

Библиотеки Python для Data Science: продолжение

Курсовой проект

Соковнин Игорь Леонидович

Постановка задачи

В [1]: [#3.3. Box plot, или ящик с усами](https://ru.coursera.org/lecture/vvedeniye-dannye/3-4-diagramma-rassieianiia-DW6HN)
[#https://ru.coursera.org/lecture/vvedeniye-dannye/3-4-diagramma-rassieianiia-DW6HN](https://ru.coursera.org/lecture/vvedeniye-dannye/3-4-diagramma-rassieianiia-DW6HN)

Задача

Требуется, на основании имеющихся данных о клиентах банка, построить модель, используя обучающий датасет, для прогнозирования невыполнения долговых обязательств по текущему кредиту. Выполнить прогноз для примеров из тестового датасета.

Наименование файлов с данными

course_project_train.csv - обучающий датасет

course_project_test.csv - тестовый датасет

Целевая переменная

Credit Default - факт невыполнения кредитных обязательств

Метрика качества

F1-score (sklearn.metrics.f1_score)

Требования к решению

Целевая метрика

- $F1 > 0.5$
- Метрика оценивается по качеству прогноза для главного класса (1 - просрочка по кредиту)

Решение должно содержать

1. Тетрадка Jupyter Notebook с кодом Вашего решения, названная по образцу {ФИО}_solution.ipynb, пример SShirkin_solution.ipynb
2. Файл CSV с прогнозами целевой переменной для тестового датасета, названный по образцу {ФИО}_predictions.csv, пример SShirkin_predictions.csv

Рекомендации для файла с кодом (ipynb)

1. Файл должен содержать заголовки и комментарии (markdown)
2. Повторяющиеся операции лучше оформлять в виде функций
3. Не делать вывод большого количества строк таблиц (5-10 достаточно)
4. По возможности добавлять графики, описывающие данные (около 3-5)
5. Добавлять только лучшую модель, то есть не включать в код все варианты решения проекта
6. Скрипт проекта должен отрабатывать от начала и до конца (от загрузки данных до выгрузки предсказаний)
7. Весь проект должен быть в одном скрипте (файл ipynb).
8. Допускается применение библиотек Python и моделей машинного обучения, которые были в данном курсе.

Сроки сдачи

Сдать проект нужно в течение 5 дней после окончания последнего вебинара. Оценки работ, сданных до дедлайна, будут представлены в виде рейтинга, ранжированного по заданной метрике качества. Проекты, сданные после дедлайна или сданные повторно, не попадают в рейтинг, но можно будет узнать результат.

Этапы выполнения курсового проекта

Построение модели классификации

1. [Описание данных](#)
2. [Загрузка данных](#)
3. [Обзор обучающего датасета](#) +
4. [Обработка выбросов](#) +
5. [Обработка пропусков](#) +
6. [Анализ данных](#) +
7. [Отбор признаков](#)
8. [Балансировка классов](#)
9. [Подбор моделей, получение бейзлана](#)
10. Выбор наилучшей модели, настройка гиперпараметров
11. Проверка качества, борьба с переобучением
12. Интерпретация результатов

Прогнозирование на тестовом датасете

1. Выполнить для тестового датасета те же этапы обработки и построения признаков
2. Спрогнозировать целевую переменную, используя модель, построенную на обучающем датасете
3. Прогнозы должны быть для всех примеров из тестового датасета (для всех строк)
4. Соблюдать исходный порядок примеров из тестового датасета

Подключение библиотек и скриптов

```
In [2]: import numpy as np
import pandas as pd
import matplotlib
#import matplotlib.image as img
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

import warnings
warnings.filterwarnings('ignore')

matplotlib.rcParams.update({'font.size': 14})
```

Построение модели классификации

Описание датасета

- 1. **Home Ownership** - домовладение
- 2. **Annual Income** - годовой доход
- 3. **Years in current job** - количество лет на текущем месте работы
- 4. **Tax Liens** - налоговые обременения
- 5. **Number of Open Accounts** - количество открытых счетов
- 6. **Years of Credit History** - количество лет кредитной истории

