

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

Steel Plate Defect Prediction

[Kaggle competition 'Steel Plate Defect Prediction'](#)

Описание исходного датасета:

Датасет содержит **27 признаков**, которые описывают стальные пластины и, связанные с ними измерения. В качестве ответа **7 классов**, которые указывают на наличие определенного дефекта.

Задача:

Построить модель многоклассовой классификации для прогнозирования дефектов стальных пластин

EDA

```
In [2]: data = pd.read_csv('./train.csv')
```

```
In [3]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19219 entries, 0 to 19218
Data columns (total 35 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   id                                    19219 non-null  int64
1   X_Minimum                            19219 non-null  int64
2   X_Maximum                            19219 non-null  int64
3   Y_Minimum                            19219 non-null  int64
4   Y_Maximum                            19219 non-null  int64
5   Pixels_Areas                         19219 non-null  int64
6   X_Perimeter                          19219 non-null  int64
7   Y_Perimeter                          19219 non-null  int64
8   Sum_of_Luminosity                    19219 non-null  int64
9   Minimum_of_Luminosity                19219 non-null  int64
10  Maximum_of_Luminosity                 19219 non-null  int64
11  Length_of_Conveyer                   19219 non-null  int64
12  TypeOfSteel_A300                     19219 non-null  int64
13  TypeOfSteel_A400                     19219 non-null  int64
14  Steel_Plate_Thickness                 19219 non-null  int64
15  Edges_Index                           19219 non-null  float64
16  Empty_Index                           19219 non-null  float64
17  Square_Index                          19219 non-null  float64
18  Outside_X_Index                       19219 non-null  float64
19  Edges_X_Index                         19219 non-null  float64
20  Edges_Y_Index                         19219 non-null  float64
21  Outside_Global_Index                  19219 non-null  float64
22  LogOfAreas                           19219 non-null  float64
23  Log_X_Index                           19219 non-null  float64
24  Log_Y_Index                           19219 non-null  float64
25  Orientation_Index                     19219 non-null  float64
26  Luminosity_Index                      19219 non-null  float64
27  SigmoidOfAreas                       19219 non-null  float64
28  Pastry                                19219 non-null  int64
29  Z_Scratch                             19219 non-null  int64
30  K_Scratch                             19219 non-null  int64
31  Stains                                19219 non-null  int64
32  Dirtiness                             19219 non-null  int64
33  Bumps                                 19219 non-null  int64
34  Other_Faults                          19219 non-null  int64
dtypes: float64(13), int64(22)
memory usage: 5.1 MB
```

Данные состоят из 34 колонок:

- 0 - id
- 1-27 - признаки
- 28-34 - таргеты

По информации видим, что все данные представлены в числовом виде.

```
In [4]: # признаки (сразу убираем id)
```

```
features = data.columns[1:28]
# классы дефектов
targets = data.columns[28:]
```

Признаки

Посмотрим сводную статистику по признакам

```
In [5]: data[features[0:10]].describe()
```

```
Out[5]:
```

| | X_Minimum | X_Maximum | Y_Minimum | Y_Maximum | Pixels_Areas | X_Perimeter | Y_Perimeter | Sum_of_Luminos |
|-------|--------------|--------------|--------------|--------------|---------------|--------------|--------------|----------------|
| count | 19219.000000 | 19219.000000 | 1.921900e+04 | 1.921900e+04 | 19219.000000 | 19219.000000 | 19219.000000 | 1.921900e+ |
| mean | 709.854675 | 753.857641 | 1.849756e+06 | 1.846605e+06 | 1683.987616 | 95.654665 | 64.124096 | 1.918467e+ |
| std | 531.544189 | 499.836603 | 1.903554e+06 | 1.896295e+06 | 3730.319865 | 177.821382 | 101.054178 | 4.420247e+ |
| min | 0.000000 | 4.000000 | 6.712000e+03 | 6.724000e+03 | 6.000000 | 2.000000 | 1.000000 | 2.500000e+ |
| 25% | 49.000000 | 214.000000 | 6.574680e+05 | 6.575020e+05 | 89.000000 | 15.000000 | 14.000000 | 9.848000e+ |
| 50% | 777.000000 | 796.000000 | 1.398169e+06 | 1.398179e+06 | 168.000000 | 25.000000 | 23.000000 | 1.823800e+ |
| 75% | 1152.000000 | 1165.000000 | 2.368032e+06 | 2.362511e+06 | 653.000000 | 64.000000 | 61.000000 | 6.797800e+ |
| max | 1705.000000 | 1713.000000 | 1.298766e+07 | 1.298769e+07 | 152655.000000 | 7553.000000 | 903.000000 | 1.159141e+ |

- "Y" признаки очень сильно выбиваются по масштабу - необходима нормировка данных
- max, min, sum, perimeter стоит проверить на корреляцию признаков

```
In [6]: data[features[10:20]].describe()
```

```
Out[6]:
```

| | Length_of_Conveyer | TypeOfSteel_A300 | TypeOfSteel_A400 | Steel_Plate_Thickness | Edges_Index | Empty_Index | Squar |
|-------|--------------------|------------------|------------------|-----------------------|--------------|--------------|-------|
| count | 19219.000000 | 19219.000000 | 19219.000000 | 19219.000000 | 19219.000000 | 19219.000000 | 19219 |
| mean | 1459.350747 | 0.402674 | 0.596337 | 76.213122 | 0.352939 | 0.409309 | 0 |
| std | 145.568687 | 0.490449 | 0.490644 | 53.931960 | 0.318976 | 0.124143 | 0 |
| min | 1227.000000 | 0.000000 | 0.000000 | 40.000000 | 0.000000 | 0.000000 | 0 |
| 25% | 1358.000000 | 0.000000 | 0.000000 | 40.000000 | 0.058600 | 0.317500 | 0 |
| 50% | 1364.000000 | 0.000000 | 1.000000 | 69.000000 | 0.238500 | 0.413500 | 0 |
| 75% | 1652.000000 | 1.000000 | 1.000000 | 80.000000 | 0.656100 | 0.494600 | 0 |
| max | 1794.000000 | 1.000000 | 1.000000 | 300.000000 | 0.995200 | 0.927500 | 1 |

- "TypeOfSteel_A300", "TypeOfSteel_A400", возможно, бинарные признаки

```
In [34]: data[features[20:]].describe()
```

```
Out[34]:
```

| | Outside_Global_Index | LogOfAreas | Log_X_Index | Log_Y_Index | Orientation_Index | Luminosity_Index | SigmoidOfArea |
|-------|----------------------|--------------|--------------|--------------|-------------------|------------------|---------------|
| count | 19219.000000 | 19219.000000 | 19219.000000 | 19219.000000 | 19219.000000 | 19219.000000 | 19219.000000 |
| mean | 0.591899 | 2.473475 | 1.312667 | 1.389737 | 0.102742 | -0.138382 | 0.57190 |
| std | 0.482050 | 0.760575 | 0.467848 | 0.405549 | 0.487681 | 0.120344 | 0.33221 |
| min | 0.000000 | 0.778200 | 0.301000 | 0.000000 | -0.988400 | -0.885000 | 0.11900 |
| 25% | 0.000000 | 1.949400 | 1.000000 | 1.079200 | -0.272700 | -0.192500 | 0.25320 |
| 50% | 1.000000 | 2.227900 | 1.146100 | 1.322200 | 0.111100 | -0.142600 | 0.47290 |
| 75% | 1.000000 | 2.814900 | 1.431400 | 1.707600 | 0.529400 | -0.084000 | 0.99940 |
| max | 1.000000 | 4.554300 | 2.997300 | 4.033300 | 0.991700 | 0.642100 | 1.00000 |

