Regression and Time Series Project 1 (House Price Prediction) version 2

December 9, 2021

1 Import libraries

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression,Lasso,Ridge,LassoCV
```

2 Load and Review data

[2]:
```
House_data = pd.read_csv("../data/kc_house_data.csv",engine='python')
```

[3]:
```
House_data.shape
```

[3]:
```
(21613, 21)
```

[4]:
```
House_data.head(5)
```

[4]:
```
           id             date     price  bedrooms  bathrooms  sqft_living  \
0  7129300520  20141013T000000  221900.0         3       1.00         1180
1  6414100192  20141209T000000  538000.0         3       2.25         2570
2  5631500400  20150225T000000  180000.0         2       1.00          770
3  2487200875  20141209T000000  604000.0         4       3.00         1960
4  1954400510  20150218T000000  510000.0         3       2.00         1680

   sqft_lot  floors  waterfront  view  …  grade  sqft_above  sqft_basement  \
0      5650     1.0           0     0  …      7        1180              0
1      7242     2.0           0     0  …      7        2170            400
2     10000     1.0           0     0  …      6         770              0
3      5000     1.0           0     0  …      7        1050            910
4      8080     1.0           0     0  …      8        1680              0
```

```
   yr_built  yr_renovated  zipcode      lat     long  sqft_living15  \
0      1955             0    98178  47.5112 -122.257           1340
1      1951          1991    98125  47.7210 -122.319           1690
2      1933             0    98028  47.7379 -122.233           2720
3      1965             0    98136  47.5208 -122.393           1360
4      1987             0    98074  47.6168 -122.045           1800

   sqft_lot15
0        5650
1        7639
2        8062
3        5000
4        7503

[5 rows x 21 columns]
```

[5]:
```
House_data.info()
```

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

[6]:
```
House_data.describe()
```

[6]:
```
                 id         price      bedrooms     bathrooms   sqft_living  \
count  2.161300e+04  2.161300e+04  21613.000000  21613.000000  21613.000000
mean   4.580302e+09  5.400881e+05      3.370842      2.114757   2079.899736
std    2.876566e+09  3.671272e+05      0.930062      0.770163    918.440897
min    1.000102e+06  7.500000e+04      0.000000      0.000000    290.000000
25%    2.123049e+09  3.219500e+05      3.000000      1.750000   1427.000000
50%    3.904930e+09  4.500000e+05      3.000000      2.250000   1910.000000
75%    7.308900e+09  6.450000e+05      4.000000      2.500000   2550.000000
max    9.900000e+09  7.700000e+06     33.000000      8.000000  13540.000000

           sqft_lot        floors    waterfront          view     condition  \
count  2.161300e+04  21613.000000  21613.000000  21613.000000  21613.000000
mean   1.510697e+04      1.494309      0.007542      0.234303      3.409430
std    4.142051e+04      0.539989      0.086517      0.766318      0.650743
min    5.200000e+02      1.000000      0.000000      0.000000      1.000000
25%    5.040000e+03      1.000000      0.000000      0.000000      3.000000
50%    7.618000e+03      1.500000      0.000000      0.000000      3.000000
75%    1.068800e+04      2.000000      0.000000      0.000000      4.000000
max    1.651359e+06      3.500000      1.000000      4.000000      5.000000

              grade    sqft_above  sqft_basement      yr_built  yr_renovated  \
count  21613.000000  21613.000000   21613.000000  21613.000000  21613.000000
mean       7.656873   1788.390691     291.509045   1971.005136     84.402258
std        1.175459    828.090978     442.575043     29.373411    401.679240
min        1.000000    290.000000       0.000000   1900.000000      0.000000
25%        7.000000   1190.000000       0.000000   1951.000000      0.000000
50%        7.000000   1560.000000       0.000000   1975.000000      0.000000
75%        8.000000   2210.000000     560.000000   1997.000000      0.000000
max       13.000000   9410.000000    4820.000000   2015.000000   2015.000000

            zipcode           lat          long  sqft_living15     sqft_lot15
count  21613.000000  21613.000000  21613.000000   21613.000000   21613.000000
mean   98077.939805     47.560053   -122.213896    1986.552492   12768.455652
std       53.505026      0.138564      0.140828     685.391304   27304.179631
min    98001.000000     47.155900   -122.519000     399.000000     651.000000
25%    98033.000000     47.471000   -122.328000    1490.000000    5100.000000
50%    98065.000000     47.571800   -122.230000    1840.000000    7620.000000
75%    98118.000000     47.678000   -122.125000    2360.000000   10083.000000
max    98199.000000     47.777600   -121.315000    6210.000000  871200.000000
```

[7]:
```
House_data.bedrooms.value_counts()
```

[7]:
```
3     9824
4     6882
2     2760
5     1601
6      272
```

```
1      199
7       38
8       13
0       13
9        6
10       3
11       1
33       1
Name: bedrooms, dtype: int64
```

[8]:
```
House_data.floors.value_counts()
```

[8]:
```
1.0    10680
2.0     8241
1.5     1910
3.0      613
2.5      161
3.5        8
Name: floors, dtype: int64
```

[9]:
```
House_data.view.value_counts()
```

[9]:
```
0    19489
2      963
3      510
1      332
4      319
Name: view, dtype: int64
```

[10]:
```
House_data.waterfront.value_counts()
```

[10]:
```
0    21450
1      163
Name: waterfront, dtype: int64
```

[11]:
```
House_data.condition.value_counts()
```

[11]:
```
3    14031
4     5679
5     1701
2      172
1       30
Name: condition, dtype: int64
```

[12]:
```
House_data.grade.value_counts()
```

[12]:
```
7     8981
8     6068
```

```
9      2615
6      2038
10     1134
11      399
5       242
12       90
4        29
13       13
3         3
1         1
Name: grade, dtype: int64
```

3 Observations

1. The maximum number of bedrooms in a house are 33. So might wanted to look at that record and check if it is an outlier.
2. one floor houses are the most common type of houses sold
3. Very few houses have view to a waterfront and these houses might be costly
4. House condition is rated from 1 to 5 and the most common rating is 3
5. House grade is rated from 1 to 13 and the most common rating is 7

4 Converting day hours to date time object

[13]:
```
House_data['date'] = pd.to_datetime(House_data['date']).dt.to_period('m')

House_data.head()
```

[13]:
```
           id     date     price  bedrooms  bathrooms  sqft_living  sqft_lot  \
0  7129300520  2014-10  221900.0         3       1.00         1180      5650
1  6414100192  2014-12  538000.0         3       2.25         2570      7242
2  5631500400  2015-02  180000.0         2       1.00          770     10000
3  2487200875  2014-12  604000.0         4       3.00         1960      5000
4  1954400510  2015-02  510000.0         3       2.00         1680      8080

   floors  waterfront  view  …  grade  sqft_above  sqft_basement  yr_built  \
0     1.0           0     0  …      7        1180              0      1955
1     2.0           0     0  …      7        2170            400      1951
2     1.0           0     0  …      6         770              0      1933
3     1.0           0     0  …      7        1050            910      1965
4     1.0           0     0  …      8        1680              0      1987

   yr_renovated  zipcode      lat     long  sqft_living15  sqft_lot15
0             0    98178  47.5112 -122.257           1340        5650
1          1991    98125  47.7210 -122.319           1690        7639
2             0    98028  47.7379 -122.233           2720        8062
3             0    98136  47.5208 -122.393           1360        5000
```

