

cosxu2ucv

May 1, 2023

TP Machine Learning INSA LYON

Copyright (C) 2023 Kanaan Kevin, Foltête François, Reis Alexis, Vote Robin

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>.

Le but du TP est de prédire les prix de maisons en fonction de différentes variables

0.1 Importation des librairies à utiliser

```
[130]: import datetime as dt
import pandas as pd
import numpy as np
#Libraries for data visualization
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import KFold
from sklearn.metrics import explained_variance_score
```

```
[131]: import warnings
warnings.filterwarnings('ignore')
```

0.2 Visualisation générale des données

0.2.1 Read csv file

```
[132]: df = pd.read_csv('kc_house_data.csv')
```

0.2.2 Affichages des 5 premières lignes du dataset

```
[133]: df.head()
```

```
[133]:
```

	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]

0.2.3 Describe pour visualiser les données avec pandas

```
[134]: df.info()  
#Visualize the data that is in the csv  
df.describe()
```

```
<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
```

```
[134]:
```

	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

```
[135]: df.isnull().sum()
```

```
[135]: id          0
       date        0
       price       0
       bedrooms    0
       bathrooms   0
       sqft_living  0
       sqft_lot     0
       floors       0
       waterfront   0
       view         0
       condition    0
       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
dtype: int64
```

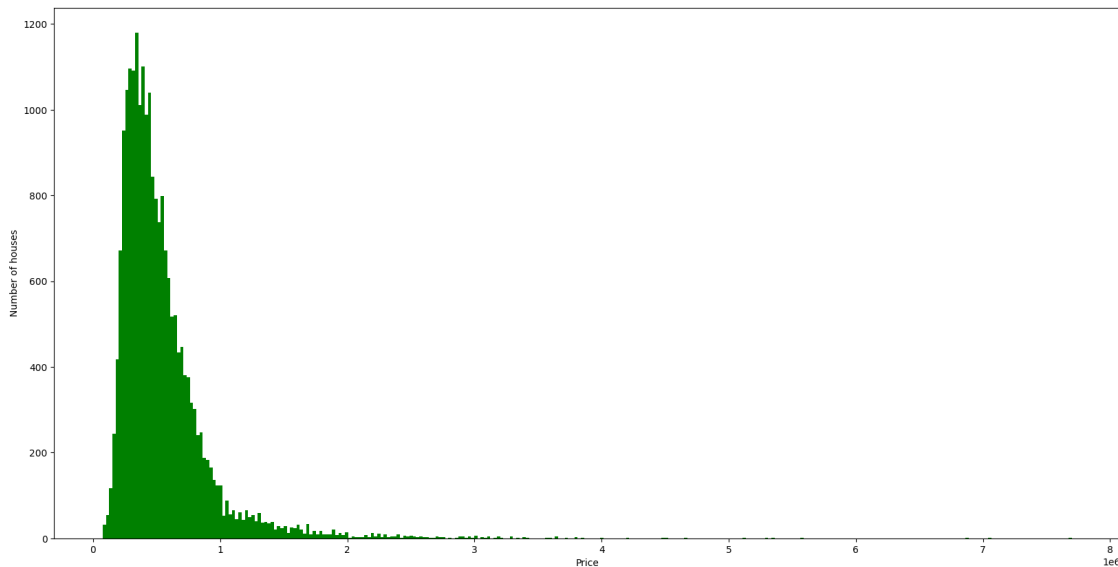
On constate qu'aucune variable ne contient de valeur nulle.

0.3 Visualisation par variable - On utilise pyplot pour afficher des infos utiles sur nos données

Pour chaque variable on affiche un histogramme pour en apprendre plus sur les variables

0.3.1 Price info

```
[136]: plt.figure(figsize=(20,10))
plt.hist(df['price'], bins=300, color='green')
plt.xlabel('Price')
plt.ylabel('Number of houses')
plt.show()
```



```
[137]: #price describe
df['price'].describe()
```

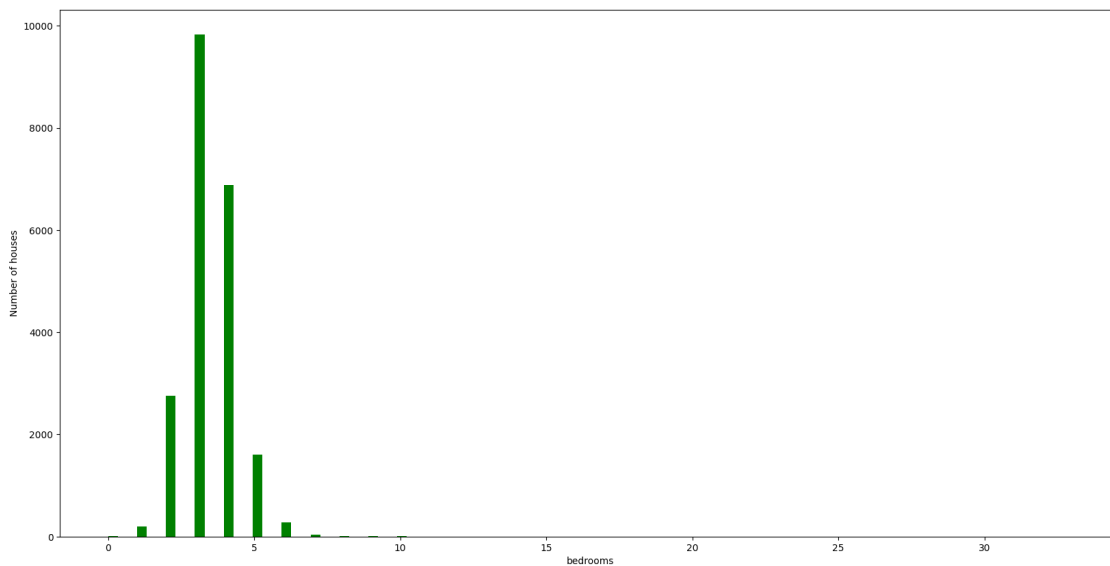
```
[137]: count      2.161300e+04
mean       5.400881e+05
std        3.671272e+05
min        7.500000e+04
25%        3.219500e+05
50%        4.500000e+05
```

```
75%      6.450000e+05
max      7.700000e+06
Name: price, dtype: float64
```

La distribution des prix est normale.

0.3.2 Bedrooms info

```
[138]: plt.figure(figsize=(20,10))
plt.hist(df['bedrooms'], bins=100, color='green')
plt.xlabel('bedrooms')
plt.ylabel('Number of houses')
plt.show()
```



```
[139]: df['bedrooms'].describe()
```

```
[139]: count      21613.000000
mean         3.370842
std          0.930062
min          0.000000
25%          3.000000
50%          3.000000
75%          4.000000
max          33.000000
Name: bedrooms, dtype: float64
```

```
[140]: df['bedrooms'].value_counts()
```

```
[140]: 3      9824
        4      6882
        2      2760
        5      1601
        6       272
        1       199
        7        38
        0        13
        8        13
        9         6
       10         3
       11         1
       33         1
Name: bedrooms, dtype: int64
```

```
[141]: # print info on the row with 33 bedrooms
df[df['bedrooms'] == 33]
```

```
[141]:          id          date    price  bedrooms  bathrooms \
15870  2402100895  20140625T000000  640000.0         33         1.75

          sqft_living  sqft_lot  floors  waterfront  view  ...  grade \
15870          1620      6000     1.0           0     0  ...      7

          sqft_above  sqft_basement  yr_built  yr_renovated  zipcode      lat \
15870          1040           580      1947           0     98103  47.6878

          long  sqft_living15  sqft_lot15
15870 -122.331          1330          4700

[1 rows x 21 columns]
```

```
[142]: df[df['bedrooms'] == 11]
```

```
[142]:          id          date    price  bedrooms  bathrooms  sqft_living \
8757  1773100755  20140821T000000  520000.0         11         3.0         3000

          sqft_lot  floors  waterfront  view  ...  grade  sqft_above \
8757          4960     2.0           0     0  ...      7          2400

          sqft_basement  yr_built  yr_renovated  zipcode      lat      long \
8757           600      1918          1999     98106  47.556 -122.363

          sqft_living15  sqft_lot15
8757          1420          4960

[1 rows x 21 columns]
```

```
[143]: df[df['bedrooms'] == 0]
#id 6994 valeur abérante
```

```
[143]:
```

	id	date	price	bedrooms	bathrooms	\
875	6306400140	20140612T000000	1095000.0	0	0.00	
3119	3918400017	20150205T000000	380000.0	0	0.00	
3467	1453602309	20140805T000000	288000.0	0	1.50	
4868	6896300380	20141002T000000	228000.0	0	1.00	
6994	2954400190	20140624T000000	1295650.0	0	0.00	
8477	2569500210	20141117T000000	339950.0	0	2.50	
8484	2310060040	20140925T000000	240000.0	0	2.50	
9773	3374500520	20150429T000000	355000.0	0	0.00	
9854	7849202190	20141223T000000	235000.0	0	0.00	
12653	7849202299	20150218T000000	320000.0	0	2.50	
14423	9543000205	20150413T000000	139950.0	0	0.00	
18379	1222029077	20141029T000000	265000.0	0	0.75	
19452	3980300371	20140926T000000	142000.0	0	0.00	

	sqft_living	sqft_lot	floors	waterfront	view	...	grade	\
875	3064	4764	3.5	0	2	...	7	
3119	1470	979	3.0	0	2	...	8	
3467	1430	1650	3.0	0	0	...	7	
4868	390	5900	1.0	0	0	...	4	
6994	4810	28008	2.0	0	0	...	12	
8477	2290	8319	2.0	0	0	...	8	
8484	1810	5669	2.0	0	0	...	7	
9773	2460	8049	2.0	0	0	...	8	
9854	1470	4800	2.0	0	0	...	7	
12653	1490	7111	2.0	0	0	...	7	
14423	844	4269	1.0	0	0	...	7	
18379	384	213444	1.0	0	0	...	4	
19452	290	20875	1.0	0	0	...	1	

	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	\
875	3064	0	1990	0	98102	47.6362	
3119	1470	0	2006	0	98133	47.7145	
3467	1430	0	1999	0	98125	47.7222	
4868	390	0	1953	0	98118	47.5260	
6994	4810	0	1990	0	98053	47.6642	
8477	2290	0	1985	0	98042	47.3473	
8484	1810	0	2003	0	98038	47.3493	
9773	2460	0	1990	0	98031	47.4095	
9854	1470	0	1996	0	98065	47.5265	
12653	1490	0	1999	0	98065	47.5261	
14423	844	0	1913	0	98001	47.2781	
18379	384	0	2003	0	98070	47.4177	
19452	290	0	1963	0	98024	47.5308	

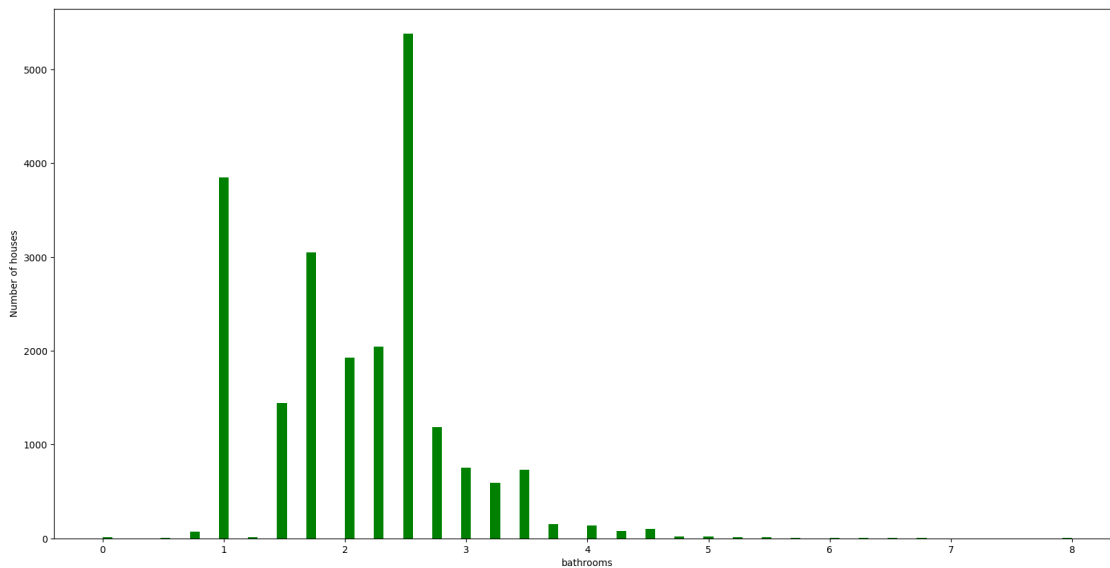
	long	sqft_living15	sqft_lot15
875	-122.322	2360	4000
3119	-122.356	1470	1399
3467	-122.290	1430	1650
4868	-122.261	2170	6000
6994	-122.069	4740	35061
8477	-122.151	2500	8751
8484	-122.053	1810	5685
9773	-122.168	2520	8050
9854	-121.828	1060	7200
12653	-121.826	1500	4675
14423	-122.250	1380	9600
18379	-122.491	1920	224341
19452	-121.888	1620	22850

[13 rows x 21 columns]

On remarque que des maisons ont 0 et 33 chambres, ce sont des valeurs aberrantes que nous nettoierons.

