse_after_fit():
    # Perform linear regression
    X_log_simplified = kchd[['sqft_living', 'grade', 'view', 'log_avgZipCodePrice']]
    y_log = kchd['logPrice']
    model_log_simplified = LinearRegression()
    model_log_simplified.fit(X_log_simplified, y_log)
    y_pred_log_simplified = model_log_simplified.predict(X_log_simplified)

    # Calculate mean squared error
    mse = mean_squared_error(y_log, y_pred_log_simplified)
    messagebox.showinfo("Mean Squared Error (MSE) after Fit", f"MSE: {mse}")
```

```python
def partial_f_test_second_accuracy():
    # Perform linear regression
    x = kchd[['sqft_living', 'grade', 'view', 'avgZipCodePrice']]
    y = kchd['price']
    regression = LinearRegression()
    regression.fit(x, y)

    # Using statsmodels
    x = sm.add_constant(x)
    model = sm.OLS(y, x).fit()
    summary = model.summary()

    # Display summary in a messagebox
    messagebox.showinfo("Partial F-Test (Second Accuracy)", summary)

def partial_f_test_third_accuracy():
    # Perform linear regression
    x = kchd[['sqft_living', 'grade', 'view', 'log_avgZipCodePrice']]
    y = kchd['logPrice']

    # Using sklearn
    regression = linear_model.LinearRegression()
    regression.fit(x, y)
    predictions_sklearn = regression.predict(x)

    # Using statsmodels
    x = sm.add_constant(x)
    model = sm.OLS(y, x).fit()
    summary = model.summary()

    # Display summary in a messagebox
    messagebox.showinfo("Partial F-Test (Third Accuracy)", summary)
```

```python
root = tk.Tk()
root.title("Housing Data Analysis")

# Define colors for the theme
background_color = "#000"  # black
button_color = "#000"  # black
button_hover_color = "#000"  # black
button_text_color = "purple"

# Configure root window
root.configure(bg=background_color)

# Configure style for buttons
style = ttk.Style(root)
style.configure('Custom.TButton', background=button_color, foreground=button_text_color)
style.map('Custom.TButton', background=[('active', button_hover_color)])

# Grid layout for buttons
row = 0

correlation_button = ttk.Button(root, text="Correlation Average Matrix", command=calculate_average_correlation_matrix,
                                style='Custom.TButton')
correlation_button.grid(row=row, column=0, padx=10, pady=5)

correlation_button = ttk.Button(root, text="Correlation Average Matrix update", command=calculate_average_correlation_matrix_2,
                                style='Custom.TButton')
correlation_button.grid(row=row, column=1, padx=10, pady=5)



create_variables_button = ttk.Button(root, text="Create Additional Variables", command=create_additional_variables,
                                     style='Custom.TButton')
create_variables_button.grid(row=row, column=2, padx=10, pady=5)

boxplot_button = ttk.Button(root, text="Plot Boxplot of Price vs. Zipcode", command=plot_boxplot,
                            style='Custom.TButton')
boxplot_button.grid(row=row, column=3, padx=10, pady=5)
```

```python
row += 1

scatter_button = ttk.Button(root, text="Plot Scatter Plot of Latitude and Longitude", command=plot_scatter,
                            style='Custom.TButton')
scatter_button.grid(row=row, column=0, padx=10, pady=5)

linear_regression_button = ttk.Button(root, text="Perform Linear Regression", command=perform_linear_regression,
                                      style='Custom.TButton')
linear_regression_button.grid(row=row, column=1, padx=10, pady=5)

second_accuracy_button = ttk.Button(root, text="Calculate Second Accuracy", command=calculate_second_accuracy,
                                    style='Custom.TButton')
second_accuracy_button.grid(row=row, column=2, padx=10, pady=5)

scatter_button = ttk.Button(root, text="multi graph", command=countplot,
                            style='Custom.TButton')
scatter_button.grid(row=row, column=3, padx=10, pady=5)


row += 1

third_accuracy_button = ttk.Button(root, text="Calculate Third Accuracy", command=calculate_third_accuracy,
                                   style='Custom.TButton')
third_accuracy_button.grid(row=row, column=0, padx=10, pady=5)

mse_before_fit_button = ttk.Button(root, text="Calculate MSE before Fit", command=calculate_mse_before_fit,
                                   style='Custom.TButton')
mse_before_fit_button.grid(row=row, column=1, padx=10, pady=5)

mse_after_fit_button = ttk.Button(root, text="Calculate MSE after Fit", command=calculate_mse_after_fit,
                                  style='Custom.TButton')
mse_after_fit_button.grid(row=row, column=2, padx=10, pady=5)

scatter_button = ttk.Button(root, text="3D graph", command=d_graph,
                            style='Custom.TButton')
scatter_button.grid(row=row, column=3, padx=10, pady=5)
```