```
4              0    98074  47.6168 -122.045           1800         7503

[5 rows x 21 columns]
```

5 To checlk if there are any null values or missing data

[14]:
```
House_data.isnull().any()
```

[14]:
```
id               False
date             False
price            False
bedrooms         False
bathrooms        False
sqft_living      False
sqft_lot         False
floors           False
waterfront       False
view             False
condition        False
grade            False
sqft_above       False
sqft_basement    False
yr_built         False
yr_renovated     False
zipcode          False
lat              False
long             False
sqft_living15    False
sqft_lot15       False
dtype: bool
```

There are no null values or missing values in the data. No data cleansing is required.

6 To check if there are duplicate entries

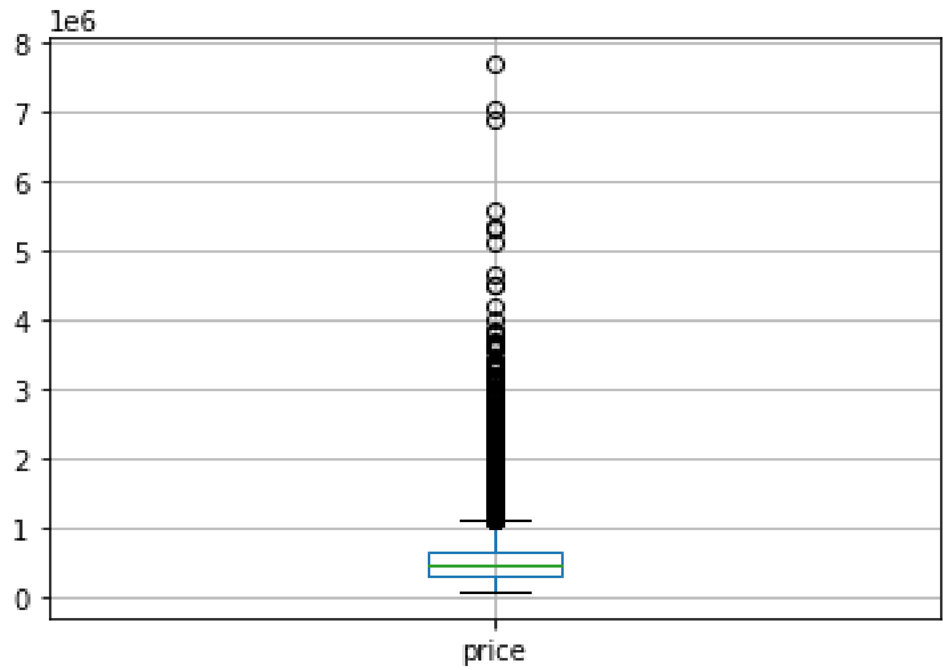
[15]:
```
House_data.duplicated().sum()
```

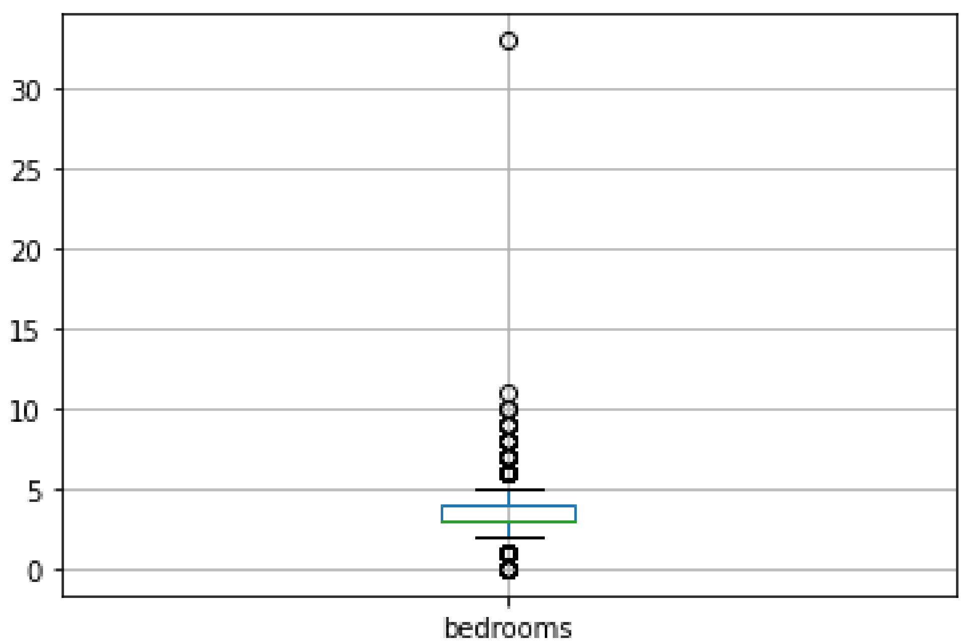
[15]:
```
0
```

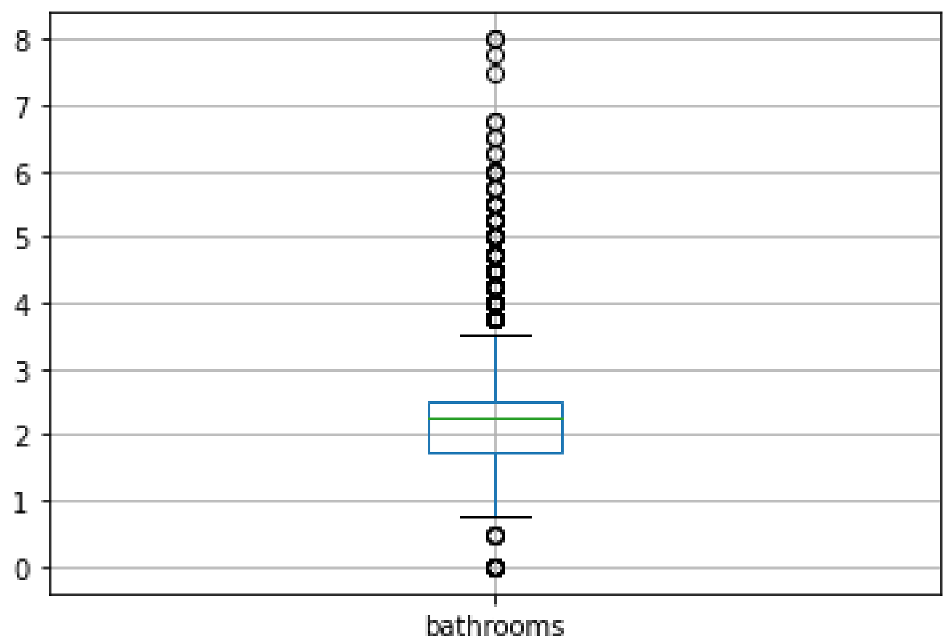
7 Univariate analysis

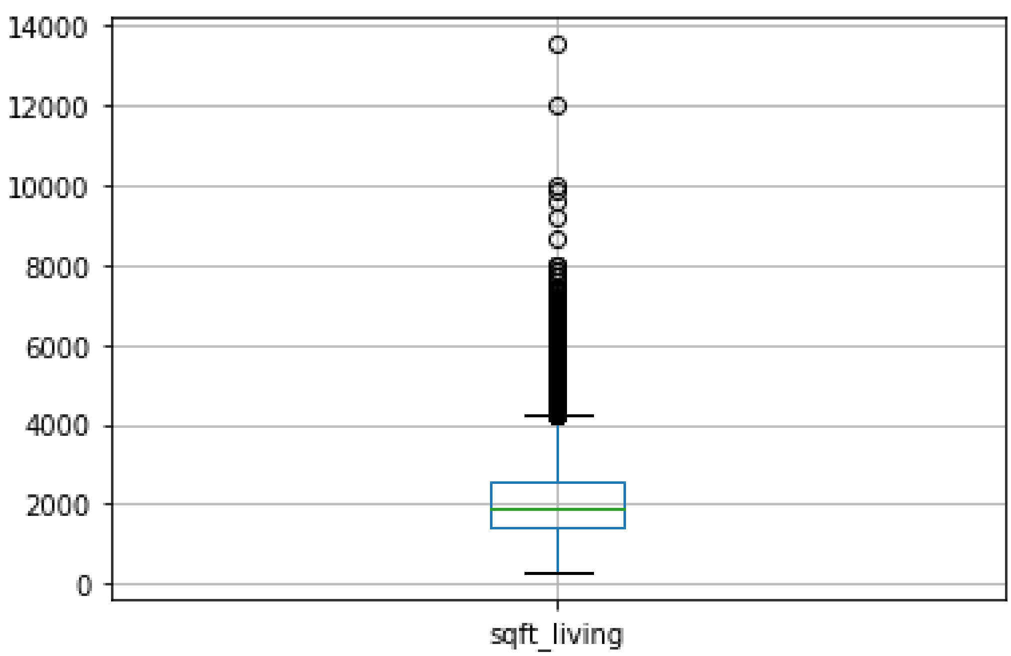
8 Plotted the box plot of different columns to check if there are any outliers

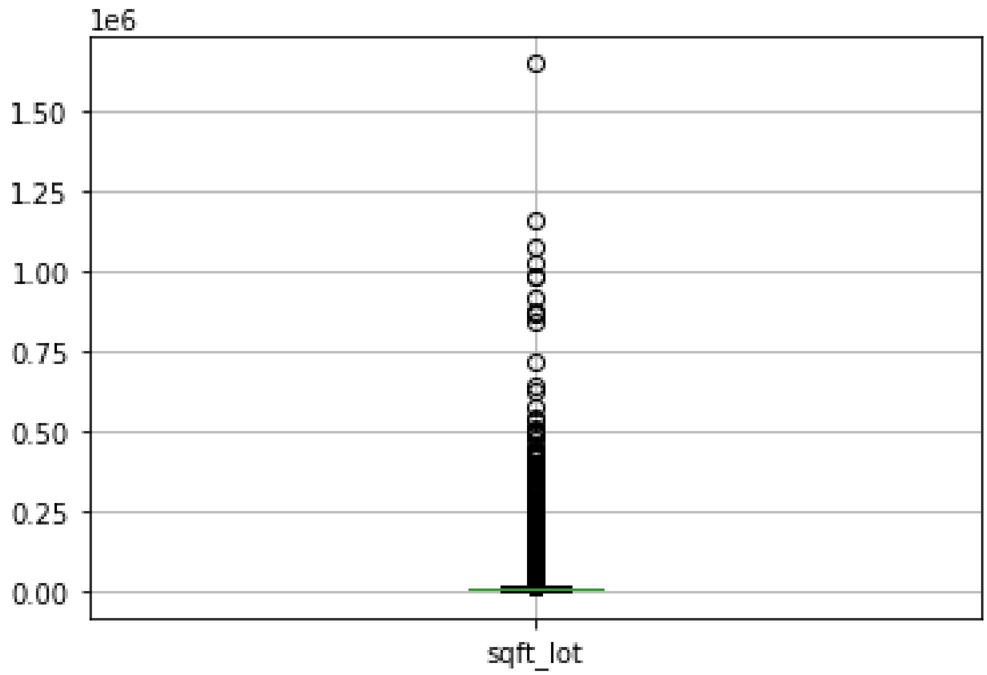
[16]:
```
for i in House_data.iloc[:, 2:].columns:
    House_data.iloc[:, 1:].boxplot(column=i)
    plt.show()
```

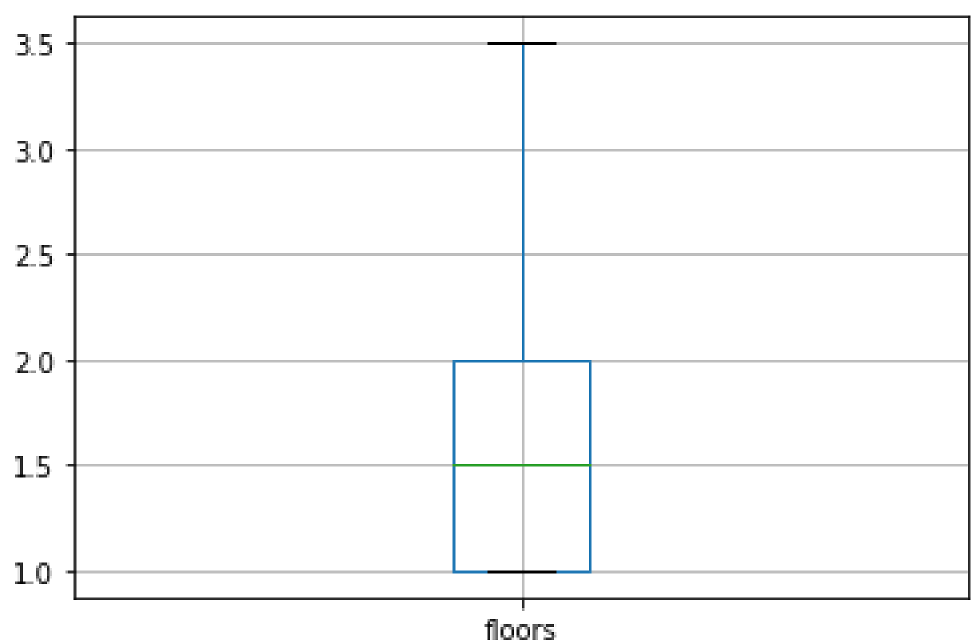


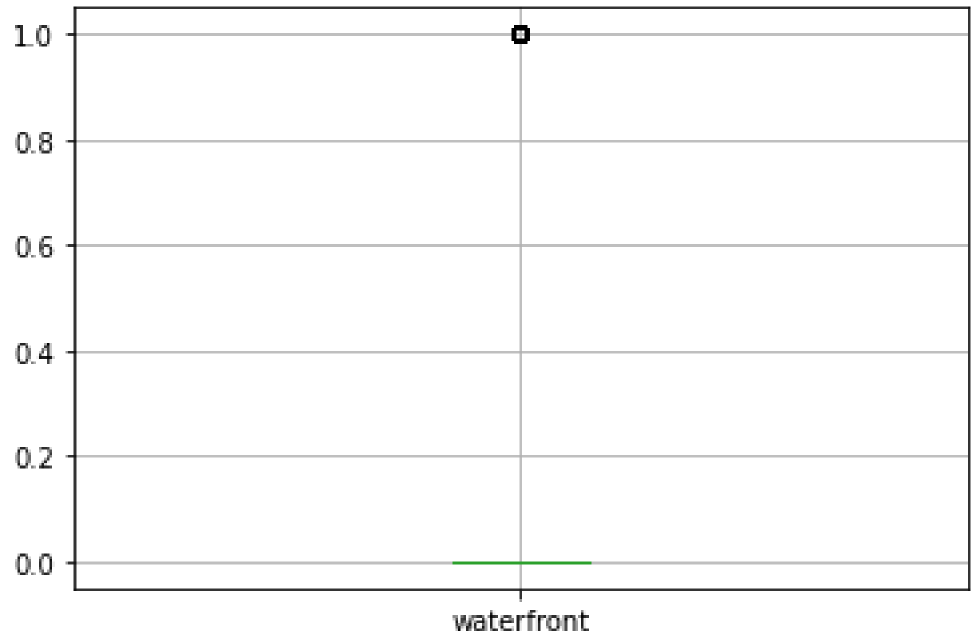


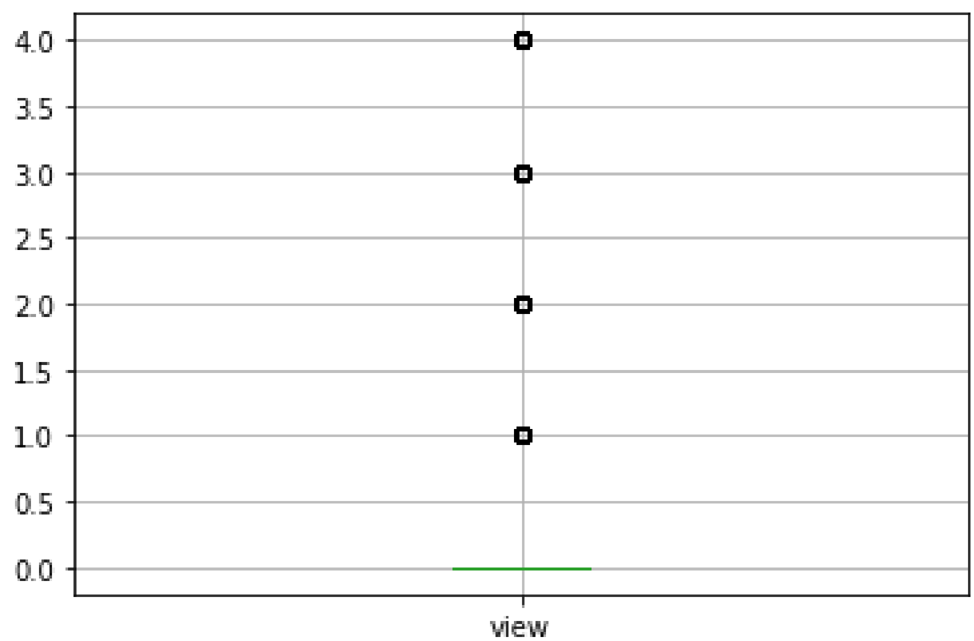

