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

3.77 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_squared_er

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

pd.options.display.max_rows=300

import utils as ut

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

```
/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/statsmodels/ts
a/base/tsa_model.py:7: FutureWarning: pandas.Int64Index is deprecated 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,
/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/statsmodels/ts
a/base/tsa_model.py:7: FutureWarning: pandas.Float64Index is deprecated an
d will be removed from pandas in a future version. Use pandas.Index with t
he appropriate dtype instead.
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
```

Obtain the data

In [2]:

```
# read in the data
df = pd.read_csv("data/kc_house_data.csv")
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	floors
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0

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 numerical
```

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	3.904
price	21597.0	5.402966e+05	3.673681e+05	7.800000e+04	3.220000e+05	4.500

price	21597.0	5.402966e+05	3.673681e+05	7.800000e+04	3.220000e+05	4.500
bedrooms	21597.0	3.373200e+00	9.262989e-01	1.000000e+00	3.000000e+00	3.000
bathrooms	21597.0	2.115826e+00	7.689843e-01	5.000000e-01	1.750000e+00	2.250
sqft_living	21597.0	2.080322e+03	9.181061e+02	3.700000e+02	1.430000e+03	1.910
sqft_lot	21597.0	1.509941e+04	4.141264e+04	5.200000e+02	5.040000e+03	7.618
floors	21597.0	1.494096e+00	5.396828e-01	1.000000e+00	1.000000e+00	1.500
sqft_above	21597.0	1.788597e+03	8.277598e+02	3.700000e+02	1.190000e+03	1.560
yr_built	21597.0	1.971000e+03	2.937523e+01	1.900000e+03	1.951000e+03	1.975
yr_renovated	17755.0	8.363678e+01	3.999464e+02	0.000000e+00	0.000000e+00	0.000
zipcode	21597.0	9.807795e+04	5.351307e+01	9.800100e+04	9.803300e+04	9.806
lat	21597.0	4.756009e+01	1.385518e-01	4.715590e+01	4.747110e+01	4.757
long	21597.0	-1.222140e+02	1.407235e-01	-1.225190e+02	-1.223280e+02	-1.222
sqft_living15	21597.0	1.986620e+03	6.852305e+02	3.990000e+02	1.490000e+03	1.840
sqft_lot15	21597.0	1.275828e+04	2.727444e+04	6.510000e+02	5.100000e+03	7.620

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.drop_duplicates(subset=['id'], inplace=True)
```

In [8]:

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

Out[8]:

```
False
```

In [9]:

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

Out[9]:

```
id                 0
date               0
price              0
bedrooms           0
bathrooms          0
sqft_living        0
sqft_lot           0
floors             0
waterfront      2353
```

```
view               63
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_built            0
yr_renovated     3804
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
dtype: int64
```

In [10]:

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

In [11]:

```
df.head()
```

Out[11]:

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

5 rows × 21 columns

In [12]:

```
df
```

Out[12]:

	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	
...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	

21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076

21420 rows × 21 columns

In [13]:

```
def waterfront11(x):
    if x == "NO":
        return 0
    if x == "YES":
        return 1
```

In [14]:

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

Out[14]:

	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	
...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	

21420 rows × 22 columns

In [15]:

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

In [16]:

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

Out[16]:

```
0.0    21274
1.0      146
Name: waterfront1, dtype: int64
```

In [17]:

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

Out[17]:

	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	
...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	

21420 rows × 21 columns

In [18]:

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

Out[18]:

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

13	6054650070	10/7/2014	400000.0	3	1.75	1370	9680	1
14	1175000570	3/12/2015	530000.0	5	2.00	1810	4850	1
15	9297300055	1/24/2015	650000.0	4	3.00	2950	5000	2
16	1875500060	7/31/2014	395000.0	3	2.00	1890	14040	2
17	6865200140	5/29/2014	485000.0	4	1.00	1600	4300	1
18	16000397	12/5/2014	189000.0	2	1.00	1200	9850	1
19	7983200060	4/24/2015	230000.0	3	1.00	1250	9774	1
20	6300500875	5/14/2014	385000.0	4	1.75	1620	4980	1
21	2524049179	8/26/2014	2000000.0	3	2.75	3050	44867	1
22	7137970340	7/3/2014	285000.0	5	2.50	2270	6300	2
23	8091400200	5/16/2014	252700.0	2	1.50	1070	9643	1
24	3814700200	11/20/2014	329000.0	3	2.25	2450	6500	2
25	1202000200	11/3/2014	233000.0	3	2.00	1710	4697	1
26	1794500383	6/26/2014	937000.0	3	1.75	2450	2691	2
27	3303700376	12/1/2014	667000.0	3	1.00	1400	1581	1
28	5101402488	6/24/2014	438000.0	3	1.75	1520	6380	1
29	1873100390	3/2/2015	719000.0	4	2.50	2570	7173	2
30	8562750320	11/10/2014	580500.0	3	2.50	2320	3980	2
31	2426039314	12/1/2014	280000.0	2	1.50	1190	1265	3
32	461000390	6/24/2014	687500.0	4	1.75	2330	5000	1
33	7589200193	11/10/2014	535000.0	3	1.00	1090	3000	1
34	7955080270	12/3/2014	322500.0	4	2.75	2060	6659	1
35	9547205180	6/13/2014	696000.0	3	2.50	2300	3060	1
36	9435300030	5/28/2014	550000.0	4	1.00	1660	34848	1
37	2768000400	12/30/2014	640000.0	4	2.00	2360	6000	2
38	7895500070	2/13/2015	240000.0	4	1.00	1220	8075	1
39	2078500320	6/20/2014	605000.0	4	2.50	2620	7553	2
40	5547700270	7/15/2014	625000.0	4	2.50	2570	5520	2
41	7766200013	8/11/2014	775000.0	4	2.25	4220	24186	1
42	7203220400	7/7/2014	861990.0	5	2.75	3595	5639	2
43	9270200160	10/28/2014	685000.0	3	1.00	1570	2280	2
44	1432701230	7/29/2014	309000.0	3	1.00	1280	9656	1
45	8035350320	7/18/2014	488000.0	3	2.50	3160	13603	2
46	8945200830	3/25/2015	210490.0	3	1.00	990	8528	1
47	4178300310	7/16/2014	785000.0	4	2.50	2290	13416	2
48	9215400105	4/28/2015	450000.0	3	1.75	1250	5963	1

49	822039084	3/11/2015	1350000.0	3	2.50	2753	65005	1

50 rows × 22 columns

In [19]:

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

Out[19]:

	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	
...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	

21420 rows × 21 columns

In [20]:

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

Out[20]:

	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	

...	...	...	...	...	...	...	...
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076

21420 rows × 21 columns

In [21]:

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

Out[21]:

```
NONE         19253
AVERAGE        956
GOOD           505
FAIR           329
EXCELLENT      314
NaN             63
Name: view, dtype: int64
```

In [22]:

```
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 [23]:

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

Out[23]:

	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	
...	...	...	...	...	...	...	...	

21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076

21420 rows × 21 columns

In [24]:

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

Out[24]:

	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	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	

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	5245600105	9/16/2014	228000.0	3	1.00	1190	9199
51	7231300125	2/17/2015	345000.0	5	2.50	3150	9134
52	7518505990	12/31/2014	600000.0	3	1.75	1410	4080
53	3626039271	2/5/2015	585000.0	2	1.75	1980	8550
54	4217401195	3/3/2015	920000.0	5	2.25	2730	6000
55	9822700295	5/12/2014	885000.0	4	2.50	2830	5000
56	9478500640	8/19/2014	292500.0	4	2.50	2250	4495
57	2799800710	4/7/2015	301000.0	3	2.50	2420	4750
58	7922800400	8/27/2014	951000.0	5	3.25	3250	14342
59	8079040320	2/23/2015	430000.0	4	3.00	1850	9976
60	1516000055	12/10/2014	650000.0	3	2.25	2150	21235
61	9558200045	8/28/2014	289000.0	3	1.75	1260	8400
62	5072410070	10/21/2014	505000.0	3	1.75	2519	8690
63	9528102996	12/7/2014	549000.0	3	1.75	1540	1044
64	1189001180	6/3/2014	425000.0	3	2.25	1660	6000
65	3253500160	11/20/2014	317625.0	3	2.75	2770	3809
66	3394100030	9/9/2014	975000.0	4	2.50	2720	11049
67	3717000160	10/9/2014	287000.0	4	2.50	2240	4648
68	1274500060	8/25/2014	204000.0	3	1.00	1000	12070
69	1802000060	6/12/2014	1330000.0	5	2.25	3200	20158
70	1525059190	9/12/2014	1040000.0	5	3.25	4770	50094
71	1049000060	1/5/2015	325000.0	3	2.00	1260	5612
72	8820901275	6/10/2014	571000.0	4	2.00	2750	7807

73	5416510140	7/10/2014	360000.0	4	2.50	2380	5000
74	3444100400	3/16/2015	349000.0	3	1.75	1790	50529
75	3276920270	11/5/2014	832500.0	4	4.00	3430	35102
76	4036801170	10/13/2014	380000.0	4	1.75	1760	7300
77	2391600320	4/20/2015	480000.0	3	1.00	1040	5060
78	6300000287	6/9/2014	410000.0	3	1.00	1410	5060
79	1531000030	3/23/2015	720000.0	4	2.50	3450	39683
80	5104520400	12/2/2014	390000.0	3	2.50	2350	5100
81	7437100340	12/22/2014	360000.0	4	2.50	1900	5889
82	9418400240	10/28/2014	355000.0	2	1.00	2020	6720
83	1523059105	1/28/2015	356000.0	3	1.50	1680	8712
84	1133000671	6/2/2014	315000.0	3	1.00	960	6634
85	4232902595	11/14/2014	940000.0	3	1.50	2140	3600
86	2599001200	11/3/2014	305000.0	5	2.25	2660	8400
87	3342103156	6/18/2014	461000.0	3	3.25	2770	6278
88	1332700270	5/19/2014	215000.0	2	2.25	1610	2040
89	3869900162	9/4/2014	335000.0	2	1.75	1030	1066
90	2791500270	5/22/2014	243500.0	4	2.50	1980	7403
91	5036300431	3/11/2015	1100000.0	5	2.75	3520	6353
92	4168000060	2/26/2015	153000.0	3	1.00	1200	10500
93	6021501535	7/25/2014	430000.0	3	1.50	1580	5000
95	1483300570	9/8/2014	905000.0	4	2.50	3300	10250
96	3422049190	3/30/2015	247500.0	3	1.75	1960	15681
97	1099611230	9/12/2014	199000.0	4	1.50	1160	6400
98	722079104	7/11/2014	314000.0	3	1.75	1810	41800
99	7338200240	5/16/2014	437500.0	3	2.50	2320	36847

100	1952200240	6/11/2014	850830.0	3	2.50	2070	13241
101	5200100125	10/27/2014	555000.0	3	2.00	1980	3478
102	7214720075	12/12/2014	699950.0	3	2.25	2190	107593
103	2450000295	10/7/2014	1090000.0	3	2.50	2920	8113
104	6197800045	9/24/2014	290000.0	3	1.00	1210	33919
105	1328310370	4/2/2015	375000.0	3	2.50	2340	10005
106	546000875	5/23/2014	460000.0	3	1.00	1670	4005
107	3530510041	7/23/2014	188500.0	2	1.75	1240	2493
108	1853000400	3/5/2015	680000.0	4	2.50	3140	28037
109	3134100116	8/27/2014	470000.0	5	1.75	2030	12342
110	9545230140	7/25/2014	597750.0	4	2.50	2310	9624
111	3362400511	3/4/2015	570000.0	3	1.75	1260	3328
112	2525310310	9/16/2014	272500.0	3	1.75	1540	12600
113	6126500060	11/24/2014	329950.0	3	1.75	2080	5969
114	8961960160	10/28/2014	480000.0	4	2.50	3230	16171
115	3626039325	11/21/2014	740500.0	3	3.50	4380	6350
116	3362400431	6/26/2014	518500.0	3	3.50	1590	1102
117	4060000240	6/23/2014	205425.0	2	1.00	880	6780
118	3454800060	1/8/2015	171800.0	4	2.00	1570	9600
119	1695900060	5/11/2015	535000.0	4	1.00	1610	2982
120	7278700070	1/2/2015	660000.0	3	2.50	2400	6474
121	6675500070	11/19/2014	391500.0	3	2.00	1450	9132
122	3626039187	4/6/2015	395000.0	2	1.00	770	6000
123	3524049083	11/4/2014	445000.0	4	1.75	2100	4400
124	3275860240	6/18/2014	770000.0	3	2.25	2910	10204
125	4389200955	3/2/2015	1450000.0	4	2.75	2750	17789
126	4058801670	7/17/2014	445000.0	3	2.25	2100	8201
127	8732020310	7/17/2014	260000.0	4	2.25	2160	8811

128	2331300505	6/13/2014	822500.0	5	3.50	2320	4960
129	7853210060	4/6/2015	430000.0	4	2.50	2070	4310
130	3668000070	1/5/2015	212000.0	3	1.75	1060	7875
131	9545240070	4/28/2015	660500.0	4	2.25	2010	9603
132	1243100136	6/12/2014	784000.0	3	3.50	3950	111078
133	8929000270	5/12/2014	453246.0	3	2.50	2010	2287
134	2767602356	1/26/2015	675000.0	4	3.50	2140	2278
135	921049315	8/13/2014	199000.0	3	1.75	1320	17390
136	3655000070	8/5/2014	220000.0	4	1.75	2020	7840
137	4027700812	5/29/2014	452000.0	4	2.25	2590	10002
138	3992700335	7/7/2014	382500.0	2	1.00	1190	4440
139	2767603505	5/7/2014	519950.0	3	2.25	1170	1249
140	4232901525	6/27/2014	665000.0	2	1.00	1110	3200
141	1777500060	7/8/2014	527700.0	5	2.50	2820	9375
142	1432900240	5/8/2015	205000.0	3	1.00	1610	8579
143	6140100875	4/15/2015	420000.0	3	1.00	1060	8097
144	6071600370	2/27/2015	500000.0	4	2.25	2030	8517
145	1526069017	12/3/2014	921500.0	4	2.50	3670	315374
146	809001525	6/25/2014	890000.0	4	1.00	2550	4000
147	3224079105	8/6/2014	430000.0	2	2.50	2420	60984
148	8075400570	10/30/2014	258000.0	5	2.00	2260	12500
149	1994200024	11/4/2014	511000.0	3	1.00	1430	3455
150	3362900810	8/20/2014	532170.0	3	2.00	1360	3090
151	1324300398	4/9/2015	560000.0	3	1.00	1110	5000
152	537000445	3/31/2015	282950.0	3	1.00	1250	8200
153	7855801670	4/1/2015	2250000.0	4	3.25	5180	19850
154	7920100045	5/16/2014	350000.0	1	1.00	700	5100
155	8960000030	7/28/2014	215000.0	3	1.00	1180	7669

156	6388930390	11/20/2014	650000.0	5	3.50	3960	25245
157	8731900200	8/7/2014	320000.0	4	2.75	2640	7500
158	8029200135	11/13/2014	247000.0	3	2.00	1270	7198
159	1081200350	10/3/2014	320000.0	4	1.75	1760	11180
160	84000105	5/7/2014	255000.0	5	2.25	2060	8632
161	3756500060	3/9/2015	438000.0	3	1.75	1780	9660
162	7215720160	3/4/2015	900000.0	3	2.50	3400	16603
163	3574800520	6/20/2014	441000.0	3	2.75	1910	7280
164	2617300160	8/12/2014	420000.0	3	2.00	2020	38332
165	2558660270	12/8/2014	370000.0	3	1.75	1580	7000
166	2009000370	2/19/2015	269950.0	2	1.75	1340	7250
167	1836980160	3/24/2015	807100.0	4	2.50	2680	4499
168	3261020370	6/5/2014	653000.0	3	2.50	2680	9750
169	1755700060	6/11/2014	371500.0	3	2.00	1370	8336
170	4330600435	3/16/2015	284000.0	3	1.75	1560	21000
171	9542800700	1/2/2015	272000.0	3	1.75	2160	7140
172	1999700045	5/2/2014	313000.0	3	1.50	1340	7912
173	1762600070	1/16/2015	917500.0	4	2.50	3880	35003
174	1687900520	9/29/2014	673000.0	4	2.25	2590	8190
175	7234600798	2/10/2015	425000.0	3	2.50	1120	1100
176	3881900445	7/9/2014	399950.0	5	2.75	1970	5400
177	2254502445	5/30/2014	385000.0	3	1.00	1220	4800
178	5437810320	11/17/2014	269950.0	3	1.50	1950	7560
179	9158100075	1/7/2015	330000.0	2	1.00	1350	8220
180	3830630310	7/25/2014	260000.0	3	2.50	1670	5797
181	8123100045	4/14/2015	470000.0	4	3.00	2380	5125

182	3127200041	6/13/2014	589000.0	4	3.00	2440	9600
183	6661200320	7/23/2014	163500.0	2	1.50	1050	3419
184	11510310	9/5/2014	835000.0	4	2.75	3130	13412
185	825059270	11/21/2014	1100000.0	5	3.00	4090	12850
186	8731951370	4/15/2015	269000.0	4	1.75	1490	10000
187	1954440060	5/5/2014	560000.0	3	2.50	1900	8744
188	2264500350	4/18/2015	615000.0	4	1.00	1330	2400
189	1115810060	12/5/2014	585188.0	3	2.25	2230	10026
190	9477200200	8/18/2014	305000.0	3	1.75	1650	9480
191	1432600560	11/5/2014	166950.0	3	1.00	1190	8820
192	2287000060	9/12/2014	799000.0	3	2.50	2140	9897
193	3663500060	6/25/2014	400000.0	3	2.50	2180	7508
194	3996900125	12/1/2014	230000.0	3	1.00	1060	10228
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	3542300060	3/11/2015	210000.0	3	1.00	860	11725

200 rows × 21 columns

In [25]:

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

In [26]:

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

Out[26]:

```
7      8889
8      6041
9      2606
6      1995
10     1130
11      396
```

```
5       234
12       88
4        27
13       13
3         1
Name: grade, dtype: int64
```

In [27]:

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

In [28]:

```
df
```

Out[28]:

	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	
...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	

21420 rows × 21 columns

In [29]:

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

In [30]:

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

Out[30]:

```
7.0     8889
8.0     6041
9.0     2606
6.0     1995
10.0    1130
11.0     396
5.0      234
12.0      88
4.0       27
13.0      13
3.0        1
Name: grade, dtype: int64
```

In [31]:

```
df
```

Out[31]:

	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	
...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	

21420 rows × 21 columns

In [32]:

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

Out[32]:

```
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 [33]:

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

Out[33]:

```
0.0       12717
?           452
600.0       216
500.0       206
700.0       205
          ...  
1920.0        1
3480.0        1
2730.0        1
2720.0        1
248.0         1
Name: sqft_basement, Length: 304, dtype: int64
```

In [34]:

```
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[34]:

```
0.0       13169
600.0       216
500.0       206
700.0       205
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 [35]:

```
df.isnull().sum()
#sanity check
```

Out[35]:

```
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 [36]:

```
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 been vi
```

In [37]:

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

Out[37]:

	count	mean	std	min	25%	
price	21420.0	540739.303922	367931.109953	78000.0000	322500.0000	45000(
bedrooms	21420.0	3.373950	0.925405	1.0000	3.0000	;
bathrooms	21420.0	2.118429	0.768720	0.5000	1.7500	:
sqft_living	21420.0	2083.132633	918.808412	370.0000	1430.0000	192(
sqft_lot	21420.0	15128.038002	41530.796838	520.0000	5040.0000	761‹

floors	21420.0	1.495985	0.540081	1.0000	1.0000	
grade	21420.0	7.662792	1.171971	3.0000	7.0000	
sqft_above	21420.0	1791.170215	828.692965	370.0000	1200.0000	156
sqft_basement	21420.0	285.904342	440.008202	0.0000	0.0000	
yr_built	21420.0	1971.092997	29.387141	1900.0000	1952.0000	197
yr_renovated	21420.0	68.956723	364.552298	0.0000	0.0000	
zipcode	21420.0	98077.874370	53.477480	98001.0000	98033.0000	9806
lat	21420.0	47.560197	0.138589	47.1559	47.4712	4
long	21420.0	-122.213784	0.140791	-122.5190	-122.3280	-12
sqft_living15	21420.0	1988.384080	685.537057	399.0000	1490.0000	184
sqft_lot15	21420.0	12775.718161	27345.621867	651.0000	5100.0000	762
waterfront1	21420.0	0.006816	0.082280	0.0000	0.0000	
condition1	21420.0	1.078758	0.269367	1.0000	1.0000	