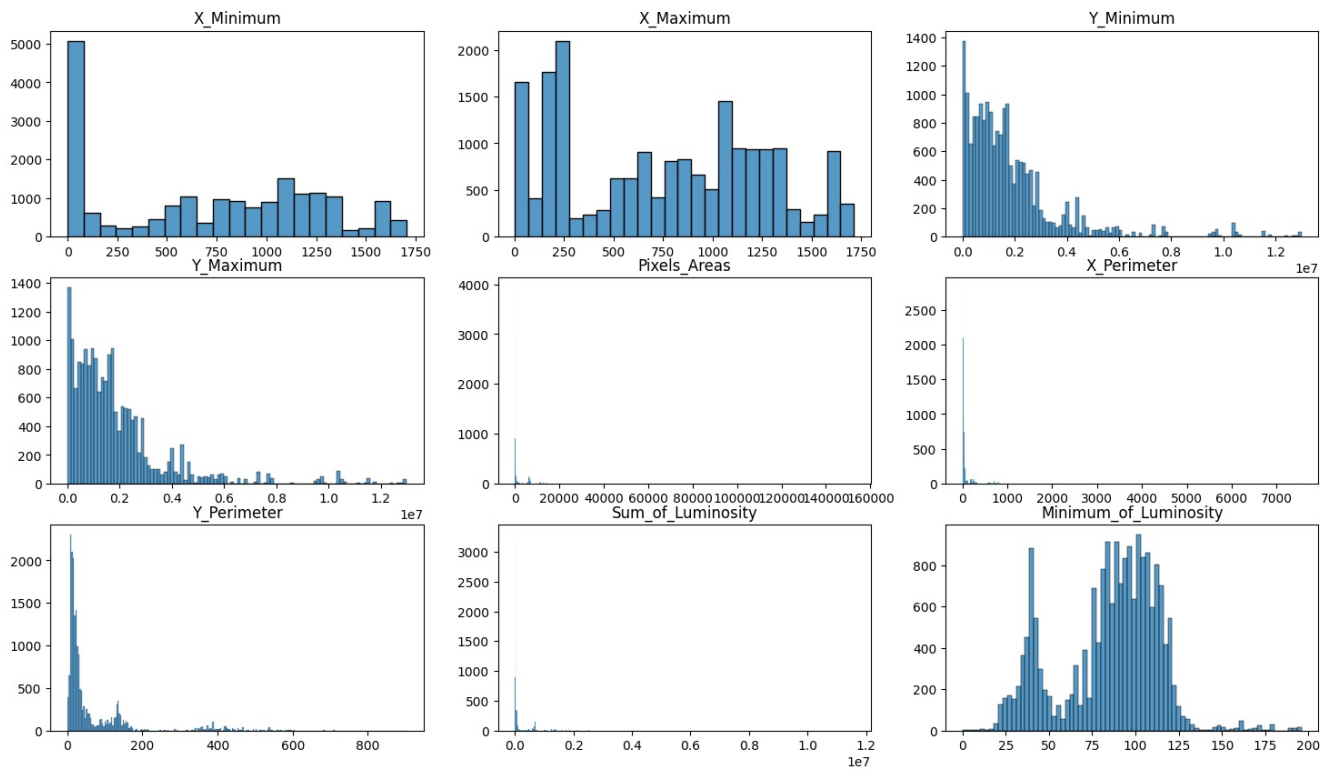
- "LogOfAreas", "SigmoidOfAreas" - похожи на производные признаки от Areas, тоже стоит проверить на мультиколлинеарность

Вывод:

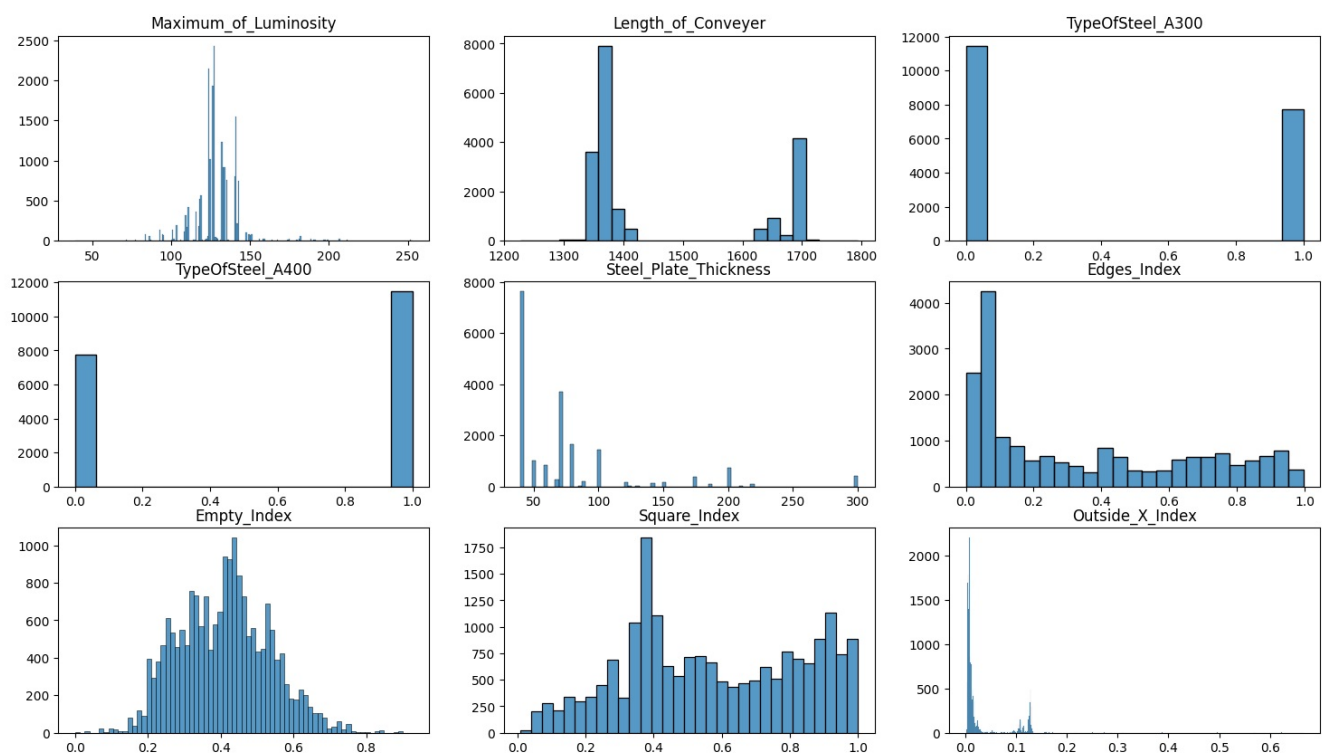
- нормировка
- проверка на мультиколлинеарность

Отрисует признаки на графиках

```
In [8]: fig, ax = plt.subplots(3,3,figsize=(18,10))
for i in range(len(features[:9])):
    sns.histplot(data[features[i]],ax=ax[i//3,i%3])
    ax[i//3,i%3].set_title(features[i])
    ax[i//3,i%3].set_ylabel(' ')
    ax[i//3,i%3].set_xlabel(' ')
```

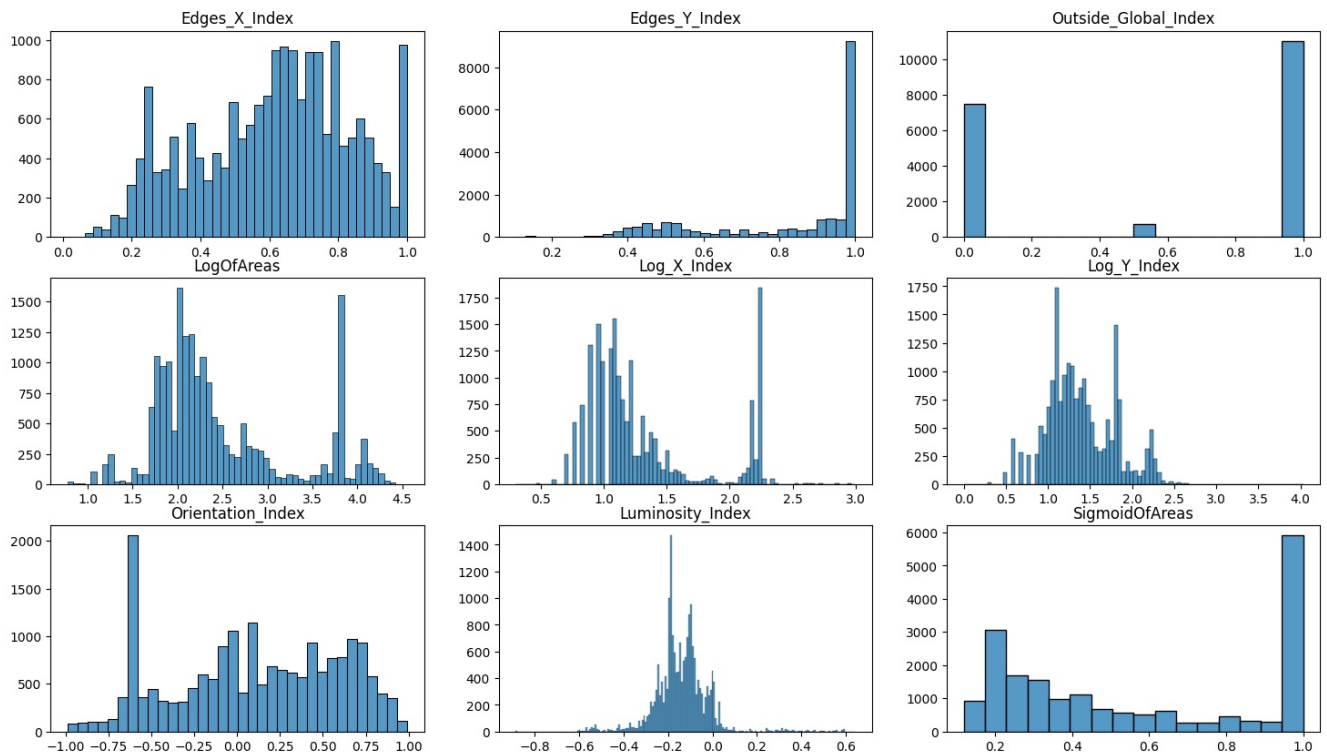


```
In [9]: fig, ax = plt.subplots(3,3,figsize=(18,10))
for i in range(len(features[9:18])):
    sns.histplot(data[features[i+9]],ax=ax[i//3,i%3])
    ax[i//3,i%3].set_title(features[i+9])
    ax[i//3,i%3].set_ylabel(' ')
    ax[i//3,i%3].set_xlabel(' ')
```



```
In [10]: fig, ax = plt.subplots(3,3,figsize=(18,10))
for i in range(len(features[18:])):
    sns.histplot(data[features[i+18]],ax=ax[i//3,i%3])
    ax[i//3,i%3].set_title(features[i+18])
    ax[i//3,i%3].set_ylabel(' ')
```

```
ax[i//3,i%3].set_xlabel('')
```