8
7
6
5
4
3
2
1
0
bathrooms

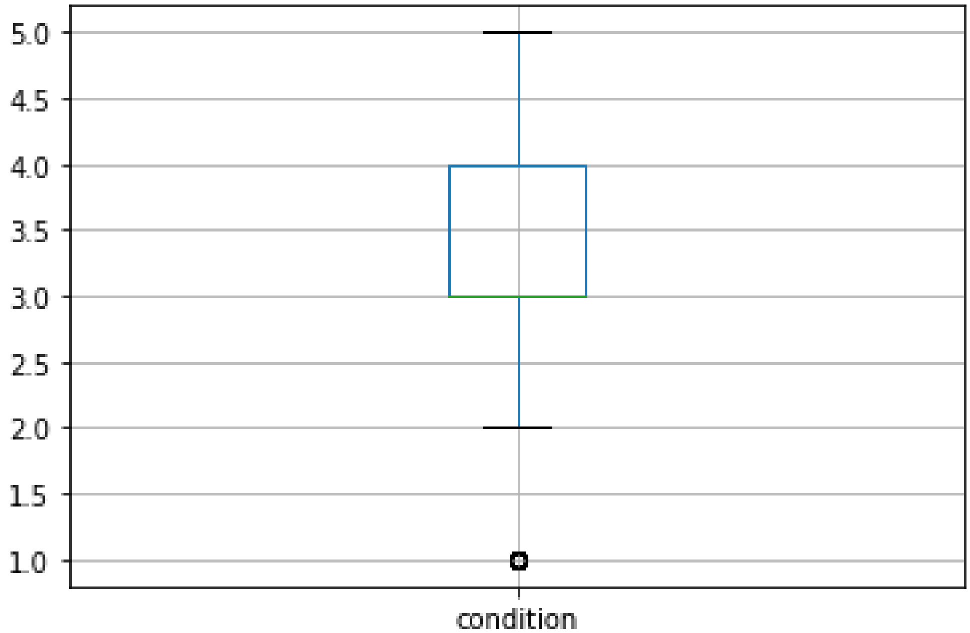

14000
12000
10000
8000
6000
4000
2000
0
sqft_living

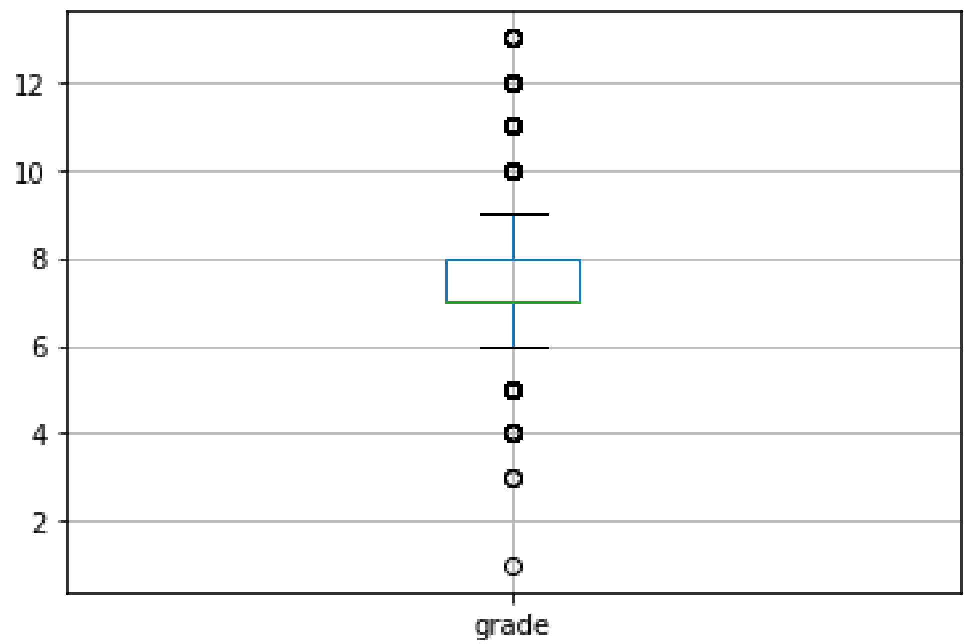

1e6
1.50
1.25
1.00
0.75
0.50
0.25
0.00
sqft_lot

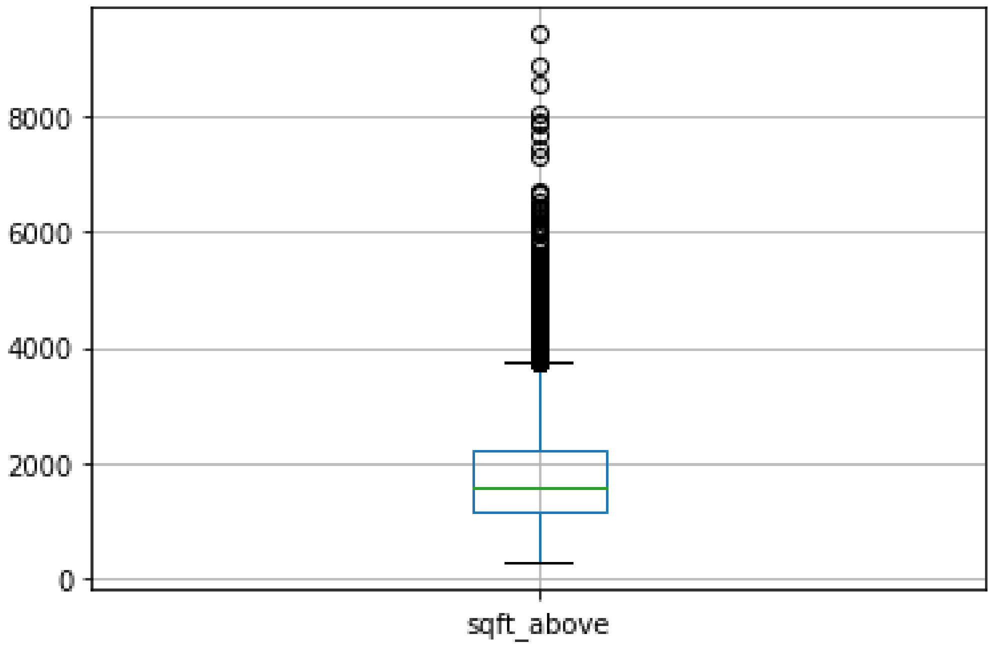

3.5
3.0
2.5
2.0
1.5
1.0
floors

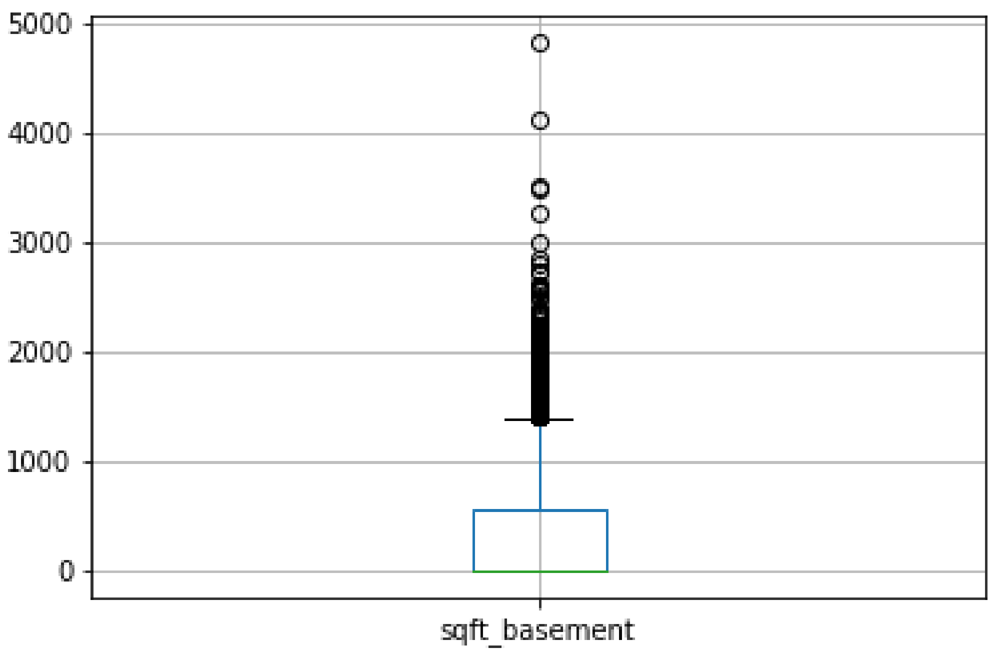

1.0
0.8
0.6
0.4
0.2
0.0
waterfront

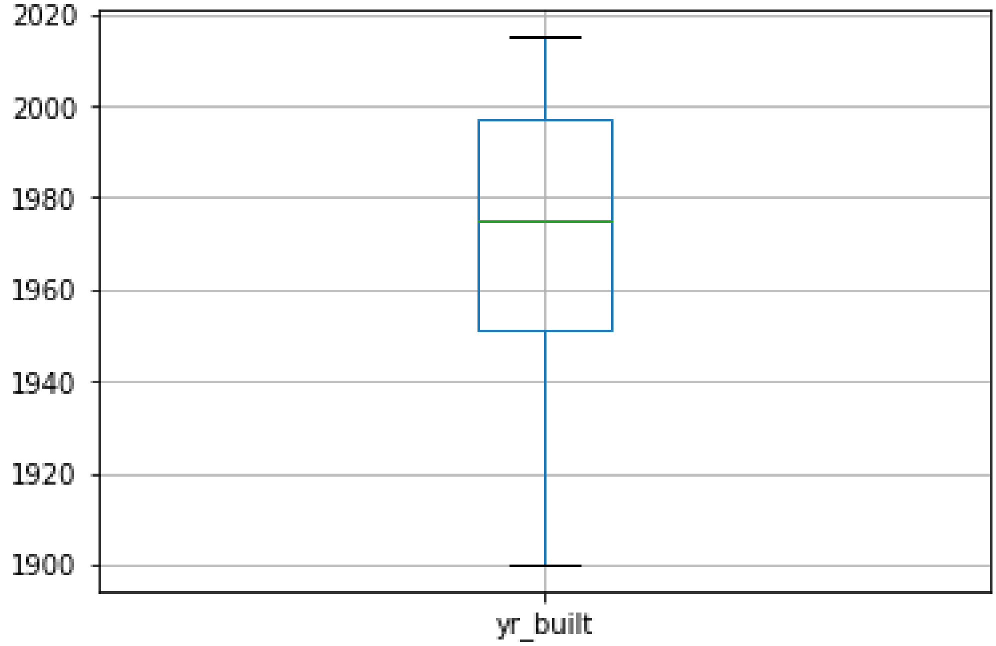

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
view

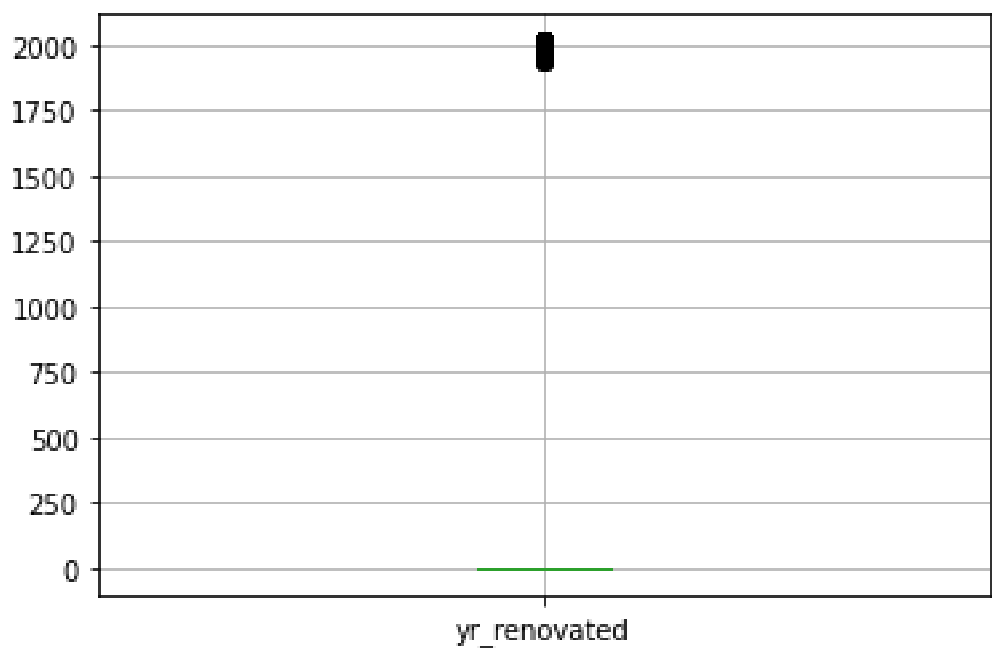

5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
condition

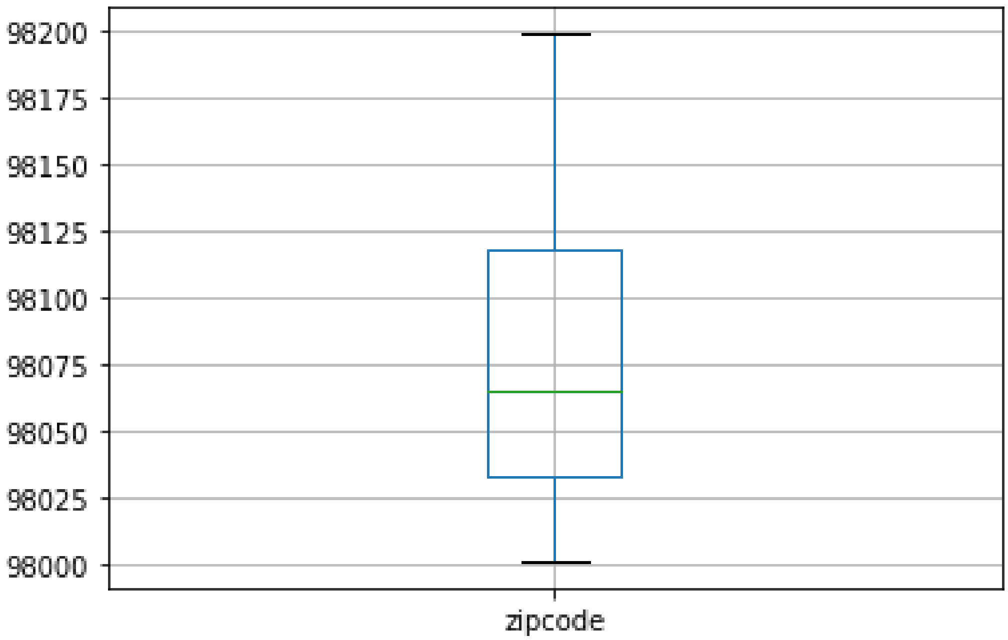

12
10
8
6
4
2
grade

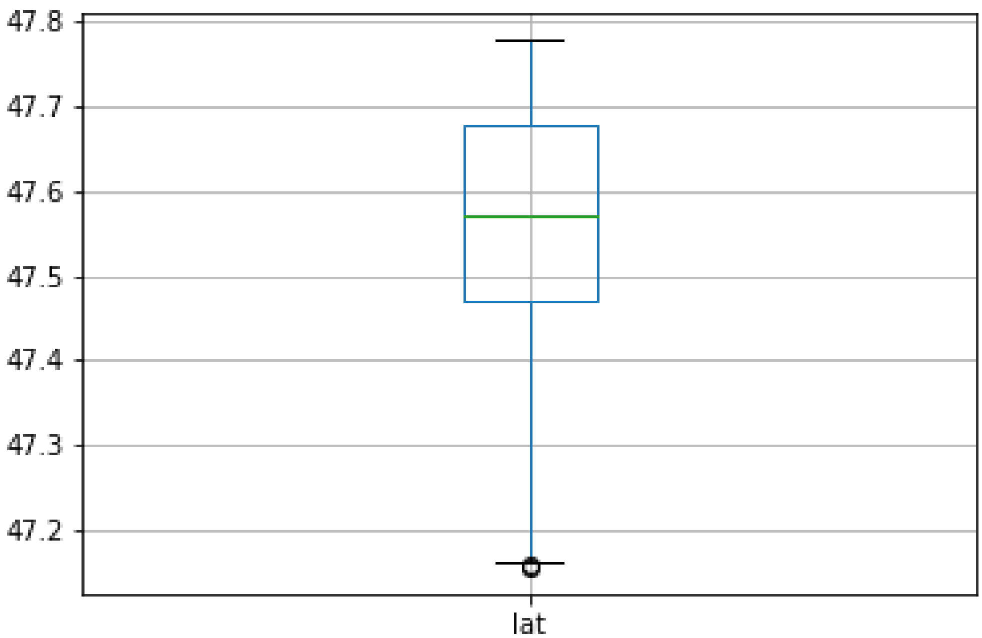

8000
6000
4000
2000
0
sqft_above

