Two Sigma Connect: Rental Listing Inquiries

https://github.com/tatvch/Two-Sigma-Connect-Rental-Listing-Inquiries/blob/main/two-sigma-connect-rental-listing-inquiries.ipynb

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))


# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/two-sigma-connect-rental-listing-inquiries/images_sample.zip
/kaggle/input/two-sigma-connect-rental-listing-inquiries/Kaggle-renthop.torrent
/kaggle/input/two-sigma-connect-rental-listing-inquiries/sample_submission.csv.zip
/kaggle/input/two-sigma-connect-rental-listing-inquiries/test.json.zip
/kaggle/input/two-sigma-connect-rental-listing-inquiries/train.json.zip
```

```
import pandas as pd
import numpy as np
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import itertools
import matplotlib.pyplot as plt
import string
import re
import collections
from sklearn import  preprocessing
from wordcloud import WordCloud
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from xgboost import XGBClassifier
import xgboost as xgb
from sklearn.metrics import make_scorer, f1_score, accuracy_score, mean_absolute_error, confusion_matrix
import optuna

%matplotlib inline
import itertools




import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()
%matplotlib inline
sns.set_style("whitegrid")
import warnings
```

```
train = pd.read_json('../input/two-sigma-connect-rental-listing-inquiries/train.json.zip')
test = pd.read_json('../input/two-sigma-connect-rental-listing-inquiries/test.json.zip')
```

Basic Insight of Dataset (Базовое представление о наборе данных)

```
train. head(3)
```

	bathrooms	bedrooms	building_id	created	description	display_address	features	latitude	listing_id	longitude	manager_id	
4	1.0	1	8579a0b0d54db803821a35a4a615e97a	2016-06-16 05:55:27	Spacious 1 Bedroom 1 Bathroom in Williamsburg!...	145 Borinquen Place	[Dining Room, Pre-War, Laundry in Building, Di...	40.7108	7170325	-73.9539	a10db4590843d78c784171a107bdacb4	[https://photos.renthop.
6	1.0	2	b8e75fc949a6cd8225b455648a951712	2016-06-01 05:44:33	BRAND NEW GUT RENOVATED TRUE 2 BEDROOMFind you...	East 44th	[Doorman, Elevator, Laundry in Building, Dishw...	40.7513	7092344	-73.9722	955db33477af4f40004820b4aed804a0	[https://photos.renthop.
9	1.0	2	cd759a988b8f23924b5a2058d5ab2b49	2016-06-14 15:19:59	**FLEX 2 BEDROOM WITH FULL PRESSURIZED WALL**L...	East 56th Street	[Doorman, Elevator, Laundry in Building, Laund...	40.7575	7158677	-73.9625	c8b10a317b766204f08e613cef4ce7a0	[https://photos.renthop.

```
train.shape
```

```
(49352, 15)
```

```
print(train.columns.values)
```

```
['bathrooms' 'bedrooms' 'building_id' 'created' 'description'
 'display_address' 'features' 'latitude' 'listing_id' 'longitude'
 'manager_id' 'photos' 'price' 'street_address' 'interest_level']
```

```
train.describe()
```

	bathrooms	bedrooms	latitude	listing_id	longitude	price
count	49352.00000	49352.000000	49352.000000	4.935200e+04	49352.000000	4.935200e+04
mean	1.21218	1.541640	40.741545	7.024055e+06	-73.955716	3.830174e+03
std	0.50142	1.115018	0.638535	1.262746e+05	1.177912	2.206687e+04
min	0.00000	0.000000	0.000000	6.811957e+06	-118.271000	4.300000e+01
25%	1.00000	1.000000	40.728300	6.915888e+06	-73.991700	2.500000e+03
50%	1.00000	1.000000	40.751800	7.021070e+06	-73.977900	3.150000e+03
75%	1.00000	2.000000	40.774300	7.128733e+06	-73.954800	4.100000e+03
max	10.00000	8.000000	44.883500	7.753784e+06	0.000000	4.490000e+06

```
train.describe(include = "all")
```

	bathrooms	bedrooms	building_id	created	description	display_address	features	latitude	listing_id	longitude	manager_id	photos	price	street_
count	49352.00000	49352.000000	49352	49352	49352	49352	49352	49352.000000	4.935200e+04	49352.000000	49352	49352	4.935200e+04	
unique	NaN	NaN	7585	48675	38244	8826	10254	NaN	NaN	NaN	3481	45677	NaN	
top	NaN	NaN	0	2016-04-08 01:14:27		Broadway	[]	NaN	NaN	NaN	e6472c7237327dd3903b3d6f6a94515a	[]	NaN	B
freq	NaN	NaN	8286	3	1647	438	3218	NaN	NaN	NaN	2533	3615	NaN	
mean	1.21218	1.541640	NaN	NaN	NaN	NaN	NaN	40.741545	7.024055e+06	-73.955716	NaN	NaN	3.830174e+03	
std	0.50142	1.115018	NaN	NaN	NaN	NaN	NaN	0.638535	1.262746e+05	1.177912	NaN	NaN	2.206687e+04	
min	0.00000	0.000000	NaN	NaN	NaN	NaN	NaN	0.000000	6.811957e+06	-118.271000	NaN	NaN	4.300000e+01	
25%	1.00000	1.000000	NaN	NaN	NaN	NaN	NaN	40.728300	6.915888e+06	-73.991700	NaN	NaN	2.500000e+03	
50%	1.00000	1.000000	NaN	NaN	NaN	NaN	NaN	40.751800	7.021070e+06	-73.977900	NaN	NaN	3.150000e+03	
75%	1.00000	2.000000	NaN	NaN	NaN	NaN	NaN	40.774300	7.128733e+06	-73.954800	NaN	NaN	4.100000e+03	
max	10.00000	8.000000	NaN	NaN	NaN	NaN	NaN	44.883500	7.753784e+06	0.000000	NaN	NaN	4.490000e+06	

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 49352 entries, 4 to 124009
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   bathrooms        49352 non-null  float64
 1   bedrooms         49352 non-null  int64
 2   building_id      49352 non-null  object
 3   created          49352 non-null  object
 4   description      49352 non-null  object
 5   display_address  49352 non-null  object
 6   features         49352 non-null  object
 7   latitude         49352 non-null  float64
 8   listing_id       49352 non-null  int64
 9   longitude        49352 non-null  float64
 10  manager_id       49352 non-null  object
 11  photos           49352 non-null  object
 12  price            49352 non-null  int64
 13  street_address   49352 non-null  object
 14  interest_level   49352 non-null  object
dtypes: float64(3), int64(3), object(9)
memory usage: 6.0+ MB
```

```
train.isnull().sum()
```

```
bathrooms          0
bedrooms           0
building_id        0
created            0
description        0
display_address    0
features           0
latitude           0
listing_id         0
longitude          0
manager_id         0
photos             0
price              0
street_address     0
interest_level     0
dtype: int64
```

Data visualization and pre-processing (Визуализация данных и предварительная обработка)

'interest_level'¶

```
train.groupby(train['interest_level']).mean()
```

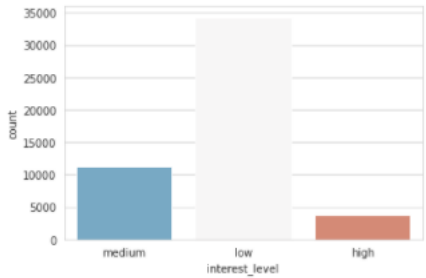
	bathrooms	bedrooms	latitude	listing_id	longitude	price
interest_level						
high	1.116176	1.546496	40.748007	7.017844e+06	-73.964613	2700.293045
low	1.238741	1.514759	40.739504	7.026373e+06	-73.951667	4176.599142
medium	1.163906	1.622050	40.745567	7.019098e+06	-73.965033	3158.767388

```
train['interest_level'].value_counts()
```

```
low       34284
medium    11229
high       3839
Name: interest_level, dtype: int64
```

```
sns.set_style('whitegrid')
sns.countplot(x='interest_level',data=train,palette='RdBu_r')
```

<AxesSubplot:xlabel='interest_level', ylabel='count'>



'bathrooms'

'bathrooms'¶

```
# Value Counts
train['bathrooms'].value_counts()
```

```
1.0     39422
2.0      7660
```