- **7. Maximum Open Credit** - наибольший открытый кредит
- **8. Number of Credit Problems** - количество проблем с кредитом
- **9. Months since last delinquent** - количество месяцев с последней просрочки платежа
- **10. Bankruptcies** - банкротства
- **11. Purpose** - цель кредита
- **12. Term** - срок кредита
- **13. Current Loan Amount** - текущая сумма кредита
- **14. Current Credit Balance** - текущий кредитный баланс
- **15. Monthly Debt** - ежемесячный долг
- **16. Credit Score** - Кредитный рейтинг?
- **17. Credit Default** - факт невыполнения кредитных обязательств (0 - погашен вовремя, 1 - просрочка)

Пути к директориям и файлам

```
In [3]: # input
TRAIN_DATASET_PATH = './course_project/course_project_train.csv'
TEST_DATASET_PATH = './course_project/course_project_test.csv'

# output
PREP_DATASET_PATH = './training_project/training_project_data_prep.csv'
```

Загрузка данных

```
In [4]: df_train = pd.read_csv(TRAIN_DATASET_PATH)
df_train.head()
```

Out[4]:

	Home Ownership	Annual Income	Years in current job	Tax Liens	Number of Open Accounts	Years of Credit History	Maximum Open Credit	Number of Credit Problems	Months since last delinquent	Bankruptcies
0	Own Home	482087.0	NaN	0.0	11.0	26.3	685960.0	1.0	NaN	1.0
1	Own Home	1025487.0	10+ years	0.0	15.0	15.3	1181730.0	0.0	NaN	0.0
2	Home Mortgage	751412.0	8 years	0.0	11.0	35.0	1182434.0	0.0	NaN	0.0
3	Own Home	805068.0	6 years	0.0	8.0	22.5	147400.0	1.0	NaN	1.0
4	Rent	776264.0	8 years	0.0	13.0	13.6	385836.0	1.0	NaN	0.0

```
B [5]: df_test = pd.read_csv(TEST_DATASET_PATH)
df_test.head()
```

Out[5]:

	Home Ownership	Annual Income	Years in current job	Tax Liens	Number of Open Accounts	Years of Credit History	Maximum Open Credit	Number of Credit Problems	Months since last delinquent	Bankruptcies
0	Rent	NaN	4 years	0.0	9.0	12.5	220968.0	0.0	70.0	0.0
1	Rent	231838.0	1 year	0.0	6.0	32.7	55946.0	0.0	8.0	0.0
2	Home Mortgage	1152540.0	3 years	0.0	10.0	13.7	204600.0	0.0	NaN	0.0
3	Home Mortgage	1220313.0	10+ years	0.0	16.0	17.0	456302.0	0.0	70.0	0.0
4	Home Mortgage	2340952.0	6 years	0.0	11.0	23.6	1207272.0	0.0	NaN	0.0

```
B [6]: df_train.shape # Получим описание pandas DataFrame (количество строк и столбцов)
```

Out[6]: (7500, 17)

```
B [7]: print('Строк в train:', df_train.shape[0]) # gives number of row count
print('Столбцов в train:', df_train.shape[1]) # gives number of col count

print('\nСтрок test:', df_test.shape[0])
print('Столбцов в test:', df_test.shape[1])
```

Строк в train: 7500
Столбцов в train: 17

Строк test: 2500
Столбцов в test: 16

```
B [8]: df_train.iloc[0] # Получаем первую строку (index=0)
```

```
Out[8]: Home Ownership          Own Home
Annual Income                482087
Years in current job          NaN
Tax Liens                     0
Number of Open Accounts       11
Years of Credit History       26.3
Maximum Open Credit           685960
Number of Credit Problems     1
Months since last delinquent   NaN
Bankruptcies                  1
Purpose                       debt consolidation
Term                          Short Term
Current Loan Amount           1e+08
Current Credit Balance        47386
Monthly Debt                  7914
Credit Score                  749
Credit Default                0
Name: 0, dtype: object
```