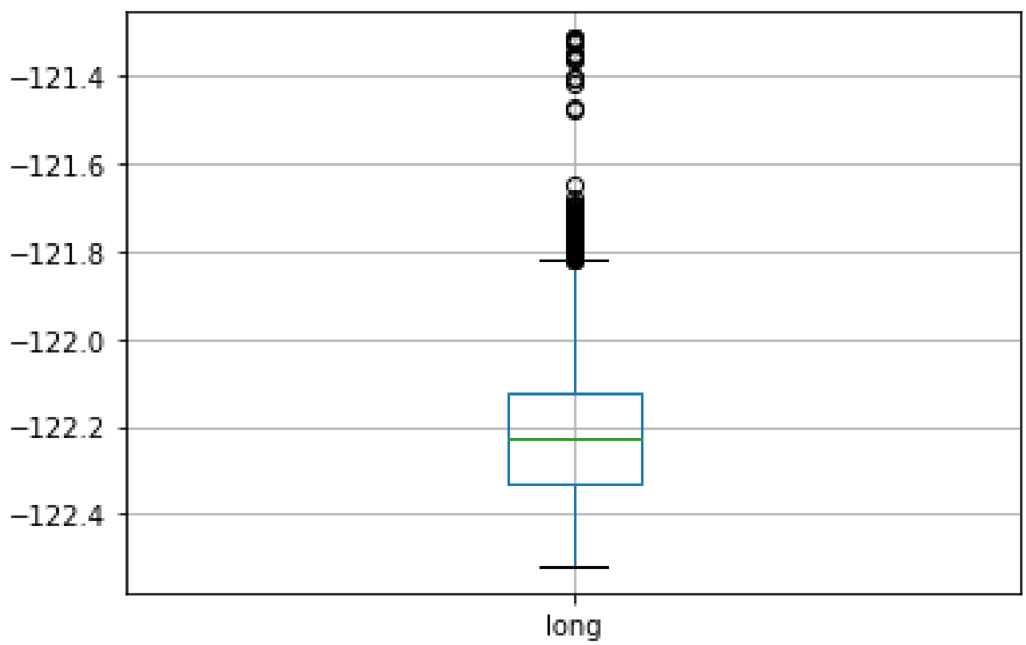

5000
4000
3000
2000
1000
0
sqft_basement

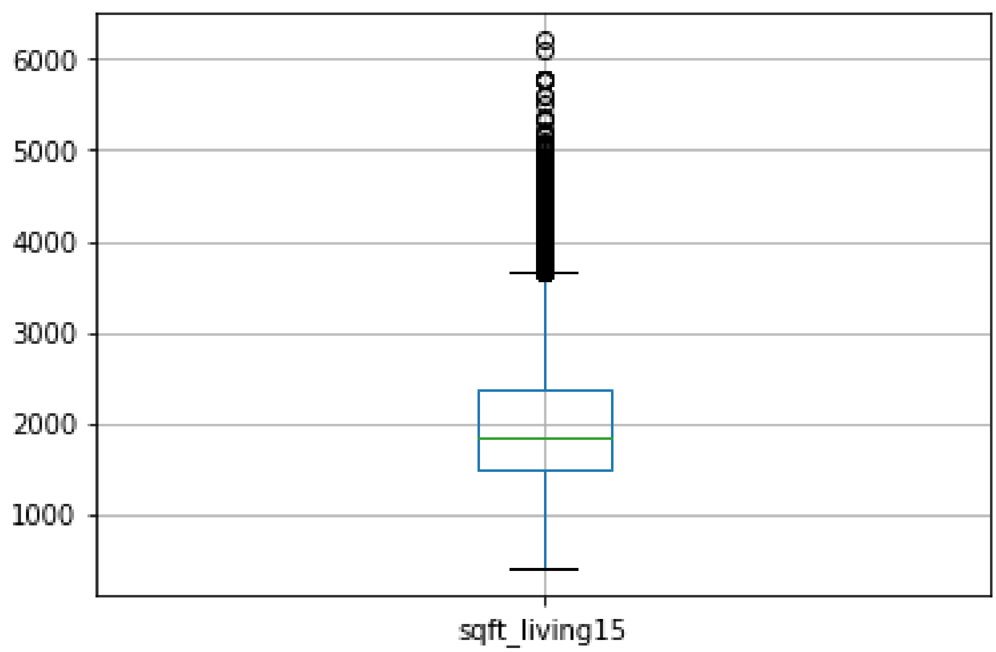

2020
2000
1980
1960
1940
1920
1900
yr_built

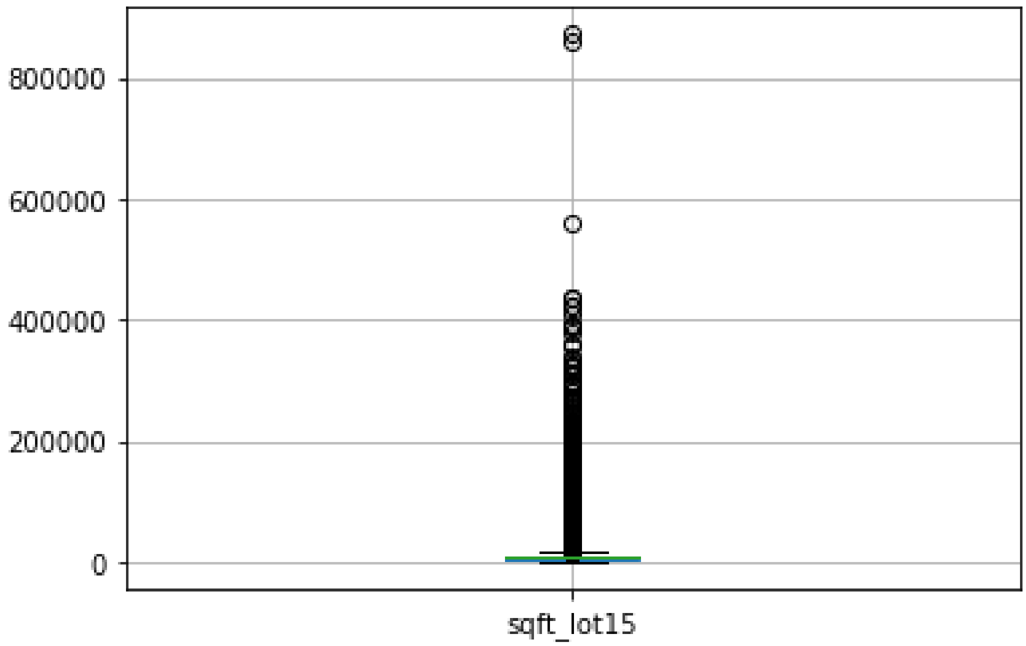

2000
1750
1500
1250
1000
750
500
250
0
yr_renovated


98200
98175
98150
98125
98100
98075
98050
98025
98000
zipcode


47.8
47.7
47.6
47.5
47.4
47.3
47.2
lat


−121.4
−121.6
−121.8
−122.0
−122.2
−122.4
long

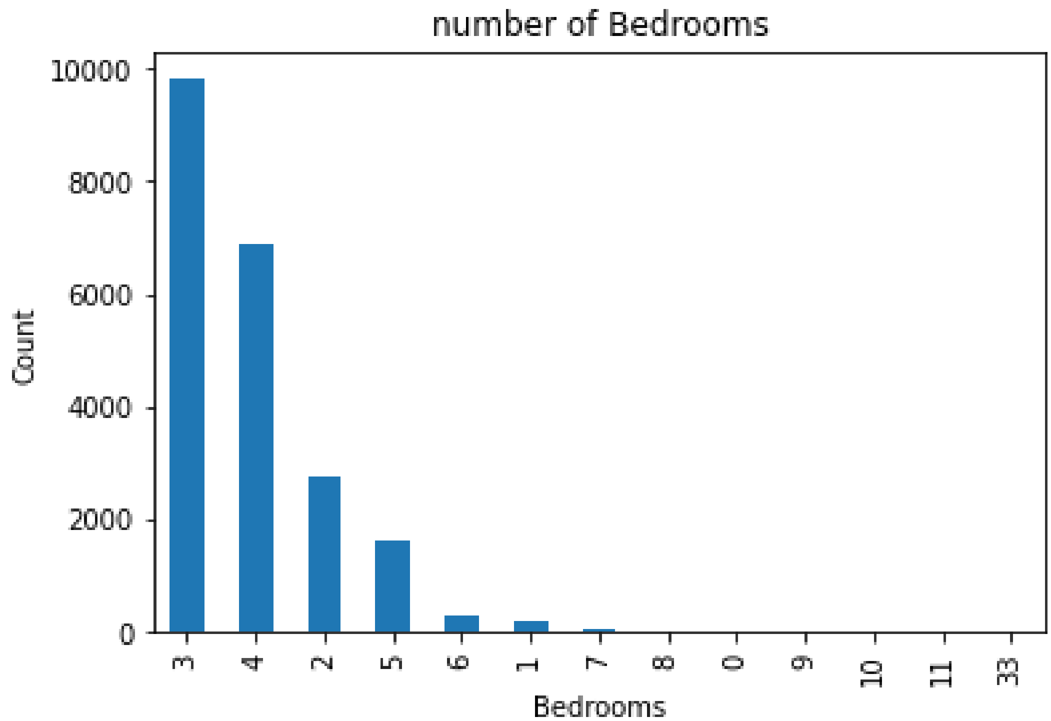

6000
5000
4000
3000
2000
1000
sqft_living15



9 To see the # of houses sold based on number of bedrooms

[17]:
```
House_data['bedrooms'].value_counts().plot(kind='bar')
plt.title('number of Bedrooms')
plt.xlabel('Bedrooms')
plt.ylabel('Count')
sns.despine