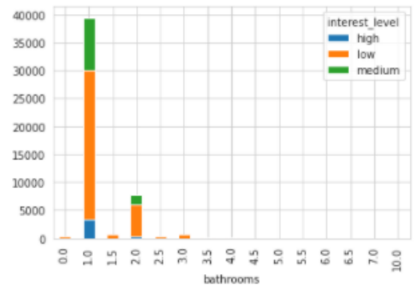
```
3.0       745
1.5       645
0.0       313
2.5       277
4.0       159
3.5        70
4.5        29
5.0        20
5.5         5
6.0         4
6.5         1
7.0         1
10.0        1
Name: bathrooms, dtype: int64
```

```
train.groupby(['bathrooms']) ['interest_level'].value_counts(normalize=True)
```

```
bathrooms  interest_level
0.0        low               0.977636
           medium            0.019169
           high              0.003195
1.0        low               0.674268
           medium            0.239156
           high              0.086576
1.5        low               0.937984
           medium            0.062016
2.0        low               0.726632
           medium            0.220235
           high              0.053133
2.5        low               0.989170
           medium            0.010830
3.0        low               0.900671
           medium            0.080537
           high              0.018792
3.5        low               1.000000
4.0        low               0.943396
           medium            0.031447
           high              0.025157
4.5        low               1.000000
5.0        low               1.000000
5.5        low               1.000000
6.0        low               1.000000
6.5        low               1.000000
7.0        low               1.000000
10.0       low               1.000000
Name: interest_level, dtype: float64
```

```
train.pivot_table('price', 'bathrooms', 'interest_level', 'count').plot(kind='bar', stacked=True)
```

<AxesSubplot:xlabel='bathrooms'>



'bedrooms'

```
# Value Counts
train['bedrooms'].value_counts()
```

```
1    15752
2    14623
```

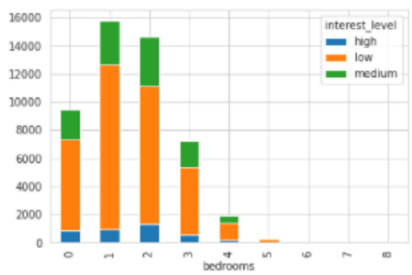
```
0     9475
3     7276
4     1929
5      247
6       46
8        2
7        2
Name: bedrooms, dtype: int64
```

```
train.groupby(['bedrooms']) ['interest_level'].value_counts(normalize=True)
```

```
bedrooms  interest_level
0         low               0.687916
          medium            0.222691
          high              0.089393
1         low               0.743715
          medium            0.196420
          high              0.059865
2         low               0.670246
          medium            0.240443
          high              0.089311
3         low               0.649670
          medium            0.268966
          high              0.081363
4         low               0.639191
          medium            0.283567
          high              0.077242
5         low               0.983806
          high              0.008097
          medium            0.008097
6         low               0.956522
          medium            0.043478
7         low               0.500000
          medium            0.500000
8         low               1.000000
Name: interest_level, dtype: float64
```

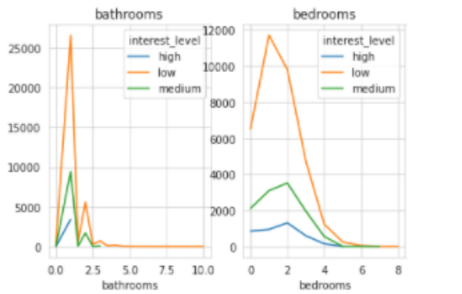
```
train.pivot_table('price', 'bedrooms', 'interest_level', 'count').plot(kind='bar', stacked=True)
```

```
<AxesSubplot:xlabel='bedrooms'>
```




```
# 'bathrooms' and 'bedrooms'
fig, axes = plt.subplots(ncols=2)
train.pivot_table('price', ['bathrooms'], 'interest_level', 'count').plot(ax=axes[0], title='bathrooms')
train.pivot_table('price', ['bedrooms'], 'interest_level', 'count').plot(ax=axes[1], title='bedrooms')
```

```
<AxesSubplot:title={'center':'bedrooms'}, xlabel='bedrooms'>
```



'building_id

```
# Most advertised buildings
train.building_id.value_counts().nlargest(10)
```

```
0                                   8286
96274288c84ddd7d5c5d8e425ee75027     275
11e1dec9d14b1a9e528386a2504b3afc     215
80a120d6bc3aba97f40fee8c2204524b     213
bb8658a3e432fb62a440615333376345     212
f68bf347f99df026f4faad43cc604048     191
c94301249b8c09429d329864d58e5b82     167
ce6d18bf3238e668b2bf23f4110b7b67     165
57ef86c28a8ae482dc3a3c3af28e8e48     159
128d4af0683efc5e1eded8dc8044d5e3     153
Name: building_id, dtype: int64
```

'created'

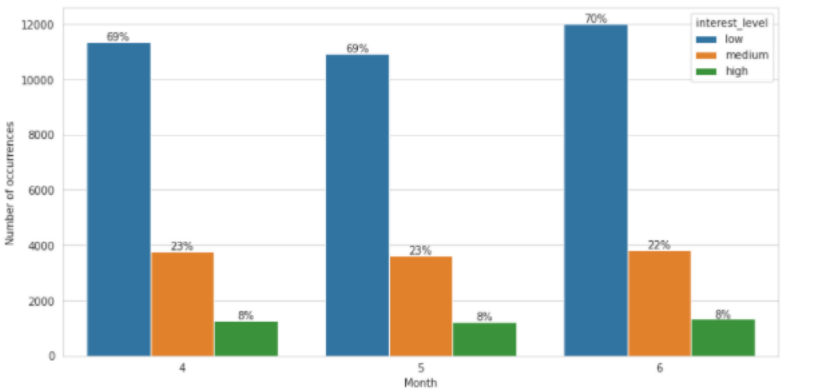
```
# Convertion to Python Date  (Преобразование в дату Python)
train.created = pd.to_datetime(train.created, format='%Y-%m-%d %H:%M:%S')
test.created = pd.to_datetime(test.created, format='%Y-%m-%d %H:%M:%S')
```

```
#  Month, Day of Week and Hour Features  (Месяц, день недели и часы)
train['month'] = train.created.dt.month
train['day_of_week'] = train.created.dt.weekday
train['hour'] = train.created.dt.hour

train.head(2)
```

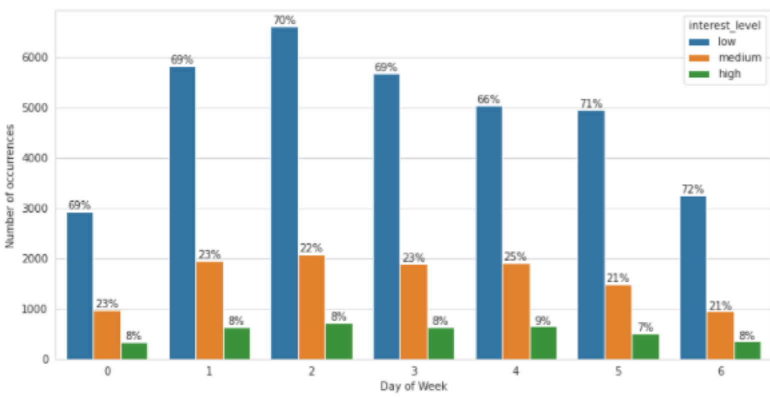
isplay_address	features	latitude	listing_id	longitude	manager_id	photos	price	street_address	interest_level	month	day_of_week	hour
145 Borinquen Place	[Dining Room, Pre-War, Laundry in Building, Di...	40.7108	7170325	-73.9539	a10db4590843d78c784171a107bdacb4	[https://photos.renthop.com/2/7170325_3bb5ac84...	2400	145 Borinquen Place	medium	6	3	5
East 44th	[Doorman, Elevator, Laundry in Building, Dishw...	40.7513	7092344	-73.9722	955db33477af4f40004820b4aed804a0	[https://photos.renthop.com/2/7092344_7663c19a...	3800	230 East 44th	low	6	2	5

```
# Iterest per month
fig = plt.figure(figsize=(12,6))
ax = sns.countplot(x="month", hue="interest_level", hue_order=['low', 'medium', 'high'],
                   data=train);
plt.xlabel('Month');
```