0.3.3 Bathrooms info

```
[144]: plt.figure(figsize=(20,10))
plt.hist(df['bathrooms'], bins=100, color='green')
plt.xlabel('bathrooms')
plt.ylabel('Number of houses')
plt.show()
```



```
[145]: #print info about bathrooms
df['bathrooms'].describe()
```

```
[145]: count      21613.000000
      mean         2.114757
      std         0.770163
      min         0.000000
      25%         1.750000
      50%         2.250000
      75%         2.500000
      max         8.000000
      Name: bathrooms, dtype: float64
```

```
[146]: df['bathrooms'].value_counts()
```

```
[146]: 2.50      5380
      1.00      3852
      1.75      3048
      2.25      2047
      2.00      1930
      1.50      1446
      2.75      1185
      3.00       753
      3.50       731
      3.25       589
      3.75       155
      4.00       136
      4.50       100
      4.25        79
      0.75        72
      4.75        23
      5.00        21
      5.25        13
      0.00         10
      5.50         10
      1.25         9
      6.00         6
      0.50         4
      5.75         4
      6.75         2
      8.00         2
      6.25         2
      6.50         2
      7.50         1
      7.75         1
      Name: bathrooms, dtype: int64
```

```
[147]: #Show the line where bathrooms is 8
df[df['bathrooms'] == 8]
```

```
[147]:
```

	id	date	price	bedrooms	bathrooms	\
7252	6762700020	20141013T000000	7700000.0	6	8.0	
12777	1225069038	20140505T000000	2280000.0	7	8.0	

	sqft_living	sqft_lot	floors	waterfront	view	...	grade	\
7252	12050	27600	2.5	0	3	...	13	
12777	13540	307752	3.0	0	4	...	12	

	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	\
7252	8570	3480	1910	1987	98102	47.6298	
12777	9410	4130	1999	0	98053	47.6675	

	long	sqft_living15	sqft_lot15
7252	-122.323	3940	8800
12777	-121.986	4850	217800

[2 rows x 21 columns]

```
[148]: df[df['bathrooms'] == 0]
#id 875 abb rant
```

```
[148]:
```

	id	date	price	bedrooms	bathrooms	\
875	6306400140	20140612T000000	1095000.0	0	0.0	
1149	3421079032	20150217T000000	75000.0	1	0.0	
3119	3918400017	20150205T000000	380000.0	0	0.0	
5832	5702500050	20141104T000000	280000.0	1	0.0	
6994	2954400190	20140624T000000	1295650.0	0	0.0	
9773	3374500520	20150429T000000	355000.0	0	0.0	
9854	7849202190	20141223T000000	235000.0	0	0.0	
10481	203100435	20140918T000000	484000.0	1	0.0	
14423	9543000205	20150413T000000	139950.0	0	0.0	
19452	3980300371	20140926T000000	142000.0	0	0.0	

	sqft_living	sqft_lot	floors	waterfront	view	...	grade	\
875	3064	4764	3.5	0	2	...	7	
1149	670	43377	1.0	0	0	...	3	
3119	1470	979	3.0	0	2	...	8	
5832	600	24501	1.0	0	0	...	3	
6994	4810	28008	2.0	0	0	...	12	
9773	2460	8049	2.0	0	0	...	8	
9854	1470	4800	2.0	0	0	...	7	
10481	690	23244	1.0	0	0	...	7	
14423	844	4269	1.0	0	0	...	7	
19452	290	20875	1.0	0	0	...	1	

	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	\
875	3064	0	1990	0	98102	47.6362	
1149	670	0	1966	0	98022	47.2638	
3119	1470	0	2006	0	98133	47.7145	
5832	600	0	1950	0	98045	47.5316	
6994	4810	0	1990	0	98053	47.6642	
9773	2460	0	1990	0	98031	47.4095	
9854	1470	0	1996	0	98065	47.5265	
10481	690	0	1948	0	98053	47.6429	
14423	844	0	1913	0	98001	47.2781	
19452	290	0	1963	0	98024	47.5308	

	long	sqft_living15	sqft_lot15
875	-122.322	2360	4000
1149	-121.906	1160	42882
3119	-122.356	1470	1399
5832	-121.749	990	22549
6994	-122.069	4740	35061
9773	-122.168	2520	8050
9854	-121.828	1060	7200
10481	-121.955	1690	19290
14423	-122.250	1380	9600
19452	-121.888	1620	22850

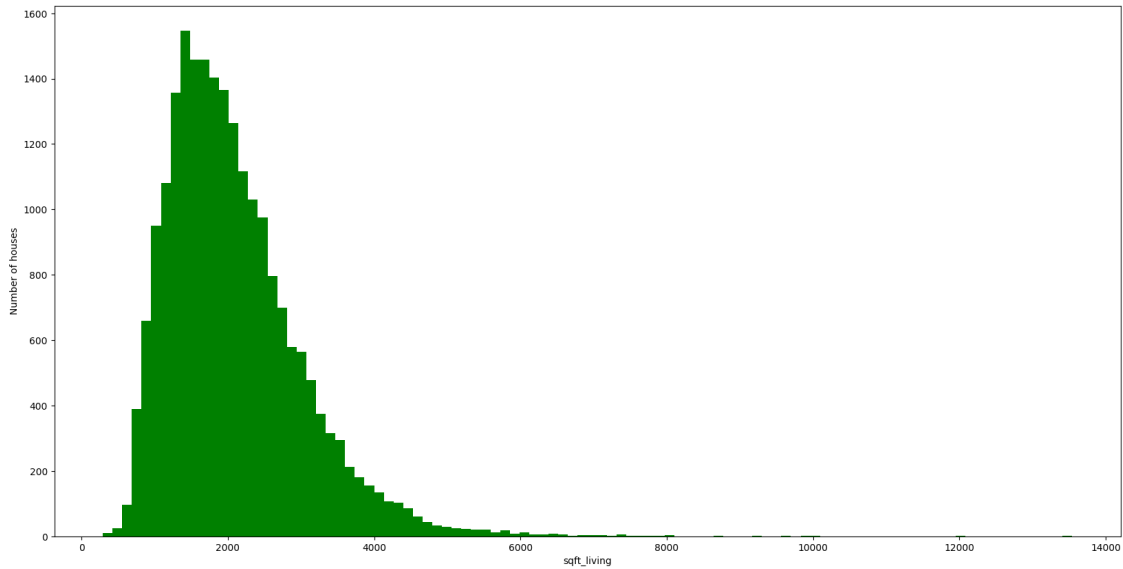
[10 rows x 21 columns]

On remarque ici que le nombre de bathrooms est un décimal et pas un int parce que on considère plusieurs types de Bathrooms. Source : <https://www.kaggle.com/datasets/harlfoxem/housesalesprediction/discussion/24804?resource=download>
On a des “full-bathrooms” des “semi” etc...

0.3.4 sqft living info

Représente la superficie intérieure

```
[149]: plt.figure(figsize=(20,10))
plt.hist(df['sqft_living'], bins=100, color='green')
plt.xlabel('sqft_living')
plt.ylabel('Number of houses')
plt.show()
```



```
[150]: #print info about sqft_living
df['sqft_living'].describe()
```

```
[150]: count      21613.000000
      mean       2079.899736
      std        918.440897
      min        290.000000
      25%       1427.000000
      50%       1910.000000
      75%       2550.000000
      max      13540.000000
      Name: sqft_living, dtype: float64
```

Avoir une visualisation des tranches de 100 square foot

```
[151]: # On arrondis au millier pour pouvoir compter la fréquence de chaque tranche
      ↪ (de 100)
sqft_living_arrondis = [sq // 100 * 100 for sq in df['sqft_living']]
# Vérification de l'arrondis
# print(sqft_living_100[:100])
```

```
[152]: pd_sqft_living_arrondis = pd.DataFrame(sqft_living_arrondis, dtype=int)
pd_sqft_living_arrondis.value_counts()
```

```
[152]: 1400      1158
      1600      1135
      1500      1123
      1700      1078
```

```

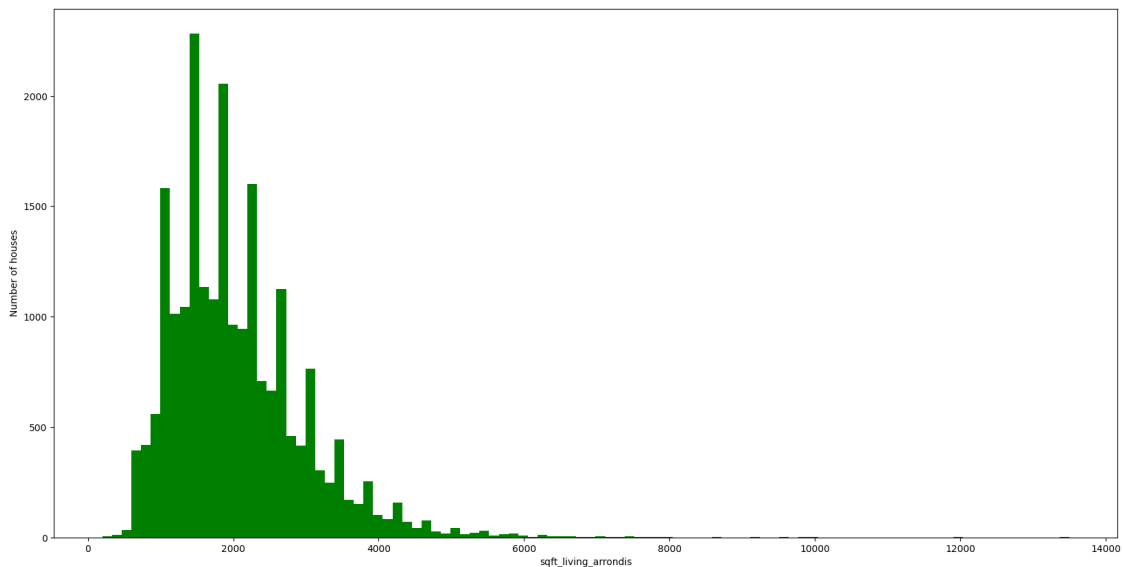
1800      1066
...
200        1
9200       1
8600       1
7600       1
13500      1
Length: 83, dtype: int64

```

```

[153]: plt.figure(figsize=(20,10))
plt.hist(sqft_living_arrondis, bins=100, color='green')
plt.xlabel('sqft_living_arrondis')
plt.ylabel('Number of houses')
plt.show()

```



Vérification de la maison avec la valeur **max** pour *sqft_living*.

```

[154]: df[df['sqft_living'] == 13540.000000]

```

```

[154]:
      id      date      price  bedrooms  bathrooms \
12777 1225069038 20140505T000000 2280000.0         7         8.0

      sqft_living  sqft_lot  floors  waterfront  view  ...  grade  \
12777      13540    307752     3.0           0     4  ...     12

      sqft_above  sqft_basement  yr_built  yr_renovated  zipcode    lat  \
12777      9410         4130     1999           0     98053  47.6675

```

```

      long  sqft_living15  sqft_lot15
12777 -121.986          4850      217800

```

```
[1 rows x 21 columns]
```

Cette maison coûte 2 280 000 \$ cela paraît cohérent.

Vérification de la maison avec la valeur **min** pour *sqft_living*.

```
[155]: df[df['sqft_living'] == 290]
```

```

[155]:
      id      date      price  bedrooms  bathrooms \
19452  3980300371  20140926T000000  142000.0          0          0.0

      sqft_living  sqft_lot  floors  waterfront  view  ...  grade  \
19452          290      20875      1.0          0      0  ...      1

      sqft_above  sqft_basement  yr_built  yr_renovated  zipcode      lat  \
19452          290              0      1963              0      98024  47.5308

      long  sqft_living15  sqft_lot15
19452 -121.888          1620      22850

```

```
[1 rows x 21 columns]
```

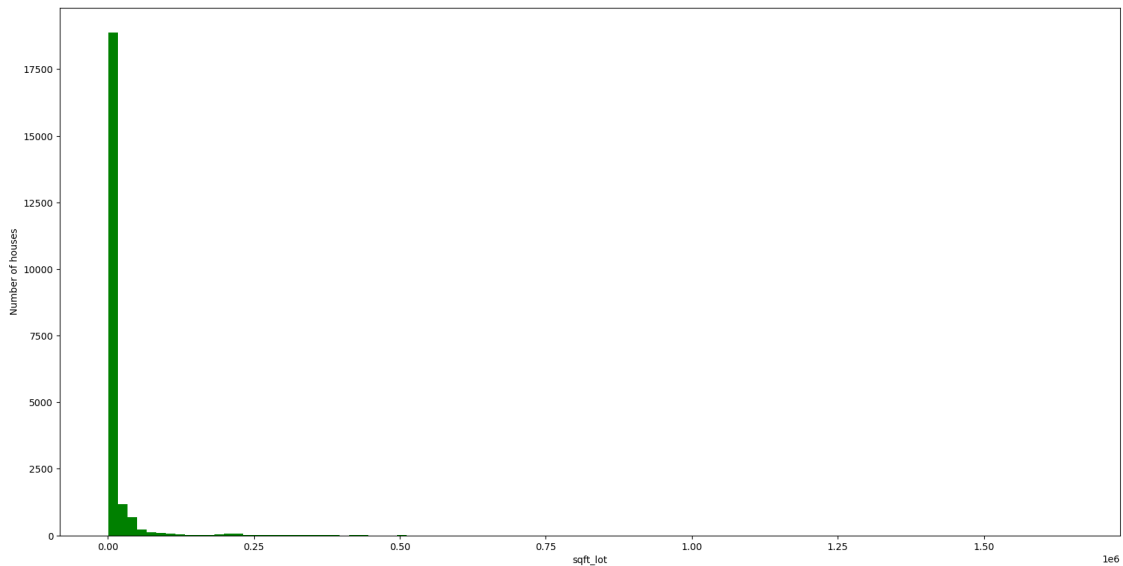
En regardant sur google maps, on observe un cabanon autour d'un lac. De plus le terrain est grand. Le prix paraît cohérent.

