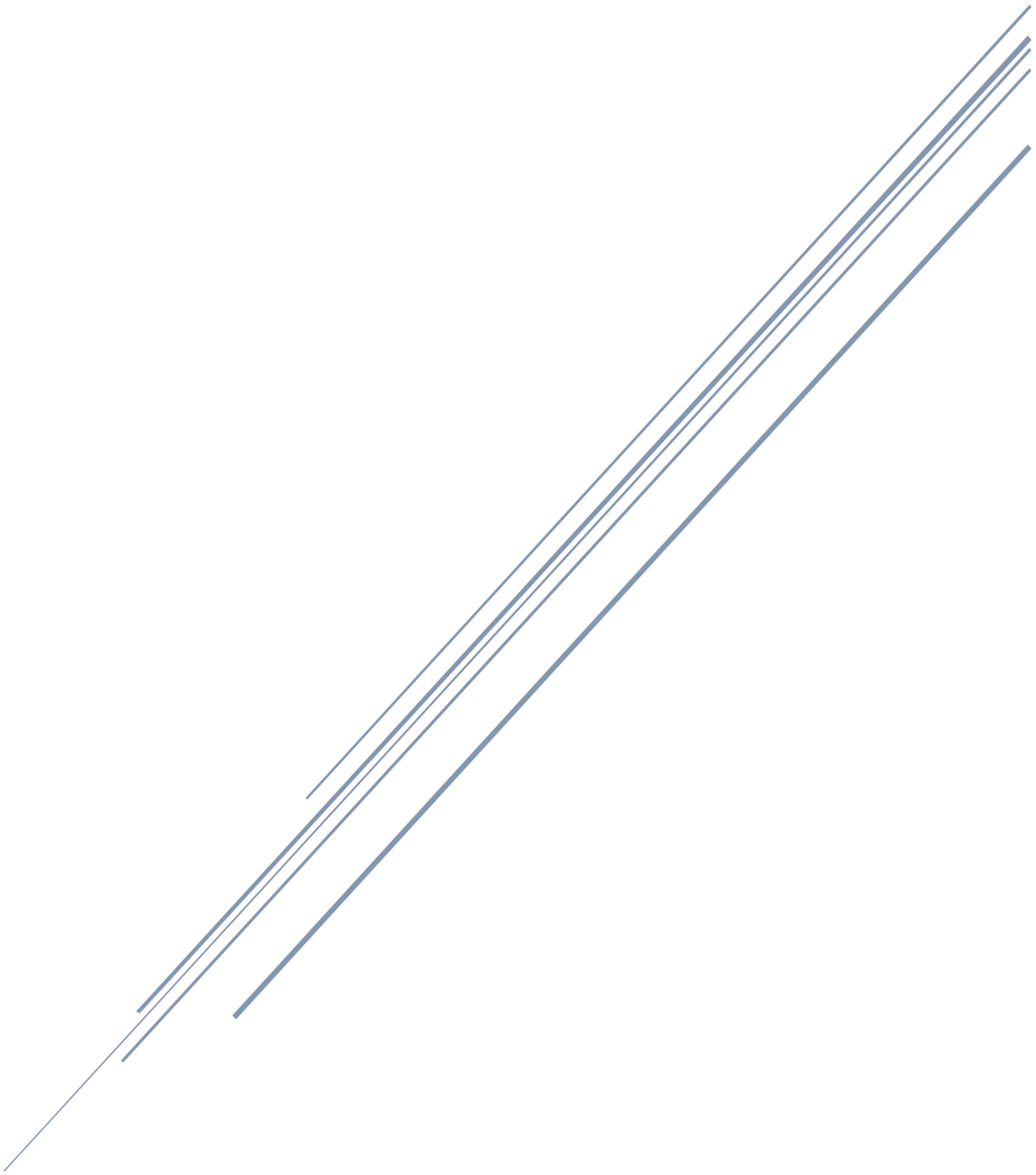
DATA MINING HOMEWORK REPORT

TAHSIN ILKHAS ZADEH STD.NO: 400422034

1. New York City Airbnb Open Data analysis
2. Apartment Rental Offers In Germany Data analysis



Shahid Beheshti University, Department of computer science

Data Mining Course: Dr.Farahani, Dr.Parand

Teacher Assistant: Ali Sharifi

April 6, 2022

1. New York City Airbnb Open Data analysis

What we did in this homework:

1. Processing data
    * Delete duplicates
    * Manage missing data (Data cleaning)
        - Clean feature column if it has more than 50% Null values
        - Fill Null values of numerical data by mean value of feature
        - Fill Null values of categorical data by most frequents
        - Fill empty dates
    * Data normalization and removing the outliers
    * Removing columns that doesn't have useful info
    * Removing columns that doesn't have high correlation to data
    * Reduce features by grouping
2. Create virtualization to have a better understanding of the data
3. Provide general information in aggregate mode about ads such as number of ads, number of Ads in each geographical area, review of general price indices and ...
4. If the number of comments for an Ad can be considered an indicator of the number of customers finding the owners of the ad who have the most customers and investigate the causes
5. Perform arbitrary hypothesis tests on data and respond to and interpret them
6. Build a model to predict parameters such as price and presentation of these models and their interpretation.
7. Data visualization
8. Time calculation for each part

Data overview

Data Overview

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
t1 = time.time()
pd.set_option('display.max_columns',None)  #to show all columns
airbnb=pd.read_csv(root+"AB_NYC_2019.csv",parse_dates=['last_review'])  #reading dataset
airbnb.head(3)
```

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_revi
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.97237	Private room	149	1	
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225	1	
2	3647	THE VILLAGE OF HARLEM....NEW	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.94190	Private room	150	3	

✓ 0s completed at 6:32 PM

The dataset name: New York City Airbnb Open Data, AB_NYC_2019

Number of features in dataset: 16

Number of records in dataset: 48895

```
airbnb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              48895 non-null  int64
 1   name                            48879 non-null  object
 2   host_id                         48895 non-null  int64
 3   host_name                       48874 non-null  object
 4   neighbourhood_group             48895 non-null  object
 5   neighbourhood                   48895 non-null  object
 6   latitude                        48895 non-null  float64
 7   longitude                       48895 non-null  float64
 8   room_type                       48895 non-null  object
 9   price                           48895 non-null  int64
 10  minimum_nights                  48895 non-null  int64
 11  number_of_reviews               48895 non-null  int64
 12  last_review                     38843 non-null  datetime64[ns]
 13  reviews_per_month               38843 non-null  float64
 14  calculated_host_listings_count  48895 non-null  int64
 15  availability_365                48895 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(7), object(5)
```

```
s_time = time.time()
print("list of features and their types:")
print(airbnb.dtypes)
print(f'run time : {time.time() - s_time}')
```

```
list of features and their types:
id                                 object
name                               object
host_id                            object
host_name                          object
neighbourhood_group                object
neighbourhood                      object
latitude                          float64
longitude                         float64
room_type                          object
price                               int64
minimum_nights                      int64
number_of_reviews                   int64
last_review                datetime64[ns]
reviews_per_month                 float64
calculated_host_listings_count      int64
availability_365                    int64
x                                 float64
y                                 float64
z                                 float64
dtype: object
run time : 0.0037946701049804688
```

Plotting data on map to have better view

```
[5] BBox = ((airbnb.longitude.min(),  airbnb.longitude.max(), airbnb.latitude.min(), airbnb.latitude.max() ))
```

```
fig, ax = plt.subplots(figsize = (8,7))
#ruh_m = plt.imread('geo:40.7015,-74.4859?z=9')
ax.scatter(airbnb.longitude, airbnb.latitude,zorder=1, alpha= 0.2, c='b', s=10)
ax.set_title('Plotting Data places on NY Map')
ax.set_xlim(BBox[0],BBox[1])
ax.set_ylim(BBox[2],BBox[3])
#ax.imshow(X,rhum='NY.png', zorder=0, extent = BBox, aspect= 'equal')
```

```
(40.49979, 40.91306)
```


```
[7] s_time = time.time()
    airbnb['x'] = np.cos(airbnb['latitude']) * np.cos(airbnb['longitude'])
    airbnb['y'] = np.cos(airbnb['latitude']) * np.sin(airbnb['longitude'])
    airbnb['z'] = np.sin(airbnb['latitude'])
    print(f'run time : {time.time() - s_time}')

    run time : 0.01864051818847656
```

Processing data

Preprocessing data

```
s_time = time.time()
print(airbnb.count())
print(f'run time : {time.time() - s_time}')
```

```
id                                48895
name                              48879
host_id                           48895
host_name                         48874
neighbourhood_group               48895
neighbourhood                     48895
latitude                          48895
longitude                         48895
room_type                         48895
price                             48895
minimum_nights                    48895
number_of_reviews                 48895
last_review                       38843
reviews_per_month                 38843
calculated_host_listings_count    48895
availability_365                  48895
x                                 48895
y                                 48895
z                                 48895
dtype: int64
run time : 0.02509760856628418
```

Count missing data

Checking the missing data

As we can see, the number of records are not the same, so it's clear that we have some null data that should be managed:

```
s_time = time.time()
print("number of null values in each column:")
print(airbnb.isna().sum() ) #null count for each feature
print(f'run time : {time.time() - s_time}')
```

```
number of null values in each column:
id                                0
name                             16
host_id                           0
host_name                        21
neighbourhood_group               0
neighbourhood                     0
latitude                          0
longitude                         0
room_type                         0
price                             0
minimum_nights                    0
number_of_reviews                 0
last_review                   10052
reviews_per_month             10052
calculated_host_listings_count    0
availability_365                  0
x                                 0
y                                 0
z                                 0
dtype: int64
run time : 0.02463984489440918
```

✓ 0s completed at 6:32 PM

Drop duplicates ,drop columns with 50% null values and drop columns which have no useful info

Delete duplicates

```
s_time = time.time()
airbnb=airbnb.drop_duplicates()  #delete duplicate records
print(f'run time : {time.time() - s_time}')
```

```
run time : 0.0800178050994873
```

Delete culomns with more than %50 null values (if such columns exist):

```
[16] s_time = time.time()
     print('\nfeatures:\t\tpercent of null values:')
     print(airbnb.isna().sum()/len(airbnb))
     print(f'run time : {time.time() - s_time}')
```

```
features:               percent of null values:
id                                0.000000
name                              0.000327
host_id                           0.000000
host_name                         0.000429
neighbourhood_group               0.000000
neighbourhood                     0.000000
latitude                          0.000000
longitude                         0.000000
room_type                         0.000000
price                             0.000000
minimum_nights                    0.000000
number_of_reviews                 0.000000
```

```
s_time = time.time()
nullcolumns=airbnb.columns[((airbnb.isna().sum()/len(airbnb)) > 0.50)] #mask
print(f'run time : {time.time() - s_time}')
```

```
run time : 0.028334617614746094
```

```
[18] s_time = time.time()
     airbnb=airbnb.drop(nullcolumns,axis=1)
     airbnb.shape
     print(f'run time : {time.time() - s_time}')
```

```
run time : 0.010929107666015625
```

Delete culomns without usefull information(if such columns exist):

```
[19] s_time = time.time()
     airbnb=airbnb.drop(columns=['id'])
     print(airbnb.shape)
     print(f'run time : {time.time() - s_time}')
```

```
(48895, 18)
run time : 0.016750574111938477
```

we can see that number of columns have reduced

Fill numeric nulls by mean()

Filling NaN numeric datas with mean of each column

```
[20] s_time = time.time()
     print('\nfeatures: \t\t\t mean values:')
     print(airbnb._get_numeric_data().mean())
     print(f'run time : {time.time() - s_time}')
```

```
features:                       mean values:
latitude                          40.728949
longitude                        -73.952170
price                            152.720687
minimum_nights                     7.029962
number_of_reviews                 23.274466
reviews_per_month                  1.373221
calculated_host_listings_count     7.143982
availability_365                 112.781327
x                                 -0.123336
y                                 -0.983530
z                                  0.111364
dtype: float64
run time : 0.0052852630615234375
```

```
[21] s_time = time.time()
     airbnb.fillna(airbnb._get_numeric_data().mean(),inplace = True)
     print(f'run time : {time.time() - s_time}')
```

```
run time : 0.007885217666625977
```

Fill null dates

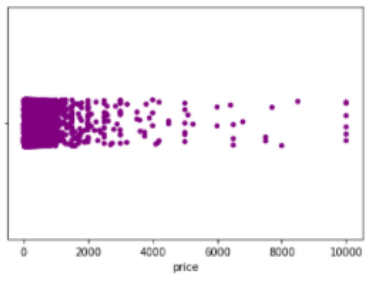
Fill NaN date datas

```
s_time = time.time()
airbnb.last_review=airbnb.last_review.fillna('20120101')    #fill nan dates
print(airbnb.head())
print(f'run time : {time.time() - s_time}')
```

Dataset plot before normalization

Dataset plot before normalization

```
import seaborn as sns

sns.stripplot(x=airbnb.price,color='purple',marker='o')
plt.show()
```



Normalization refers to rescaling real-valued numeric attributes into a 00 to 11 range. Data normalization is used in machine learning to make model training less sensitive to the scale of

features. This allows our model to converge to better weights and, in turn, leads to a more accurate model.

Normalization makes the features more consistent with each other, which allows the model to predict outputs more accurately.

Data normalization

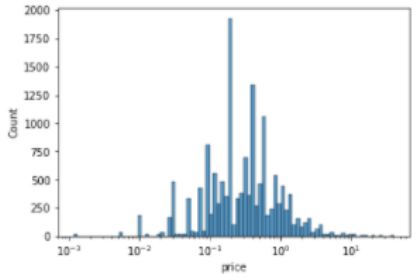
```
s_time = time.time()
for cols in airbnb.columns:
   if (airbnb[cols].dtypes == 'int64' or airbnb[cols].dtypes == 'float64'):
       #normalized = preprocessing.normalize(airbnb[cols])
       airbnb[cols]=(airbnb[cols]-airbnb[cols].mean())/airbnb[cols].std()
       print(airbnb[cols])
       print('-------------------------------------')

print(f'run time : {time.time() - s_time}')
```

```
0       -1.493834
1        0.452431
2        1.468384
3       -0.803389
4        1.275647
           ...
48890   -0.924607
48891   -0.497136
48892    1.573464
48893    0.523768
48894    0.643519
Name: latitude, Length: 48895, dtype: float64
-------------------------------------
0       -0.437648
1       -0.684632
2        0.222494
3       -0.164448
4        0.177214
```

Dataset distribution plot before deleting outlier data

```
s_time = time.time()
import seaborn as sns
sns.histplot(airbnb.price,log_scale=True)
plt.show()
print(f'run time : {time.time() - s_time}')
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/arraylike.py:364: Ru
  result = getattr(ufunc, method)(*inputs, **kwargs)
```




```
run time : 0.7372860908508301
```

Delete outliers

Delete outlier data

```
[31] airbnb.shape
```

```
(48895, 18)
```

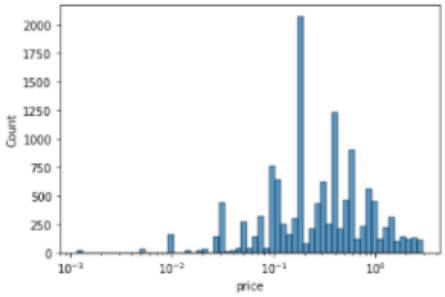
```
s_time = time.time()
for cols in airbnb.columns:
   if (airbnb[cols].dtypes == 'int64' or airbnb[cols].dtypes == 'float64'):
       upper_range=(airbnb[cols].mean()+3 )* airbnb[cols].std()
       lower_range=(airbnb[cols].mean()-3 )* airbnb[cols].std()

       indexes=airbnb[(airbnb[cols]>upper_range)|(airbnb[cols]< lower_range)].index
       airbnb=airbnb.drop(indexes)
print(f'run time : {time.time() - s_time}')
```

```
run time : 0.1155390739440918
```

Dataset distribution plot after deleting outlier data

```
import seaborn as sns
sns.histplot(x=airbnb.price,log_scale=True)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/arraylike.py:364: RuntimeWarnin
  result = getattr(ufunc, method)(*inputs, **kwargs)
```



Fill categorical data

filling categorical data

we can replace null datas using 2 methods:

1. replace with most frequents of each columns or
2. replace with pre-phrases

```
s_time = time.time()

print('list of most frequents:\n')  #fill with most frequents
for cols in airbnb.columns:
   if (airbnb[cols].dtypes == 'object' or airbnb[cols].dtypes == 'bool'):
      print(cols,": ",airbnb[cols].value_counts().head(1),"\n")

print(f'run time : {time.time() - s_time}')
```

```
list of most frequents:

name :  Brooklyn Apartment    12
Name: name, dtype: int64

host_id :  137358866    103
Name: host_id, dtype: int64

host_name :  Michael    353
Name: host_name, dtype: int64

neighbourhood_group :  Manhattan    19658
Name: neighbourhood_group, dtype: int64
```

Fill NaN values with most frequents

```
[36] s_time = time.time()
     #fill NaN values with most frequents
     for cols in airbnb.columns:
        if (airbnb[cols].dtypes == 'object' or airbnb[cols].dtypes == 'bool'):
            airbnb[cols].fillna(airbnb[cols].value_counts().head(1).index[0],
                                inplace=True)
     print(f'run time : {time.time() - s_time}')

     run time : 0.06427884101867676
```

```
airbnb.isna().sum()
```

```
name                            0
host_id                         0
host_name                       0
neighbourhood_group             0
neighbourhood                   0
latitude                        0
longitude                       0
room_type                       0
```

```
[41] s_time = time.time()
     for cols in airbnb.columns:
        if (airbnb[cols].dtypes == 'object' or airbnb[cols].dtypes == 'bool'):
           print('column: {}, unique values : {}'.format(cols,airbnb[cols].nunique()))

     print(f'run time : {time.time() - s_time}')
```

```
column: name, unique values : 41874
column: host_id, unique values : 33883
column: host_name, unique values : 10417
column: neighbourhood_group, unique values : 4
column: neighbourhood, unique values : 130
column: room_type, unique values : 3
run time : 0.02962017059326172
```

Reduce number of categories