```

[17]:
```
<function seaborn.utils.despine(fig=None, ax=None, top=True, right=True,
left=False, bottom=False, offset=None, trim=False)>
```

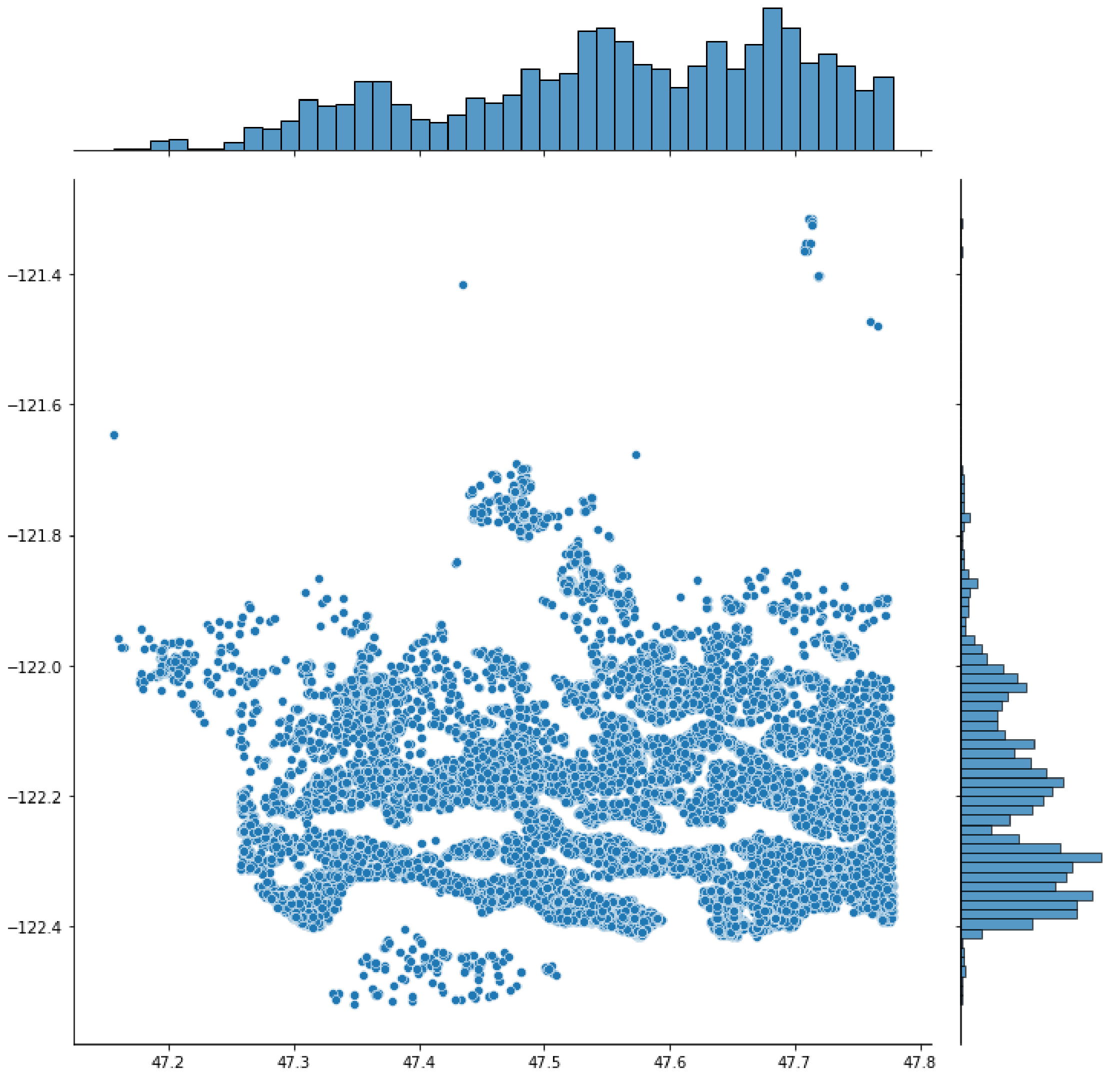


The most common type of house sold are the ones with three bedrooms. So this is helpful as we can understand 3 bedroom houses have more demand followed by 4 bedroom. So # of bedrooms might be an important factor while we fix the price.

10 To check if location (lattitude and longitude have an impact on # of houses sold

[18]:
```
plt.figure(figsize=(10,10))
sns.jointplot(x=House_data.lat.values, y=House_data.long.values, height=10)
plt.ylabel('Longitude', fontsize=12)
plt.xlabel('Latitude', fontsize=12)
plt.show()
sns.despine
```