0.3.5 sqft lot info

```

[156]: plt.figure(figsize=(20,10))
plt.hist(df['sqft_lot'], bins=100, color='green')
plt.xlabel('sqft_lot')
plt.ylabel('Number of houses')
plt.show()

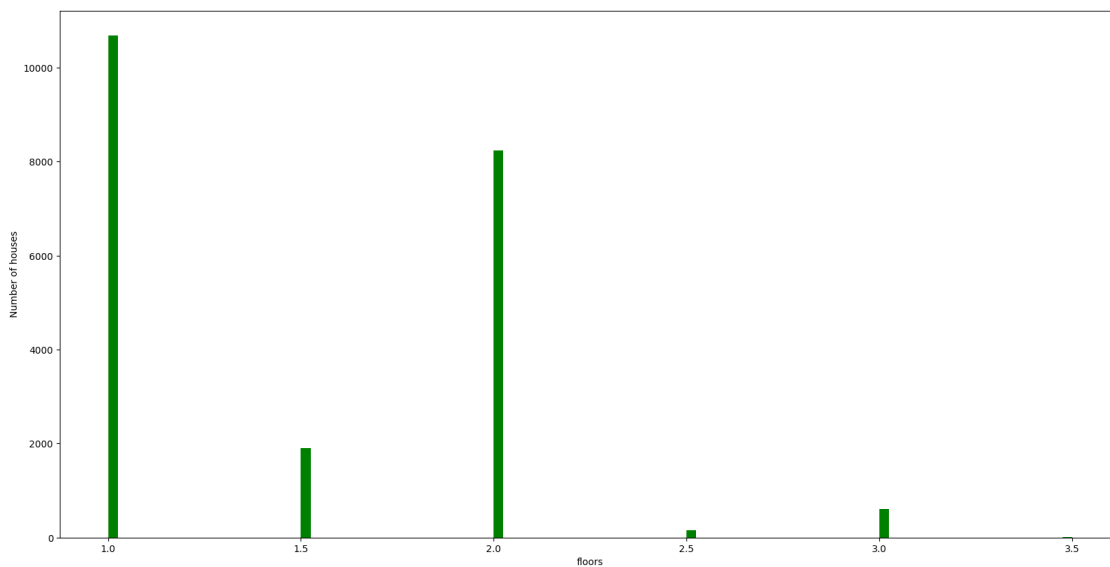
```



On constate que cette variable ne suit pas une loi normale. Nous allons donc par la suite prendre son logarithme.

0.3.6 Floors info

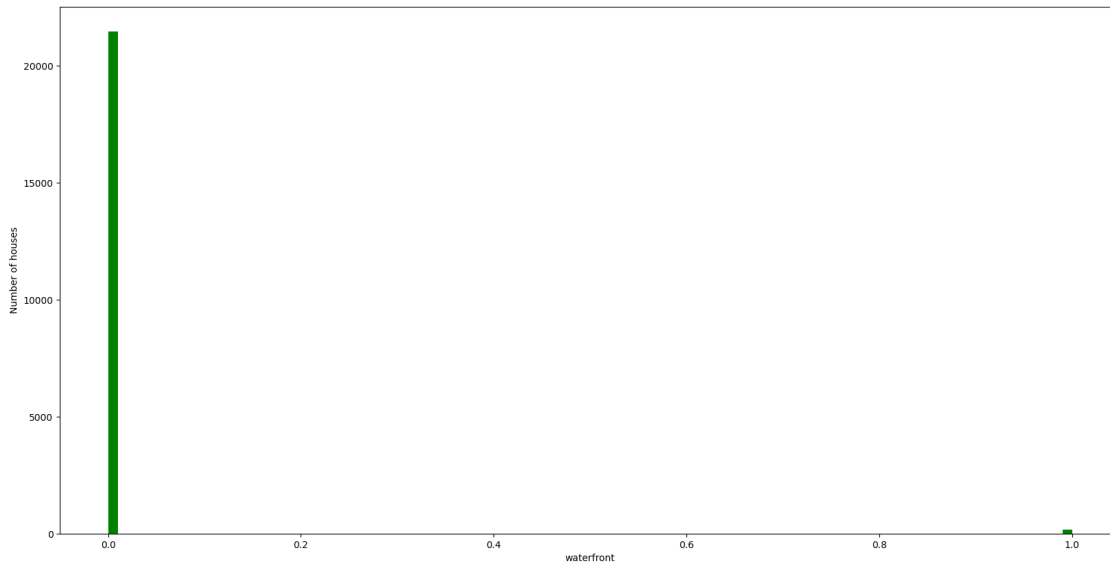
```
[157]: plt.figure(figsize=(20,10))
plt.hist(df['floors'], bins=100, color='green')
plt.xlabel('floors')
plt.ylabel('Number of houses')
plt.show()
```



Nombre d'étages. Les demi-étages peuvent correspondre à des mezzanines.

0.3.7 Waterfront info

```
[158]: plt.figure(figsize=(20,10))
plt.hist(df['waterfront'], bins=100, color='green')
plt.xlabel('waterfront')
plt.ylabel('Number of houses')
plt.show()
```



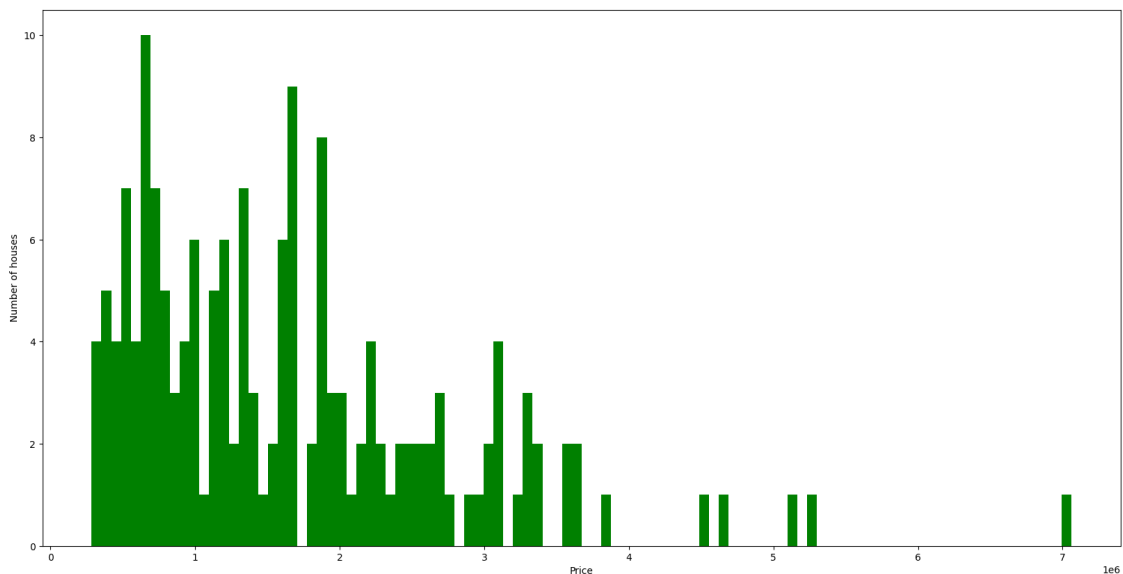
```
[159]: df['waterfront'].value_counts()
```

```
[159]: 0    21450
      1     163
      Name: waterfront, dtype: int64
```

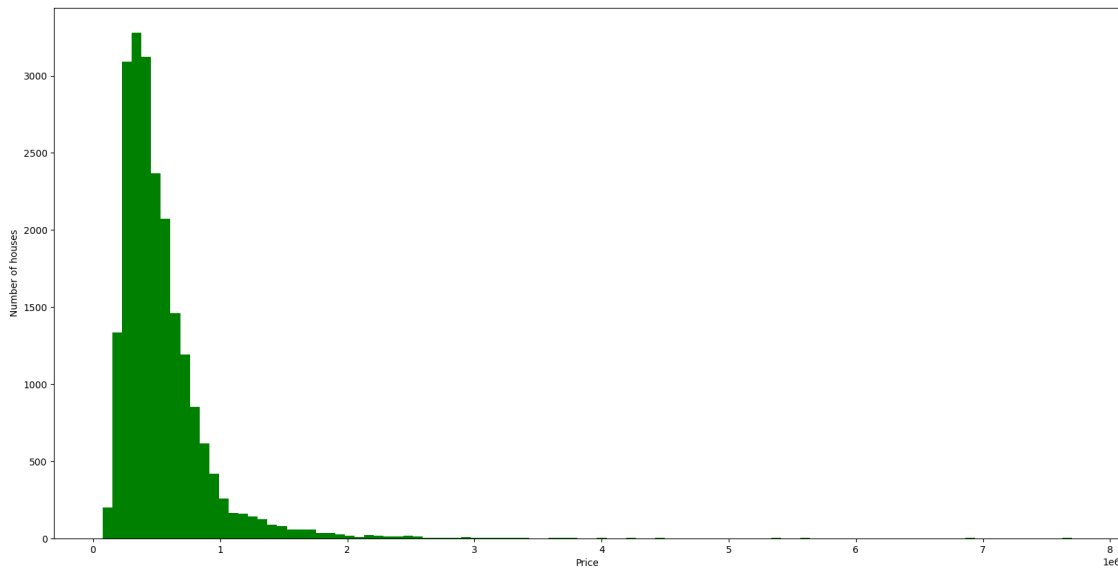
```
[160]: df['waterfront'].describe()
```

```
[160]: count    21613.000000
      mean         0.007542
      std         0.086517
      min         0.000000
      25%         0.000000
      50%         0.000000
      75%         0.000000
      max         1.000000
      Name: waterfront, dtype: float64
```

```
[161]: #Show the price of the houses with a waterfront
df[df['waterfront'] == 1]['price'].describe()
#Price histogramme of the houses with a waterfront
plt.figure(figsize=(20,10))
plt.hist(df[df['waterfront'] == 1]['price'], bins=100, color='green')
plt.xlabel('Price')
plt.ylabel('Number of houses')
plt.show()
```



```
[162]: #Show the price of the houses without a waterfront
df[df['waterfront'] == 0]['price'].describe()
#Price histogramme of the houses without a waterfront
plt.figure(figsize=(20,10))
plt.hist(df[df['waterfront'] == 0]['price'], bins=100, color='green')
plt.xlabel('Price')
plt.ylabel('Number of houses')
plt.show()
```



```
[163]: #Select the house with waterfront
waterfronts=df[df['waterfront'] == 1]
#Get the row with the highest price in the waterfronts
waterfronts[waterfronts['price'] == waterfronts['price'].max()]
```

```
[163]:           id           date      price  bedrooms  bathrooms \
3914  9808700762  20140611T000000  7062500.0           5           4.5

           sqft_living  sqft_lot  floors  waterfront  view  ...  grade  sqft_above  \
3914           10040      37325      2.0           1      2  ...      11           7680

           sqft_basement  yr_built  yr_renovated  zipcode    lat    long  \
3914           2360      1940           2001    98004  47.65 -122.214

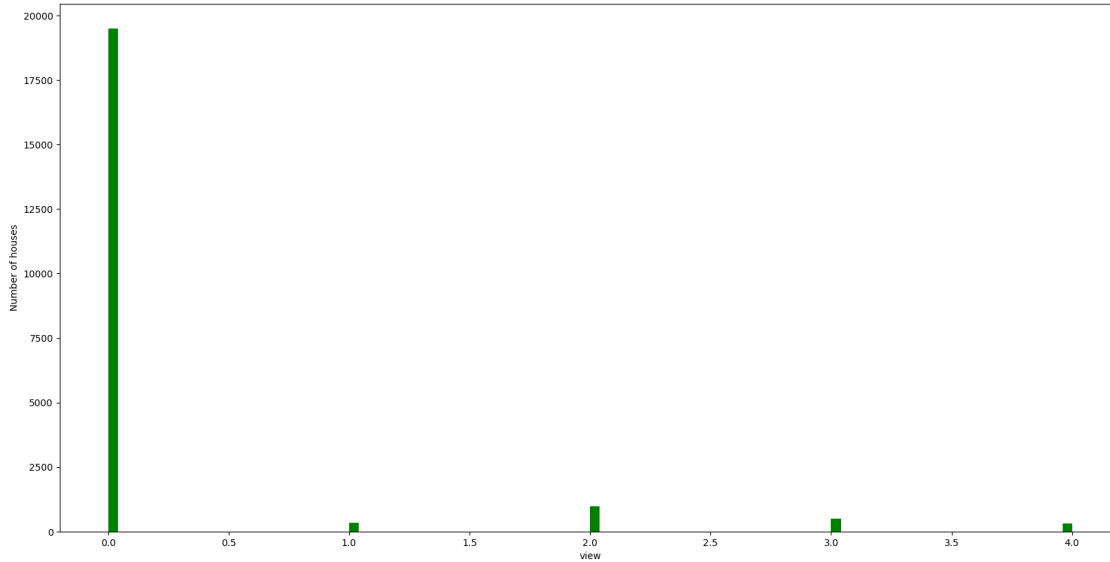
           sqft_living15  sqft_lot15
3914           3930      25449

[1 rows x 21 columns]
```

0 ou 1 en fonction de si la maison est en face de l'eau ou non.