In [38]:
```
df
```

Out[38]:

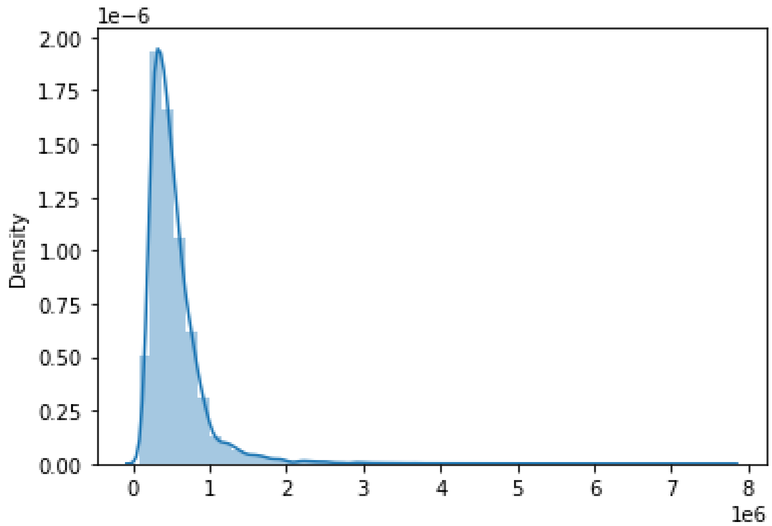
	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	grade	sqft_above
0	221900.0	3	1.00	1180	5650	1.0	7.0	1180
1	538000.0	3	2.25	2570	7242	2.0	7.0	2170
2	180000.0	2	1.00	770	10000	1.0	6.0	770
3	604000.0	4	3.00	1960	5000	1.0	7.0	1050
4	510000.0	3	2.00	1680	8080	1.0	8.0	1680
...	...	...	...	...	...	...	...	...
21592	360000.0	3	2.50	1530	1131	3.0	8.0	1530
21593	400000.0	4	2.50	2310	5813	2.0	8.0	2310
21594	402101.0	2	0.75	1020	1350	2.0	7.0	1020
21595	400000.0	3	2.50	1600	2388	2.0	8.0	1600
21596	325000.0	2	0.75	1020	1076	2.0	7.0	1020

21420 rows × 18 columns

In [39]:
```
#I will set a range for each feature and get rid of outliers, I will look
#from .describe
```

In [40]:
```
ut.plot(df,["price"])
```

Out[40]: <AxesSubplot:ylabel='Density'>



In [41]:
```
df=df[(df['price'] < 12000000) & (df['price'] >100000)] # limiting my pric
df
```

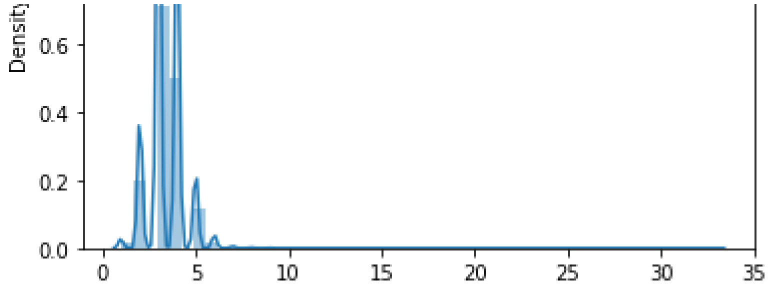
Out[41]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	grade	sqft_above
0	221900.0	3	1.00	1180	5650	1.0	7.0	1180
1	538000.0	3	2.25	2570	7242	2.0	7.0	2170
2	180000.0	2	1.00	770	10000	1.0	6.0	770
3	604000.0	4	3.00	1960	5000	1.0	7.0	1050
4	510000.0	3	2.00	1680	8080	1.0	8.0	1680
...	...	...	...	...	...	...	...	...
21592	360000.0	3	2.50	1530	1131	3.0	8.0	1530
21593	400000.0	4	2.50	2310	5813	2.0	8.0	2310
21594	402101.0	2	0.75	1020	1350	2.0	7.0	1020
21595	400000.0	3	2.50	1600	2388	2.0	8.0	1600
21596	325000.0	2	0.75	1020	1076	2.0	7.0	1020

21390 rows × 18 columns

In [42]:
```
ut.plot(df,["bedrooms"])
```

Out[42]: <AxesSubplot:ylabel='Density'>



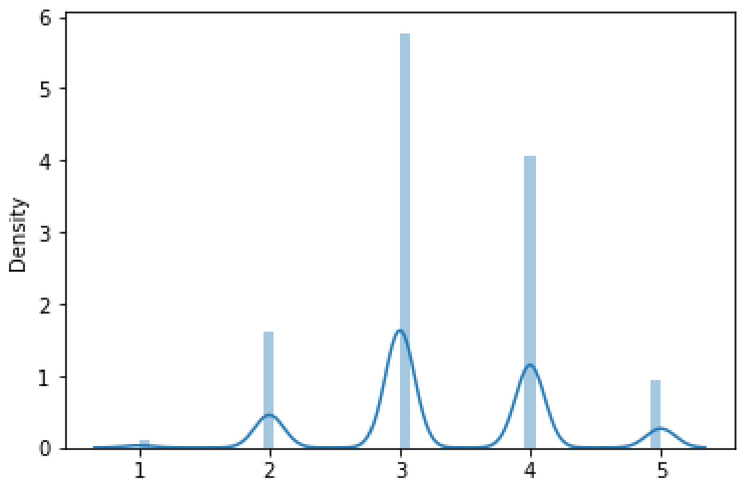
In [43]:

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

In [44]:

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

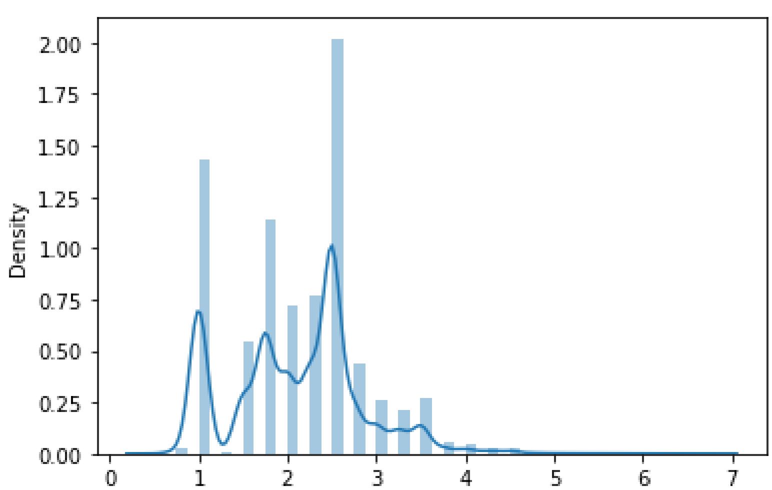
Out[44]: <AxesSubplot:ylabel='Density'>



In [45]:

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

Out[45]: <AxesSubplot:ylabel='Density'>

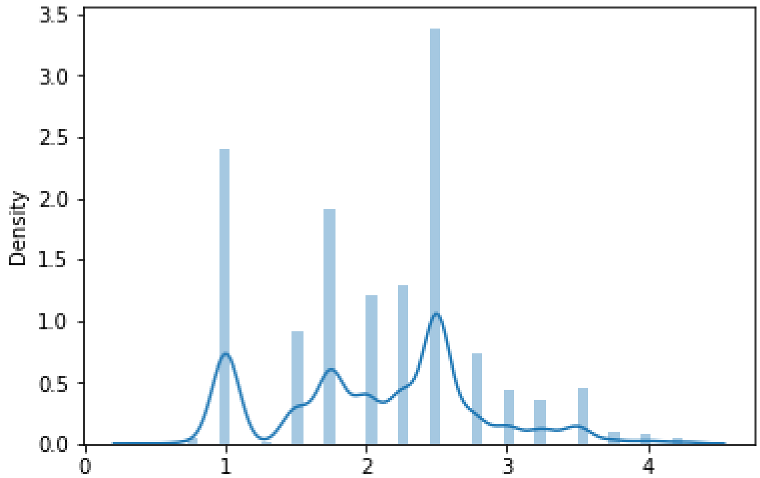


In [46]:

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

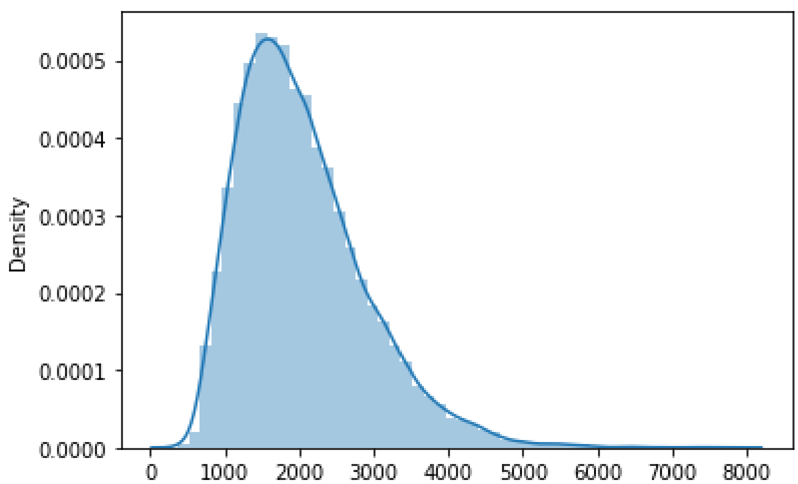
In [47]:
```
ut.plot(df,['bathrooms'])
```

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



In [48]:
```
ut.plot(df,['sqft_living'])
```

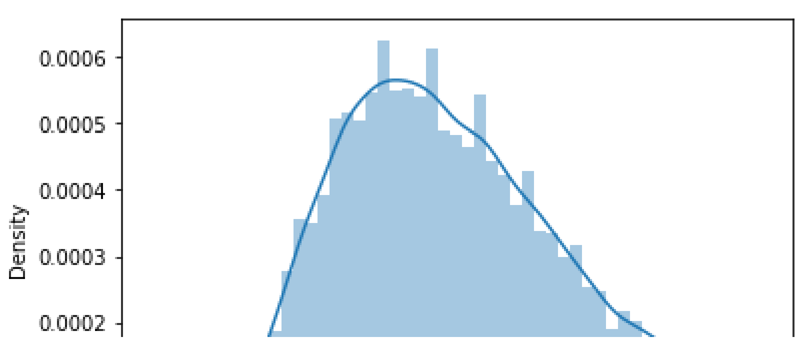
Out[48]: <AxesSubplot:ylabel='Density'>

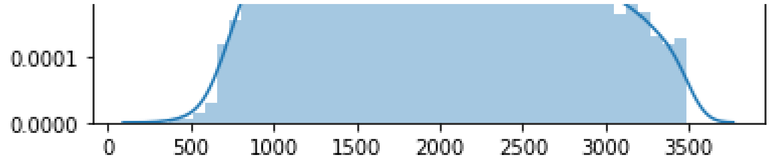


In [49]:
```
df=df[(df["sqft_living"]<3500)]
```

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

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

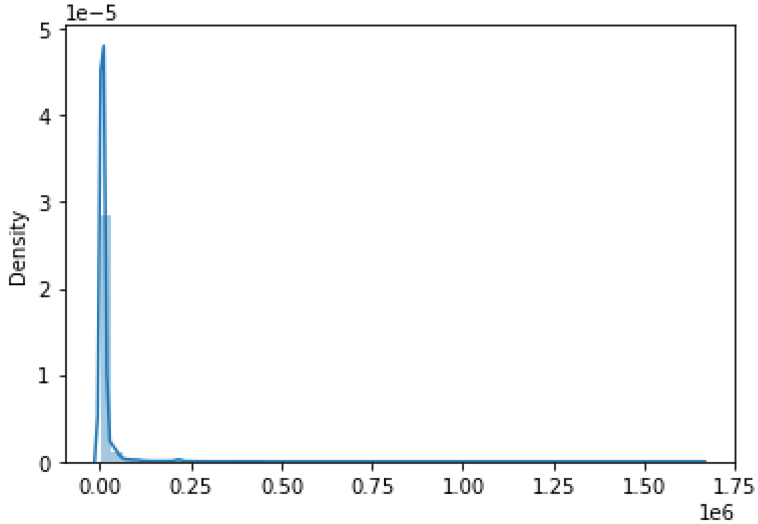




In [51]:

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

Out[51]: <AxesSubplot:ylabel='Density'>



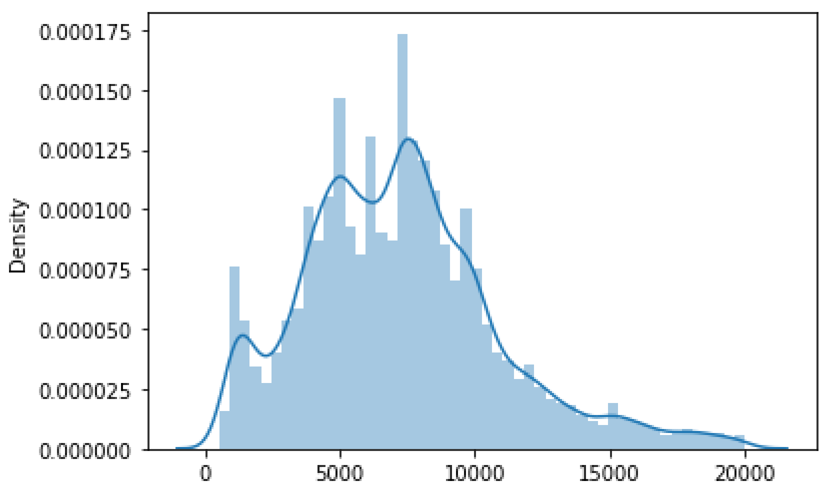
In [52]:

```
df=df[(df["sqft_lot"]<20000)]
```

In [53]:

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

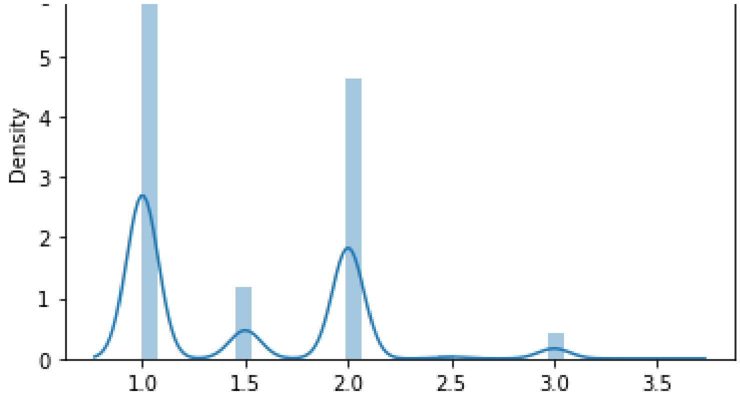
Out[53]: <AxesSubplot:ylabel='Density'>



In [54]:

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

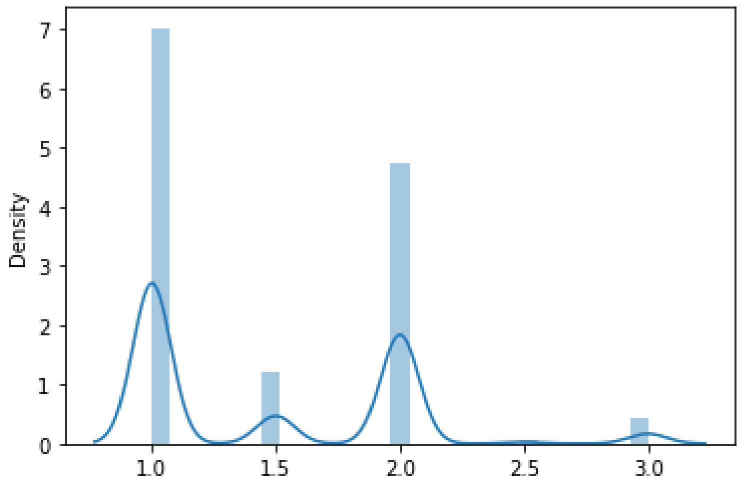
Out[54]: <AxesSubplot:ylabel='Density'>



In [55]:
```
df=df[(df["floors"]<3.5)]
```

In [56]:
```
ut.plot(df,["floors"])
```

Out[56]: <AxesSubplot:ylabel='Density'>



In [57]:
```
df["condition1"].unique()
```

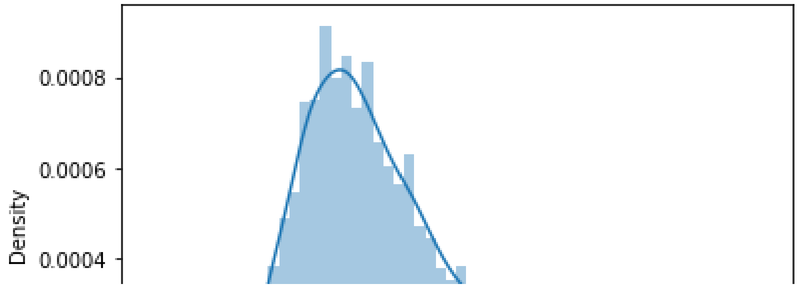
Out[57]: array([1, 2])

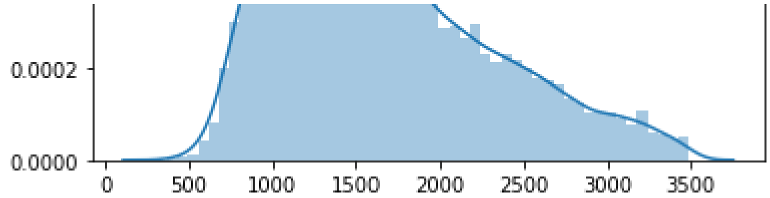
list = [] for i in list:

df["grade"].unique()

In [58]:
```
ut.plot(df,["sqft_above"])
```

Out[58]: <AxesSubplot:ylabel='Density'>





In [59]:

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

Out[59]:

```
1425       1
3087       1
2198       1
1333       1
2531       1
        ...
1140     169
1220     173
1200     190
1010     194
1300     195
Name: sqft_above, Length: 658, dtype: int64
```

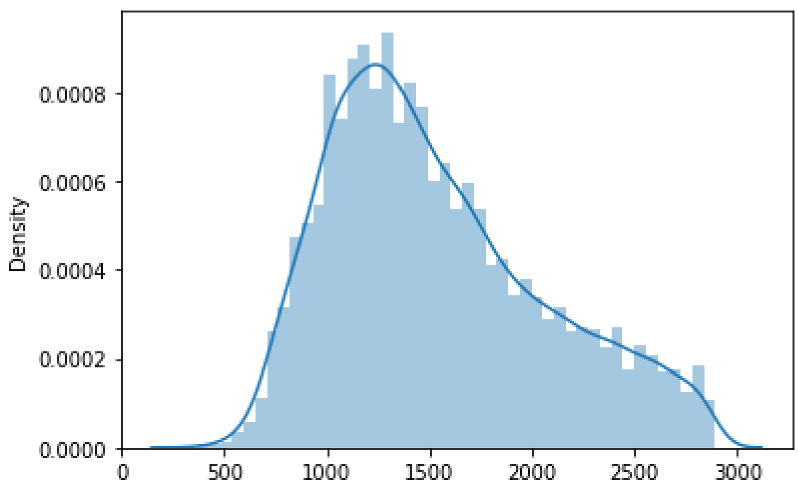
In [60]:

```
df=df[(df['sqft_above'] <2900)]
```

In [61]:

