AHMET16 / house_price_phase2 Public

Code Issues Pull requests Actions Projects Wiki Security

main

house_price_phase2 / Housing_Price_Project.ipynb

AHMET16 Add files via upload History

1 contributor

2.57 MB

Housing Price Project

Overview

The goal of this project is to predict the housing sale prices in King County through a regression model. This prediction can give the seller and buyer an estimate of the housing price in King County and how specific features can affect the sale price. Based on this estimation, the buyers can find a house according to their budget, and the homeowners can get an evaluation of their house value, maybe renovate it before selling.

Business Problem

The king county real estate agency will use this prediction model to give their clients an estimate of the housing price when purchasing or selling houses. The agency will estimate the price based on certain features like the location of the house, the number of bedrooms, and the size of the house.

Data Understanding

The king county dataset was provided to me as part of this project by Flatiron School. The dataset consists of 21597 rows, 21 columns with different house features (continuous and categorical). These features will help to understand which factor will affect the selling price. Below is the description of each variable in the data frame:

- price - Price of the house sold, prediction target

- id - unique identified for a house

- date - the date when the house was sold

- bedrooms - number of bedrooms
- bathrooms - number of bathrooms
- sqft_living - square footage of the house’s interior living space
- sqft_lots - square footage of the land
- floors - number of floors
- waterfront - House which has a view to a waterfront
- view - Has been viewed by potential buyers
- condition - condition of the house coded from 1 to 5 where 1: Poor Worn out, and 5:Very Good
- grade - index from 1 to 13, where 1–3 falls short of building construction and design, 7 has an average level of construction and design, and 11–13 have a

high quality level of construction and design

- sqft_above square footage of house apart from basement
- sqft_basement - square footage of the basement
- yr_built - the year where the house was built
- yr_renovated - Year when house was renovated, and if not 0
- zipcode - zip code
- lat - Latitude coordinate
- long - Longitude coordinate
- sqft_living15 - The square footage of interior housing living space for the nearest 15 neighbors
- sqft_lot15 - The square footage of the land lots of the nearest 15 neighbors

In [1]:

```
# Imports the necessary libraries
import pandas as pd
import numpy as np
# Setting random seed for reproducibility
np.random.seed(1000)

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error, mean_square
from math import sqrt

# model tools
import statsmodels.api as sm
from statsmodels.formula.api import ols

from statsmodels.api import add_constant

import utils as ut

import warnings
warnings.filterwarnings('ignore')
```

```
/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/statsmodel
s/tsa/base/tsa_model.py:7: FutureWarning: pandas.Int64Index is depreca
ted and will be removed from pandas in a future version. Use pandas.In
dex with the appropriate dtype instead.
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/statsmodel
s/tsa/base/tsa_model.py:7: FutureWarning: pandas.Float64Index is depre
cated and will be removed from pandas in a future version. Use pandas.
Index with the appropriate dtype instead.
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
```

Obtain the data

In [2]:

```
# read in the data
df = pd.read_csv("/Users/karaoglan/Desktop/PHASE_2 PROJECT/kc_house_da
```

```
df.info()
df.head()
```

```
<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
```

Out[2]:

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

5 rows × 21 columns

In [3]:

```
#### - from the above data information, I noticed that the following:
#### - date in not in datetime format
#### - sqft_basement is an object need to see why and turn it to numer
```

In [4]:

```
df.describe().T
```

Out[4]:

	count	mean	std	min	25%

id	21597.0	4.580474e+09	2.876736e+09	1.000102e+06	2.123049e+09
price	21597.0	5.402966e+05	3.673681e+05	7.800000e+04	3.220000e+05
bedrooms	21597.0	3.373200e+00	9.262989e-01	1.000000e+00	3.000000e+00
bathrooms	21597.0	2.115826e+00	7.689843e-01	5.000000e-01	1.750000e+00
sqft_living	21597.0	2.080322e+03	9.181061e+02	3.700000e+02	1.430000e+03
sqft_lot	21597.0	1.509941e+04	4.141264e+04	5.200000e+02	5.040000e+03
floors	21597.0	1.494096e+00	5.396828e-01	1.000000e+00	1.000000e+00
sqft_above	21597.0	1.788597e+03	8.277598e+02	3.700000e+02	1.190000e+03
yr_built	21597.0	1.971000e+03	2.937523e+01	1.900000e+03	1.951000e+03
yr_renovated	17755.0	8.363678e+01	3.999464e+02	0.000000e+00	0.000000e+00
zipcode	21597.0	9.807795e+04	5.351307e+01	9.800100e+04	9.803300e+04
lat	21597.0	4.756009e+01	1.385518e-01	4.715590e+01	4.747110e+01
long	21597.0	-1.222140e+02	1.407235e-01	-1.225190e+02	-1.223280e+02
sqft_living15	21597.0	1.986620e+03	6.852305e+02	3.990000e+02	1.490000e+03
sqft_lot15	21597.0	1.275828e+04	2.727444e+04	6.510000e+02	5.100000e+03

Scrub the data

In [5]:

```
# check if we have duplicate house
df[['id']].duplicated().sum() # check if we have duplicate houses
```

Out[5]:

```
177
```

In [6]:

```
df["id"].drop_duplicates(inplace=True)
```

In [7]:

```
df["id"].duplicated().any() #sanity check
```

Out[7]:

```
True
```

In [8]:

```
# check for null alues n the data
df.isnull().sum() # check for null values in the data
```

Out[8]:

```
id                 0
date               0
price              0
bedrooms           0
bathrooms          0
sqft_living        0
sqft_lot           0
floors             0
waterfront      2376
view              63
```

```
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_built            0
yr_renovated     3842
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
dtype: int64
```

In [9]:

```
# I'll check the null in waterfront and yr_renovated, and drop the vie
#because it is not important if the house was viewed or not
```

In [10]:

```
df.head()
```

Out[10]:

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

5 rows × 21 columns

In [11]:

```
df
```

Out[11]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lo
0	7129300520	10/13/2014	221900.0	3	1.00	1180	565(
1	6414100192	12/9/2014	538000.0	3	2.25	2570	724:
2	5631500400	2/25/2015	180000.0	2	1.00	770	1000(
3	2487200875	12/9/2014	604000.0	4	3.00	1960	500(
4	1954400510	2/18/2015	510000.0	3	2.00	1680	808(
...	...	...	...	...	...	...	..
21592	263000018	5/21/2014	360000.0	3	2.50	1530	113
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	581:

21594	1523300141	6/23/2014	402101.0	2	0.75	1020	135(
21595	291310100	1/16/2015	400000.0	3	2.50	1600	238
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	107

21597 rows × 21 columns

In [12]:

```
def waterfront11(x):
    if x == "NO":
        return 0
    if x == "YES":
        return 1
 # df[waterfront1] = np.where(df.waterfront == 'YES', 1, 0)
```

In [13]:

```
df["waterfront1"] = df["waterfront"].apply(waterfront11)
df
```

Out[13]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lo
0	7129300520	10/13/2014	221900.0	3	1.00	1180	565(
1	6414100192	12/9/2014	538000.0	3	2.25	2570	724
2	5631500400	2/25/2015	180000.0	2	1.00	770	1000(
3	2487200875	12/9/2014	604000.0	4	3.00	1960	500(
4	1954400510	2/18/2015	510000.0	3	2.00	1680	808(
...	...	...	...	...	...	...	..
21592	263000018	5/21/2014	360000.0	3	2.50	1530	113
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	581
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	135(
21595	291310100	1/16/2015	400000.0	3	2.50	1600	238
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	107

21597 rows × 22 columns

In [14]:

```
df["waterfront1"] = df["waterfront1"].fillna(0)
```

In [15]:

```
df["waterfront1"].value_counts(dropna=False)
```

Out[15]:

```
0.0    21451
1.0      146
Name: waterfront1, dtype: int64
```

In [16]:

```
df.drop("waterfront", axis=1,inplace=True)
df
```

Out[16]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lo
0	7129300520	10/13/2014	221900.0	3	1.00	1180	565(
1	6414100192	12/9/2014	538000.0	3	2.25	2570	724:
2	5631500400	2/25/2015	180000.0	2	1.00	770	1000(
3	2487200875	12/9/2014	604000.0	4	3.00	1960	500(
4	1954400510	2/18/2015	510000.0	3	2.00	1680	808(
...	...	...	...	...	...	...	...	..
21592	263000018	5/21/2014	360000.0	3	2.50	1530	113
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	581:
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	135(
21595	291310100	1/16/2015	400000.0	3	2.50	1600	238
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	107(

21597 rows × 21 columns

In [17]:

```
df['condition1'] = df['condition'].map(lambda x: len(x.split()))
df.head(50)
```

Out[17]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080
5	7237550310	5/12/2014	1230000.0	4	4.50	5420	101930
6	1321400060	6/27/2014	257500.0	3	2.25	1715	6819
7	2008000270	1/15/2015	291850.0	3	1.50	1060	9711
8	2414600126	4/15/2015	229500.0	3	1.00	1780	7470
9	3793500160	3/12/2015	323000.0	3	2.50	1890	6560
10	1736800520	4/3/2015	662500.0	3	2.50	3560	9796
11	9212900260	5/27/2014	468000.0	2	1.00	1160	6000
12	114101516	5/28/2014	310000.0	3	1.00	1430	19901
13	6054650070	10/7/2014	400000.0	3	1.75	1370	9680

14	1175000570	3/12/2015	530000.0	5	2.00	1810	4850
15	9297300055	1/24/2015	650000.0	4	3.00	2950	5000
16	1875500060	7/31/2014	395000.0	3	2.00	1890	14040
17	6865200140	5/29/2014	485000.0	4	1.00	1600	4300
18	16000397	12/5/2014	189000.0	2	1.00	1200	9850
19	7983200060	4/24/2015	230000.0	3	1.00	1250	9774
20	6300500875	5/14/2014	385000.0	4	1.75	1620	4980
21	2524049179	8/26/2014	2000000.0	3	2.75	3050	44867
22	7137970340	7/3/2014	285000.0	5	2.50	2270	6300
23	8091400200	5/16/2014	252700.0	2	1.50	1070	9643
24	3814700200	11/20/2014	329000.0	3	2.25	2450	6500
25	1202000200	11/3/2014	233000.0	3	2.00	1710	4697
26	1794500383	6/26/2014	937000.0	3	1.75	2450	2691
27	3303700376	12/1/2014	667000.0	3	1.00	1400	1581
28	5101402488	6/24/2014	438000.0	3	1.75	1520	6380
29	1873100390	3/2/2015	719000.0	4	2.50	2570	7173
30	8562750320	11/10/2014	580500.0	3	2.50	2320	3980
31	2426039314	12/1/2014	280000.0	2	1.50	1190	1265
32	461000390	6/24/2014	687500.0	4	1.75	2330	5000
33	7589200193	11/10/2014	535000.0	3	1.00	1090	3000
34	7955080270	12/3/2014	322500.0	4	2.75	2060	6659
35	9547205180	6/13/2014	696000.0	3	2.50	2300	3060
36	9435300030	5/28/2014	550000.0	4	1.00	1660	34848
37	2768000400	12/30/2014	640000.0	4	2.00	2360	6000
38	7895500070	2/13/2015	240000.0	4	1.00	1220	8075
39	2078500320	6/20/2014	605000.0	4	2.50	2620	7553
40	5547700270	7/15/2014	625000.0	4	2.50	2570	5520
41	7766200013	8/11/2014	775000.0	4	2.25	4220	24186
42	7203220400	7/7/2014	861990.0	5	2.75	3595	5639
43	9270200160	10/28/2014	685000.0	3	1.00	1570	2280
44	1432701230	7/29/2014	309000.0	3	1.00	1280	9656
45	8035350320	7/18/2014	488000.0	3	2.50	3160	13603
46	8945200830	3/25/2015	210490.0	3	1.00	990	8528
47	4178300310	7/16/2014	785000.0	4	2.50	2290	13416
48	9215400105	4/28/2015	450000.0	3	1.75	1250	5963

49	822039084	3/11/2015	1350000.0	3	2.50	2753	65005

50 rows × 22 columns

In [18]:

```
df.drop("condition", axis=1,inplace=True)
df
```

Out[18]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lo
0	7129300520	10/13/2014	221900.0	3	1.00	1180	565
1	6414100192	12/9/2014	538000.0	3	2.25	2570	724
2	5631500400	2/25/2015	180000.0	2	1.00	770	1000
3	2487200875	12/9/2014	604000.0	4	3.00	1960	500
4	1954400510	2/18/2015	510000.0	3	2.00	1680	808
...	...	...	...	...	...	...	..
21592	263000018	5/21/2014	360000.0	3	2.50	1530	113
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	581
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	135
21595	291310100	1/16/2015	400000.0	3	2.50	1600	238
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	107

21597 rows × 21 columns

In [19]:

```
df.fillna(0)
```

Out[19]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lo
0	7129300520	10/13/2014	221900.0	3	1.00	1180	565
1	6414100192	12/9/2014	538000.0	3	2.25	2570	724
2	5631500400	2/25/2015	180000.0	2	1.00	770	1000
3	2487200875	12/9/2014	604000.0	4	3.00	1960	500
4	1954400510	2/18/2015	510000.0	3	2.00	1680	808
...	...	...	...	...	...	...	..

...	...	...	...	...	...	...	..
21592	263000018	5/21/2014	360000.0	3	2.50	1530	113
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	581
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	135
21595	291310100	1/16/2015	400000.0	3	2.50	1600	238
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	107

21597 rows × 21 columns

In [20]:

```
df["view"].value_counts(dropna=False)
```

Out[20]:

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

In [21]:

```
def view(x):
    if x == "NONE":
        return 0
    if x == "AVERAGE":
        return 2
    if x == "GOOD":
        return 3
    if x == "FAIR":
        return 1
    if x == "EXCELLENT":
        return 4
```

In [22]:

```
df["view"] = df["view"].apply(view)
df
```

Out[22]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lo
0	7129300520	10/13/2014	221900.0	3	1.00	1180	565
1	6414100192	12/9/2014	538000.0	3	2.25	2570	724
2	5631500400	2/25/2015	180000.0	2	1.00	770	1000
3	2487200875	12/9/2014	604000.0	4	3.00	1960	500
4	1954400510	2/18/2015	510000.0	3	2.00	1680	808
...	...	...	...	...	...	...	..

21592	263000018	5/21/2014	360000.0	3	2.50	1530	113
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	581:
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	135(
21595	291310100	1/16/2015	400000.0	3	2.50	1600	238:
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	107(

21597 rows × 21 columns

In [23]:

```
df.head(200)
```

Out[23]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080
...	...	...	...	...	...	...	...
195	7796450200	5/15/2014	256883.0	3	2.50	1690	5025
196	7549802535	11/11/2014	423000.0	4	2.00	1970	6480
197	3278600320	7/23/2014	465000.0	3	2.50	2150	4084
198	2824079053	1/13/2015	440000.0	3	2.50	1910	66211
199	1222069094	10/14/2014	385000.0	3	1.75	1350	155073

200 rows × 21 columns

In [24]:

```
df.grade=df.grade.replace(['7 Average','8 Good','9 Better','6 Low Aver
                [7,8,9,6,10,11,5,12,4,13,3])
```

In [25]:

```
df["grade"].value_counts(dropna=False)
```

Out[25]:

```
7      8974
8      6065
9      2615
```

```
6      2038
10     1134
11      399
5       242
12       89
4        27
13       13
3         1
Name: grade, dtype: int64
```

In [26]:
```
df['grade'] = df['grade'].astype(float)
```

In [27]:
```
df
```

Out[27]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lo
0	7129300520	10/13/2014	221900.0	3	1.00	1180	565(
1	6414100192	12/9/2014	538000.0	3	2.25	2570	724:
2	5631500400	2/25/2015	180000.0	2	1.00	770	1000(
3	2487200875	12/9/2014	604000.0	4	3.00	1960	500(
4	1954400510	2/18/2015	510000.0	3	2.00	1680	808(
...	...	...	...	...	...	...	..
21592	263000018	5/21/2014	360000.0	3	2.50	1530	113
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	581:
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	135(
21595	291310100	1/16/2015	400000.0	3	2.50	1600	238(
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	107(

21597 rows × 21 columns

In [28]:
```
#df['grade'] = df['grade'].map(lambda x: len(x.split()))
```

In [29]:
```
df["grade"].value_counts(dropna=False)
```

Out[29]:
```
7.0     8974
8.0     6065
9.0     2615
6.0     2038
10.0    1134
11.0     399
5.0      242
12.0      89
4.0       27
13.0      13
3.0        1
Name: grade, dtype: int64
```

In [30]:

```
df
```

Out[30]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lo
0	7129300520	10/13/2014	221900.0	3	1.00	1180	565(
1	6414100192	12/9/2014	538000.0	3	2.25	2570	724:
2	5631500400	2/25/2015	180000.0	2	1.00	770	1000(
3	2487200875	12/9/2014	604000.0	4	3.00	1960	500(
4	1954400510	2/18/2015	510000.0	3	2.00	1680	808(
...	...	...	...	...	...	...	..
21592	263000018	5/21/2014	360000.0	3	2.50	1530	113
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	581:
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	135(
21595	291310100	1/16/2015	400000.0	3	2.50	1600	238{
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	107(

21597 rows × 21 columns

In [31]:

```
df["yr_renovated"] = df["yr_renovated"].fillna(value = 0)
# I filled the null with 0 because I think null here means not renovat
df["yr_renovated"].unique()
```

Out[31]:

```
array([   0., 1991., 2002., 2010., 1992., 2013., 1994., 1978., 2005.,
       2003., 1984., 1954., 2014., 2011., 1983., 1945., 1990., 1988.,
       1977., 1981., 1995., 2000., 1999., 1998., 1970., 1989., 2004.,
       1986., 2007., 1987., 2006., 1985., 2001., 1980., 1971., 1979.,
       1997., 1950., 1969., 1948., 2009., 2015., 1974., 2008., 1968.,
       2012., 1963., 1951., 1962., 1953., 1993., 1996., 1955., 1982.,
       1956., 1940., 1976., 1946., 1975., 1964., 1973., 1957., 1959.,
       1960., 1967., 1965., 1934., 1972., 1944., 1958.])
```

In [32]:

```
df["sqft_basement"].unique()
#checking what is making sqft_basement an object
df["sqft_basement"].value_counts()
```

Out[32]:

```
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
```

In [33]:

In [33]:
```
df["sqft_basement"] = df["sqft_basement"].replace("?", 0).astype(float
#replace the ? with 0 and change it to float type
df["sqft_basement"].value_counts()
```

Out[33]:
```
0.0       13280
600.0       217
500.0       209
700.0       208
800.0       201
          ...
1920.0        1
3480.0        1
2730.0        1
2720.0        1
248.0         1
Name: sqft_basement, Length: 303, dtype: int64
```

In [34]:
```
df.isnull().sum()
#sanity check
```

Out[34]:
```
id                0
date              0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
view             63
grade             0
sqft_above        0
sqft_basement     0
yr_built          0
yr_renovated      0
zipcode           0
lat               0
long              0
sqft_living15     0
sqft_lot15        0
waterfront1       0
condition1        0
dtype: int64
```

In [35]:
```
df = df.drop(["id","date","view"], axis = 1)

# drop unwanted columns. id: there is no use of the id in the model,
#same as the selling date, and I don't need  view if the house has bee
```

In [36]:
```
df.describe()
```

Out[36]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	
count	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	2159
mean	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	
std	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	
min	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	

min	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02
25%	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06