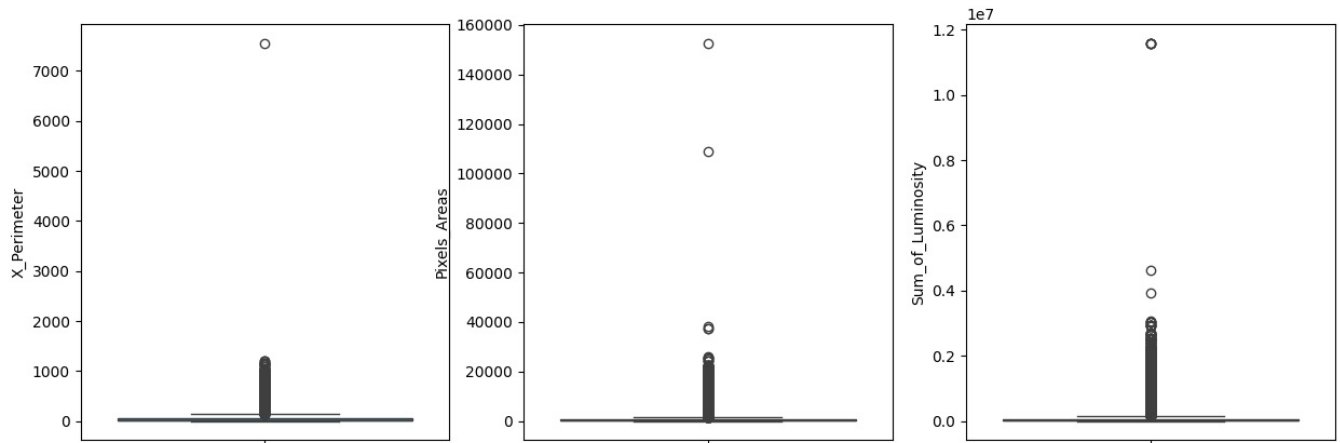


Вывод по графикам:

- "X_Perimeter", "Pixels_Areas", "Sum_Of_Luminosity" - проверить на выбросы
- "Outside_Global_Index" - категориальная (3 категории)?
- "TypeOfSteel_X" - бинарные

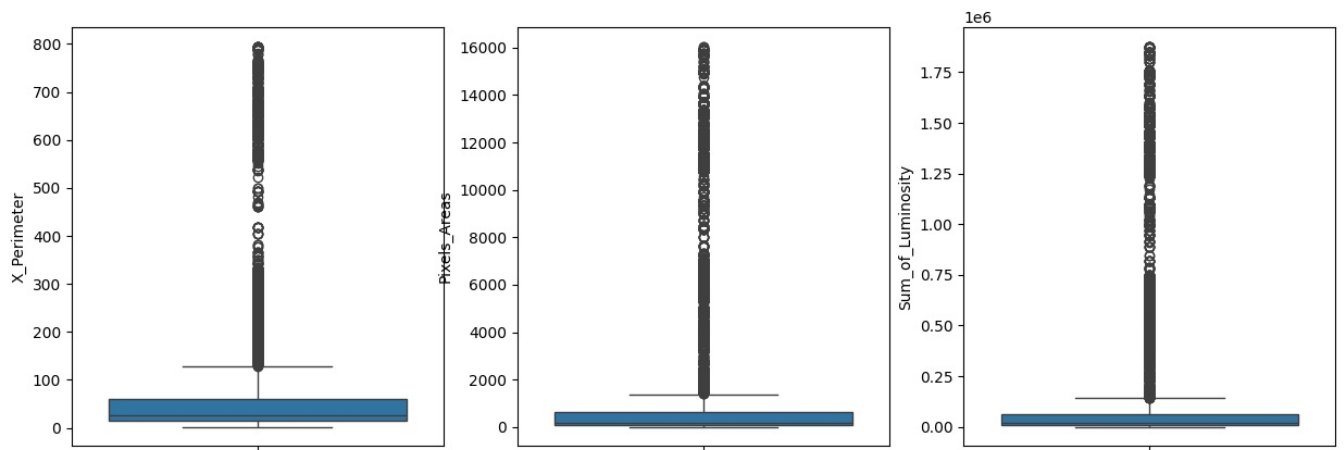
Выбросы

```
In [11]: cols = "X_Perimeter", "Pixels_Areas", "Sum_of_Luminosity"
fig,ax = plt.subplots(1,3,figsize=(15,5))
for i in range(len(cols)):
    sns.boxplot(data[cols[i]],ax=ax[i])
```



Посмотрим, что будет если отбросим 1% экстремальных значений

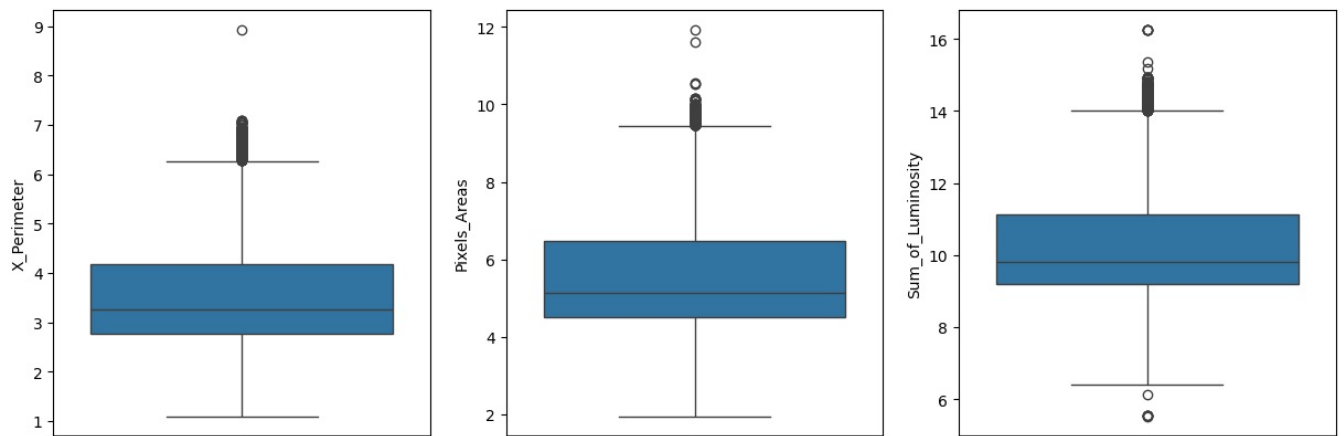
```
In [12]: cols = "X_Perimeter", "Pixels_Areas", "Sum_of_Luminosity"
fig,ax = plt.subplots(1,3,figsize=(15,5))
k = int(data.shape[0]*0.99)
for i in range(len(cols)):
    sns.boxplot(data[cols[i]].sort_values()[:k],ax=ax[i])
```



Графики получаются лучше, но теперь видно, что присутствуют длинные "хвосты" распределения

Применим **log1p** преобразование

```
In [13]: cols = "X_Perimeter", "Pixels_Areas", "Sum_of_Luminosity"
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
for i in range(len(cols)):
    sns.boxplot(np.log1p(data[cols[i]]), ax=ax[i])
```



Вывод:

Думаю оставить строки, как есть, но применить к столбцам **log1p** преобразование, чтобы избавиться от влияние "хвостов"