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

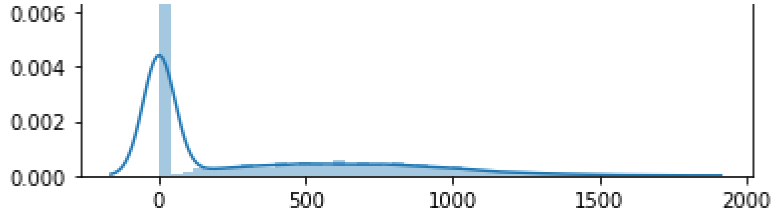
Out[61]: <AxesSubplot:ylabel='Density'>



In [62]:

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

Out[62]: <AxesSubplot:ylabel='Density'>



In [63]:

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

Out[63]:

```
248.0     1
207.0     1
283.0     1
556.0     1
266.0     1
143.0     1
65.0      1
508.0     1
10.0      1
516.0     1
862.0     1
602.0     1
20.0      1
274.0     1
276.0     1
176.0     1
295.0     1
225.0     1
172.0     1
1730.0    1
792.0     1
1620.0    1
1630.0    1
652.0     1
415.0     1
1525.0    1
518.0     1
861.0     1
906.0     1
784.0     1
1750.0    1
1135.0    1
1710.0    1
704.0     1
875.0     1
506.0     1
243.0     1
235.0     2
435.0     2
1520.0    2
515.0     2
1660.0    2
1690.0    2
414.0     2
1530.0    2
1560.0    2
1680.0    2
1570.0    2
1700.0    2
1720.0    2
```

```
265.0          3
1610.0         3
1490.0         3
1600.0         4
1550.0         4
1480.0         4
40.0           4
1540.0         5
1650.0         5
145.0          5
1430.0         5
1510.0         5
1470.0         5
1590.0         5
1310.0         5
1410.0         6
1460.0         6
70.0           6
1360.0         6
1350.0         7
1580.0         7
1440.0         7
1420.0         8
1320.0         9
1450.0         9
1290.0         9
60.0          10
1380.0        10
1390.0        10
50.0          11
230.0         11
1500.0        11
1260.0        11
1340.0        12
1280.0        12
1330.0        13
1240.0        13
1210.0        13
1370.0        14
1150.0        16
1190.0        16
1270.0        17
1140.0        17
110.0         18
1160.0        18
1400.0        18
1170.0        19
90.0          20
1230.0        20
80.0          20
1220.0        21
1130.0        21
1110.0        22
410.0         24
160.0         24
210.0         24
1180.0        25
130.0         25
170.0         26
490.0         26
```

```
1300.0       27
1080.0       27
1090.0       29
1120.0       30
190.0        32
1050.0       32
930.0        33
710.0        33
1250.0       35
970.0        35
320.0        35
260.0        36
180.0        36
590.0        36
370.0        37
870.0        37
610.0        38
1070.0       38
1030.0       38
1020.0       38
690.0        39
990.0        39
100.0        39
270.0        41
150.0        41
660.0        42
1200.0       42
220.0        42
980.0        43
390.0        43
1040.0       43
1060.0       44
330.0        44
810.0        45
1010.0       46
830.0        46
470.0        46
540.0        46
510.0        47
890.0        47
250.0        49
950.0        50
120.0        50
140.0        50
790.0        51
820.0        51
1100.0       52
760.0        52
920.0        52
310.0        52
570.0        53
560.0        54
460.0        54
640.0        54
340.0        54
730.0        55
880.0        55
940.0        55
740.0        55
960.0        56
910.0        56
```

```
910.0       56
630.0       58
850.0       59
280.0       60
350.0       60
430.0       62
860.0       63
440.0       64
520.0       64
680.0       64
290.0       64
770.0       64
360.0       65
380.0       66
650.0       67
780.0       67
670.0       68
550.0       70
240.0       71
840.0       72
420.0       73
580.0       74
620.0       82
530.0       90
720.0       90
480.0       92
450.0       92
750.0       94
200.0       97
1000.0     105
900.0      112
300.0      133
400.0      167
800.0      172
700.0      183
600.0      186
500.0      191
0.0      10267
Name: sqft_basement, dtype: int64
```

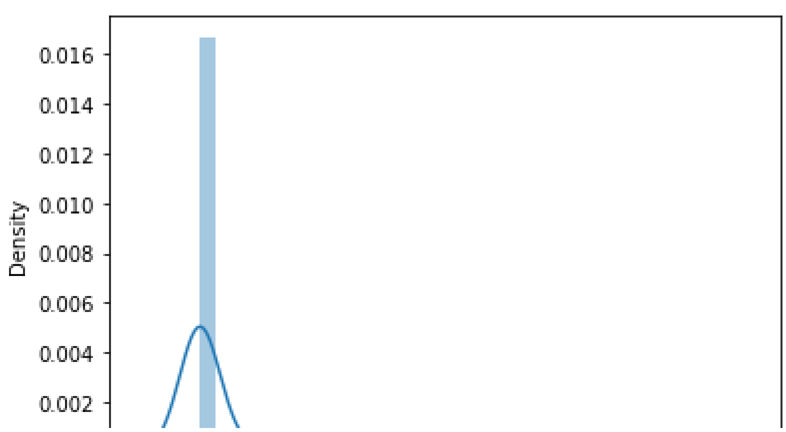
In [64]:

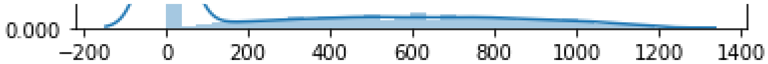
```python
df=df[(df["sqft_basement"]<1200)]
```

In [65]:

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

Out[65]: <AxesSubplot:ylabel='Density'>





In [66]:

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

Out[66]:

```
1935     16
1934     16
1933     17
1902     25
1901     25
1932     26
2015     27
1936     30
1938     39
1904     42
1903     43
1913     46
1914     49
1931     50
1917     51
1937     53
1915     55
1907     57
1911     63
1905     64
1921     65
1912     69
1916     71
1982     71
1900     73
1923     75
1930     77
1908     78
1919     80
1922     82
1971     82
1945     83
1906     83
1909     87
1939     89
1970     91
1920     91
1995    102
1973    103
1927    104
2011    107
1918    108
1929    108
1997    110
1946    113
1928    114
1972    115
2010    116
1910    120
1996    120
1974    123
1924    127
2000    128
1964    128
```

```
1991    129
1944    132
2012    133
1992    136
1940    137
1993    139
2013    140
1965    141
1925    146
1941    147
1975    149
1980    149
1943    150
1981    151
1998    153
1983    155
1985    156
2002    156
1986    157
1984    159
1957    163
1958    164
1949    164
1926    167
1956    172
1976    175
1988    179
1994    180
1999    181
1953    181
2001    185
1989    185
1961    186
1952    188
1966    195
1948    196
1960    197
1951    201
1963    201
1942    201
2009    203
1950    208
1987    209
1990    211
1947    224
1969    225
1955    230
1979    245
1962    249
1954    264
1978    277
1959    280
2008    280
1967    287
2004    291
1977    295
2006    298
2007    300
2003    305
1968    309
2005    314
```

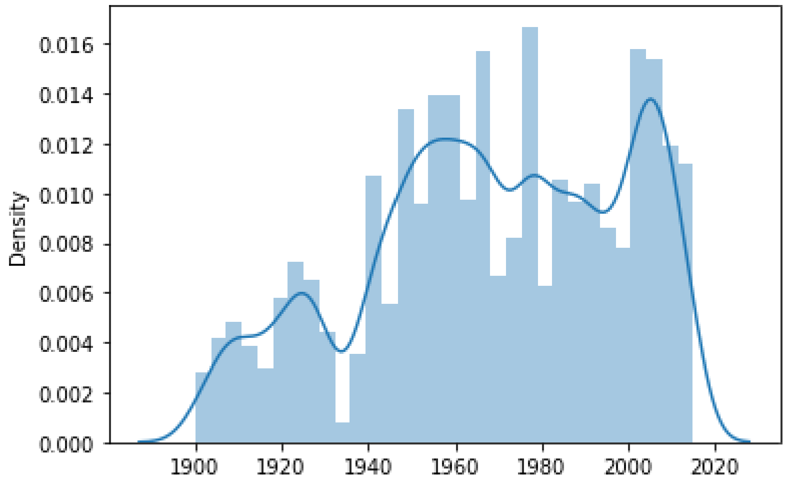
```
2014    364
Name: yr_built, dtype: int64
```

In [67]:

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

Out[67]:

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



In [68]:

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

Out[68]:

```
1957.0         1
1934.0         1
1959.0         1
1944.0         1
1948.0         1
1946.0         1
1950.0         1
1953.0         1
1962.0         1
1951.0         1
1971.0         1
1976.0         1
1955.0         2
1960.0         2
1940.0         2
1967.0         2
1981.0         2
1972.0         2
1974.0         2
1956.0         2
1965.0         3
1963.0         3
1978.0         3
1958.0         3
1945.0         3
1964.0         3
1975.0         3
1969.0         3
1973.0         3
1982.0         4
1977.0         4
1968.0         4
1980.0         5
```

```
1979.0        5
1995.0        6
2015.0        7
1970.0        7
2012.0        7
1992.0        7
1998.0        8
1996.0        8
1987.0        8
1999.0        8
2011.0        9
1985.0        9
1994.0        9
1986.0        9
2010.0        9
1988.0       10
2001.0       10
1993.0       10
1997.0       10
1990.0       10
2008.0       11
2002.0       11
1984.0       11
1991.0       12
1989.0       13
1983.0       13
2006.0       15
2007.0       16
2009.0       16
2004.0       17
2000.0       20
2003.0       20
2005.0       22
2013.0       25
2014.0       60
0.0       16011
Name: yr_renovated, dtype: int64
```

In [69]:

```
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 is
```

In [70]:

```
df["renovated"].value_counts()
```

Out[70]:

```
0    16011
1      520
Name: renovated, dtype: int64
```

In [71]:

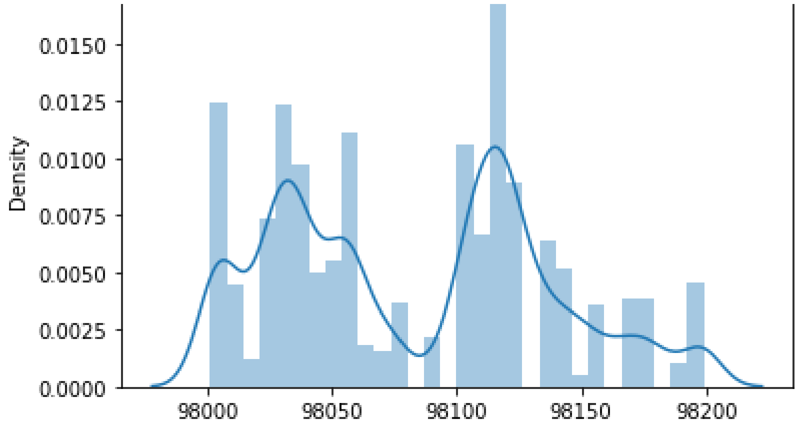
```
df =df.drop(["yr_renovated"],axis=1)

#drop "yr_renovated"
```

In [72]:

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

Out[72]:

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



In [73]:

```
df['zipcode'].value_counts()
```

Out[73]:

```
98103    571
98115    535
98117    519
98034    463
98133    458
98118    457
98038    448
98023    419
98052    411
98042    406
98155    388
98125    366
98058    354
98126    339
98056    333
98106    311
98144    305
98116    304
98059    299
98033    297
98001    290
98122    271
98006    266
98074    261
98029    258
98146    256
98199    254
98107    251
98003    244
98136    242
98031    240
98198    239
98178    236
98092    236
98168    228
98055    227
98008    227
98028    226
98030    225
98027    210
98112    207
98065    197
98053    192
```

```
98166    191
98105    188
98002    185
98177    183
98108    172
98119    168
98004    167
98011    157
98022    145
98072    139
98045    135
98040    134
98019    128
98188    117
98075    110
98032    104
98007    104
98005     99
98109     99
98102     90
98148     53
98014     50
98010     50
98077     29
98070     28
98024     25
98039     15
Name: zipcode, dtype: int64
```

In [74]:

```
df['zipcode'].nunique()
```

Out[74]:

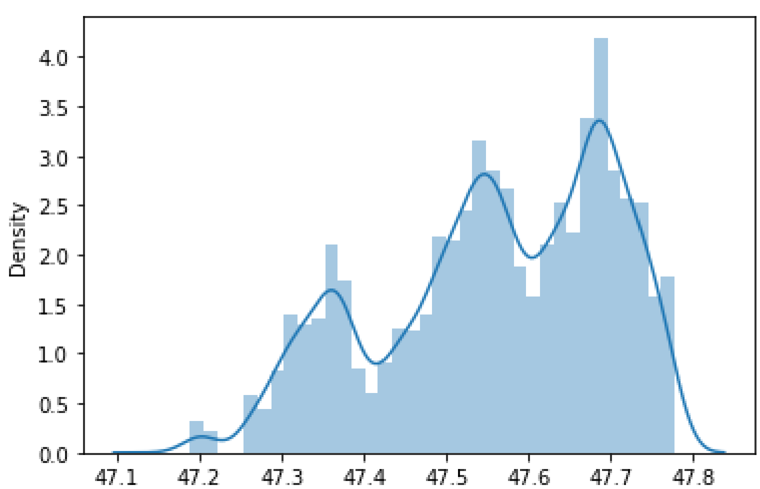
```
70
```

In [75]:

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

Out[75]:

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

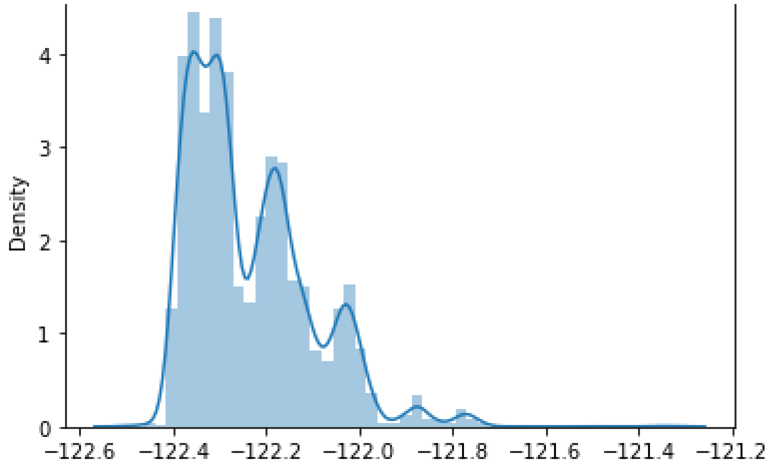


In [76]:

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

Out[76]:

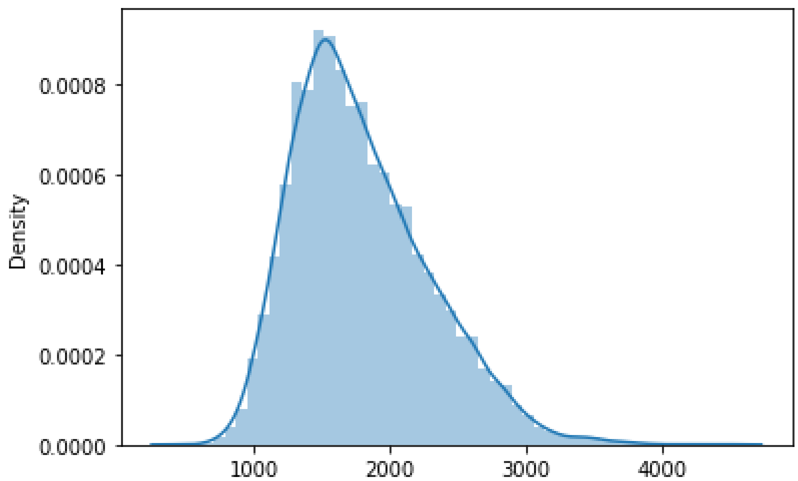
```
<AxesSubplot:ylabel='Density'>
```



In [77]:

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

Out[77]: <AxesSubplot:ylabel='Density'>



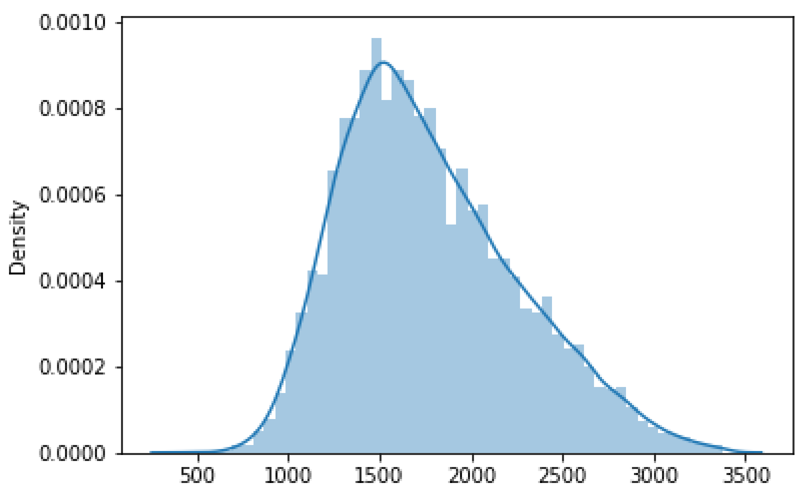
In [78]:

```
df=df[(df['sqft_living15']<3400)]
```

In [79]:

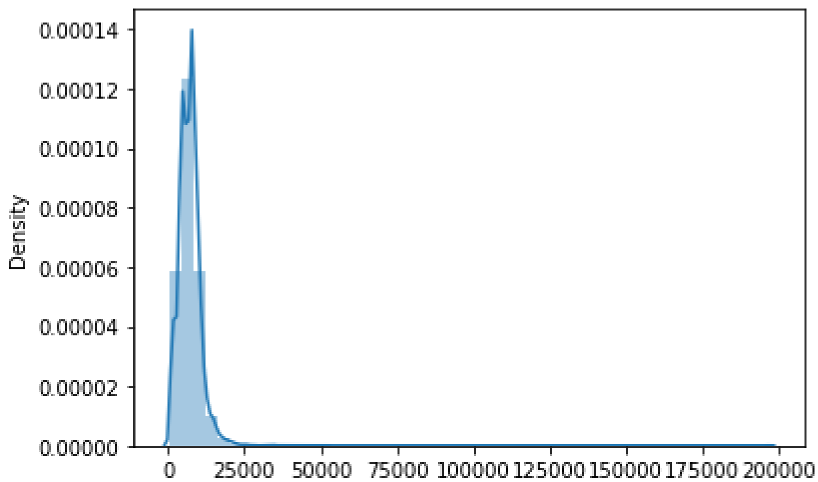
```
ut.plot(df,['sqft_living15'])
```

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



In [80]:
```
ut.plot(df,["sqft_lot15"])
```

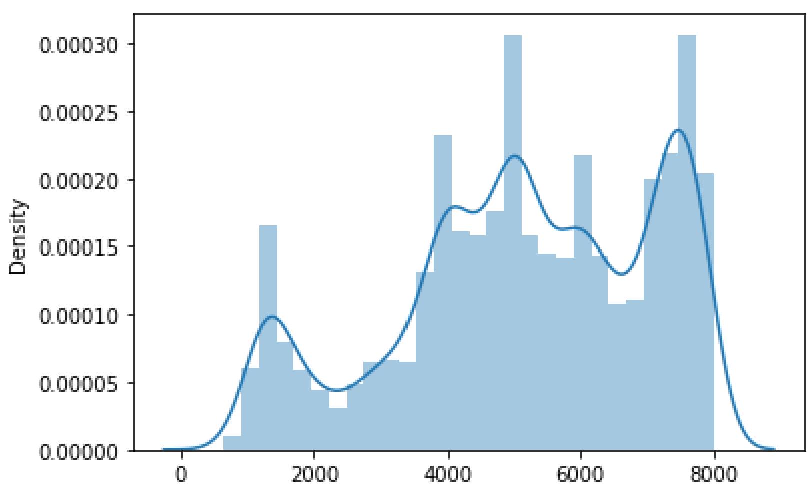
Out[80]: <AxesSubplot:ylabel='Density'>



In [81]:
```
df=df[(df["sqft_lot15"]<8000)]
```

In [82]:
```
ut.plot(df,["sqft_lot15"])
```

Out[82]: <AxesSubplot:ylabel='Density'>



Explore the data

In [83]:
```
df.describe().T
```

Out[83]:

	count	mean	std	min	25%	
price	10529.0	466858.469655	217031.551785	102500.0000	315000.0000	42500
bedrooms	10529.0	3.124513	0.811660	1.0000	3.0000	
bathrooms	10529.0	1.979224	0.686776	0.5000	1.5000	
sqft_living	10529.0	1733.007978	578.861277	370.0000	1290.0000	167

sqft_lot	10529.0	5249.797037	2384.637208	520.0000	3800.0000	512
floors	10529.0	1.560737	0.570238	1.0000	1.0000	
grade	10529.0	7.351125	0.876973	4.0000	7.0000	
sqft_above	10529.0	1505.343717	538.305979	370.0000	1080.0000	140
sqft_basement	10529.0	222.150917	332.701595	0.0000	0.0000	
yr_built	10529.0	1967.776712	34.754510	1900.0000	1941.0000	197
zipcode	10529.0	98092.229936	48.797051	98001.0000	98045.0000	9810
lat	10529.0	47.570408	0.128475	47.1934	47.5051	4
long	10529.0	-122.255095	0.127961	-122.4560	-122.3540	-12
sqft_living15	10529.0	1717.520847	464.329666	620.0000	1380.0000	164
sqft_lot15	10529.0	5153.735208	1922.835439	651.0000	4000.0000	518
waterfront1	10529.0	0.000570	0.023866	0.0000	0.0000	
condition1	10529.0	1.084623	0.278334	1.0000	1.0000	
renovated	10529.0	0.033431	0.179769	0.0000	0.0000	