B [9]: `df_train.info()` *# Рассмотрим типы признаков*

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7500 entries, 0 to 7499
Data columns (total 17 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Home Ownership                        7500 non-null   object
1   Annual Income                        5943 non-null   float64
2   Years in current job                 7129 non-null   object
3   Tax Liens                            7500 non-null   float64
4   Number of Open Accounts             7500 non-null   float64
5   Years of Credit History             7500 non-null   float64
6   Maximum Open Credit                 7500 non-null   float64
7   Number of Credit Problems           7500 non-null   float64
8   Months since last delinquent        3419 non-null   float64
9   Bankruptcies                        7486 non-null   float64
10  Purpose                             7500 non-null   object
11  Term                               7500 non-null   object
12  Current Loan Amount                 7500 non-null   float64
13  Current Credit Balance              7500 non-null   float64
14  Monthly Debt                       7500 non-null   float64
15  Credit Score                       5943 non-null   float64
16  Credit Default                     7500 non-null   int64
dtypes: float64(12), int64(1), object(4)
memory usage: 996.2+ KB
```

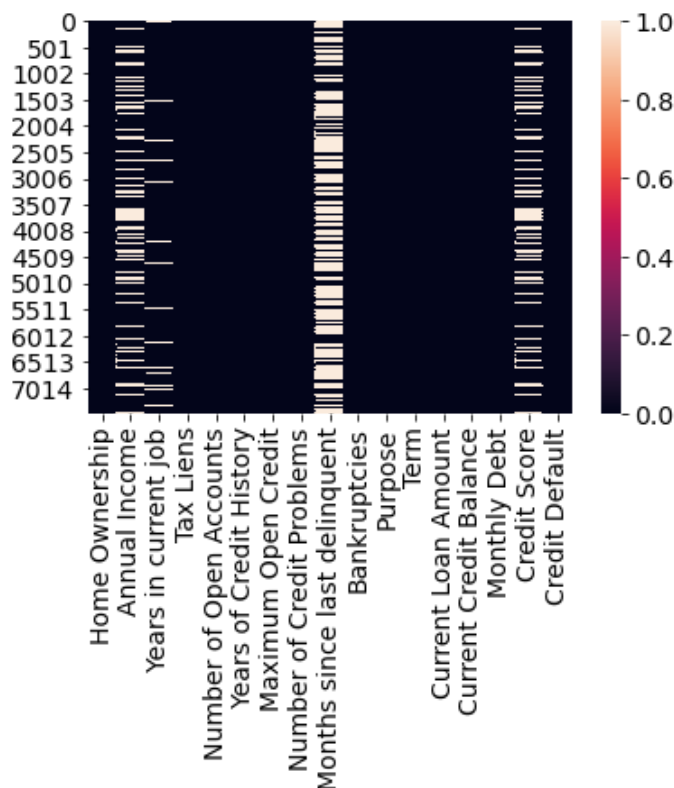
B [10]: `#df_train.dtypes`

representing null/NaN values using seaborn plotting techniques

representing using heatmap()