In [37]:

```
df
```

Out[37]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	grade	sqft_ab
0	221900.0	3	1.00	1180	5650	1.0	7.0	1
1	538000.0	3	2.25	2570	7242	2.0	7.0	2
2	180000.0	2	1.00	770	10000	1.0	6.0	
3	604000.0	4	3.00	1960	5000	1.0	7.0	1
4	510000.0	3	2.00	1680	8080	1.0	8.0	1
...	...	...	...	...	...	...	...	
21592	360000.0	3	2.50	1530	1131	3.0	8.0	1
21593	400000.0	4	2.50	2310	5813	2.0	8.0	2
21594	402101.0	2	0.75	1020	1350	2.0	7.0	1
21595	400000.0	3	2.50	1600	2388	2.0	8.0	1
21596	325000.0	2	0.75	1020	1076	2.0	7.0	1

21597 rows × 18 columns

In [38]:

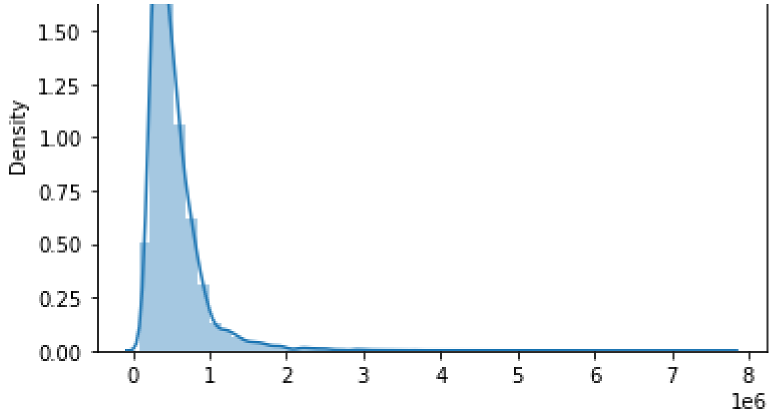
```
#I will set a range for each feature and get rid of outliers, I will l
#from .describe
```

In [39]:

```
ut.plot(df,["price"])
df.price.describe()
```

Out[39]:

```
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
```



In [40]:

```
df=df[(df['price'] < 12000000) & (df['price'] >100000)] # limiting my
df
```

Out[40]:

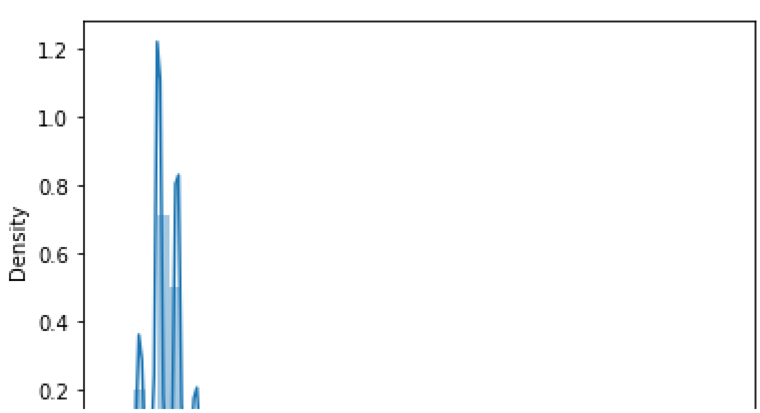
	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	grade	sqft_abo
0	221900.0	3	1.00	1180	5650	1.0	7.0	1
1	538000.0	3	2.25	2570	7242	2.0	7.0	2
2	180000.0	2	1.00	770	10000	1.0	6.0	
3	604000.0	4	3.00	1960	5000	1.0	7.0	10
4	510000.0	3	2.00	1680	8080	1.0	8.0	16
...	...	...	...	...	...	...	...	
21592	360000.0	3	2.50	1530	1131	3.0	8.0	15
21593	400000.0	4	2.50	2310	5813	2.0	8.0	2
21594	402101.0	2	0.75	1020	1350	2.0	7.0	10
21595	400000.0	3	2.50	1600	2388	2.0	8.0	16
21596	325000.0	2	0.75	1020	1076	2.0	7.0	10

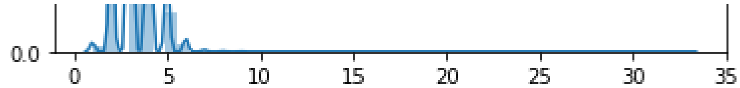
21567 rows × 18 columns

In [41]:

```
ut.plot(df,["bedrooms"])
```

Out[41]: <AxesSubplot:ylabel='Density'>





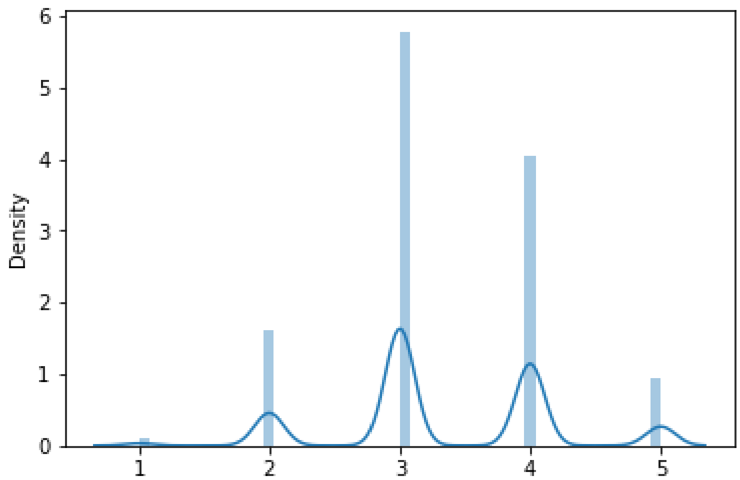
In [42]:

```
df=df[(df['bedrooms']<6)]
# remove the outlier
```

In [43]:

```
ut.plot(df,['bedrooms'])
```

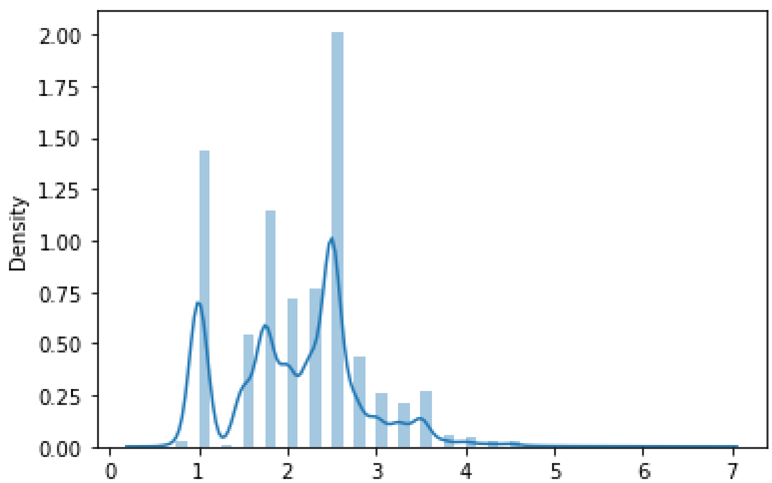
Out[43]: <AxesSubplot:ylabel='Density'>



In [44]:

```
ut.plot(df,['bathrooms'])
```

Out[44]: <AxesSubplot:ylabel='Density'>



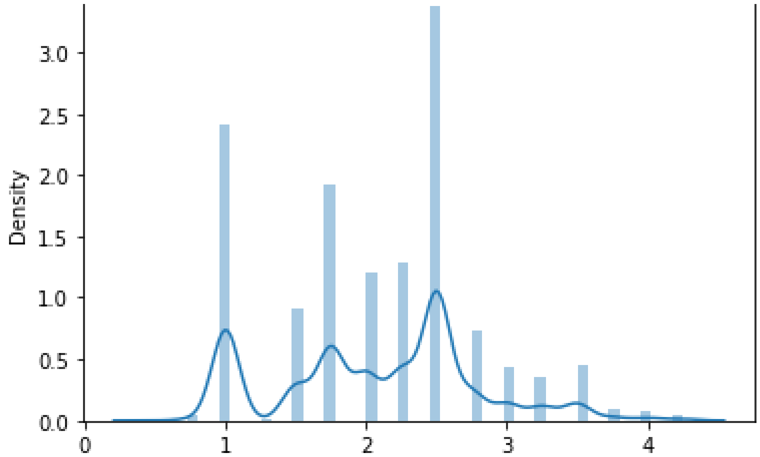
In [45]:

```
df=df[(df['bathrooms']<4.5)]
#remove the outliner
```

In [46]:

```
ut.plot(df,['bathrooms'])
```

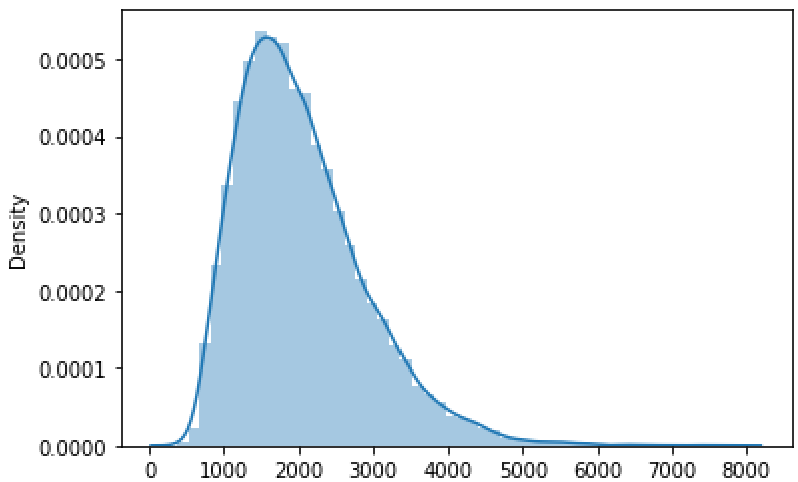
Out[46]: <AxesSubplot:ylabel='Density'>



In [47]:

```
ut.plot(df,['sqft_living'])
```

Out[47]: <AxesSubplot:ylabel='Density'>



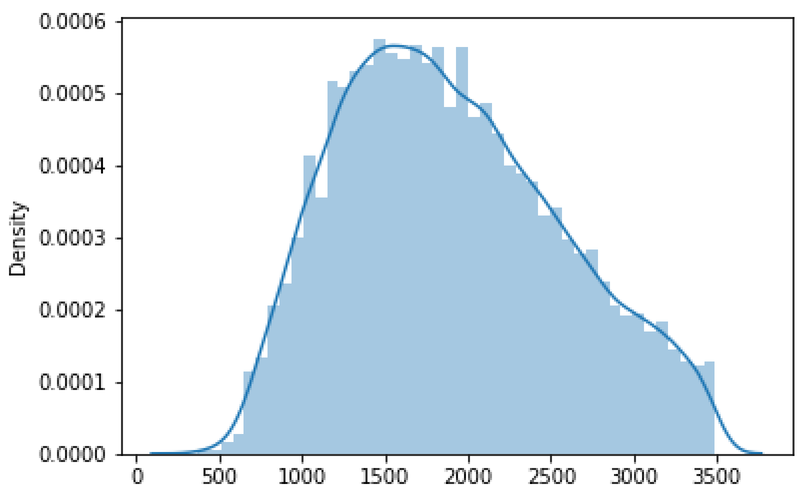
In [48]:

```
df=df[(df["sqft_living"]<3500)]
```

In [49]:

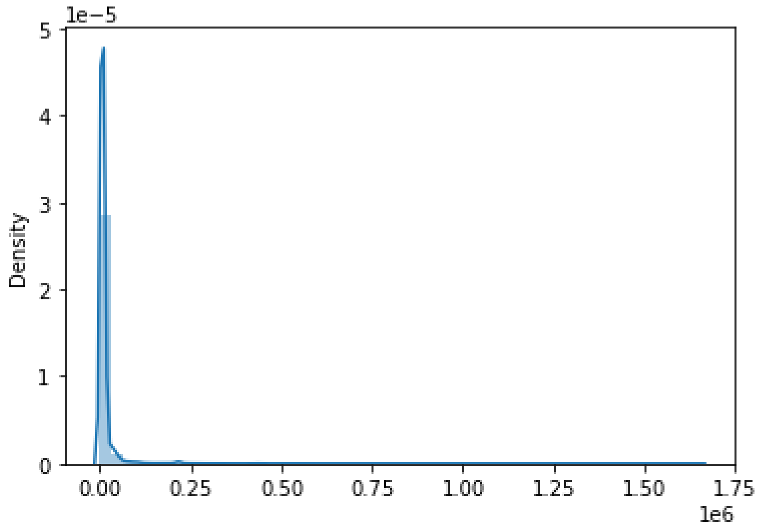
```
ut.plot(df,["sqft_living"])
```

Out[49]: <AxesSubplot:ylabel='Density'>



In [50]:
```
ut.plot(df,["sqft_lot"])
```

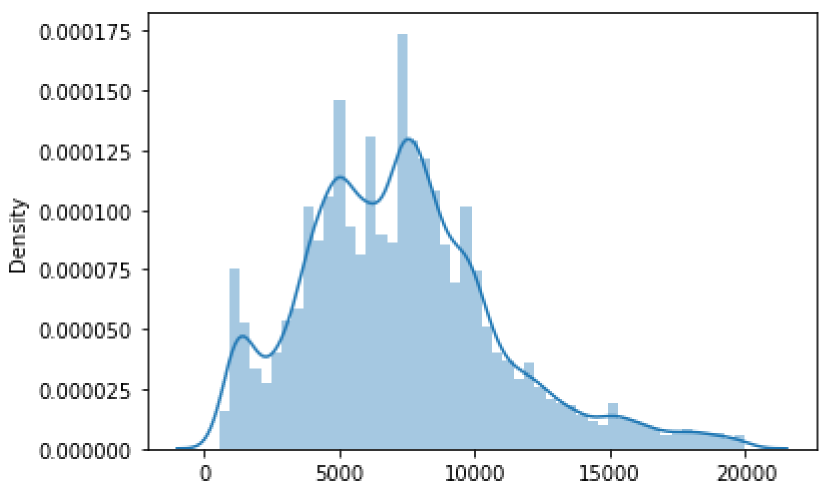
Out[50]: <AxesSubplot:ylabel='Density'>



In [51]:
```
df=df[(df["sqft_lot"]<20000)]
```

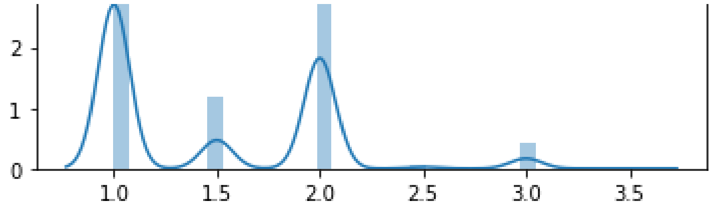
In [52]:
```
ut.plot(df,["sqft_lot"])
```

Out[52]: <AxesSubplot:ylabel='Density'>



In [53]:
```
ut.plot(df,['floors'])
```

Out[53]: <AxesSubplot:ylabel='Density'>



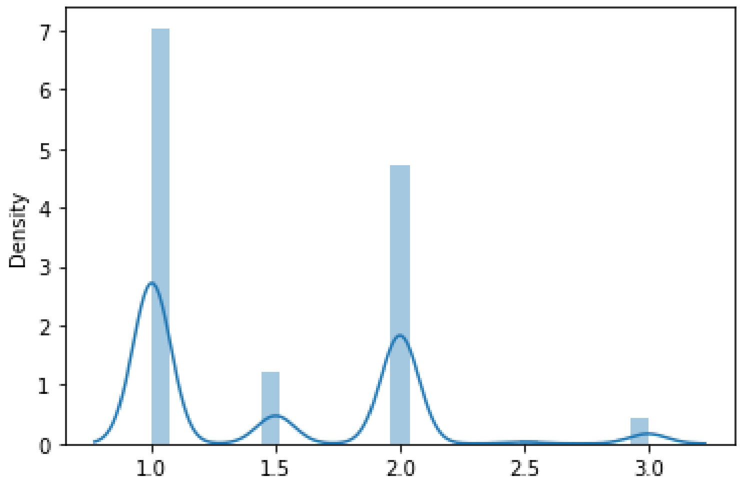
In [54]:

```
df=df[(df["floors"]<3.5)]
```

In [55]:

```
ut.plot(df,["floors"])
```

Out[55]: <AxesSubplot:ylabel='Density'>



In [56]:

```
df["condition1"].unique()
```

Out[56]: array([1, 2])

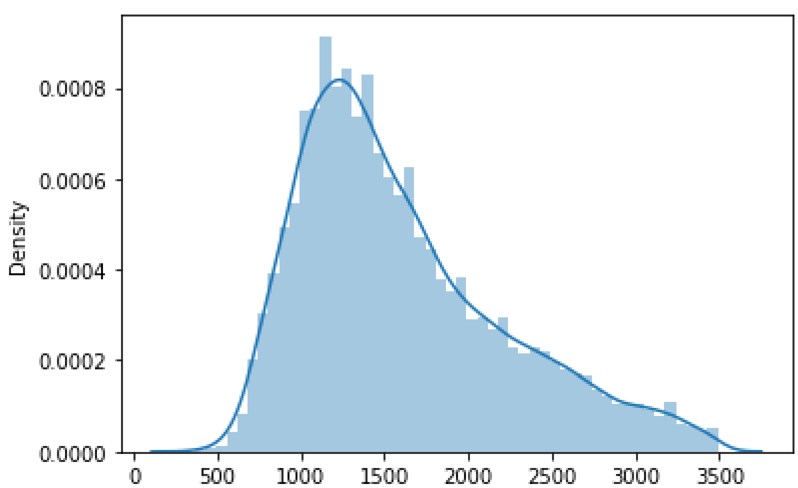
list = [] for i in list:

df["grade"].unique()

In [57]:

```
ut.plot(df,["sqft_above"])
```

Out[57]: <AxesSubplot:ylabel='Density'>



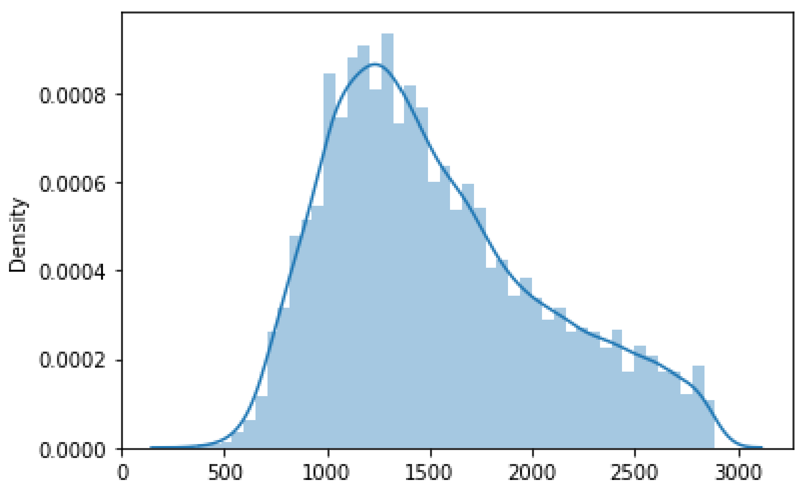
In [58]:
```python
df["sqft_above"].value_counts().sort_values(ascending=True)
```

Out[58]:
```
1425      1
3087      1
2198      1
1333      1
2531      1
       ...
1140    170
1220    179
1200    192
1300    196
1010    200
Name: sqft_above, Length: 658, dtype: int64
```