In [84]:

```
df.shape
```

Out[84]:

```
(10529, 18)
```

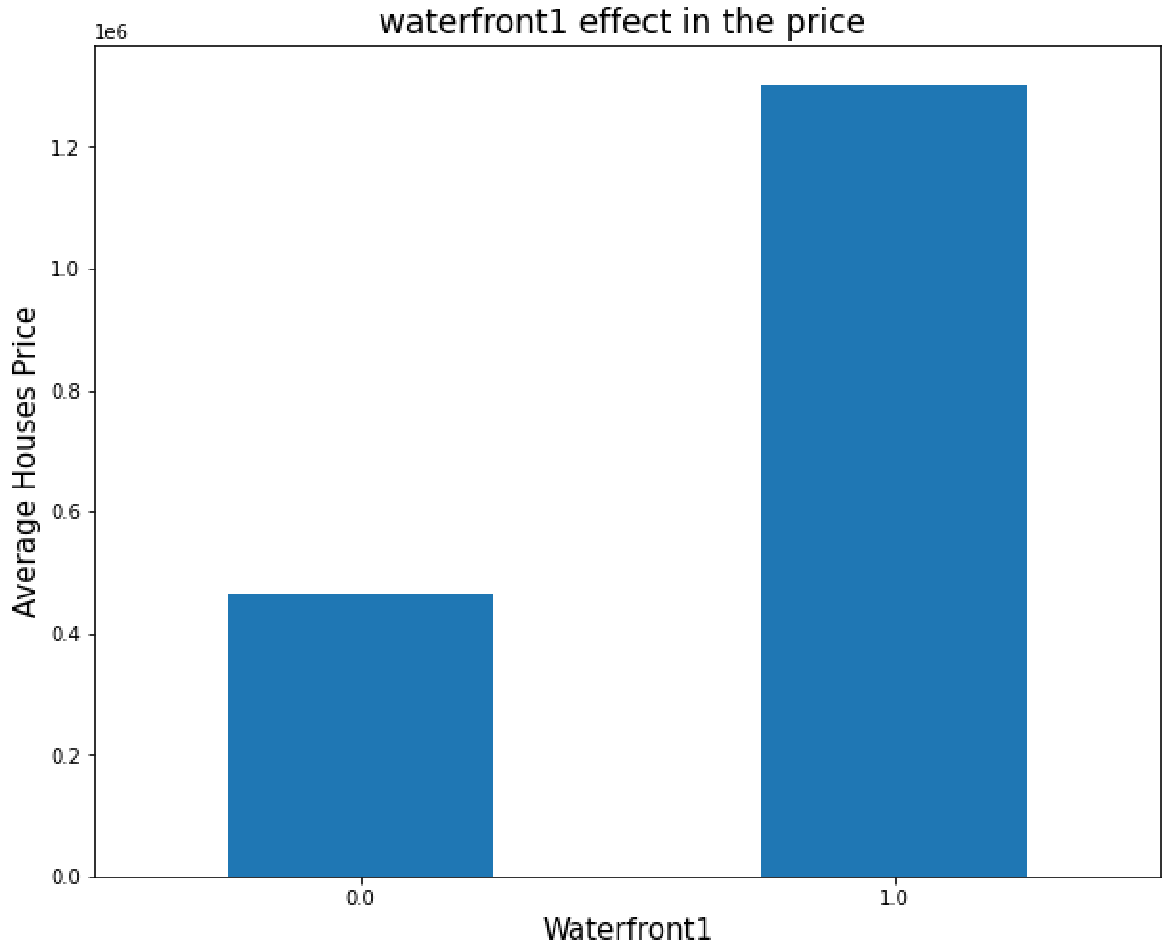
In [85]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10529 entries, 0 to 21596
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          10529 non-null  float64
 1   bedrooms       10529 non-null  int64  
 2   bathrooms      10529 non-null  float64
 3   sqft_living    10529 non-null  int64  
 4   sqft_lot       10529 non-null  int64  
 5   floors         10529 non-null  float64
 6   grade          10529 non-null  float64
 7   sqft_above     10529 non-null  int64  
 8   sqft_basement  10529 non-null  float64
 9   yr_built       10529 non-null  int64  
 10  zipcode        10529 non-null  int64  
 11  lat            10529 non-null  float64
 12  long           10529 non-null  float64
 13  sqft_living15  10529 non-null  int64  
 14  sqft_lot15     10529 non-null  int64  
 15  waterfront1    10529 non-null  float64
 16  condition1     10529 non-null  int64  
 17  renovated      10529 non-null  int64  
dtypes: float64(8), int64(10)
memory usage: 1.5 MB
```

In [86]:

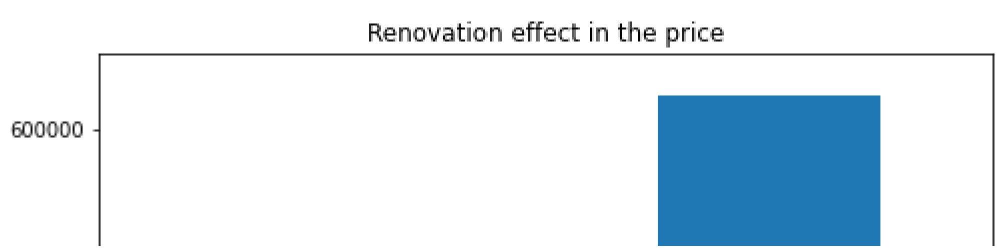
```
# plotting houses to the mean of price
df.groupby("waterfront1")["price"].mean().plot(kind="bar",figsize=(10,8));
plt.title("waterfront1 effect in the price ", fontsize=17)
plt.ylabel("Average Houses Price",fontsize=15)
plt.xlabel("Waterfront1",fontsize=15)
plt.xticks(rotation=0)
plt.show()
#the houses with waterfront selling price are higher than one without wate
```

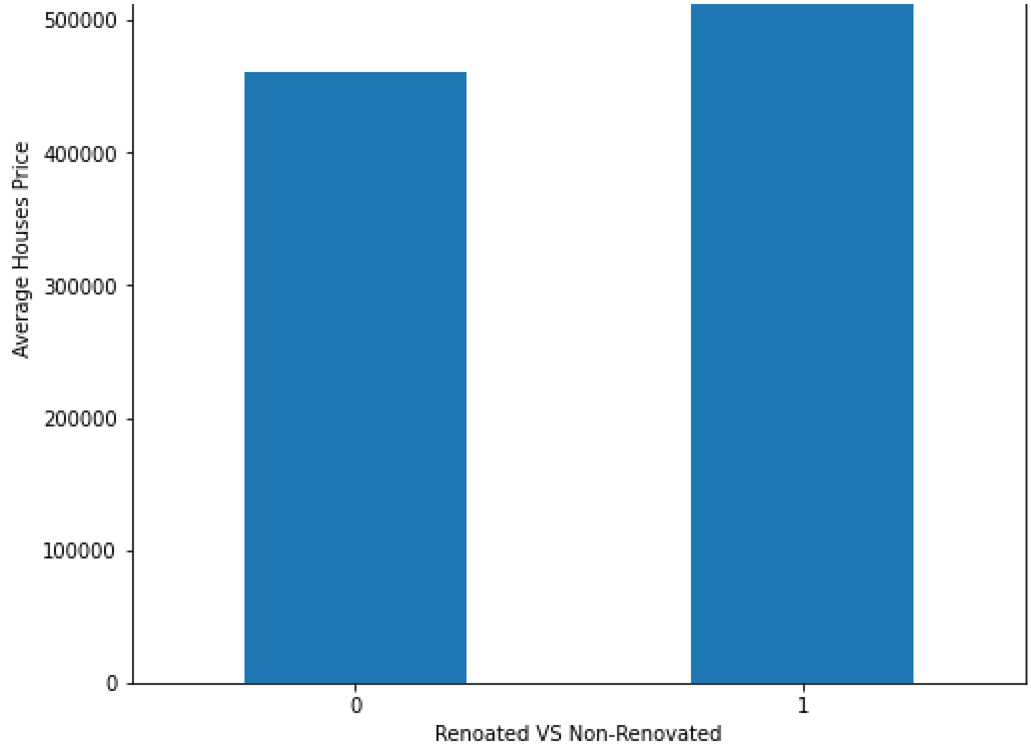


In [87]:

```
# 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
```

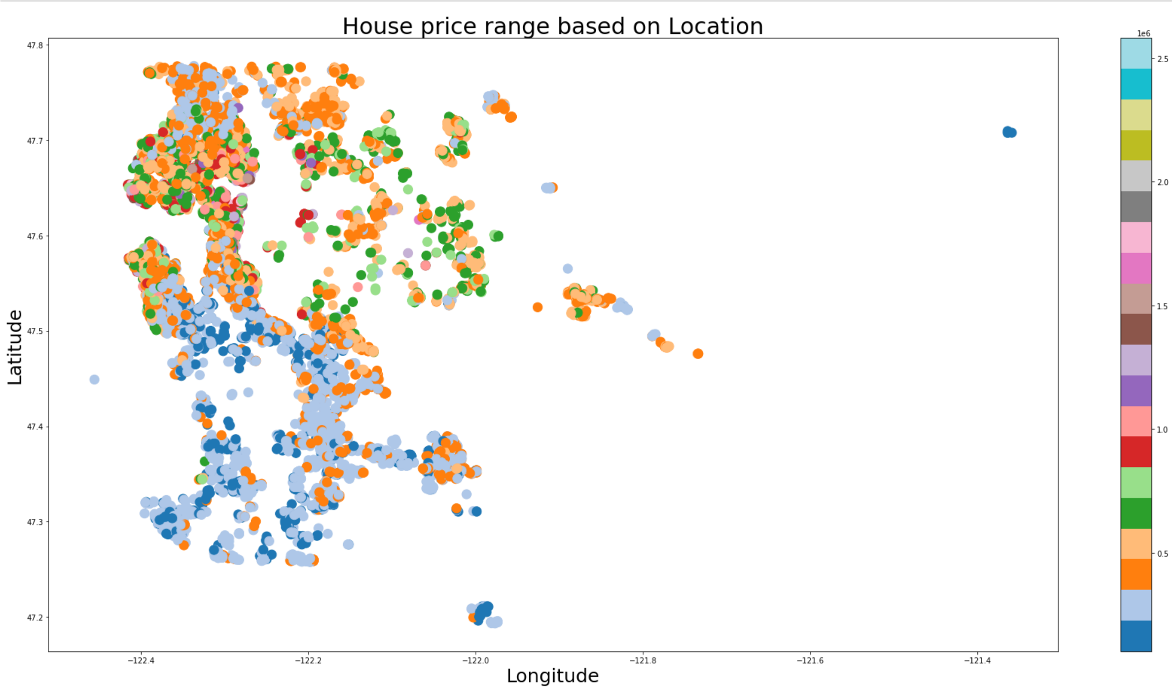
Out[87]: (array([0, 1]), [Text(0, 0, '0'), Text(1, 0, '1')])





In [88]:

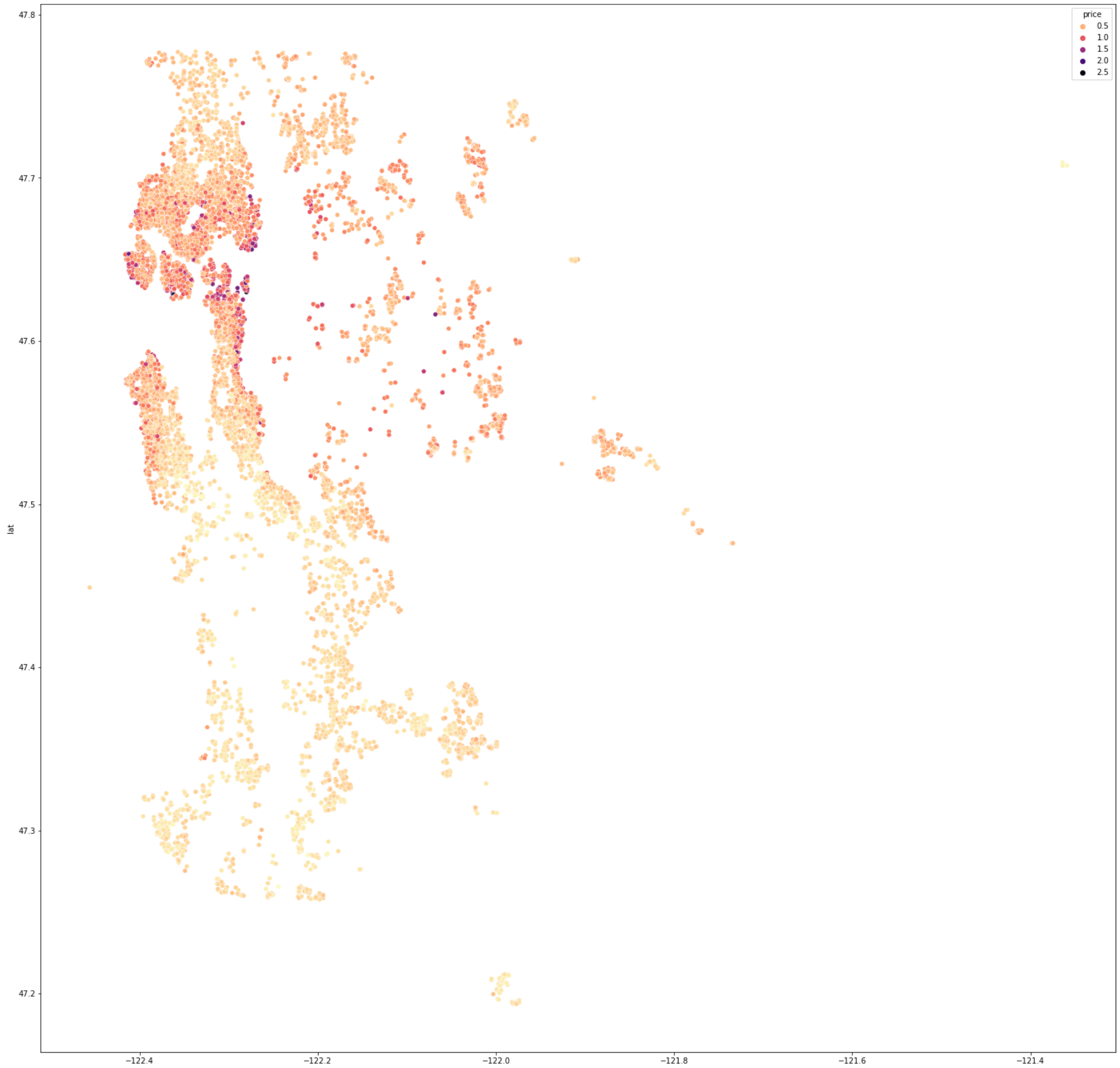
```
#Visualizing Longitude to Latitude to check how the price vary by location

plt.figure(figsize=(30,15))
plt.scatter(x=df['long'], y=df['lat'], c =df["price"], cmap='tab20',marker
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 [89]:

```
plt.figure(figsize=(25,25))
```

```
plt.figure(figsize=(15,15))

sns.scatterplot(data=df, x="long", y="lat", hue="price", palette="magma_r'
plt.savefig("images/HousepricebasedonLocation1.png")
```



In [90]:

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

Out[90]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	g
price	1.000000	0.217635	0.316676	0.524187	-0.119704	0.199558	0.54
bedrooms	0.217635	1.000000	0.459647	0.620960	0.199903	0.160663	0.27
bathrooms	0.316676	0.459647	1.000000	0.674282	-0.133333	0.544056	0.5
sqft_living	0.524187	0.620960	0.674282	1.000000	0.147021	0.315873	0.59
sqft_lot	-0.119704	0.199903	-0.133333	0.147021	1.000000	-0.460309	-0.1
floors	0.199558	0.160663	0.544056	0.315873	-0.460309	1.000000	0.49
grade	0.545287	0.274848	0.577811	0.596588	-0.161140	0.499053	1.00
sqft_above	0.367255	0.514048	0.611275	0.822041	0.120000	0.509824	0.59
sqft_basement	0.307694	0.240133	0.178605	0.397831	0.060177	-0.268194	0.07

sqft_basement	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
yr_built	-0.149486	0.122484	0.546071	0.249853	-0.167701	0.540712	0.43
zipcode	0.183199	-0.158983	-0.240784	-0.174464	-0.175007	-0.117675	-0.10
lat	0.448126	-0.126717	-0.092086	-0.059186	-0.215156	0.006946	0.07
long	-0.147717	0.150683	0.267263	0.235449	0.165764	0.144362	0.11
sqft_living15	0.390720	0.376324	0.472416	0.671648	0.150245	0.240301	0.51
sqft_lot15	-0.148172	0.189503	-0.146611	0.123844	0.824939	-0.485599	-0.16
waterfront1	0.092035	-0.013470	0.013761	0.020846	-0.008139	0.014905	0.01
condition1	0.116600	0.048797	-0.034529	0.021195	0.043614	-0.137415	-0.09
renovated	0.136677	0.019641	0.036016	0.054588	-0.001758	-0.015177	0.02

In [91]:

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

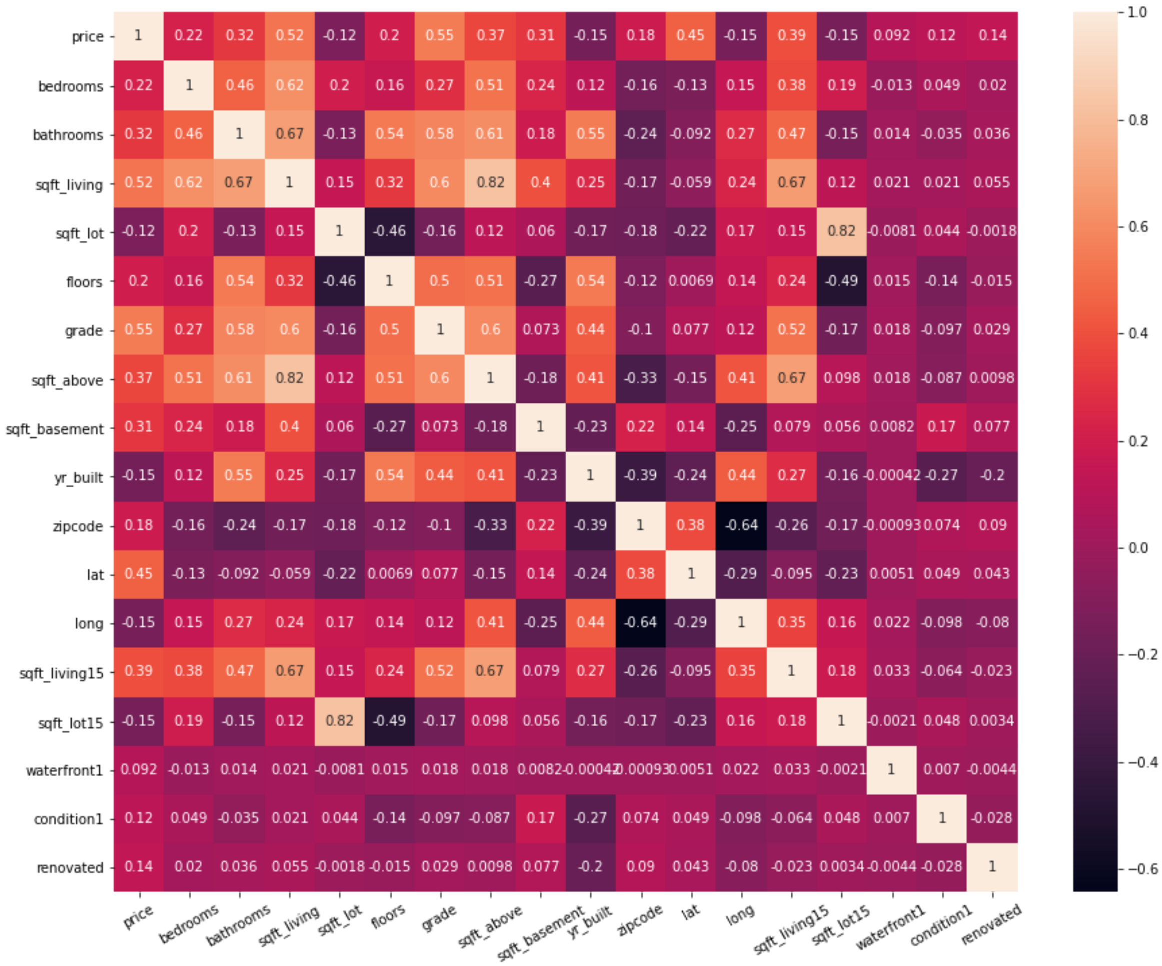
Out[91]:

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

In [92]:

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

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



In [93]:

```
# 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=False)

# zip the variable name columns (Which were only named level_0 and level_
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()
```

Out[93]:

pairs	cc
(price, price)	1.000000
(sqft_lot, sqft_lot15)	0.824939
(sqft_above, sqft_living)	0.822041
(sqft_living, bathrooms)	0.674282
(sqft_living15, sqft_above)	0.672670

In [94]:

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

Out[94]:

pairs	cc
(sqft_lot, sqft_lot15)	0.824939
(sqft_above, sqft_living)	0.822041

In [95]:

```
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 ser
#land lots of the nearest 15 neighbors
# for sqft_above, sqft_living: i dropped the sqft_above because the square
#living space is more important than the qft_above basement
```

In [96]:

```
df.corr()
```

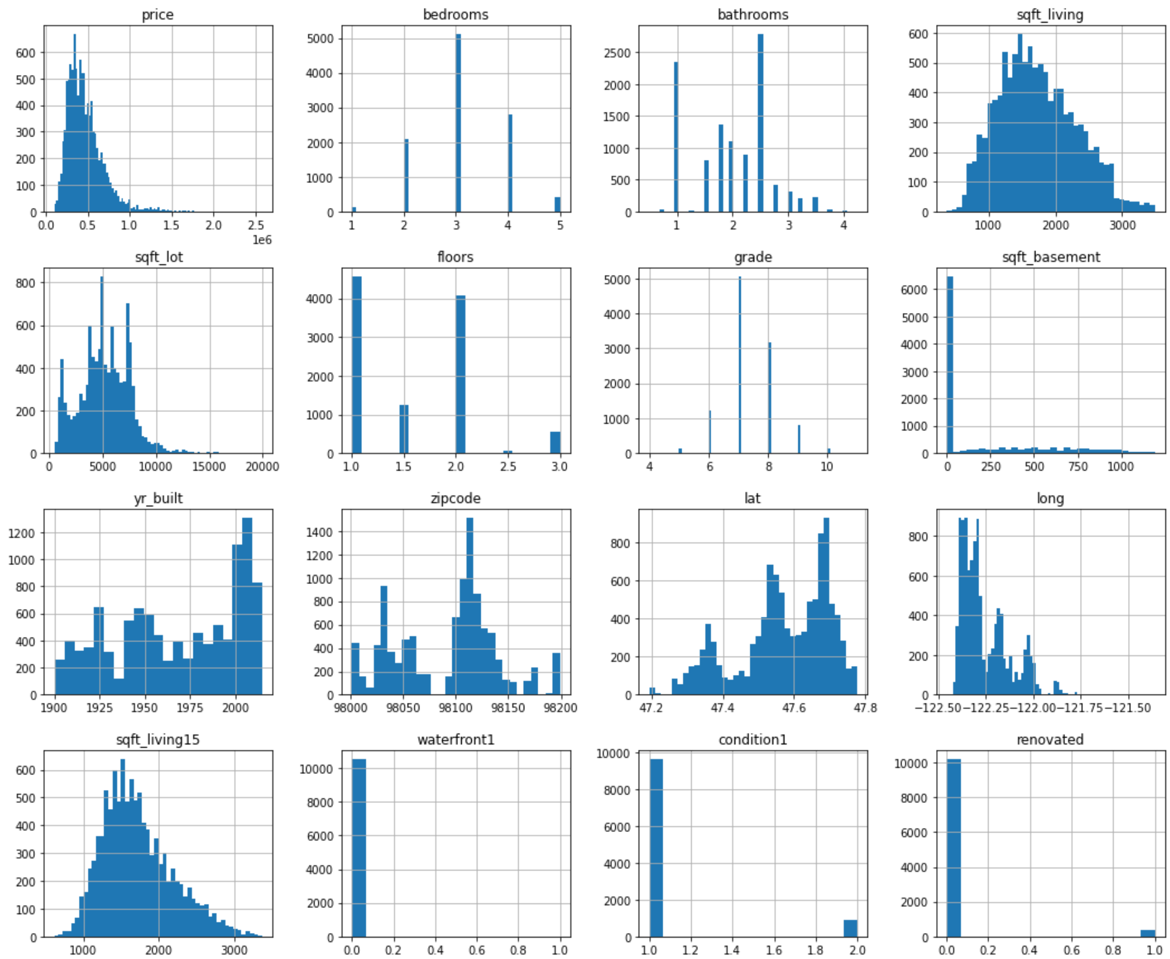
Out[96]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	(
price	1.000000	0.217635	0.316676	0.524187	-0.119704	0.199558	0.54
bedrooms	0.217635	1.000000	0.459647	0.620960	0.199903	0.160663	0.27
bathrooms	0.316676	0.459647	1.000000	0.674282	-0.133333	0.544056	0.5
sqft_living	0.524187	0.620960	0.674282	1.000000	0.147021	0.315873	0.59
sqft_lot	-0.119704	0.199903	-0.133333	0.147021	1.000000	-0.460309	-0.1
floors	0.199558	0.160663	0.544056	0.315873	-0.460309	1.000000	0.49
grade	0.545287	0.274848	0.577811	0.596588	-0.161140	0.499053	1.00
sqft_basement	0.307694	0.240133	0.178605	0.397831	0.060177	-0.268194	0.07
yr_built	-0.149486	0.122484	0.546071	0.249853	-0.167701	0.540712	0.4
zipcode	0.183199	-0.158983	-0.240784	-0.174464	-0.175007	-0.117675	-0.1
lat	0.448126	-0.126717	-0.092086	-0.059186	-0.215156	0.006946	0.07
long	-0.147717	0.150683	0.267263	0.235449	0.165764	0.144362	0.11
sqft_living15	0.390720	0.376324	0.472416	0.671648	0.150245	0.240301	0.5

waterfront1	0.092035	-0.013470	0.013761	0.020846	-0.008139	0.014905	0.01
condition1	0.116600	0.048797	-0.034529	0.021195	0.043614	-0.137415	-0.0
renovated	0.136677	0.019641	0.036016	0.054588	-0.001758	-0.015177	0.02

In [97]:

```
# check how our histograms are looking

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



Model

Baseline Model

In [98]:

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

In [99]:

```
# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r
```

In [100…

```
# 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 [101…

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

Out[101…

```
LinearRegression()
```

In [102…

```
y_train_pred = lr.predict(X_train_scaled)
y_test_pred = lr.predict(X_test_scaled)
```

In [103…

```
# Evaluate
ut.evaluate_model(y_train, y_test, y_train_pred, y_test_pred)
```

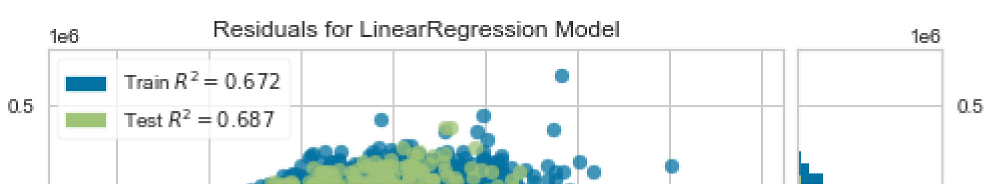
```
Train R2: 0.672
Test R2: 0.687
---
Train MAE: 88905.328
Test MAE: 89051.591
---
Train RMSE: 123298.817
Test RMSE: 125355.358
```

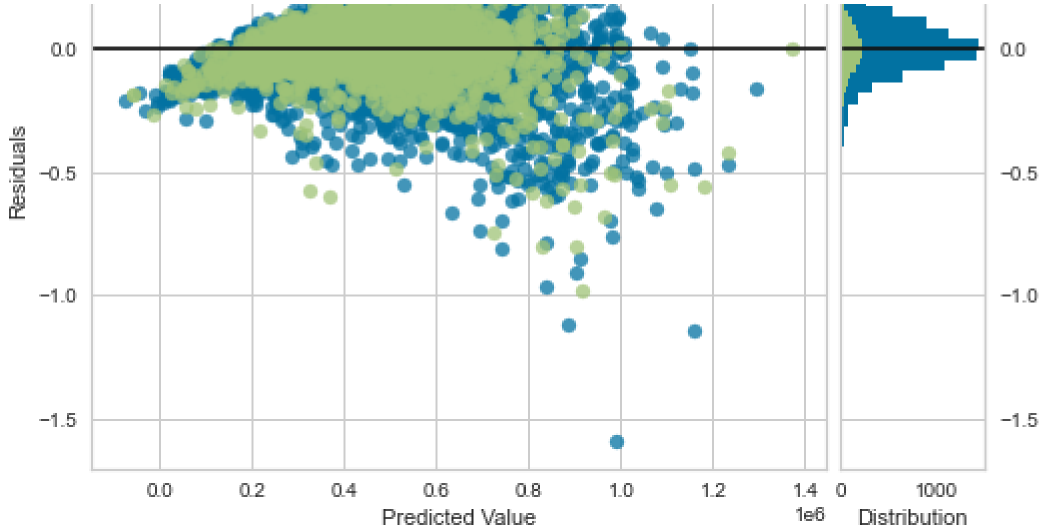
In [104…

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

In [105…

```
# 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()
```



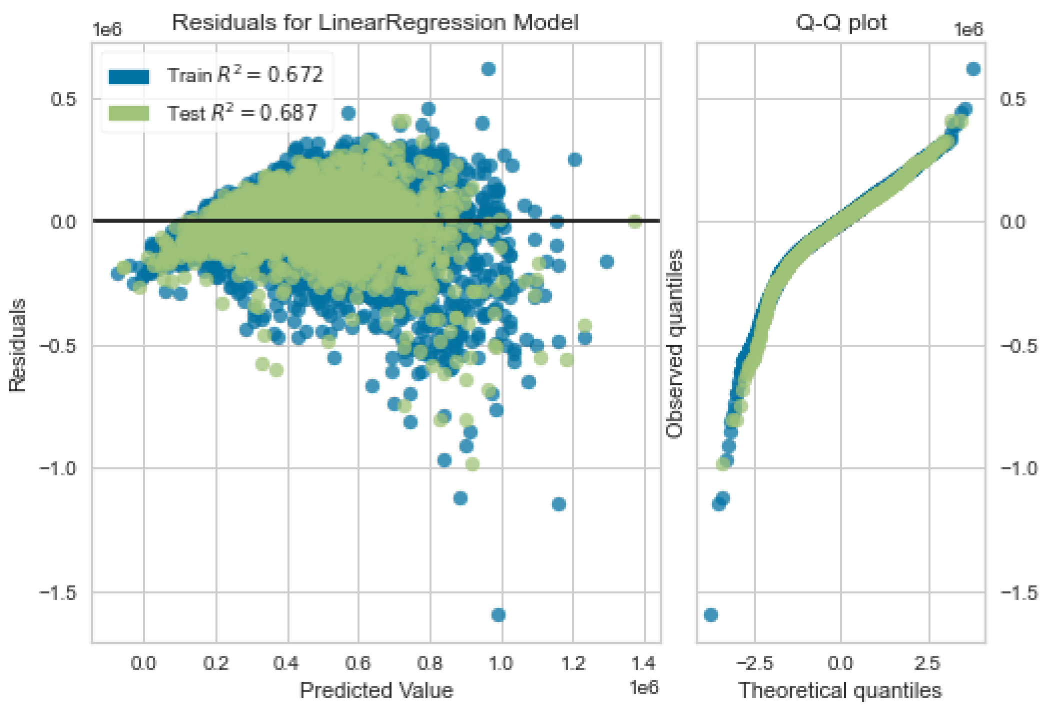


Out[105…

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

In [106…

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



Out[106…

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

In [107…

```python
# Import statsmodels.api as sm
from statsmodels.formula.api import ols
```

In [108…

```python
# The predicted values of the baseline model are not equally scattered, ha
```

In [109…

```python
important_features = ["bedrooms","bathrooms","sqft_living","sqft_lot","flo
```

In [110…

```
# set the features recommended by our feature selector

X_train_perm = X_train[important_features]

predictors_int = sm.add_constant(X_train_perm)
modelols = sm.OLS(y_train, predictors_int).fit()
modelols.summary()
```

Out[110…

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.672
Model:	OLS	**Adj. R-squared:**	0.671
Method:	Least Squares	**F-statistic:**	1147.
Date:	Fri, 01 Apr 2022	**Prob (F-statistic):**	0.00
Time:	10:56:57	**Log-Likelihood:**	-1.1069e+05
No. Observations:	8423	**AIC:**	2.214e+05
Df Residuals:	8407	**BIC:**	2.215e+05
Df Model:	15		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-3.295e+07	3.21e+06	-10.277	0.000	-3.92e+07	-2.67e+07
bedrooms	-1.711e+04	2208.763	-7.748	0.000	-2.14e+04	-1.28e+04
bathrooms	2.175e+04	3394.399	6.407	0.000	1.51e+04	2.84e+04
sqft_living	126.0804	5.171	24.380	0.000	115.943	136.218
sqft_lot	-6.7681	0.749	-9.040	0.000	-8.236	-5.301
floors	9122.9712	3940.404	2.315	0.021	1398.809	1.68e+04
grade	9.653e+04	2329.897	41.433	0.000	9.2e+04	1.01e+05
sqft_basement	-4.1516	5.860	-0.708	0.479	-15.639	7.336
yr_built	-2352.5732	61.695	-38.132	0.000	-2473.511	-2231.635
zipcode	29.8071	37.840	0.788	0.431	-44.369	103.984
lat	5.393e+05	1.18e+04	45.715	0.000	5.16e+05	5.62e+05
long	-6.917e+04	1.52e+04	-4.553	0.000	-9.9e+04	-3.94e+04
sqft_living15	48.1926	4.293	11.227	0.000	39.778	56.607
waterfront1	5.768e+05	6.18e+04	9.337	0.000	4.56e+05	6.98e+05
condition1	3.398e+04	5114.392	6.643	0.000	2.4e+04	4.4e+04
renovated	1.588e+04	7893.579	2.011	0.044	401.996	3.13e+04

Omnibus:	2971.832	**Durbin-Watson:**	2.037
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	29515.252
Skew:	1.403	**Prob(JB):**	0.00

Kurtosis:	11.731	**Cond. No.**	2.34e+08

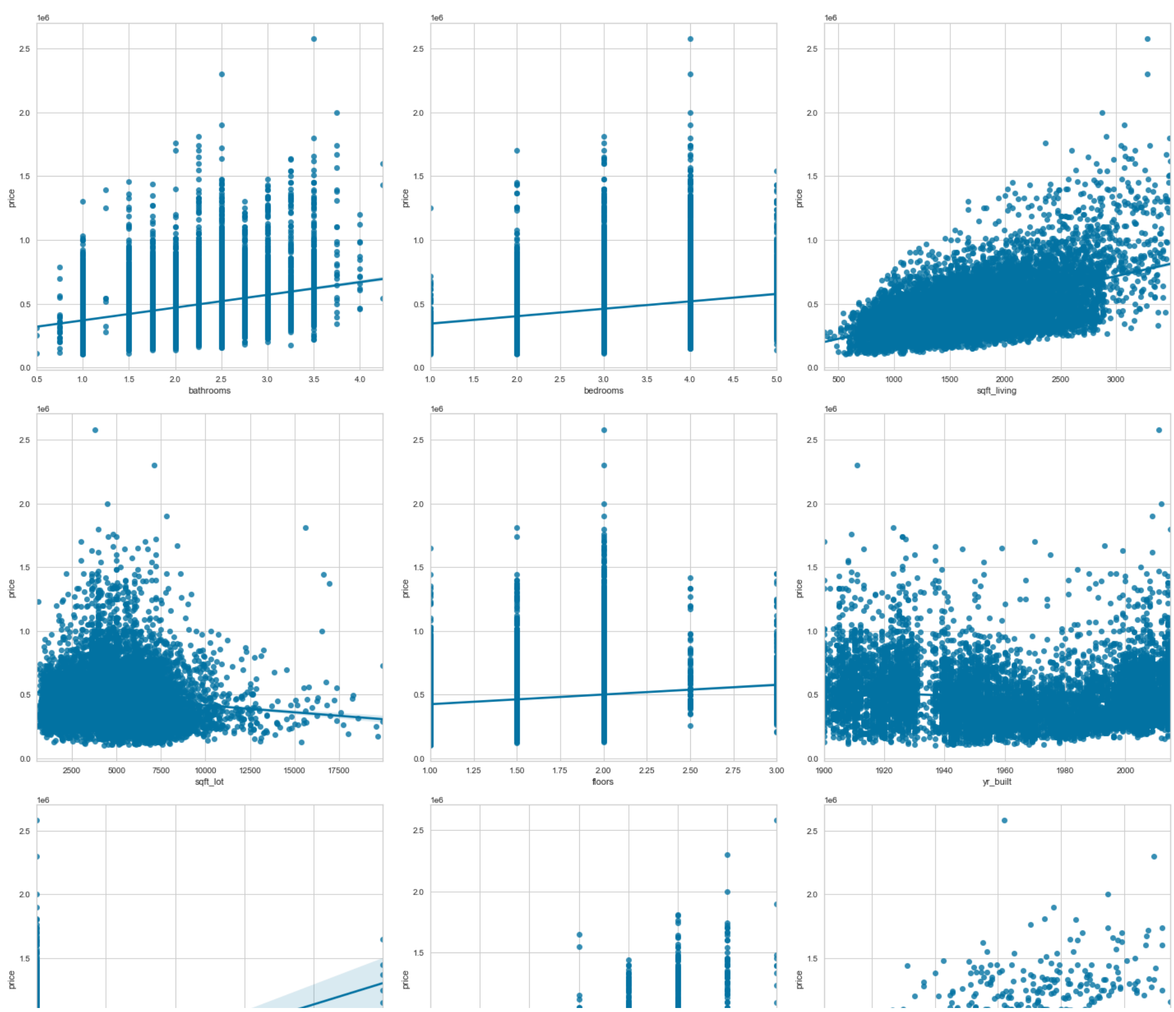
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.34e+08. This might indicate that there are strong multicollinearity or other numerical problems.