```
<Figure size 720x720 with 0 Axes>
```



[18]: <function seaborn.utils.despine(fig=None, ax=None, top=True, right=True, left=False, bottom=False, offset=None, trim=False)>

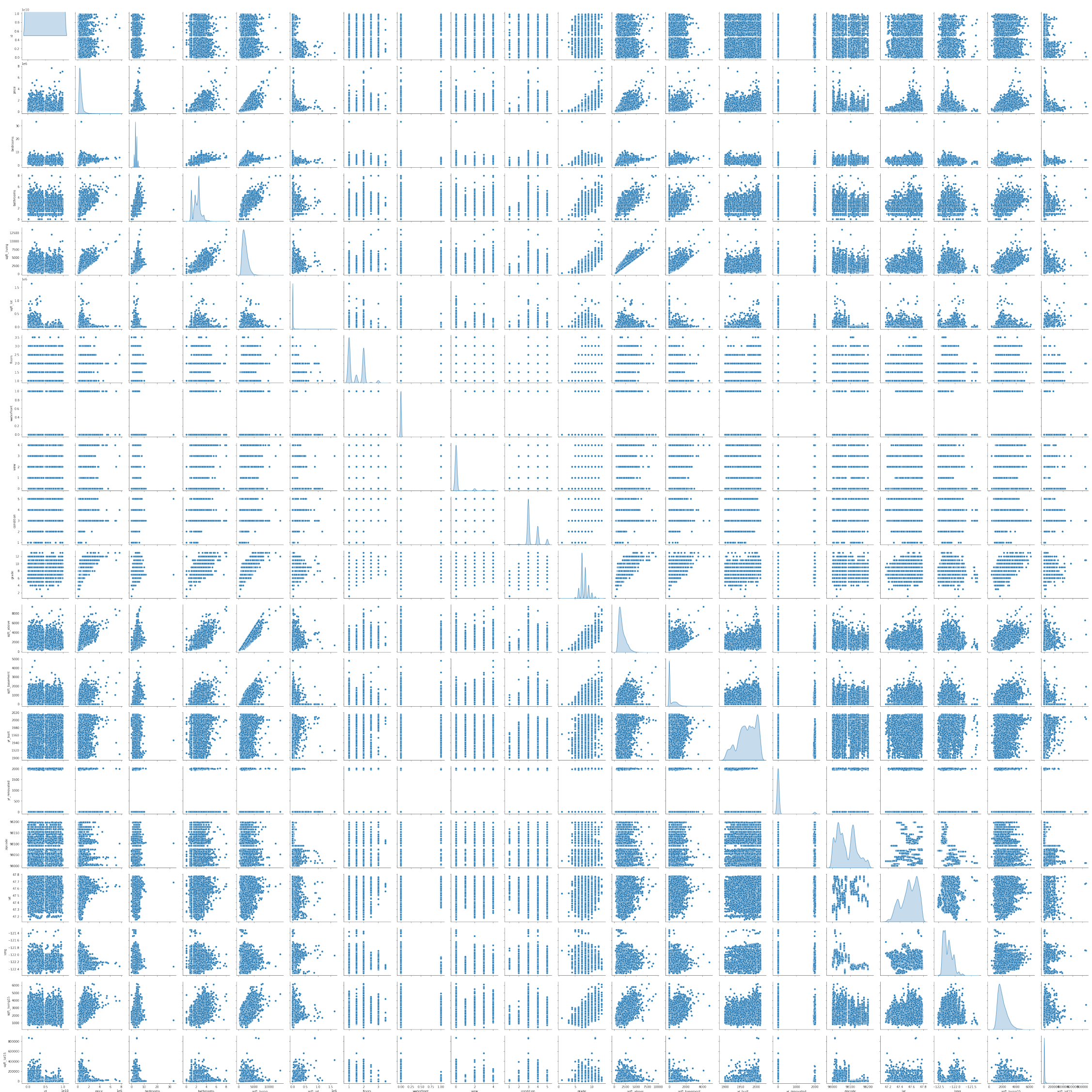
11 Observation

we see large number of houses between latitudes 47.5 and 47.8 and in terms of longitude there are large number of houses between -122.2 and -122.4. This location might be the ideal location for people to live and house prices might vary based up on this lattitude and longitude

12 Bivariate analysis

[19]:
```
HData_attr = House_data.iloc[:, 0:21]
sns.pairplot(HData_attr, diag_kind='kde')
```

[19]: <seaborn.axisgrid.PairGrid at 0x7fa1f9f833d0>



[20]:
```
house_corr = House_data.corr(method ='pearson')
house_corr
```

[20]:
```
                   id     price  bedrooms  bathrooms  sqft_living  sqft_lot  \
id           1.000000 -0.016762  0.001286   0.005160    -0.012258 -0.132109
price       -0.016762  1.000000  0.308350   0.525138     0.702035  0.089661
```

bedrooms	0.001286	0.308350	1.000000	0.515884	0.576671	0.031703
bathrooms	0.005160	0.525138	0.515884	1.000000	0.754665	0.087740
sqft_living	-0.012258	0.702035	0.576671	0.754665	1.000000	0.172826
sqft_lot	-0.132109	0.089661	0.031703	0.087740	0.172826	1.000000
floors	0.018525	0.256794	0.175429	0.500653	0.353949	-0.005201
waterfront	-0.002721	0.266369	-0.006582	0.063744	0.103818	0.021604
view	0.011592	0.397293	0.079532	0.187737	0.284611	0.074710
condition	-0.023783	0.036362	0.028472	-0.124982	-0.058753	-0.008958
grade	0.008130	0.667434	0.356967	0.664983	0.762704	0.113621
sqft_above	-0.010842	0.605567	0.477600	0.685342	0.876597	0.183512
sqft_basement	-0.005151	0.323816	0.303093	0.283770	0.435043	0.015286
yr_built	0.021380	0.054012	0.154178	0.506019	0.318049	0.053080
yr_renovated	-0.016907	0.126434	0.018841	0.050739	0.055363	0.007644
zipcode	-0.008224	-0.053203	-0.152668	-0.203866	-0.199430	-0.129574
lat	-0.001891	0.307003	-0.008931	0.024573	0.052529	-0.085683
long	0.020799	0.021626	0.129473	0.223042	0.240223	0.229521
sqft_living15	-0.002901	0.585379	0.391638	0.568634	0.756420	0.144608
sqft_lot15	-0.138798	0.082447	0.029244	0.087175	0.183286	0.718557

	floors	waterfront	view	condition	grade	\
id	0.018525	-0.002721	0.011592	-0.023783	0.008130	
price	0.256794	0.266369	0.397293	0.036362	0.667434	
bedrooms	0.175429	-0.006582	0.079532	0.028472	0.356967	
bathrooms	0.500653	0.063744	0.187737	-0.124982	0.664983	
sqft_living	0.353949	0.103818	0.284611	-0.058753	0.762704	
sqft_lot	-0.005201	0.021604	0.074710	-0.008958	0.113621	
floors	1.000000	0.023698	0.029444	-0.263768	0.458183	
waterfront	0.023698	1.000000	0.401857	0.016653	0.082775	
view	0.029444	0.401857	1.000000	0.045990	0.251321	
condition	-0.263768	0.016653	0.045990	1.000000	-0.144674	
grade	0.458183	0.082775	0.251321	-0.144674	1.000000	
sqft_above	0.523885	0.072075	0.167649	-0.158214	0.755923	
sqft_basement	-0.245705	0.080588	0.276947	0.174105	0.168392	
yr_built	0.489319	-0.026161	-0.053440	-0.361417	0.446963	
yr_renovated	0.006338	0.092885	0.103917	-0.060618	0.014414	
zipcode	-0.059121	0.030285	0.084827	0.003026	-0.184862	
lat	0.049614	-0.014274	0.006157	-0.014941	0.114084	
long	0.125419	-0.041910	-0.078400	-0.106500	0.198372	
sqft_living15	0.279885	0.086463	0.280439	-0.092824	0.713202	
sqft_lot15	-0.011269	0.030703	0.072575	-0.003406	0.119248	

	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	\
id	-0.010842	-0.005151	0.021380	-0.016907	-0.008224	
price	0.605567	0.323816	0.054012	0.126434	-0.053203	
bedrooms	0.477600	0.303093	0.154178	0.018841	-0.152668	
bathrooms	0.685342	0.283770	0.506019	0.050739	-0.203866	
sqft_living	0.876597	0.435043	0.318049	0.055363	-0.199430	

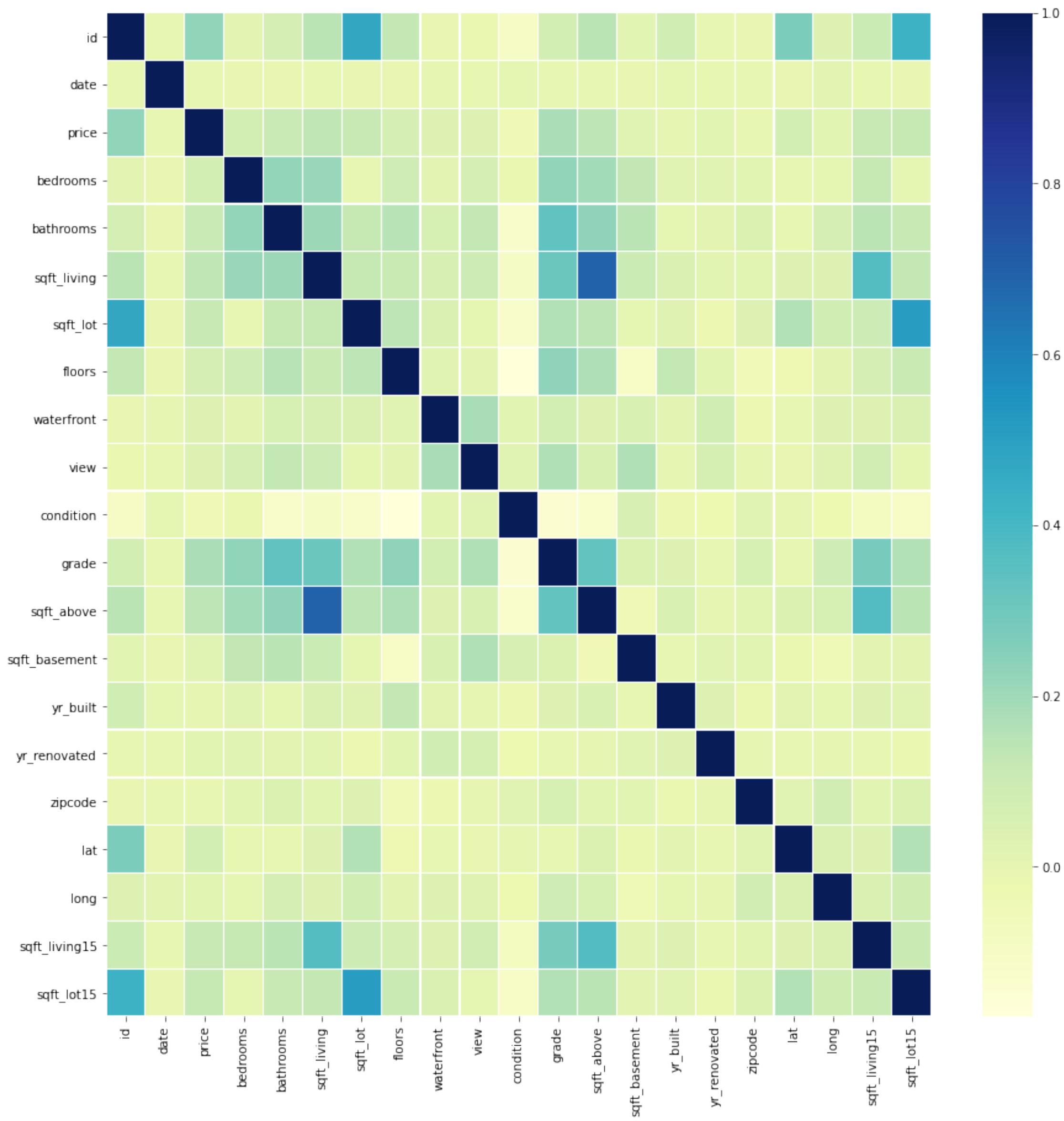
sqft_lot	0.183512	0.015286	0.053080	0.007644	-0.129574
floors	0.523885	-0.245705	0.489319	0.006338	-0.059121
waterfront	0.072075	0.080588	-0.026161	0.092885	0.030285
view	0.167649	0.276947	-0.053440	0.103917	0.084827
condition	-0.158214	0.174105	-0.361417	-0.060618	0.003026
grade	0.755923	0.168392	0.446963	0.014414	-0.184862
sqft_above	1.000000	-0.051943	0.423898	0.023285	-0.261190
sqft_basement	-0.051943	1.000000	-0.133124	0.071323	0.074845
yr_built	0.423898	-0.133124	1.000000	-0.224874	-0.346869
yr_renovated	0.023285	0.071323	-0.224874	1.000000	0.064357
zipcode	-0.261190	0.074845	-0.346869	0.064357	1.000000
lat	-0.000816	0.110538	-0.148122	0.029398	0.267048
long	0.343803	-0.144765	0.409356	-0.068372	-0.564072
sqft_living15	0.731870	0.200355	0.326229	-0.002673	-0.279033
sqft_lot15	0.194050	0.017276	0.070958	0.007854	-0.147221

	lat	long	sqft_living15	sqft_lot15
id	-0.001891	0.020799	-0.002901	-0.138798
price	0.307003	0.021626	0.585379	0.082447
bedrooms	-0.008931	0.129473	0.391638	0.029244
bathrooms	0.024573	0.223042	0.568634	0.087175
sqft_living	0.052529	0.240223	0.756420	0.183286
sqft_lot	-0.085683	0.229521	0.144608	0.718557
floors	0.049614	0.125419	0.279885	-0.011269
waterfront	-0.014274	-0.041910	0.086463	0.030703
view	0.006157	-0.078400	0.280439	0.072575
condition	-0.014941	-0.106500	-0.092824	-0.003406
grade	0.114084	0.198372	0.713202	0.119248
sqft_above	-0.000816	0.343803	0.731870	0.194050
sqft_basement	0.110538	-0.144765	0.200355	0.017276
yr_built	-0.148122	0.409356	0.326229	0.070958
yr_renovated	0.029398	-0.068372	-0.002673	0.007854
zipcode	0.267048	-0.564072	-0.279033	-0.147221
lat	1.000000	-0.135512	0.048858	-0.086419
long	-0.135512	1.000000	0.334605	0.254451
sqft_living15	0.048858	0.334605	1.000000	0.183192
sqft_lot15	-0.086419	0.254451	0.183192	1.000000

13 Observations from bivariate analysis

1. sqft_living, bathrooms, grade, sqft_above, sqft_living15 are the metrics which have high correlation to the target variable price. So these metrics might pop up as important metrics from the regression execrise
2. Also few of the above highlighgted variables are also correlated with each other because of which only few of them might have significant impact on price because of multi-collinearity

[21]:
```
plt.figure(figsize=(15,15))
corr = House_data.apply(lambda x: pd.factorize(x)[0]).corr()
ax = sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns,
                 linewidths=.2, cmap="YlGnBu")
```



14 Data pre processing

[22]:
```
House_data.columns
```

[22]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

15 Defining the independent and dependent variables

[23]:
```
X = House_data.drop(['price','id','date', 'zipcode', 'yr_renovated'],axis=1)
y = House_data[['price']]
```

16 splitting the data into train and test

[24]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
↪random_state=110)
```