0.3.8 View info

```
[164]: plt.figure(figsize=(20,10))
plt.hist(df['view'], bins=100, color='green')
plt.xlabel('view')
plt.ylabel('Number of houses')
plt.show()
```



```
[165]: df['view'].describe()
```

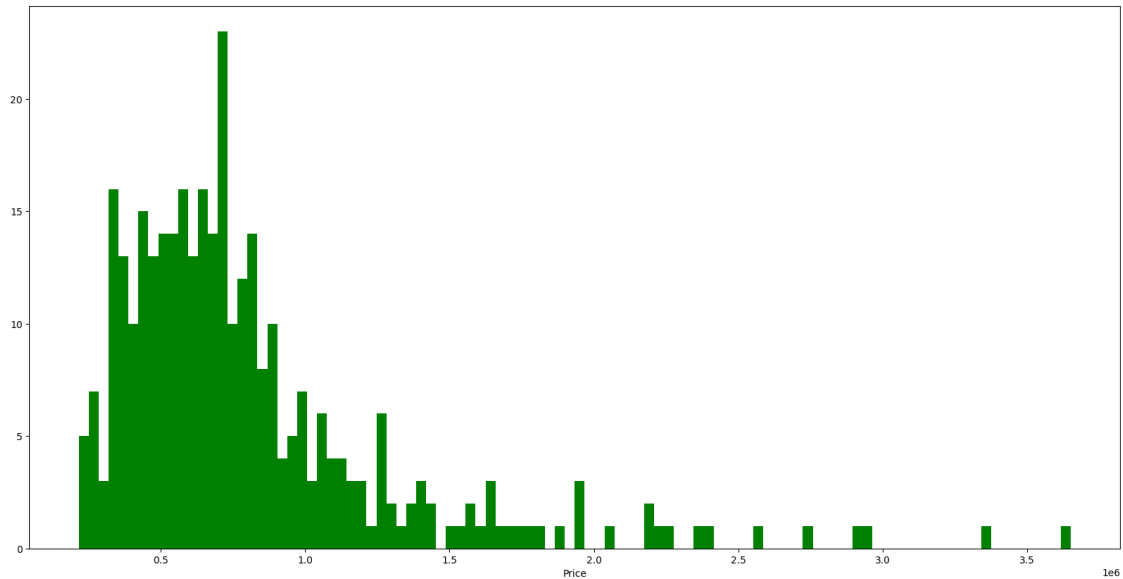
```
[165]: count    21613.000000
      mean      0.234303
      std       0.766318
      min       0.000000
      25%       0.000000
      50%       0.000000
      75%       0.000000
      max       4.000000
      Name: view, dtype: float64
```

```
[166]: df['view'].value_counts()
```

```
[166]: 0    19489
      2     963
      3     510
      1     332
      4     319
      Name: view, dtype: int64
```

```
[167]: #Show the price of the houses with a view
      df[df['view'] == 1]['price'].describe()
      #Price histogramme of the houses with a view
      plt.figure(figsize=(20,10))
      plt.hist(df[df['view'] == 1]['price'], bins=100, color='green')
      plt.xlabel('Price')
```

```
[167]: Text(0.5, 0, 'Price')
```



```
[168]: #Choose the row with the highest price and view value of 4
views=df[(df['view'] == 4)]
views[views['price'] == views['price'].max()]
```

```
[168]:
```

	id	date	price	bedrooms	bathrooms	\
9254	9208900037	20140919T000000	6885000.0	6	7.75	

	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	\
9254	9890	31374	2.0	0	4	...	13	8860	

	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	\
9254	1030	2001	0	98039	47.6305	-122.24	

	sqft_living15	sqft_lot15
9254	4540	42730

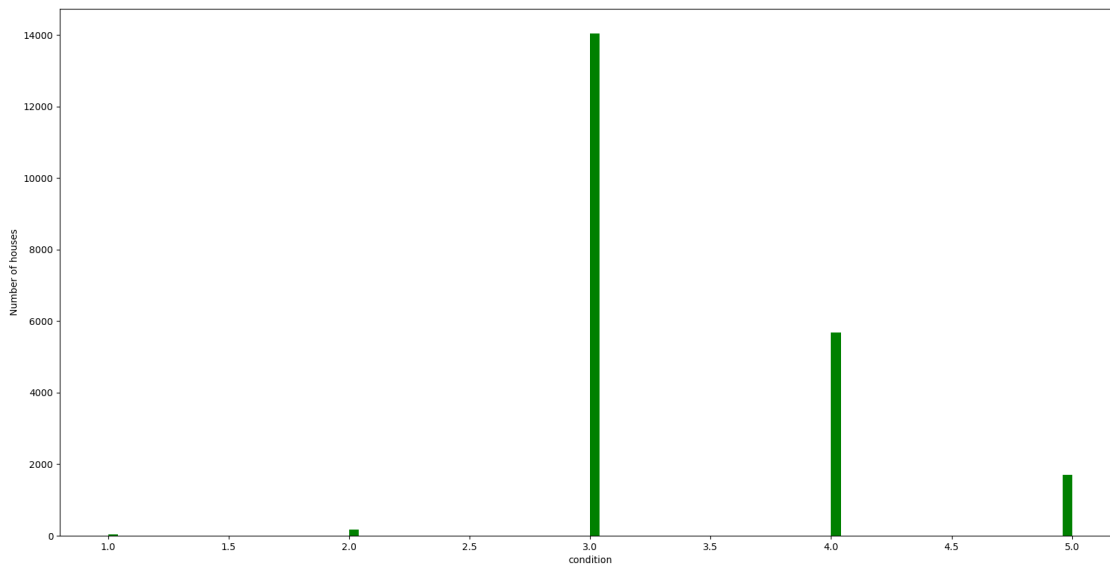
[1 rows x 21 columns]

Nombre de vue(s) de la maison (vue sur quelque chose).

0.3.9 Condition info

```
[169]: plt.figure(figsize=(20,10))
plt.hist(df['condition'], bins=100, color='green')
plt.xlabel('condition')
plt.ylabel('Number of houses')
```

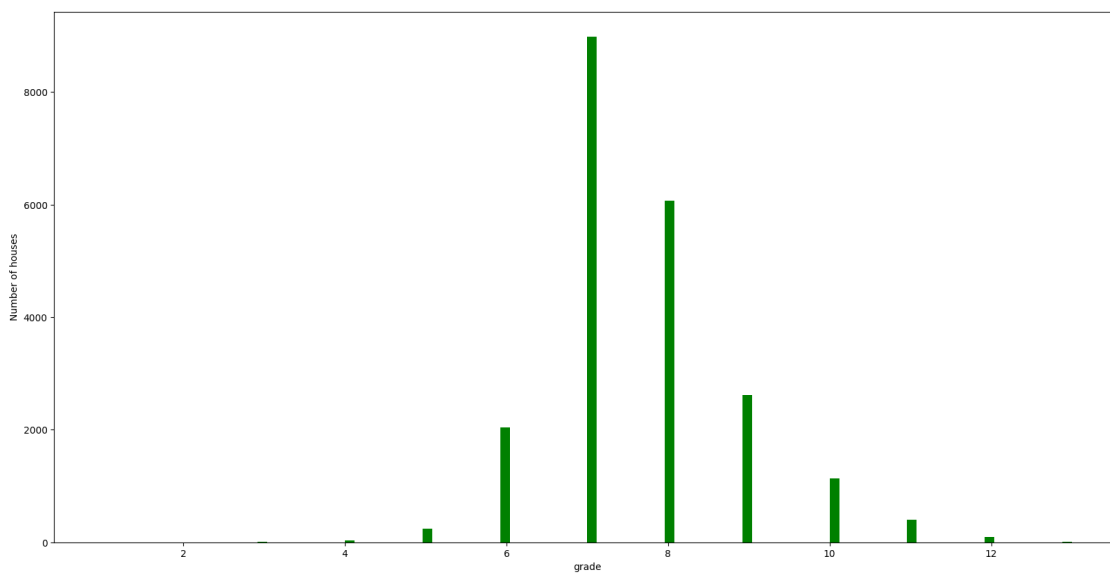
```
plt.show()
```



Les notes varient de 1 à 5. Indique la condition de la propriété (1 = mauvaise, 5 = excellente).

0.3.10 Grade info

```
[170]: plt.figure(figsize=(20,10))
plt.hist(df['grade'], bins=100, color='green')
plt.xlabel('grade')
plt.ylabel('Number of houses')
plt.show()
```



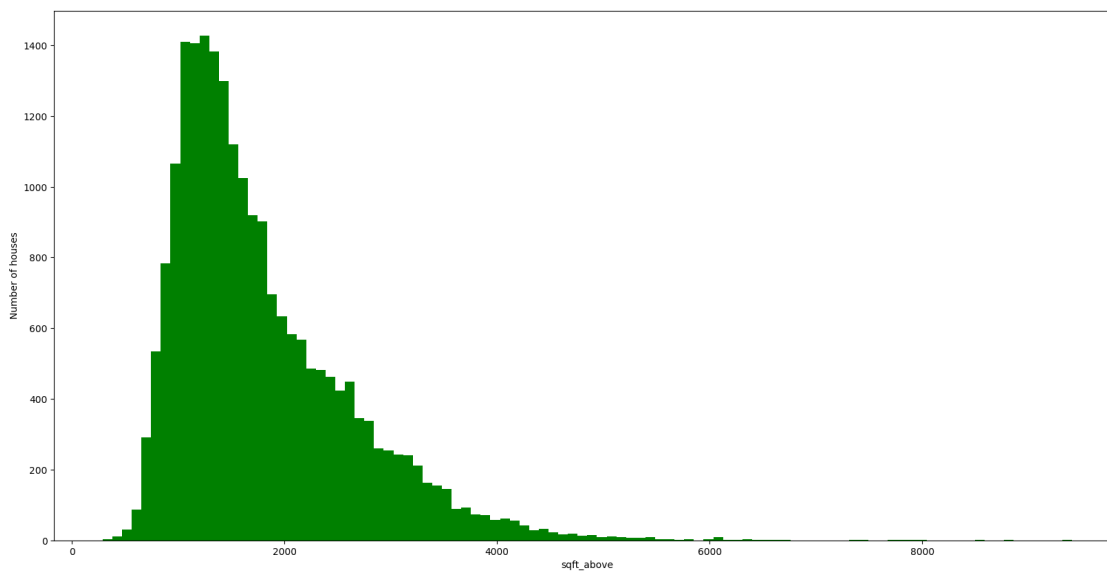
```
[171]: df['grade'].describe()
```

```
[171]: count      21613.000000  
      mean         7.656873  
      std         1.175459  
      min         1.000000  
      25%         7.000000  
      50%         7.000000  
      75%         8.000000  
      max        13.000000  
      Name: grade, dtype: float64
```

Les notes varient de 1 à 13. Elles représentent la qualité de la construction de la maison. On peut voir que la majorité des maisons ont une note comprise entre 7 et 8.