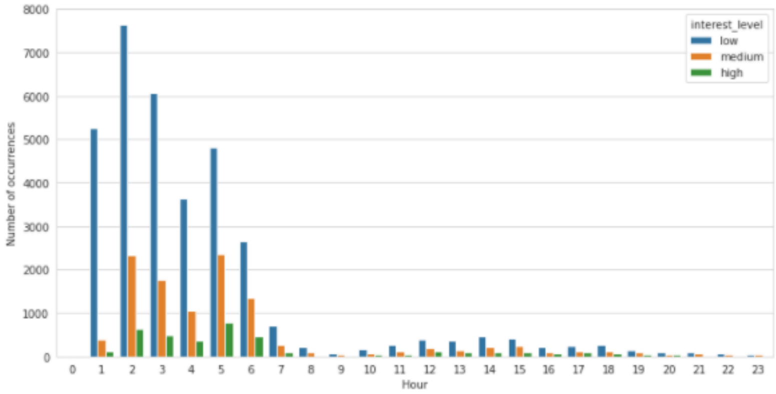
```
plt.ylabel('Number of occurrences')

# Adding percents over bars
height = [p.get_height() for p in ax.patches]
ncol = int(len(height)/3)
total = [height[i] + height[i + ncol] + height[i + 2*ncol] for i in range(ncol)] * 3
for i, p in enumerate(ax.patches):
    ax.text(p.get_x()+p.get_width()/2,
            height[i] + 50,
            '{:1.0%}'.format(height[i]/total[i]),
            ha="center")
```




```
# Iterest per Day of Week
fig = plt.figure(figsize=(12,6))
ax = sns.countplot(x="day_of_week", hue="interest_level",
                   hue_order=['low', 'medium', 'high'], data=train);
plt.xlabel('Day of Week');
plt.ylabel('Number of occurrences');

### Adding percents over bars
height = [p.get_height() for p in ax.patches]
ncol = int(len(height)/3)
total = [height[i] + height[i + ncol] + height[i + 2*ncol] for i in range(ncol)] * 3
for i, p in enumerate(ax.patches):
    ax.text(p.get_x()+p.get_width()/2,
            height[i] + 50,
            '{:1.0%}'.format(height[i]/total[i]),
            ha="center")
```




```
# Iterest per Hour
fig = plt.figure(figsize=(12,6))
sns.countplot(x="hour", hue="interest_level", hue_order=['low', 'medium', 'high'], data=train);
plt.xlabel('Hour');
plt.ylabel('Number of occurrences');
```



'display_address'

```
# Number of unique Display Addresses
print('Number of Unique Display Addresses is {}'.format(train.display_address.value_counts().shape[0]))
```

```
Number of Unique Display Addresses is 8826
```

```
# 20 most popular Display Addresses
train.display_address.value_counts().nlargest(20)
```

```
Broadway              438
East 34th Street      355
Second Avenue         349
Wall Street           332
West 37th Street      287
West Street           258
First Avenue          244
Gold Street           241
Washington Street     237
York Avenue           228
John Street           214
Water Street          214
East 39th Street      200
East 89th Street      195
West 54th Street      193
Lexington Avenue      189
Fifth Avenue          189
West 42nd Street      184
Christopher Street    180
Third Avenue          178
Name: display_address, dtype: int64
```

```
# Top 20 northernmost points (Топ-20 самых северных точек)
train.latitude.nlargest(20)
```

```
78568     44.8835
16405     44.6038
18267     43.0346
81815     42.8725
4719      42.8724
872       42.3459
24747     42.3459
80360     42.3459
85995     42.3459
62409     42.3033
```

```
73065     42.3033
117255    42.2509
39046     42.2019
41022     42.2019
57131     42.2019
114889    42.2019
72896     41.7530
18023     41.0868
71920     41.0868
100346    41.0412
Name: latitude, dtype: float64
```

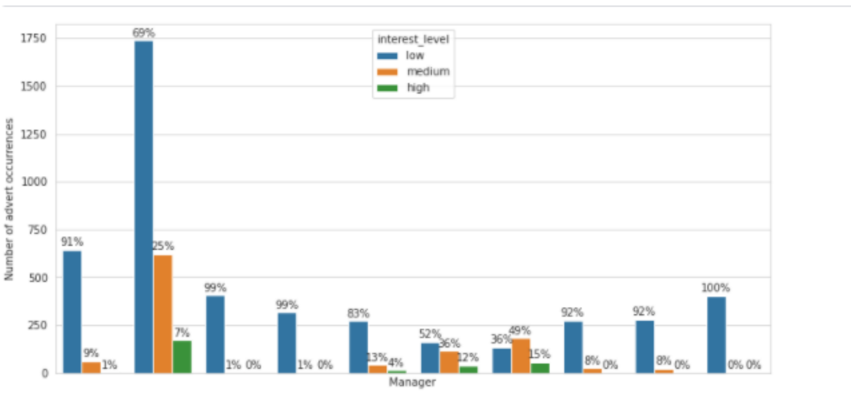
```
# Rent interest graph of New-York
sns.lmplot(x="longitude", y="latitude", fit_reg=False, hue='interest_level',
           hue_order=['low', 'medium', 'high'], size=9, scatter_kws={'alpha':0.4,'s':30},
           data=train[(train.longitude>train.longitude.quantile(0.1))
                        &(train.longitude<train.longitude.quantile(0.9))
                        &(train.latitude>train.latitude.quantile(0.1))
                        &(train.latitude<train.latitude.quantile(0.9))]);
plt.xlabel('Longitude');
plt.ylabel('Latitude');
```

'manager_id'

```
train.manager_id.value_counts().nlargest(10)
```

```
e6472c7237327dd3903b3d6f6a94515a    2533
6e5c10246156ae5bdcd9b487ca99d96a     711
8f5a9c893f6d602f4953fcc0b8e6e9b4     410
62b685cc0d876c3a1a51d63a0d6a8082     402
cb87dadbca78fad02b388dc9e8f25a5b     373
9df32cb8dda19d3222d66e69e258616b     330
b7de4cb395920136663132057fa89d84     320
2aa9bfa5f67ed9997ea341dee8a3a271     316
ad3d8ddc52c7e0859b5c6c7f7949c3bd     305
c9c33695ee2a2f818e9f1d8f7d1c4b39     299
Name: manager_id, dtype: int64
```

```
# Let's get a list of top 10 managers
top10managers = train.manager_id.value_counts().nlargest(10).index.tolist()
# ...and plot number of different Interest Level rental adverts for each of them
fig = plt.figure(figsize=(12,6))
ax = sns.countplot(x="manager_id", hue="interest_level",
                   data=train[train.manager_id.isin(top10managers)]);
plt.xlabel('Manager');
plt.ylabel('Number of advert occurrences');
### Manager_ids are too long. Let's remove them
plt.tick_params(
    axis='x',          # changes apply to the x-axis
    which='both',      # both major and minor ticks are affected
    bottom='off',      # ticks along the bottom edge are off
    top='off',         # ticks along the top edge are off
    labelbottom='off');


plt.xticks([])
# Adding percents over bars
height = [0 if np.isnan(p.get_height()) else p.get_height() for p in ax.patches]
ncol = int(len(height)/3)
total = [height[i] + height[i + ncol] + height[i + 2*ncol] for i in range(ncol)] * 3
for i, p in enumerate(ax.patches):
    ax.text(p.get_x()+p.get_width()/2,
            height[i] + 20,
            '{:1.0%}'.format(height[i]/total[i]),
            ha="center")
```



'photos_number'

```
# count of photos
train["photos_number"] = train["photos"].apply(len)
train["photos_number"].value_counts()
```

```
5     8733
4     7887
6     6739
7     4952
3     4553
8     3972
0     3615
9     1772
10    1390
2     1334
1     1178
12     816
11     784
13     377
14     232
15     197
16     152
18     114
17     109
20      87
22      73
32      62
```