```
B [11]: sns.heatmap(df_train.isnull())
```

```
Out[11]: <AxesSubplot:>
```



1. Обзор данных (Обзор обучающего датасета)

Обзор целевой переменной

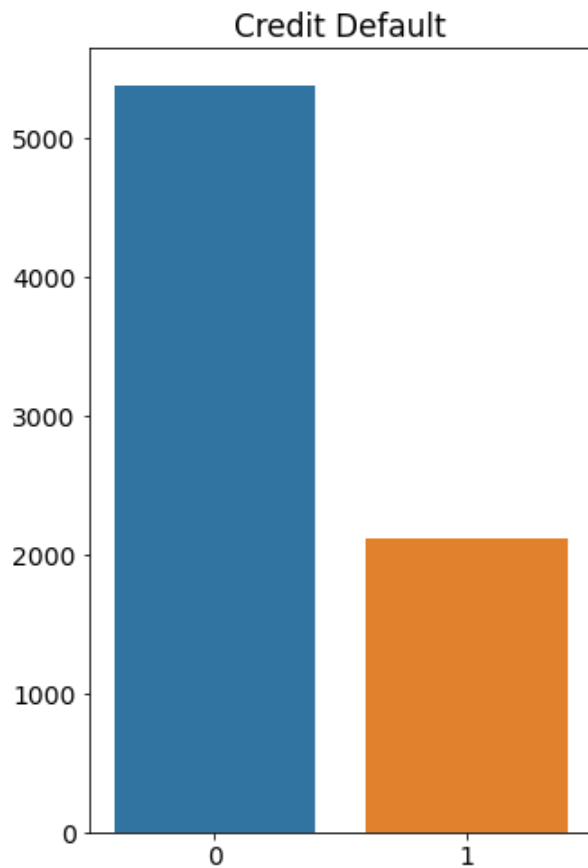
```
B [12]: df_train['Credit Default'].value_counts() # Количество различных значений признака 'Cre
```

```
Out[12]: 0    5387
         1    2113
         Name: Credit Default, dtype: int64
```

```
B [13]: counts = df_train['Credit Default'].value_counts()

plt.figure(figsize=(5,8))
plt.title('Credit Default')
sns.barplot(counts.index, counts.values)

plt.show()
```



Приведение типов

```
B [14]: for colname in ['Tax Liens', 'Number of Credit Problems', 'Bankruptcies']:
        df_train[colname] = df_train[colname].astype(str)
```

```
B [15]: df_train.dtypes
```

```
Out[15]: Home Ownership      object
Annual Income      float64
Years in current job  object
Tax Liens          object
Number of Open Accounts  float64
Years of Credit History  float64
Maximum Open Credit  float64
Number of Credit Problems  object
Months since last delinquent  float64
Bankruptcies       object
Purpose            object
Term              object
Current Loan Amount  float64
Current Credit Balance  float64
Monthly Debt        float64
Credit Score        float64
Credit Default      int64
dtype: object
```

Обзор количественных признаков

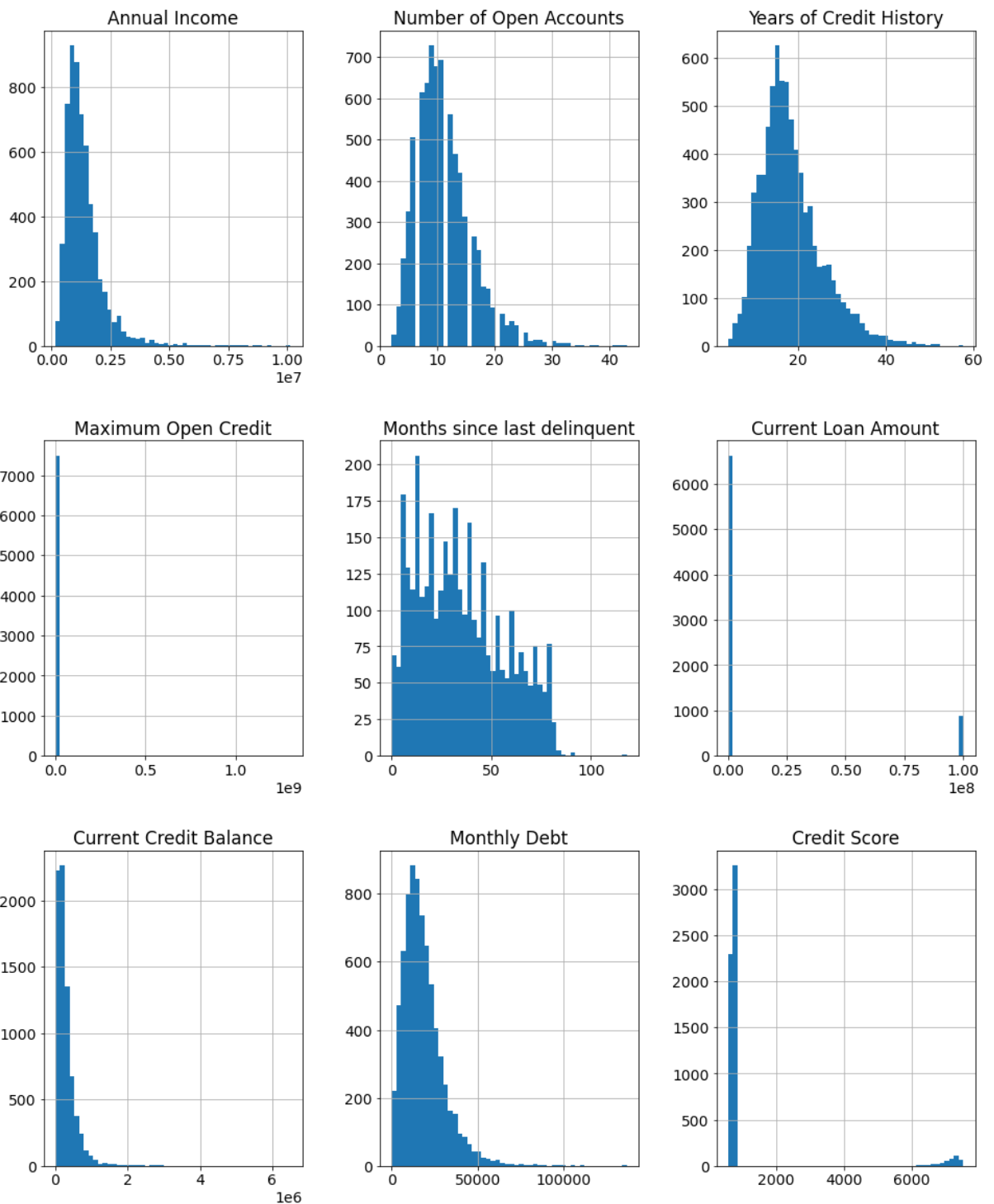
```
B [16]: df_train.describe().T # Анализ количественные признаки
```