0.3.11 Sqft above info

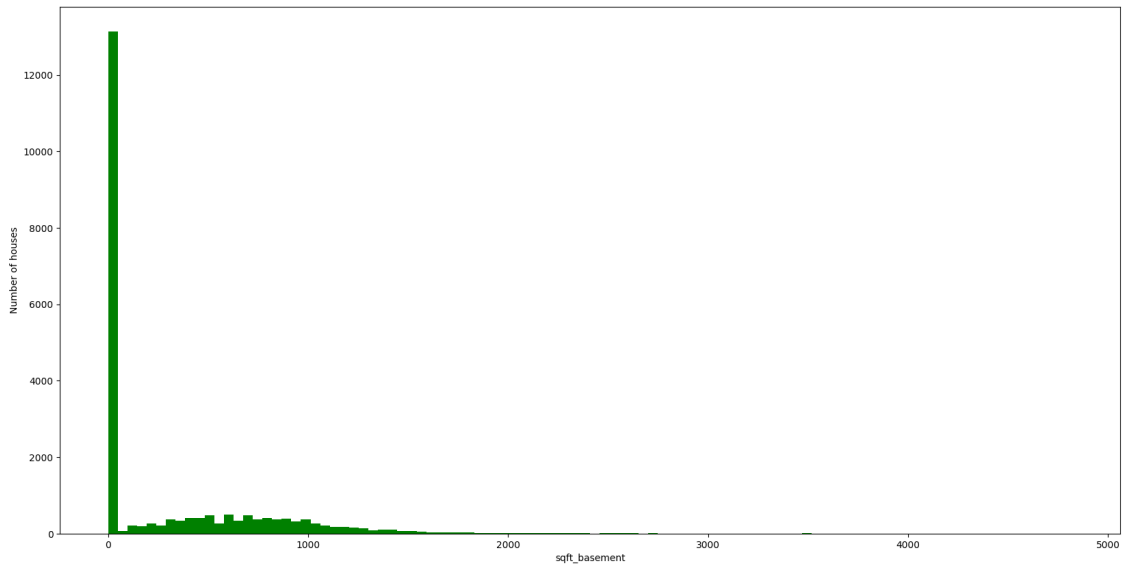
```
[172]: plt.figure(figsize=(20,10))  
      plt.hist(df['sqft_above'], bins=100, color='green')  
      plt.xlabel('sqft_above')  
      plt.ylabel('Number of houses')  
      plt.show()
```



Surface vivable au dessus du niveau du sol (i.e. exclu le sous-sol). La distribution est normale.

0.3.12 Sqft basement info

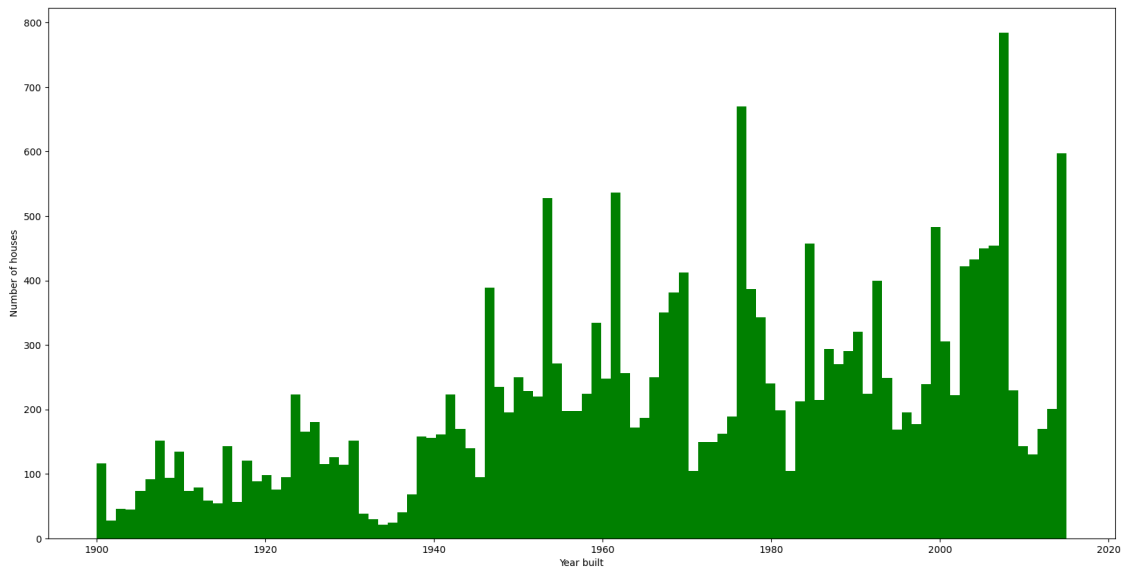
```
[173]: plt.figure(figsize=(20,10))
plt.hist(df['sqft_basement'], bins=100, color='green')
plt.xlabel('sqft_basement')
plt.ylabel('Number of houses')
plt.show()
```



Surface du sous-sol. La distribution est normale si on ne considère pas les maisons sans sous-sol.

0.3.13 Year built info

```
[174]: plt.figure(figsize=(20,10))
plt.hist(df['yr_built'], bins=100, color='green')
plt.xlabel('Year built')
plt.ylabel('Number of houses')
plt.show()
```

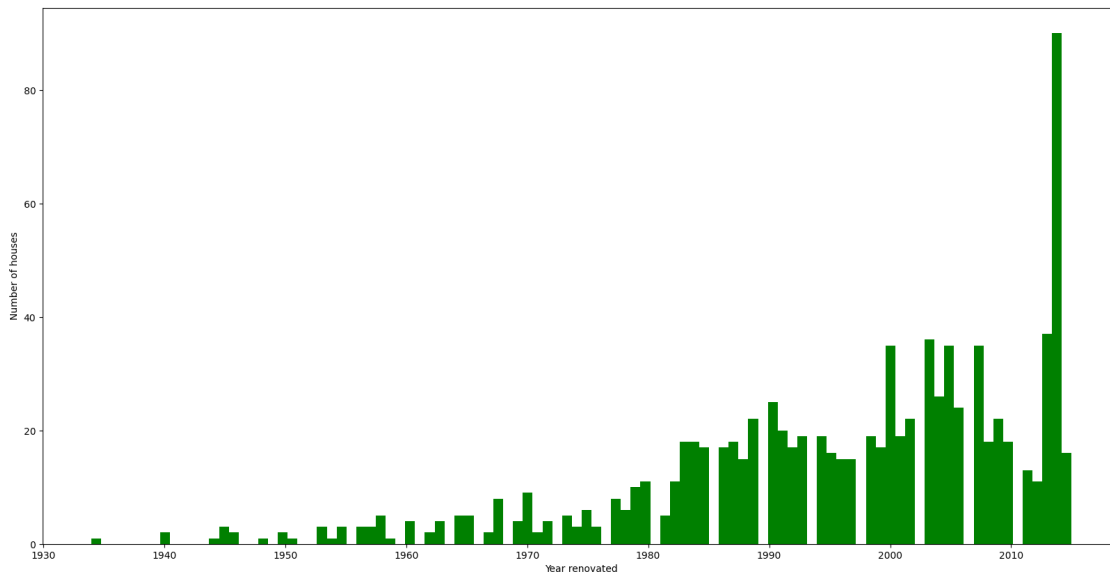
```
[175]: df['yr_built'].describe()
```

```
[175]: count    21613.000000
      mean      1971.005136
      std        29.373411
      min      1900.000000
      25%      1951.000000
      50%      1975.000000
      75%      1997.000000
      max      2015.000000
      Name: yr_built, dtype: float64
```

Les maisons sont toutes construites entre 1900 et 2015. La moitié des maisons ont été construites après 1975.

0.3.14 Year renovated info

```
[225]: renovated_houses = df[df['yr_renovated'] != 0]
      plt.figure(figsize=(20,10))
      plt.hist(renovated_houses["yr_renovated"], bins=100, color='green')
      plt.xlabel('Year renovated')
      plt.ylabel('Number of houses')
      plt.show()
```



```
[177]: # Uniquement les maisons qui ont été rénovées
renovated_houses['yr_renovated'].describe()
```

```
[177]: count      914.000000
      mean      1995.827133
      std       15.517107
      min      1934.000000
      25%      1987.000000
      50%      2000.000000
      75%      2007.000000
      max      2015.000000
      Name: yr_renovated, dtype: float64
```

```
[178]: # Toutes les maisons
df['yr_renovated'].describe()
```

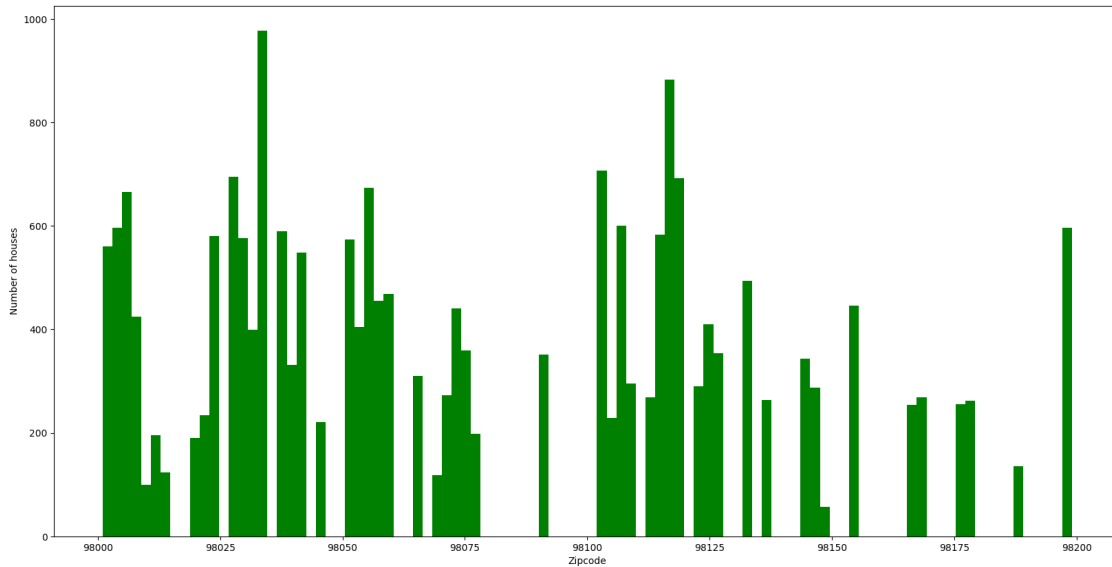
```
[178]: count      21613.000000
      mean        84.402258
      std       401.679240
      min         0.000000
      25%         0.000000
      50%         0.000000
      75%         0.000000
      max      2015.000000
      Name: yr_renovated, dtype: float64
```

Cette variable nous permet de savoir quand est-ce qu'une maison a été rennovée. Les maisons qui n'ont pas été rennovées ont une valeur de 0. On peut voir que la majorité des maisons n'ont pas

été rennovées. Pour celles qui ont été rennovées, elles l'ont été entre 1934 et 2015.

0.3.15 Zipcode info

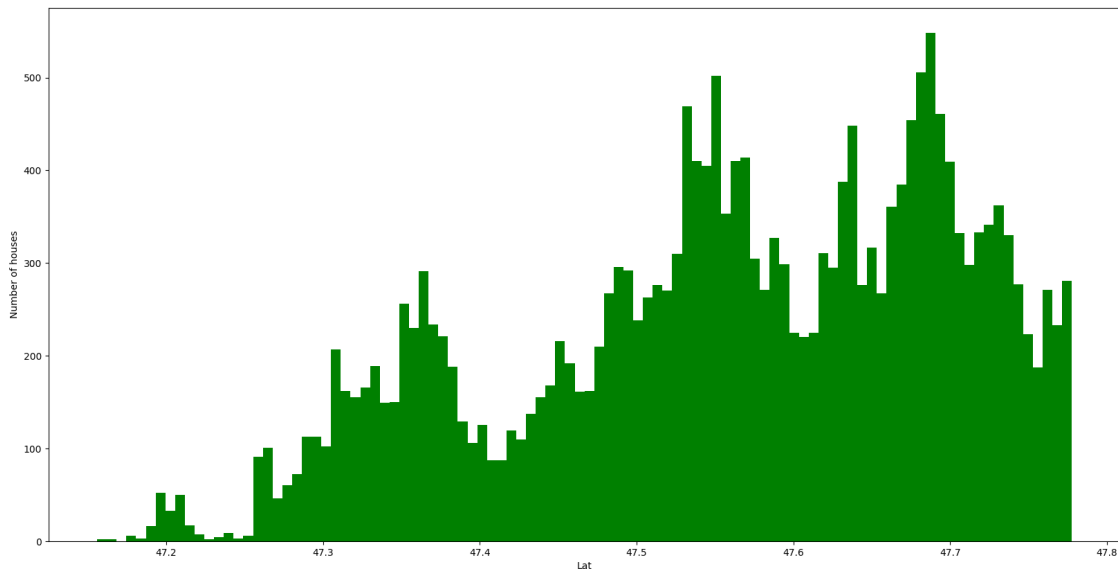
```
[179]: plt.figure(figsize=(20,10))
plt.hist(df['zipcode'], bins=100, color='green')
plt.xlabel('Zipcode')
plt.ylabel('Number of houses')
plt.show()
```



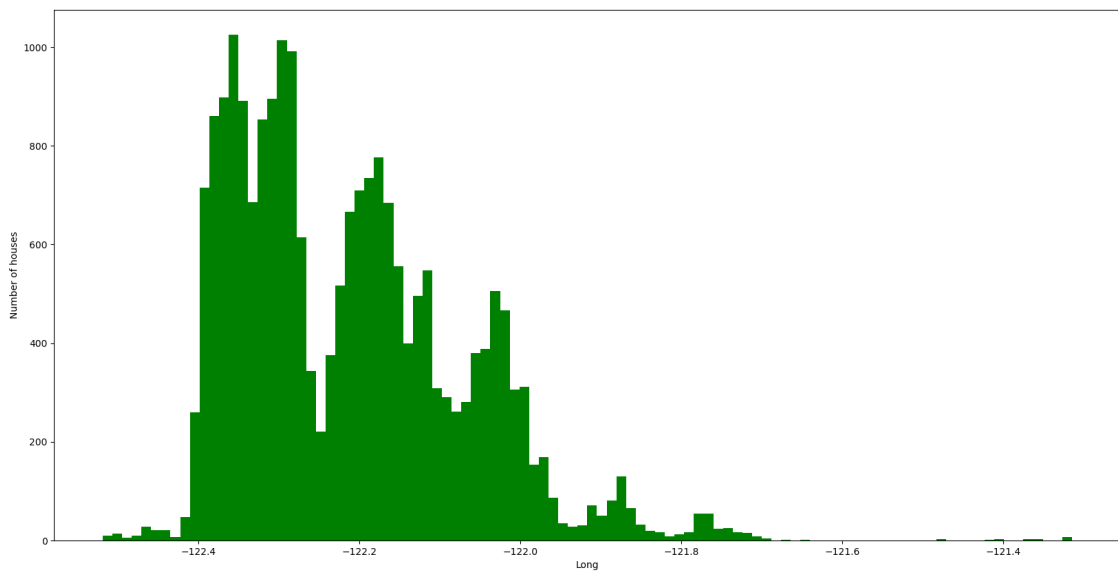
Rien de particulier à remarquer.