```
[44] #airbnb=df.copy()
```

```
s_time = time.time()
others = list(airbnb['neighbourhood'].value_counts().tail(3).index)
def edit_name(x):
    if x in others:
        return 'other'
    else:
        return x

airbnb['neighbourhood_edit'] = airbnb['neighbourhood'].apply(edit_name)
airbnb = airbnb.drop(columns = ['neighbourhood'])
print(airbnb['neighbourhood_edit'].value_counts()*100 / len(airbnb))

print(f'run time : {time.time() - s_time}')
```

```
Williamsburg          8.709194
Bedford-Stuyvesant    8.158868
Harlem                5.896679
Bushwick              5.498571
Upper West Side       4.283172
                          ...
Mill Basin            0.009367
Olinville             0.007025
Richmond Hill         0.007025
Souvten Duyvil        0.007025
```

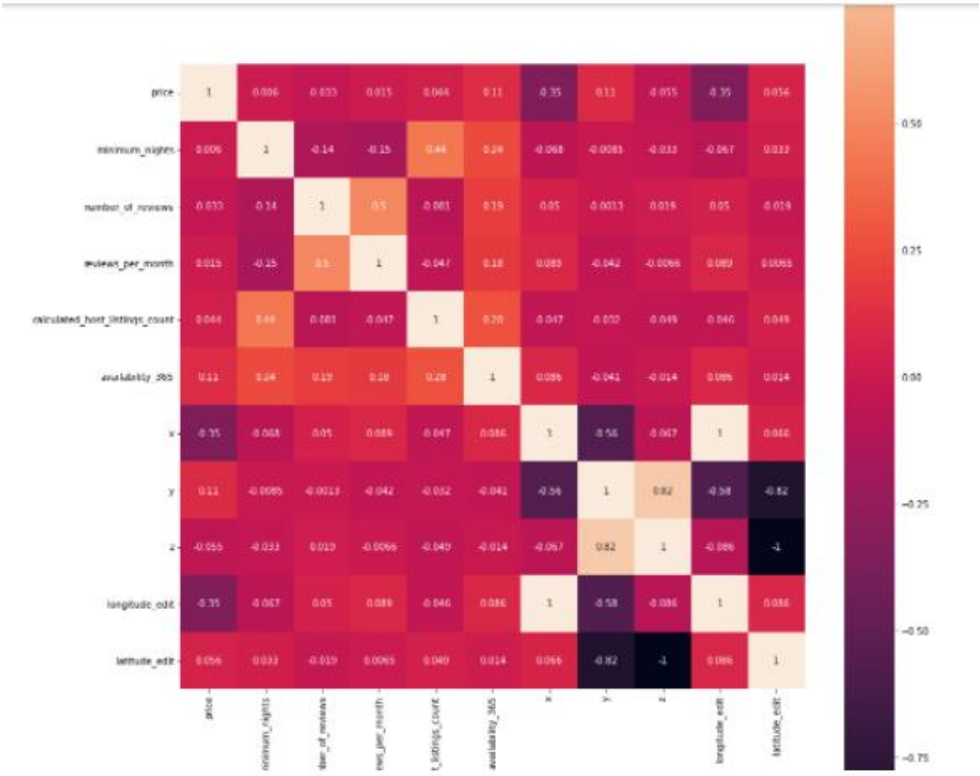
Correlation matrix

Correlation matrix

Using this matrix, we can see the linear convergence of the data. The closer the numbers to 1,means more convergent , and the closer numbers to -1, means more inversely relations. In the next steps, we can further study the variables that are more convergent

```
[52] #corr=airbnb.corr()
```

```
s_time = time.time()
corr=airbnb.corr()
import seaborn as sns
import matplotlib.pyplot as plt

f,ax=plt.subplots(figsize=(15,20))
#sns.heatmap(corr, square = True ,annot = True)

sns.heatmap(corr,square= True,annot=True,vmax=None, cmap=None, center=None,
            cbar=True, cbar_kws=None, cbar_ax=None,  xticklabels="auto", yticklabels="auto")

print(f'run time : {time.time() - s_time}')
```



convert categorical data to dummies

```
s_time=time.time()

columns1=[]
for cols in airbnb.columns:
    if airbnb[cols].dtype == 'object' or airbnb[cols].dtype == 'bool':
        columns1.append(cols)
print(columns1)

print(f'run time : {time.time() - s_time}')
```

```
['host_id', 'host_name', 'room_type', 'neighbourhood_edit', 'name_edit', 'neighbourhood_group_edit']
run time : 0.0013256072998046875
```

```
[56] columns1=['room_type', 'neighbourhood_edit', 'name_edit', 'neighbourhood_group_edit']
```

```
[57] s_time=time.time()

s=pd.Series(columns1)
dummies_feature=pd.get_dummies(s)
print(dummies_feature)

print(f'run time : {time.time() - s_time}')
```

Add dummies to dataset and removing data, which converted to dummies

```
s_time=time.time()

s=pd.Series(columns1)
dummies_feature=pd.get_dummies(s)
print(dummies_feature)

print(f'run time : {time.time() - s_time}')
```

```
   name_edit  neighbourhood_edit  neighbourhood_group_edit  room_type
0          0                   0                         0          1
1          0                   1                         0          0
2          1                   0                         0          0
3          0                   0                         1          0
run time : 0.011436939239501953
```

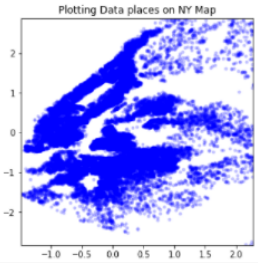
```
[58] s_time=time.time()
airbnb.drop(columns=columns1,inplace=True)
airbnb=pd.concat([airbnb,dummies_feature],axis=1)
print(airbnb.shape)
print(f'run time : {time.time() - s_time}')
```

```
(42703, 18)
run time : 0.028095722198486328
```

Plot data on map

```
s_time=time.time()
BBox = ((airbnb.longitude_edit.min(),  airbnb.longitude_edit.max(), airbnb.latitude_edit.min(), airbnb.latitude_edit.max() ))
fig, ax = plt.subplots(figsize = (5,5))
ax.scatter(airbnb.longitude_edit, airbnb.latitude_edit, zorder=1, alpha= 0.2, c='b', s=10)
ax.set_title('Plotting Data places on NY Map')
ax.set_xlim(BBox[0],BBox[1])
ax.set_ylim(BBox[2],BBox[3])
#ax.imshow(X,rhum='NY.png', zorder=0, extent = BBox, aspect= 'equal')

print(f'run time : {time.time() - s_time}')
```

```
run time : 0.024181365966796875
```



0s completed at 6:32 PM

We can see that data has reduced.

PCA

PCA

Principal Component Analysis is basically a statistical procedure to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
[66] s_time=time.time()
     print(airbnb.reset_index)

     print(f'run time : {time.time() - s_time}')
```

```
<bound method DataFrame.reset_index of         host_id      host_name     price  minimum_nights  number_of_reviews  \
0          2787           John -0.015493       -0.293993         -0.320410
1          2845       Jennifer  0.300970       -0.293993          0.487660
2          4632      Elisabeth -0.011329       -0.196482         -0.522428
3          4632      Elisabeth -0.011329       -0.196482         -0.522428
4          7192          Laura -0.302808        0.144805         -0.320410
...         ...            ...       ...             ...               ...
48890   8232441        Sabrina -0.344448       -0.245238         -0.522428
48891   6570630        Marisol -0.469368       -0.147727         -0.522428
```

0s completed at 6:32 PM

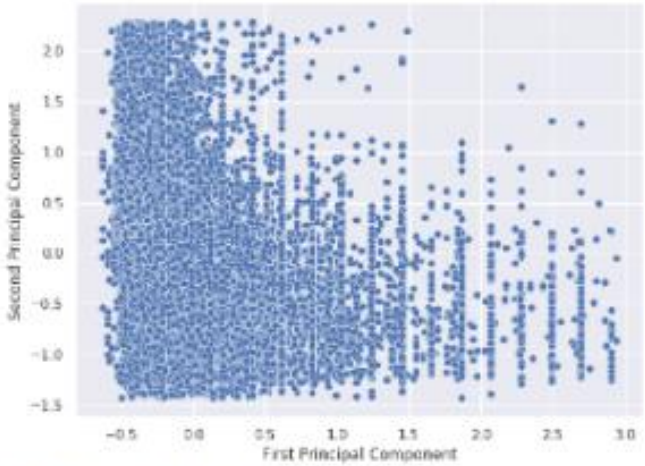
Split features and target

```
s_time=time.time()
# Splitting data into y as target and x as features
y= airbnb['price']
X= airbnb.drop(columns=['price','last_review','host_id','host_name'])
print(X.dtypes)
print(f'run time : {time.time() - s_time}')
```

```
[68] print(X.shape)
     print(y.shape)
```

```
(42703, 14)
(42703,)
```

```
s_time=time.time()
# giving a larger plot
plt.figure(figsize =(8, 6))

sns.set(rc={'figure.figsize':(15,10)})
sns.scatterplot(x='price',y='x', data=airbnb)

# labeling x and y axes
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.show()

print(f'run time : {time.time() - s_time}')
```



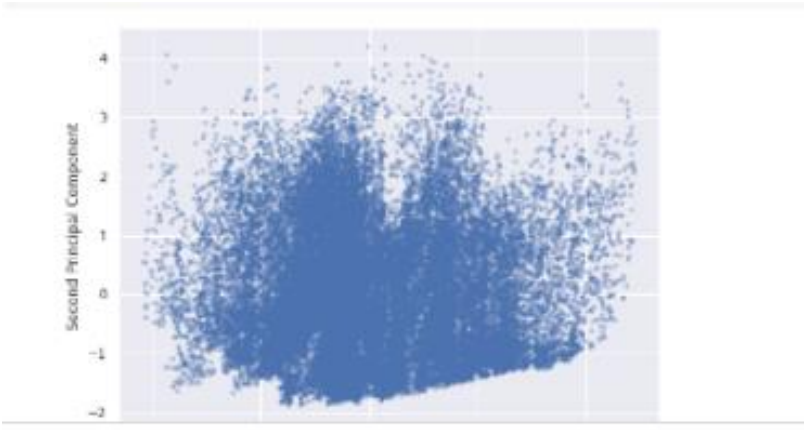

```
run time : 0.39535951614379883
```

PCA(second code)

```
s_time = time.time()
from sklearn.decomposition import PCA
pca = PCA(0.70)
x_pca =pca.fit_transform(X)
print(x_pca.shape)
print(x_pca[:1])

print(f'run time : {time.time() - s_time}')
```

```
(42703, 3)
[[-2.62678489  1.24242722 -1.07819435]]
run time : 0.3700432777404785
```

```
[71] s_time = time.time()

     # giving a larger plot
     plt.figure(figsize =(8, 6))
     plt.scatter(x_pca[:,0], x_pca[:,1],marker='.',cmap ='plasma',alpha=0.3)

     # labeling x and y axes
     plt.xlabel('First Principal Component')
     plt.ylabel('Second Principal Component')
     plt.show()

     print(f'run time : {time.time() - s_time}')
```




```
s_time = time.time()

from sklearn.decomposition import PCA
pca = PCA(0.80)
x_pca =pca.fit_transform(X)
print(x_pca.shape)
print(x_pca[:1])
print(f'run time : {time.time() - s_time}')
```

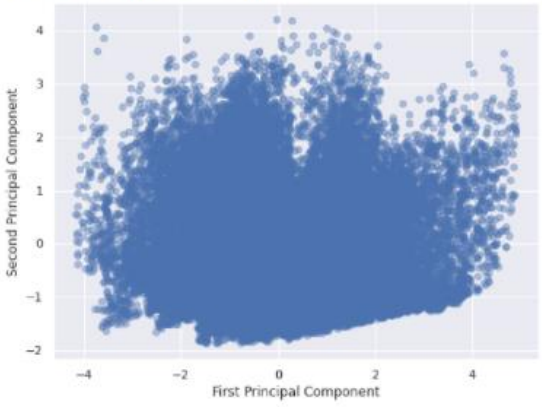
```
(42703, 4)
[[-2.62678489  1.24242722 -1.07819435 -1.27128183]]
run time : 0.04254031181335449
```

```
[73] s_time = time.time()

from sklearn.decomposition import PCA
pca = PCA(0.90)
x_pca =pca.fit_transform(X)
print(x_pca.shape)
print(x_pca[:1])
print(f'run time : {time.time() - s_time}')
```

```
(42703, 4)
[[-2.62678489  1.24242722 -1.07819435 -1.27128183]]
run time : 0.0467228889465332
```

```
s_time = time.time()
# giving a larger plot
plt.figure(figsize =(8, 6))
plt.scatter(x_pca[:, 0], x_pca[:, 1], cmap ='plasma',alpha=0.4)

# labeling x and y axes
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')

print(f'run time : {time.time() - s_time}')
```

```
run time : 0.054419517517089844
```



Linear Regression

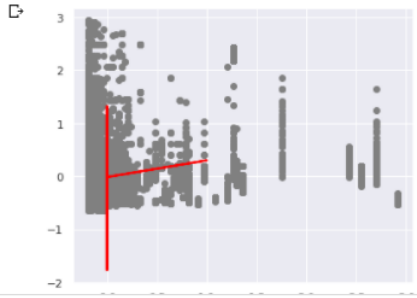
```
s_time = time.time()
from sklearn.linear_model import LinearRegression

#df=airbnb._get_numeric_data()
model=LinearRegression()
model.fit(X,y)
y_pred=model.predict(X)

print(f'run time : {time.time() - s_time}')
```

```
run time : 0.029028654098510742
```

```
[78] s_time = time.time()

     plt.figure(figsize =(6,5))
     plt.scatter(X['calculated_host_listings_count'],y,color='gray')
     plt.plot(X['neighbourhood_edit'],y_pred,color='red',linewidth=2)
     plt.show()
     print(f'run time : {time.time() - s_time}')
```

```
print(f'run time : {time.time() - s_time}')

run time : 0.029028654098510742
```

```
s_time = time.time()

plt.figure(figsize =(6,5))
plt.scatter(X['calculated_host_listings_count'],y,color='gray')
plt.plot(X['neighbourhood_edit'],y_pred,color='red',linewidth=2)
plt.show()
print(f'run time : {time.time() - s_time}')
```



✓ 0s completed at 6:32 PM

linear regression

```
[79] import numpy as np
     from sklearn.linear_model import LinearRegression
```

```
[80] s_time = time.time()
     x = np.array(airbnb['price']).reshape((-1, 1))
     y = np.array(X)
     model = LinearRegression()
     model.fit(x, y)
     model = LinearRegression().fit(x, y)
     print(f'run time : {time.time() - s_time}')

     run time : 0.027324676513671875
```

Get results

properties of the model

```
[81] s_time = time.time()
     r_sq = model.score(x, y)
     print('coefficient of determination:', r_sq)
     print('intercept:')
     print(model.intercept_)
     print('slope:')
     print(model.coef_)
     print(f'run time : {time.time() - s_time}')
```

✓ 0s completed at 6:32 PM

```
s_time = time.time()
r_sq = model.score(x, y)
print('coefficient of determination:', r_sq)
print('intercept:')
print(model.intercept_)
print('slope:')
print(model.coef_)
print(f'run time : {time.time() - s_time}')
```

```
coefficient of determination: 0.019546657377633687
intercept:
[-5.13258841e-02 -1.30521833e-01 -1.43048355e-01 -1.01463568e-01
 -6.63442724e-02 -1.14087594e-01 -5.32741449e-02 -5.52750111e-02
 -1.10846912e-01  5.57091686e-02  2.36589435e-05  2.55171989e-05
  9.99927190e-01  2.36341668e-05]
slope:
[[ 5.76590364e-03]
 [-4.75835418e-02]
 [ 2.40910708e-02]
 [ 3.40810195e-02]
 [ 2.42951951e-01]
 [-5.25852305e-01]
 [ 1.80322621e-01]
 [-1.15648691e-01]
 [-5.19442979e-01]
 [ 1.16919839e-01]
 [ 4.65130310e-06]
 [ 4.04584437e-05]
 [-4.92836214e-05]
 [ 4.17387456e-06]]
run time : 0.01967144012451172
```

```
s_time = time.time()
y_pred = model.predict(x)
print('predicted response:', y_pred, sep='\n')
print(f'run time : {time.time() - s_time}')
```

```
predicted response:
[[-5.14152147e-02 -1.29784625e-01 -1.43421595e-01 ...  2.48903798e-05
   9.99927953e-01  2.35695013e-05]
 [-4.95905174e-02 -1.44843073e-01 -1.35797654e-01 ...  3.76939956e-05
   9.99912357e-01  2.48903798e-05]
 [-5.13912055e-02 -1.29982762e-01 -1.43321280e-01 ...  2.50588484e-05
   9.99927748e-01  2.35868813e-05]
 ...
 [-5.22315267e-02 -1.23047951e-01 -1.46832306e-01 ...  1.91624464e-05
   9.99934931e-01  2.29785820e-05]
 [-5.36720772e-02 -1.11159702e-01 -1.52851207e-01 ...  9.05432866e-06
   9.99947244e-01  2.19357833e-05]
 [-5.28317561e-02 -1.18094514e-01 -1.49340182e-01 ...  1.49507307e-05
   9.99940061e-01  2.25440825e-05]]
run time : 0.010792970657348633
```

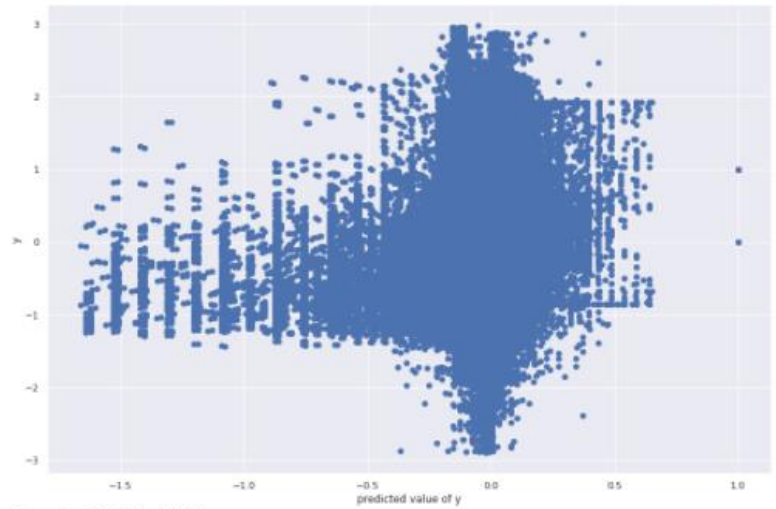
Another linear regression

Linear Regression

```
[83] s_time = time.time()
     from sklearn import metrics
     prediction = []
     def linearregression(X, y_pred, x, y):
         linreg = LinearRegression()
         linreg.fit(X, y_pred)
         y_pred = linreg.predict(X)

     print('MAE:', metrics.mean_absolute_error(y, y_pred))
     print('MSE:', metrics.mean_squared_error(y, y_pred))
     print('R2_score:', metrics.r2_score(y, y_pred))

     plt.scatter(y_pred,y)
     plt.xlabel('predicted value of y')
     plt.ylabel('y')
     plt.figure()
     linearregression(X, y_pred, x, y)
     print(f'run time : {time.time() - s_time}')
```

```
MAE: 0.38346036850335974
MSE: 0.38493322235486255
R2_score: 0.019546657377633687
run time : 0.14904546737670898
```

```
MAE: 0.38346036850335974
MSE: 0.38493322235486255
R2_score: 0.019546657377633687
run time : 0.14904546737670898
```