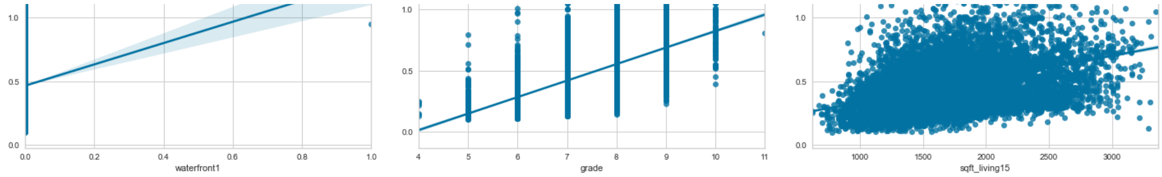
In [111…

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

In [112…

```
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()
```





In [113…

```
# the best fit line is not clear in year built
# floors and sqft_lot are not linearly related to the price
```

Second Model

In [114…

```
from sklearn.preprocessing import PolynomialFeatures
```

In [115…

```
# copy of the original dataframe
df5 = df.copy()
X5 = df.drop("price", axis=1)
y5 = df["price"]
# train test split
X_train5, X_test5, y_train5, y_test5 = train_test_split(X5, y5, test_size=
```

In [116…

```
# Instantiate PolynomialFeatures
poly = PolynomialFeatures(degree=2, interaction_only=False)
```

In [117…

```
# Fit and transform X_train
poly.fit(X_train5)
X_train_poly = poly.transform(X_train5)
X_test_poly = poly.transform(X_test5)
```

In [118…

```
# Instantiate MinMaxScaler
scaler = MinMaxScaler()
# Fit and transform X_train_poly
scaler.fit(X_train_poly)

X_train_poly_sc = scaler.transform(X_train_poly)
X_test_poly_sc = scaler.transform(X_test_poly)
```

In [119…

```
# Instantiate and fit a linear regression model to the polynomial transfor
lr = LinearRegression()

lr.fit(X_train_poly_sc, y_train5)
# grab predictions for train and test set
y_train_poly_preds = lr.predict(X_train_poly_sc)
y_test_poly_preds = lr.predict(X_test_poly_sc)
```

In [120…

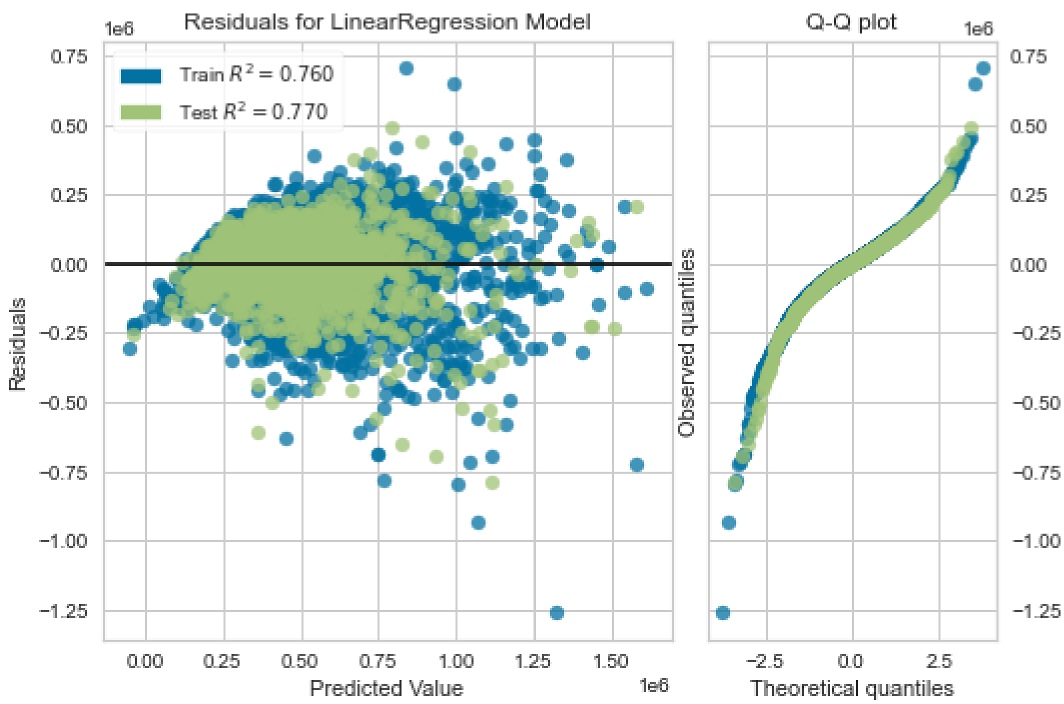
```
# Evaluate
ut.evaluate_model(y_train5,y_test5,y_train_poly_preds, y_test_poly_preds)
```

```
Train R2: 0.760
Test R2: 0.770
```

```
---
Train MAE: 74780.242
Test MAE: 75661.596
---
Train RMSE: 105485.365
Test RMSE: 107607.811
```

In [121…

```
visualizer = ResidualsPlot(lr, hist=False, qqplot=True)
visualizer.fit(X_train_poly_sc, y_train5)  # Fit the training data to the
visualizer.score(X_test_poly_sc, y_test5)  # Evaluate the model on the tes
visualizer.show()
```



Out[121…

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

In [122…

```
# Instantiate MinMaxScaler
scaler = MinMaxScaler()
# Fit and transform X_train_poly
scaler.fit(X_train_poly)

X_train_poly_sc = scaler.transform(X_train_poly)
X_test_poly_sc = scaler.transform(X_test_poly)
```

In [123…

```
# Instantiate and fit a linear regression model to the polynomial transfor
lr = LinearRegression()

lr.fit(X_train_poly_sc, y_train5)
# grab predictions for train and test set
y_train_poly_preds = lr.predict(X_train_poly_sc)
y_test_poly_preds = lr.predict(X_test_poly_sc)
```

In [124…

```
# Evaluate
ut.evaluate_model(y_train5, y_test5, y_train_poly_preds, y_test_poly_preds
```

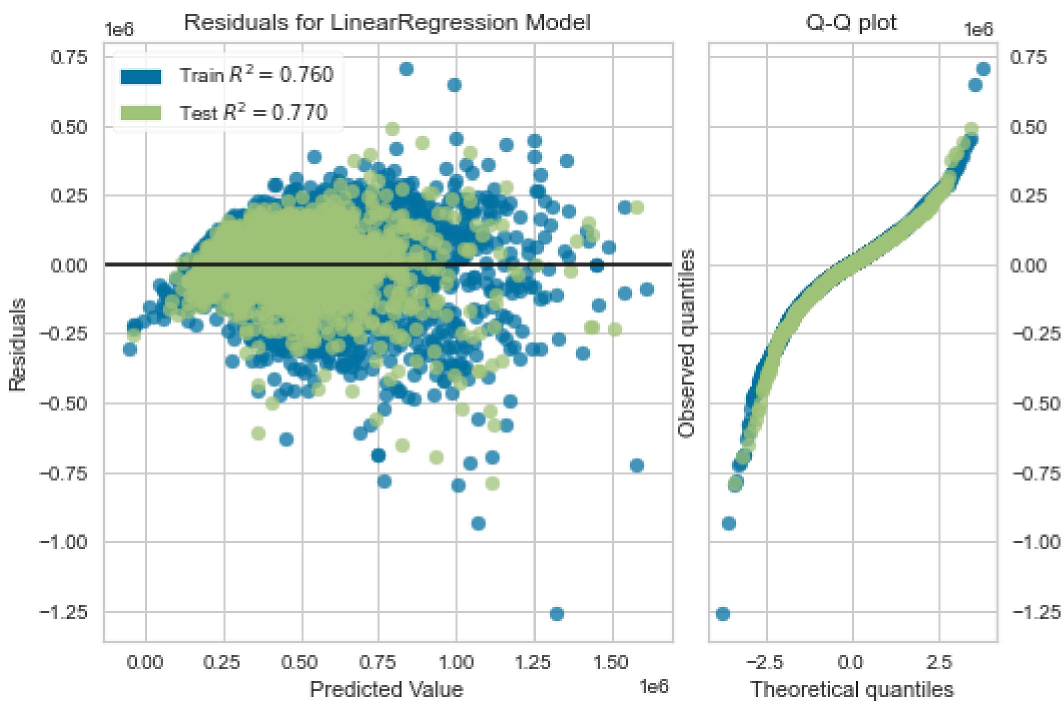
```
Train R2: 0.760
Test R2: 0.770
---
Train MAE: 74780.242
Test MAE: 75661.596
---
Train RMSE: 105485.365
Test RMSE: 107607.811
```

In [125…

```
# the second model can predecit 75 % variance in the price and approximati
```

In [126…

```
visualizer = ResidualsPlot(lr, hist=False, qqplot=True)
visualizer.fit(X_train_poly_sc, y_train5)  # Fit the training data to the
visualizer.score(X_test_poly_sc, y_test5)  # Evaluate the model on the tes
visualizer.show()
```



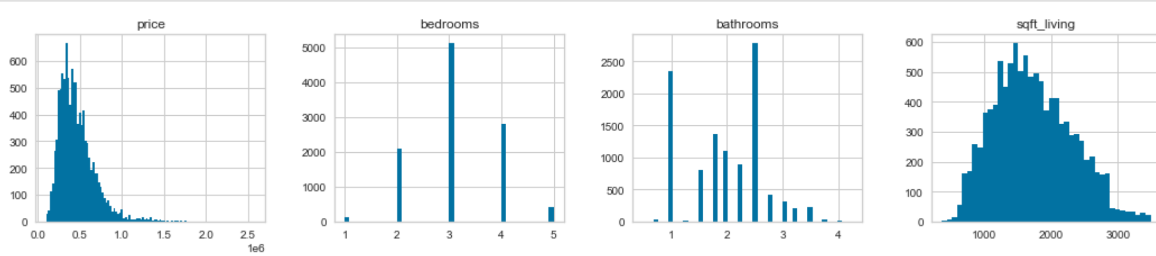
Out[126…

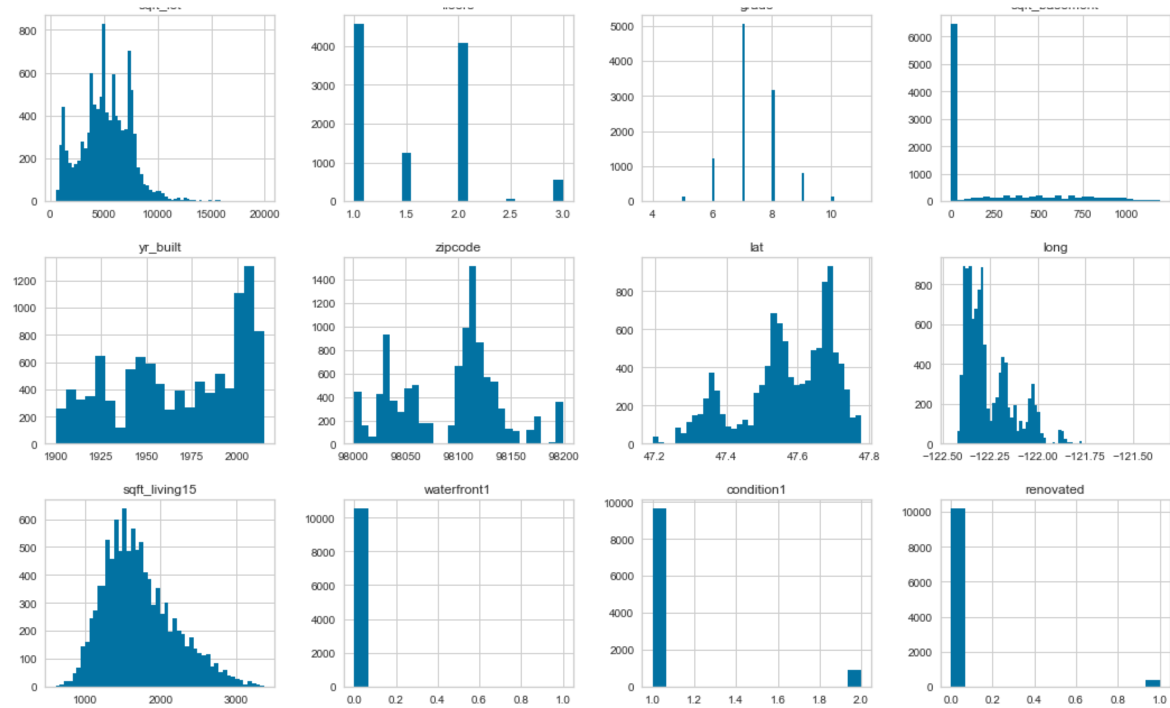
```
<AxesSubplot:title={'center':'Residuals for LinearRegression Model'}, xlab
el='Predicted Value', ylabel='Residuals'>
```

Third Model

In [127…

```
# check all variables shape of distribution by using histogram
fig = plt.figure(figsize = (18,15))
ax = fig.gca()
df.hist(ax = ax, bins='auto');
```



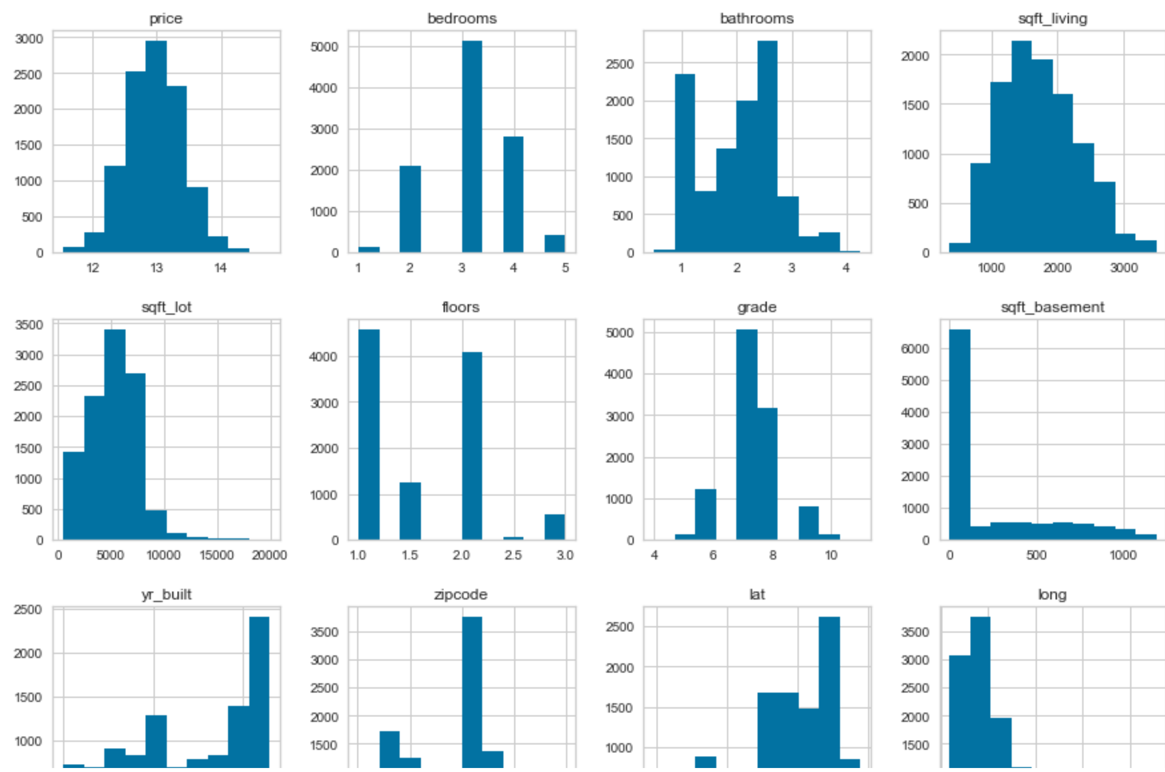


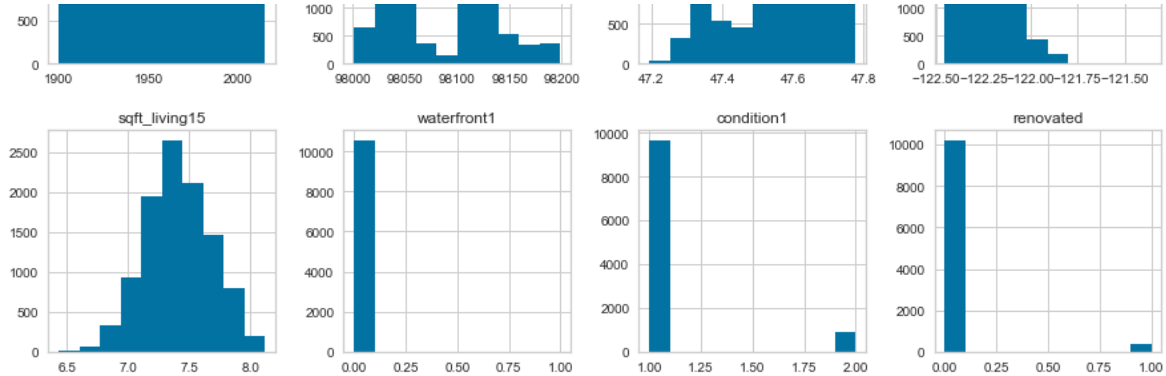
In [128…

```
# price, sqft_living, sqft_living15, are all continuous and almost normall
# sqft_lot, lat, long are all continuous but not normally distributed
# the not renovated percentage is way higher than the renovated, same appl
# bedrooms, bathrooms, floors, condition, grade,yr_built are all had ordir
```

In [129…

```
# Try log transform with every feature, but it only worked with the price
# now price and 'sqft_living15' distribution are way better
df4 = df.copy()
df4['price'] = np.log1p(df4['price'])
df4['sqft_living15'] = np.log1p(df4['sqft_living15'])
df4.hist(figsize  = [15, 15]);
```





In [130…

```
#one hot encode Zip code
```

In [131…

```
#df4["zipcode"] = pd.get_dummies(df4["zipcode"], columns="zipcode", drop_i
```

In [132…

```
#min max scale binary data
df4["waterfront1"]=(df4["waterfront1"] - min(df4["waterfront1"])) /(max(df
df4["renovated"] = (df4["renovated"] - min(df4["renovated"])) / (max(df4['
```

In [133…

```
X2 = df4.drop('price', axis=1)
y2 = df4['price']

X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y2, test_size=

# Instantiate a linear regression model
lr2 = LinearRegression()
# Fit our model on our normalized data
lr2.fit(X_train2, y_train2)
y_train_pred2 = lr2.predict(X_train2)
y_test_pred2 = lr2.predict(X_test2)
```

In [134…

```
# Evaluate
ut.evaluate_model(y_train2, y_test2, y_train_pred2, y_test_pred2)
```

```
Train R2: 0.722
Test R2: 0.733
---
Train MAE: 0.176
Test MAE: 0.176
---
Train RMSE: 0.227
Test RMSE: 0.228
```

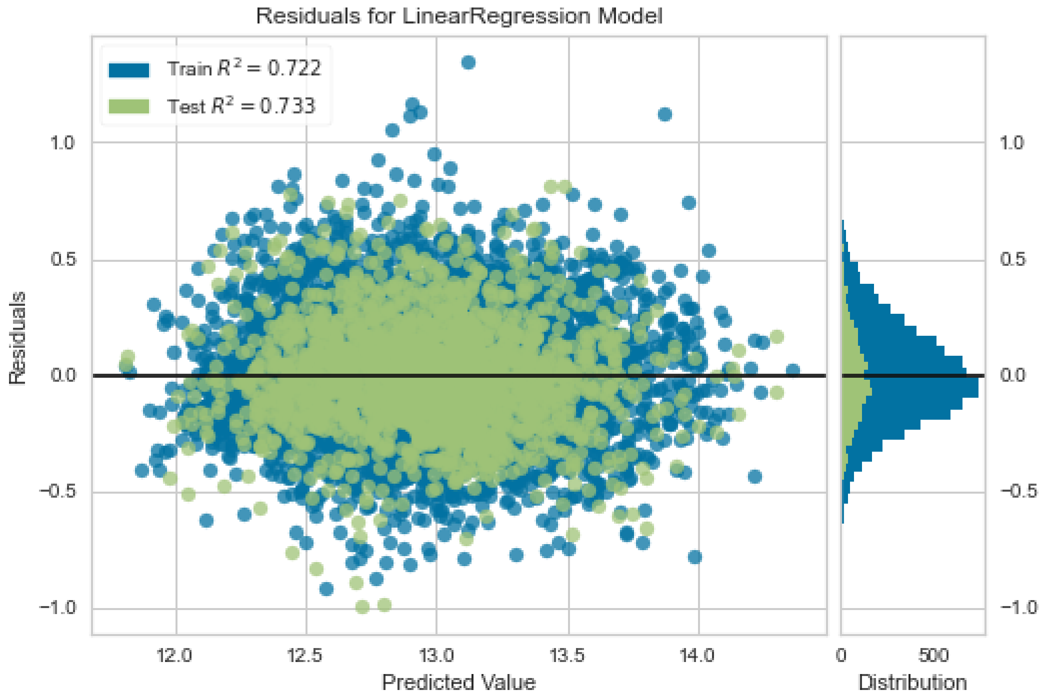
In [135…

```
# after log transformation, the model can predict 72 % variance in the pri
# and $0.22(RMSE)
```