In [59]:
```python
df=df[(df['sqft_above'] <2900)]
```

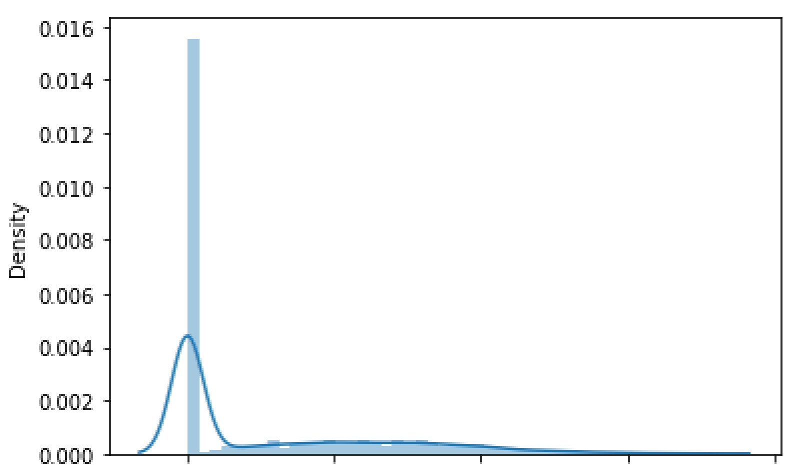
In [60]:
```python
ut.plot(df,["sqft_above"])
```

Out[60]: <AxesSubplot:ylabel='Density'>



In [61]:
```python
ut.plot(df,["sqft_basement"])
```

Out[61]: <AxesSubplot:ylabel='Density'>



In [62]:

```
df["sqft_basement"].value_counts().sort_values(ascending=True)
```

Out[62]:

```
248.0        1
295.0        1
283.0        1
266.0        1
207.0        1
         ...
800.0      172
700.0      185
600.0      187
500.0      194
0.0      10363
Name: sqft_basement, Length: 208, dtype: int64
```

In [63]:

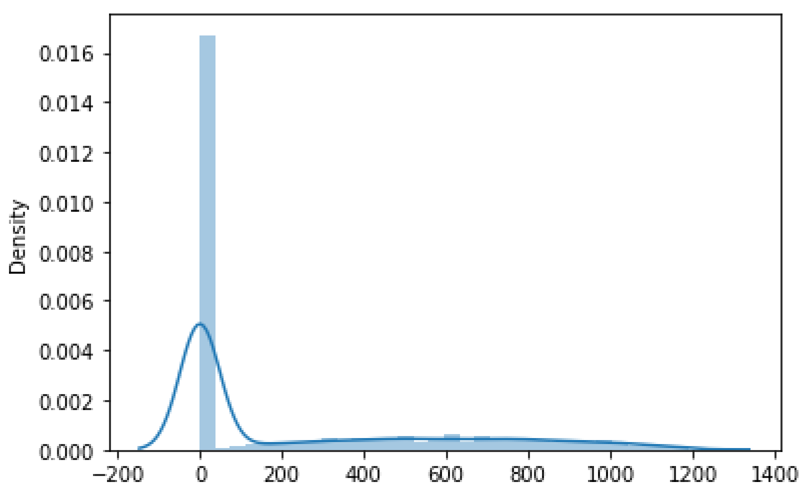
```
df=df[(df["sqft_basement"]<1200)]
```

In [64]:

```
ut.plot(df,["sqft_basement"])
```

Out[64]:

```
<AxesSubplot:ylabel='Density'>
```



In [65]:

```
df["yr_built"].value_counts().sort_values(ascending=True)
```

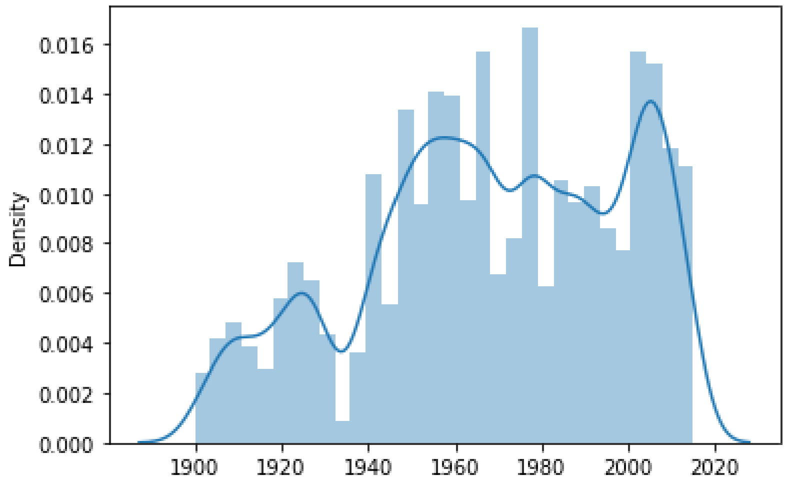
Out[65]:

```
1935     16
1934     16
1933     18
1902     25
1901     25
       ...
2007    301
2003    305
1968    311
2005    314
2014    364
Name: yr_built, Length: 116, dtype: int64
```

In [66]:

```
ut.plot(df,["yr_built"])
```

Out[66]:
```
<AxesSubplot:ylabel='Density'>
```



In [67]:
```python
df["yr_renovated"].value_counts().sort_values(ascending=True)
```

Out[67]:
```
1957.0        1
1934.0        1
1959.0        1
1944.0        1
1948.0        1
           ...
2003.0       20
2005.0       22
2013.0       25
2014.0       60
0.0       16157
Name: yr_renovated, Length: 69, dtype: int64
```

In [68]:
```python
df["renovated"]= df["yr_renovated"].apply(lambda x:1 if x!=0 else 0)
# assign the value in the "yr_renovated" columns to binary value if it
```

In [69]:
```python
df["renovated"].value_counts()
```

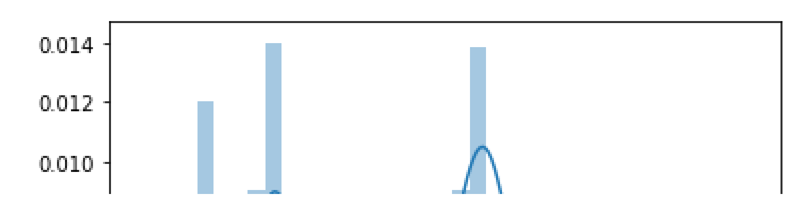
Out[69]:
```
0    16157
1      523
Name: renovated, dtype: int64
```

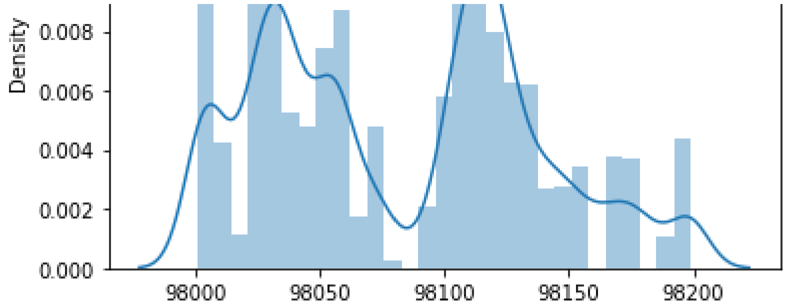
In [70]:
```python
df =df.drop(["yr_renovated"],axis=1)

#drop "yr_renovated"
```

In [71]:
```python
ut.plot(df,['zipcode'])
```

Out[71]:
```
<AxesSubplot:ylabel='Density'>
```





In [72]:
```
df['zipcode'].value_counts()
```

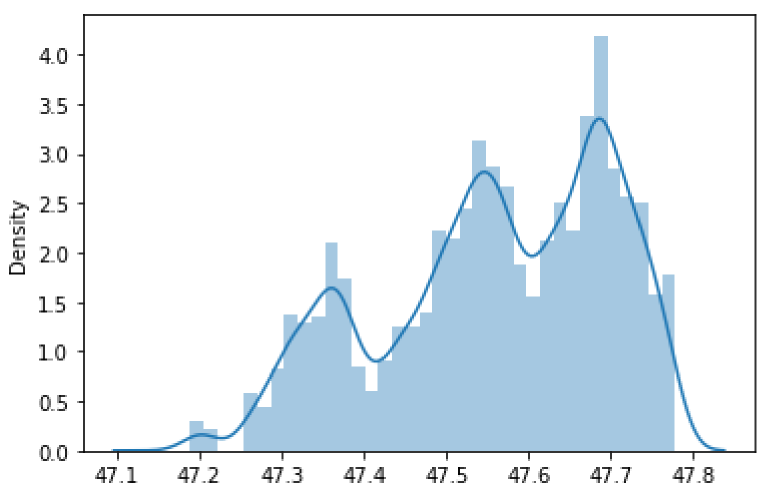
Out[72]:
```
98103    573
98115    542
98117    524
98133    467
98118    466
        ... 
98014     50
98077     29
98070     28
98024     25
98039     16
Name: zipcode, Length: 70, dtype: int64
```

In [73]:
```
df['zipcode'].nunique()
```

Out[73]:
```
70
```

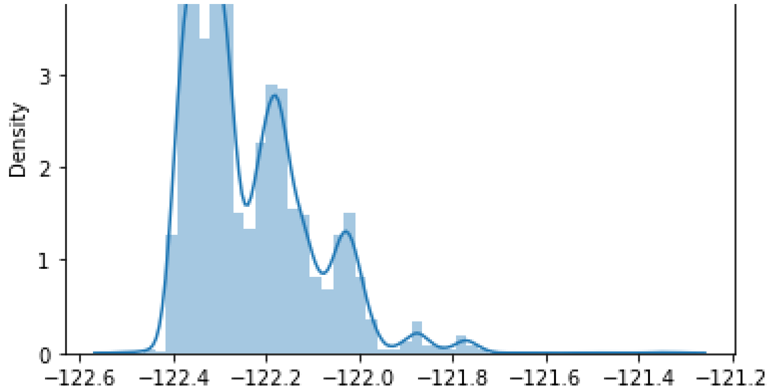
In [74]:
```
ut.plot(df,['lat'])
```

Out[74]:
```
<AxesSubplot:ylabel='Density'>
```



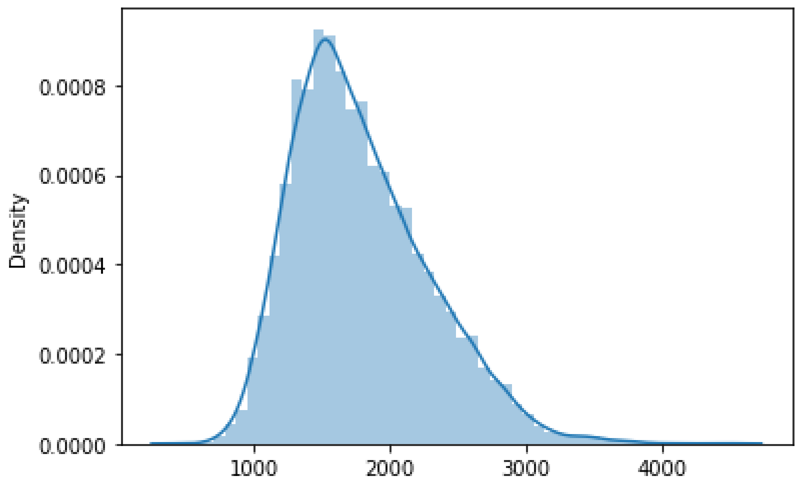
In [75]:
```
ut.plot(df,['long'])
```

Out[75]:
```
<AxesSubplot:ylabel='Density'>
```



In [76]:
```
ut.plot(df,['sqft_living15'])
```

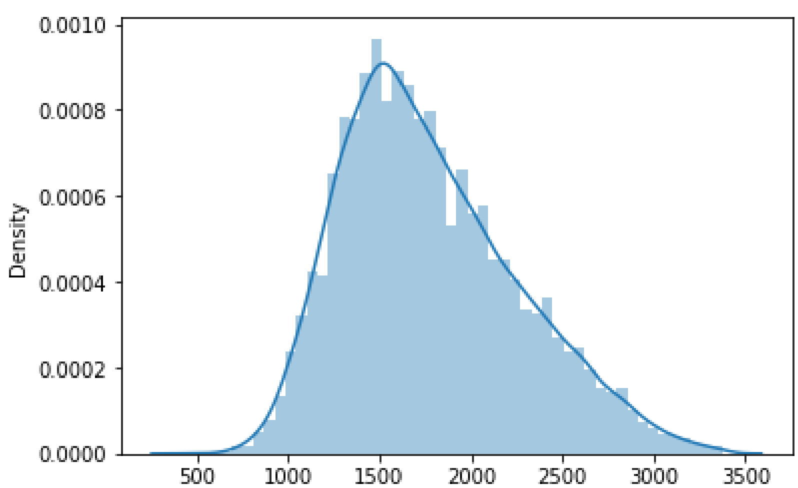
Out[76]: <AxesSubplot:ylabel='Density'>



In [77]:
```
df=df[(df['sqft_living15']<3400)]
```

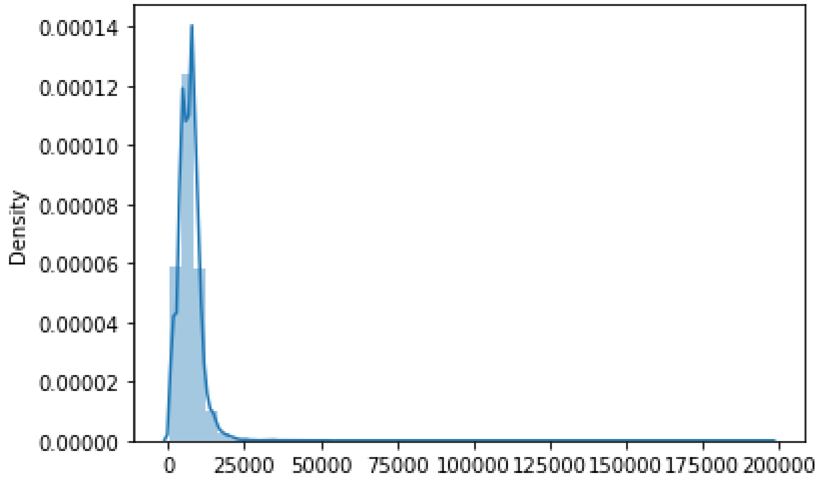
In [78]:
```
ut.plot(df,['sqft_living15'])
```

Out[78]: <AxesSubplot:ylabel='Density'>



In [79]:
```
ut.plot(df,["sqft_lot15"])
```

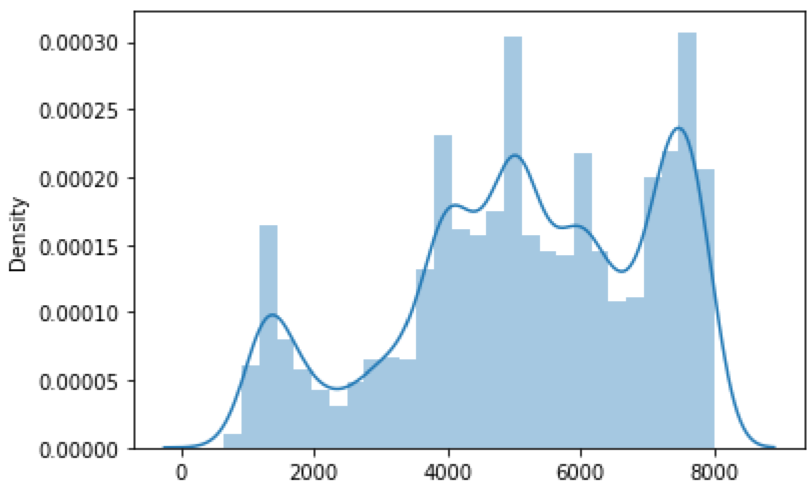
Out[79]: <AxesSubplot:ylabel='Density'>



In [80]:
```
df=df[(df["sqft_lot15"]<8000)]
```

In [81]:
```
ut.plot(df,["sqft_lot15"])
```

Out[81]: <AxesSubplot:ylabel='Density'>



Explore the data

In [82]:
```
df.describe()
```

Out[82]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	
count	1.061500e+04	10615.000000	10615.000000	10615.000000	10615.000000	1061
mean	4.666713e+05	3.123787	1.976613	1731.258691	5256.174282	
std	2.169005e+05	0.812062	0.687662	578.651917	2385.108910	
min	1.025000e+05	1.000000	0.500000	370.000000	520.000000	
25%	3.150000e+05	3.000000	1.500000	1290.000000	3800.000000	
50%	4.250000e+05	3.000000	2.000000	1670.000000	5120.000000	

75%	5.659985e+05	4.000000	2.500000	2130.000000	7000.000000
max	2.580000e+06	5.000000	4.250000	3490.000000	19969.000000

In [83]:

```
df.shape
```

Out[83]:

```
(10615, 18)
```

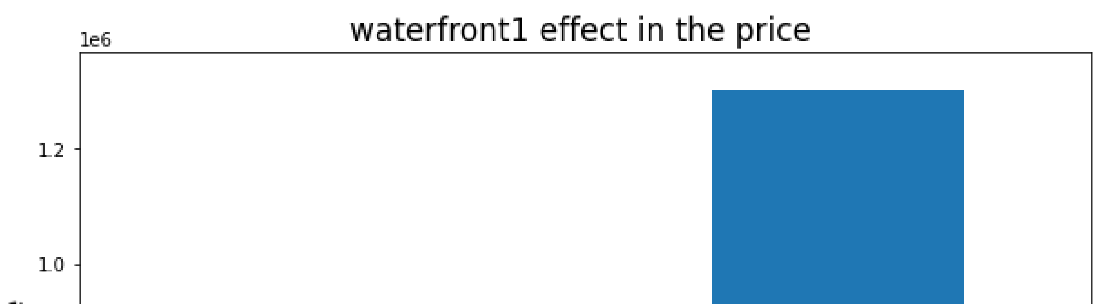
In [84]:

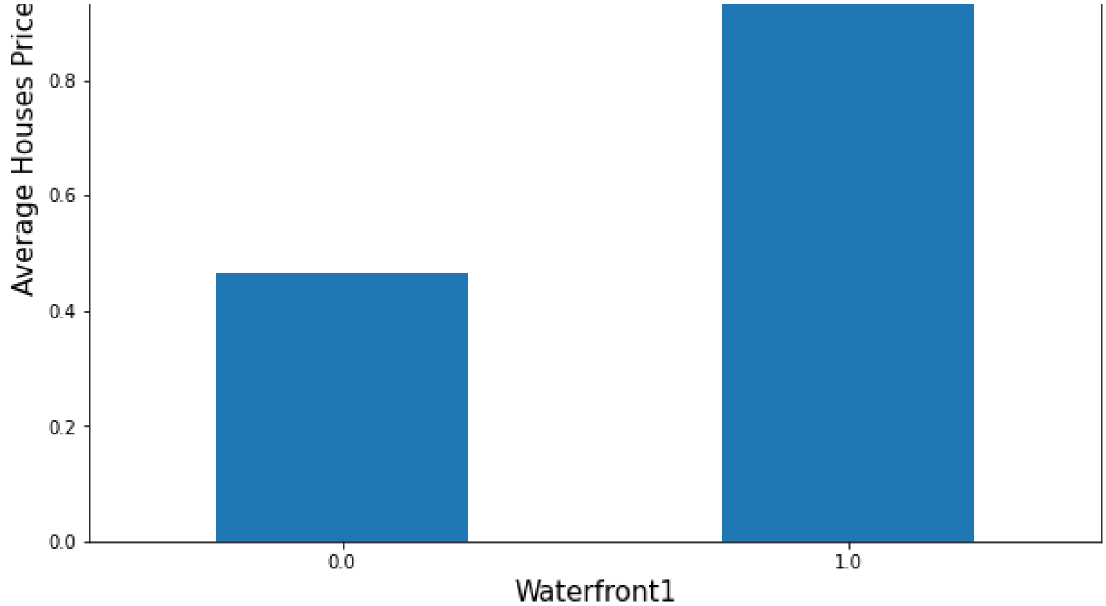
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10615 entries, 0 to 21596
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          10615 non-null  float64
 1   bedrooms       10615 non-null  int64  
 2   bathrooms      10615 non-null  float64
 3   sqft_living    10615 non-null  int64  
 4   sqft_lot       10615 non-null  int64  
 5   floors         10615 non-null  float64
 6   grade          10615 non-null  float64
 7   sqft_above     10615 non-null  int64  
 8   sqft_basement  10615 non-null  float64
 9   yr_built       10615 non-null  int64  
 10  zipcode        10615 non-null  int64  
 11  lat            10615 non-null  float64
 12  long           10615 non-null  float64
 13  sqft_living15  10615 non-null  int64  
 14  sqft_lot15     10615 non-null  int64  
 15  waterfront1    10615 non-null  float64
 16  condition1     10615 non-null  int64  
 17  renovated      10615 non-null  int64  
dtypes: float64(8), int64(10)
memory usage: 1.5 MB
```

In [85]:

```
# plotting houses to the mean of price
df.groupby("waterfront1")["price"].mean().plot(kind="bar",figsize=(10,
plt.title("waterfront1 effect in the price ", fontsize=17)
plt.ylabel("Average Houses Price",fontsize=15)
plt.xlabel("Waterfront1",fontsize=15)
plt.xticks(rotation=0)
plt.show()
#the houses with waterfront selling price are higher than one without
```





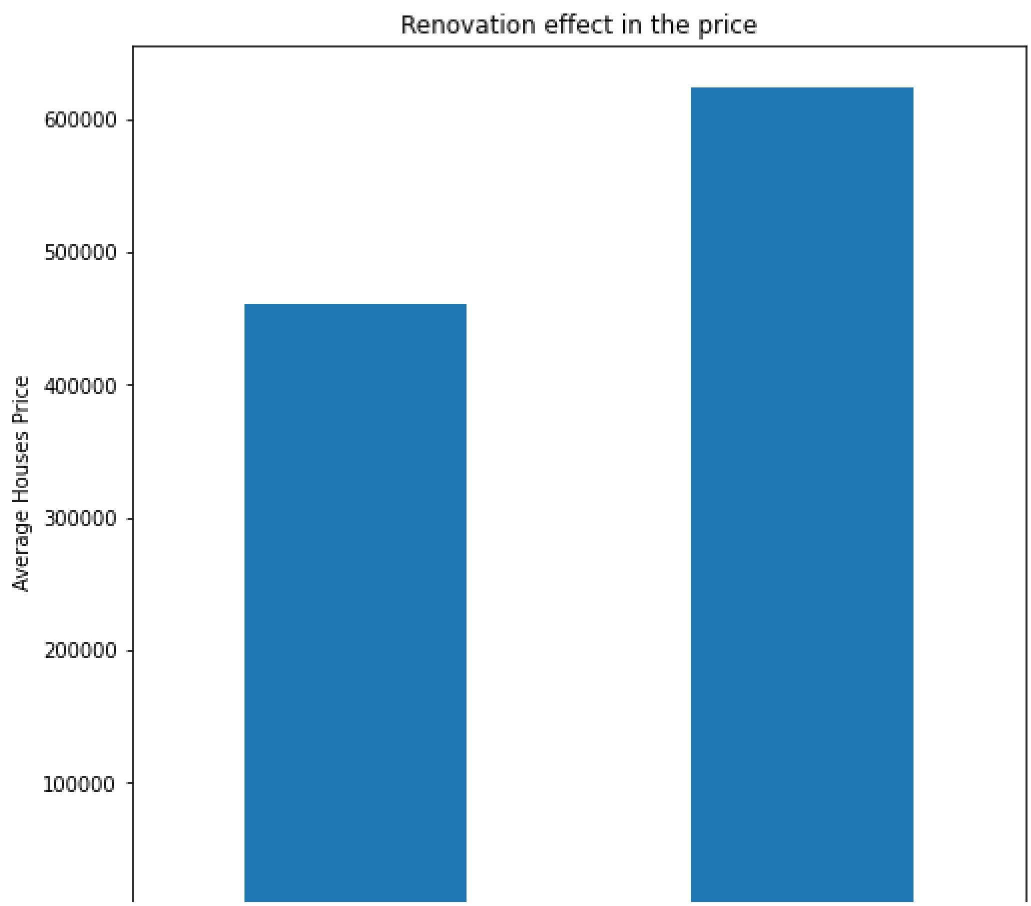
In [86]:

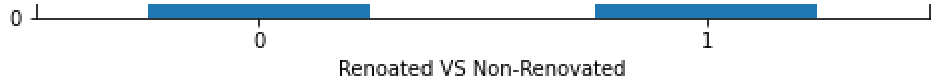
```
# plotting houses to the mean of price
df.groupby("renovated")["price"].mean().plot(kind="bar",figsize=(8,8))
plt.title("Renovation effect in the price")
plt.ylabel("Average Houses Price")
plt.xlabel("Renoated VS Non-Renovated")
plt.xticks(rotation=0)


#the renovated houses selling price is higher than non-renovated one
```

Out[86]:

```
(array([0, 1]), [Text(0, 0, '0'), Text(1, 0, '1')])
```

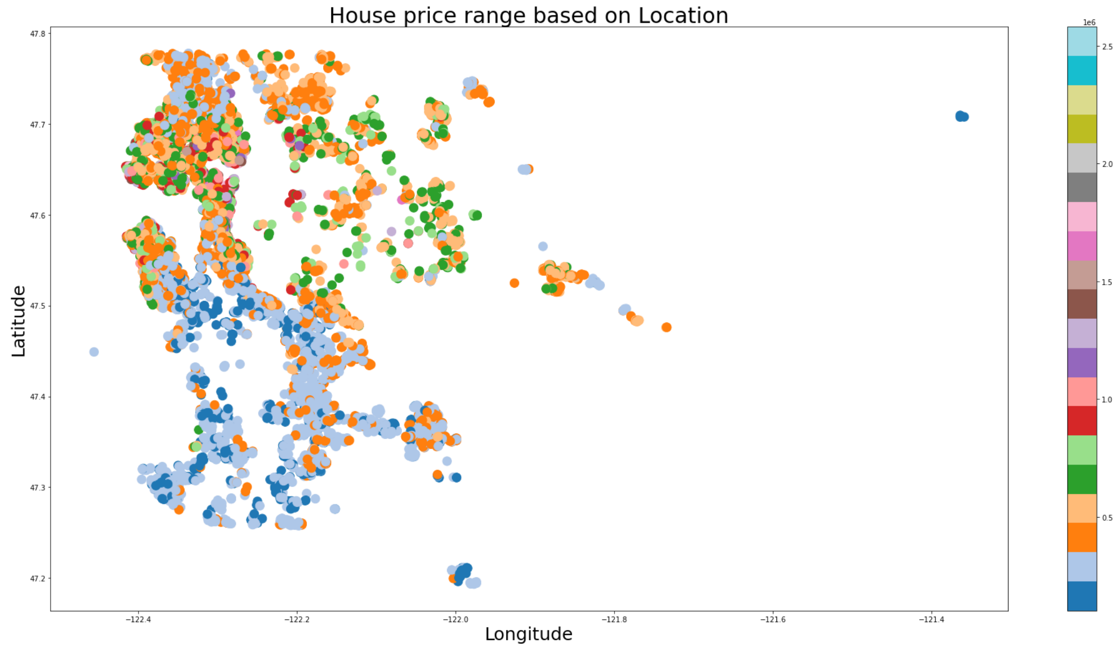




In [87]:

```
#Visualizing Longitude to Latitude to check how the price vary by loca

plt.figure(figsize=(30,15))
plt.scatter(x=df['long'], y=df['lat'], c =df["price"], cmap='tab20',ma
plt.title("House price range based on Location", fontsize=30)
plt.xlabel('Longitude', fontsize=25)
plt.ylabel("Latitude", fontsize=25)
plt.colorbar()
plt.show()
# #visualize relationships between numeric columns
#sns.pairplot(df)
```



In [ ]:

In [88]:

```
#check for multicollinearity between other variables
df.corr()
```

Out[88]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
price	1.000000	0.217521	0.317398	0.524076	-0.121710	0.200429
bedrooms	0.217521	1.000000	0.459538	0.620795	0.196718	0.161048
bathrooms	0.317398	0.459538	1.000000	0.674968	-0.135715	0.545188
sqft_living	0.524076	0.620795	0.674968	1.000000	0.144007	0.317505
sqft_lot	-0.121710	0.196718	-0.135715	0.144007	1.000000	-0.460450
floors	0.200429	0.161048	0.545188	0.317505	-0.460450	1.000000
grade	0.545136	0.273676	0.578527	0.596887	-0.163530	0.500605
sqft_above	0.367155	0.513461	0.611607	0.821877	0.117604	0.511201

sqft_basement	0.307881	0.241146	0.179640	0.398600	0.058932	-0.267215
yr_built	-0.148666	0.122999	0.546293	0.251041	-0.168738	0.541077
zipcode	0.182910	-0.157813	-0.240871	-0.174772	-0.173527	-0.118175
lat	0.449232	-0.126219	-0.090225	-0.057829	-0.216380	0.007585
long	-0.147646	0.150634	0.267481	0.236029	0.164471	0.145409
sqft_living15	0.390753	0.374412	0.473042	0.670969	0.147843	0.241927
sqft_lot15	-0.150082	0.187530	-0.148948	0.121196	0.824229	-0.485805
waterfront1	0.091737	-0.013388	0.013778	0.020841	-0.008168	0.014946
condition1	0.117260	0.050640	-0.032313	0.022928	0.043133	-0.136871
renovated	0.135027	0.018854	0.034931	0.054396	-0.000545	-0.015384

In [89]:

```
#set 0.75 high correlaion as a cut-off
abs(df.corr()) >0.75
```

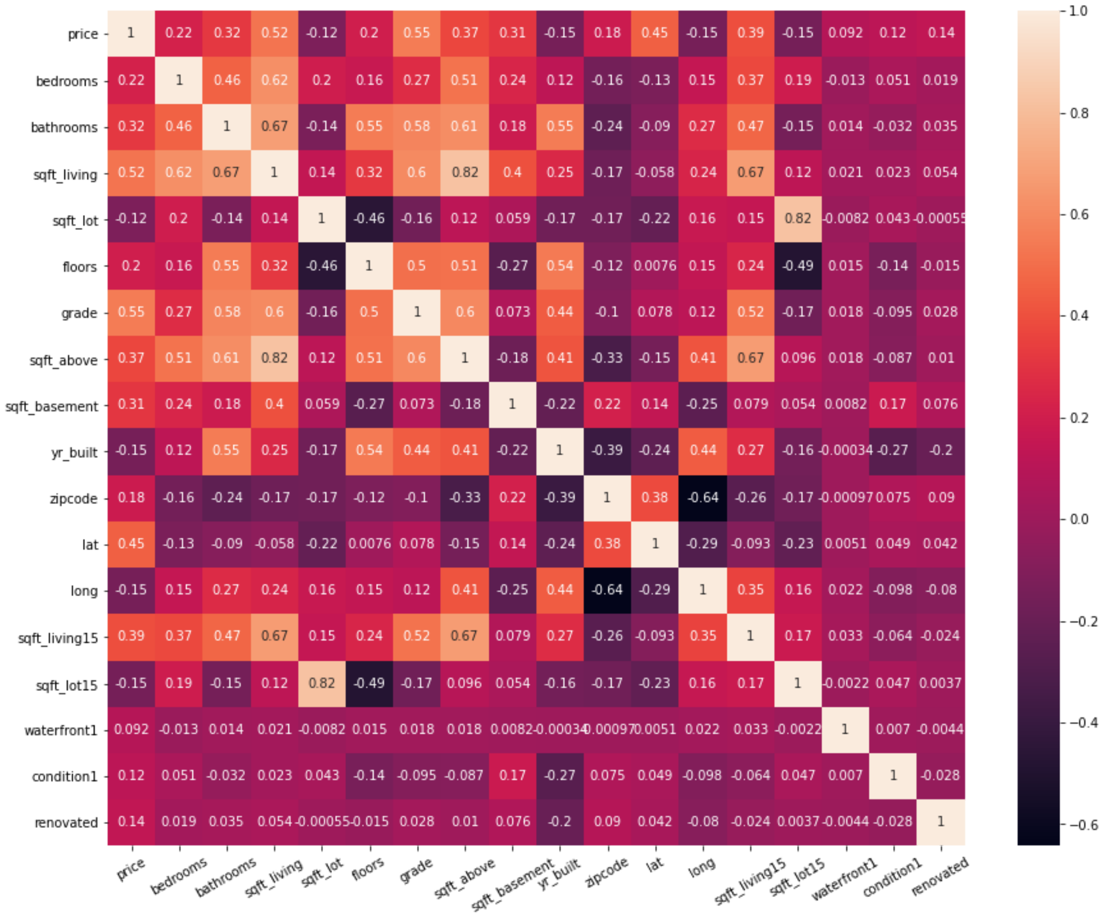
Out[89]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	grade	sqf
price	True	False	False	False	False	False	False	
bedrooms	False	True	False	False	False	False	False	
bathrooms	False	False	True	False	False	False	False	
sqft_living	False	False	False	True	False	False	False	
sqft_lot	False	False	False	False	True	False	False	
floors	False	False	False	False	False	True	False	
grade	False	False	False	False	False	False	True	
sqft_above	False	False	False	True	False	False	False	
sqft_basement	False	False	False	False	False	False	False	
yr_built	False	False	False	False	False	False	False	
zipcode	False	False	False	False	False	False	False	
lat	False	False	False	False	False	False	False	
long	False	False	False	False	False	False	False	
sqft_living15	False	False	False	False	False	False	False	
sqft_lot15	False	False	False	False	True	False	False	
waterfront1	False	False	False	False	False	False	False	
condition1	False	False	False	False	False	False	False	
renovated	False	False	False	False	False	False	False	

In [90]:

```
# visualize correlations between numeric columns to check if there is
plt.figure(figsize=(15,12))
```

```
plt.figure(figsize=(15,12))
ax = sns.heatmap(df.corr(),annot=True)
plt.xticks(rotation=30)
plt.show()
```



In [91]:

```
# save absolute value of correlation matrix as a data frame
# converts all values to absolute value
# stacks the row:column pairs into a multindex
# reset the index to set the multindex to seperate columns
# sort values. 0 is the column automatically generated by the stacking

df3=df.corr().abs().stack().reset_index().sort_values(0, ascending=Fal

# zip the variable name columns (Which were only named level_0 and lev
df3['pairs'] = list(zip(df3.level_0, df3.level_1))

# set index to pairs
df3.set_index(['pairs'], inplace = True)

#d rop level columns
df3.drop(columns=['level_1', 'level_0'], inplace = True)

# rename correlation column as cc rather than 0
df3.columns = ['cc']

# drop duplicates.
df3.drop_duplicates(inplace=True)
df3.head(10)
```

Out[91]: cc

pairs	
(price, price)	1.000000
(sqft_lot, sqft_lot15)	0.824229
(sqft_living, sqft_above)	0.821877
(sqft_living, bathrooms)	0.674968
(sqft_living15, sqft_above)	0.672134
(sqft_living15, sqft_living)	0.670969
(long, zipcode)	0.642847
(sqft_living, bedrooms)	0.620795
(sqft_above, bathrooms)	0.611607
(grade, sqft_living)	0.596887

In [92]:

```
df3[(df3.cc>.75) & (df3.cc <1)]
#assingning the range for unwanted correlation
```

Out[92]:

	cc
pairs	
(sqft_lot, sqft_lot15)	0.824229
(sqft_living, sqft_above)	0.821877

In [93]:

```
df = df.drop(["sqft_lot15","sqft_above"],axis =1)

# drop columns that cause high correlation so won't mess up my model
# for sqft_lot, sqft_lot15: i dropped sqft_lot15 because it makes more
#land lots of the nearest 15 neighbors
# for sqft_above, sqft_living: i dropped the sqft_above because the sq
#living space is more important than the qft_above basement
```

In [94]:

```
df.corr()
```

Out[94]:

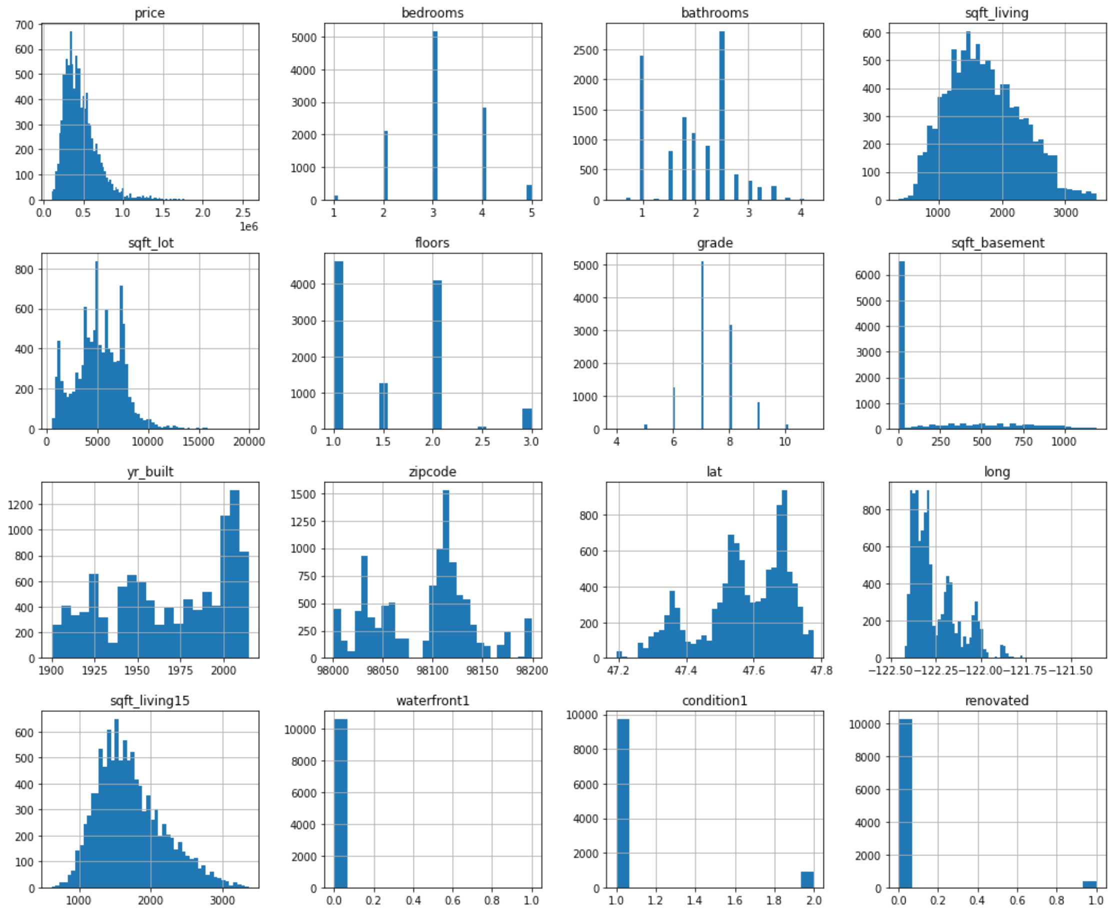
	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
price	1.000000	0.217521	0.317398	0.524076	-0.121710	0.200429
bedrooms	0.217521	1.000000	0.459538	0.620795	0.196718	0.161048
bathrooms	0.317398	0.459538	1.000000	0.674968	-0.135715	0.545188
sqft_living	0.524076	0.620795	0.674968	1.000000	0.144007	0.317505
sqft_lot	-0.121710	0.196718	-0.135715	0.144007	1.000000	-0.460450
floors	0.200429	0.161048	0.545188	0.317505	-0.460450	1.000000
grade	0.545136	0.273676	0.578527	0.596887	-0.163530	0.500605
sqft_basement	0.307881	0.241146	0.179640	0.398600	0.058932	-0.267215
yr_built	0.148666	0.122999	0.546293	0.251041	0.168738	0.541077

yr_built	-0.148666	0.122999	0.546293	0.251041	-0.168738	0.541077
zipcode	0.182910	-0.157813	-0.240871	-0.174772	-0.173527	-0.118175
lat	0.449232	-0.126219	-0.090225	-0.057829	-0.216380	0.007585
long	-0.147646	0.150634	0.267481	0.236029	0.164471	0.145409
sqft_living15	0.390753	0.374412	0.473042	0.670969	0.147843	0.241927
waterfront1	0.091737	-0.013388	0.013778	0.020841	-0.008168	0.014946
condition1	0.117260	0.050640	-0.032313	0.022928	0.043133	-0.136871
renovated	0.135027	0.018854	0.034931	0.054396	-0.000545	-0.015384

In [95]:

```
# check how our histograms are looking

df.hist(figsize=(18,15), bins='auto');
```



Model 1

Baseline Model

In [96]:

```
# set  X and y
X = df.drop('price', axis=1)
y = df['price']
```

In [97]:

```
# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
```

In [98]:

```
from statsmodels.formula.api import ols
```

In [99]:

```
#X_train["bedrooms"]
```

In [100…

```
#model = sm.OLS(y_train,sm.add_constant(X_train["bedrooms"],["bathroom
#model.summary()
#X_train, X_test, y_train, y_test
```

In [101…

```
X_train_base = add_constant(X_train[["bedrooms","bathrooms"]])
X_test_base = add_constant(X_test[["bedrooms","bathrooms"]])
model = sm.OLS(y_train, X_train_base).fit()
model.summary()
```

Out[101…

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.110
Model:	OLS	**Adj. R-squared:**	0.110
Method:	Least Squares	**F-statistic:**	523.6
Date:	Thu, 21 Apr 2022	**Prob (F-statistic):**	3.58e-215
Time:	10:25:43	**Log-Likelihood:**	-1.1588e+05
No. Observations:	8492	**AIC:**	2.318e+05
Df Residuals:	8489	**BIC:**	2.318e+05
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.156e+05	9210.246	23.410	0.000	1.98e+05	2.34e+05
bedrooms	2.504e+04	3077.336	8.136	0.000	1.9e+04	3.11e+04
bathrooms	8.708e+04	3630.444	23.986	0.000	8e+04	9.42e+04

Omnibus:	2467.345	**Durbin-Watson:**	2.025
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	8375.044
Skew:	1.454	**Prob(JB):**	0.00
Kurtosis:	6.900	**Cond. No.**	16.9

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly

specified.

In [102…

```
y_pred = model.predict(X_test_base)
error = y_pred - y_test  #error
abs_error = abs(y_pred - y_test)  # absolute error
mean_abs_error = abs_error.mean()
squared_error = error**2
mean_squared_error = squared_error.mean()
rmse = mean_squared_error ** 0.5

print('BASELINE  :','MAE:',mean_abs_error, 'MSE:', mean_squared_error,
```

```
BASELINE  : MAE: 155574.75782023964 MSE: 43250156697.37297 RMSE: 20796
6.72016785035
```

MODEL 2.

Including all features

In [103…

```
X_train_all = add_constant(X_train)
X_test_all = add_constant(X_test)
model_all = sm.OLS(y_train, X_train_all).fit()
print(model_all.summary())
y_pred_all=model_all.predict(X_test_all)

error_all = y_pred_all - y_test
squared_error_all = error_all**2
rmse_all = squared_error_all.mean() ** 0.5
mean_abs_error = abs_error.mean()


print('RMSE_all:', rmse_all, "MAE:" ,mean_abs_error)
```

```
                            OLS Regression Results
==============================================================================
========
Dep. Variable:                  price   R-squared:
0.678
Model:                            OLS   Adj. R-squared:
0.677
Method:                 Least Squares   F-statistic:
1189.
Date:                Thu, 21 Apr 2022   Prob (F-statistic):
0.00
Time:                        10:25:43   Log-Likelihood:            -1.
1156e+05
No. Observations:                8492   AIC:
2.232e+05
Df Residuals:                    8476   BIC:
2.233e+05
Df Model:                          15
Covariance Type:            nonrobust
==============================================================================
===========
                    coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----------
```

```
const          -3.198e+07   3.19e+06    -10.009      0.000   -3.82e+07
-2.57e+07
bedrooms       -1.825e+04   2169.057     -8.414      0.000   -2.25e+04
-1.4e+04
bathrooms       1.976e+04   3403.243      5.806      0.000    1.31e+04
2.64e+04
sqft_living      126.3621      5.111     24.726      0.000     116.344
136.380
sqft_lot          -6.8310      0.735     -9.299      0.000      -8.271
-5.391
floors          1.076e+04   3926.657      2.740      0.006    3062.558
1.85e+04
grade           9.515e+04   2272.029     41.880      0.000    9.07e+04
9.96e+04
sqft_basement     -2.3190      5.806     -0.399      0.690     -13.699
9.061
yr_built       -2306.1684     61.523    -37.485      0.000   -2426.769
-2185.568
zipcode          -12.3531     37.603     -0.329      0.743     -86.064
61.358
lat             5.379e+05   1.18e+04     45.418      0.000    5.15e+05
5.61e+05
long           -9.483e+04   1.52e+04     -6.244      0.000   -1.25e+05
-6.51e+04
sqft_living15     52.1525      4.223     12.350      0.000      43.874
60.431
waterfront1     6.571e+05   6.15e+04     10.680      0.000    5.37e+05
7.78e+05
condition1      3.982e+04   5045.897      7.891      0.000    2.99e+04
4.97e+04
renovated       1.922e+04   7644.004      2.514      0.012    4232.692
3.42e+04
======================================================================
========
Omnibus:                     2338.525   Durbin-Watson:
2.013
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            1
3360.938
Skew:                           1.198   Prob(JB):
0.00
Kurtosis:                       8.658   Cond. No.
2.35e+08
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 2.35e+08. This might indicate that
there are
strong multicollinearity or other numerical problems.
RMSE_all: 127050.13208070699 MAE: 155574.75782023964
```

Dedected problems

In [104…

```
# Detected Problems:
# 1 - Variable type : Numerical / Categorical / Ordinal
```

```
# 2 - Multicollinearity - there are the binaries you identified above
# take only 1 of every 2 out there and use it
# 3 - Distributions of variables (dependent-independent) - try taking
#- there are 2 important issues here
#- A -first you will take logarithm of price and try-
# - B - then just take the asymmetric variables and move forward by ta
# there will be at least 6 models in total - their comparison is rmse
```

MODEL 2.A

PROBLEM 1 Categorical-Numerical

In [105…

```
# Problem 1
# Variable Types 1: Nominal, Ordinal, Interval, Ratio
# Variable Types 2: Categorical, Numerical

df.dtypes
df.describe()
# 1- Zipcode (omit or make dummy vars)
#2- yr_built: calculate age
#3- basement-dummify
```

Out[105…

	price	bedrooms	bathrooms	sqft_living	sqft_lot	
count	1.061500e+04	10615.000000	10615.000000	10615.000000	10615.000000	1061
mean	4.666713e+05	3.123787	1.976613	1731.258691	5256.174282	
std	2.169005e+05	0.812062	0.687662	578.651917	2385.108910	
min	1.025000e+05	1.000000	0.500000	370.000000	520.000000	
25%	3.150000e+05	3.000000	1.500000	1290.000000	3800.000000	
50%	4.250000e+05	3.000000	2.000000	1670.000000	5120.000000	
75%	5.659985e+05	4.000000	2.500000	2130.000000	7000.000000	
max	2.580000e+06	5.000000	4.250000	3490.000000	19969.000000	

In [106…

```
X_train_step1 = X_train.drop(columns = ['zipcode'])
X_train_step1['built_age'] = 2022 - X_train_step1.yr_built
X_train_step1['basement_dummy'] = np.where(X_train_step1.sqft_basement

X_train_step1 = X_train_step1.drop(columns = ['yr_built','sqft_basemen

X_test_step1 = X_test.drop(columns = ['zipcode'])
X_test_step1['built_age'] = 2022 - X_test_step1.yr_built
X_test_step1['basement_dummy'] = np.where(X_test_step1.sqft_basement >

X_test_step1 = X_test_step1.drop(columns = ['yr_built','sqft_basement'
```

In [107…

```
X_train_step1 = add_constant(X_train_step1)
X_test_step1 = add_constant(X_test_step1)
model_step1 = sm.OLS(y_train, X_train_step1).fit()
```

```
print(model_step1.summary())
y_pred_step1=model_step1.predict(X_test_step1)


error_step1 = y_pred_step1 - y_test
squared_error_step1 = error_step1**2
rmse_step1 = squared_error_step1.mean() ** 0.5
mean_abs_error = abs_error.mean()

print('RMSE_all:', rmse_step1, "MAE:", mean_abs_error)
```

```
                            OLS Regression Results
==============================================================================
========
Dep. Variable:                  price   R-squared:
0.678
Model:                            OLS   Adj. R-squared:
0.678
Method:                 Least Squares   F-statistic:
1276.
Date:                Thu, 21 Apr 2022   Prob (F-statistic):
0.00
Time:                        10:25:43   Log-Likelihood:               -1.
1156e+05
No. Observations:                8492   AIC:
2.231e+05
Df Residuals:                    8477   BIC:
2.233e+05
Df Model:                          14
Covariance Type:            nonrobust
==============================================================================
============
                    coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------------
const         -3.601e+07   1.67e+06    -21.625      0.000   -3.93e+07
-3.27e+07
bedrooms      -1.794e+04   2165.786     -8.283      0.000   -2.22e+04
-1.37e+04
bathrooms      1.797e+04   3400.501      5.284      0.000    1.13e+04
2.46e+04
sqft_living     121.7257      4.668     26.075      0.000     112.575
130.877
sqft_lot         -6.2516      0.735     -8.510      0.000      -7.692
-4.812
floors         1.578e+04   3780.314      4.175      0.000    8371.347
2.32e+04
grade          9.532e+04   2262.009     42.139      0.000    9.09e+04
9.98e+04
lat            5.343e+05   1.16e+04     46.235      0.000    5.12e+05
5.57e+05
long          -8.108e+04   1.33e+04     -6.080      0.000   -1.07e+05
-5.49e+04
sqft_living15    53.5419      4.176     12.820      0.000      45.355
61.729
waterfront1    6.564e+05   6.15e+04     10.673      0.000    5.36e+05
7.77e+05
condition1      3.98e+04   5033.541      7.906      0.000    2.99e+04
4.97e+04
```

```
renovated        1.979e+04   7640.356      2.590      0.010     4814.894
3.48e+04
built_age        2302.6528     61.293     37.568      0.000     2182.504
2422.801
basement_dummy   9275.2288   3427.026      2.706      0.007     2557.423
1.6e+04
======================================================================
========
Omnibus:                     2345.423   Durbin-Watson:
2.012
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              1
3399.623
Skew:                           1.202   Prob(JB):
0.00
Kurtosis:                       8.665   Cond. No.
7.77e+06
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 7.77e+06. This might indicate that
there are
strong multicollinearity or other numerical problems.
RMSE_all: 126907.97412298272 MAE: 155574.75782023964
```

2.A.1 Second trial one-hot encoding(dummifyying zipcodes)

In [108…

```
# Step1 Second trial
# One-hot encoding (dummifying zipcodes)
# set  X and y

zipcode_onehot = pd.get_dummies(df.zipcode) #Zipcode ( make dummy vari
z_colnames = ['zip_'+ str(i) for i in zipcode_onehot.columns] #One-hot
zipcode_onehot.columns = z_colnames
df_onehot = df.join(zipcode_onehot)

df_onehot = df_onehot.drop(columns = ['zipcode'])
df_onehot['built_age'] = 2022 - df_onehot.yr_built
df_onehot['basement_dummy'] = np.where(df_onehot.sqft_basement > 0 ,1
df_onehot = df_onehot.drop(columns = ['yr_built','sqft_basement'])

X_oh = df_onehot.drop('price', axis=1)
y_oh = df_onehot['price']
X_train_oh, X_test_oh, y_train_oh, y_test_oh = train_test_split(X_oh,
weird_zipcodes = [i for i in X_oh.columns if (X_train_oh[i].sum() == 0
X_train_oh = X_train_oh.drop(columns = weird_zipcodes)
X_test_oh = X_test_oh.drop(columns = weird_zipcodes)


X_train_oh = add_constant(X_train_oh)
X_test_oh = add_constant(X_test_oh)
model_step1_oh = sm.OLS(y_train_oh, X_train_oh).fit()
print(model_step1_oh.summary())
y_pred_step_oh=model_step1_oh.predict(X_test_oh)
```

```
error_step_oh = y_pred_step_oh - y_test
squared_error_step_oh = error_step_oh**2
rmse_step_oh = squared_error_step_oh.mean() ** 0.5
mean_abs_error = abs_error.mean()

print('RMSE_all:', rmse_step_oh,'MAE:', mean_abs_error)
```

```
                            OLS Regression Results
======================================================================
========
Dep. Variable:                  price   R-squared:
0.788
Model:                            OLS   Adj. R-squared:
0.786
Method:                 Least Squares   F-statistic:
391.6
Date:                Thu, 21 Apr 2022   Prob (F-statistic):
0.00
Time:                        10:25:44   Log-Likelihood:             -1.
0978e+05
No. Observations:                8492   AIC:
2.197e+05
Df Residuals:                    8411   BIC:
2.203e+05
Df Model:                          80
Covariance Type:            nonrobust
======================================================================
============
                     coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
------------
const            2.04e+07   8.95e+06      2.280      0.023    2.86e+06
3.79e+07
bedrooms       -9073.5270   1799.560     -5.042      0.000   -1.26e+04
-5545.946
bathrooms       1.394e+04   2796.392      4.984      0.000    8455.188
1.94e+04
sqft_living      133.0963      3.882     34.289      0.000     125.487
140.705
sqft_lot           7.0672      0.664     10.641      0.000       5.765
8.369
floors         -5687.1039   3192.786     -1.781      0.075   -1.19e+04
571.541
grade           6.364e+04   1965.723     32.374      0.000    5.98e+04
6.75e+04
lat            -1.214e+05     7.8e+04     -1.557      0.120   -2.74e+05
3.15e+04
long            1.218e+05   6.76e+04      1.802      0.072   -1.07e+04
2.54e+05
sqft_living15     39.0473      3.545     11.015      0.000      32.099
45.996
waterfront1     7.216e+05   5.04e+04     14.328      0.000    6.23e+05
8.2e+05
condition1       4.01e+04   4140.043      9.685      0.000     3.2e+04
4.82e+04
renovated       2.817e+04   6279.228      4.486      0.000    1.59e+04
4.05e+04
zip_98001      -3.827e+05   4.15e+04     -9.221      0.000   -4.64e+05
-3.01e+05
```

```
zip_98002      -3.444e+05   4.18e+04     -8.234      0.000   -4.26e+05
-2.62e+05
zip_98003      -3.803e+05   4.21e+04     -9.029      0.000   -4.63e+05
-2.98e+05
zip_98004       1.866e+05   4.26e+04      4.377      0.000    1.03e+05
2.7e+05
zip_98005      -1.655e+04   4.56e+04     -0.363      0.717   -1.06e+05
7.28e+04
zip_98006      -1.725e+05    4.4e+04     -3.918      0.000   -2.59e+05
-8.62e+04
zip_98007      -1.399e+05    4.1e+04     -3.411      0.001    -2.2e+05
-5.95e+04
zip_98008      -1.343e+05   3.92e+04     -3.427      0.001   -2.11e+05
-5.75e+04
zip_98010      -3.488e+05    5.5e+04     -6.337      0.000   -4.57e+05
-2.41e+05
zip_98011      -1.896e+05   4.15e+04     -4.574      0.000   -2.71e+05
-1.08e+05
zip_98014      -2.983e+05   6.35e+04     -4.695      0.000   -4.23e+05
-1.74e+05
zip_98019      -2.786e+05   4.54e+04     -6.142      0.000   -3.68e+05
-1.9e+05
zip_98022      -4.153e+05   5.13e+04     -8.091      0.000   -5.16e+05
-3.15e+05
zip_98023      -3.803e+05   4.25e+04     -8.938      0.000   -4.64e+05
-2.97e+05
zip_98024      -3.197e+05   1.09e+05     -2.939      0.003   -5.33e+05
-1.07e+05
zip_98027      -1.393e+05   3.94e+04     -3.534      0.000   -2.17e+05
-6.2e+04
zip_98028      -2.056e+05   4.07e+04     -5.052      0.000   -2.85e+05
-1.26e+05
zip_98029       -1.64e+05   4.04e+04     -4.057      0.000   -2.43e+05
-8.47e+04
zip_98030      -3.853e+05   4.01e+04     -9.613      0.000   -4.64e+05
-3.07e+05
zip_98031      -3.719e+05   3.91e+04     -9.509      0.000   -4.49e+05
-2.95e+05
zip_98032      -3.504e+05   4.19e+04     -8.370      0.000   -4.32e+05
-2.68e+05
zip_98033       2055.1756   3.85e+04      0.053      0.957   -7.35e+04
7.76e+04
zip_98034      -1.549e+05   3.84e+04     -4.038      0.000    -2.3e+05
-7.97e+04
zip_98038      -3.625e+05   4.21e+04     -8.607      0.000   -4.45e+05
-2.8e+05
zip_98042      -3.645e+05   4.05e+04     -8.994      0.000   -4.44e+05
-2.85e+05
zip_98045      -3.615e+05   5.58e+04     -6.481      0.000   -4.71e+05
-2.52e+05
zip_98052      -1.244e+05    3.9e+04     -3.192      0.001   -2.01e+05
-4.8e+04
zip_98053      -1.067e+05   4.14e+04     -2.576      0.010   -1.88e+05
-2.55e+04
zip_98055      -3.082e+05   3.77e+04     -8.163      0.000   -3.82e+05
-2.34e+05
zip_98056      -2.618e+05   3.71e+04     -7.060      0.000   -3.34e+05
-1.89e+05
zip_98058      -3.541e+05   3.83e+04     -9.250      0.000   -4.29e+05
-2.79e+05
```