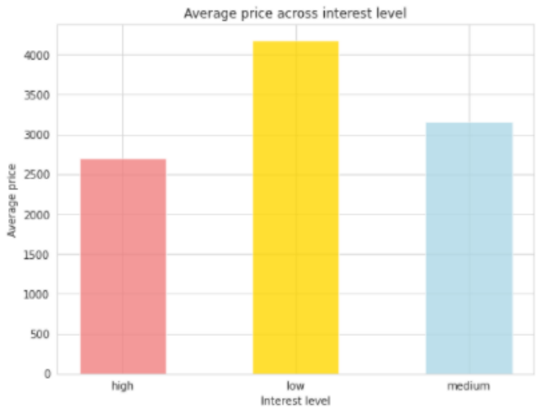
```
19      53
21      21
26      19
37      18
25      17
23      16
24      16
45      12
28      11
27       7
38       6
34       5
30       5
29       4
36       3
35       3
46       2
60       1
44       1
68       1
43       1
50       1
31       1
Name: photos_number, dtype: int64
```

price

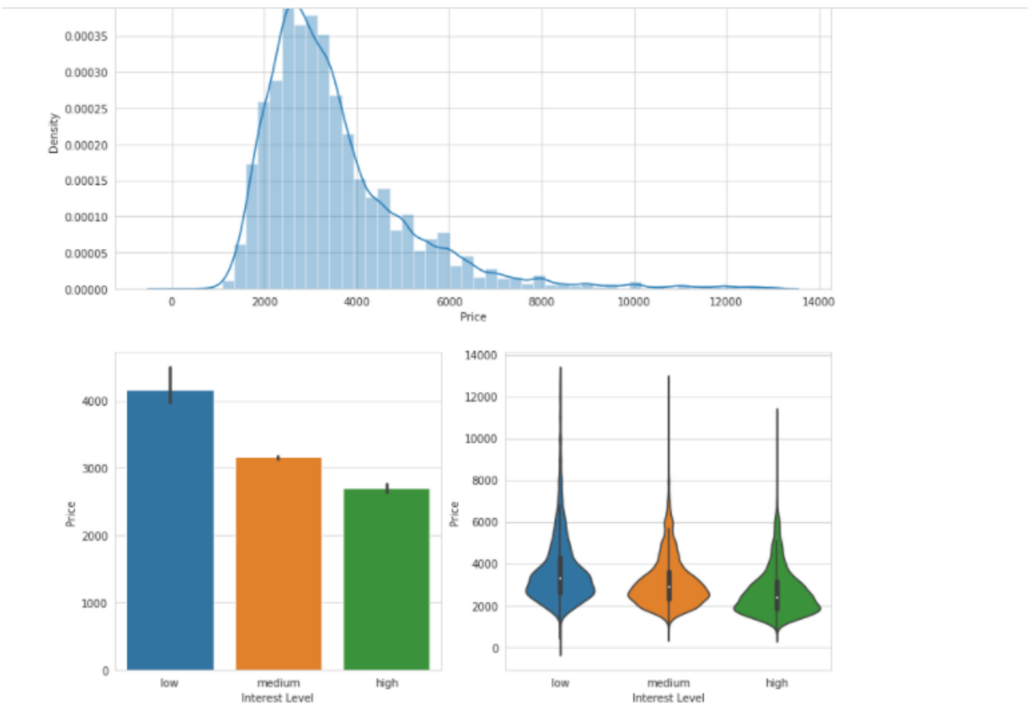
```
# Value Counts
train['price'].value_counts()
```

```
2500      1106
3200       881
3000       840
2700       777
2400       772
          ...
5753         1
2417         1
9625         1
5433         1
135000       1
Name: price, Length: 2808, dtype: int64
```

```
# Price exploration
prices=train.groupby('interest_level', as_index=False)['price'].mean()
colors = ['lightcoral','gold','lightblue']

fig=plt.figure(figsize=(8,6))
plt.bar(prices.interest_level, prices.price, color=colors, width=0.5, alpha=0.8)
#set titles
plt.xlabel('Interest level')
plt.ylabel('Average price')
plt.title('Average price across interest level')
plt.show()
```




```
# Price exploration
fig = plt.figure(figsize=(12,12))
# Price distribution
sns.distplot(train.price[train.price<=train.price.quantile(0.99)], ax=plt.subplot(211));
plt.xlabel('Price');
plt.ylabel('Density');

### Average Price per Interest Level
sns.barplot(x="interest_level", y="price", order=['low', 'medium', 'high'],
            data=train, ax=plt.subplot(223));
plt.xlabel('Interest Level');
plt.ylabel('Price');

### Violinplot of price for every Interest Level
sns.violinplot(x="interest_level", y="price", order=['low', 'medium', 'high'],
               data=train[train.price<=train.price.quantile(0.99)],
               ax=plt.subplot(224));
plt.xlabel('Interest Level');
plt.ylabel('Price');
```



'description'

```
train['description'].iloc[0]
```

```
'Spacious 1 Bedroom 1 Bathroom in Williamsburg!Apartment Features:- Renovated Eat in Kitchen With Dishwasher- Renovated Bathroom- Beautiful Hardwood Floors- Lots of Sunlight- Great Closet Space- Freshly Painted- Heat and Hot Water Included- Live in Super Nearby L, J, M & G Trains !<br /><br />Contact Information:Kenneth BeakExclusive AgentC: 064-692-8838Email: kagglemanager@renthop.com, Text or Email to schedule a private viewing!<br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><p><a website_redacted '
```

```
# REMOVE UNNECESSARY WORDS FROM DESCRIPTION  (УДАЛИТЬ НЕНУЖНЫЕ СЛОВА ИЗ ОПИСАНИЯ)

train['description'] = train['description'].apply(lambda x: x.replace("<br />", ""))
train['description'] = train['description'].apply(lambda x: x.replace("br", ""))
train['description'] = train['description'].apply(lambda x: x.replace("<p><a", ""))
train['description'].iloc[0]
```

```
'Spacious 1 Bedroom 1 Bathroom in Williamsburg!Apartment Features:- Renovated Eat in Kitchen With Dishwasher- Renovated Bathroom- Beautiful Hardwood Floors- Lots of Sunlight- Great Closet Space- Freshly Painted- Heat and Hot Water Included- Live in Super Nearby L, J, M & G Trains !Contact Information:Kenneth BeakExclusive AgentC: 064-692-8838Email: kagglemanager@renthop.com, Text or Email to schedule a private viewing! website_redacted '
```

```
# description contains email
regex = r'[\w\.-]+@[\w\.-]+'
train['has_email'] = train['description'].apply(lambda x: 1 if re.findall(regex, x) else 0)
train['has_email'].value_counts()
```

```
0    32077
1    17275
Name: has_email, dtype: int64
```

```
# description contains phone
train['has_phone'] = train['description'].apply(lambda x:re.sub('['+string.punctuation+']', '', x).split())\
        .apply(lambda x: [s for s in x if s.isdigit()])\
        .apply(lambda x: len([s for s in x if len(str(s))==10]))\
        .apply(lambda x: 1 if x>0 else 0)
train['has_phone'].value_counts()
```

```
0    31492
1    17860
Name: has_phone, dtype: int64
```

'features'

```
# count of "features"
train["num_features"] = train["features"].apply(len)
train["num_features"].value_counts()
```

```
3     6211
4     5459
2     4938
5     4547
1     4340
6     3835
7     3374
0     3218
8     2840
9     2453
10    2217
11    1681
```

```
12    1377
13    1009
14     737
15     456
16     283
17     161
18      89
19      45
20      24
21      14
22      13
26       8
23       6
24       5
27       3
28       3
25       2
32       1
36       1
39       1
31       1
Name: num_features, dtype: int64
```

```
# CONVERT LOWER ALL OF WORDS
train[["features"]] = train[["features"]].apply(
    lambda _: [list(map(str.strip, map(str.lower, x))) for x in _])
train[["features"]]
```

	features
4	[dining room, pre-war, laundry in building, di...
6	[doorman, elevator, laundry in building, dishw...
9	[doorman, elevator, laundry in building, laund...
10	[]
15	[doorman, elevator, fitness center, laundry in...
...	...
124000	[elevator, dishwasher, hardwood floors]
124002	[common outdoor space, cats allowed, dogs allo...
124004	[dining room, elevator, pre-war, laundry in bu...
124008	[pre-war, laundry in unit, dishwasher, no fee,...
124009	[dining room, elevator, laundry in building, d...