0.3.16 Latitude et longitude info

```
[180]: plt.figure(figsize=(20,10))
plt.hist(df['lat'], bins=100, color='green')
plt.xlabel('Lat')
plt.ylabel('Number of houses')
plt.show()
```



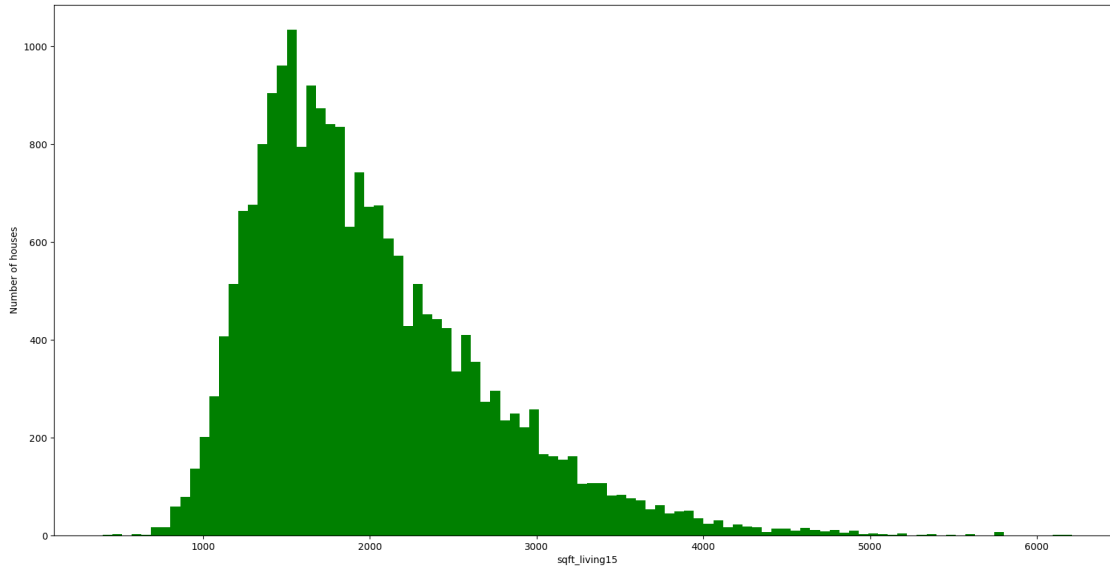
```
[181]: plt.figure(figsize=(20,10))
plt.hist(df['long'], bins=100, color='green')
plt.xlabel('Long')
plt.ylabel('Number of houses')
plt.show()
```



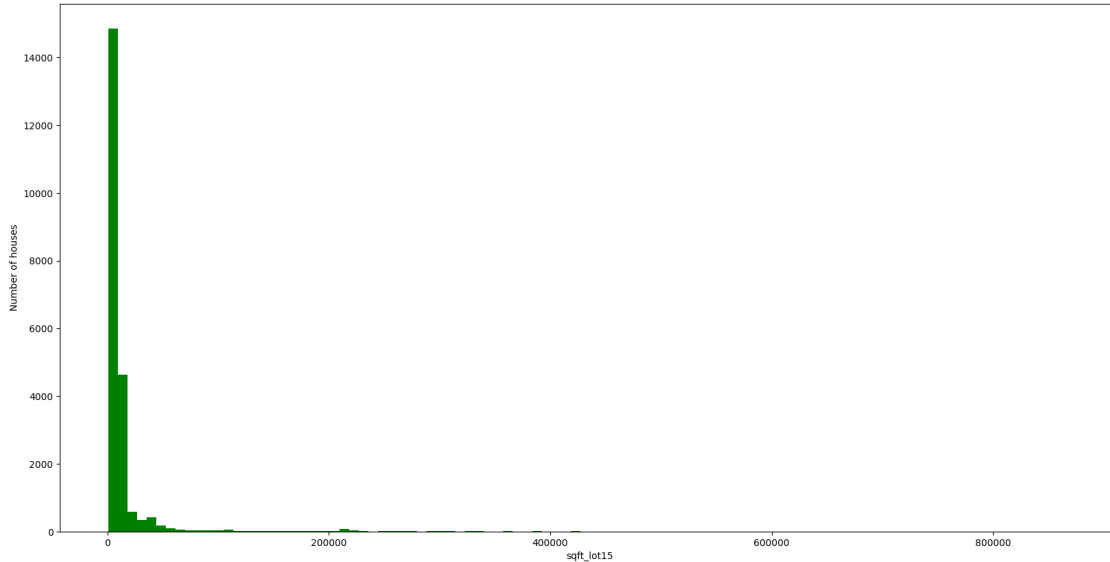
Rien de particulier à remarquer

0.3.17 Sqft living15 et Sqft lot15 info

```
[182]: plt.figure(figsize=(20,10))
plt.hist(df['sqft_living15'], bins=100, color='green')
plt.xlabel('sqft_living15')
plt.ylabel('Number of houses')
plt.show()
```



```
[183]: plt.figure(figsize=(20,10))
plt.hist(df['sqft_lot15'], bins=100, color='green')
plt.xlabel('sqft_lot15')
plt.ylabel('Number of houses')
plt.show()
```



La distribution de sqft_lot15 est exponentielle. On va donc prendre son logarithme. Celle de sqft_living15 est normale.

0.3.18 Taille de la maison et du terrain

SQDT Living 15 : Surface habitable moyenne des 15 voisins les plus proches. Maison qui sont 25% plus grande que la moyenne des 15 voisins les plus proches (3930 maisons) et donc avantagées :

```
[184]: # sqft_living15 : average size of 15 closest houses
# Houses that are 25% bigger than their neighbours in average
# Those houses have a comparable advantage on their nearest neighbours
df[df['sqft_living'] > 1.25*df['sqft_living15']]
```

```
[184]:
```

	id	date	price	bedrooms	bathrooms	\
1	6414100192	20141209T000000	538000.0	3	2.25	
3	2487200875	20141209T000000	604000.0	4	3.00	
10	1736800520	20150403T000000	662500.0	3	2.50	
14	1175000570	20150312T000000	530000.0	5	2.00	
15	9297300055	20150124T000000	650000.0	4	3.00	
...	
21593	8672200110	20150317T000000	1088000.0	5	3.75	
21597	191100405	20150421T000000	1575000.0	4	3.25	
21600	249000205	20141015T000000	1537000.0	5	3.75	
21606	7936000429	20150326T000000	1007500.0	4	3.50	
21609	6600060120	20150223T000000	400000.0	4	2.50	

	sqft_living	sqft_lot	floors	waterfront	view	...	grade	\
1	2570	7242	2.0	0	0	...	7	
3	1960	5000	1.0	0	0	...	7	

10	3560	9796	1.0	0	0	...	8
14	1810	4850	1.5	0	0	...	7
15	2950	5000	2.0	0	3	...	9
...
21593	4170	8142	2.0	0	2	...	10
21597	3410	10125	2.0	0	0	...	10
21600	4470	8088	2.0	0	0	...	11
21606	3510	7200	2.0	0	0	...	9
21609	2310	5813	2.0	0	0	...	8

	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	\
1	2170	400	1951	1991	98125	47.7210	
3	1050	910	1965	0	98136	47.5208	
10	1860	1700	1965	0	98007	47.6007	
14	1810	0	1900	0	98107	47.6700	
15	1980	970	1979	0	98126	47.5714	
...
21593	4170	0	2006	0	98056	47.5354	
21597	3410	0	2007	0	98040	47.5653	
21600	4470	0	2008	0	98004	47.6321	
21606	2600	910	2009	0	98136	47.5537	
21609	2310	0	2014	0	98146	47.5107	

	long	sqft_living15	sqft_lot15
1	-122.319	1690	7639
3	-122.393	1360	5000
10	-122.145	2210	8925
14	-122.394	1360	4850
15	-122.375	2140	4000
...
21593	-122.181	3030	7980
21597	-122.223	2290	10125
21600	-122.200	2780	8964
21606	-122.398	2050	6200
21609	-122.362	1830	7200

[3930 rows x 21 columns]

0.4 Nettoyage des données

```
[185]: # Formate date
df['date'] = df['date'].astype('datetime64[ns]')
df['date'] = pd.to_datetime(df['date'])
df['date']=df['date'].map(dt.datetime.toordinal)
```

On remarque certaines maisons sont plus grandes que les terrains sur lequel elles sont construites :

```
[186]: # Houses that are somehow bigger than the lot they are in..
df[df['sqft_living']/df['floors'] > df['sqft_lot']]
```

```
[186]:
```

	id	date	price	bedrooms	bathrooms	sqft_living	\
1549	8816400885	735514	450000.0	4	1.75	1640	
3452	2559950110	735710	1234570.0	2	2.50	2470	
5800	2770604103	735445	450000.0	3	2.50	1530	
13253	2877104196	735573	760000.0	3	2.00	1780	
13278	3277800845	735425	370000.0	3	1.00	1170	
15743	9828702895	735528	700000.0	4	1.75	2420	
16931	5016002275	735386	610000.0	5	2.50	3990	
17434	2062600020	735422	530000.0	2	2.50	1785	

	sqft_lot	floors	waterfront	view	...	grade	sqft_above	\
1549	1480	1.0	0	0	...	7	820	
3452	609	3.0	0	0	...	11	1910	
5800	762	2.0	0	0	...	8	1050	
13253	1750	1.0	0	2	...	8	1400	
13278	1105	1.0	0	0	...	7	1170	
15743	520	1.5	0	0	...	7	2420	
16931	3839	1.0	0	0	...	8	1990	
17434	779	2.0	0	0	...	7	1595	

	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	\
1549	820	1912	0	98105	47.6684	-122.314	
3452	560	2011	0	98112	47.6182	-122.312	
5800	480	2007	0	98119	47.6420	-122.374	
13253	380	1927	2014	98103	47.6797	-122.357	
13278	0	1965	0	98126	47.5448	-122.375	
15743	0	1900	0	98112	47.6209	-122.302	
16931	2000	1962	0	98112	47.6236	-122.299	
17434	190	1975	0	98004	47.5959	-122.198	

	sqft_living15	sqft_lot15
1549	1420	2342
3452	2440	1229
5800	1610	1482
13253	1780	3750
13278	1380	1399
15743	1200	1170
16931	2090	5000
17434	1780	794

[8 rows x 21 columns]

```
[187]: # Maison qui sont plus grandes que le terrain sur lequel elles sont construites
df = df.drop(df[df['sqft_living']/df['floors'] > df['sqft_lot']].index)
```



```
[188]: # Maison qui a 33 chambres pour une superficie d'environ 162m2 et 1 seul étage
df = df.drop(df[df['bedrooms'] == 33].index)
df = df.drop(df[df['bedrooms'] == 0].index)
df = df.drop(df[df['bathrooms'] == 0].index)
```

```
[189]: # Passage de la variable sqft_lot à son logarithme
df["sqft_lot"] = np.log(df['sqft_lot'])
```

```
[190]: # Passage de la variable sqft_lot15 à son logarithme
df["sqft_lot15"] = np.log(df['sqft_lot15'])
```