Бинарные и категориальные переменные

```
In [14]: feature = 'Outside_Global_Index'
data[feature].value_counts()
```

```
Out[14]:
```

| | count |
|----------------------|-------|
| Outside_Global_Index | |
| 1.0 | 11022 |
| 0.0 | 7490 |
| 0.5 | 706 |
| 0.7 | 1 |

dtype: int64

```
In [15]: feature = 'TypeOfSteel_A300'
data[feature].value_counts()
```

```
Out[15]:
```

| | count |
|------------------|-------|
| TypeOfSteel_A300 | |
| 0 | 11480 |
| 1 | 7739 |

dtype: int64

```
In [16]: feature = 'TypeOfSteel_A400'
data[feature].value_counts()
```

```
Out[16]:
```

| | count |
|------------------|-------|
| TypeOfSteel_A400 | |
| 1 | 11461 |
| 0 | 7758 |

dtype: int64

Вывод:

"TypeOfSteel_X" - бинарные

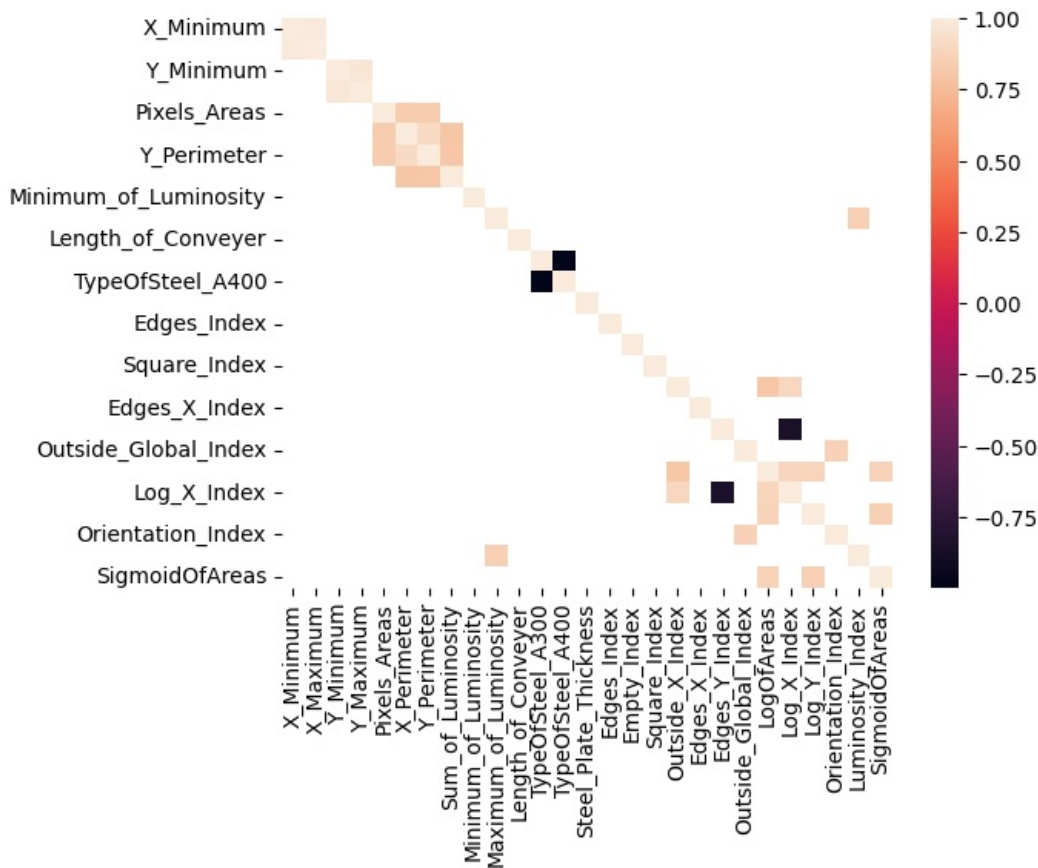
"Outside_Global_Index" - непонятен тип признака, возможно категориальный

Корреляция

Посчитаем корреляцию и выделим те значения, которые больше 80%

```
In [17]: p = 0.8
sns.heatmap(data[features].corr(),mask=np.abs(data[features].corr())<p)
```

```
Out[17]: <Axes: >
```



Посмотрим на все такие пары

```
In [18]: data_corr = data[features].corr()
idx = np.where(np.abs(data_corr.mask(np.eye(len(data_corr), dtype=bool)))>p)

pairs = list(set(tuple(sorted(x)) for x in list(zip(idx[0], idx[1]))))
pd.Series(pairs)

for i,j in pairs:
    print(features[i], '\t\t', features[j], '\t\t', data_corr.loc[features[i], features[j]])
```

| | | |
|-----------------------|-------------------|---------------------|
| X_Minimum | X_Maximum | 0.9897672786491646 |
| LogOfAreas | SigmoidOfAreas | 0.8724531424401515 |
| Outside_Global_Index | Orientation_Index | 0.8639872069439484 |
| Log_Y_Index | SigmoidOfAreas | 0.8524500761700224 |
| Pixels_Areas | Y_Perimeter | 0.8345434453321343 |
| LogOfAreas | Log_X_Index | 0.8882390325856594 |
| Maximum_of_Luminosity | Luminosity_Index | 0.8538555122645558 |
| X_Perimeter | Sum_of_Luminosity | 0.8020724927380157 |
| Y_Minimum | Y_Maximum | 0.9695524650996821 |
| Y_Perimeter | Sum_of_Luminosity | 0.8091705936142481 |
| Pixels_Areas | X_Perimeter | 0.835078733573749 |
| TypeOfSteel_A300 | TypeOfSteel_A400 | -0.9977316148553689 |
| Outside_X_Index | Log_X_Index | 0.8983409899912759 |
| X_Perimeter | Y_Perimeter | 0.9125793077170433 |
| Edges_Y_Index | Log_X_Index | -0.8518128006577472 |
| Outside_X_Index | LogOfAreas | 0.8119602292784268 |
| LogOfAreas | Log_Y_Index | 0.8792281394891333 |

```
In [19]: indx_x = list(map(lambda x:(features[x[0]],features[x[1]]),pairs))
vals_x = list(map(lambda x:data_corr.loc[x[0],x[1]],indx_x))

pd.Series(data=vals_x,index=indx_x,name='corr').sort_values(ascending=False)
```

```
Out[19]:
```

| | corr |
|---|-----------|
| (X_Minimum, X_Maximum) | 0.989767 |
| (Y_Minimum, Y_Maximum) | 0.969552 |
| (X_Perimeter, Y_Perimeter) | 0.912579 |
| (Outside_X_Index, Log_X_Index) | 0.898341 |
| (LogOfAreas, Log_X_Index) | 0.888239 |
| (LogOfAreas, Log_Y_Index) | 0.879228 |
| (LogOfAreas, SigmoidOfAreas) | 0.872453 |
| (Outside_Global_Index, Orientation_Index) | 0.863987 |
| (Maximum_of_Luminosity, Luminosity_Index) | 0.853856 |
| (Log_Y_Index, SigmoidOfAreas) | 0.852450 |
| (Pixels_Areas, X_Perimeter) | 0.835079 |
| (Pixels_Areas, Y_Perimeter) | 0.834543 |
| (Outside_X_Index, LogOfAreas) | 0.811960 |
| (Y_Perimeter, Sum_of_Luminosity) | 0.809171 |
| (X_Perimeter, Sum_of_Luminosity) | 0.802072 |
| (Edges_Y_Index, Log_X_Index) | -0.851813 |
| (TypeOfSteel_A300, TypeOfSteel_A400) | -0.997732 |

dtype: float64

Вывод:

Координаты сильно коррелируют друг с другом, площадь и периметр коррелируют между собой

Думаю стоит преобразовать сильно коррелирующие столбцы в один, и удалить некоторые другие

```
In [ ]: # X Y максимум и минимум
data_new = data[features]
data_new['X_sum'] = (data_new['X_Minimum'] + data_new['X_Maximum'])/2
data_new['Y_sum'] = (data_new['Y_Minimum'] + data_new['Y_Maximum'])/2

# периметр
data_new['Mean_perimeter'] = (data_new['X_Perimeter'] + data_new['Y_Perimeter'])/2

# удалим некоторые столбцы
data_new = data_new.drop(['LogOfAreas', 'Luminosity_Index', 'Log_Y_Index', 'Log_X_Index', 'Edges_Y_Index', 'Outside_X_Index', 'Outside_Y_Index'])

# удалим использованные
data_new = data_new.drop(['X_Minimum', 'X_Maximum', 'Y_Minimum', 'Y_Maximum', 'X_Perimeter', 'Y_Perimeter'],axis=1)
```

```
In [21]: (data['TypeOfSteel_A300']+data['TypeOfSteel_A400']).value_counts()
```

```
Out[21]:
```