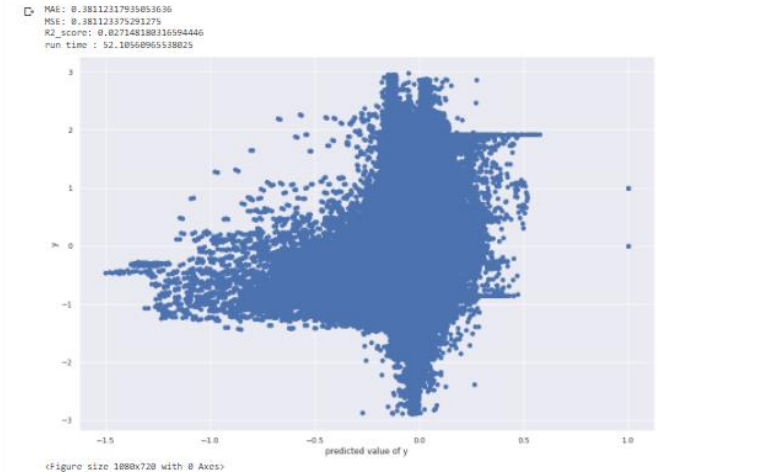

```
<Figure size 1080x720 with 0 Axes>
```

Random forest regression

Random Forest Regresson

```
[84] s_time = time.time()

     from sklearn.ensemble import RandomForestRegressor
     from sklearn import metrics

     def RandomForest(X, y_pred, x, y):
         randomForest = RandomForestRegressor()
         randomForest.fit(X, y_pred)
         y_pred = randomForest.predict(X)

         print('MAE:', metrics.mean_absolute_error(y, y_pred))
         print('MSE:', metrics.mean_squared_error(y, y_pred))
         print('R2_score:', metrics.r2_score(y, y_pred))

         plt.scatter(y_pred,y,cmap ='plasma')
         plt.xlabel('predicted value of y')
         plt.ylabel('y')
         plt.figure()

     RandomForest(X, y_pred, x, y)
     print(f'run time : {time.time() - s_time}')
```

```
MAE: 0.38112317935053636
MSE: 0.381123375291275
R2_score: 0.027148180316594446
run time : 52.10560965538025
```




```
<Figure size 1080x720 with 0 Axes>
```

Polynomial regression

polynomial regression

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

s_time = time.time()
x = np.array(airbnb['price']).reshape((-1, 1))
y = np.array(X)
transformer = PolynomialFeatures(degree=2, include_bias=False)

transformer.fit(x)   #we need to fit before transfer

x_ = transformer.transform(x)
x_ = PolynomialFeatures(degree=2, include_bias=False).fit_transform(x)

print(x_)

model = LinearRegression().fit(x_, y)   #creating model and fit to x

print(f'run time : {time.time() - s_time}')
```

```
[[-1.54929110e-02  2.40038292e-04]
 [ 3.00970468e-01  9.05832225e-02]
 [-1.13289192e-02  1.28344410e-04]
 ...
 [-1.57068633e-01  2.46705555e-02]
 [-4.06908143e-01  1.65574237e-01]
 [-2.61168429e-01  6.82089482e-02]]
run time : 0.021266222000122070
```

Get results

properties of the model

```
s_time = time.time()

r_sq = model.score(x_, y)
print('coefficient of determination:', r_sq)
print('intercept:')
print(model.intercept_)
print('coefficients:')
print(model.coef_)

print(f'run time : {time.time() - s_time}')
```

```
coefficient of determination: 0.028055647030316537
intercept:
[-4.91476551e-02 -1.19673174e-01 -1.42076960e-01 -9.98523200e-02
 -7.79253034e-02 -2.09207904e-01 -2.14993032e-02 -7.49888136e-02
 -2.04832568e-01  7.56448357e-02  3.14679088e-05  4.04555177e-05
  9.99896770e-01  3.13065747e-05]
coefficients:
[[ 1.42790483e-02 -8.39120903e-03]
 [-5.18386817e-03 -4.17923739e-02]
 [ 2.78875610e-02 -3.74211233e-03]
 [ 4.03782415e-02 -6.20702546e-03]
 [ 1.97689955e-01  4.46136983e-02]
 [-8.97609801e-01  3.66432733e-01]
 [ 3.04507830e-01 -1.22406477e-01]
 [-1.92695893e-01  7.59436389e-02]
 [-8.86765920e-01  3.62061694e-01]
 [ 1.94834152e-01 -7.67983295e-02]
 [ 3.51709823e-05 -3.00825392e-05]
```

Fitting logistic regression to dataset

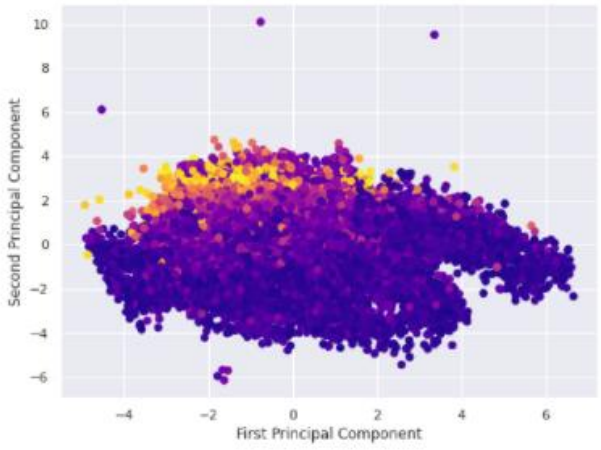
Fitting Logistic Regression To the dataset

```
s_time = time.time()

# Importing standardscalar module
from sklearn.preprocessing import StandardScaler

scalar = StandardScaler()
df=airbnb._get_numeric_data()

# fitting
scalar.fit(df)
scaled_data = scalar.transform(df)

# Importing PCA
from sklearn.decomposition import PCA

# Let's say, components = 2
pca = PCA(n_components = 2)
pca.fit(scaled_data)
x_pca = pca.transform(scaled_data)

print(x_pca.shape)

print(f'run time : {time.time() - s_time}')
```

✓ 0s completed at 6:32 PM

```
s_time = time.time()
# giving a larger plot
plt.figure(figsize =(8, 6))

plt.scatter(x_pca[:, 0], x_pca[:, 1], c = airbnb['price'], cmap ='plasma')

# labeling x and y axes
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')


print(f'run time : {time.time() - s_time}')
```

```
run time : 0.05212259292602539
```



Quantile regression

Quantile regression provides an alternative to ordinary least squares (OLS) regression and related methods, which typically assume that associations between independent and dependent variables are the same at all levels.

In OLS regression, the goal is to minimize the distances between the values predicted by the regression line and the observed values.

In contrast, quantile regression differentially weights the distances between the values predicted by the regression line and the observed values, and then tries to minimize the weighted distances.[1]

This kind of regression may work good on this dataset.

[1] Buchinsky M. Recent advances in quantile regression models: a practical guideline for empirical research. *Journal of Human Resources.* 1998;33(1):88–126. [Google Scholar]

Review of general price indices

prices

```
[90] airbnb["price"].describe()
```

```
count    42703.000000
mean        -0.051896
std          0.451936
min         -0.635928
25%         -0.344448
50%         -0.177889
75%          0.092771
max          2.945105
Name: price, dtype: float64
```

quantiles

```
s_time = time.time()

visual_data = airbnb.copy()

print("5%: ",visual_data.quantile(0.05)['price'])
print("25%: ", visual_data.quantile(0.25)['price'])
print("50%: ", visual_data.quantile(0.5)['price'])
print("75%: ", visual_data.quantile(0.75)['price'])
print("95%: ", visual_data.quantile(0.95)['price'])

print(f'run time : {time.time() - s_time}')
```

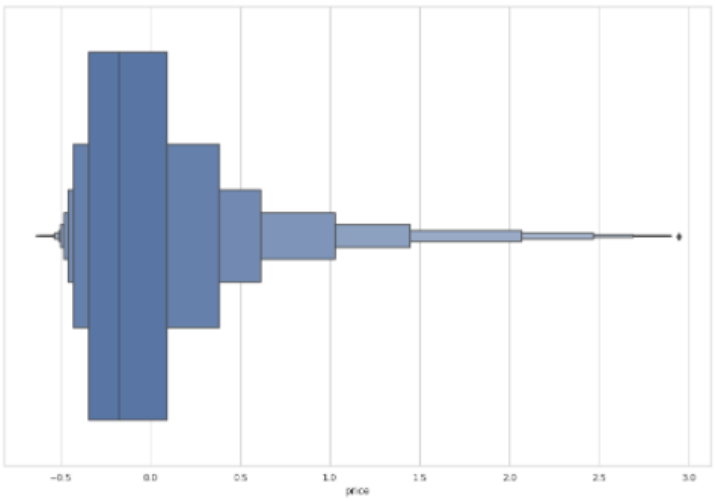
```
5%:  -0.4693680201575976
25%:  -0.34444826535350115
50%:  -0.17788859228137247
75%:  0.09277087646083658
95%:  0.8214694461513994
run time : 0.11752533912658691
```

Data visualization

Plot quantile values

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the
  import pandas.util.testing as tm
5 th percentile  :  -0.4693680201575976
25th percentile  :  -0.34444826535350115
50th percentile  :  -0.17788859228137247
75th percentile  :  0.09277087646083658
```

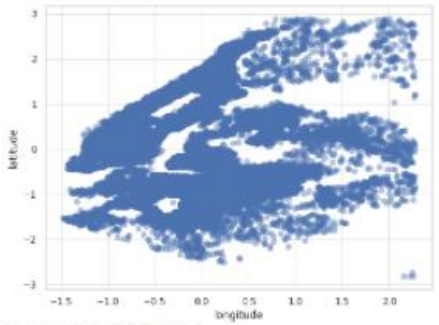
```
import seaborn as sns
sns.set_theme(style="whitegrid")
ax = sns.boxenplot(x=airbnb['price'])
```




```
[95] s_time = time.time()
```

Scatter plot of quantile values for price

```
s_time = time.time()

price_95_percent = visual_data[visual_data['price'] <= visual_data.quantile(0.95)['price']]
plt.figure(figsize =(8, 6))
plt.scatter(price_95_percent['longitude_edit'], price_95_percent['latitude_edit'], cmap='plasma', alpha=0.4)
plt.xlabel('longitude')
plt.ylabel('latitude')
plt.show()

print(f'run time : {time.time() - s_time}')
```

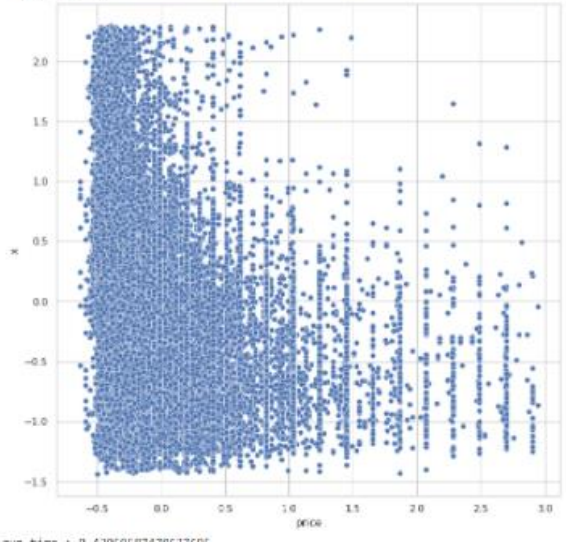



```
run time : 1.2987497882843018
```

Scatter plot of price values base on place

```
s_time = time.time()

from pylab import rcParams
rcParams['figure.figsize']=10,10
plot1=sns.scatterplot(airbnb['price'],airbnb['x'])
plt.show()

print(f'run time : {time.time() - s_time}')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args
  FutureWarning
```



Find Max and Min price values

Most (normalized) price is 2.94510527782104

Id number advertises most price in record 5351: 14733148

Maximum price in each neighbourhood_edit:
0.0 2.945105
1.0 0.300970

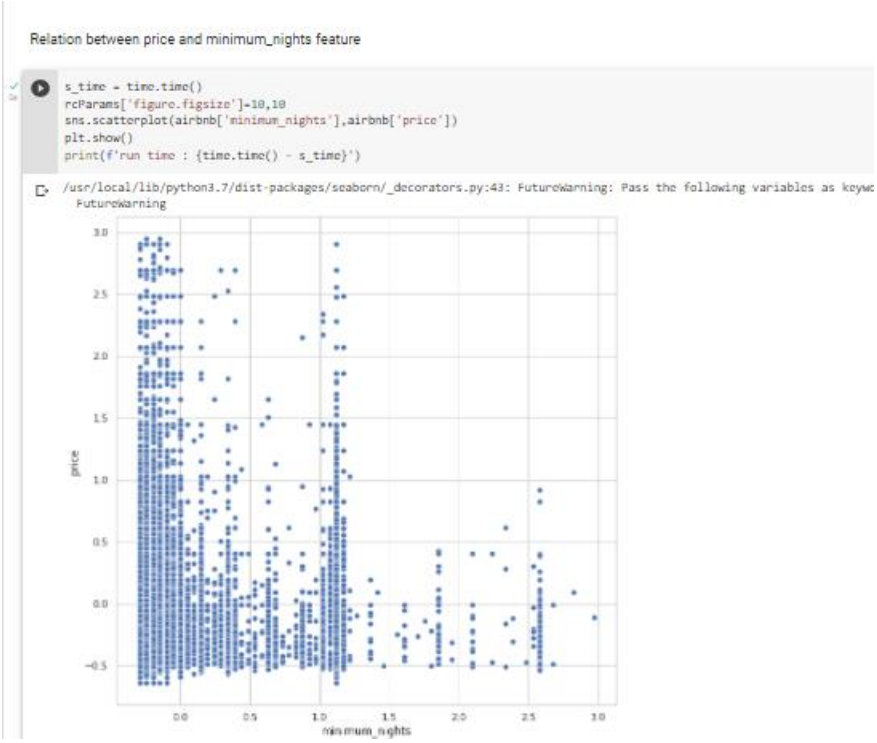
Minimum (normalized) price is -0.6359276932297263
Id number advertises minimum price in record 25753: 1641537

Minimum price in each neighbourhood_group_edit:
0.0 -0.015493
1.0 -0.635928

Average price in each neighbourhood_edit
0.0 -0.051904
1.0 0.300970

Relation between price and minimum nights

As we can see in next scatter plot, there is a direct relationship between price and the number of nights :

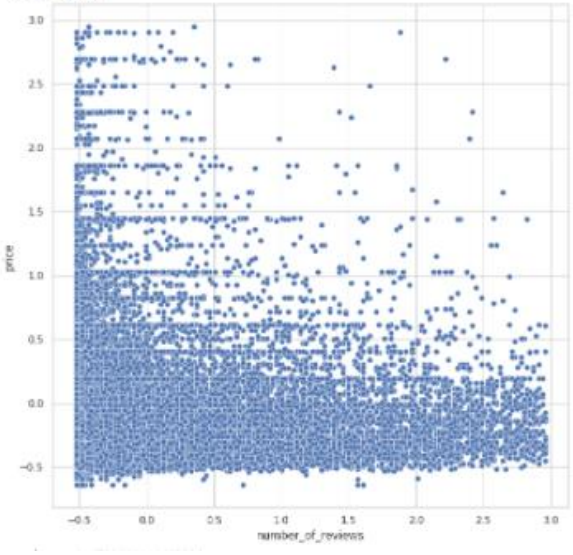


Relation between price and number of reviews:

As we can see in next scatter plot, there is a reverse relationship between price and the number of reviews, less price leads to more reviews:

```
s_time = time.time()
rcParams['figure.figsize']=10,10
sns.scatterplot(airbnb['number_of_reviews'],airbnb['price'])
plt.show()
print(f'run time : {time.time() - s_time}')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as
  FutureWarning
```



Relation between price and room type:

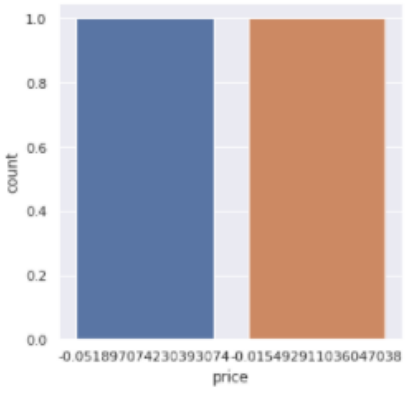
Room type in second category have more prices

```
[157] airbnb.groupby("room_type")["price"].mean()
```

```
room_type
0.0   -0.051897
1.0   -0.015493
Name: price, dtype: float64
```

```
plt.figure(figsize=(5,5))
sns.countplot(airbnb.groupby("room_type")["price"].mean())
plt.show()
```

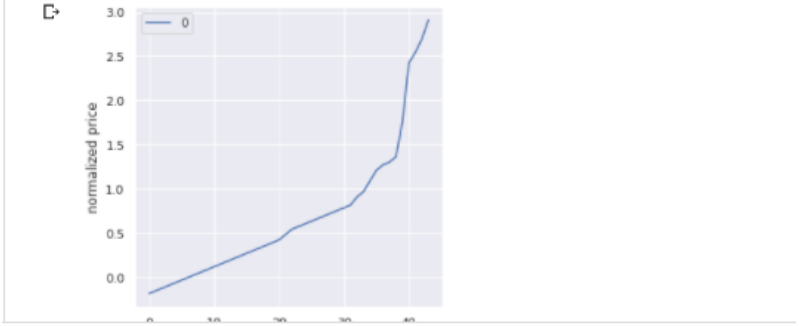
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureW
  FutureWarning
```



Relation between price and host listings count:

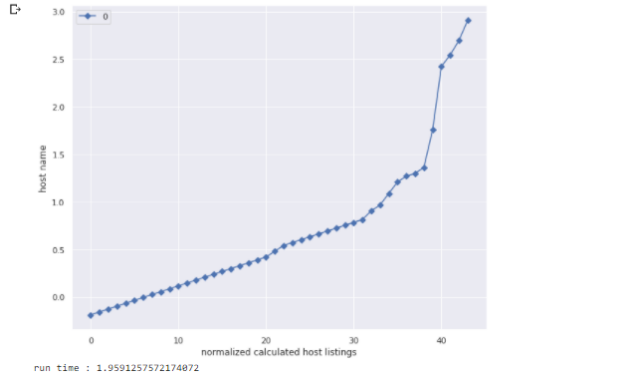
Plot based on relation between price and host listings count, shows straight relation

```
import matplotlib.pyplot as plt
import seaborn as sns

s_time=time.time()
rcParams['figure.figsize']=5,5

d1=airbnb.groupby("calculated_host_listings_count")['price']

df1=pd.DataFrame(d1)
df1.plot(ylabel='normalized price',xlabel='normalized calculated host listings')
plt.show()
print(f'run time : {time.time() - s_time}')
```



Relation between calculated host listing and host name:

```
d1=airbnb.groupby("calculated_host_listings_count")['host_name']
df1=pd.DataFrame(d1)
df1.plot(ylabel='host name',xlabel='normalized calculated host listings',marker='D',)
plt.show()
print(f'run time : {time.time() - s_time}')
```




```
run time : 1.9591257572174072
```

Mask : Prices after 2-7-2016, which have more listing-counts than mean:

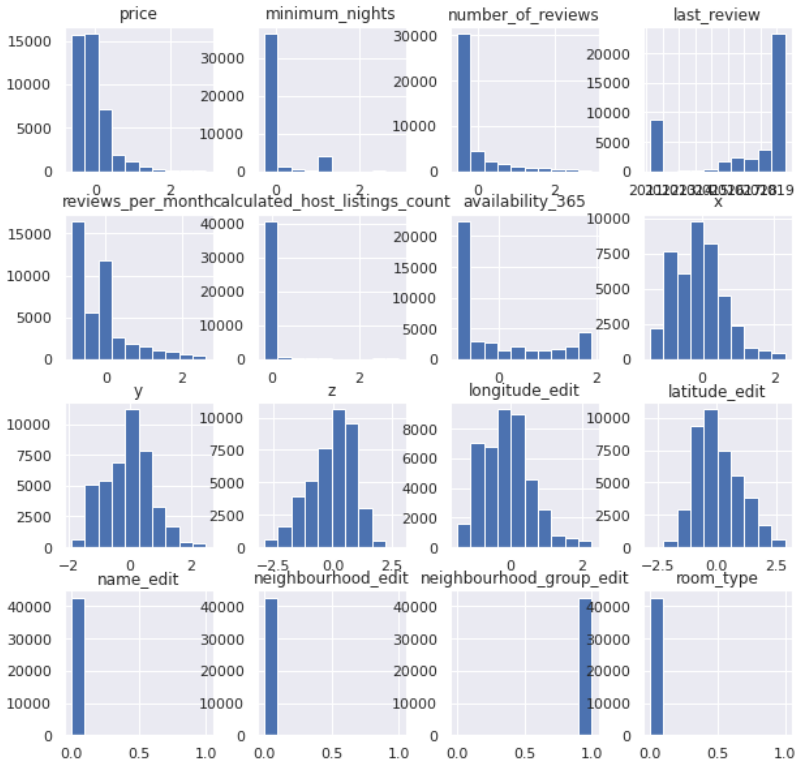
```
s_time=time.time()

import datetime
mask1=airbnb['calculated_host_listings_count']>airbnb['calculated_host_listings_count'].mean()
mask2=airbnb['last_review']> datetime.datetime(2016, 7, 2)
masked_df = airbnb[~(mask1 & mask2)]
df1=pd.DataFrame(masked_df)
print(df1)

print(f'run time : {time.time() - s_time}')
```

```
48892  23492952  Ilgar & Aysel -0.157069        0.144805          -0.522428
48893  30985759            Taz -0.406908       -0.293993          -0.522428
48894  68119814     Christophe -0.261168       -0.001461          -0.522428

      last_review  reviews_per_month  calculated_host_listings_count  \
1      2019-05-21      -6.631313e-01                       -0.156103
2      2012-01-01       2.668494e-15                       -0.186450
3      2012-01-01       2.668494e-15                       -0.186450
4      2018-11-19      -8.500753e-01                       -0.186450
5      2019-06-22      -5.229233e-01                       -0.186450
...           ...                ...                             ...
48890  2012-01-01       2.668494e-15                       -0.156103
48891  2012-01-01       2.668494e-15                       -0.156103
```

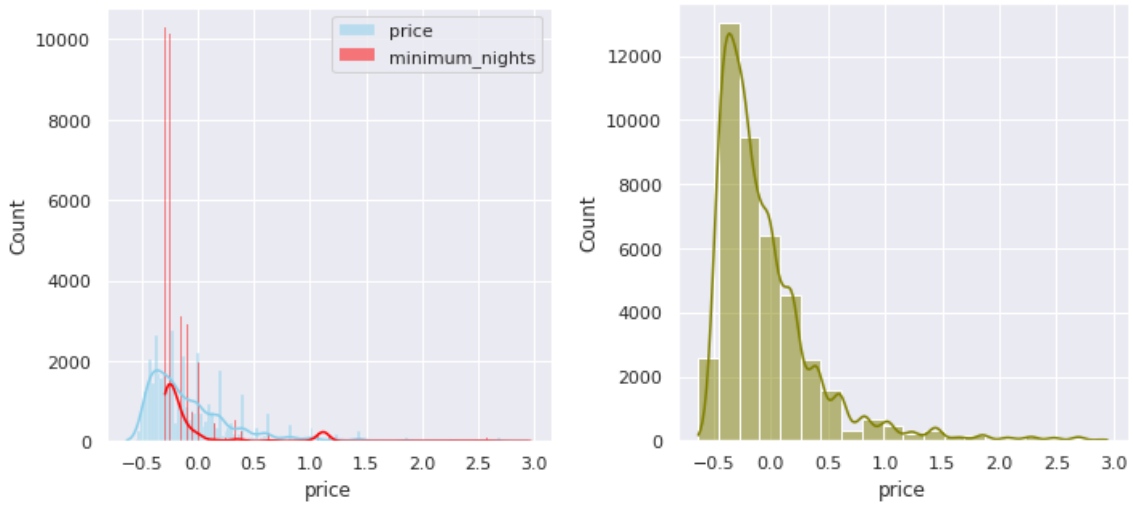
Minimum normalized price averages in neighborhood groups 0 and 1 are 1.022403 and 0.166088, so second group has less average price

Prices less than 1000:

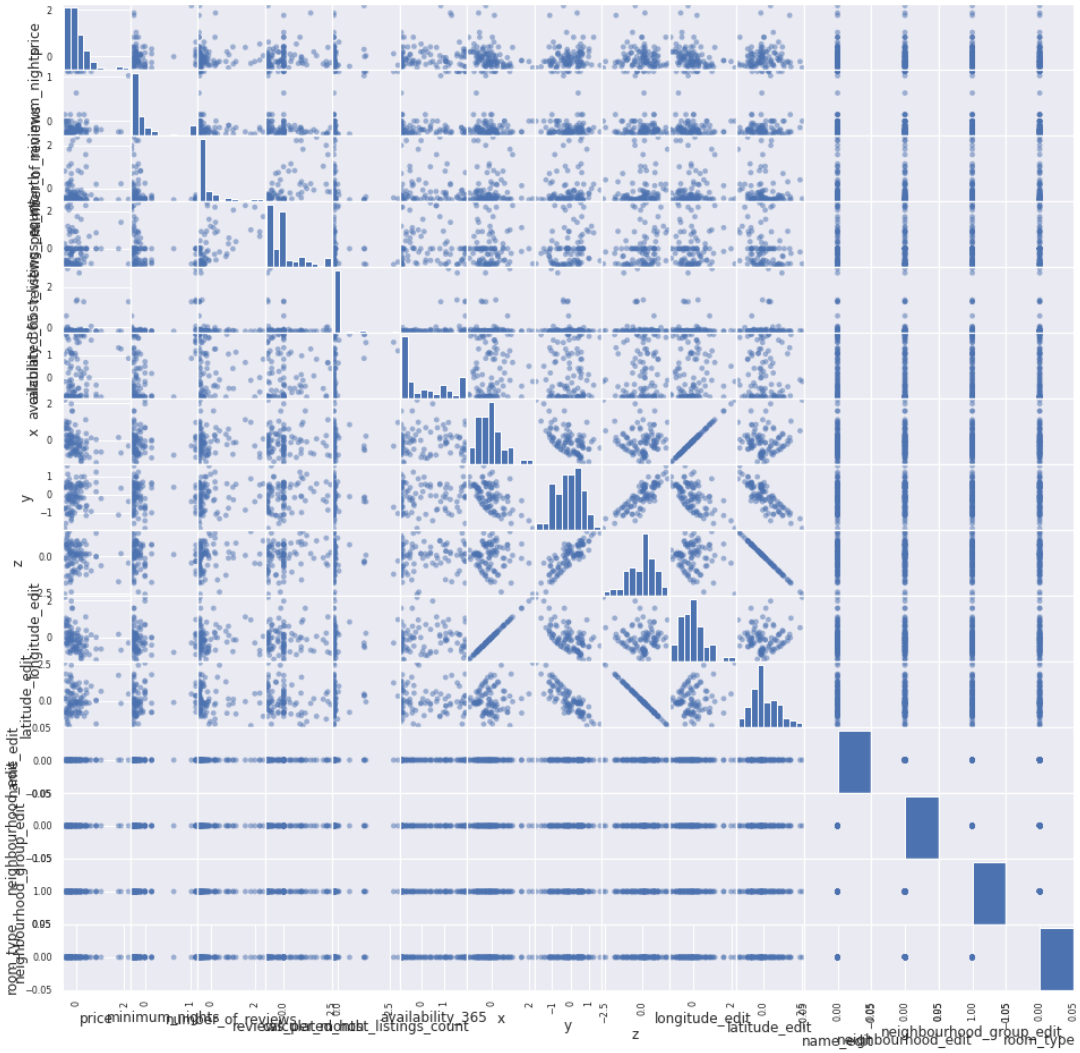
Relations between features for prices less than 250



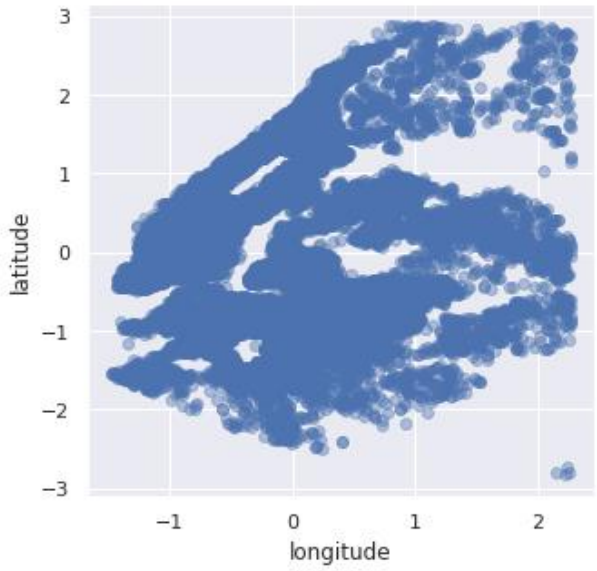
Distribution plot



Scatter matrix of features:



Place of processed data on map:



One-Hot Encoding

For categorical variables where no such ordinal relationship exists, the integer encoding is not enough.

In fact, using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories).In this case, a one-hot encoding can be applied to the integer representation. This is where the integer-encoded variable is removed and a new binary variable is added for each unique integer value.

```
s_time = time.time()
data_onehot1 = pd.get_dummies(airbnb, columns=['neighbourhood_group_edit',
                                               "room_type"],
                              prefix = ['ng',"rt"],drop_first=True)
data_onehot1.drop(["name_edit"], axis=1, inplace=True)
print(data_onehot1.shape)
data_onehot2 = pd.get_dummies(airbnb,
                              columns=['neighbourhood_group_edit',
                                       'number_of_reviews',"room_type"],
                               prefix = ['ng',"nr","rt"],drop_first=True)

print(data_onehot2.shape)

XL1= data_onehot2.loc[:, data_onehot2.columns != 'price']
YL1 = data_onehot2["price"]

print(f'run time : {time.time() - s_time}')
```

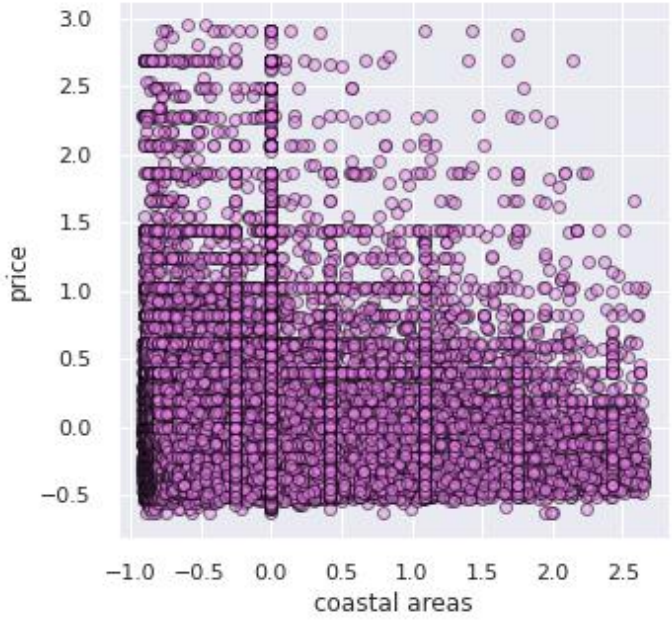
```
(42703, 17)
(42703, 172)
run time : 0.1159830093383789
```

```
[ ] s_time = time.time()
    from sklearn.model_selection import RandomizedSearchCV

    n_estimators = [int(x) for x in np.linspace(start = 200, stop = 1000, num = 5)]
    max_features = ['auto', 'sqrt']
    max_depth = [int(x) for x in np.linspace(10, 110, num = 6)]
    max_depth.append(None)
```

✓ 0s completed at 6:32 PM

{'bootstrap': [True, False],
'max_depth': [10, 30, 50, 70, 90, 110, None],
'max_features': ['auto', 'sqrt'],
'min_samples_leaf': [1, 2, 4],
'min_samples_split': [2, 5, 10],
'n_estimators': [200, 400, 600, 800, 1000]}

Prices after classification neighbourhood_group_edit in coastal and not-coastal areas:



Property type classification:

Property type classification

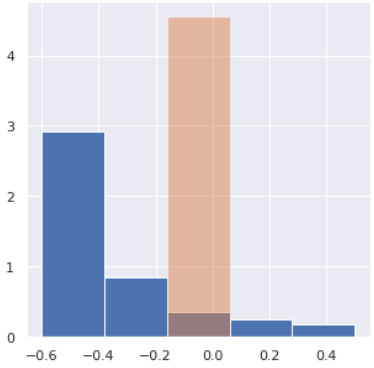
```
s_time = time.time()
from scipy.special.orthogonal import roots_hermitenorm

def classify(p):
    bigs = ['House', 'Entire home/apt', 'Condominium', 'Townhouse']
    room =['Private room']

    if p in bigs:
        return 'enough space for family'
    elif p in room:
        return 'small place for family,suitable for up to 2 persons'
    else:
        return 'other'

airbnb['property_group'] = airbnb['room_type'].apply(classify)
print(airbnb['property_group'])