```
zip_98059       -2.83e+05   3.79e+04     -7.461      0.000   -3.57e+05
-2.09e+05
zip_98065      -2.702e+05   4.54e+04     -5.957      0.000   -3.59e+05
-1.81e+05
zip_98072       -1.95e+05   4.19e+04     -4.656      0.000   -2.77e+05
-1.13e+05
zip_98074      -1.818e+05    4.1e+04     -4.430      0.000   -2.62e+05
-1.01e+05
zip_98075      -1.595e+05   4.22e+04     -3.779      0.000   -2.42e+05
-7.68e+04
zip_98092      -4.194e+05   4.21e+04     -9.973      0.000   -5.02e+05
-3.37e+05
zip_98102       1.209e+05   3.81e+04      3.175      0.002    4.63e+04
1.96e+05
zip_98103       1820.6791   3.72e+04      0.049      0.961   -7.11e+04
7.47e+04
zip_98105       7.358e+04   3.74e+04      1.967      0.049     256.339
1.47e+05
zip_98106      -2.021e+05   3.66e+04     -5.513      0.000   -2.74e+05
-1.3e+05
zip_98107       1.026e+04   3.78e+04      0.271      0.786   -6.38e+04
8.44e+04
zip_98108      -2.274e+05   3.66e+04     -6.205      0.000   -2.99e+05
-1.56e+05
zip_98109       1.345e+05   3.81e+04      3.530      0.000    5.98e+04
2.09e+05
zip_98112       1.453e+05   3.68e+04      3.945      0.000    7.31e+04
2.17e+05
zip_98115         -1.4e+04   3.71e+04     -0.378      0.706   -8.66e+04
5.86e+04
zip_98116      -2.806e+04    3.7e+04     -0.758      0.448   -1.01e+05
4.45e+04
zip_98117      -7331.0780   3.77e+04     -0.194      0.846   -8.13e+04
6.66e+04
zip_98118      -1.798e+05   3.59e+04     -5.013      0.000    -2.5e+05
-1.1e+05
zip_98119       1.255e+05   3.76e+04      3.336      0.001    5.17e+04
1.99e+05
zip_98122      -1.966e+04   3.64e+04     -0.541      0.589   -9.09e+04
5.16e+04
zip_98125      -1.313e+05    3.8e+04     -3.451      0.001   -2.06e+05
-5.67e+04
zip_98126      -1.305e+05   3.67e+04     -3.555      0.000   -2.02e+05
-5.85e+04
zip_98133      -1.578e+05   3.85e+04     -4.100      0.000   -2.33e+05
-8.23e+04
zip_98136        -7.89e+04   3.72e+04     -2.123      0.034   -1.52e+05
-6044.633
zip_98144        -7.43e+04   3.62e+04     -2.053      0.040   -1.45e+05
-3353.279
zip_98146      -2.201e+05   3.76e+04     -5.855      0.000   -2.94e+05
-1.46e+05
zip_98148      -2.933e+05   5.13e+04     -5.723      0.000   -3.94e+05
-1.93e+05
zip_98155      -1.813e+05   3.97e+04     -4.561      0.000   -2.59e+05
-1.03e+05
zip_98166      -2.472e+05   3.98e+04     -6.215      0.000   -3.25e+05
-1.69e+05
zip_98168      -2.685e+05   3.81e+04     -7.047      0.000   -3.43e+05
-1.94e+05
```

```
zip_98177      -1.208e+05   4.06e+04     -2.979       0.003       -2e+05
-4.13e+04
zip_98178      -2.957e+05   3.69e+04     -8.012       0.000    -3.68e+05
-2.23e+05
zip_98188      -3.533e+05   4.73e+04     -7.464       0.000    -4.46e+05
-2.61e+05
zip_98198      -3.218e+05   3.97e+04     -8.098       0.000       -4e+05
-2.44e+05
zip_98199       3.981e+04   3.77e+04      1.056       0.291    -3.41e+04
1.14e+05
built_age        942.8598     57.209     16.481       0.000      830.716
1055.003
basement_dummy -1.574e+04   2902.436     -5.425       0.000    -2.14e+04
-1.01e+04
======================================================================
========
Omnibus:                     2865.157   Durbin-Watson:
2.007
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            2
7795.095
Skew:                           1.335   Prob(JB):
0.00
Kurtosis:                      11.451   Cond. No.
5.13e+07
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 5.13e+07. This might indicate that
there are
strong multicollinearity or other numerical problems.
RMSE_all: 105758.8389997831 MAE: 155574.75782023964
```

PROBLEM 2.B

MULTICOLLINEARITY

In [109…

```
# Trying to solve problem 2 - Multicollinearity
# pairs
# (price, price)        1.000000
# (log_price, price)    0.949407
# (sqft_living, bathrooms)      0.674968
# (sqft_living15, sqft_living)  0.670969
# (long, zipcode)       0.642847
# (sqft_living, bedrooms)       0.620795
# (grade, sqft_living)  0.596887
# (grade, bathrooms)    0.578527
# (log_price, grade)    0.546385
# (yr_built, bathrooms) 0.546293
```

In [110…

```
# log_y_train = np.log(y_train)
# log_y_test = np.log( y_test)
```

In [111…

```
# model_step2 = sm.OLS(log_y_train, X_train_step2).fit()
# print(model_step2.summary())

# y_pred_step2 = model_step2.predict(X_test_step2)
# error_step2 = y_pred_step2 - log_y_test


# # removing the log transformation
# y_pred_step2_transformed = np.exp(y_pred_step2)
# error_step2_transformed = y_pred_step2_transformed - y_test
# squared_error_step2 = error_step2_transformed**2
# rmse_step2 = squared_error_step2.mean() ** 0.5
# mean_abs_error = abs_error.mean()

# # print(y_pred_step2)
# # print(y_pred_step2_transformed)
# print('RMSE_all:', rmse_step3, 'MAE:', mean_abs_error)
```

In [112…

```
X_train_step2 = X_train_step1.drop(columns = [ 'sqft_living15', 'sqft_
X_test_step2 = X_test_step1.drop(columns = ['sqft_living15', 'sqft_lot
```

In [113…

```
X_train_step2 = add_constant(X_train_step2)
X_test_step2 = add_constant(X_test_step2)
model_step2 = sm.OLS(y_train, X_train_step2).fit()
print(model_step2.summary())
y_pred_step2=model_step2.predict(X_test_step2)


error_step2 = y_pred_step2 - y_test
squared_error_step2 = error_step2**2
rmse_step2 = squared_error_step2.mean() ** 0.5
mean_abs_error = abs_error.mean()

print('RMSE_all:', rmse_step2, 'MAE:', mean_abs_error)
```

```
                            OLS Regression Results
======================================================================
========
Dep. Variable:                  price   R-squared:
0.669
Model:                            OLS   Adj. R-squared:
0.669
Method:                 Least Squares   F-statistic:
1714.
Date:                Thu, 21 Apr 2022   Prob (F-statistic):
0.00
Time:                        10:25:44   Log-Likelihood:           -1.
1168e+05
No. Observations:                8492   AIC:
2.234e+05
Df Residuals:                    8481   BIC:
2.235e+05
Df Model:                          10
Covariance Type:            nonrobust
======================================================================
============
                   coef    std err          t      P>|t|      [0.025
```

```
0.975]
------------------------------------------------------------------------------
------------
const          -2.682e+07   5.45e+05    -49.251      0.000   -2.79e+07
-2.58e+07
bedrooms        -2.12e+04   2176.534     -9.738      0.000   -2.55e+04
-1.69e+04
bathrooms        2.03e+04   3401.854      5.967      0.000    1.36e+04
2.7e+04
sqft_living      132.5160      4.039     32.810      0.000     124.599
140.433
floors          3.041e+04   3303.436      9.206      0.000    2.39e+04
3.69e+04
grade           1.043e+05   2203.775     47.332      0.000       1e+05
1.09e+05
lat             5.485e+05   1.15e+04     47.617      0.000    5.26e+05
5.71e+05
waterfront1     6.694e+05   6.23e+04     10.740      0.000    5.47e+05
7.92e+05
condition1      3.576e+04   5056.226      7.072      0.000    2.58e+04
4.57e+04
built_age       2456.5121     55.710     44.095      0.000    2347.307
2565.718
basement_dummy  1.592e+04   3160.466      5.036      0.000    9720.303
2.21e+04
==============================================================================
========
Omnibus:                     2297.665   Durbin-Watson:
2.016
Prob(Omnibus):                  0.000   Jarque-Bera (JB):                1
2968.533
Skew:                           1.179   Prob(JB):
0.00
Kurtosis:                       8.576   Cond. No.
7.36e+05
==============================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 7.36e+05. This might indicate that
there are
strong multicollinearity or other numerical problems.
RMSE_all: 128545.69942046574 MAE: 155574.75782023964
```

In [114…

```
# # Step1 Second trial
# # One-hot encoding (dummifying zipcodes)
# # set  X and y

# zipcode_onehot = pd.get_dummies(df.zipcode)
# z_colnames = ['zip_'+ str(i) for i in zipcode_onehot.columns]
# zipcode_onehot.columns = z_colnames
# df_onehot = df.join(zipcode_onehot)

# df_onehot = df_onehot.drop(columns = ['zipcode'])
# df_onehot['built_age'] = 2022 - df_onehot.yr_built
# df_onehot['basement_dummy'] = np.where(df_onehot.sqft_basement > 0 ,
```

```
# df_onehot = df_onehot.drop(columns = ['yr_built','sqft_basement'])

# X_oh = df_onehot.drop('price', axis=1)
# y_oh = df_onehot['price']
# X_train_oh, X_test_oh, y_train_oh, y_test_oh = train_test_split(X_oh
# weird_zipcodes = [i for i in X_oh.columns if (X_train_oh[i].sum() ==
# X_train_oh = X_train_oh.drop(columns = weird_zipcodes)
# X_test_oh = X_test_oh.drop(columns = weird_zipcodes)


# X_train_oh = add_constant(X_train_oh)
# X_test_oh = add_constant(X_test_oh)
# model_step1_oh = sm.OLS(y_train_oh, X_train_oh).fit()
# print(model_step1_oh.summary())
# y_pred_step_oh=model_step1_oh.predict(X_test_oh)

# error_step_oh = y_pred_step_oh - y_test
# squared_error_step_oh = error_step_oh**2
# rmse_step_oh = squared_error_step_oh.mean() ** 0.5
# mean_abs_error = abs_error.mean()

# print('RMSE_all:', rmse_step_oh,'MAE:', mean_abs_error)
```

In [ ]:

```

```

PROBLEM 2.C

Distributions of variables (dependent-independent)

log of unsymmetrical variables

In [115…

```
# Solve PROBLEM3
log_y_train = np.log(y_train)
log_y_test = np.log( y_test)
# print(y_train)
# log_y_train
```

In [116…

```
# df['log_price'] = np.log1p(df['price'])
#df['sqft_living15'] = np.log1p(df['sqft_living15'])
#df['sqft_lot'] = np.log1p(df['sqft_lot'])
#df['sqft_living'] = np.log1p(df['sqft_living'])
#df['bathrooms'] = np.log1p(df['bathrooms'])
#df['sqft_living15'] = np.log1p(df['sqft_living15'])
# df.hist(figsize  = [15, 15]);
```

In [117…

```
model_step3 = sm.OLS(log_y_train, X_train_step2).fit()   #xtrain step1
print(model_step3.summary())
y_pred_step3=model_step3.predict(X_test_step2)
error_step3 = y_pred_step3 - log_y_test

# removing the log transformation
y_pred_step3_transformed = np.exp(y_pred_step3)
error_step3_transformed = y_pred_step3_transformed - y_test
```

```
squared_error_step3 = error_step3_transformed**2
rmse_step3 = squared_error_step3.mean() ** 0.5
mean_abs_error = abs_error.mean()

# print(y_pred_step3)
# print(y_pred_step3_transformed)
print('RMSE_all:', rmse_step3, 'MAE:', mean_abs_error)
```

```
                            OLS Regression Results
==============================================================================
========
Dep. Variable:                  price   R-squared:
0.711
Model:                            OLS   Adj. R-squared:
0.711
Method:                 Least Squares   F-statistic:
2087.
Date:                Thu, 21 Apr 2022   Prob (F-statistic):
0.00
Time:                        10:25:44   Log-Likelihood:
295.30
No. Observations:                8492   AIC:
-568.6
Df Residuals:                    8481   BIC:
-491.1
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
============
                    coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------------
const           -59.3206      1.022    -58.030      0.000     -61.324
-57.317
bedrooms         -0.0293      0.004     -7.176      0.000      -0.037
-0.021
bathrooms         0.0430      0.006      6.727      0.000       0.030
0.055
sqft_living       0.0003   7.58e-06     33.882      0.000       0.000
0.000
floors            0.0629      0.006     10.137      0.000       0.051
0.075
grade             0.1898      0.004     45.874      0.000       0.182
0.198
lat               1.4725      0.022     68.103      0.000       1.430
1.515
waterfront1       0.7701      0.117      6.582      0.000       0.541
0.999
condition1        0.0597      0.009      6.287      0.000       0.041
0.078
built_age         0.0041      0.000     38.893      0.000       0.004
0.004
basement_dummy    0.0358      0.006      6.039      0.000       0.024
0.047
==============================================================================
========
Omnibus:                      252.413   Durbin-Watson:
2.007
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
```

```
421.320
Skew:                          -0.267   Prob(JB):
3.25e-92
Kurtosis:                       3.951   Cond. No.
7.36e+05
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 7.36e+05. This might indicate that
there are
strong multicollinearity or other numerical problems.
RMSE_all: 121137.5587617928 MAE: 155574.75782023964
```

In [118…

```
model_step3.summary()
```

Out[118…

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.711
Model:	OLS	**Adj. R-squared:**	0.711
Method:	Least Squares	**F-statistic:**	2087.
Date:	Thu, 21 Apr 2022	**Prob (F-statistic):**	0.00
Time:	10:25:44	**Log-Likelihood:**	295.30
No. Observations:	8492	**AIC:**	-568.6
Df Residuals:	8481	**BIC:**	-491.1
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-59.3206	1.022	-58.030	0.000	-61.324	-57.317
bedrooms	-0.0293	0.004	-7.176	0.000	-0.037	-0.021
bathrooms	0.0430	0.006	6.727	0.000	0.030	0.055
sqft_living	0.0003	7.58e-06	33.882	0.000	0.000	0.000
floors	0.0629	0.006	10.137	0.000	0.051	0.075
grade	0.1898	0.004	45.874	0.000	0.182	0.198
lat	1.4725	0.022	68.103	0.000	1.430	1.515
waterfront1	0.7701	0.117	6.582	0.000	0.541	0.999
condition1	0.0597	0.009	6.287	0.000	0.041	0.078
built_age	0.0041	0.000	38.893	0.000	0.004	0.004
basement_dummy	0.0358	0.006	6.039	0.000	0.024	0.047

Omnibus:	252.413	**Durbin-Watson:**	2.007

Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	421.320
Skew:	-0.267	**Prob(JB):**	3.25e-92
Kurtosis:	3.951	**Cond. No.**	7.36e+05

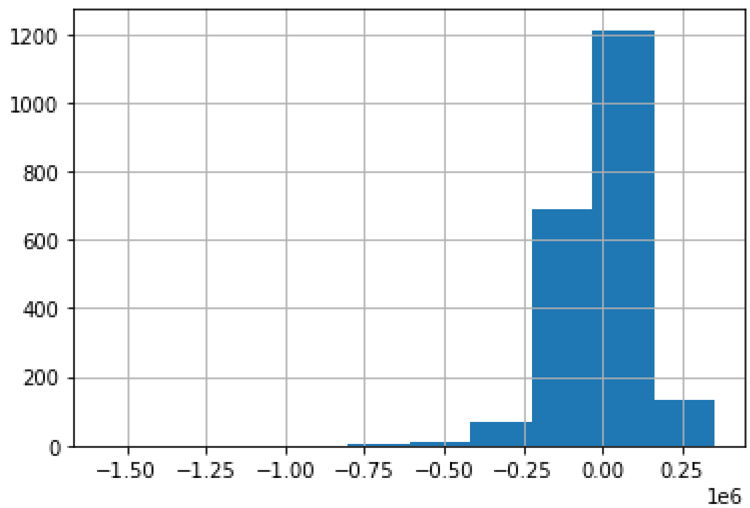
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.36e+05. This might indicate that there are strong multicollinearity or other numerical problems.

In [119…

```
# checking error distributions - assumption
#model 4
error_step2.hist()
```

Out[119…

```
<AxesSubplot:>
```

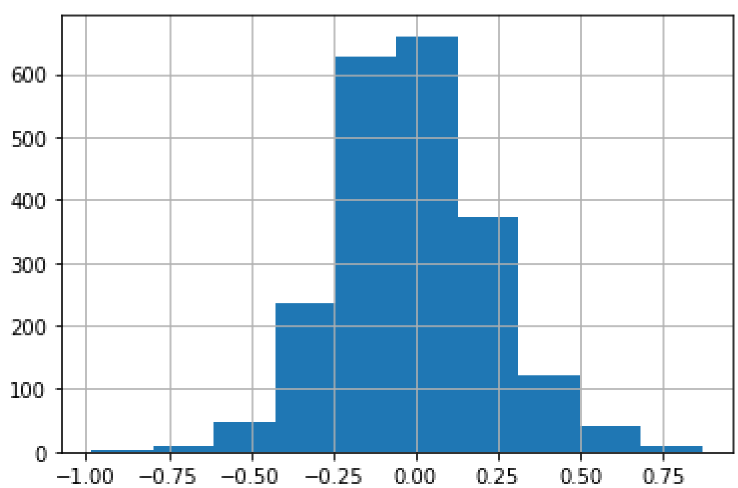


In [120…

```
error_step3.hist()
```

Out[120…

```
<AxesSubplot:>
```



In [121…

```
# Instantiate a scaler
#scaler = StandardScaler()

# train on train data
#scaler.fit(X_train)
# transform both train and test data
#X_train_scaled = scaler.transform(X_train)
#X_test_scaled = scaler.transform(X_test)
```

In [ ]:

In [122…

```
# houses = df.drop(columns=['renovated',"condition1","waterfront1","sq

# outcome = 'price'
# predictors = houses.drop(['price'], axis=1)
# pred_sum = "+".join(predictors.columns)
# formula = outcome + "~" + pred_sum
# mode = ols(formula = formula, data=houses).fit()
# mode.summary()
```

In [123…

```
# mode.predict(houses)
```

In [124…

```
# Instantiate a scaler
scaler = StandardScaler()

# train on train data
scaler.fit(X_train)
# transform both train and test data
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [125…

```
# Instantiate a linear regression model
lr = LinearRegression()
# Fit our model on our scaled data
lr.fit(X_train_scaled, y_train)
```

Out[125…

```
LinearRegression()
```

In [126…

```
y_train_pred2 = lr.predict(X_train_scaled)
y_test_pred2 = lr.predict(X_test_scaled)
```

In [127…

```
# Evaluate
ut.evaluate_model(y_train, y_test, y_train_pred2, y_test_pred2)
```

```
Train R2: 0.678
Test R2: 0.663
---
Train MAE: 88818.999
Test MAE: 89774.669
---
```

```
Train RMSE: 122809.155
Test RMSE: 127050.132
```

In [128…

```
#the baseline model can predict 67 % variance in the price and approxi
#and for root square error we have about $125000 off because root squa
```

In [129…

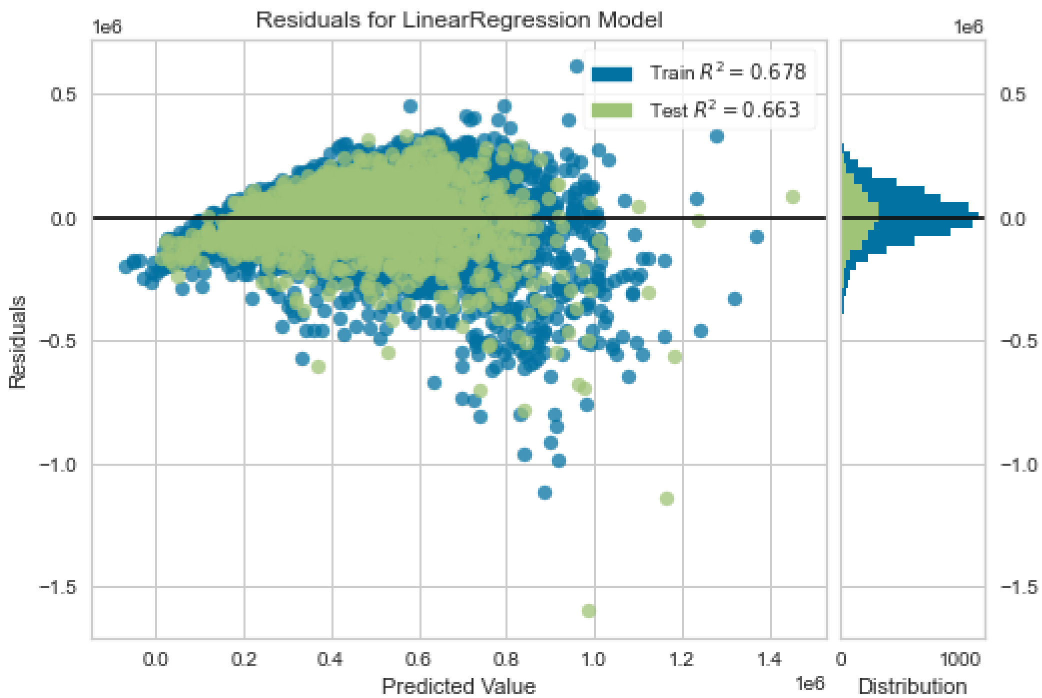
```
# visualizing our residuals
# https://www.scikit-yb.org/en/latest/api/regressor/residuals.html

from yellowbrick.regressor import ResidualsPlot


visualizer = ResidualsPlot(lr)

visualizer.fit(X_train_scaled, y_train)
#fit the traning data to the visualizer

visualizer.score(X_test_scaled, y_test)
#Evaluate the model on the test data
visualizer.show()
```