| | count |
|---|-------|
| 1 | 19198 |
| 0 | 20 |
| 2 | 1 |

dtype: int64

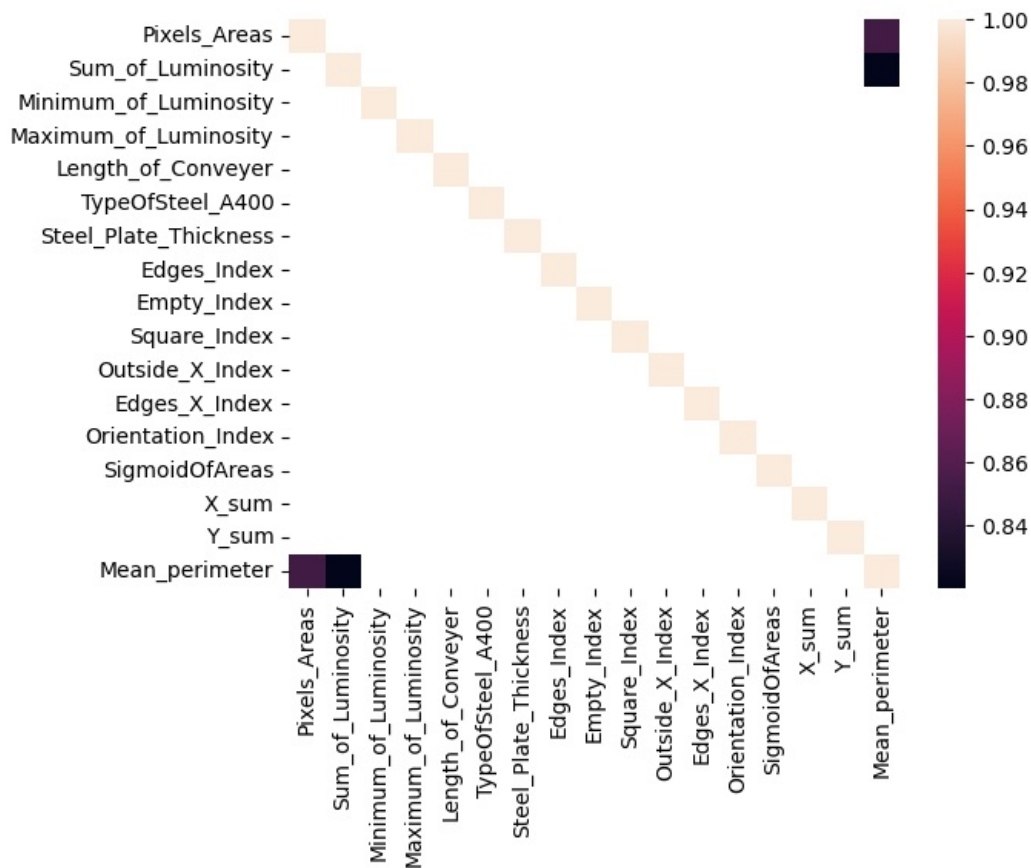
Возможно данные зашумлены или есть какой-то 3 тип стали, думаю стоит удалить строки, где типы стали совпадают

```
In [22]: data_new = data_new.drop(data[data['TypeOfSteel_A300']+data['TypeOfSteel_A400'] != 1].index)
data_new = data_new.drop('TypeOfSteel_A300',axis=1)
```

Еще раз проверим корреляцию признаков в новом датасете

```
In [23]: p = 0.8
sns.heatmap(data_new.corr(),mask=np.abs(data_new.corr())<p)
```

```
Out[23]: <Axes: >
```



Вывод:

Думаю стоит убрать периметр вообще

Пропуски

```
In [24]: data.isnull().sum().sum()
```

```
Out[24]: np.int64(0)
```

Вывод:

Пропусков нет

Таргеты

```
In [25]: data[targets].describe()
```


| | Pastry | Z_Scratch | K_Scratch | Stains | Dirtiness | Bumps | Other_Faults |
|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| count | 19219.000000 | 19219.000000 | 19219.000000 | 19219.000000 | 19219.000000 | 19219.000000 | 19219.000000 |
| mean | 0.076279 | 0.059837 | 0.178573 | 0.029554 | 0.025235 | 0.247828 | 0.341225 |
| std | 0.265450 | 0.237190 | 0.383005 | 0.169358 | 0.156844 | 0.431762 | 0.474133 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 50% | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 75% | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| max | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 |

По сводным статистикам ничего особенного нет

```
In [26]: data[targets].sum(axis=1).value_counts()
```

```
Out[26]:
```

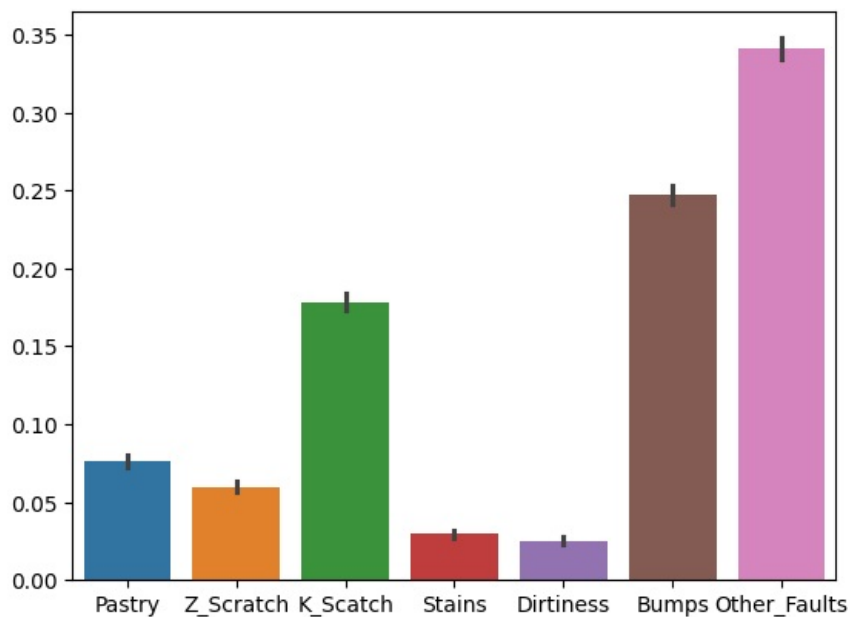
| | count |
|---|-------|
| 1 | 18380 |
| 0 | 818 |
| 2 | 21 |

dtype: int64

Чаще всего в ответе 1 класс

```
In [27]: sns.barplot(data[targets])
```

```
Out[27]: <Axes: >
```



Видно, что классы плохо сбалансированы

```
In [27]:
```

Суммарный итог по EDA

- избавиться от мультиколлинеарности признаков
- пропусков в данных нет
- нормировка признаков
- не забывать про дисбаланс классов

Преобразования на основе EDA

```
In [28]: def prepare_data(data, scalers, filename, train=True):
         data_new = data.copy()
```

```

# X Y максимум и минимум
data_new['X_sum'] = (data_new['X_Minimum'] + data_new['X_Maximum'])/2
data_new['Y_sum'] = (data_new['Y_Minimum'] + data_new['Y_Maximum'])/2

# периметер
#data_new['Mean_perimeter'] = (data_new['X_Perimeter'] + data_new['Y_Perimeter'])/2

# удалим некоторые столбцы
data_new = data_new.drop(['LogOfAreas', 'Luminosity_Index', 'Log_Y_Index', 'Log_X_Index', 'Edges_Y_Index', 'Outs

# удалим использованные
data_new = data_new.drop(['X_Minimum', 'X_Maximum', 'Y_Minimum', 'Y_Maximum', 'X_Perimeter', 'Y_Perimeter'],axis

data_new = data_new.drop(['id'],axis=1)

# только для обучения
if train:
    data_new = data_new.drop(data_new[data_new['TypeOfSteel_A300']+data_new['TypeOfSteel_A400'] != 1].index

data_new = data_new.drop(['TypeOfSteel_A300'],axis=1)

cols = ["Pixels_Areas", "Sum_of_Luminosity"]
for col in cols:
    data_new[col] = np.log1p(data_new[col])

# scale
features = list(set(data_new.columns)-set(targets)-{'TypeOfSteel_A400'})
features.sort()
for scaler in scalers:
    if train:
        data_new[features] = scaler.fit_transform(data_new[features])
    else:
        data_new[features] = scaler.transform(data_new[features])

#
# save
data_new.to_csv(filename,index=False)

```

In [29]: test_data = pd.read_csv('test.csv')

In [30]: from sklearn.preprocessing import MinMaxScaler, StandardScaler
scalers = [MinMaxScaler(feature_range=(-1, 1))]
scalers = [StandardScaler()]
prepare_data(data,scalers,'ready_train.csv')
prepare_data(pd.read_csv('./test.csv'),scalers,'ready_test.csv',train=False)

In [31]: pd.read_csv('ready_test.csv')