print(f'run time : {time.time() - s_time}')
```

Loading...

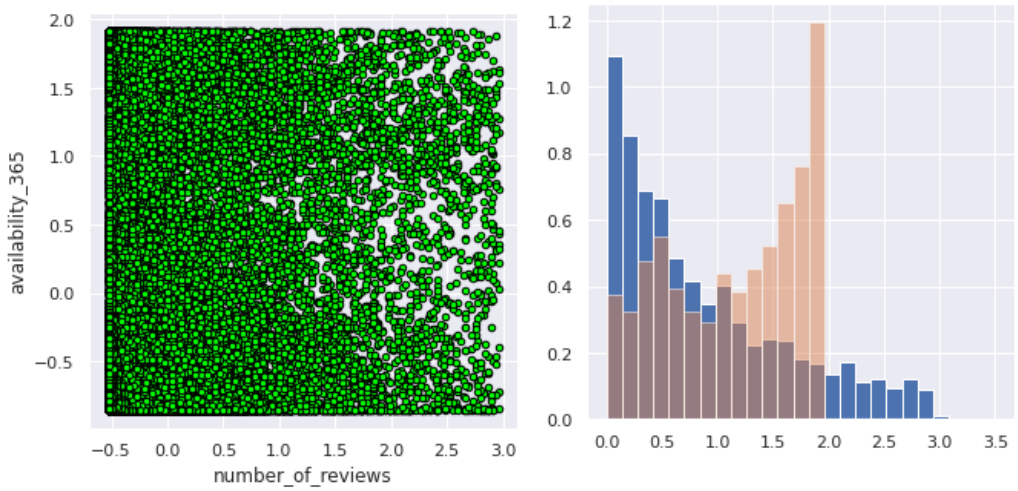
```
0        other
1        other
2        other
3        other
4        other
        ...
48890    other
48891    other
48892    other
48893    other
```

2.6684937897181106e-15 is the max rate of reviews per month

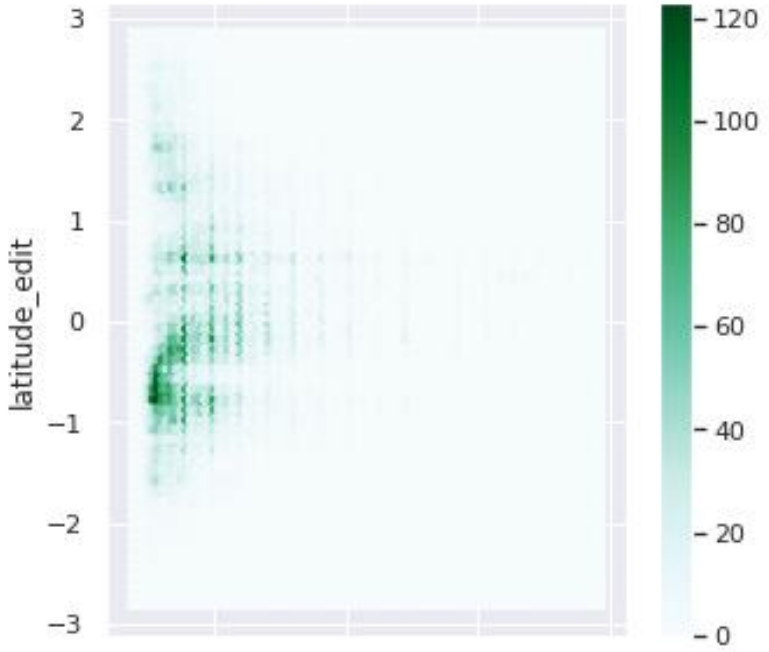
Relation between room type and number of reviews:

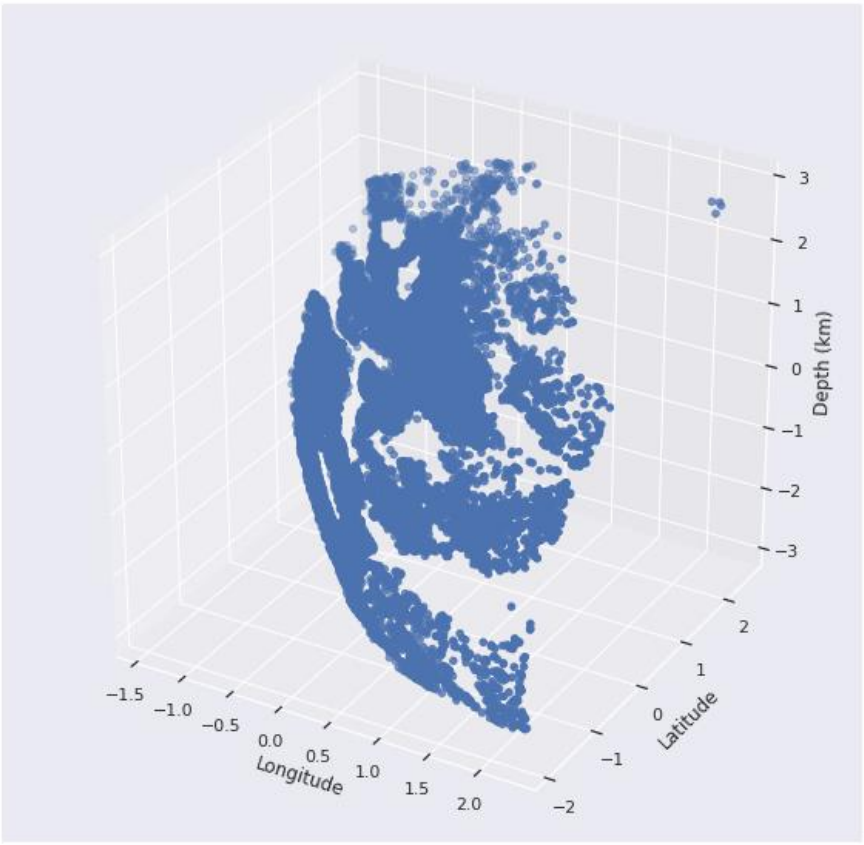


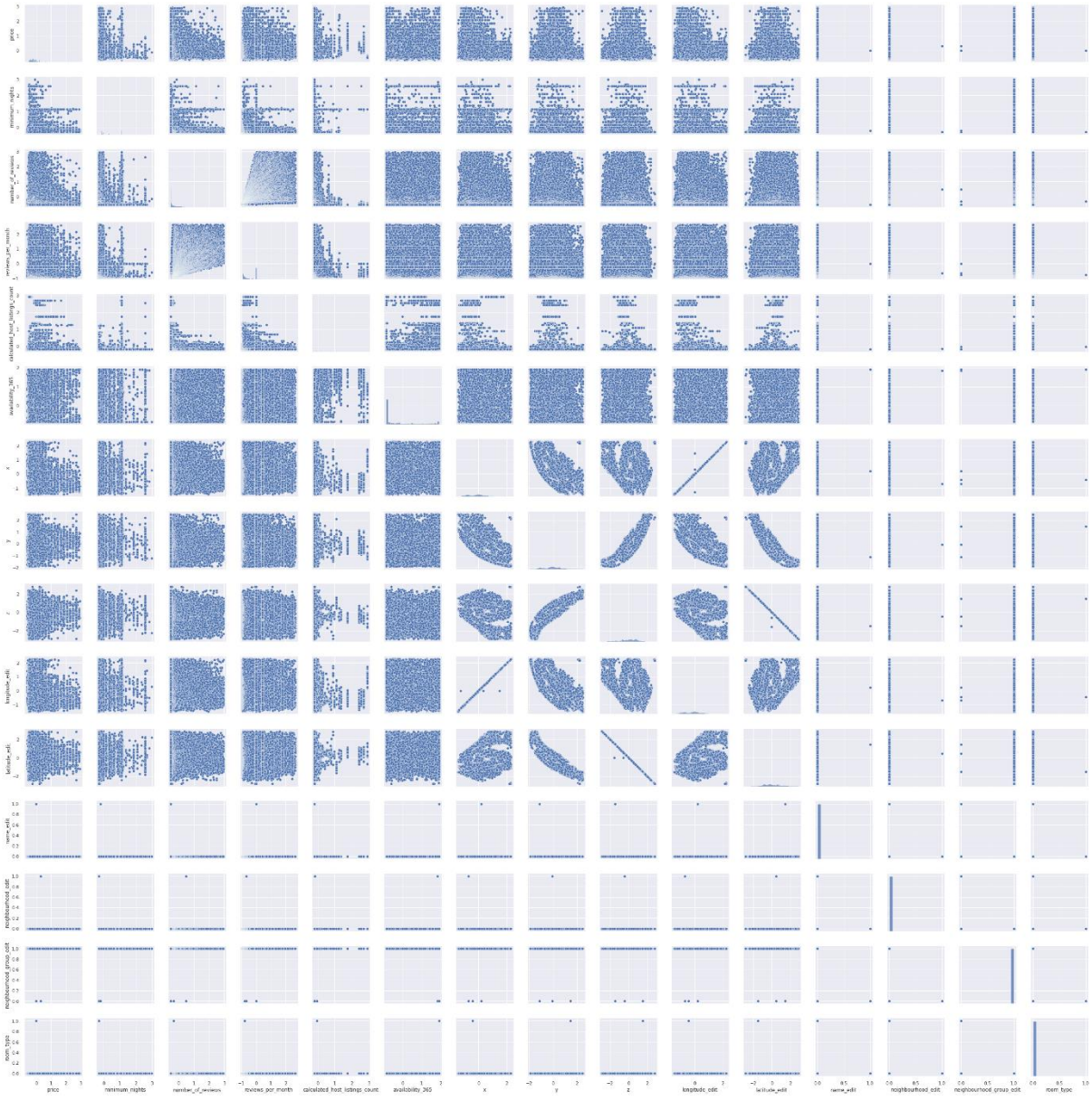
Relation between availability_365 and number of reviews (reverse relation):



Price in data places:




3
2
1
0
−1
−2
−3
Depth (km)
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Longitude
2
1
0
−1
−2
Latitude


price
minimum nights
number_of_reviews
reviews_per_month
calculated_host_listings_count
availability_365
x
y
z
longitude_edit
latitude_edit
name_edit
neighbourhood_edit
neighbourhood group_edit
room type

Hypothesis tests

1. Normality test:

* Shapiro-Wilk Normality Test : showed price distribution Probably is not Gaussian
* Anderson-Darling Normality Test: showed that price:

  Stat=2230.099

  Probably not Gaussian at the 15.0% level

  Probably not Gaussian at the 10.0% level

  Probably not Gaussian at the 5.0% level

  Probably not Gaussian at the 2.5% level

  Probably not Gaussian at the 1.0% level

2. Correlation test:

* The Pearson's Correlation test : showed price and room type with coefficient p=0.936 are Probably independent
* The Spearman's Rank Correlation Test : showed price and latitude Probably are dependent

3. Stationary Tests

* Augmented Dickey-Fuller test: showed minimum night feature with stat=-32.202, p=0.000 is Probably Stationary
* The Kwiatkowski-Phillips-Schmidt-Shin test : showed calculated host listings with stat=3.733, p=0.010 is Probably not Stationary

4. Parametric Statistical Hypothesis Tests

* Student's t-test : showed price and calculated_host_listings_count with stat=18.573, p=0.000 Probably have different distributions

* The Analysis of Variance Test showed price and availability-365 with stat=120.396, p=0.000 Probably have different distributions

5. Nonparametric Statistical Hypothesis Tests

* The Mann-Whitney U Test showed coastal_neig and host_name with stat=2732992.000, p=0.000 Probably have different distributions
* The Wilcoxon Signed-Rank Test showed price and x (latitude) with stat=442061720.000, p=0.000 Probably have different distributions
* The Kruskal-Wallis H Test: showed that host id and host_name with stat=63995.120, p=0.000 Probably have different distributions

2. Apartment Rental Offers In Germany Data analysis

What we did in this homework:

1. Processing data

    * Delete duplicates
    * Manage missing data (Data cleaning)
        - Clean feature column if it has more than 50% Null values
        - Fill Null values of numerical data by mean value of feature
        - Fill Null values of categorical data by most frequents
        - Fill empty dates
    * Data normalization and removing the outliers
    * Removing columns that doesn't have useful info
    * Removing columns that doesn't have high correlation to data
    * Reduce features by grouping
2. Create virtualization to have a better understanding of the data
3. Provide general information in aggregate mode about ads such as number of ads, number of Ads in each geographical area, review of general price indices and ...
4. If the number of comments for an Ad can be considered an indicator of the number of customers finding the owners of the ad who have the most customers and investigate the causes
5. Perform arbitrary hypothesis tests on data and respond to and interpret them
6. Build a model to predict parameters such as price and presentation of these models and their interpretation.
7. Feature engineering from the original variable (totalRent) to create a better model
8. Trying PysPark on dataset
9. Time calculation for each part

Data overview

The dataset name: apartment-rental-offers-in-Germany, immo_data.csv

Number of features in dataset: 49

Number of records in dataset: 268850

Data Overview

Import all important libraries

```
In [90]:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import plotly.express as px
```

```
In [91]:
pd.set_option('display.max_columns',None)  #to show all columns
imo=pd.read_csv(root+"immo_data.csv")  #reading dataset
```

```
In [92]:
s_time = time.time()
print("The dataset name : apartment-rental-offers-in-germany ,immo_data.csv")
print("Number of features in dataset: ", len(imo.columns))
print("Number of records in dataset: ", len(imo))
print(f'run time : {time.time() - s_time}')
```

```
The dataset name : apartment-rental-offers-in-germany ,immo_data.csv
Number of features in dataset:  49
Number of records in dataset:  268850
run time : 0.0008890628814697266
```

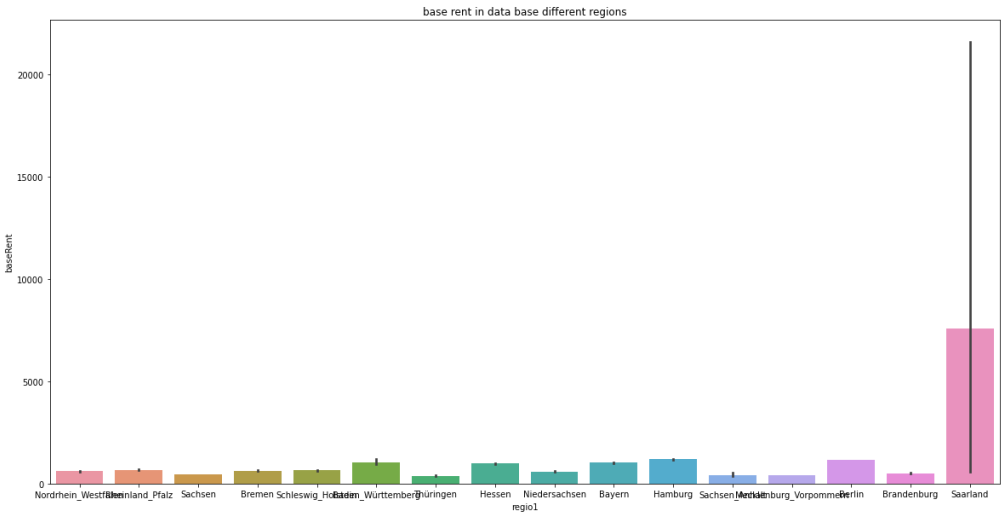
```
In [93]:
imo.head(3)
```

Out[93]:

	regio1	serviceCharge	heatingType	telekomTvOffer	telekomHybridUploadSpeed	newlyConst	balcony	picture

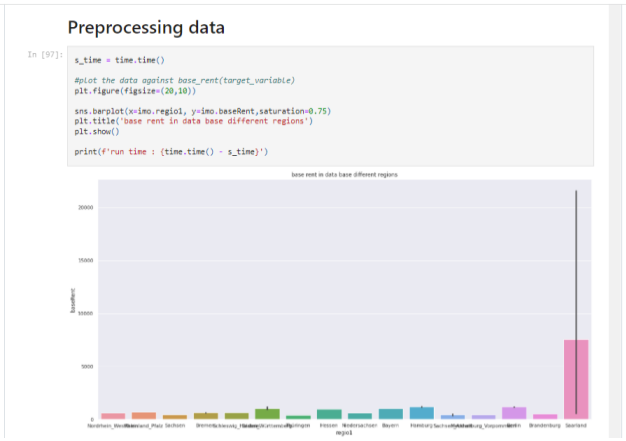
```
In [94]:
s_time = time.time()
print("list of features and their types:")
print('-------------------------------------------')
imo.info()
print(f'run time : {time.time() - s_time}')
```

```
list of features and their types:
-------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 268850 entries, 0 to 268849
Data columns (total 49 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   regio1                  268850 non-null  object
 1   serviceCharge           261941 non-null  float64
 2   heatingType             223994 non-null  object
 3   telekomTvOffer          236231 non-null  object
 4   telekomHybridUploadSpeed  45020 non-null   float64
 5   newlyConst              268850 non-null  bool
 6   balcony                 268850 non-null  bool
 7   picturecount            268850 non-null  int64
 8   pricetrend              267018 non-null  float64
 9   telekomUploadSpeed      235492 non-null  float64
 10  totalRent               228333 non-null  float64
 11  yearConstructed         211805 non-null  float64
 12  scoutId                 268850 non-null  int64
 13  noParkSpaces            93052 non-null   float64
 14  firingTypes             211886 non-null  object
 15  hasKitchen              268850 non-null  bool
 16  geo_bln                 268850 non-null  object
 17  cellar                  268850 non-null  bool
 18  yearConstructedRange    211805 non-null  float64
 19  baseRent                268850 non-null  float64
 20  houseNumber             197832 non-null  object
 21  livingSpace             268850 non-null  float64
 22  geo_krs                 268850 non-null  object
 23  condition               200361 non-null  object
```

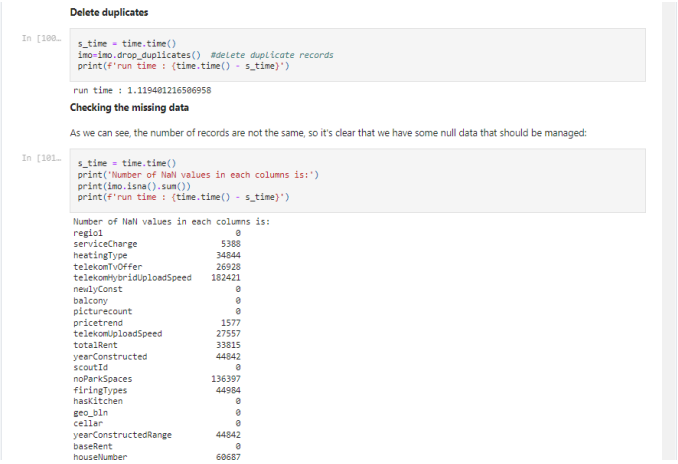


Preprocessing data

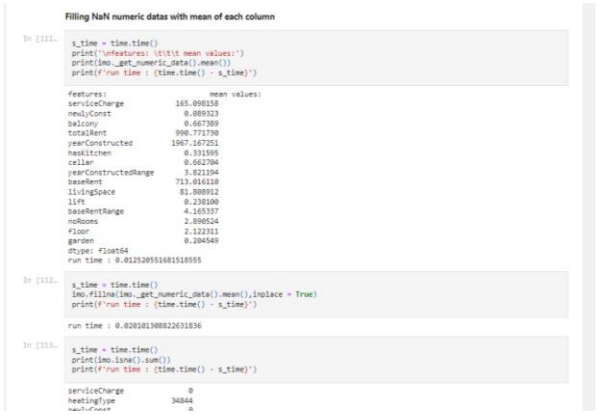
Most data preprocessing steps (not all) are similar to Exercise 1, including:



* Finding and deleting duplicate data



* finding null data and filling in the numeric null data with the average of each column



* filling in the categorical data with the most frequent data column

* delete columns that contain more than 30% of null data

Delete culomns with more than %30 null values (if such columns exist):

```
In [102…
s_time = time.time()
print('\nfeatures:\t\tpercent of null values:')
print(imo.isna().sum()/len(imo))
print(f'run time : {time.time() - s_time}')
```

```
features:               percent of null values:
regio1                      0.000000
serviceCharge               0.024676
heatingType                 0.159576
telekomTvOffer              0.123323
telekomHybridUploadSpeed    0.835437
newlyConst                  0.000000
balcony                     0.000000
picturecount                0.000000
pricetrend                  0.007222
telekomUploadSpeed          0.126203
totalRent                   0.154863
yearConstructed             0.205364
scoutId                     0.000000
noParkSpaces                0.624660
firingTypes                 0.206014
haskitchen                  0.000000
geo_bln                     0.000000
cellar                      0.000000
yearConstructedRange        0.205364
baseRent                    0.000000
houseNumber                 0.277929
livingSpace                 0.000000
geo_krs                     0.000000
condition                   0.244264
interiorQual                0.409111
petsAllowed                 0.421701
street                      0.000000
streetPlain                 0.277916
lift                        0.000000
baseRentRange               0.000000
typeOfFlat                  0.125603
```

* delete columns that do not have useful information, and reduce the number of columns by grouping

Delete culomns without usefull information (if such columns exist):

```
In [108…
s_time = time.time()
print(imo.head(3))
print(f'run time : {time.time() - s_time}')
```