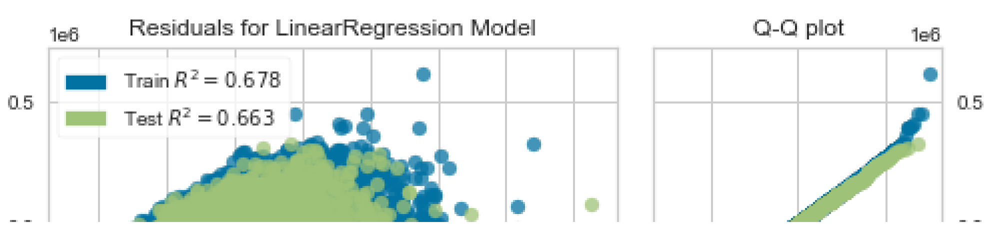


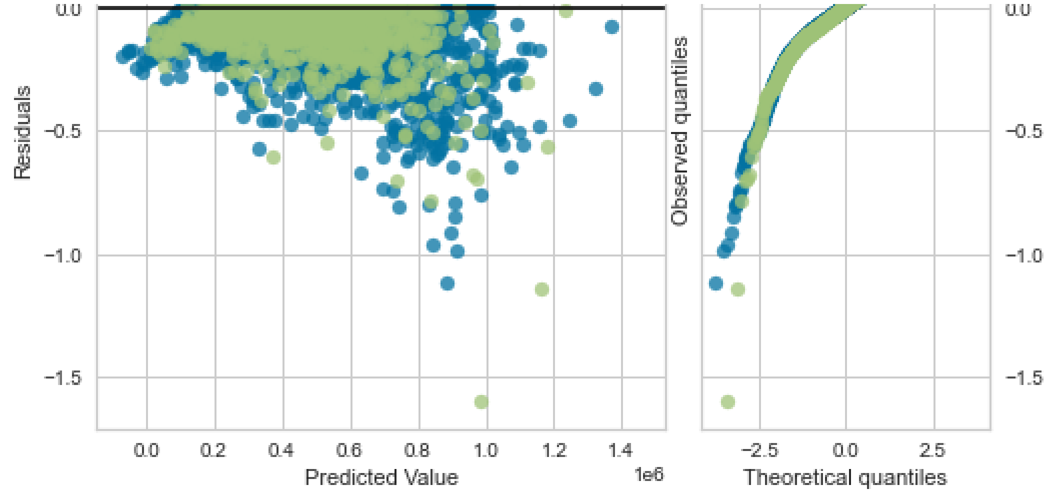
Out[129…

```
<AxesSubplot:title={'center':'Residuals for LinearRegression Model'},
xlabel='Predicted Value', ylabel='Residuals'>
```

In [130…

```
visualizer = ResidualsPlot(lr, hist=False, qqplot=True)
visualizer.fit(X_train_scaled, y_train)
visualizer.score(X_test_scaled, y_test)
visualizer.show()
```





Out[130…

```
<AxesSubplot:title={'center':'Residuals for LinearRegression Model'},
xlabel='Predicted Value', ylabel='Residuals'>
```

In [131…

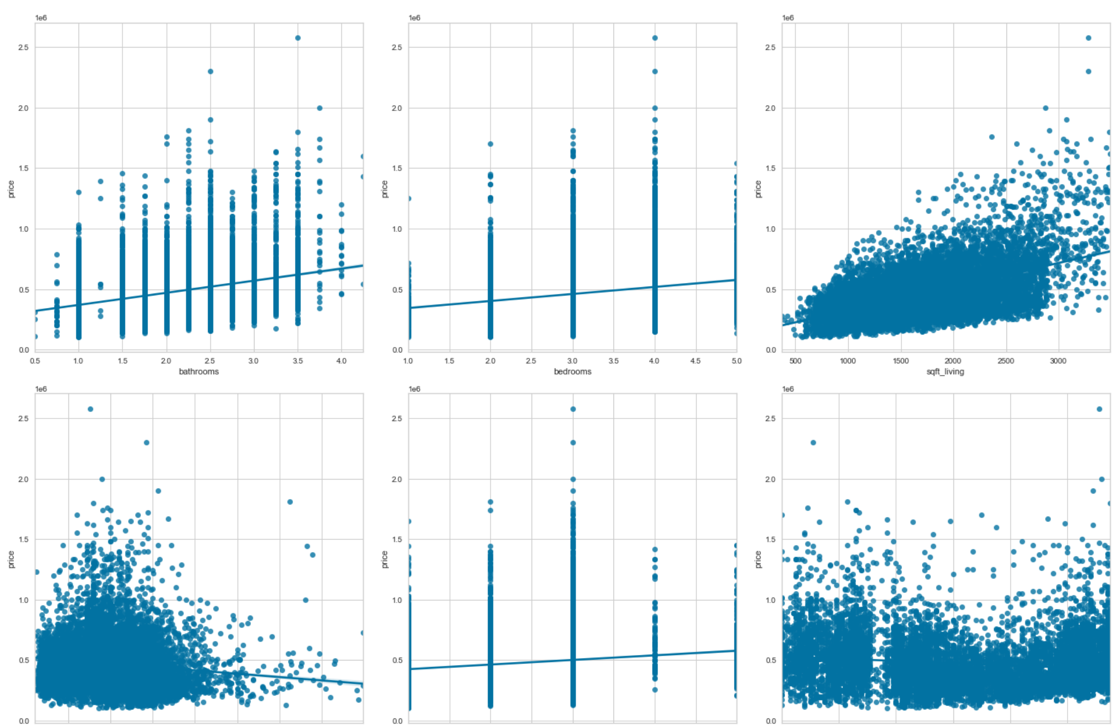
```
# The predicted values of the baseline model are not equally scatterec
```

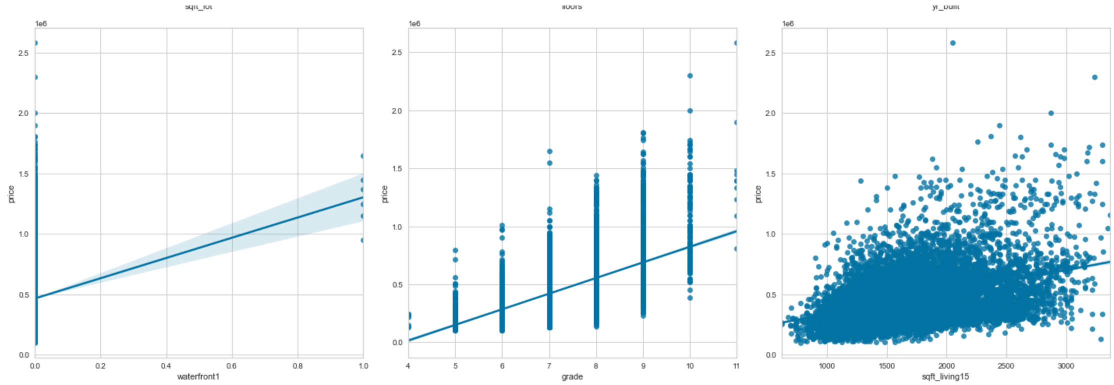
In [132…

```
# examine the relationship of each of the following feature against th
```

In [133…

```
fig, axs = plt.subplots(ncols = 3, nrows = 3, figsize = (20, 20))
sns.regplot(y = df['price'], x = X['bathrooms'], ax = axs[0, 0])
sns.regplot(y = df['price'], x = X['bedrooms'], ax = axs[0, 1])
sns.regplot(y = df['price'], x = X['sqft_living'], ax = axs[0, 2])
sns.regplot(y = df['price'], x = X['sqft_lot'], ax = axs[1, 0])
sns.regplot(y = df['price'], x = X['floors'], ax = axs[1, 1])
sns.regplot(y = df['price'], x = X['yr_built'], ax = axs[1, 2])
sns.regplot(y = df['price'], x = X['waterfront1'], ax = axs[2, 0])
sns.regplot(y = df['price'], x = X['grade'], ax = axs[2, 1])
sns.regplot(y = df['price'], x = X['sqft_living15'], ax = axs[2, 2])
plt.tight_layout()
```





Interpretation of Regression Coefficients

Feature Importances

Based on our latest model, we observed that p values above 0.05 were meaningless, and considering the coefficients, we found the strongest features.

- 1. lot
- 1. waterfront
- 1. floors
- 1. grade
- 1. condition
- 1. bathrooms
- Grade will increase the predicated price %180
- latitude will increase the predicated price %147
- Waterfront will increase the predicated price %77
- floors will increase the predicated price %6
- Bathrooms will increase the predicated price %2
- basement will increase the predicated price %3

the location of the house and the waterfront view seem to greaty affect the price of the house

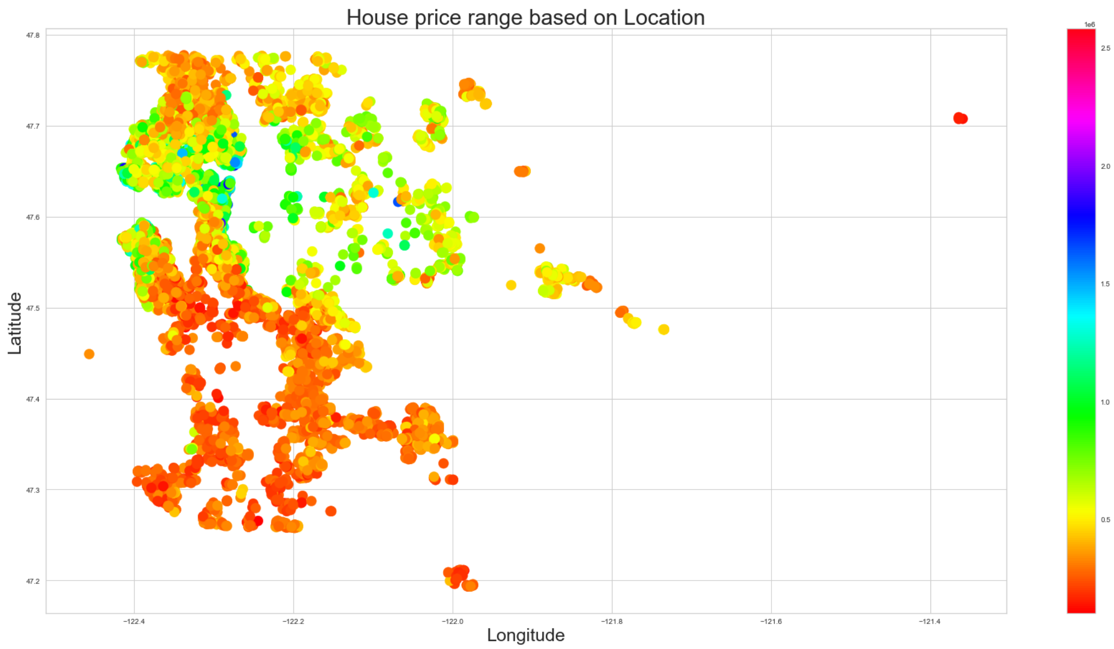
1. Is there any relationship between the house's location and its sale price?

The predicted price will increase with the increase in latitude and decrease in longitude and as the location move to the lower northwest with few scattered houses in the middle to east. These will help the buyer to get an estimate of the housing price range based on the location, and their allocated budget.

In [134…

```
# Visualizing Longitude to Latitude to check how the price vary by loc
plt.figure(figsize= (30, 15))
plt.scatter(x=df['long'], y=df['lat'], c=df['price'], cmap='hsv', mark
plt.title('House price range based on Location',fontsize=30)
plt.xlabel('Longitude',fontsize=25)
plt.ylabel('Latitude',fontsize=25)
```

```
plt.ylabel('Latitude',fontsize=25)
plt.colorbar()
plt.show;
```



2. What are the top ten zip codes that have the highest selling houses in King County?

After looking up the corresponding cities to each zip code, the top ten selling cities in terms of the price mean are Bellevue, Seattle, Mercer Island, Cottage Lake, Maltby, Union Hill-Novelty Hill, Sammamish.

In [135…

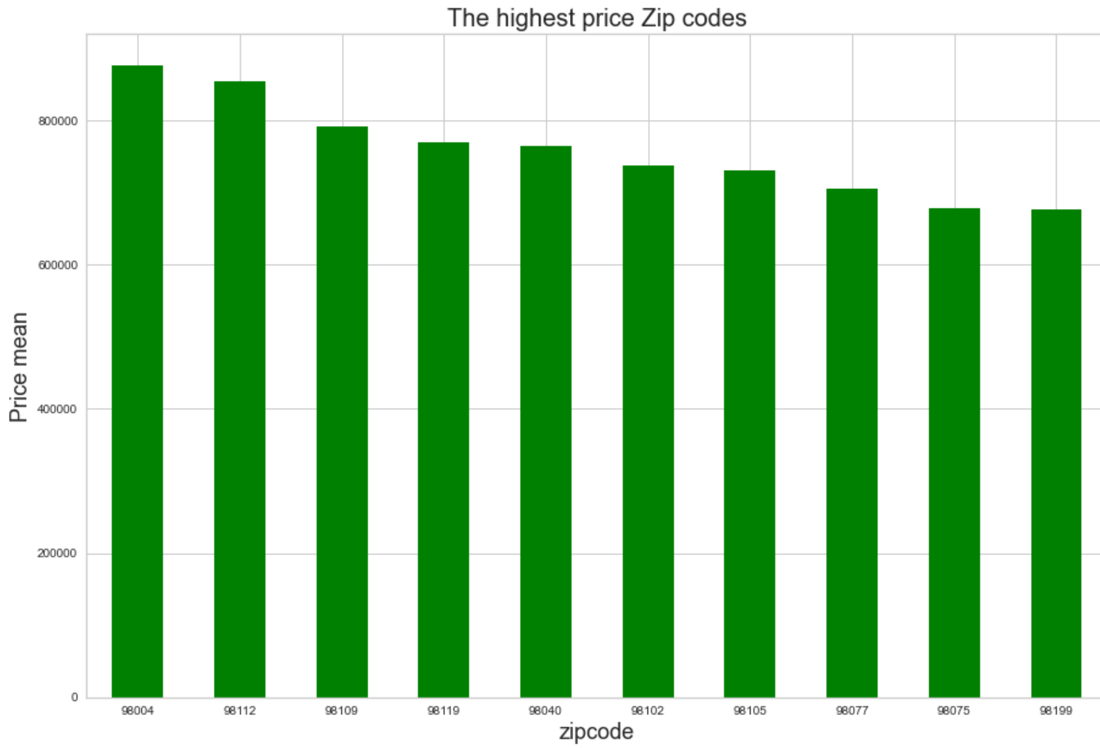
```
# group by zipcode and get the mean of prices in a zipcode
top_ten= df.groupby('zipcode')['price'].mean().sort_values(ascending=F
top_ten.head(20)
```

Out[135…

```
zipcode
98004    876144.950000
98112    853475.984694
98109    790953.826531
98119    770200.748503
98040    765300.000000
98102    738225.533333
98105    730829.549451
98077    705000.000000
98075    678133.288462
98199    676242.224000
Name: price, dtype: float64
```

In [136…

```
# plot top 10 highest house price as reported by zipcode
fig = top_ten.plot(kind = 'bar',color='green', figsize=(15,10))
plt.title('The highest price Zip codes',fontsize=20)
plt.xlabel('zipcode',fontsize=18)
plt.ylabel('Price mean',fontsize=18)
plt.xticks(rotation=0);
plt.show()
```



3. What are the top ten affordable zip codes in King County?

After looking up the corresponding cities to each zip code, the most affordable cities in terms of the price mean are Tukwila, Auburn, Numclaw, Wabash, Birch, Krain, Cumberland, Bayne, Osceola, Maywood, Upper Mill, Bayne Junction, Boise, Veazie, Naco, Stampede,Kent, Lakeland North, Black Diamond, Franklin, and more

In [137…

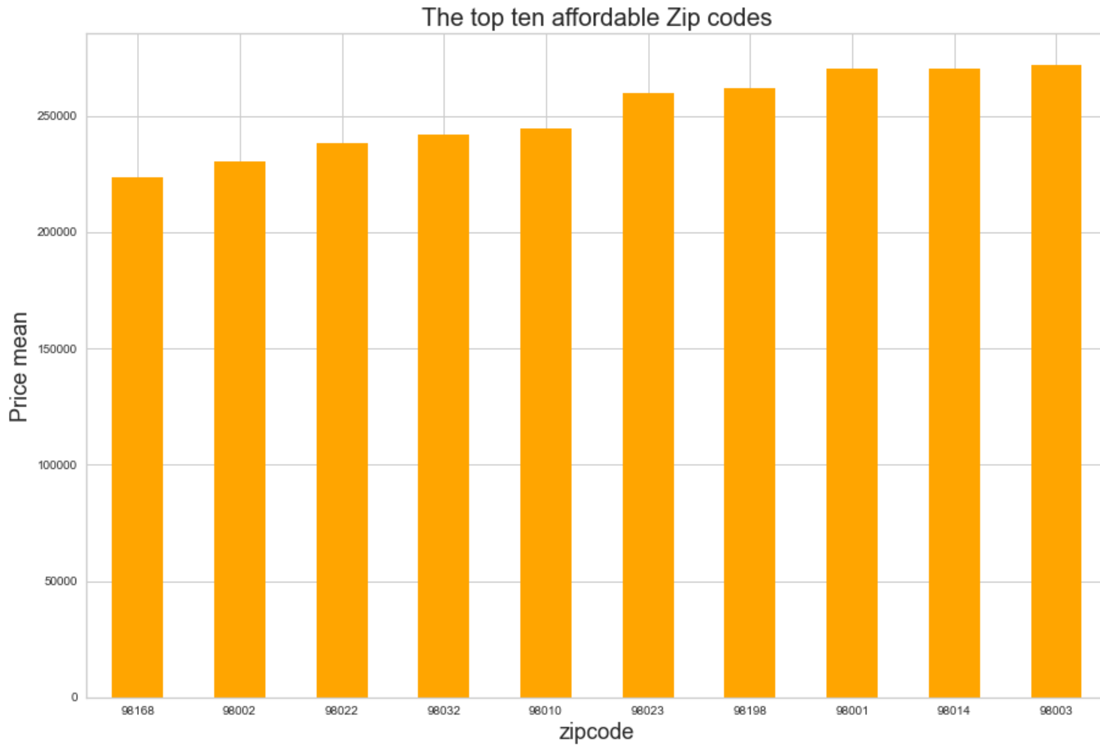
```
# group by zipcode and get the mean of prices in a zipcode
top_ten= df.groupby('zipcode')['price'].mean().sort_values(ascending=T
top_ten.head(20)
```