49352 rows × 1 columns

MOST FREQUENT FEATURES EXTRACTION (ИЗВЛЕЧЕНИЕ НАИБОЛЕЕ ЧАСТОТЫХ ФУНКЦИЙ)

```
feature_value_train = train['features'].tolist()
feature_value_test = test['features'].tolist()
```

```
feature_lst_train = []
for i in range(len(feature_value_train)):
    feature_lst_train += feature_value_train[i]

uniq_feature_train = list(set(feature_lst_train))

# print(uniq_feature) #all unique features
len(uniq_feature_train)
```

```
1293
```

```
# see the frequency of each feature
import collections
def most_common(lst):
```

```
    features = collections.Counter(lst)
    feature_value = features.keys()
    frequency = features.values()
    data = [('feature_value', feature_value),
            ('frequency', frequency),]
    df = pd.DataFrame.from_dict(dict(data))
    return df.sort_values(by = 'frequency', ascending = False)

df_features_train = most_common(feature_lst_train)

df_features_test = most_common(feature_lst_test)

df_features_train
```

	feature_value	frequency
8	elevator	26273
4	hardwood floors	23558
6	cats allowed	23540
5	dogs allowed	22035
7	doorman	20967
...	...	...
734	** central park steal! * massive studio suprem...	1
251	12th st & 3rd ave	1
736	full kitchen	1
737	new stainless appliances	1
1292	available 06/04/16 firepalce	1

1293 rows × 2 columns

```
df_features_train.head(20)
```

	feature_value	frequency
8	elevator	26273
4	hardwood floors	23558
6	cats allowed	23540
5	dogs allowed	22035
7	doorman	20967
3	dishwasher	20806
2	laundry in building	18944
9	no fee	18079
11	fitness center	13257
10	laundry in unit	9435
1	pre-war	9149
14	roof deck	6555
34	outdoor space	5270
0	dining room	5150
15	high speed internet	4299
23	balcony	3058
16	swimming pool	2730
30	new construction	2608
33	terrace	2313
32	exclusive	2167

```
def newColumn(name, df, series):
    feature = pd.Series(0,df.index,name = name)# data : 0
    for row,word in enumerate(series):
        if name in word:
            feature.iloc[row] = 1
    df[name] = feature # feature : series ; value in series : 1 or 0
    return df

# select features based on frequency
```

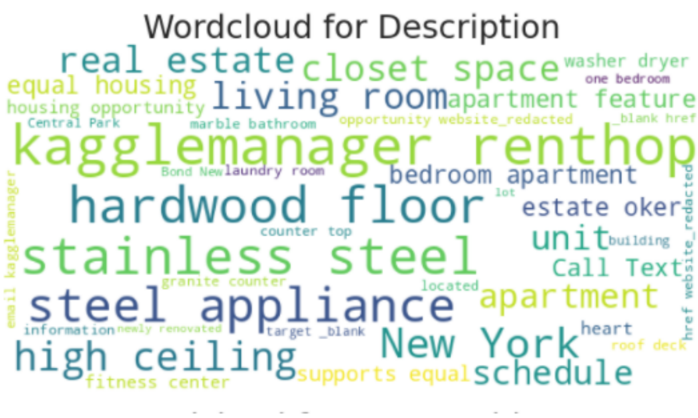
```
facilities = [
'elevator', 'cats allowed', 'hardwood floors', 'dogs allowed', 'doorman', 'dishwasher', 'no fee', 'laundry in building', 'fitness cent
'laundry in unit', 'pre-war', 'roof deck', 'outdoor space','dining room', 'high speed internet', 'balcony', 'swimming pool',
'new construction', 'terrace']
for name in facilities:
    train = newColumn(name, train, train['features'])
    test = newColumn(name, test, test['features'])
train.head(5)
```

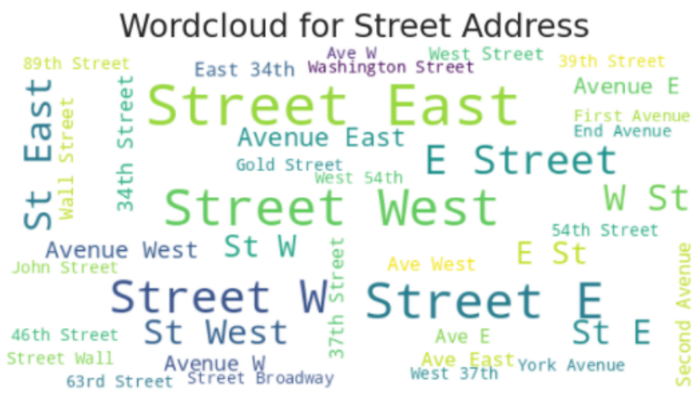
	bathrooms	bedrooms	building_id	created	description	display_address	features	latitude	listing_id	longitude	...	laundry in unit	pre-war	roof deck	outdoor space	dining room	high speed internet	balc
4	1.0	1	8579a0b0d54db803821a35a4a615e97a	2016-06-16 05:55:27	Spacious 1 Bedroom 1 Bathroom in Williamsburg!...	145 Borinquen Place	[dining room, pre-war, laundry in building, di...	40.7108	7170325	-73.9539	...	0	1	0	0	1	0	
6	1.0	2	b8e75fc949a6cd8225b455648a951712	2016-06-01 05:44:33	BRAND NEW GUT RENOVATED TRUE 2 BEDROOMFind you...	East 44th	[doorman, elevator, laundry in building, dishw...	40.7513	7092344	-73.9722	...	0	0	0	0	0	0	
9	1.0	2	cd759a988b8f23924b5a2058d5ab2b49	2016-06-14 15:19:59	**FLEX 2 BEDROOM WITH FULL PRESSURIZED WALL**L...	East 56th Street	[doorman, elevator, laundry in building, laund...	40.7575	7158677	-73.9625	...	1	0	0	0	0	0	
10	1.5	3	53a5b119ba8f7b61d4e010512e0dfc85	2016-06-24 07:54:24	A Brand New 3 Bedroom 1.5 bath ApartmentEnjoy ...	Metropolitan Avenue	[]	40.7145	7211212	-73.9425	...	0	0	0	0	0	0	
15	1.0	0	bfb9405149bfff42a92980b594c28234	2016-06-28 03:50:23	Over-sized Studio w abundant closets. Availabl...	East 34th Street	[doorman, elevator, fitness center, laundry in...	40.7439	7225292	-73.9743	...	0	0	0	0	0	0	

5 rows × 41 columns

```
#WORDCLOUD FOR DESCRIPTION AND DISPLAY ADDRESS
#Preprocessing
text = ''
text_da = ''
text_desc = ''
text_str = ''
for ind, row in train.iterrows():
    for feature in row['features']:
        text = " ".join([text, "_".join(feature.strip().split(" "))])
    text_da = " ".join([text_da,"_".join(row['display_address'].strip().split(" "))])
    text_desc = " ".join([text_desc, row['description']])
    text_str = " ".join([text_str, row['street_address']])
text = text.strip()
text_da = text_da.strip()
text_desc = text_desc.strip()
text_str = text_str.strip()


# wordcloud for features
plt.figure(figsize=(12,6))
wordcloud = WordCloud(background_color='white', width=600, height=300, max_font_size=50, max_words=40).generate(text)
wordcloud.recolor(random_state=0)
plt.imshow(wordcloud)
plt.title("Wordcloud for features", fontsize=30)
plt.axis("off")
plt.show()


# wordcloud for display address
plt.figure(figsize=(12,6))
wordcloud = WordCloud(background_color='white', width=600, height=300, max_font_size=50, max_words=40).generate(text_da)
wordcloud.recolor(random_state=0)
plt.imshow(wordcloud)
plt.title("Wordcloud for Display Address", fontsize=30)
plt.axis("off")
plt.show()


# wordcloud for description
plt.figure(figsize=(12,6))
wordcloud = WordCloud(background_color='white', width=600, height=300, max_font_size=50, max_words=40).generate(text_desc)
wordcloud.recolor(random_state=0)
```

```
plt.imshow(wordcloud)
plt.title("Wordcloud for Description", fontsize=30)
plt.axis("off")
plt.show()

# wordcloud for street address
plt.figure(figsize=(12,6))
wordcloud = WordCloud(background_color='white', width=600, height=300, max_font_size=50, max_words=40).generate(text_str)
wordcloud.recolor(random_state=0)
plt.imshow(wordcloud)
plt.title("Wordcloud for Street Address", fontsize=30)
plt.axis("off")
plt.show()
```





Our target 'INTEREST LEVEL' is an object.(Наша цель «УРОВЕНЬ ИНТЕРЕСА» — это объект.)