```
row += 1

partial_f_test_second_accuracy_button = ttk.Button(root, text="Partial F-Test (Second Accuracy)",
                                                   command=partial_f_test_second_accuracy,
                                                   style='Custom.TButton')
partial_f_test_second_accuracy_button.grid(row=row, column=0, padx=10, pady=5)

partial_f_test_third_accuracy_button = ttk.Button(root, text="Partial F-Test (Third Accuracy)",
                                                  command=partial_f_test_third_accuracy,
                                                  style='Custom.TButton')
partial_f_test_third_accuracy_button.grid(row=row, column=1, padx=10, pady=5)

qq_plot_button = ttk.Button(root, text="Q_Q Plot & Histogram E", command=qqplot_histo,
                            style='Custom.TButton')
qq_plot_button.grid(row=row, column=2, padx=10, pady=5)

scatter_button = ttk.Button(root, text="Plot Scatter Plot of Standard and Predict", command=scatter_plot,
                            style='Custom.TButton')
scatter_button.grid(row=row, column=3, padx=10, pady=5)


row += 1

scatter_button = ttk.Button(root, text="Plot Scatter Plot of logprice and grade", command=scatter,
                            style='Custom.TButton')
scatter_button.grid(row=row, column=0, padx=10, pady=5)

scatter_button = ttk.Button(root, text="Latitude and Longitude(price)", command=scatt,
                            style='Custom.TButton')
scatter_button.grid(row=row, column=1, padx=10, pady=5)

density_button = ttk.Button(root, text="Density Price", command=density,
                            style='Custom.TButton')
density_button.grid(row=row, column=2, padx=10, pady=5)

scatter_button = ttk.Button(root, text="some columns and target", command=target,
                            style='Custom.TButton')
scatter_button.grid(row=row, column=3, padx=10, pady=5)


row += 1

pair_button = ttk.Button(root, text="Sample_1", command=pairplot_1,
                               style='Custom.TButton')
pair_button.grid(row=row, column=0, padx=10, pady=5)

pair_button = ttk.Button(root, text="Sample_2", command=pairplot_2,
                               style='Custom.TButton')
pair_button.grid(row=row, column=1, padx=10, pady=5)

pair_button = ttk.Button(root, text="Sample_3", command=pairplot_3,
                               style='Custom.TButton')
pair_button.grid(row=row, column=2, padx=10, pady=5)

root.mainloop()
```

```
[ ]:
```

Conclusion

This project aimed to predict house prices using a dataset containing various features of residential homes. Here are the key takeaways:

Data Cleaning and Preprocessing:
We began by importing necessary libraries and reading the CSV data containing house information. Exploratory analysis techniques like head, tail, describe, and info functions were used to understand the data.
Missing values were handled by dropping rows with missing entries.
Categorical features were converted to categorical data types.
New features were created, including logarithmic transformations.

Feature Engineering and Exploratory Data Analysis:
-Correlation matrices were created to identify relationships between features and the target variable (price).
-Pair plots were visualized to explore relationships between pairs of features and the target variable.
-Box plots and scatter plots were used to analyze the distribution of features and their impact on price.
-The impact of zip code on price was investigated using boxplots.
-The distribution of price was visualized with density plots and scatter plots.

Modeling and Evaluation:
Two main approaches were used for modeling: linear regression for price prediction and linear regression for log(price) prediction.
-R-squared metric was used to evaluate model performance, indicating the proportion of variance explained by the model.
-Mean squared error (MSE) was calculated to assess the average squared difference between predicted and actual prices.
-Feature importance was analyzed by examining the coefficients of the models.
-Residual analysis was performed to check for normality and homoscedasticity of residuals.
-An ANOVA table was presented to assess the significance of each feature in the model.

Partial F-tests:
Partial F-tests were conducted to evaluate the statistical significance of individual features in the models.
-The results from sklearn and statsmodels libraries were compared.

3D Visualization:
A 3D scatter plot using plotly was created to visualize the relationship between living area, bedrooms, and price.

Overall, the project successfully employed various techniques for data cleaning, feature engineering, model building, and evaluation to predict house prices in King County. The analysis provides insights into factors influencing house prices and can be a valuable resource for further investigation.