In [136…

```
visualizer = ResidualsPlot(lr2)

visualizer.fit(X_train2, y_train2)  # Fit the training data to the visuali
visualizer.score(X_test2, y_test2)  # Evaluate the model on the test data
visualizer.show()
```

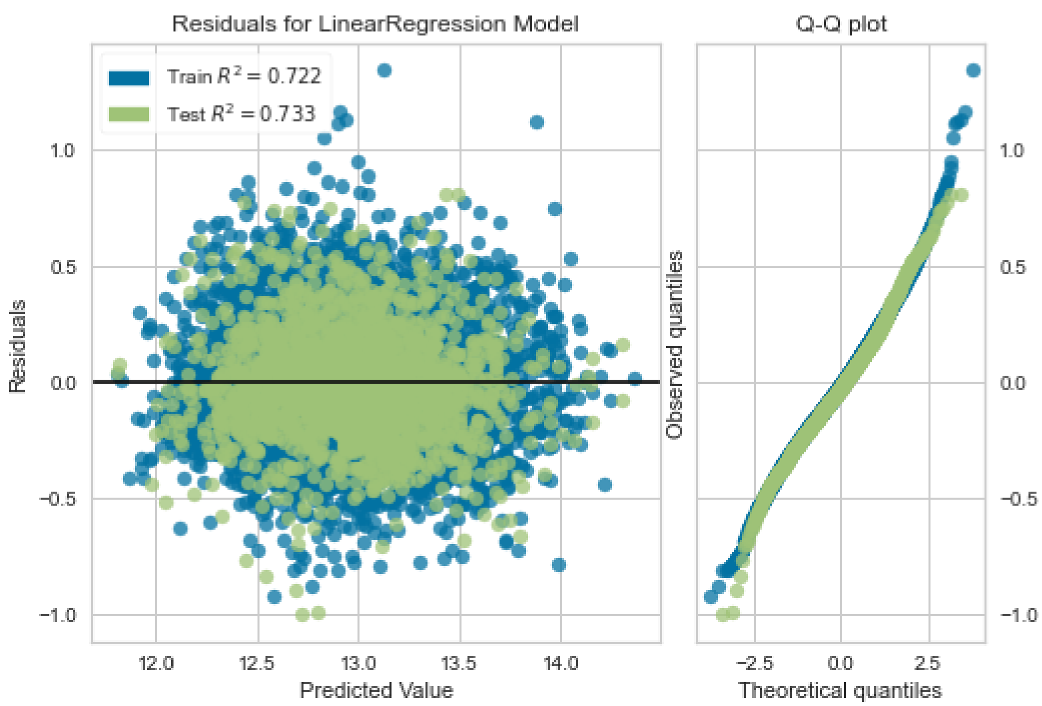
```
visualizer.show()
```



Out[136… <AxesSubplot:title={'center':'Residuals for LinearRegression Model'}, xlabel='Predicted Value', ylabel='Residuals'>

In [137…

```
visualizer = ResidualsPlot(lr2, hist=False, qqplot=True)
visualizer.fit(X_train2, y_train2)  # Fit the training data to the visuali
visualizer.score(X_test2, y_test2)  # Evaluate the model on the test data
visualizer.show()
```



Out[137… <AxesSubplot:title={'center':'Residuals for LinearRegression Model'}, xlabel='Predicted Value', ylabel='Residuals'>

In [138…

```
#The predicted values of the this model are equally scattered which achie
```

Final Model

In [139…

```
X6 = df4.drop('price', axis=1)
y6 = df4['price']

X_train6, X_test6, y_train6, y_test6 = train_test_split(X6, y6, test_size=
```

In [140…

```
from sklearn.preprocessing import PolynomialFeatures

# Instantiate PolynomialFeatures
poly = PolynomialFeatures(degree=2, interaction_only=False)

# Fit and transform X_train
X_poly_train = poly.fit_transform(X_train6)

# Instantiate and fit a linear regression model and normalize it to the po
reg_poly = LinearRegression(normalize=True).fit(X_poly_train, y_train6)

# Transform the test data into polynomial features
X_poly_test = poly.transform(X_test6)

# Get predicted values for transformed polynomial test data
y_pred = reg_poly.predict(X_poly_test)

# Transform the full data
X_poly = poly.transform(X_train6)

# Now, we want to see what the model predicts for the entire data
y_poly = reg_poly.predict(X_poly)
```

In [141…

```
ut.evaluate_model(y_train6, y_test6, y_poly, y_pred)
```

```
Train R2: 0.771
Test R2: 0.771
---
Train MAE: 0.157
Test MAE: 0.160
---
Train RMSE: 0.206
Test RMSE: 0.211
```

In [142…

```
#this model can predict 77 % variance in the price and approximately $0.16
#not all the data have perfect linaer realthion with the price
```

In [143…

```
#The predicted values of the this model are equally scattered which achiev
```

In [144…

```
#from yellowbrick.regressor import PredictionError
#visualizer = PredictionError(reg_poly, bestfit= True, is_fitted='auto')
#visualizer.fit(X_poly_train, y_train6)
#visualizer.score(X_poly_test, y_test6)
#visualizer.show()
```

In [145…

```
# "The prediction error visualizer plots the actual (measured) vs. expecte
#The dotted black line is the less than 45 degree line that indicates err
# https://buildmedia.readthedocs.org/media/pdf/yellowbrick/develop/yellowb
```

In [146…

```
# look at the coefficients with the names of each col
pd.DataFrame.from_dict(dict(zip(X6, reg_poly.coef_)), orient='index')[0].s
```

Out[146…

```
sqft_living15    6.674927e+02
long             1.947252e+02
bathrooms        4.611453e+01
sqft_living      1.726830e+01
zipcode          6.024535e-01
condition1       6.219759e-02
floors           1.296813e-02
bedrooms        -9.912057e-11
yr_built        -3.687360e-02
sqft_lot        -1.022184e-01
lat             -2.565942e-01
renovated       -3.868146e+00
grade           -4.417890e+00
sqft_basement   -1.254884e+01
waterfront1     -1.595438e+01
Name: 0, dtype: float64
```

In [147…

```
#check for multicollinearity

# 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

df8=df4.corr().abs().stack().reset_index().sort_values(0, ascending=False)

# zip the variable name columns (Which were only named level_0 and level_
df8['pairs'] = list(zip(df8.level_0, df8.level_1))

# set index to pairs
df8.set_index(['pairs'], inplace = True)

#d rop level columns
df8.drop(columns=['level_1', 'level_0'], inplace = True)

# rename correlation column as cc rather than 0
df8.columns = ['cc']

# drop duplicates.
df8.drop_duplicates(inplace=True)
df8.head()
```

Out[147…

	cc
pairs	
(price, price)	1.000000
(sqft_living, bathrooms)	0.674282

(sqft_living15, sqft_living)	0.665396
(zipcode, long)	0.643643
(bedrooms, sqft_living)	0.620960

In [148…

```
from sklearn.preprocessing import PolynomialFeatures

# Instantiate PolynomialFeatures
poly = PolynomialFeatures(degree=2, interaction_only=False)

# Fit and transform X_train
X_poly_train = poly.fit_transform(X_train6)

# Instantiate and fit a linear regression model to the polynomial transfo
reg_poly1 = LinearRegression().fit(X_poly_train, y_train6)

# Transform the test data into polynomial features
X_poly_test = poly.transform(X_test6)

# Get predicted values for transformed polynomial test data
y_pred = reg_poly.predict(X_poly_test)

# Transform the full data
X_poly = poly.transform(X_train6)

# Now, we want to see what the model predicts for the entire data
y_poly = reg_poly.predict(X_poly)
```

In [149…

```
# look at unscaled  coefficients with the names of each col
pd.DataFrame.from_dict(dict(zip(X6, reg_poly1.coef_)), orient='index')[0].
```

Out[149…

```
sqft_living15    667.492651
long             194.725151
bathrooms         46.114552
sqft_living       17.268315
zipcode            0.602454
floors             0.012968
bedrooms           0.002208
condition1         0.001108
yr_built          -0.036874
sqft_lot          -0.102218
lat               -0.256594
renovated         -0.769890
grade             -4.417896
sqft_basement    -12.548843
waterfront1      -15.954365
Name: 0, dtype: float64
```

Interpret:

Feature Importances

Each positive coefficient of the model give the anticipated change in the sale, so for a one-unit increase in the independent variable A positive coefficient show a positive correlation relationship. As the feauture increases, the price increases. While the negative coffient indicate that the price decreases as the independent variable decreases

- From the above features' coefficient, we can see that the most strongest feature is longitude in the first place, and the latitude in the third place, which means that the price is highly correlated with the location of the house
- The square footage of interior housing living space for the nearest 15 neighbors come in the second place, and the square footage of living space in the 4th place, which means the living space of a house and the nearest 15 neighbors are also highly correlated with the price

Interpretation of Regression Coefficients

Example A: No transformations

DV = Intercept + B1 * IV + Error "One unit increase in IV is associated with a (B1) unit increase in DV."

Example B: Outcome transformed

log(DV) = Intercept + B1 IV + Error "One unit increase in IV is associated with a (B1 100) percent increase in DV."

Example C: Exposure transformed

DV = Intercept + B1 * log(IV) + Error "One percent increase in IV is associated with a (B1 / 100) unit increase in DV."

Example D: Outcome transformed and exposure transformed

log(DV) = Intercept + B1 * log(IV) + Error "One percent increase in IV is associated with a (B1) percent increase in DV."

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 [150…

```
# Visualizing Longitude to Latitude to check how the price vary by locatio
plt.figure(figsize= (30, 15))
plt.scatter(x=df4['long'], y=df['lat'], c=df['price'], cmap='hsv', marker=
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.savefig("images/HousepricebasedonLocation.png")
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 [151…

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

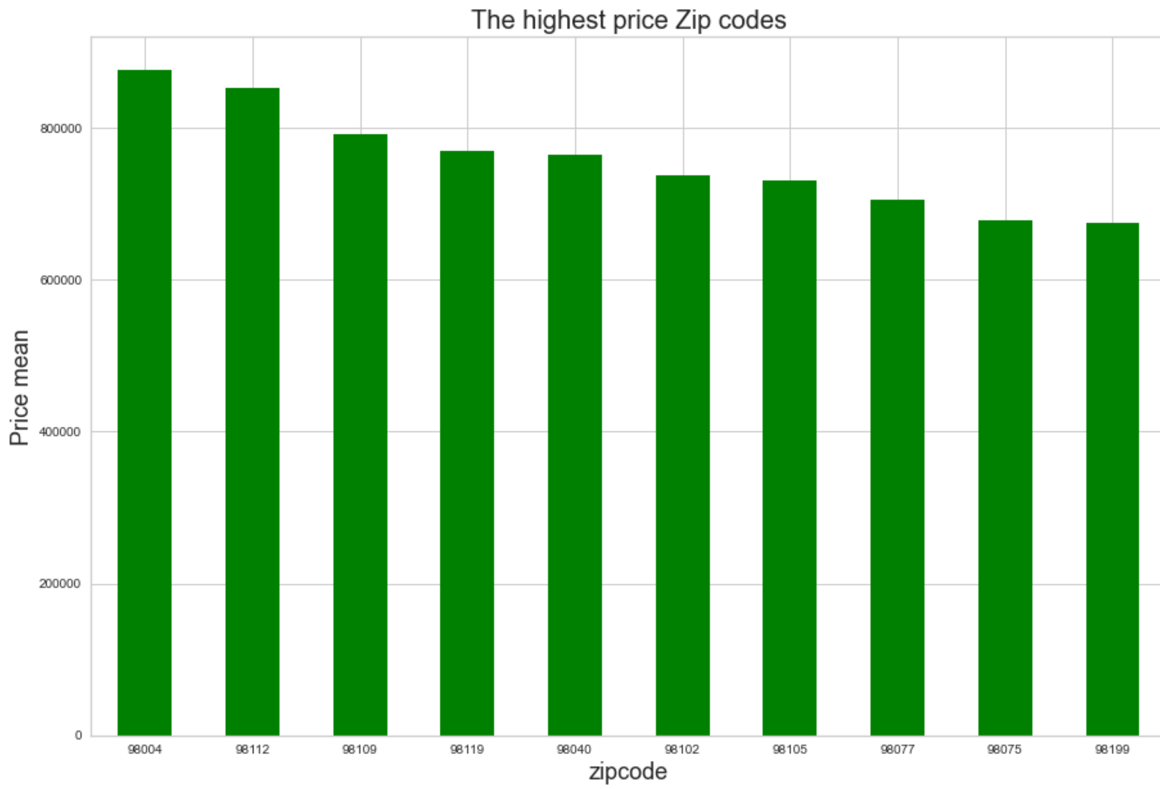
Out[151…

```
zipcode
98004    876144.950000
98112    852622.015385
98109    790953.826531
98119    770200.748503
98040    765300.000000
98102    738225.533333
98105    730829.549451
98077    705000.000000
98075    678133.288462
98199    675472.112450
Name: price, dtype: float64
```

In [152…

```
# 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.savefig("images/The highest price zipcode.png")
plt.show()
```



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

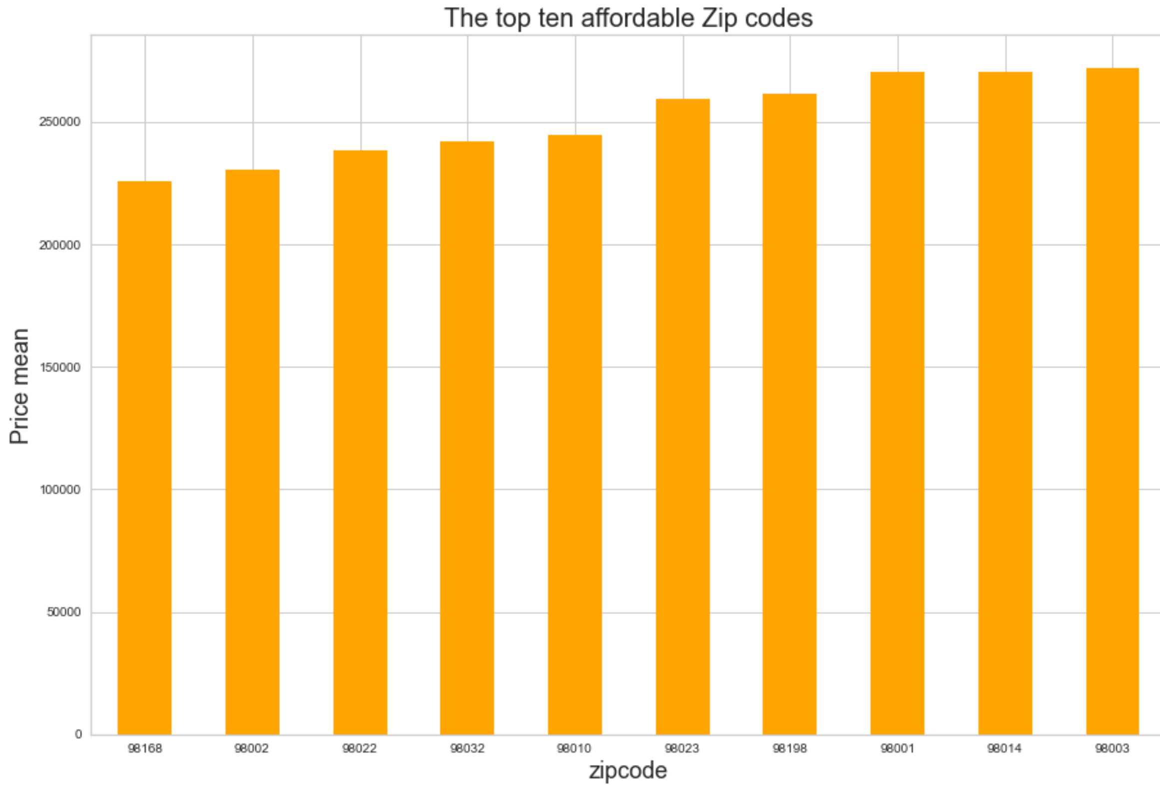
After looking up the corresponding cities to each zip code, the most affordable cities in terms of the price mean are Tukwila, Auburn, Numclaw, Wabash, Birch, Krain, Cumberland, Bayne, Osceola, Maywood, Upper Mill, Bayne Junction, Boise, Veazie, Naco, Stampede,Kent, Lakeland North, Black Diamond, Franklin,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

In [153…
```
# group by zipcode and get the mean of prices in a zipcode
top_ten= df.groupby('zipcode')['price'].mean().sort_values(ascending=True)
top_ten.head(20)
```

Out[153…
```
zipcode
98168    226001.785714
98002    230308.966942
98022    238203.108696
98032    241882.044444
98010    244655.000000
98023    259351.428571
98198    261387.270270
98001    270265.829630
98014    270400.000000
98003    272020.265957
Name: price, dtype: float64
```

In [154…
```
# 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.savefig("images/The top ten affordable zipcode.png")
plt.show()
```

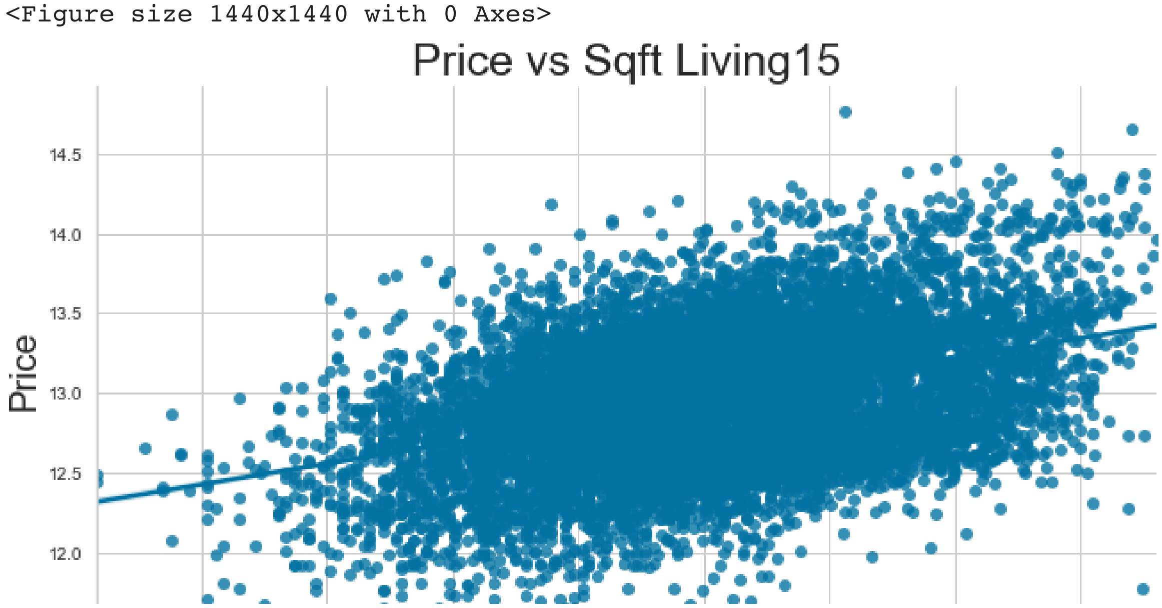


4. Which features are important to predict the price of the house?

In [155…