```
                regio1  serviceCharge                  heatingType  \
0  Nordrhein_Westfalen          245.0                central_heating
1      Rheinland_Pfalz          134.0  self_contained_central_heating
2              Sachsen          255.0                 floor_heating

  telekomTvOffer  newlyConst  balcony  picturecount  pricetrend  \
0  ONE_YEAR_FREE       False    False             6        4.62
1  ONE_YEAR_FREE       False     True             8        3.47
2  ONE_YEAR_FREE        True     True             8        2.72

   telekomUploadSpeed  totalRent  yearConstructed    scoutId firingTypes  \
0                10.0      840.0           1965.0   96107057         oil
1                10.0        NaN           1871.0  111378734         gas
2                 2.4     1300.0           2019.0  113147523         NaN

   hasKitchen              geo_bln  cellar  yearConstructedRange  baseRent  \
0       False  Nordrhein_Westfalen    True                   2.0     595.0
1       False      Rheinland_Pfalz   False                   1.0     800.0
2       False              Sachsen    True                   9.0     965.0
```

we have to predict rental price ('totaRent') so we should drop all the rows that doesn't deal with rental price

we have to predict rental price (totaRent) so we should drop all the rows that doesn't deal with rental price

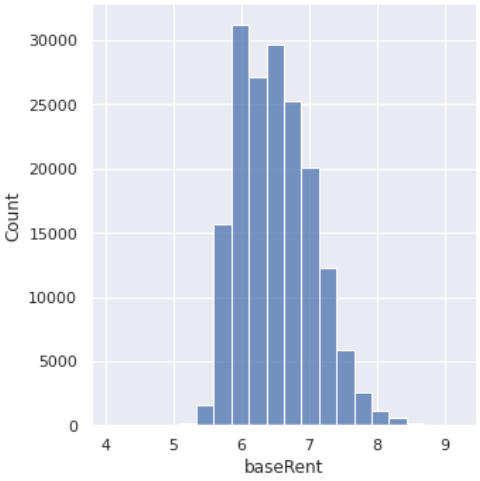
```
s_time = time.time()

imo=imo.drop(columns=['telekomTvOffer','pricetrend','livingSpaceRange','street','description',
                      'facilities','geo_krs','geo_plz','scoutId','regio1','telekomUploadSpeed',
                      'telekomTvOffer','pricetrend','regio3','noRoomsRange','picturecount','geo_bln','date',
                      'houseNumber','streetPlain','firingTypes'])
print(imo.shape)
print(f'run time : {time.time() - s_time}')
```

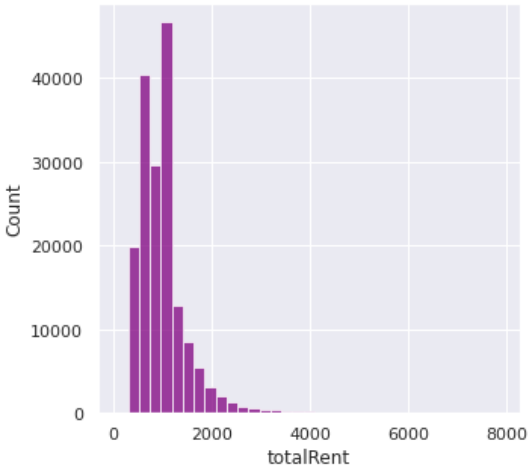
```
(218354, 19)
run time : 0.027890682220458984
```

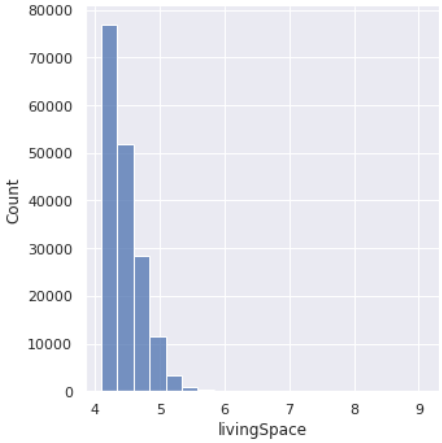
```
s_time = time.time()

print('Number of NaN values in each columns :')
print(imo.isna().sum())
print(f'run time : {time.time() - s_time}')
```

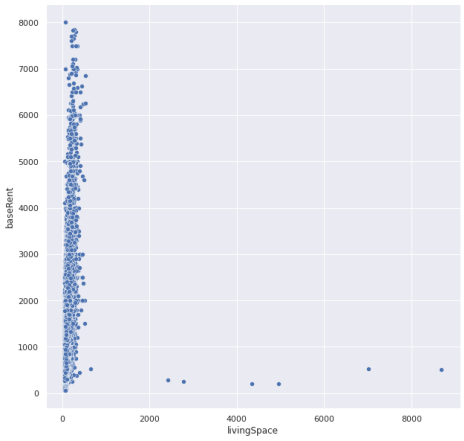
Next step we delete outlier data.

Here is some visualization about data:

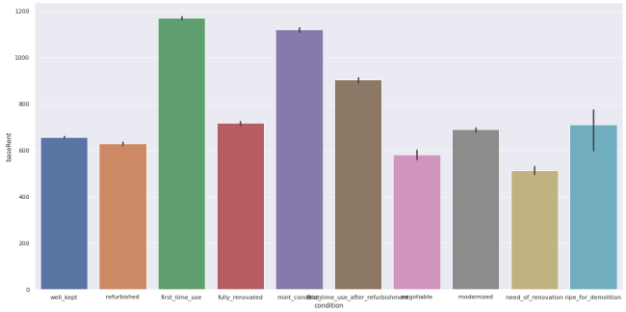


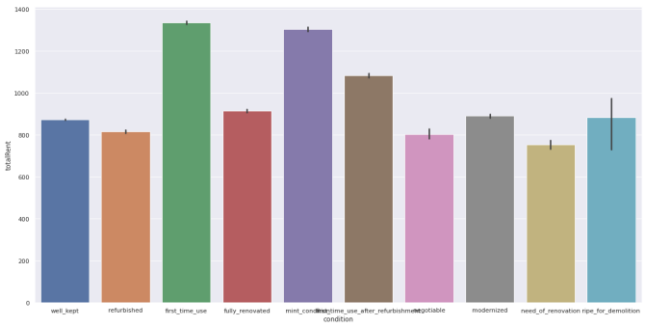




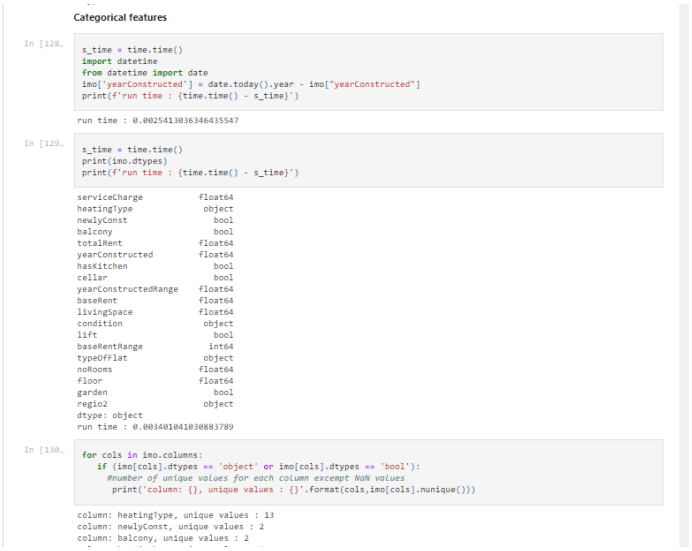


We can see the relation between living space and base rent.





Reducing categorical features:



Categorical features

In [128…
```
s_time = time.time()
import datetime
from datetime import date
imo['yearConstructed'] = date.today().year - imo["yearConstructed"]
print(f'run time : {time.time() - s_time}')
```
run time : 0.0025413036346435547

In [129…
```
s_time = time.time()
print(imo.dtypes)
print(f'run time : {time.time() - s_time}')
```
```
serviceCharge           float64
heatingType              object
newlyConst                 bool
balcony                    bool
totalRent               float64
yearConstructed         float64
hasKitchen                 bool
cellar                     bool
yearConstructedRange    float64
baseRent                float64
livingSpace             float64
condition                object
lift                       bool
baseRentRange             int64
typeOfFlat               object
noRooms                 float64
floor                   float64
garden                     bool
regio2                   object
dtype: object
run time : 0.0034010410308837891
```

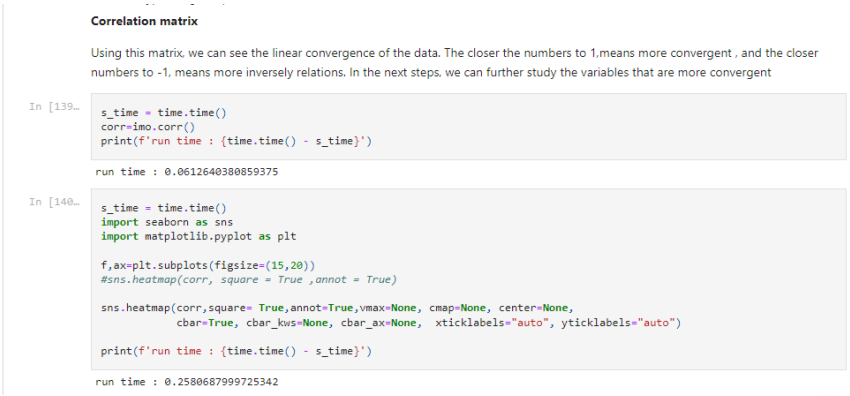
In [130…
```
for cols in imo.columns:
    if (imo[cols].dtypes == 'object' or imo[cols].dtypes == 'bool'):
        #number of unique values for each column excempt NaN values
        print('column: {}, unique values : {}'.format(cols,imo[cols].nunique()))
```
```
column: heatingType, unique values : 13
column: newlyConst, unique values : 2
column: balcony, unique values : 2
```

Reduce number of categories

In [131…
```
s_time = time.time()

otherscondition = imo['condition'].value_counts().tail(3).index

othersregion = list(imo['condition'].value_counts().tail(3).index)
def editcondition(dflist):
    if dflist in otherscondition:
        return 'Other'
    else:
        return dflist

imo['condition_edit'] =imo['condition'].apply(editcondition)
print(imo['condition_edit'].value_counts())

print(f'run time : {time.time() - s_time}')
```
```
well_kept                           82358
first_time_use                      18536
fully_renovated                     16659
mint_condition                      16465
refurbished                         15239
first_time_use_after_refurbishment  11310
modernized                          10708
Other                                2344
Name: condition_edit, dtype: int64
run time : 0.13721585273742676
```

In [132…
```
s_time = time.time()

othersheatingType = imo['heatingType'].value_counts().head(3).index

othersregion = list(imo['heatingType'].value_counts().head(3).index)
def editheatingType(dflist):
    if dflist in othersheatingType:
        return 'Other'
    else:
        return dflist

imo['heatingType_edit'] =imo['heatingType'].apply(editheatingType)

imo = imo.drop(columns = ['heatingType'])
imo['heatingType_edit'].value_counts()*100 / len(imo)
```

Correlation matrix

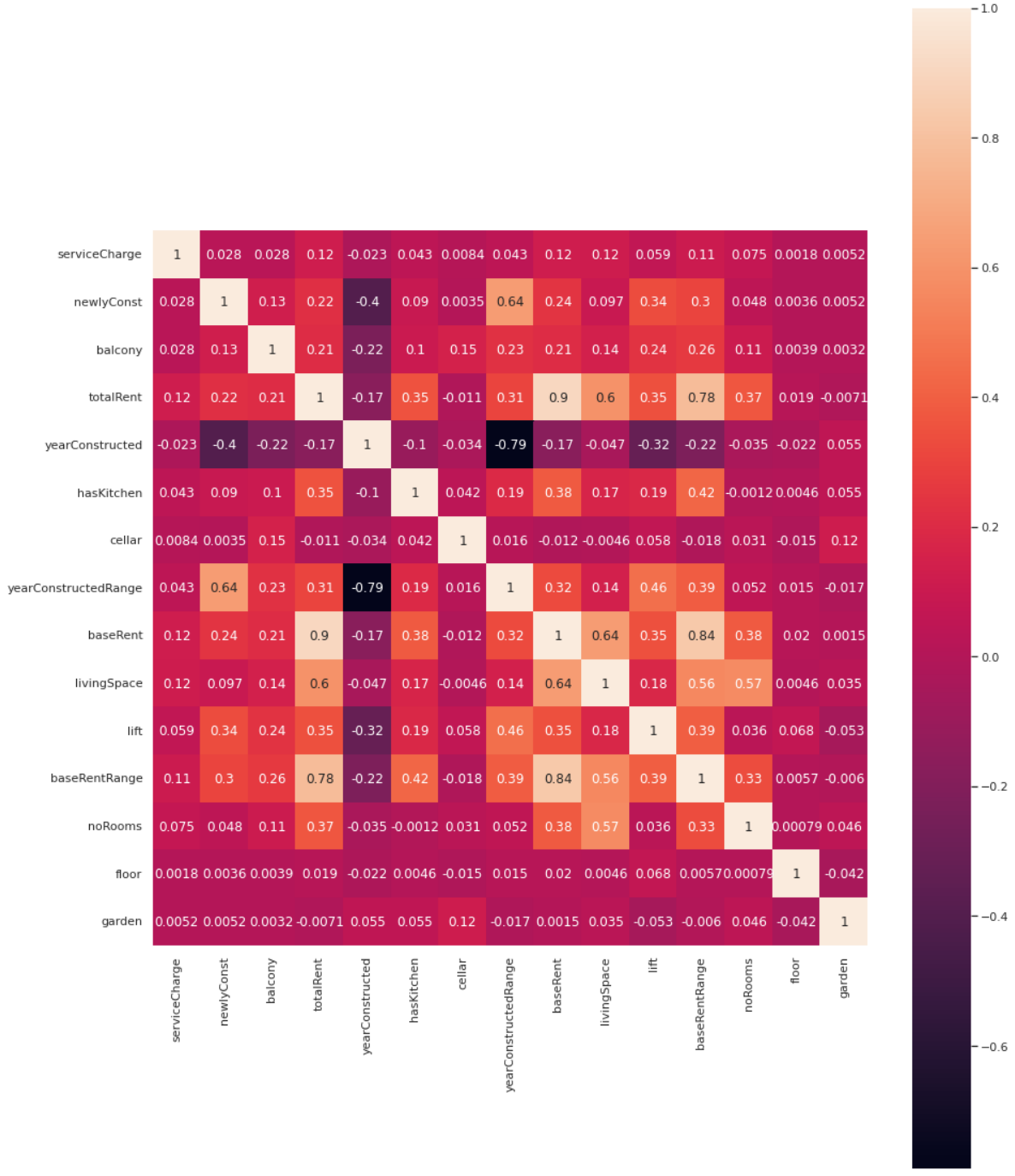


Correlation matrix

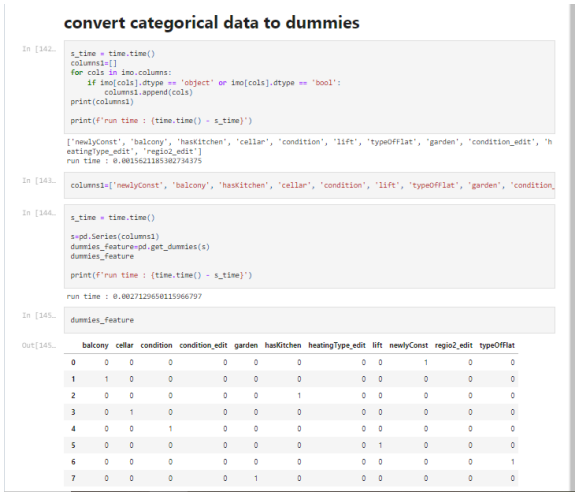
Using this matrix, we can see the linear convergence of the data. The closer the numbers to 1,means more convergent , and the closer numbers to -1, means more inversely relations. In the next steps, we can further study the variables that are more convergent

In [139…
```
s_time = time.time()
corr=imo.corr()
print(f'run time : {time.time() - s_time}')
```
run time : 0.0612640380859375

In [140…
```
s_time = time.time()
import seaborn as sns
import matplotlib.pyplot as plt

f,ax=plt.subplots(figsize=(15,20))
#sns.heatmap(corr, square = True ,annot = True)

sns.heatmap(corr,square= True,annot=True,vmax=None, cmap=None, center=None,
            cbar=True, cbar_kws=None, cbar_ax=None,  xticklabels="auto", yticklabels="auto")

print(f'run time : {time.time() - s_time}')
```
run time : 0.2580687999725342


serviceCharge 1 0.028 0.028 0.12 -0.023 0.043 0.0084 0.043 0.12 0.12 0.059 0.11 0.075 0.0018 0.0052
newlyConst 0.028 1 0.13 0.22 -0.4 0.09 0.0035 0.64 0.24 0.097 0.34 0.3 0.048 0.0036 0.0052
balcony 0.028 0.13 1 0.21 -0.22 0.1 0.15 0.23 0.21 0.14 0.24 0.26 0.11 0.0039 0.0032
totalRent 0.12 0.22 0.21 1 -0.17 0.35 -0.011 0.31 0.9 0.6 0.35 0.78 0.37 0.019 -0.0071
yearConstructed -0.023 -0.4 -0.22 -0.17 1 -0.1 -0.034 -0.79 -0.17 -0.047 -0.32 -0.22 -0.035 -0.022 0.055
hasKitchen 0.043 0.09 0.1 0.35 -0.1 1 0.042 0.19 0.38 0.17 0.19 0.42 -0.0012 0.0046 0.055
cellar 0.0084 0.0035 0.15 -0.011 -0.034 0.042 1 0.016 -0.012 -0.0046 0.058 -0.018 0.031 -0.015 0.12
yearConstructedRange 0.043 0.64 0.23 0.31 -0.79 0.19 0.016 1 0.32 0.14 0.46 0.39 0.052 0.015 -0.017
baseRent 0.12 0.24 0.21 0.9 -0.17 0.38 -0.012 0.32 1 0.64 0.35 0.84 0.38 0.02 0.0015
livingSpace 0.12 0.097 0.14 0.6 -0.047 0.17 -0.0046 0.14 0.64 1 0.18 0.56 0.57 0.0046 0.035
lift 0.059 0.34 0.24 0.35 -0.32 0.19 0.058 0.46 0.35 0.18 1 0.39 0.036 0.068 -0.053
baseRentRange 0.11 0.3 0.26 0.78 -0.22 0.42 -0.018 0.39 0.84 0.56 0.39 1 0.33 0.0057 -0.006
noRooms 0.075 0.048 0.11 0.37 -0.035 -0.0012 0.031 0.052 0.38 0.57 0.036 0.33 1 0.00079 0.046
floor 0.0018 0.0036 0.0039 0.019 -0.022 0.0046 -0.015 0.015 0.02 0.0046 0.068 0.0057 0.00079 1 -0.042
garden 0.0052 0.0052 0.0032 -0.0071 0.055 0.055 0.12 -0.017 0.0015 0.035 -0.053 -0.006 0.046 -0.042 1
serviceCharge
newlyConst
balcony
totalRent
yearConstructed
hasKitchen
cellar
yearConstructedRange
baseRent
livingSpace
lift
baseRentRange
noRooms
floor
garden
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

Convert categorical data to dummies:



convert categorical data to dummies

```
s_time = time.time()
columns1=[]
for cols in imo.columns:
    if imo[cols].dtype == 'object' or imo[cols].dtype == 'bool':
        columns1.append(cols)
print(columns1)

print(f'run time : {time.time() - s_time}')
```

```
['newlyConst', 'balcony', 'hasKitchen', 'cellar', 'condition', 'lift', 'typeOfFlat', 'garden', 'condition_edit', 'heatingType_edit', 'regio2_edit']
run time : 0.0015621185302734375
```

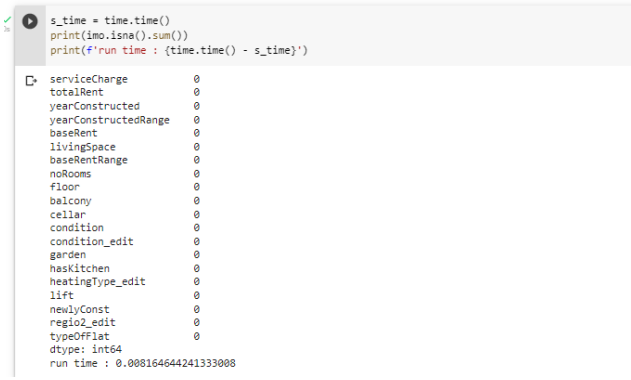
```
columns1=['newlyConst', 'balcony', 'hasKitchen', 'cellar', 'condition', 'lift', 'typeOfFlat', 'garden', 'condition_
```