Let's convert to the numeric to analyze easily (Давайте перейдем к числовому, чтобы легко анализировать)

0 : low; 1 : medium; 2 : high

```
train['target'] = train['interest_level'].apply(lambda x: 0 if x=='low'
                                                      else 1 if x=='medium'
                                                      else 2)
# train_df['low'] = train_df['interest_level'].apply(lambda x: 1 if x=='low' else 0)
# train_df['medium'] = train_df['interest_level'].apply(lambda x: 1 if x=='medium' else 0)
# train_df['high'] = train_df['interest_level'].apply(lambda x: 1 if x=='high' else 0)
```

DROP UNNECESSARY COLUMNS (УДАЛИТЬ НЕНУЖНЫЕ СТОЛБЦЫ)

```
# TRAINING DATASET
train.drop('interest_level', axis=1, inplace=True)
train.drop('created', axis=1, inplace=True)
train.drop('description', axis=1, inplace=True)
train.drop('features', axis=1, inplace=True)
train.drop('photos', axis=1, inplace=True)


# TEST DATASET
test.drop('created', axis=1, inplace=True)
test.drop('description', axis=1, inplace=True)
test.drop('features', axis=1, inplace=True)
test.drop('photos', axis=1, inplace=True)
```

LABEL ECONDING FOR CATEGORICAL VARIABLES (КОДИРОВАНИЕ МЕТКИ ДЛЯ CATEGORICAL ПЕРЕМЕННЫХ)

```
categorical = ["display_address", "manager_id", "building_id", "street_address"]
for f in categorical:
        if train[f].dtype=='object':
            lbl = preprocessing.LabelEncoder()
            lbl.fit(list(train[f].values) + list(test[f].values))
```

```
            train[f] = lbl.transform(list(train[f].values))
            test[f] = lbl.transform(list(test[f].values))
```

XGBOOST (алгоритм градиентного бустинга на деревьях)

```
X = train.drop(['target'], axis = 1)
y = train.target


X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.2,
                                                    random_state = 42,
                                                   stratify = y)

# Импорт XGBoost и создание необходимых объектов.
import xgboost as xgb
dtrain = xgb.DMatrix(X_train, label=y_train)
dvalid = xgb.DMatrix(X_test, label=y_test)
```

```
kf = KFold(n_splits=5, shuffle=False)

X_train = X_train.values
y_train = y_train.values
scores = []

for train, test in kf.split(X_train, y_train):
    model = XGBClassifier(n_estimators=1000, learning_rate=0.05, max_depth = 10)
    model.fit(X_train[train], y_train[train])
    scores.append(model.score(X_train[test], y_train[test]))
```

```
/opt/conda/lib/python3.7/site-packages/xgboost/sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be removed in a future releas
e. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starti
ng with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[09:11:19] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to
'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[09:14:47] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to
'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[09:18:12] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to
'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[09:21:39] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to
'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[09:25:06] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to
'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

```
def objective(trial):
    params = {
        'booster':trial.suggest_categorical('booster', ['gbtree', 'dart', 'gblinear']),
        'learning_rate':trial.suggest_loguniform("learning_rate", 0.01, 0.1),
        'max_depth':trial.suggest_int("max_depth", 3, 11),
        'subsample':trial.suggest_uniform("subsample", 0.0, 1.0),
        'colsample_bytree':trial.suggest_uniform("colsample_bytree", 0.0, 1.0),
    }

    model = XGBClassifier(**params)
    cv = KFold(n_splits=3, shuffle=True, random_state=None)
    scorer = make_scorer(f1_score, greater_is_better=True)

    bst = xgb.train(params, dtrain)
    preds = bst.predict(dvalid)
```

```
    pred_labels = np.rint(preds)
    f1_scores = f1_score(y_test, pred_labels, average='micro')
    return f1_scores
```

[63]:

```
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=100, timeout=600)
```

```
[I 2022-01-31 09:36:36,444] A new study created in memory with name: no-name-81c9b5ae-266e-462e-8612-97a102499a16
[I 2022-01-31 09:36:36,549] Trial 0 finished with value: 0.6858474318711377 and parameters: {'booster': 'gblinear', 'learning_rate': 0.06978904317996577, 'max_depth': 6, 's
ubsample': 0.9356262269592085, 'colsample_bytree': 0.31565247845473343}. Best is trial 0 with value: 0.6858474318711377.
[09:36:36] WARNING: ../src/learner.cc:576:
Parameters: { "colsample_bytree", "max_depth", "subsample" } might not be used.

  This could be a false alarm, with some parameters getting used by language bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually being used
  but getting flagged wrongly here. Please open an issue if you find any such cases.


[I 2022-01-31 09:36:37,011] Trial 1 finished with value: 0.7004356194914396 and parameters: {'booster': 'gbtree', 'learning_rate': 0.07173704735625912, 'max_depth': 5, 'sub
sample': 0.8638878938477487, 'colsample_bytree': 0.9975799460850865}. Best is trial 1 with value: 0.7004356194914396.
[I 2022-01-31 09:36:37,610] Trial 2 finished with value: 0.695167662850775 and parameters: {'booster': 'dart', 'learning_rate': 0.04382555397071061, 'max_depth': 10, 'subsa
mple': 0.13852027249682264, 'colsample_bytree': 0.6641450086117234}. Best is trial 1 with value: 0.7004356194914396.
[I 2022-01-31 09:36:37,837] Trial 3 finished with value: 0.6895957856346875 and parameters: {'booster': 'dart', 'learning_rate': 0.07349061085539796, 'max_depth': 5, 'subsa
mple': 0.22800641244192255, 'colsample_bytree': 0.20003343219851843}. Best is trial 1 with value: 0.7004356194914396.
[I 2022-01-31 09:36:38,387] Trial 4 finished with value: 0.7078310201600647 and parameters: {'booster': 'gbtree', 'learning_rate': 0.01303026709851739, 'max_depth': 9, 'su
bsample': 0.11263090322674385, 'colsample_bytree': 0.9824350314546796}. Best is trial 4 with value: 0.7078310201600647.
[I 2022-01-31 09:36:38,479] Trial 5 finished with value: 0.6938506736906088 and parameters: {'booster': 'gblinear', 'learning_rate': 0.0376409904857424, 'max_depth': 6, 'su
bsample': 0.6357376351259171, 'colsample_bytree': 0.23948830183893877}. Best is trial 4 with value: 0.7078310201600647.
[09:36:38] WARNING: ../src/learner.cc:576:
Parameters: { "colsample_bytree", "max_depth", "subsample" } might not be used.

  This could be a false alarm, with some parameters getting used by language bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually being used
  but getting flagged wrongly here. Please open an issue if you find any such cases.


[I 2022-01-31 09:36:38,985] Trial 6 finished with value: 0.7079323270185391 and parameters: {'booster': 'gbtree', 'learning_rate': 0.028045617713100244, 'max_depth': 10, 's
ubsample': 0.35771348116623136, 'colsample_bytree': 0.4429199521044903}. Best is trial 6 with value: 0.7079323270185391.
[I 2022-01-31 09:36:39,275] Trial 7 finished with value: 0.697700334312633 and parameters: {'booster': 'dart', 'learning_rate': 0.057598783424493524, 'max_depth': 5, 'subsa
mple': 0.49496425109357645, 'colsample_bytree': 0.31719596815568096}. Best is trial 6 with value: 0.7079323270185391.
[I 2022-01-31 09:36:39,370] Trial 8 finished with value: 0.6946611285584035 and parameters: {'booster': 'gblinear', 'learning_rate': 0.019116512101822268, 'max_depth': 7,
```