```
Out[16]:
```

	count	mean	std	min	25%	50%	75%	max
Annual Income	5943.0	1.366392e+06	8.453392e+05	164597.0	844341.0	1168386.0	1640137.00	1.014934e+07
Number of Open Accounts	7500.0	1.113093e+01	4.908924e+00	2.0	8.0	10.0	14.00	4.300000e+01
Years of Credit History	7500.0	1.831747e+01	7.041946e+00	4.0	13.5	17.0	21.80	5.770000e+01
Maximum Open Credit	7500.0	9.451537e+05	1.602622e+07	0.0	279229.5	478159.0	793501.50	1.304726e+09
Months since last delinquent	3419.0	3.469260e+01	2.168881e+01	0.0	16.0	32.0	50.00	1.180000e+02
Current Loan Amount	7500.0	1.187318e+07	3.192612e+07	11242.0	180169.0	309573.0	519882.00	1.000000e+08
Current Credit Balance	7500.0	2.898332e+05	3.178714e+05	0.0	114256.5	209323.0	360406.25	6.506797e+06
Monthly Debt	7500.0	1.831445e+04	1.192676e+04	0.0	10067.5	16076.5	23818.00	1.366790e+05
Credit Score	5943.0	1.151087e+03	1.604451e+03	585.0	711.0	731.0	743.00	7.510000e+03
Credit Default	7500.0	2.817333e-01	4.498740e-01	0.0	0.0	0.0	1.00	1.000000e+00


```
B [17]: df_num_features = df_train.select_dtypes(include=['float32', 'float64', 'int8', 'int16',
df_num_features.hist(figsize=(16, 20), bins=50, grid=True)
```