Out[31]:

| | Pixels_Areas | Sum_of_Luminosity | Minimum_of_Luminosity | Maximum_of_Luminosity | Length_of_Conveyer | TypeOfStee |
|-------|--------------|-------------------|-----------------------|-----------------------|--------------------|------------|
| 0 | 0.455149 | 0.366509 | -0.620536 | -0.115630 | 1.349841 | |
| 1 | 0.124376 | 0.103994 | 0.248393 | 0.236473 | -0.724051 | |
| 2 | -0.017067 | -0.061148 | 0.561207 | 0.377314 | -0.682848 | |
| 3 | -0.749448 | -0.760497 | 0.769750 | 0.799838 | 1.583325 | |
| 4 | -0.431151 | -0.423867 | 0.630722 | 0.377314 | 1.569591 | |
| ... | ... | ... | ... | ... | ... | ... |
| 12809 | 0.028590 | -0.236041 | -0.203450 | -1.805724 | -0.730918 | |
| 12810 | -0.921775 | -1.040179 | 0.978293 | 0.870258 | -0.669114 | |
| 12811 | 2.306449 | 2.296038 | -2.115094 | 1.011099 | -0.408160 | |
| 12812 | 0.148621 | 0.067892 | -0.655293 | -0.326892 | -0.655379 | |
| 12813 | 1.670664 | 1.636196 | -1.628494 | 0.870258 | -0.710317 | |

12814 rows × 6 columns

In [31]:

```
In [23]: import torch
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from torch.utils.data import DataLoader, Dataset

import torchvision
import torchvision.transforms as transforms

import torch.nn as nn
import torch.nn.functional as F

from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score

from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import ConfusionMatrixDisplay
```

```
In [24]: targets = ['Pastry', 'Z_Scratch', 'K_Scratch', 'Stains', 'Dirtiness', 'Bumps', 'Other_Faults']
data = pd.read_csv('./ready_train.csv')

features = list(set(list(data.columns)) - set(targets))
```

Dataset и DataLoader

```
In [25]: from imblearn.under_sampling import RandomUnderSampler

class SteelPlateDataset(Dataset):
    def __init__(self, X, y=None, balance=False):
        if balance:
            rus = RandomUnderSampler()
            self.X, self.y = rus.fit_resample(X, y)
        else:
            self.X = X
            self.y = y

    def __len__(self):
        return len(self.X)

    def __getitem__(self, index):
        if self.y is not None:
            return self.X.iloc[index].values.astype(np.float32), self.y.iloc[index].astype(np.float32).reshape((1,))
        return self.X.iloc[index].values.astype(np.float32)
```

Построение структуры нейросети

Для каждого класса будем строить свою модель обучения

Функции активации Relu, на последнем сигмоида, которая даст вероятность обнаружение данного класса дефекта.

```
In [26]: class Net(nn.Module):
    def __init__(self, input_size):
        super(Net, self).__init__()

        self.mainSeq = nn.Sequential(
            nn.Linear(input_size, 64),
            nn.ReLU(),

            nn.Linear(64, 32),
            nn.ReLU(),

            nn.Linear(32, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.mainSeq(x)
```

Обучение

Глобальные параметры для обучения

```
In [27]: # параметры
EPOCHS = 50
LEARNING_RATE = 1e-4
#
```

Функция для обучения модели предсказания для 1 класса

```
In [28]: from tqdm import tqdm

def train_one_class(model,optimizer,criterion,train_loader,val_loader=None):

    train_tqdm = tqdm(range(EPOCHS))
    train_errors = []
    val_errors = []
    for epoch in train_tqdm:
        # train step
        model.train()
        train_loss = 0
        for X, y in train_loader:
            optimizer.zero_grad()
            output = model(X)
            loss = criterion(output, y)
            loss.backward()
            optimizer.step()
            train_loss += loss.item()
        train_loss = train_loss / len(train_loader.dataset)
        train_errors.append(train_loss)

        train_tqdm.set_description(f"Loss = {train_loss:.5}")
        # val step
        val_loss = 0
        if val_loader is not None:
            model.eval()
            for X, y in val_loader:
                output = model(X)
                loss = criterion(output, y)
                val_loss += loss.item()
            val_loss = val_loss / len(val_loader.dataset)
            val_errors.append(val_loss)

        train_tqdm.set_description(f"Loss = {train_loss:.5}, Val = {val_loss:.5}")

    return [train_errors,val_errors]
```

Цикл для обучения по всем классам

Используем **BCELoss**, т.к. для каждого класса строим свой классификатор, соответственно бинарная кросс энтропия в качестве функции потерь

В качестве оптимизатора **Адам**

Количество **Эпох** и **Скорость обучения** выбраны экспериментальным путем

```
In [29]: def train_one(y_class):

    #
    X_train, X_val, y_train, y_val = train_test_split(data[features], data[y_class], test_size=0.3, stratify=da

    train_data = SteelPlateDataset(X_train,y_train,balance=True)
    val_data = SteelPlateDataset(X_val,y_val)

    train_loader = torch.utils.data.DataLoader(train_data, batch_size=32)
    val_loader = torch.utils.data.DataLoader(val_data, batch_size=32)
    #
    model = Net(X_train.shape[1])

    criterion = nn.BCELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE, weight_decay=0.01)

    err = train_one_class(model,optimizer,criterion,train_loader,val_loader)
    preds = None
    y_true = None
    model.eval()
    for x,y in val_loader:
        if preds is None:
            preds = model(x).detach().numpy()
            y_true = y
        else:
            preds = np.vstack([preds,model(x).detach().numpy()])
            y_true = np.vstack([y_true,y])

    rcs = roc_auc_score(y_val,preds).item()
    print(rcs)
    return model,err
```

```
In [30]: def train_all():
models = list()
errors = list()
for y_class in targets:
    model,err = train_one(y_class)

    models.append(model)
    errors.append(err)
return models,errors
```

Параллельная версия функции обучения для ускорения

```
In [31]: import multiprocessing as mp
import time

def process_train_one(y_class):
    model,err = train_one(y_class)
    return y_class, model, err

def train_all_parallel():
    start_time = time.time()

    # Создание пула процессов
    with mp.Pool(processes=7) as pool:
        # Список асинхронных задач
        async_results = []

        # Запускаем каждую задачу асинхронно
        for i in range(7):
            async_result = pool.apply_async(process_train_one, args=(targets[i],))
            async_results.append(async_result)

        # Собираем результаты
        results = [async_result.get() for async_result in async_results]

    end_time = time.time()
    total_time = end_time - start_time
    print(f"Время выполнения: {total_time:.2f} секунд")
    models = list()
    errors = list()
    d = dict()

    for i in range(len(targets)):
        y_class,model,err = results[i]
        d[y_class] = model,err

    for i in range(len(targets)):
        models.append(d[targets[i]][0])
        errors.append(d[targets[i]][1])

    #
    return models,errors
```

```
In [32]: models,errors = train_all_parallel()
```

```
Loss = 0.018042, Val = 0.019319: 100%|██████████| 50/50 [03:51<00:00, 4.64s/it]
Loss = 0.018402, Val = 0.019193: 44%|██████████| 22/50 [03:56<04:57, 10.63s/it]
0.863028339132599

Loss = 0.008187, Val = 0.0099902: 100%|██████████| 50/50 [04:01<00:00, 4.83s/it]
Loss = 0.013152: 86%|██████████| 43/50 [04:03<00:36, 5.25s/it]
0.9776491634220772

Loss = 0.012764, Val = 0.013391: 100%|██████████| 50/50 [04:31<00:00, 5.44s/it]
Loss = 0.018322: 52%|██████████| 26/50 [04:33<03:26, 8.60s/it]
0.9030861669811446

Loss = 0.015856, Val = 0.016568: 100%|██████████| 50/50 [04:43<00:00, 5.67s/it]
Loss = 0.020117, Val = 0.020443: 46%|██████████| 23/50 [04:44<04:15, 9.45s/it]
0.8401854066985647

Loss = 0.0054678, Val = 0.0064049: 100%|██████████| 50/50 [05:36<00:00, 6.72s/it]
Loss = 0.018173, Val = 0.019099: 82%|██████████| 41/50 [05:36<00:35, 3.99s/it]
0.9753697169495456

Loss = 0.018101, Val = 0.019072: 100%|██████████| 50/50 [06:02<00:00, 7.25s/it]
Loss = 0.020007: 80%|██████████| 40/50 [06:02<00:35, 3.59s/it]
0.747454313694347

Loss = 0.020028, Val = 0.020347: 100%|██████████| 50/50 [06:21<00:00, 7.64s/it]
0.682805842941624
Время выполнения: 383.24 секунд
```

Сохраним модели

```
In [33]: def save_model(model,file_path):
torch.save(model.state_dict(), file_path)
```

```
In [34]: for i in range(len(targets)):
save_model(models[i],f'./{targets[i]}.model')
```

Метрики качества

Отрисовывает:

1. Кривые ошибки на обучении и валидации
2. AUC ROC

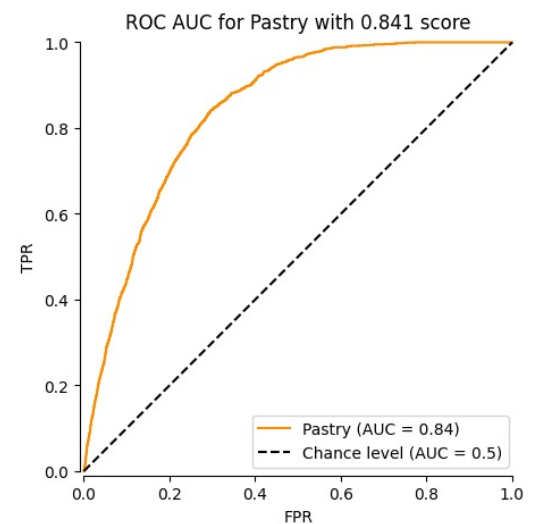
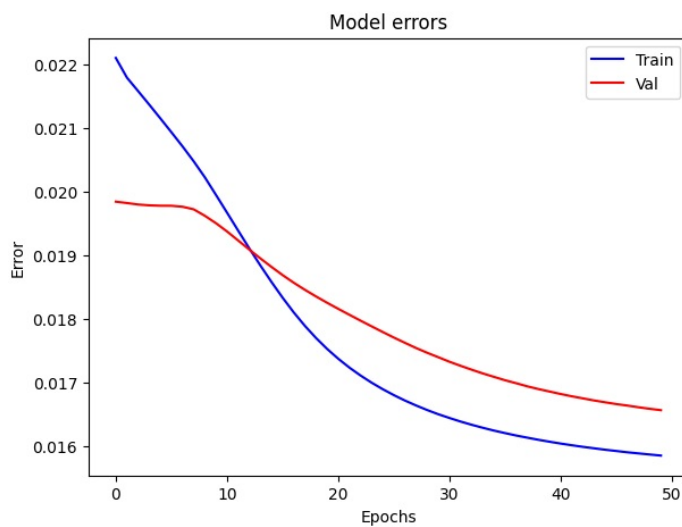
```
In [35]: def plot_metrics(model,y_class,train_error,val_error):
    X,y = data[features],data[y_class]
    x_loader = torch.utils.data.DataLoader(SteelPlateDataset(X,y), batch_size=len(X))
    with torch.no_grad():
        for d,_ in x_loader:
            pred = model(d).detach().numpy()

    fig,ax = plt.subplots(1,2,figsize=(15,5))

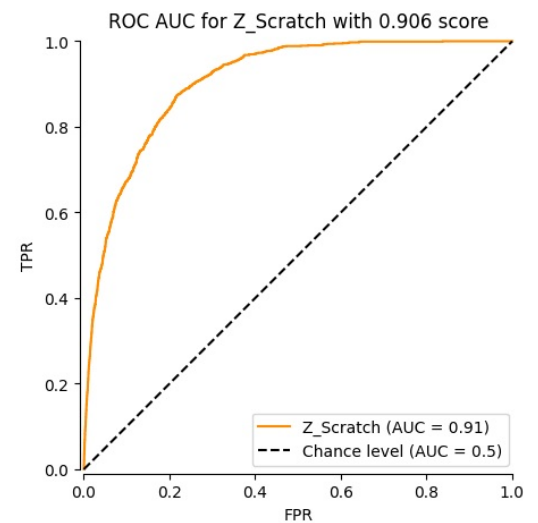
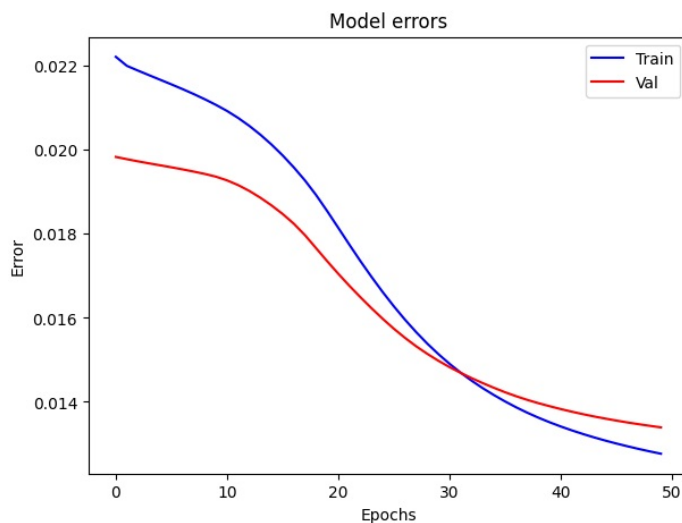
    # lineplot
    sns.lineplot(train_error,c='b',ax=ax[0],label='Train')
    sns.lineplot(val_error,c='r',ax=ax[0],label='Val')
    ax[0].set(xlabel='Epochs',ylabel='Error', title='Model errors')
    # roc auc
    rcs = roc_auc_score(y,pred)

    display = RocCurveDisplay.from_predictions(y,pred,name=y_class,color="darkorange",plot_chance_level=True,de
_ = display.ax_.set(xlabel="FPR", ylabel="TPR", title=f"ROC AUC for {y_class} with {rcs:.4} score")
```

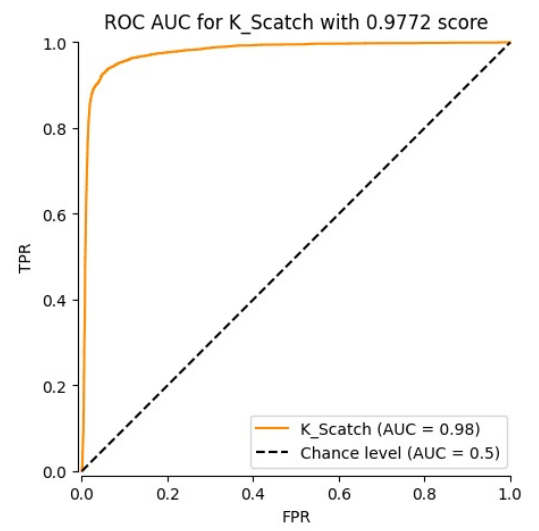
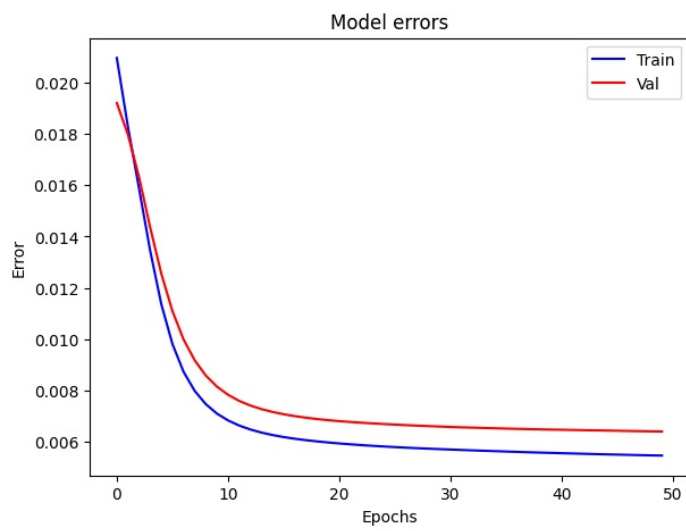
```
In [36]: i = 0
plot_metrics(models[i],targets[i],errors[i][0],errors[i][1])
```



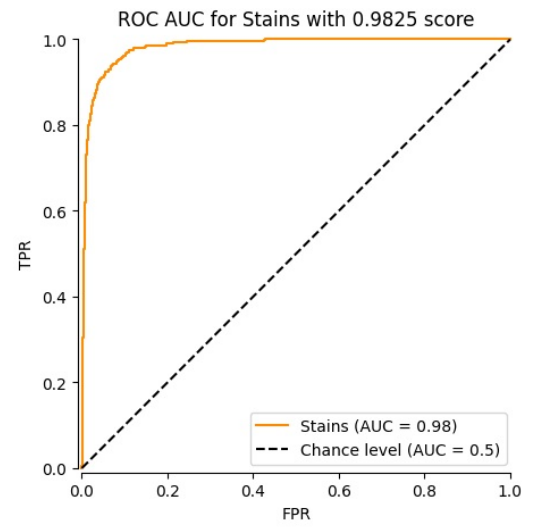
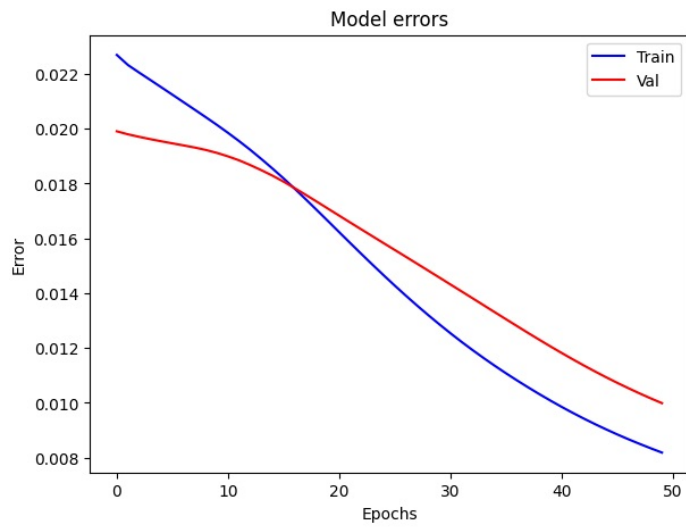
```
In [37]: i = 1
plot_metrics(models[i],targets[i],errors[i][0],errors[i][1])
```