```
s_time = time.time()

s=pd.Series(columns1)
dummies_feature=pd.get_dummies(s)
dummies_feature

print(f'run time : {time.time() - s_time}')
```

```
run time : 0.002712965011596679
```

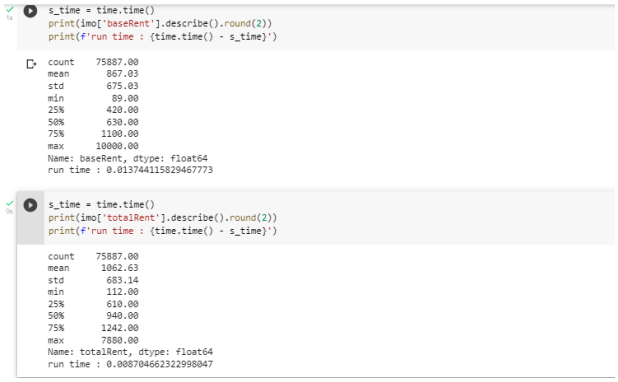
```
dummies_feature
```

	balcony	cellar	condition	condition_edit	garden	hasKitchen	heatingType_edit	lift	newlyConst	regio2_edit	typeOfFlat
0	0	0	0	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	1	0	0	0
6	0	0	0	0	0	0	0	0	0	0	1
7	0	0	0	0	1	0	0	0	0	0	0



```
s_time = time.time()
print(imo.isna().sum())
print(f'run time : {time.time() - s_time}')
```

```
serviceCharge           0
totalRent               0
yearConstructed         0
yearConstructedRange    0
baseRent                0
livingSpace             0
baseRentRange           0
noRooms                 0
floor                   0
balcony                 0
cellar                  0
condition               0
condition_edit          0
garden                  0
hasKitchen              0
heatingType_edit        0
lift                    0
newlyConst              0
regio2_edit             0
typeOfFlat              0
dtype: int64
run time : 0.008164644241333008
```



```
s_time = time.time()
print(imo['baseRent'].describe().round(2))
print(f'run time : {time.time() - s_time}')
```

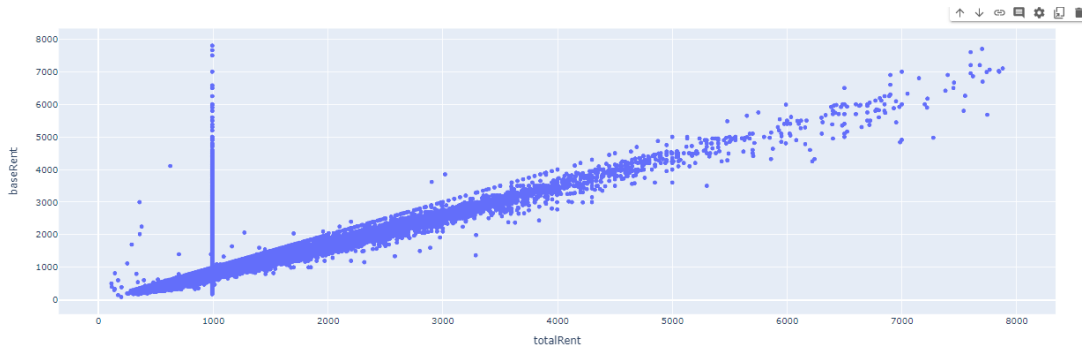
```
count    75887.00
mean       867.03
std        675.03
min         89.00
25%        420.00
50%        630.00
75%       1100.00
max      10000.00
Name: baseRent, dtype: float64
run time : 0.013744115829467773
```



```
s_time = time.time()
print(imo['totalRent'].describe().round(2))
print(f'run time : {time.time() - s_time}')
```

```
count    75887.00
mean      1062.63
std        683.14
min        112.00
25%        610.00
50%        940.00
75%       1242.00
max       7880.00
Name: totalRent, dtype: float64
run time : 0.008704662322998047
```

Scatter plot of base rent and total rent after deleting outliers:



PCA

PCA

Principal Component Analysis is basically a statistical procedure to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables

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

```
[394] s_time = time.time()
      print(imo.reset_index)
      print(f'run time : {time.time() - s_time}')
```

```
<bound method DataFrame.reset_index of         serviceCharge   totalRent  yearConstructed  yearConstructedRange  \
0              245.00   840.00000        57.000000             2.000000
1              245.00   840.00000        57.000000             2.000000
2              255.00  1300.00000         3.000000             9.000000
3              255.00  1300.00000         3.000000             9.000000
4              138.00   903.00000        72.000000             1.000000
...               ...         ...              ...                  ...
268839         100.00  1980.00000       122.000000             1.000000
268840         112.13  1479.64000         6.000000             9.000000
268844          80.00   670.00000        54.832749             3.821194
268848         175.00  1015.00000         3.000000             9.000000
268849         315.00   990.77173        50.000000             3.000000
```

0s completed at 2:44 AM

Split features and target:

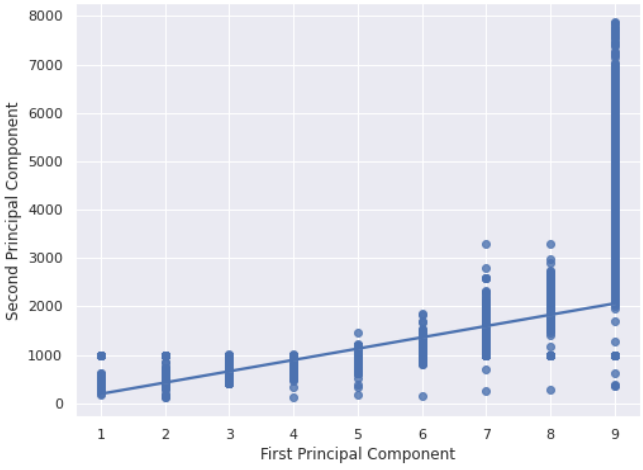
Split features and target

```
s_time = time.time()

# Splitting data into y as target and x as features
y= imo['totalRent']
X= imo.drop(columns=['totalRent'])
print('splitting target and features\n')
print(X.dtypes)
print(f'run time : {time.time() - s_time}')
```

```
splitting target and features

serviceCharge          float64
yearConstructed        float64
yearConstructedRange   float64
baseRent               float64
livingSpace            float64
baseRentRange          float64
noRooms                float64
floor                  float64
balcony                float64
cellar                 float64
condition              float64
condition_edit         float64
garden                 float64
hasKitchen             float64
```

The result:

```
s_time = time.time()
# giving a larger plot
plt.figure(figsize =(8, 6))

sns.set(rc={'figure.figsize':(15,10)})
sns.regplot(x='baseRentRange',y='totalRent', data=imo)

# labeling x and y axes
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.show()

print(f'run time : {time.time() - s_time}')
```

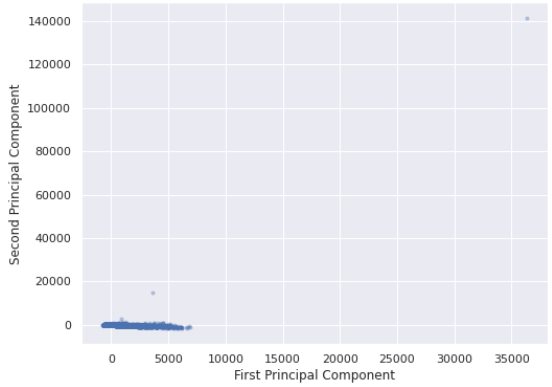


```
[400] s_time = time.time()
      from sklearn.decomposition import PCA
      pca = PCA(0.70)
      x_pca =pca.fit_transform(X)
      print(x_pca.shape)
      print(x_pca[:1])

      print(f'run time : {time.time() - s_time}')
```

```
(75884, 2)
[[-248.81220443  122.54702461]]
run time : 0.3728296756744385
```

```
s_time = time.time()

# giving a larger plot
plt.figure(figsize =(8, 6))
plt.scatter(x_pca[:,0], x_pca[:,1],marker='.',cmap ='plasma',alpha=0.3)

# labeling x and y axes
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.show()

print(f'run time : {time.time() - s_time}')
```

the result plot:




```
s_time = time.time()

from sklearn.decomposition import PCA
pca = PCA(0.80)
x_pca =pca.fit_transform(X)
print(x_pca.shape)
print(x_pca[:1])
print(f'run time : {time.time() - s_time}')
```

```
(75884, 2)
[[-248.81220443  122.54702461]]
run time : 0.07271289825439453
```

```
s_time = time.time()

from sklearn.decomposition import PCA
pca = PCA(0.90)
x_pca =pca.fit_transform(X)
print(x_pca.shape)
print(x_pca[:1])
print(f'run time : {time.time() - s_time}')
```

```
(75884, 2)
[[-248.81220443  122.54702461]]
run time : 0.07345056533813477
```

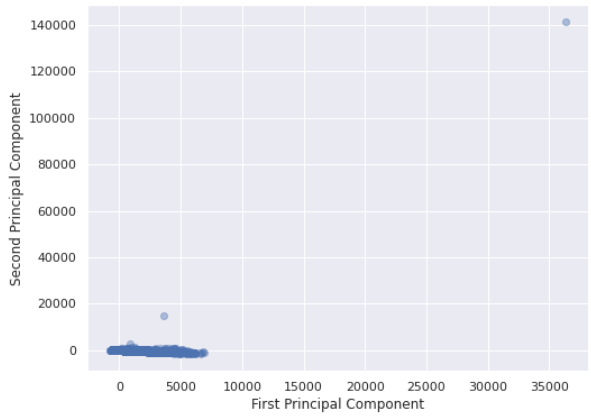
```
s_time = time.time()

from sklearn.decomposition import PCA
pca = PCA(0.99)
x_pca =pca.fit_transform(X)
print(x_pca.shape)
print(x_pca[:1])
print(f'run time : {time.time() - s_time}')
```

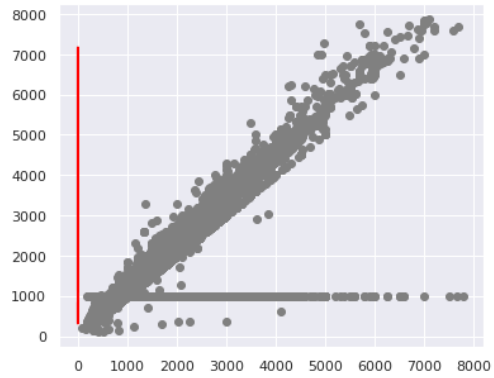
```
(75884, 2)
[[-248.81220443  122.54702461]]
run time : 0.0753939151763916
```

```
s_time = time.time()
# giving a larger plot
plt.figure(figsize =(8, 6))
plt.scatter(x_pca[:, 0], x_pca[:, 1], cmap ='plasma',alpha=0.4)

# labeling x and y axes
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')

print(f'run time : {time.time() - s_time}')
```

```
run time : 0.0278475284576416
```

the result plot:





Linear regression:

linear regression

```
[411] s_time = time.time()
      import numpy as np
      from sklearn.linear_model import LinearRegression

      x = np.array(imo['totalRent']).reshape((-1, 1))
      y = np.array(X)

[413] s_time = time.time()
      model = LinearRegression()
      print(f'run time : {time.time() - s_time}')

      run time : 0.0001101493835449218

      s_time = time.time()
      model.fit(x, y)
      model = LinearRegression().fit(x, y)

      print(f'run time : {time.time() - s_time}')

      run time : 0.08509969711303711
```

Get results

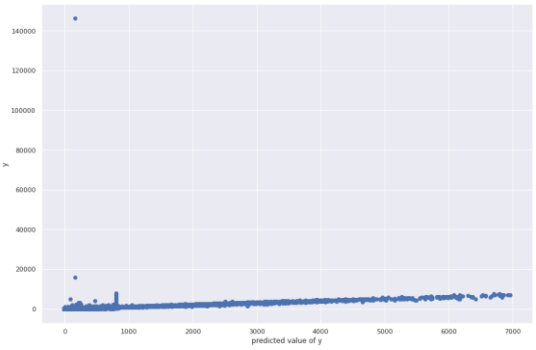
properties of the model

```
s_time = time.time()
r_sq = model.score(x, y)
print('coefficient of determination:', r_sq)
print('intercept:')
print(model.intercept_)
print('slope:')
print(model.coef_)
print(f'run time : {time.time() - s_time}')

coefficient of determination: 0.10881898717800491
intercept:
[ 8.64283250e+01  7.10347016e+01  2.43177998e+00 -8.27744493e+01
  5.42079582e+01  1.95612336e+00  2.53255747e+00  2.21262602e+00
  1.98581055e-05  6.05573737e-06  1.79677812e-05  2.75319221e-05
  2.75319221e-05  6.05573737e-06  1.53341774e-05  1.79677812e-05
  1.98581055e-05  9.99808178e-01  3.36604736e-05]
slope:
[[ 9.62195499e-02]
 [-1.09548398e-02]
 [ 1.16352258e-03]
 [ 8.93521907e-01]
 [ 3.14323366e-02]
 [ 2.60355920e-03]
 [ 4.59023486e-04]
 [ 1.40757324e-04]
```

The result plot:

Linear Regression

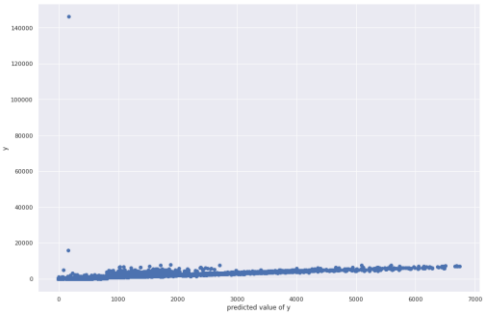
```
from pyparsing.core import LineStart
s_time = time.time()

from sklearn import metrics
prediction = []
def linearregression(X, y_pred, x, y):
    linreg = LinearRegression()
    linreg.fit(X, y_pred)
    y_pred = linreg.predict(X)

print('MAE:', metrics.mean_absolute_error(y, y_pred))
print('MSE:', metrics.mean_squared_error(y, y_pred))
print('R2_score:', metrics.r2_score(y, y_pred))

plt.scatter(y_pred,y)
plt.xlabel('predicted value of y')
plt.ylabel('y')

linearregression(X, y_pred, x, y)
print(f'run time : {time.time() - s_time}')

MAE: 11.217060927099983
MSE: 19671.452668274964
R2_score: 0.10881898717800491
run time : 0.32375025749206543
```



Random Forest Regression

Random Forest Regresson

```
s_time = time.time()

from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics

def RandomForest(X, y_pred, x, y):
    randomForest = RandomForestRegressor()
    randomForest.fit(X, y_pred)
    y_pred = randomForest.predict(X)

    print('MAE:', metrics.mean_absolute_error(y, y_pred))
    print('MSE:', metrics.mean_squared_error(y, y_pred))
    print('R2_score:', metrics.r2_score(y, y_pred))

    plt.scatter(y_pred,y,cmap ='plasma')
    plt.xlabel('predicted value of y')
    plt.ylabel('y')
    plt.figure()

RandomForest(X, y_pred, x, y)

print(f'run time : {time.time() - s_time}')

MAE: 5.547949265060455
MSE: 15436.752154712995
R2_score: 0.12657765340452643
run time : 55.42402055892944
```



Polynomial regression

```
[420] import numpy as np
      from sklearn.linear_model import LinearRegression
      from sklearn.preprocessing import PolynomialFeatures

      x = np.array(imo['totalRent']).reshape((-1, 1))
      y = np.array(X)
```

```
[421] s_time = time.time()
      transformer = PolynomialFeatures(degree=2, include_bias=False)
      print(f'run time : {time.time() - s_time}')

      run time : 9.72747802734375e-05
```

```
[422] s_time = time.time()
      transformer.fit(x)   #we need to fit before transfer
      print(f'run time : {time.time() - s_time}')

      run time : 0.0007755756378173828
```

```
[424] s_time = time.time()
      x_ = transformer.transform(x)
      x_ = PolynomialFeatures(degree=2, include_bias=False).fit_transform(x)
      print(x_)
      print(f'run time : {time.time() - s_time}')

      [[8.40000000e+02 7.05600000e+05]
       [8.40000000e+02 7.05600000e+05]
       [1.30000000e+03 1.69000000e+06]
       ...
       [6.70000000e+02 4.48900000e+05]
       [1.01500000e+03 1.03022500e+06]
       [9.90771730e+02 9.81628621e+05]]
      run time : 0.004843235015869141
```

Properties of the model:

```
s_time = time.time()
r_sq = model.score(x_, y)
print('coefficient of determination:', r_sq)
print('intercept:')
print(model.intercept_)
print('coefficients:')
print(model.coef_)
print(f'run time : {time.time() - s_time}')
```

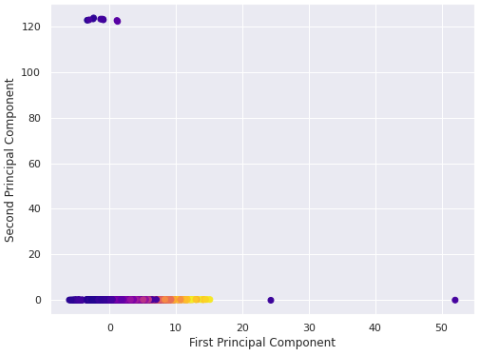
```
coefficient of determination: 0.12054695054558744
intercept:
[ 9.05010317e+01  8.25946248e+01  1.35353548e+00 -9.21291149e+01
  5.19470635e+01 -6.64213052e-02  2.47039957e+00  2.31203141e+00
  2.02433166e-05 -1.35893138e-05  1.50471282e-05  4.31710576e-05
  4.31710576e-05 -1.35893138e-05  8.10545765e-06  1.50471282e-05
  2.02433166e-05  9.99798555e-01  6.35952653e-05]
coefficients:
[[ 9.01507005e-02  1.48898264e-06]
 [-2.81805912e-02  4.22631096e-06]
 [ 2.77024362e-03 -3.94206476e-07]
 [ 9.07461545e-01 -3.42006821e-06]
 [ 3.48013564e-02 -8.26583708e-07]
 [ 5.61740724e-03 -7.39442864e-07]
 [ 5.51646632e-04 -2.27249427e-08]
 [-7.36930913e-06  3.63426358e-08]
 [-6.86038434e-09  1.40833292e-13]
 [ 3.59761009e-08 -7.18223591e-12]
 [-1.55319361e-10 -1.06779148e-12]
 [-3.68121866e-08  5.71767205e-12]
 [-3.68121866e-08  5.71767205e-12]
 [ 3.59761009e-08 -7.18223591e-12]
 [ 8.74262526e-09 -2.64282186e-12]
 [-1.55319361e-10 -1.06779148e-12]
 [-6.86038434e-09  1.40833292e-13]
 [ 7.08428144e-08 -3.51830178e-12]
 [-6.38818608e-08  1.09441677e-11]]
run time : 0.13203907012939453
```

Fitting logistic regression to dataset:

Fitting Logistic Regression To the dataset