Out[137…

```
zipcode
98168    223467.465753
98002    230126.106557
98022    238203.108696
98032    241732.434783
98010    244655.000000
98023    259564.941176
98198    261583.513274
98001    270265.829630
98014    270400.000000
98003    271748.489583
Name: price, dtype: float64
```

In [138…

```
# plot top 10 lowest house price as reported by zipcode
fig = top_ten.plot(kind = 'bar',color='orange', figsize=(15,10))
plt.title('The top ten affordable Zip codes',fontsize=20)
plt.xlabel('zipcode',fontsize=18)
plt.ylabel('Price mean',fontsize=18)
plt.xticks(rotation=0);
plt.show()
```

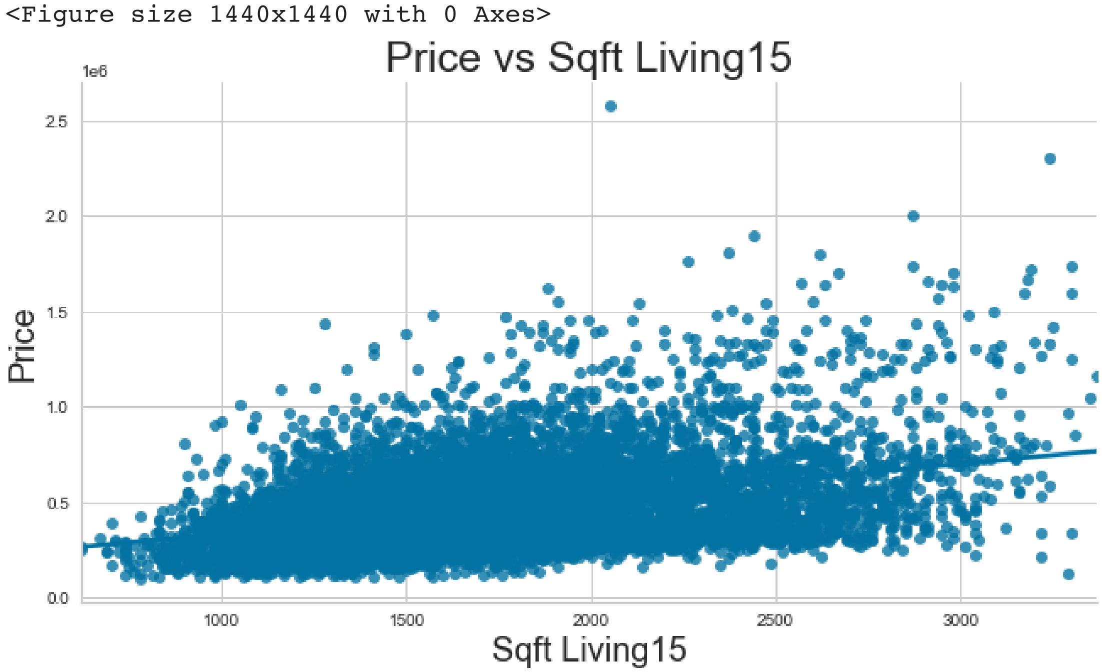


4. Which features are important to predict the price of the house?

In [139…

```
plt.figure(figsize = (20,20));
sqf=sns.lmplot(x="sqft_living15", y="price",aspect=1.8,data=df)
plt.title("Price vs Sqft Living15",fontsize=25)
sqf.set_xlabels("Sqft Living15",fontsize=20)
sqf.set_ylabels("Price",fontsize=20)
plt.show();
```

```
<Figure size 1440x1440 with 0 Axes>
```

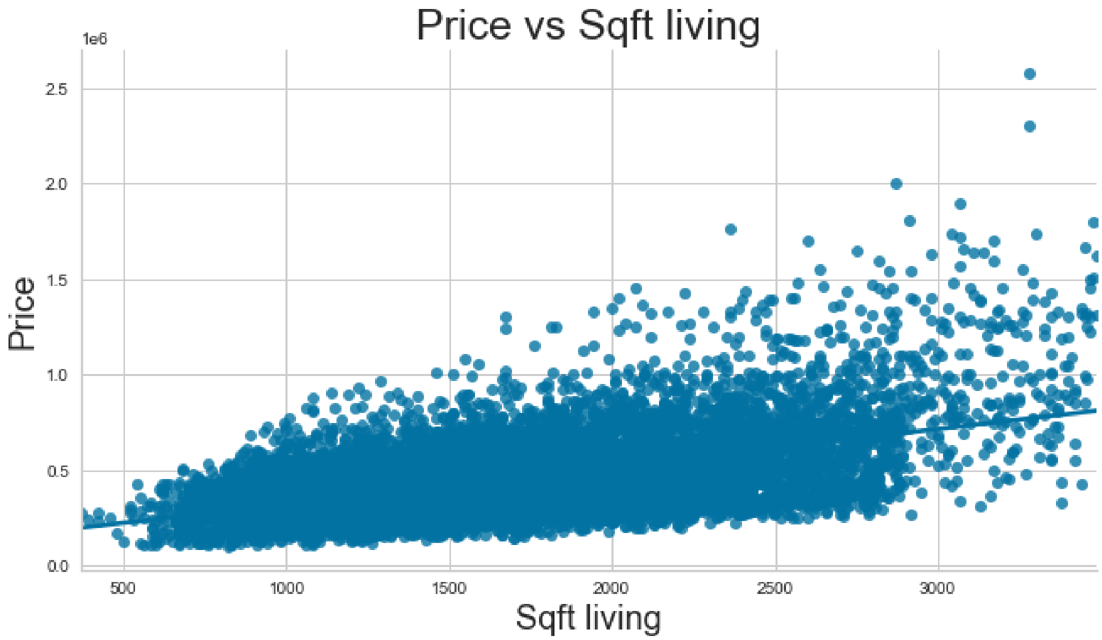


In [140…

```
plt.figure(figsize = (20,20));
```

```
sqf=sns.lmplot(x="sqft_living", y="price",aspect=1.8,data=df)
plt.title("Price vs Sqft living",fontsize=25)
sqf.set_xlabels("Sqft living",fontsize=20)
sqf.set_ylabels("Price",fontsize=20)
plt.show();
```

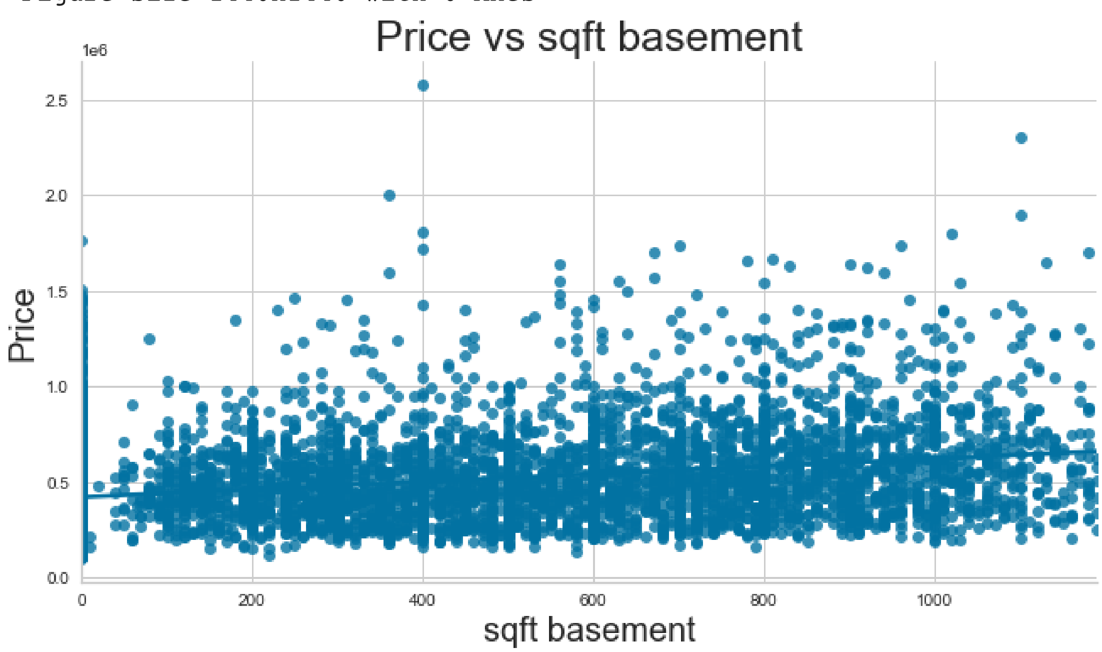
<Figure size 1440x1440 with 0 Axes>



In [141…

```
plt.figure(figsize = (20,20));
sqf=sns.lmplot(x="sqft_basement", y="price",aspect=1.8,data=df)
plt.title("Price vs sqft basement",fontsize=25)
sqf.set_xlabels("sqft basement",fontsize=20)
sqf.set_ylabels("Price",fontsize=20)
plt.show();
```

<Figure size 1440x1440 with 0 Axes>



In [142…

```
plt.figure(figsize = (20,20));
sqf=sns.lmplot(x="bathrooms", y="price",aspect=1.8,data=df)
```

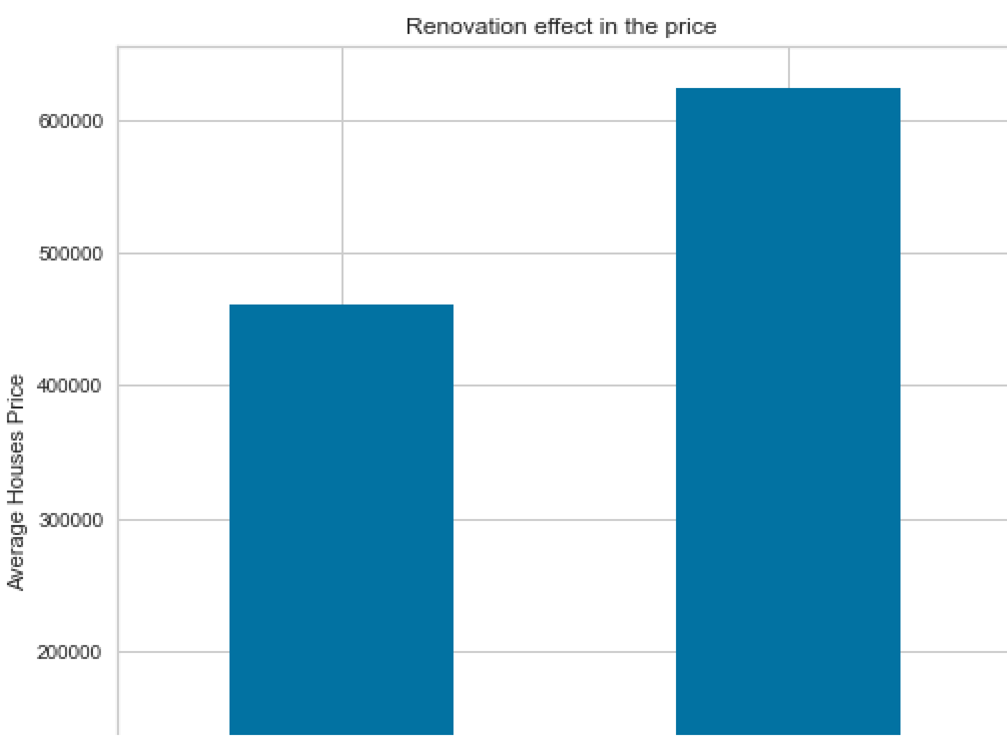
```
plt.title("Price vs Bathrooms",fontsize=25)
sqf.set_xlabels("Bathrooms",fontsize=20)
sqf.set_ylabels("Price",fontsize=20)
plt.show();
```

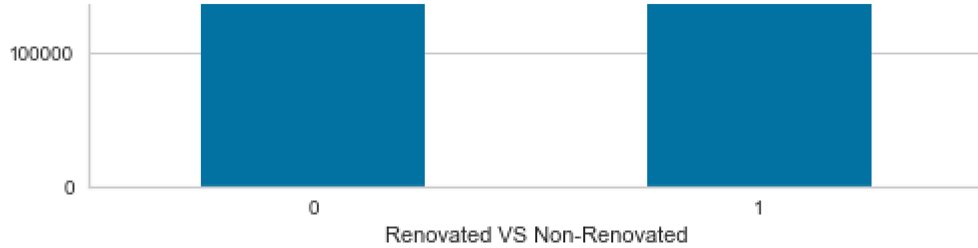
<Figure size 1440x1440 with 0 Axes>

In [143…

```
# plotting houses to the mean of price
df.groupby("renovated")["price"].mean().plot(kind="bar",figsize=(8,8))
plt.title("Renovation effect in the price ")
plt.ylabel("Average Houses Price")
plt.xlabel("Renovated VS Non-Renovated")
plt.xticks(rotation=0)
#the renovated houses selling price is higher than non-renovated one
```

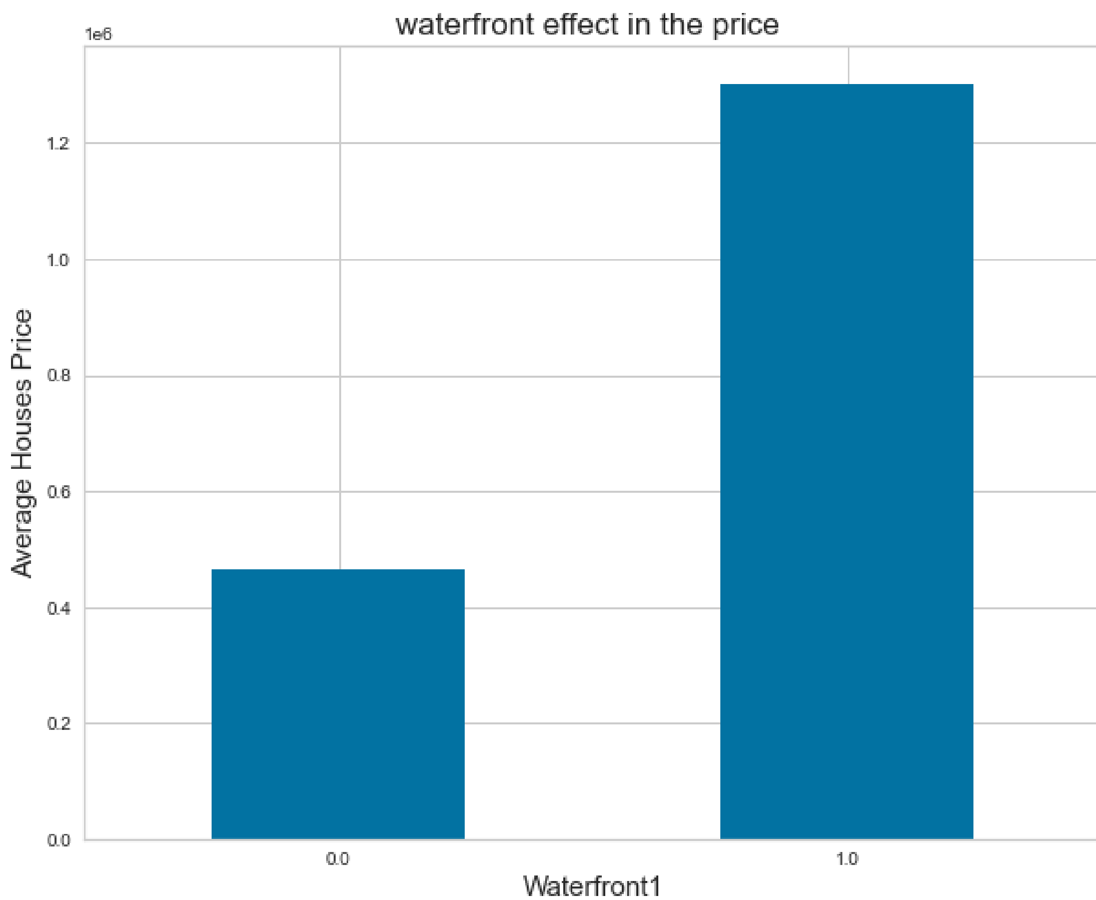
Out[143… (array([0, 1]), [Text(0, 0, '0'), Text(1, 0, '1')])





In [144…

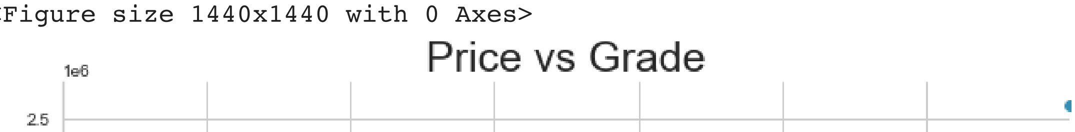
```
# plotting houses to the mean of price
df.groupby("waterfront1")["price"].mean().plot(kind="bar",figsize=(10,
plt.title("waterfront effect in the price ", fontsize=17)
plt.ylabel("Average Houses Price",fontsize=15)
plt.xlabel("Waterfront1",fontsize=15)
plt.xticks(rotation=0)
plt.show()
#the houses with waterfront selling price are higher than one without
```

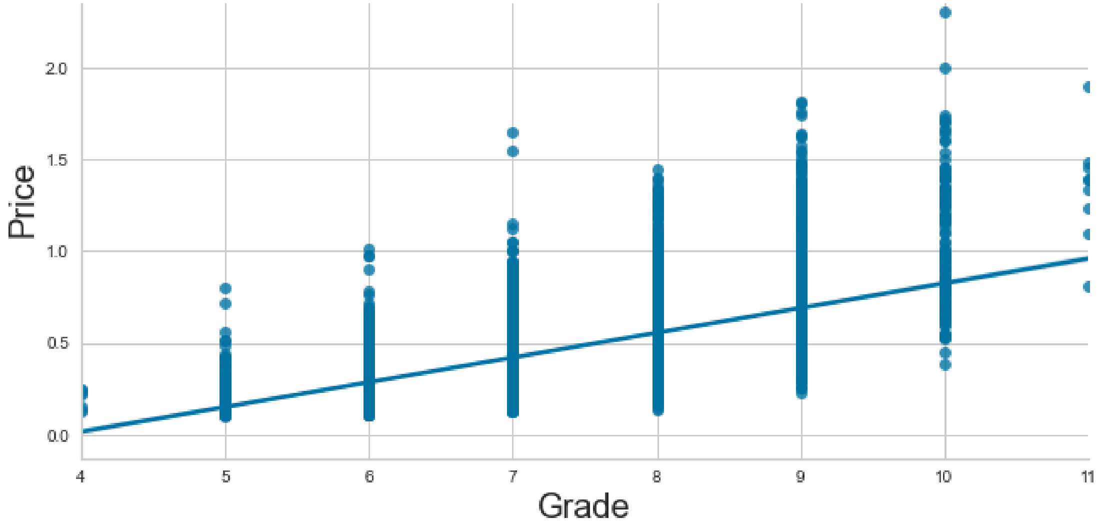


In [145…

```
plt.figure(figsize = (20,20));
sqf=sns.lmplot(x="grade", y="price",aspect=1.8,data=df)
plt.title("Price vs Grade",fontsize=25)
sqf.set_xlabels("Grade",fontsize=20)
sqf.set_ylabels("Price",fontsize=20)
plt.show();
```

```
<Figure size 1440x1440 with 0 Axes>
```





In [146…

```
# plotting houses to the mean of price
df.groupby("grade")["price"].mean().plot(kind="bar",figsize=(10,8));
plt.title("Grade effect in the price ", fontsize=17)
plt.ylabel("Average Houses Price",fontsize=15)
plt.xlabel("Grade",fontsize=15)
plt.xticks(rotation=0)
```