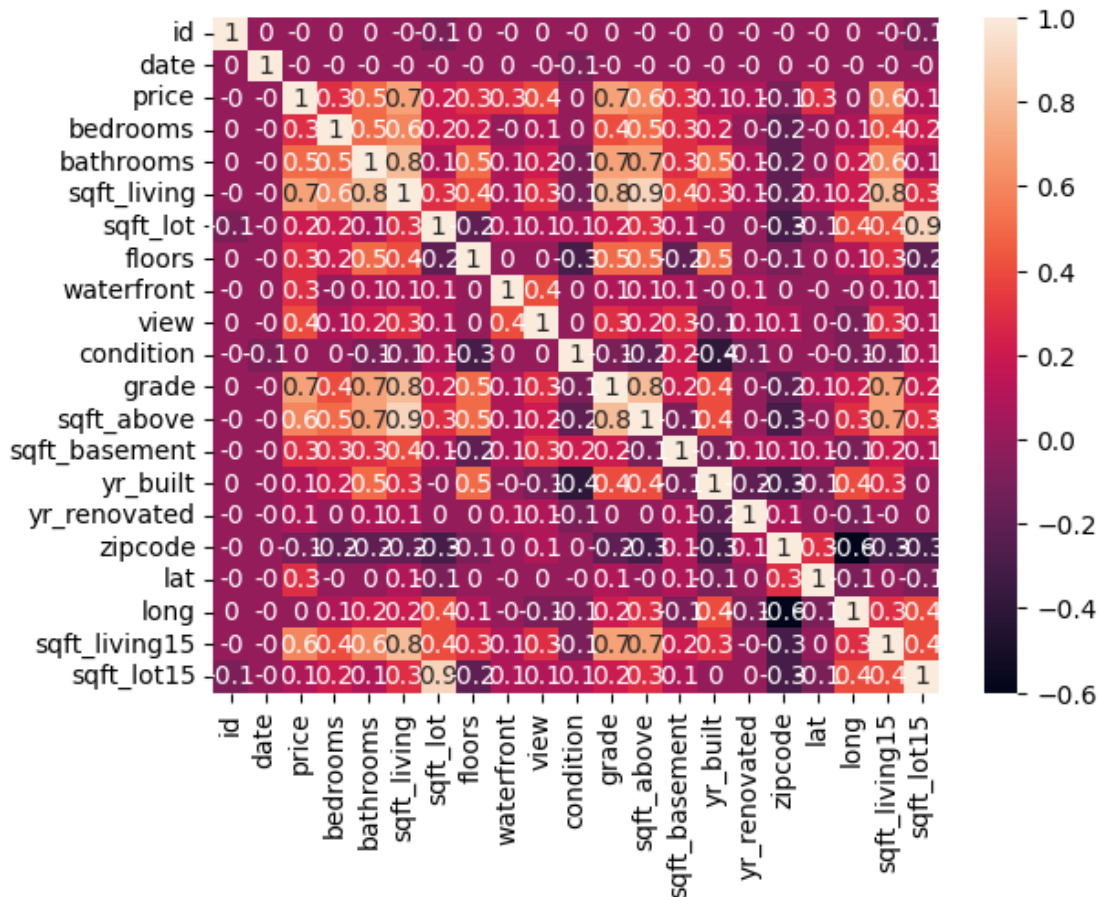
On constate qu'en prenant le logarithme de sqft_lot, l'ensemble des modèles sont plus performants.

0.5 Modèles prédictifs

0.5.1 Modèle simple de régression linéaire

```
[191]: #etude de la correlation
matrice_corr = df.corr().round(1)
sns.heatmap(data=matrice_corr, annot=True)
```

```
[191]: <Axes: >
```



Grâce à cette matrice de corrélation, on regarde la corrélation des différentes variables avec le prix. Il faut choisir les variables à retenir pour notre modèle en retenant celles qui sont le plus fort coefficient de corrélation. On constate que les coefficients vont de 0 à 0.7 (en valeurs absolues). On peut choisir de retenir les qui ont un coefficient supérieur ou égal à 0.4 en valeur absolue. On retient donc bathrooms, sqft_living, view, grade, sqft_above et sqft_living15.

Mise en place du modèle

```
[192]: #on utilise seulement les variables qui ont une corrélation avec le prix
X=pd.DataFrame(np.
    ↪c_[df['bathrooms'],df['sqft_living'],df['view'],df['grade'],df['sqft_above'],df['sqft_livin
    ↪columns =
    ↪['bathrooms','sqft_living','view','grade','sqft_above','sqft_living15'])
Y = df['price']

#base d'apprentissage et base de test

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,
    ↪random_state=5)
```

```
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(17270, 6)
(4318, 6)
(17270,)
(4318,)
```

Entrainement du modèle

```
[193]: #entrainement du modèle
```

```
lmodellineaire = LinearRegression()
lmodellineaire.fit(X_train, Y_train)
```

```
[193]: LinearRegression()
```

Evaluation du modèle

```
[194]: # Evaluation du training set
```

```
y_train_predict = lmodellineaire.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))

print("La performance du modèle sur la base d'apprentissage")
print('-----')
print("L'erreur quadratique moyenne est {}".format(rmse))
print('\n')

# model evaluation for testing set
y_test_predict = lmodellineaire.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))

print('La performance du modèle sur la base de test')
print('-----')
print("L'erreur quadratique moyenne est {}".format(rmse))
```

La performance du modèle sur la base d'apprentissage

L'erreur quadratique moyenne est 239994.52326066705

La performance du modèle sur la base de test

L'erreur quadratique moyenne est 235432.39753876044

Le modèle n'est pas trop mauvais. L'erreur quadratique moyenne est d'environ

240000, ce qui est de l'ordre du cart — type du prix des maisons (370000).

0.5.2 Modèle simple de régression linéaire avec toutes les variables

Mise en place du modèle

```
[195]: # On exclue le l'id de la maison qui n'a aucune influence sur le prix
X = df.loc[:, ~df.columns.isin(['id', 'price'])]
Y = df['price']

#base d'apprentissage et base de test

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,
↳ random_state=5)
```

Entraînement du modèle

```
[196]: #entraînement du modèle

lmodellineaire = LinearRegression()
lmodellineaire.fit(X_train, Y_train)
```

```
[196]: LinearRegression()
```

Evaluation du modèle

```
[197]: # Evaluation du training set

y_train_predict = lmodellineaire.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))

print("La performance du modèle sur la base d'apprentissage")
print('-----')
print("L'erreur quadratique moyenne est {}".format(rmse))
print('\n')

# model evaluation for testing set
y_test_predict = lmodellineaire.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))

print('La performance du modèle sur la base de test')
print('-----')
print("L'erreur quadratique moyenne est {}".format(rmse))
```

La performance du modèle sur la base d'apprentissage

L'erreur quadratique moyenne est 200279.08702527333

La performance du modèle sur la base de test

L'erreur quadratique moyenne est 197664.4612274257

L'erreur quadratique moyenne est ici plus faible (200 000\$). Le modèle est meilleur. Il peut cependant être intéressant de pousser l'analyse en utilisant un modèle avec régularisation.

0.5.3 Ridge Regression

Mise en place du modèle

```
[198]: X = df.loc[:, ~df.columns.isin(['id', 'price'])]
Y = df['price']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,
↳ random_state=5)
```

Recherche du meilleur hyperparameter alpha

```
[199]: ridge=Ridge()
parameters = {'alpha': (np.logspace(-8, 8, 100))}
#parameters={'alpha': [1e-15, 1e-10, 1e-8, 1e-3, 1e-2, 1e-1, 0.5, 1, 1.
↳ 5, 2, 5, 10, 20, 30, 35, 40, 45, 50, 55, 100]}
ridge_regressor=GridSearchCV(ridge, parameters, scoring='neg_mean_squared_error', cv=10)
ridge_regressor.fit(X_train, Y_train)

alpha_ridge = ridge_regressor.best_params_["alpha"]
print('alpha : {}'.format(alpha))
```

alpha : 1e-08

Entraînement du modèle

```
[200]: ridgereg = make_pipeline(StandardScaler(), Ridge(alpha=alpha_ridge))
ridgereg.fit(X_train, Y_train)
```

```
[200]: Pipeline(steps=[('standardscaler', StandardScaler()),
↳ ('ridge', Ridge(alpha=0.39442060594376643))])
```

Evaluation du modèle

```
[201]: # Evaluation du training set

Y_train_pred = ridgereg.predict(X_train)
Y_test_pred = ridgereg.predict(X_test)
#print('MSE train: %.3f, test: %.3f' % (mean_squared_error(Y_train,
↳ Y_train_pred), mean_squared_error(Y_test, Y_test_pred)))

Y_train_predict = ridgereg.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, Y_train_predict)))
```

```

print("La performance du modèle sur la base d'apprentissage")
print('-----')
print("L'erreur quadratique moyenne est {}".format(rmse))
print('\n')

# model evaluation for testing set
Y_test_predict = ridgereg.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, Y_test_predict)))

print('La performance du modèle sur la base de test')
print('-----')
print("L'erreur quadratique moyenne est {}".format(rmse))

```

La performance du modèle sur la base d'apprentissage

 L'erreur quadratique moyenne est 200279.08711811327

La performance du modèle sur la base de test

 L'erreur quadratique moyenne est 197664.57293403274

Les performances de la Ridge Regression sont très similaires à celles de la régression linéaire simple.
 Nous allons donc explorer la régression Lasso.

0.5.4 Lasso

0.5.5 Mise en place du modèle

```

[202]: X = df.loc[:, ~df.columns.isin(['id', 'price'])]
Y = df['price']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,
↳ random_state=5)

```

Recherche du meilleur hyperparameter alpha

```

[203]: # Attention : la recherche est longue (compter 10-15 minutes)
lasso=Lasso(normalize=True)
parameters = {'alpha': (np.logspace(-8, 8, 100))}
lasso_regressor=GridSearchCV(lasso,parameters,cv=3)
lasso_regressor.fit(X_train,Y_train)

lasso_alpha = lasso_regressor.best_params_["alpha"]

print(lasso_regressor.best_params_)
print(lasso_regressor.best_score_)

```

```
{'alpha': 1e-08}
0.7000250726816066
```

Entraînement du modèle

```
[204]: lasso = Lasso(alpha = lasso_alpha, normalize = True)
lasso.fit(X_train, Y_train)
```

```
[204]: Lasso(alpha=1e-08, normalize=True)
```

Evaluation du modèle

```
[205]: print(lasso.score(X_test, Y_test))
```

```
0.7134943916492347
```

```
[206]: # Evaluation du training set

Y_train_pred = lasso.predict(X_train)
Y_test_pred = lasso.predict(X_test)
#print('MSE train: %.3f, test: %.3f' % (mean_squared_error(Y_train,
↪Y_train_pred), mean_squared_error(Y_test, Y_test_pred)))

Y_train_predict = lasso.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, Y_train_predict)))

print("La performance du modèle sur la base d'apprentissage")
print('-----')
print("L'erreur quadratique moyenne est {}".format(rmse))
print('\n')

# model evaluation for testing set
Y_test_predict = lasso.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, Y_test_predict)))

print('La performance du modèle sur la base de test')
print('-----')
print("L'erreur quadratique moyenne est {}".format(rmse))
```

```
La performance du modèle sur la base d'apprentissage
-----
L'erreur quadratique moyenne est 200279.08702527327
```

```
La performance du modèle sur la base de test
-----
L'erreur quadratique moyenne est 197664.46122704283
```

De même, les résultats ne sont pas plus convaincants que ceux de la régression linéaire simple. Nous allons donc explorer la régression non linéaire.

0.5.6 Random Forest

Mis en place du modèle

```
[207]: X = df.loc[:, ~df.columns.isin(['id', 'price'])]
      Y = df['price']

      X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,
      ↪random_state=5)
```

Entraînement du modèle

```
[208]: RFRegressor = RandomForestRegressor(n_estimators = 100, random_state = 0)
      # n_estimators = nombre d'arbres de décision, c'est-à-dire le nombre de modèles
      ↪que l'on va construire
      RFRegressor.fit(X_train, Y_train)
```

```
[208]: RandomForestRegressor(random_state=0)
```

Evaluation du modèle

```
[209]: # Evaluation du training set

      Y_train_pred = RFRegressor.predict(X_train)
      Y_test_pred = RFRegressor.predict(X_test)
      #print('MSE train: %.3f, test: %.3f' % (mean_squared_error(Y_train,
      ↪Y_train_pred), mean_squared_error(Y_test, Y_test_pred)))

      Y_train_predict = RFRegressor.predict(X_train)
      rmse = (np.sqrt(mean_squared_error(Y_train, Y_train_predict)))

      print("La performance du modèle sur la base d'apprentissage")
      print('-----')
      print("L'erreur quadratique moyenne est {}".format(rmse))
      print('\n')

      # model evaluation for testing set
      Y_test_predict = RFRegressor.predict(X_test)
      rmse = (np.sqrt(mean_squared_error(Y_test, Y_test_predict)))

      print('La performance du modèle sur la base de test')
      print('-----')
      print("L'erreur quadratique moyenne est {}".format(rmse))
```

La performance du modèle sur la base d'apprentissage

L'erreur quadratique moyenne est 48683.80612073758

La performance du modèle sur la base de test

L'erreur quadratique moyenne est 134276.95802832497

On a un bien meilleur résultat avec le Random Forest. L'erreur quadratique moyenne est de 120 000\$ sur le dataset de test. C'est donc un jeu de donnée qui est mieux prédit par un modèle non linéaire.