17 Linear regression

[25]:
```
regression_model = LinearRegression()
regression_model.fit(X_train, y_train)
```

[25]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

[26]:
```
for idx, col_name in enumerate(X_train.columns):
    print("The coefficient for {} is {}".format(col_name, regression_model.
↪coef_[0][idx]))
```

```
The coefficient for bedrooms is -34957.95534625051
The coefficient for bathrooms is 37791.36613473123
The coefficient for sqft_living is 112.90954569248606
The coefficient for sqft_lot is 0.12284745788929285
The coefficient for floors is 1910.4256828043315
The coefficient for waterfront is 647494.3755909626
The coefficient for view is 48754.01939001865
The coefficient for condition is 28708.66374705164
The coefficient for grade is 97127.37038632511
The coefficient for sqft_above is 74.00698933918648
The coefficient for sqft_basement is 38.90255628569594
The coefficient for yr_built is -2495.663406650898
The coefficient for lat is 563607.1948990341
The coefficient for long is -118441.28306910965
The coefficient for sqft_living15 is 24.908513273173412
The coefficient for sqft_lot15 is -0.37535274720721645
```

[27]:
```python
intercept = regression_model.intercept_[0]
print("The intercept for our model is {}".format(intercept))
```

```
The intercept for our model is -37067704.28226308
```

18 Ridge regression

[52]:
```python
ridge = Ridge(alpha=.3, normalize=True)
ridge.fit(X_train,y_train)
print ("Ridge model:", (ridge.coef_))
```

```
Ridge model: [[-1.32184616e+04  3.76485124e+04  7.35796328e+01  1.02257407e-01
   1.41584163e+04  5.29609327e+05  5.14255707e+04  2.63113721e+04
   6.86695562e+04  7.35538989e+01  6.07646864e+01 -1.58239861e+03
   4.75970530e+05 -1.33538597e+05  5.44129943e+01 -1.55375155e-01]]
```

19 Lasso regression

[57]:
```python
from sklearn.metrics import r2_score, get_scorer
from sklearn.linear_model import Lasso, Ridge, LassoCV,LinearRegression
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.model_selection import KFold, RepeatedKFold, GridSearchCV,␣
 ↪cross_validate, train_test_split
```

[58]:
```python
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
lasso_alphas = np.linspace(0, 0.2, 21)
lasso = Lasso()
grid = dict()
grid['alpha'] = lasso_alphas
gscv = GridSearchCV(lasso,
                    grid,
                    scoring='neg_mean_absolute_error',
                    cv=cv,
                    n_jobs=-1)
results = gscv.fit(X_train, y_train)
print('MAE: %.5f' % results.best_score_)
print('Config: %s' % results.best_params_)
```

```
MAE: -128049.02642
Config: {'alpha': 0.2}
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:474: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 291214902483193.06, tolerance: 208396132543.8401
  model = cd_fast.enet_coordinate_descent(
```

[59]:
```
lasso = Lasso(alpha=0.2, normalize=True)
lasso.fit(X_train,y_train)
print ("Lasso model:", (lasso.coef_))
```

```
Lasso model: [-3.49310499e+04  3.75714299e+04  2.27830103e+02  1.20712994e-01
  1.90651927e+03  6.47262924e+05  4.87640814e+04  2.86651937e+04
  9.71786892e+04 -4.07038739e+01 -7.57620954e+01 -2.49404343e+03
  5.63451621e+05 -1.18207913e+05  2.46767228e+01 -3.72741170e-01]
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:474: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 20395059267863.75, tolerance: 208396132543.8401
  model = cd_fast.enet_coordinate_descent(
```

[54]:
```
print(regression_model.score(X_train, y_train))
print(regression_model.score(X_test, y_test))
```

```
0.6936811522703691
0.697004320132752
```

[55]:
```
print(ridge.score(X_train, y_train))
print(ridge.score(X_test, y_test))
```

```
0.6788592475981722
0.6858185606934188
```

[60]:
```
print(lasso.score(X_train, y_train))
print(lasso.score(X_test, y_test))
```

```
0.6936809831710318
0.6969882855752313
```

There is no considerable increase in the Rsquare value when Ridge or Lasso regression are used instead of linear regression

20 Decision Tree Regression

[33]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn import tree
from sklearn.metrics import r2_score
from sklearn import metrics
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import GridSearchCV
```

[34]:
```
dtree_up = DecisionTreeRegressor()
```

```
dtree_up.fit(X_train, y_train)               # Fitting model with x_train and␣
 ↪y_train
dtree_pred_up = dtree_up.predict(X_test)     # Predicting the results
print('RMSE:', np.sqrt(mean_squared_error(y_test, dtree_pred_up,␣
 ↪squared=False)))
print('r2 score: %.2f' % r2_score(y_test, dtree_pred_up))
print("Accuracy :",dtree_up.score(X_test, y_test))
```

```
RMSE: 439.8282262616803
r2 score: 0.71
Accuracy : 0.7072162909669256
```

21 Hyperparameter tuned Decision Tree Regression

[35]:
```
dtree = DecisionTreeRegressor(random_state=5)
d=np.arange(1,21,1)
hyperParam = [{'max_depth':d}]

gsv = GridSearchCV(dtree,hyperParam,cv=5,verbose=1)
best_model = gsv.fit(X_train, y_train)                          # Fitting model␣
 ↪with xtrain_scaler and y_train
dtree_pred_mms = best_model.best_estimator_.predict(X_test)     # Predicting␣
 ↪the results

print("Best HyperParameter: ",gsv.best_params_)

print('RMSE:', np.sqrt(mean_squared_error(y_test, dtree_pred_mms,␣
 ↪squared=False)))
print('r2 score: %.2f' % r2_score(y_test, dtree_pred_mms))
print("Accuracy :",best_model.score(X_test, y_test))
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

Best HyperParameter:  {'max_depth': 10}
RMSE: 423.2095141165393
r2 score: 0.75
Accuracy : 0.7490216788881379

[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    7.8s finished
```

22 Random forest Regressor

[36]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import GridSearchCV
```

```
from sklearn import metrics
```

[37]:
```
rf = RandomForestRegressor()
rf.fit(X_train, y_train)              # Fitting model with x_train and y_train
rf_pred = rf.predict(X_test)          # Predicting the results
print('RMSE:', np.sqrt(mean_squared_error(y_test, rf_pred, squared=False)))
print('r2 score: %.2f' % r2_score(y_test, rf_pred))
print("Accuracy :",rf.score(X_test, y_test))
```

```
<ipython-input-37-fbe1021d6042>:2: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  rf.fit(X_train, y_train)             # Fitting model with x_train and y_train
```

```
RMSE: 354.67332815079624
r2 score: 0.88
Accuracy : 0.8761977205517678
```

23 Hyperparameter tuned Random forest Regressor

[38]:
```
nEstimator = [140,160,180,200,220]
depth = [10,15,20,25,30]

RF = RandomForestRegressor()
hyperParam = [{'n_estimators':nEstimator,'max_depth': depth}]

gsv = GridSearchCV(RF,hyperParam,cv=5,verbose=1,scoring='r2',n_jobs=-1)
gsv.fit(X_train, y_train)

print("Best HyperParameter: ",gsv.best_params_)
scores = gsv.cv_results_['mean_test_score'].reshape(len(nEstimator),len(depth))
maxDepth=gsv.best_params_['max_depth']
nEstimators=gsv.best_params_['n_estimators']

model = RandomForestRegressor(n_estimators = nEstimators,max_depth=maxDepth)
model.fit(X_train, y_train)         # Fitting model with x_train and y_train

# Predicting the results:
rf_pred_tune = model.predict(X_test)
print('RMSE:', np.sqrt(mean_squared_error(y_test, rf_pred_tune, squared=False)))
print('r2 score: %.2f' % r2_score(y_test, rf_pred_tune))
print("Accuracy :",model.score(X_test, y_test))
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 125 out of 125 | elapsed:  6.2min finished
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/model_selection/_search.py:739: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  self.best_estimator_.fit(X, y, **fit_params)
```

```
Best HyperParameter:  {'max_depth': 20, 'n_estimators': 220}
```

```
<ipython-input-38-54f1e3eaead9>:16: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  model.fit(X_train, y_train)        # Fitting model with x_train and y_train
```

```
RMSE: 355.40809707410995
r2 score: 0.88
Accuracy : 0.8751686142779691
```

24 Gradient Boosting Regressor

[39]:
```
from sklearn import ensemble
clf = ensemble.GradientBoostingRegressor(n_estimators = 400, max_depth = 5,␣
↪min_samples_split = 2,
          learning_rate = 0.1, loss = 'ls')
```

[44]:
```
ensemble.GradientBoostingRegressor?
```

[40]:
```
clf.fit(X_train, y_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/ensemble/_gb.py:1454: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

[40]:
```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=5,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=400,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
```

[41]:
```
y_predict = clf.predict(X_test)
y_predict
```

[41]:
```
array([ 239436.6523064 , 1230930.7026787 ,  762546.14746042, …,
        406636.62260894,  452828.25465846,  723579.23896557])
```

[42]:
```
clf.score(X_test,y_test)
```

[42]: 0.897190695693358

[46]:
```
clf.score?
```

[45]:
```
gb_pred = clf.predict(X_test)         # Predicting the results
print('RMSE:', np.sqrt(mean_squared_error(y_test, gb_pred, squared=False)))
print('r2 score: %.2f' % r2_score(y_test, gb_pred))
print("Accuracy :",clf.score(X_test, y_test))
```

```
RMSE: 338.57467876885994
r2 score: 0.90
Accuracy : 0.897190695693358
```

25 Evaluation of different models (Comparison of different models and performance tuning)

1. Linear regression, Lasso regression, Ridge regression, Decision tree regressor, Hyperparameter tuned decision tree regressor, Random forest regressor, Hyperparameter tuned random forest regressor and Gradient boosting regressor are the different models we tried for this problem.
2. The Rsquare value that we achieved with linear regression is 70%
3. To improve this we tried Lasso and Ridge regression, however in both these cases the Rsquare value obtained was 70%. No improvement in accuracy with respect to linear regression.
4. In Lasso not many coefficients are going to zero implying that there is predictive power in many different variables and not concentrated in few variables
5. Then we tried decision tree and we are able to improve the Rsquare value to 72%
6. Then we tried decision tree with hyperparameter tuning by using Gridsearch and were able to improve the Rsquare value to 75%
7. Then we tried random forest regressor and was able to increase the Rsquare to 87%
8. Then we tried random forest with hyperparameter tuning by using Gridsearch and we found no significant improvement from 88%
9. Finally we tried gradient boosting regressor and were able to achieve R square value of 90%
10. Using grid search in our models we varied different hyperparameters and obtained the optimal hyperparameters.

26 Conclusion

1. To predict the prices of houses we first analysed different independent variables by seeing the correlation of these variables with the target variable price
2. We also looked at the multi-collinearity of this independent variables
3. We then applied multi variate linear regression and achieved a test accuracy of 70%
4. We then applied a large group of models and the best accuracy obtained from those models is from gradient boosting regressor which is 88%
5. It is once again observed that ensemble techniques help us in achieving high accuracy either it is regression or classification as it involves a large group models and best of these models is taken