```
plt.figure(figsize = (20,20));
sqf=sns.lmplot(x="sqft_living15", y="price",aspect=1.8,data=df4)
plt.title("Price vs Sqft Living15",fontsize=25)
sqf.set_xlabels("Sqft Living15",fontsize=20)
sqf.set_ylabels("Price",fontsize=20)
plt.show();
```

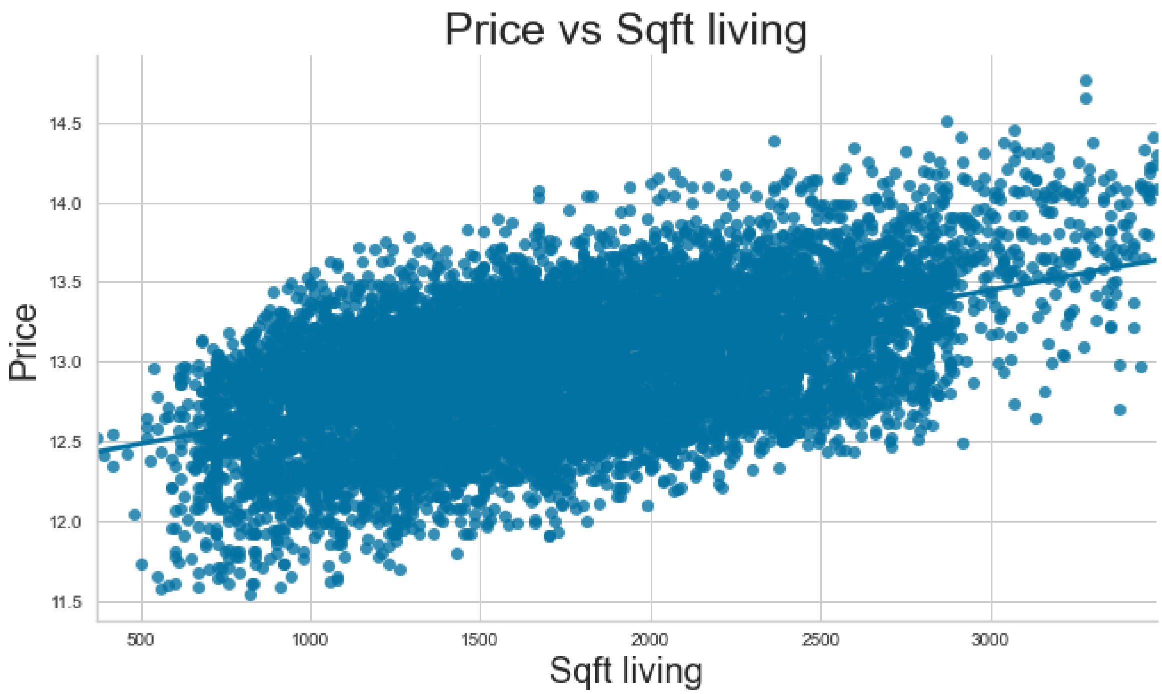
```
<Figure size 1440x1440 with 0 Axes>
```



In [156…

```
plt.figure(figsize = (20,20));
sqf=sns.lmplot(x="sqft_living", y="price",aspect=1.8,data=df4)
plt.title("Price vs Sqft living",fontsize=25)
sqf.set_xlabels("Sqft living",fontsize=20)
sqf.set_ylabels("Price",fontsize=20)
plt.show();
```

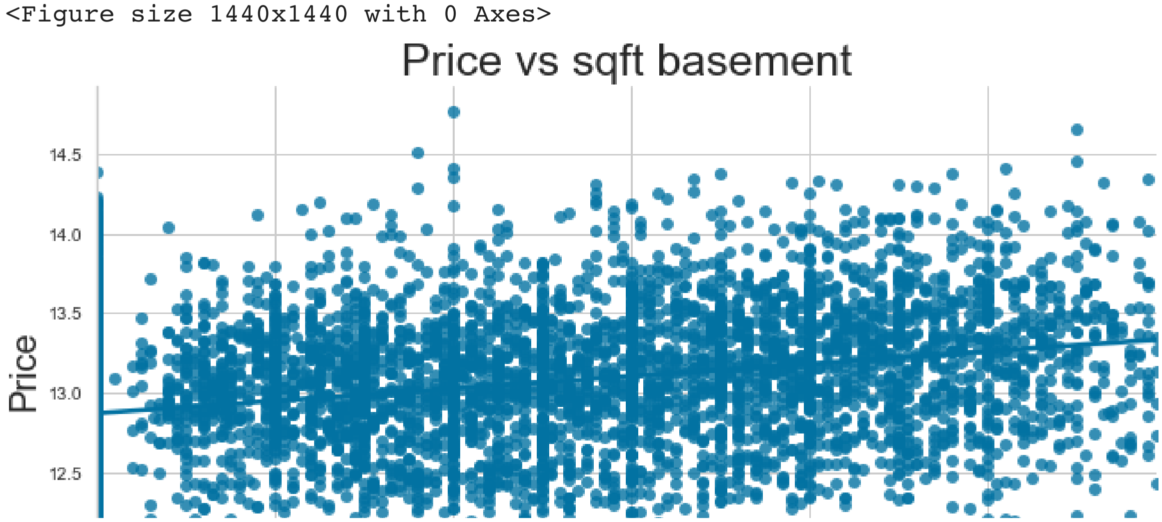
<Figure size 1440x1440 with 0 Axes>

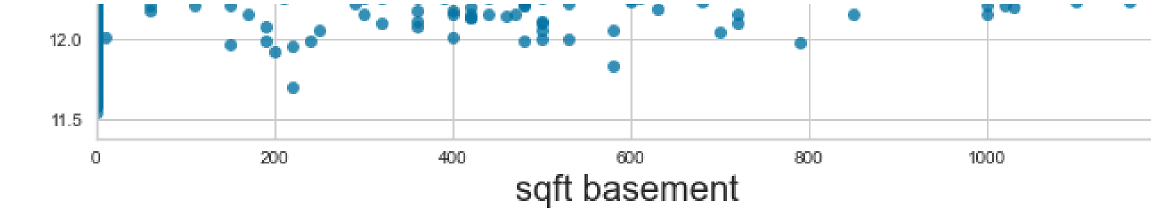


In [157…

```
plt.figure(figsize = (20,20));
sqf=sns.lmplot(x="sqft_basement", y="price",aspect=1.8,data=df4)
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 [158…

```
plt.figure(figsize = (20,20));
sqf=sns.lmplot(x="bathrooms", y="price",aspect=1.8,data=df4)
plt.title("Price vs Bathrooms",fontsize=25)
sqf.set_xlabels("Bathrooms",fontsize=20)
sqf.set_ylabels("Price",fontsize=20)
plt.show();
```

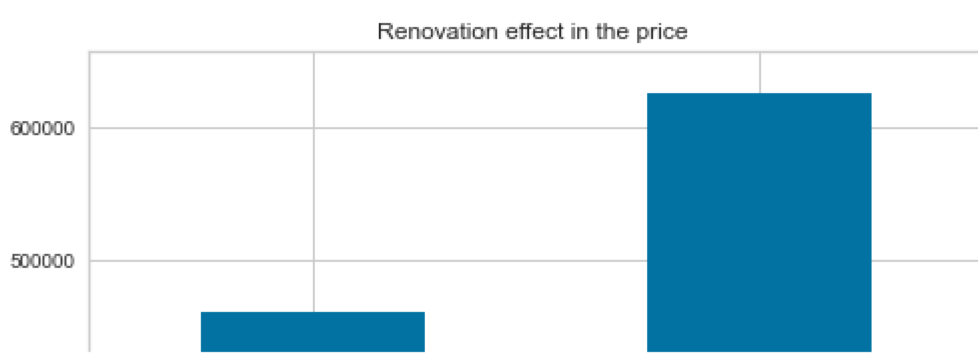
<Figure size 1440x1440 with 0 Axes>

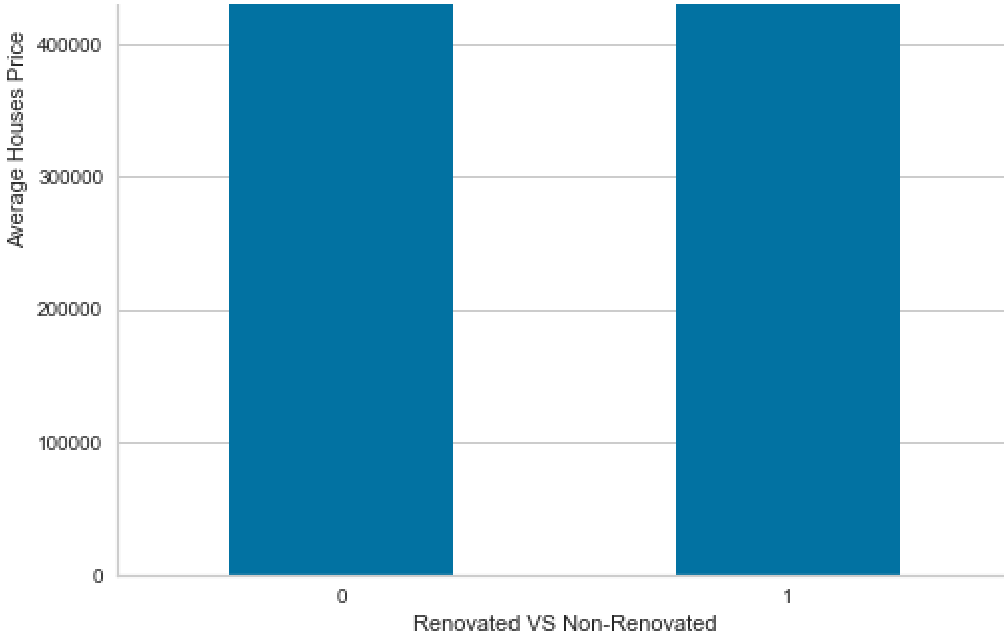
In [159…

```
# 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[159…

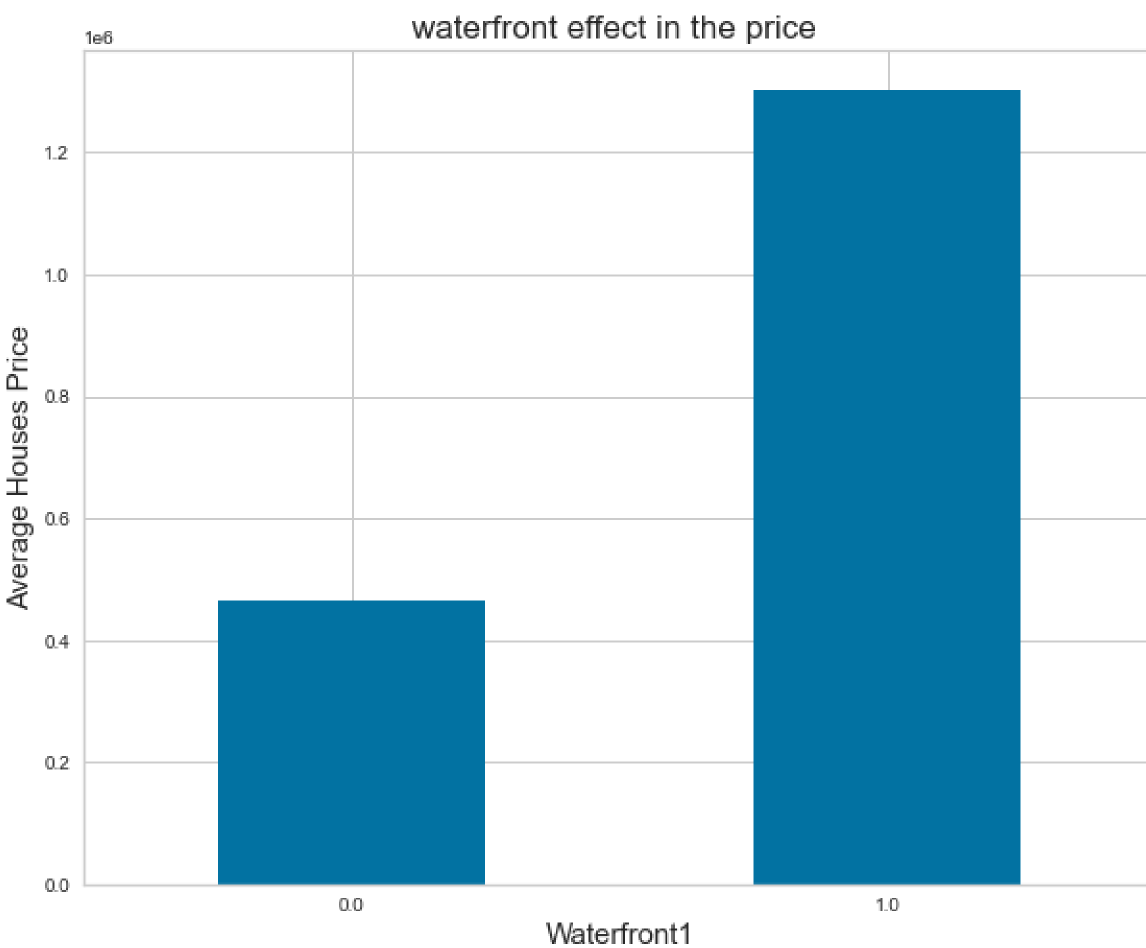
```
(array([0, 1]), [Text(0, 0, '0'), Text(1, 0, '1')])
```





In [160…

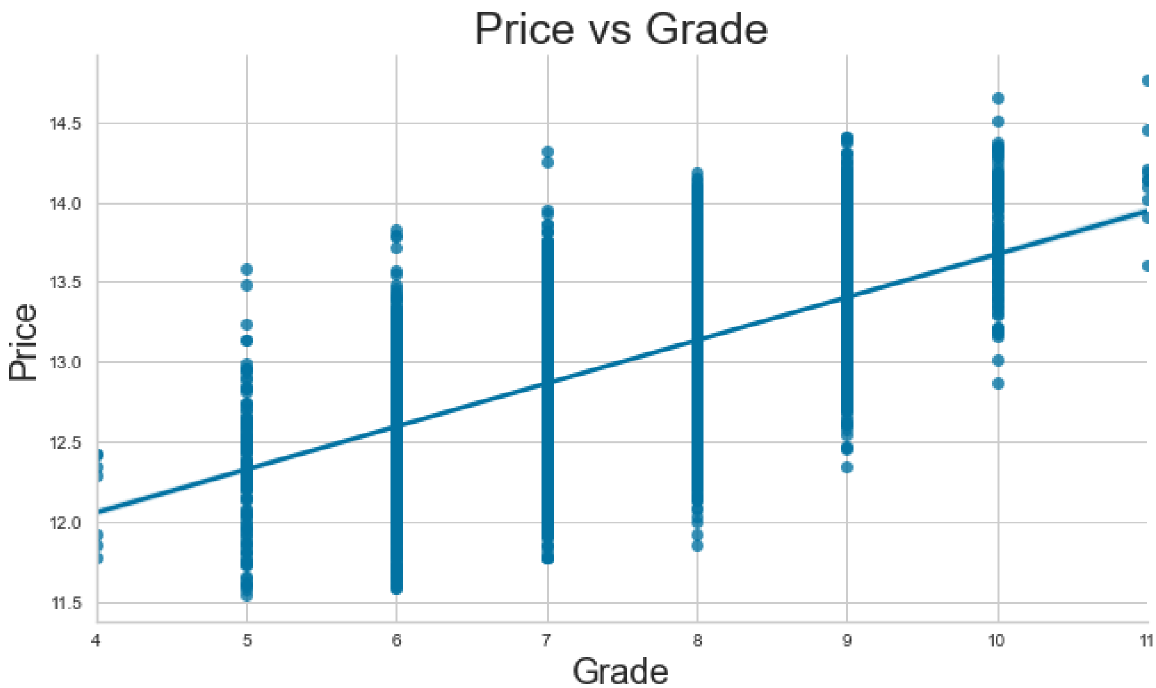
```
# plotting houses to the mean of price
df.groupby("waterfront1")["price"].mean().plot(kind="bar",figsize=(10,8));
plt.title("waterfront effect in the price ", fontsize=17)
plt.ylabel("Average Houses Price",fontsize=15)
plt.xlabel("Waterfront1",fontsize=15)
plt.xticks(rotation=0)
plt.show()
#the houses with waterfront selling price are higher than one without wate
```



In [161…

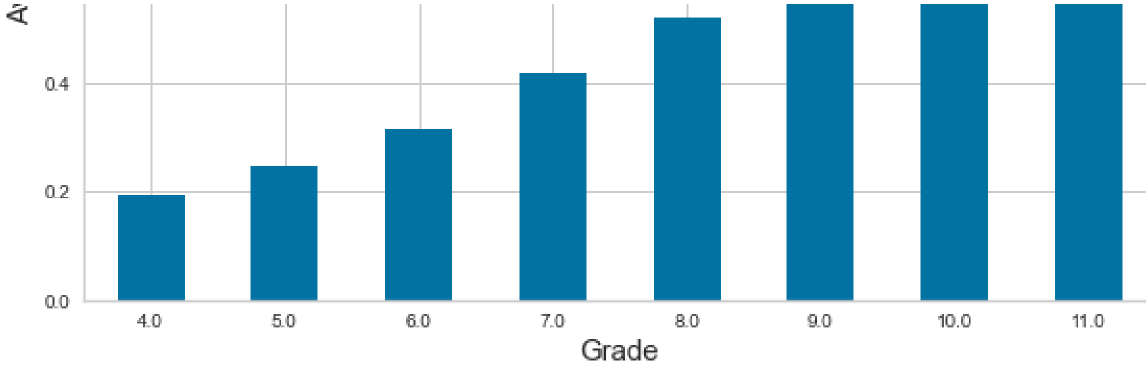
```
plt.figure(figsize = (20,20));
sqf=sns.lmplot(x="grade", y="price",aspect=1.8,data=df4)
plt.title("Price vs Grade",fontsize=25)
sqf.set_xlabels("Grade",fontsize=20)
sqf.set_ylabels("Price",fontsize=20)
plt.show();
```

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



In [162…

```
# 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)
plt.show()
#the houses with waterfront selling price are higher than one without wate
```



Conclusions

- I organized my notebook by using OSMEN data science method to deal with the king county dataset.
- I cleaned the data, removed the null values, limit each feature of the data to get rid of the outlier as much as possible, and checked for Multicollinearity.
- For the baseline model, I train test split the data using scikit-learn and scaled it using the standardized scaler, and got a R2 score of 0.67, for the Normality assumption the residuals were heteroscedastic.
- For the second model, I train test split the data using scikit-learn, Polynomial Regression, MinMaxScaler, and got a R2 score of 0.76, for Normality assumption the residuals were still heteroscedastic, I also used step forward feature selection to check if dropped any feature will make the R2 better, but it didn't.
- For the third model, I logged transform price and 'sqft_living15, one hot encode zip code, min-max scale binary data (waterfront and renovated). Train test split the data using sci-kit-learn, and got a R2 score of 0.73. For the normality assumption, the residuals were homoscedastic and the QQ plot looks good.
- For the Final mode, I copied the same data from the third model before test split, then train test split the data using scikit-learn, use polynomial regression, scaled it to be able to interpret the coefficients, and got a R2 score of 0.76, for the normality assumption the residuals were homoscedastic and the QQ plot looks good, checked for multicollinearity. after that unscaled it to be able to see how each unit of each features impacts the price.
- 15 features were included in the final model to get the best prediction, The following findings are from the features with the highest coefficients:
- The price of the house is highly affected by its location.
- Houses with larger living space, bigger basement, and more bathrooms have higher predicted price.
- The renovated houses selling price is higher than non-renovated one

- The houses with waterfront have higher selling prices than the ones without one.
- Each increase of the grade will increase the price, with grade 11 in the top

Limitaion

- The size of the dataset, a lot of features don't have a linear relationship with the target. Maybe a different non-linear model would work better.

Future work

Use APIs to get King county school district data and link it with the zip codes.

In [ ]:

```

```

In [ ]:

```

```