```
sample': 0.8875060006330022, 'colsample_bytree': 0.3509465094479508}. Best is trial 44 with value: 0.7207982980447776.
[I 2022-01-31 09:37:09,668] Trial 46 finished with value: 0.7185695471583426 and parameters: {'booster': 'dart', 'learning_rate': 0.02725459667642343, 'max_depth': 11, 'sub
sample': 0.8546276145869144, 'colsample_bytree': 0.9073816185461118}. Best is trial 44 with value: 0.7207982980447776.
[I 2022-01-31 09:37:10,868] Trial 47 finished with value: 0.7225205146388409 and parameters: {'booster': 'dart', 'learning_rate': 0.0559116972340968, 'max_depth': 11, 'subs
ample': 0.8109503247655565, 'colsample_bytree': 0.9180388244460668}. Best is trial 47 with value: 0.7225205146388409.
[I 2022-01-31 09:37:11,962] Trial 48 finished with value: 0.7140107385269983 and parameters: {'booster': 'dart', 'learning_rate': 0.05134893803041909, 'max_depth': 10, 'sub
sample': 0.7995080412618492, 'colsample_bytree': 0.9992896596601138}. Best is trial 47 with value: 0.7225205146388409.
[I 2022-01-31 09:37:13,155] Trial 49 finished with value: 0.7199878431769831 and parameters: {'booster': 'dart', 'learning_rate': 0.06454313644574437, 'max_depth': 11, 'sub
sample': 0.9204498607877944, 'colsample_bytree': 0.892668973638816}. Best is trial 47 with value: 0.7225205146388409.
[I 2022-01-31 09:37:14,167] Trial 50 finished with value: 0.7185695471583426 and parameters: {'booster': 'dart', 'learning_rate': 0.060248135916751995, 'max_depth': 10, 'su
bsample': 0.9214947866551296, 'colsample_bytree': 0.8783731647606514}. Best is trial 47 with value: 0.7225205146388409.
[I 2022-01-31 09:37:15,500] Trial 51 finished with value: 0.7221152872049438 and parameters: {'booster': 'dart', 'learning_rate': 0.0827544941242365, 'max_depth': 11, 'subs
ample': 0.9626860694949235, 'colsample_bytree': 0.9578943042505037}. Best is trial 47 with value: 0.7225205146388409.
[I 2022-01-31 09:37:16,762] Trial 52 finished with value: 0.7184682402998683 and parameters: {'booster': 'dart', 'learning_rate': 0.07884451140795008, 'max_depth': 11, 'sub
sample': 0.9611387544990767, 'colsample_bytree': 0.9443977028939987}. Best is trial 47 with value: 0.7225205146388409.
[I 2022-01-31 09:37:17,179] Trial 53 finished with value: 0.6974977205956844 and parameters: {'booster': 'dart', 'learning_rate': 0.0835645597975643, 'max_depth': 4, 'subsa
mple': 0.8921848689686307, 'colsample_bytree': 0.7887662579725412}. Best is trial 47 with value: 0.7225205146388409.
[I 2022-01-31 09:37:18,426] Trial 54 finished with value: 0.7185695471583426 and parameters: {'booster': 'dart', 'learning_rate': 0.06513016742008476, 'max_depth': 11, 'sub
sample': 0.964107850772606, 'colsample_bytree': 0.8903160811826597}. Best is trial 47 with value: 0.7225205146388409.
[I 2022-01-31 09:37:19,690] Trial 55 finished with value: 0.7188734677337657 and parameters: {'booster': 'dart', 'learning_rate': 0.09679142674751524, 'max_depth': 11, 'sub
sample': 0.8264080236895822, 'colsample_bytree': 0.914271330812408}. Best is trial 47 with value: 0.7225205146388409.
[I 2022-01-31 09:37:20,723] Trial 56 finished with value: 0.7180630128659711 and parameters: {'booster': 'dart', 'learning_rate': 0.05686516483766689, 'max_depth': 10, 'sub
sample': 0.9947833944897158, 'colsample_bytree': 0.8542756951730636}. Best is trial 47 with value: 0.7225205146388409.
[I 2022-01-31 09:37:21,755] Trial 57 finished with value: 0.7208996049032521 and parameters: {'booster': 'dart', 'learning_rate': 0.06960919279386434, 'max_depth': 11, 'sub
sample': 0.914887580750312, 'colsample_bytree': 0.7009614677768359}. Best is trial 47 with value: 0.7225205146388409.
[I 2022-01-31 09:37:22,651] Trial 58 finished with value: 0.7162394894134333 and parameters: {'booster': 'dart', 'learning_rate': 0.08662592558464569, 'max_depth': 10, 'sub
sample': 0.9287285392963995, 'colsample_bytree': 0.7211475647538554}. Best is trial 47 with value: 0.7225205146388409.
[I 2022-01-31 09:37:23,774] Trial 59 finished with value: 0.7178603991490224 and parameters: {'booster': 'dart', 'learning_rate': 0.0694620194943955, 'max_depth': 11, 'subs
ample': 0.8726077989346374, 'colsample_bytree': 0.7920385546640056}. Best is trial 47 with value: 0.7225205146388409.
[I 2022-01-31 09:37:24,562] Trial 60 finished with value: 0.7101610779049742 and parameters: {'booster': 'dart', 'learning_rate': 0.057058336691714465, 'max_depth': 10, 'su
bsample': 0.7355485892698185, 'colsample_bytree': 0.5323053693372939}. Best is trial 47 with value: 0.7225205146388409.
[I 2022-01-31 09:37:25,829] Trial 61 finished with value: 0.7232296626481612 and parameters: {'booster': 'dart', 'learning_rate': 0.07498512878761832, 'max_depth': 11, 'sub
sample': 0.944105508664244, 'colsample_bytree': 0.9569175142793807}. Best is trial 61 with value: 0.7232296626481612.
[I 2022-01-31 09:37:27,145] Trial 62 finished with value: 0.7176577854320737 and parameters: {'booster': 'dart', 'learning_rate': 0.07298159238965818, 'max_depth': 11, 'sub
sample': 0.9524412962597251, 'colsample_bytree': 0.9569688150949434}. Best is trial 61 with value: 0.7232296626481612.
[I 2022-01-31 09:37:29,061] Trial 63 finished with value: 0.7210009117617263 and parameters: {'booster': 'dart', 'learning_rate': 0.07762484032585577, 'max_depth': 11, 'sub
sample': 0.9107469611386291, 'colsample_bytree': 0.8745246607908322}. Best is trial 61 with value: 0.7232296626481612.
[I 2022-01-31 09:37:30,058] Trial 64 finished with value: 0.7184682402998683 and parameters: {'booster': 'dart', 'learning_rate': 0.07920468577833928, 'max_depth': 10, 'sub
sample': 0.9818551481283183, 'colsample_bytree': 0.8616859851558136}. Best is trial 61 with value: 0.7232296626481612.
[I 2022-01-31 09:37:31,306] Trial 65 finished with value: 0.7221152872049438 and parameters: {'booster': 'dart', 'learning_rate': 0.0905823832828135, 'max_depth': 11, 'subs
ample': 0.6955998388423109, 'colsample_bytree': 0.9567760508526284}. Best is trial 61 with value: 0.7232296626481612.
[I 2022-01-31 09:37:31,414] Trial 66 finished with value: 0.6798703272211529 and parameters: {'booster': 'gblinear', 'learning_rate': 0.0894222428617476, 'max_depth': 11,
```

```
sample': 0.7922526198957557, 'colsample_bytree': 0.9651971030949537}. Best is trial 81 with value: 0.7244453449498531.
[I 2022-01-31 09:37:47,427] Trial 84 finished with value: 0.7149225002532672 and parameters: {'booster': 'dart', 'learning_rate': 0.07630529153008128, 'max_depth': 11, 'sub
sample': 0.7509131408619348, 'colsample_bytree': 0.740918505457087}. Best is trial 81 with value: 0.7244453449498531.
[I 2022-01-31 09:37:47,536] Trial 85 finished with value: 0.6841252152770743 and parameters: {'booster': 'gblinear', 'learning_rate': 0.07492035099027723, 'max_depth': 10,
'subsample': 0.7032000123565023, 'colsample_bytree': 0.865882617676091}. Best is trial 81 with value: 0.7244453449498531.
[09:37:47] WARNING: ../src/learner.cc:576:
Parameters: { "colsample_bytree", "max_depth", "subsample" } might not be used.

  This could be a false alarm, with some parameters getting used by language bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually being used
  but getting flagged wrongly here. Please open an issue if you find any such cases.