0.5.7 Kernel Ridge Regression

Mise en place du modèle

```
[217]: X = df.loc[:, ~df.columns.isin(['id', 'price'])]
Y = df['price']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
    ↪ random_state=42)

# On normalise les données
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Entraînement du modèle

```
[218]: from sklearn.kernel_ridge import KernelRidge

# Création en entraînement du modèle
KRregressor = KernelRidge(kernel='rbf', alpha=0.1, gamma=0.1)
KRregressor.fit(X_train, Y_train)
```

```
[218]: KernelRidge(alpha=0.1, gamma=0.1, kernel='rbf')
```

Evaluation du modèle

```
[219]: Y_train_pred = KRregressor.predict(X_train)
Y_test_pred = KRregressor.predict(X_test)

Y_train_predict = KRregressor.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, Y_train_predict)))

print("La performance du modèle sur la base d'apprentissage")
print('-----')
print("L'erreur quadratique moyenne est {}".format(rmse))
print('\n')

# model evaluation for testing set
Y_test_predict = KRregressor.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, Y_test_predict)))

print('La performance du modèle sur la base de test')
print('-----')
```

```
print("L'erreur quadratique moyenne est {}".format(rmse))
```

La performance du modèle sur la base d'apprentissage

L'erreur quadratique moyenne est 65552.21319669549

La performance du modèle sur la base de test

L'erreur quadratique moyenne est 149889.87533226088

Dans l'état actuel, le modèle est moins bon que la régression linéaire simple, ce qui est étonnant. Il faudrait donc modifier les hyperparamètres pour obtenir un meilleur résultat. Cependant, les hyperparamètres présentés sont ceux qui donnent les meilleurs résultats pour un temps de calcul raisonnable.

Le modèle semble meilleur que la régression linéaire simple. Cependant, il est plus long à calculer et semble moins bon qu'une Random Forest. Nous essayons d'utiliser l'approximation de Nyström pour accélérer le calcul.

0.5.8 Approximation de nyström for kernel ridge regression

Mise en place de l'approximation

```
[220]: from sklearn.kernel_approximation import Nystroem

# Création d'un objet Nystroem avec n_components = 100
nystroem = Nystroem(kernel='rbf', n_components=100)

# On utilise la variable regressor de la partie précédente

X_train_nystroem = nystroem.fit_transform(X_train)

# Entraînement du modèle KernelRidge sur les données transformées par Nystroem
KRregressor_nys = KernelRidge(kernel='rbf', alpha=0.1, gamma=0.1)
KRregressor_nys.fit(X_train_nystroem, Y_train)

# Prédiction sur les données d'entraînement
Y_train_pred = KRregressor_nys.predict(X_train_nystroem)

# Calcul de l'erreur quadratique moyenne sur les données d'entraînement
rmse_train = np.sqrt(mean_squared_error(Y_train, Y_train_pred))

# Transformation des données de test avec la méthode de Nystroem
X_test_nystroem = nystroem.transform(X_test)

# Prédiction sur les données de test
Y_test_pred = KRregressor_nys.predict(X_test_nystroem)
```

```

# Calcul de l'erreur quadratique moyenne sur les données de test
rmse_test = np.sqrt(mean_squared_error(Y_test, Y_test_pred))

# Affichage des résultats
print("Performance du modèle sur la base d'apprentissage :")
print("Erreur quadratique moyenne : {:.2f}".format(rmse_train))
print('\n')
print("Performance du modèle sur la base de test :")
print("Erreur quadratique moyenne : {:.2f}".format(rmse_test))

```

Performance du modèle sur la base d'apprentissage :
 Erreur quadratique moyenne : 218868.57

Performance du modèle sur la base de test :
 Erreur quadratique moyenne : 187413.39

Les résultats sont moins bons avec l'approximation, ce qui est attendu et le temps de calcul est même plus grand ce qui est étonnant (on regarde le temps d'exécution des cellules de code).

0.6 Comparaison des modèles

Nous allons maintenant nous intéresser à une comparaison détaillée des modèles.

Premièrement, il est intéressant d'observer que pour les modèles de régression linéaire, introduire de la régularisation n'améliore pas la prédiction en terme d'erreur quadratique moyenne, là où pour les modèles non linéaire Random Forest et Kernel Ridge, la prédiction est améliorée. Cela peut vouloir dire que : - Le jeu de données est assez propre et peu bruité - La forme intrinsèque d'un modèle linéaire ne permet pas de faire mieux que ce qui est obtenu et introduire de la régularisation n'améliore donc pas la prédiction

Choix du nombre de pli : Nous avons utilisé la source : <https://machinelearningmastery.com/k-fold-cross-validation/> Nous avons choisi 10 plis car cette valeur est montrée efficace (empiriquement) et conseillée "par défaut". Notre jeu de données est assez grand pour que cette valeur ait du sens, il y a assez de données dans les 9 plis pour entrainer notre modèle.

```

[221]: X = df.loc[:, ~df.columns.isin(['id', 'price'])]
        Y = df['price']

# Ensemble des erreurs pour les différents modèles
RF_errors = []
lasso_errors = []
ridge_errors = []
KRR_errors = []

# Mise en place des plis
kfold = KFold(n_splits=10, shuffle=True, random_state=7)
for train_index, test_index in kfold.split(X, Y):
    X_train_CV = [X.iloc[i] for i in train_index]

```

```

X_test_CV = [X.iloc[i] for i in test_index]
Y_train_CV = [Y.iloc[i] for i in train_index]
Y_test_CV = [Y.iloc[i] for i in test_index]

RFregressor.fit(X_train_CV, Y_train_CV)
Y_test_pred_RF = RFregressor.predict(X_test_CV)

lassoreg.fit(X_train_CV, Y_train_CV)
Y_test_pred_lasso = lassoreg.predict(X_test_CV)

ridgereg.fit(X_train_CV, Y_train_CV)
Y_test_pred_ridge = ridgereg.predict(X_test_CV)

KRregressor.fit(X_train_CV, Y_train_CV)
Y_test_pred_KRR = KRregressor.predict(X_test_CV)

for i in range(len(Y_test_CV)):
    Y_test = Y_test_CV[i]
    RF_errors.append(Y_test_pred_RF[i]-Y_test)
    lasso_errors.append(Y_test_pred_lasso[i]-Y_test)
    ridge_errors.append(Y_test_pred_ridge[i]-Y_test)
    KRR_errors.append(Y_test_pred_KRR[i]-Y_test)

```

```

[222]: RF_errors = np.array(RF_errors)
lasso_errors = np.array(lasso_errors)
ridge_errors = np.array(ridge_errors)
KRR_errors = np.array(KRR_errors)

# summary of errors arrays
print('RF_errors: mean=%.3f stdv=%.3f' % (np.mean(RF_errors), np.
    ↪std(RF_errors)))
print('lasso_errors: mean=%.3f stdv=%.3f' % (np.mean(lasso_errors), np.
    ↪std(lasso_errors)))
print('ridge_errors: mean=%.3f stdv=%.3f' % (np.mean(ridge_errors), np.
    ↪std(ridge_errors)))
print('KRR_errors: mean=%.3f stdv=%.3f' % (np.mean(KRR_errors), np.
    ↪std(KRR_errors)))

```

```

RF_errors: mean=-484.946 stdv=127218.182
lasso_errors: mean=-1.864 stdv=200231.092
ridge_errors: mean=-1.863 stdv=200231.067
KRR_errors: mean=-539436.173 stdv=367578.917

```

```

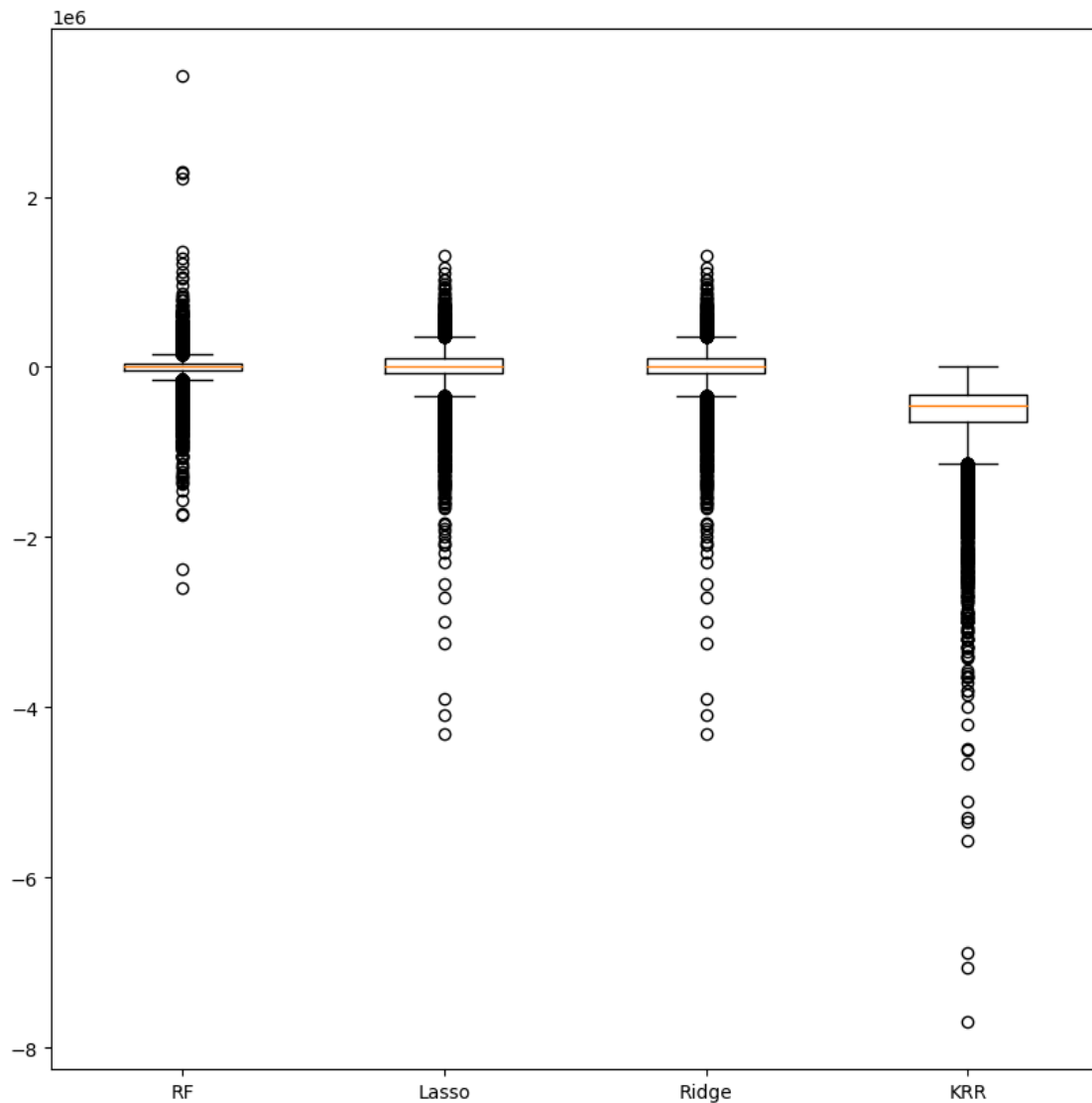
[223]: fig = plt.figure(figsize =(10, 10))

boxplot_dict = {'RF':RF_errors, 'Lasso':lasso_errors, 'Ridge':ridge_errors,
    ↪'KRR':KRR_errors}

```

```
# Creating plot
plt.boxplot(boxplot_dict.values(), labels = boxplot_dict.keys())

# show plot
plt.show()
```



Premièrement, on constate que les régressions ridge et lasso ont des résultats égaux en terme de performance de prédiction. De plus, on constate que RandomForest a une erreur systématique en moyenne plus grande par rapport à ridge et lasso (-600 contre -20) mais c'est tout à fait négligeable considérant les ordres de grandeur du problème. A l'inverse, on constate que l'écart-type des erreurs est plus faible pour RandomForest que pour ridge et lasso (donc la variance également). Cependant, considérant les ordres de grandeurs du problème, un écart-type de 200 000€ pour ridge et lasso reste convenable, mais RandomForest est encore meilleur. La Kernel Ridge regression quant à elle

semblait meilleur que les régressions linéaires Ridge et Lasso sur son jeu d'entraînement mais avec la validation croisée, on se rend compte que ce modèle commet une grande erreur systématique et a un écart-type bien plus grand. Ce modèle n'est donc pas à garder. Finalement, le modèle le plus intéressant à utiliser est donc RandomForest.