```
s_time = time.time()
# importing standardscalar module
from sklearn.preprocessing import StandardScaler

scalar = StandardScaler()
df=imo._get_numeric_data()

# fitting
scalar.fit(df)
scaled_data = scalar.transform(df)

# Importing PCA
from sklearn.decomposition import PCA

# Let's say, components = 2
pca = PCA(n_components = 2)
pca.fit(scaled_data)
x_pca = pca.transform(scaled_data)

print(x_pca.shape)
print(f'run time : {time.time() - s_time}')
```

```
(75884, 2)
run time : 0.3348081111907959
```

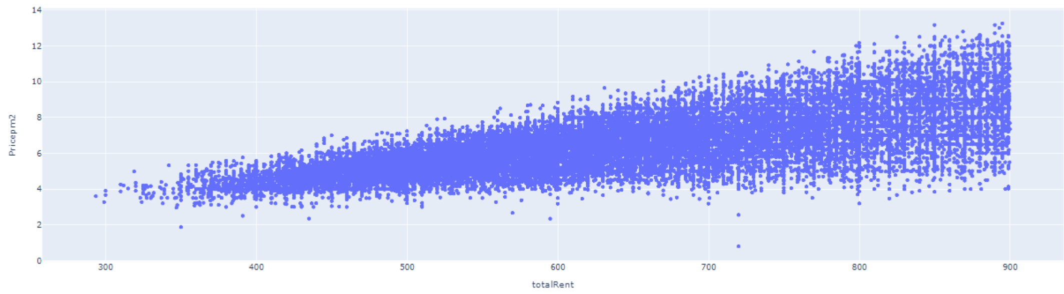


Feature Engineering

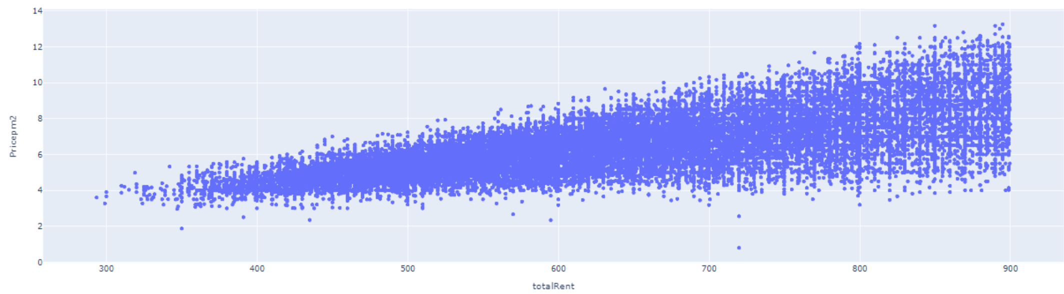
In this part, we are trying to create more variables for inspect and building a model from new variables later. For instance, we create new columns for the price per square meter and additional cost.

```
[447] imo['Pricepm2'] = imo['baseRent'] / imo['livingSpace']
      imo['additioncost'] = imo['totalRent'] - imo['baseRent']
```

```
fig = px.scatter(imo, x='totalRent', y='Pricepm2')
fig.show()
```




```
imo['Pricepm2'] = imo['baseRent'] / imo['livingSpace']
imo['additioncost'] = imo['totalRent'] - imo['baseRent']
fig = px.scatter(imo, x='totalRent', y='Pricepm2')
fig.show()
```



Multi-processing using PysPark:

```
from multiprocessing import Pool, cpu_count
```

```
[461] cpu_count()
```

2

```
s_time = time.time()

spark = SparkSession.builder.getOrCreate()

sdf = spark.createDataFrame(imo)
imo['yearConstructedRange'].round(3)
sdf.show()

print(f'run time : {time.time() - s_time}')
```

serviceCharge	totalRent	yearConstructed	yearConstructedRange	baseRent	livingSpace	baseRentRange	noRooms	floor	balcony	cellar	condition	condition_edit	garden	hasKitchen	heatingType_edit	lift	newlyConst
245.0	840.0	57.0	2.0	595.0	86.0	4.0	4.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
245.0	840.0	57.0	2.0	595.0	86.0	4.0	4.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
70.0	380.0	54.832749319931736	3.821193923186869	310.0	62.0	2.0	2.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
88.0	584.25	63.0	2.0	452.25	60.3	3.0	3.0	2.122310554803789	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
88.0	584.25	63.0	2.0	452.25	60.3	3.0	3.0	2.122310554803789	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
155.0	555.0	130.0	1.0	400.0	80.0	2.0	3.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
130.0	445.18	46.0	3.0	315.18	60.0	2.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
150.0	570.0	54.832749319931736	3.821193923186869	400.0	70.0	2.0	3.5	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
170.0	535.0	117.0	1.0	365.0	79.0	2.0	3.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
156.0	501.0	54.832749319931736	3.821193923186869	340.0	65.0	2.0	3.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
97.0	493.8	64.0	2.0	396.8	62.0	2.0	2.0	2.122310554803789	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
200.0	760.0	24.0	5.0	560.0	115.0	4.0	5.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
215.0	615.0	54.832749319931736	3.821193923186869	400.0	86.0	2.0	4.0	2.122310554803789	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
250.0	890.0	108.0	1.0	640.0	128.0	5.0	3.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
169.0	701.0	43.0	3.0	532.0	66.54	4.0	3.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
200.0	715.0	102.0	1.0	515.0	102.3	4.0	3.0	2.122310554803789	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50.0	600.0	26.0	5.0	500.0	73.98	3.0	2.0	2.122310554803789	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
76.0	483.0	38.0	4.0	315.0	65.45	2.0	4.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
175.99	509.79	58.0	2.0	333.8	60.69	2.0	3.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

```
s_time=time.time()
sdf.groupby('totalRent').avg().show()
print(f'run time : {time.time() - s_time}')
```

totalRent	avg(serviceCharge)	avg(totalRent)	avg(yearConstructed)	avg(yearConstructedRange)	avg(baseRent)	avg(livingSpace)	avg(baseRentRange)	avg(noRooms)	avg(floor)	avg(balcony)	avg(cellar)	avg(condit
837.78	189.41	837.78	146.0	1.0	648.37	72.85	5.0	3.0	1.0	0.0	0.0	
558.0	138.095	558.0	62.833187329982934	2.980298480796717	394.875	67.625	2.45	2.6375	1.9305776387009472	0.0	0.0	
496.0	126.3540042757745	496.0	70.90526334809151	2.551101224558968	347.3783783783784	66.88702702702703	1.972972972972973	2.7972972972972974	2.428545240098014	0.0	0.0	
596.0	140.38713226163222	596.0	54.807557535368865	3.266429366889277	420.43589743589746	73.38974358974359	2.717948717948718	2.9871794871794872	2.3809275354776034	0.0	0.0	
617.69	173.25	617.69	54.832749319931736	3.821193923186869	444.44	75.33	3.0	3.0	1.0	0.0	0.0	
799.52	270.0	799.52	54.832749319931736	3.821193923186869	529.52	117.67	4.0	3.0	3.0	0.0	0.0	
692.0	140.76923076923077	692.0	65.76909599690732	2.6895062899662006	512.5384615384615	71.64038461538462	3.4615384615384617	2.8653846153846154	2.086331581138753	0.0	0.0	
537.95	182.0	537.95	54.0	2.0	355.95	63.0	2.0	2.5	7.0	0.0	0.0	
735.19	134.0	735.19	42.0	3.0	507.19	75.7	4.0	3.5	1.0	0.0	0.0	
769.0	178.97222222222223	769.0	64.57394429331816	2.904709760708193	564.0277777777778	75.60888888888888	4.166666666666667	2.7777777777777777	1.569145617200421	0.0	0.0	
548.25	121.0	548.25	69.0	2.0	367.25	76.51	2.0	3.5	1.0	0.0	0.0	
875.12	109.39	875.12	42.0	3.0	667.28	109.39	5.0	5.0	1.0	0.0	0.0	
566.11	112.0	566.11	40.0	4.0	384.11	81.14	2.0	3.5	3.0	0.0	0.0	
507.55	160.0	507.55	42.0	3.0	347.55	66.2	2.0	3.0	1.0	0.0	0.0	
867.45	180.0	867.45	28.0	5.0	687.45	91.66	5.0	3.0	0.0	0.0	0.0	
787.13	201.31	787.13	60.0	2.0	585.82	68.92	4.0	2.0	3.0	0.0	0.0	
852.72	198.0	852.72	65.0	2.0	654.72	74.4	5.0	3.0	2.122310554803789	0.0	0.0	
558.5	190.0	558.5	45.0	3.5	368.5	73.0	2.0	2.75	1.0	0.0	0.0	
502.78	62.44	502.78	32.0	4.0	384.14	62.44	2.0	3.0	4.0	0.0	0.0	
684.71	146.0	684.71	27.0	5.0	458.71	63.71	3.0	2.0	2.0	0.0	0.0	

only showing top 20 rows

run time : 1.3233346939086914

Filter data:

Showing base rents less than 1500

```
s_time=time.time()
sdf.filter(sdf.baseRent < 1500).show()
print(f'run time : {time.time() - s_time}')
```

serviceCharge	totalRent	yearConstructed	yearConstructedRange	baseRent	livingSpace	baseRentRange	noRooms	floor	balcony	cellar	condition	condition_edit	garden	hasKitchen	heatingType_edit	lift	newlyConst	regio
245.0	840.0	57.0	2.0	595.0	86.0	4.0	4.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	
245.0	840.0	57.0	2.0	595.0	86.0	4.0	4.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
70.0	380.0	54.832749319931736	3.821193923186869	310.0	62.0	2.0	2.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
88.0	584.25	63.0	2.0	452.25	60.3	3.0	3.0	2.122310554803789	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
88.0	584.25	63.0	2.0	452.25	60.3	3.0	3.0	2.122310554803789	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
155.0	555.0	130.0	1.0	400.0	80.0	2.0	3.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
130.0	445.18	46.0	3.0	315.18	60.0	2.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
150.0	570.0	54.832749319931736	3.821193923186869	400.0	70.0	2.0	3.5	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
170.0	535.0	117.0	1.0	365.0	79.0	2.0	3.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
156.0	501.0	54.832749319931736	3.821193923186869	340.0	65.0	2.0	3.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
97.0	493.8	64.0	2.0	396.8	62.0	2.0	2.0	2.122310554803789	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
200.0	760.0	24.0	5.0	560.0	115.0	4.0	5.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
215.0	615.0	54.832749319931736	3.821193923186869	400.0	86.0	2.0	4.0	2.122310554803789	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
250.0	890.0	108.0	1.0	640.0	128.0	5.0	3.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
169.0	701.0	43.0	3.0	532.0	66.54	4.0	3.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
200.0	715.0	102.0	1.0	515.0	102.3	4.0	3.0	2.122310554803789	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
50.0	600.0	26.0	5.0	500.0	73.98	3.0	2.0	2.122310554803789	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
76.0	483.0	38.0	4.0	315.0	65.45	2.0	4.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
175.99	509.79	58.0	2.0	333.8	60.69	2.0	3.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
130.0	455.0	54.832749319931736	3.821193923186869	325.0	63.0	2.0	2.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

only showing top 20 rows

run time : 0.17018628120422363

Data schema:

run time : 0.17018628120422363

```
sdf.printSchema()
```

```
root
 |-- serviceCharge: double (nullable = true)
 |-- totalRent: double (nullable = true)
 |-- yearConstructed: double (nullable = true)
 |-- yearConstructedRange: double (nullable = true)
 |-- baseRent: double (nullable = true)
 |-- livingSpace: double (nullable = true)
 |-- baseRentRange: double (nullable = true)
 |-- noRooms: double (nullable = true)
 |-- floor: double (nullable = true)
 |-- balcony: double (nullable = true)
 |-- cellar: double (nullable = true)
 |-- condition: double (nullable = true)
 |-- condition_edit: double (nullable = true)
 |-- garden: double (nullable = true)
 |-- hasKitchen: double (nullable = true)
 |-- heatingType_edit: double (nullable = true)
 |-- lift: double (nullable = true)
 |-- newlyConst: double (nullable = true)
 |-- regio2_edit: double (nullable = true)
 |-- typeOfFlat: double (nullable = true)
 |-- Pricepm2: double (nullable = true)
 |-- additioncost: double (nullable = true)
```

Group data:

```
sdf.groupBy('livingSpace','baseRentRange').count().show()
```

livingSpace	baseRentRange	count
64.04	2.0	3
67.98	1.0	2
83.0	2.0	31
64.0	5.0	56
66.0	5.0	42
69.8	3.0	8
649.0	4.0	1
74.58	3.0	9
67.48	2.0	5
76.5	2.0	12
62.67	2.0	10
60.64	5.0	1
107.0	5.0	12
78.7	4.0	2
73.35	2.0	1
72.49	3.0	1
70.75	4.0	4
60.45	5.0	2
84.72	5.0	1
63.57	2.0	3

only showing top 20 rows

Ordering data by a column:

```
s_time=time.time()
sdf.orderBy("livingSpace").show()
print(f'run time : {time.time() - s_time}')
```

```
+-------------+---------+-----------------+--------------------+--------+-----------+-------------+-------+-----------------+-------+------+---------+--------------+------+----------+----------------+----+----------+-----
|serviceCharge|totalRent|   yearConstructed|yearConstructedRange|baseRent|livingSpace|baseRentRange|noRooms|            floor|balcony|cellar|condition|condition_edit|garden|hasKitchen|heatingType_edit|lift|newlyConst|regio
+-------------+---------+-----------------+--------------------+--------+-----------+-------------+-------+-----------------+-------+------+---------+--------------+------+----------+----------------+----+----------+-----
|        145.0|    582.5|54.832749319931736|   3.821193923186869|   437.5|       60.0|          3.0|    2.0|2.122310554803789|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        160.0|    790.0|54.832749319931736|   3.821193923186869|   630.0|       60.0|          5.0|    2.0|              2.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|         80.0|    416.0|             66.0|                 2.0|   336.0|       60.0|          2.0|    3.0|              1.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        120.0|    480.0|             25.0|                 5.0|   360.0|       60.0|          2.0|    2.0|              4.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        126.0|    405.0|             38.0|                 4.0|   279.0|       60.0|          1.0|    3.0|              1.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        200.0|    560.0|            122.0|                 1.0|   340.0|       60.0|          2.0|    2.0|              1.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        132.0|    397.0|            112.0|                 1.0|   265.0|       60.0|          1.0|    2.0|              2.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        125.0|    620.0|            122.0|                 1.0|   495.0|       60.0|          3.0|    2.0|              0.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        140.0|    420.0|            117.0|                 1.0|   280.0|       60.0|          1.0|    2.0|              2.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        130.0|    420.0|             61.0|                 2.0|   290.0|       60.0|          1.0|    3.0|              4.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        180.0|    760.0|54.832749319931736|   3.821193923186869|   580.0|       60.0|          4.0|    2.0|              1.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        140.0|    400.0|             41.0|                 4.0|   260.0|       60.0|          1.0|    3.0|              5.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        170.0|    500.0|             64.0|                 2.0|   330.0|       60.0|          2.0|    3.0|              1.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        120.0|    450.0|            122.0|                 1.0|   330.0|       60.0|          2.0|    2.0|2.122310554803789|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|         50.0|    400.0|            108.0|                 1.0|   300.0|       60.0|          1.0|    2.0|              2.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        140.0|    640.0|             11.0|                 8.0|   400.0|       60.0|          2.0|    3.5|              2.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        120.0|    720.0|54.832749319931736|   3.821193923186869|   600.0|       60.0|          4.0|    2.0|              2.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        110.0|    510.0|             59.0|                 2.0|   360.0|       60.0|          2.0|    2.0|              2.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        110.0|    830.0|             66.0|                 2.0|   630.0|       60.0|          5.0|    3.5|              3.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
|        110.0|    400.0|             36.0|                 4.0|   290.0|       60.0|          1.0|    3.0|              4.0|    0.0|   0.0|      0.0|           0.0|   0.0|       0.0|             0.0| 0.0|       0.0|
+-------------+---------+-----------------+--------------------+--------+-----------+-------------+-------+-----------------+-------+------+---------+--------------+------+----------+----------------+----+----------+-----
only showing top 20 rows

run time : 0.5294928550720215
```

Mask on data:

```
s_time=time.time()
from pyspark.sql.functions import when

sdf.select("yearConstructed", when(sdf.serviceCharge >= 100, "high service range")).show()
print(f'run time : {time.time() - s_time}')
```

```
+------------------+-------------------------------------------------------+
|   yearConstructed|CASE WHEN (serviceCharge >= 100) THEN high service range END|
+------------------+-------------------------------------------------------+
|              57.0|                                     high service range|
|              57.0|                                     high service range|
|54.832749319931736|                                                   null|
|              63.0|                                                   null|
|              63.0|                                                   null|
|             130.0|                                     high service range|
|              46.0|                                     high service range|
|54.832749319931736|                                     high service range|
|             117.0|                                     high service range|
|54.832749319931736|                                     high service range|
|              64.0|                                                   null|
|              24.0|                                     high service range|
|54.832749319931736|                                     high service range|
|             108.0|                                     high service range|
|              43.0|                                     high service range|
|             102.0|                                     high service range|
|              26.0|                                                   null|
|              38.0|                                                   null|
|              58.0|                                     high service range|
|54.832749319931736|                                     high service range|
+------------------+-------------------------------------------------------+
only showing top 20 rows

run time : 0.14397835731506348
```

Dropping duplicate data:

```
[477] s_time = time.time()
      sdf=sdf.drop_duplicates()

      print(f'run time : {time.time() - s_time}')

      run time : 0.006994009017944336
```

```
sdf.dtypes
```

```
[('serviceCharge', 'double'),
 ('totalRent', 'double'),
 ('yearConstructed', 'double'),
 ('yearConstructedRange', 'double'),
 ('baseRent', 'double'),
 ('livingSpace', 'double'),
 ('baseRentRange', 'double'),
 ('noRooms', 'double'),
 ('floor', 'double'),
 ('balcony', 'double'),
 ('cellar', 'double'),
 ('condition', 'double'),
 ('condition_edit', 'double'),
 ('garden', 'double'),
 ('hasKitchen', 'double'),
 ('heatingType_edit', 'double'),
 ('lift', 'double'),
 ('newlyConst', 'double'),
 ('regio2_edit', 'double'),
 ('typeOfFlat', 'double'),
 ('Pricepm2', 'double'),
 ('additioncost', 'double')]
```