[I 2022-01-31 09:37:48,735] Trial 86 finished with value: 0.7180630128659711 and parameters: {'booster': 'dart', 'learning_rate': 0.0825296981675889, 'max_depth': 11, 'subs
ample': 0.6673218048571085, 'colsample_bytree': 0.798441762214283}. Best is trial 81 with value: 0.7244453449498531.
[I 2022-01-31 09:37:49,796] Trial 87 finished with value: 0.7135042042346268 and parameters: {'booster': 'dart', 'learning_rate': 0.05996112688368001, 'max_depth': 10, 'sub
sample': 0.601807329123391, 'colsample_bytree': 0.9382692794292272}. Best is trial 81 with value: 0.7244453449498531.
[I 2022-01-31 09:37:50,583] Trial 88 finished with value: 0.7047918144058353 and parameters: {'booster': 'dart', 'learning_rate': 0.0724712807754476, 'max_depth': 11, 'subs
ample': 0.11819273712155098, 'colsample_bytree': 0.9054092867041406}. Best is trial 81 with value: 0.7244453449498531.
[I 2022-01-31 09:37:51,174] Trial 89 finished with value: 0.6716644716847331 and parameters: {'booster': 'dart', 'learning_rate': 0.0953399837262342, 'max_depth': 11, 'subs
ample': 0.024339251982770815, 'colsample_bytree': 0.8438135682746373}. Best is trial 81 with value: 0.7244453449498531.
[I 2022-01-31 09:37:52,246] Trial 90 finished with value: 0.7181643197244454 and parameters: {'booster': 'dart', 'learning_rate': 0.08746160245246554, 'max_depth': 11, 'sub
sample': 0.8090425727163245, 'colsample_bytree': 0.7651796504887598}. Best is trial 81 with value: 0.7244453449498531.
[I 2022-01-31 09:37:53,385] Trial 91 finished with value: 0.7193800020261372 and parameters: {'booster': 'dart', 'learning_rate': 0.08722027630027185, 'max_depth': 11, 'sub
sample': 0.8461096208910046, 'colsample_bytree': 0.8232368356569104}. Best is trial 81 with value: 0.7244453449498531.
[I 2022-01-31 09:37:54,549] Trial 92 finished with value: 0.7247492655252761 and parameters: {'booster': 'dart', 'learning_rate': 0.08133657073994492, 'max_depth': 11, 'sub
sample': 0.7712267687303078, 'colsample_bytree': 0.873733282499765}. Best is trial 92 with value: 0.7247492655252761.
[I 2022-01-31 09:37:55,564] Trial 93 finished with value: 0.718366933441394 and parameters: {'booster': 'dart', 'learning_rate': 0.08053821011017742, 'max_depth': 10, 'subs
ample': 0.7762580286016247, 'colsample_bytree': 0.882802654929491}. Best is trial 92 with value: 0.7247492655252761.
[I 2022-01-31 09:37:56,810] Trial 94 finished with value: 0.7187721608752913 and parameters: {'booster': 'dart', 'learning_rate': 0.06247534670308462, 'max_depth': 11, 'sub
sample': 0.7292824031719988, 'colsample_bytree': 0.9693211993256936}. Best is trial 92 with value: 0.7247492655252761.
[I 2022-01-31 09:37:58,025] Trial 95 finished with value: 0.7206969911863033 and parameters: {'booster': 'dart', 'learning_rate': 0.05430598142290329, 'max_depth': 11, 'sub
sample': 0.7543440046258224, 'colsample_bytree': 0.9176607910048715}. Best is trial 92 with value: 0.7247492655252761.
[I 2022-01-31 09:37:59,123] Trial 96 finished with value: 0.7155303414041131 and parameters: {'booster': 'dart', 'learning_rate': 0.0666834805354436, 'max_depth': 10, 'subs
ample': 0.8198190435038502, 'colsample_bytree': 0.9318438051580042}. Best is trial 92 with value: 0.7247492655252761.
[I 2022-01-31 09:38:00,310] Trial 97 finished with value: 0.7214061391956236 and parameters: {'booster': 'dart', 'learning_rate': 0.07117176877750485, 'max_depth': 11, 'sub
sample': 0.8717963590052301, 'colsample_bytree': 0.8923774456378022}. Best is trial 92 with value: 0.7247492655252761.
[I 2022-01-31 09:38:02,361] Trial 98 finished with value: 0.7208996049032521 and parameters: {'booster': 'dart', 'learning_rate': 0.09123408320190314, 'max_depth': 11, 'sub
sample': 0.7838973679124237, 'colsample_bytree': 0.9826917253042233}. Best is trial 92 with value: 0.7247492655252761.
[I 2022-01-31 09:38:02,595] Trial 99 finished with value: 0.6957755040016209 and parameters: {'booster': 'dart', 'learning_rate': 0.0839945944092854, 'max_depth': 11, 'subs
ample': 0.9403009976534672, 'colsample_bytree': 0.037944462324854655}. Best is trial 92 with value: 0.7247492655252761.
```

```
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=100, timeout=600)
```

```
new_params = study.best_params

new_model = XGBClassifier(**new_params)
new_model.fit(X, y)
preds = new_model.predict(X_test)

print('Optimized SuperLearner accuracy: ', accuracy_score(y_test, preds))
print('Optimized SuperLearner f1-score: ', f1_score(y_test, preds, average='micro'))
```

[64]:

```
new_params = study.best_params

new_model = XGBClassifier(**new_params)
new_model.fit(X, y)
preds = new_model.predict(X_test)

print('Optimized SuperLearner accuracy: ', accuracy_score(y_test, preds))
print('Optimized SuperLearner f1-score: ', f1_score(y_test, preds, average='micro'))
```

```
/opt/conda/lib/python3.7/site-packages/xgboost/sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be removed in a future releas
e. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starti
ng with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[09:47:49] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to
'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Optimized SuperLearner accuracy:  0.9196636612298653
Optimized SuperLearner f1-score:  0.9196636612298653
```

```
print("Number of finished trials: ", len(study.trials))
print("Best trial:")
trial = study.best_trial
```

```
print("  Value: {}".format(trial.value))
print("  Params: ")
for key, value in trial.params.items():
    print("    {}: {}".format(key, value))
```

```
print('Optimized SuperLearner accuracy: ', accuracy_score(y_test, preds))
print('Optimized SuperLearner f1-score: ', f1_score(y_test, preds, average='micro'))
```

```
/opt/conda/lib/python3.7/site-packages/xgboost/sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be removed in a future releas
e. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starti
ng with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[09:47:49] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to
'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Optimized SuperLearner accuracy:  0.9196636612298653
Optimized SuperLearner f1-score:  0.9196636612298653
```

```
print("Number of finished trials: ", len(study.trials))
print("Best trial:")
trial = study.best_trial

print("  Value: {}".format(trial.value))
print("  Params: ")
for key, value in trial.params.items():
    print("    {}: {}".format(key, value))
```

```
Number of finished trials:  100
Best trial:
  Value: 0.7247492655252761
  Params:
    booster: dart
    learning_rate: 0.08133657073994492
    max_depth: 11
    subsample: 0.7712267687303078
    colsample_bytree: 0.873733282499765
```

```
print("All of accuracies")
print(scores)

print("Mean of accuracies")
print(np.mean(scores))
```

```
[66]:
print("All of accuracies")
print(scores)

print("Mean of accuracies")
print(np.mean(scores))
```

```
All of accuracies
[0.7487653539318728, 0.7418946301925026, 0.7415146909827761, 0.7426545086119554, 0.7387284701114488]
Mean of accuracies
0.7427115307661111
```

Вывод:

```python
print('Optimized SuperLearner f1-score: ', f1_score(y_test, preds, average='micro'))
```

```
/opt/conda/lib/python3.7/site-packages/xgboost/sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[09:47:49] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Optimized SuperLearner accuracy:  0.9196636612298653
Optimized SuperLearner f1-score:  0.9196636612298653
```

]:

```python
print("Number of finished trials: ", len(study.trials))
print("Best trial:")
trial = study.best_trial

print("  Value: {}".format(trial.value))
print("  Params: ")
for key, value in trial.params.items():
    print("    {}: {}".format(key, value))
```

```
Number of finished trials:  100
Best trial:
  Value: 0.7247492655252761
  Params:
    booster: dart
    learning_rate: 0.08133657073994492
    max_depth: 11
    subsample: 0.7712267687303078
    colsample_bytree: 0.873733282499765
```

]:

```python
print("All of accuracies")
print(scores)

print("Mean of accuracies")
print(np.mean(scores))
```

```
All of accuracies
[0.7487653539318728, 0.7418946301925026, 0.7415146909827761, 0.7426545086119554, 0.7387284701114488]
Mean of accuracies
0.7427115307661111
```