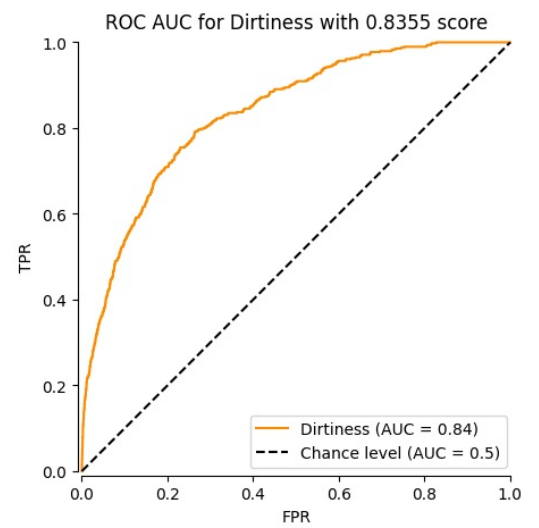
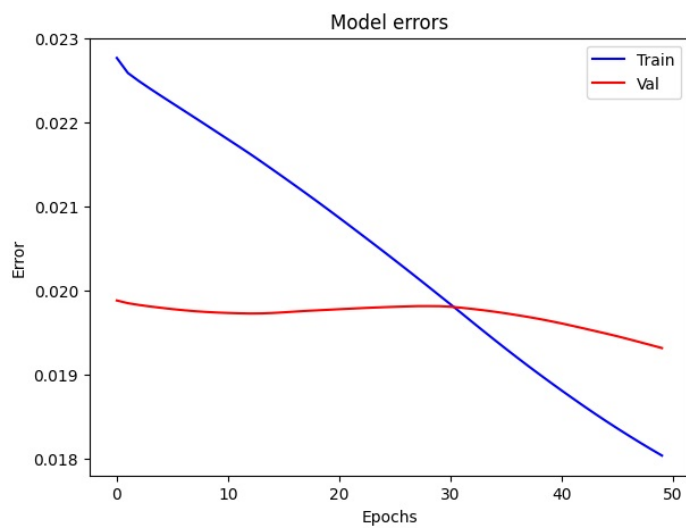
```
In [38]: i = 2
plot_metrics(models[i],targets[i],errors[i][0],errors[i][1])
```



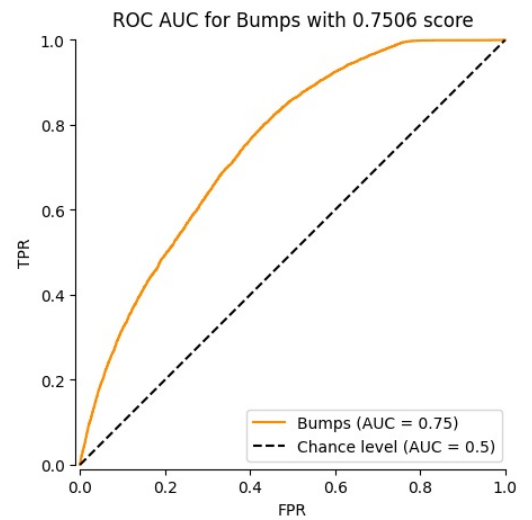
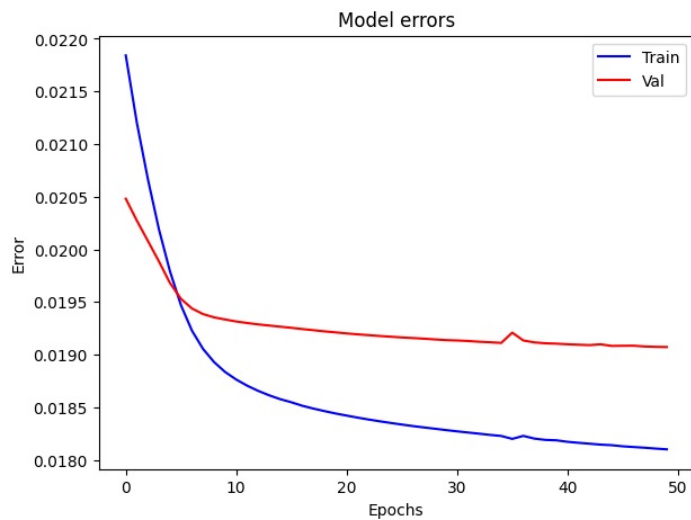
```
In [39]: i = 3
plot_metrics(models[i],targets[i],errors[i][0],errors[i][1])
```



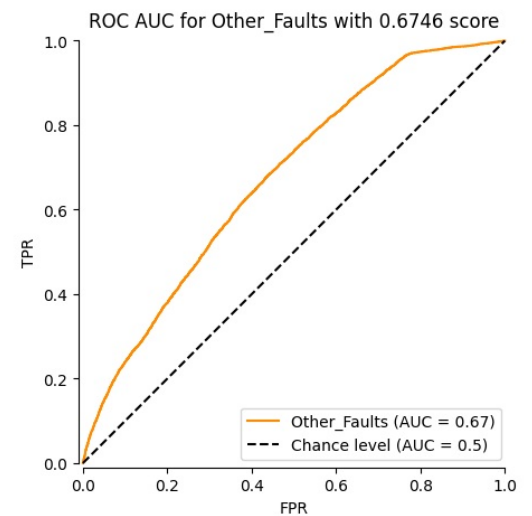
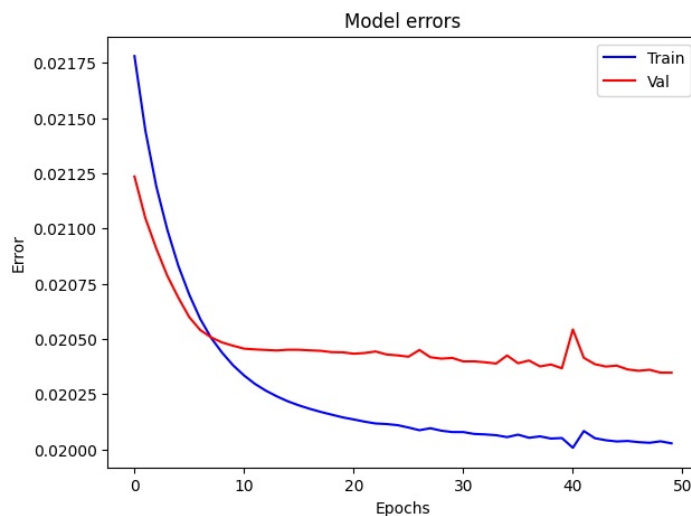
```
In [40]: i = 4
plot_metrics(models[i],targets[i],errors[i][0],errors[i][1])
```



```
In [41]: i = 5
plot_metrics(models[i],targets[i],errors[i][0],errors[i][1])
```



```
In [42]: i = 6
plot_metrics(models[i],targets[i],errors[i][0],errors[i][1])
```



Выводы:

Первые 4 класса хорошо предсказываются построенной моделью.


С классами 'Bump','Other_Faults' и 'Dirtiness' стоит еще поработать, но пока оставлю так.

Предсказания для тестовой выборки

```
In [43]: test_df = pd.read_csv('./ready_test.csv')
test_data = SteelPlateDataset(test_df[features])
test_loader = torch.utils.data.DataLoader(test_data, batch_size=len(test_df))

pred_df = test_df.copy()
pred_df['id'] = pd.read_csv('./test.csv')['id']
for i in range(len(targets)):
    with torch.no_grad():
        for d in test_loader:
            pred_df.loc[:,targets[i]] = models[i](d).numpy()
```

```
In [44]: pred_df[['id']+targets].to_csv('./submission.csv',index=False)
```

| Submission and Description | | Private Score 📄 | Public Score 📄 | Selected |
|--|--|------------------------------|-----------------------------|--------------------------|
|  submission.csv Complete (after deadline) · 11s ago | | | | |
| | | 0.84395 | 0.85226 | <input type="checkbox"/> |