```
Out[17]: array([[<AxesSubplot:title={'center':'Annual Income'}>,
<AxesSubplot:title={'center':'Number of Open Accounts'}>,
<AxesSubplot:title={'center':'Years of Credit History'}>],
[<AxesSubplot:title={'center':'Maximum Open Credit'}>,
<AxesSubplot:title={'center':'Months since last delinquent'}>,
<AxesSubplot:title={'center':'Current Loan Amount'}>],
[<AxesSubplot:title={'center':'Current Credit Balance'}>,
<AxesSubplot:title={'center':'Monthly Debt'}>,
<AxesSubplot:title={'center':'Credit Score'}>]], dtype=object)
```



Наблюдаются выбросы по следующим признакам: Current Loan Amount, Maximum Open Credit, Current Credit Balance.

Ряд признаков имеют аномально высокое значение, но вполне вероятное: . Их необходимо будет ограничить.

```

B [18]: def plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type
        '''Производим анализ фичи'''

        print(f'feature_name = {feature_name}' +
              f'\nfeature_value_max = {feature_value_max}' +
              f'\nfeature_value_min = {feature_value_min}')

        # прямая сортировка
        print('_'*50+'\n\nКоличество\n'+_'*50)
        if count_sort == 0:
            print(df_train[feature_name].value_counts().sort_values()) # по значению
        else:
            print(df_train[feature_name].value_counts().sort_index()) # по индексу

        # обратная сортировка
        # print('_'*50+'\n\nКоличество\n'+_'*50)
        # nt(df_train[feature_name].value_counts().sort_index(ascending=False).sort_values(a
        # print(df_train[feature_name].sort_values().value_counts())

        print('_'*50+'\n\nОтсортированные записи\n'+_'*50)
        print(df_train[feature_name].sort_values())

        if data_type != 2:
            print('_' * 50 + '\n\nПервичный датасет\n' +
                  f'\nМода датасета: {df_train[feature_name].mode()[0]}' +
                  f'\nМедиана датасета: {df_train[feature_name].median()}' +
                  f'\nСреднее значение датасета: {df_train[feature_name].mean()}' +
                  f'\nМаксимальное значение датасета: {df_train[feature_name].max()}' +
                  f'\nМинимальное значение датасета: {df_train[feature_name].min()}' + '\n' +
                  '_' * 50)

        if data_type == 0:
            # 1-й график
            fig, ax = plt.subplots(nrows=1, ncols=1)
            plt.xlabel(feature_name)
            plt.ylabel('Count')
            plt.title('\nПервичный датасет\n')
            #plt.title(r'$\mathrm{Histogram}$ of $IQ$:\ $ \mu=100, \sigma=15$')
            #plt.axis([0, 100000, 0, 900])
            plt.grid(True)
            df_train[feature_name].hist(bins=50)

            print('\nКоличество записей в датасете:', df_train.shape[0])
            df = df_train.loc[(df_train[feature_name] < feature_value_min)]
            print('Количество записей в датасете < {0}: {1}'.format(feature_value_min, df.sh
            df = df_train.loc[(df_train[feature_name] > feature_value_max)]
            print('Количество записей в датасете > {0}: {1}'.format(feature_value_max, df.sh
            print('_' * 50)

            df = df_train.loc[(df_train[feature_name] <= feature_value_max) & (df_train[feat

            # 2-й график
            fig, ax = plt.subplots(nrows=1, ncols=1)
            plt.xlabel(feature_name)
            plt.ylabel('Count')
            plt.title('\nОбработанный датасет')
            plt.grid(True)
            df[feature_name].hist(bins=50)

            print('\nОбработанный датасет\n' +
                  f'\nМода датасета: {df[feature_name].mode()[0]}' +
                  f'\nМедиана датасета: {df[feature_name].median()}' +
                  f'\nСреднее значение датасета: {df[feature_name].mean()}' +
                  f'\nМаксимальное значение датасета: {df[feature_name].max()}' +
                  f'\nМинимальное значение датасета: {df[feature_name].min()}' + '\n' +
                  '_' * 50)

```

```
sns.set_theme(style="ticks")
#sns.set(context='notebook', font_scale=1, color_codes=False)

# 3-й график
fig, ax = plt.subplots(nrows=1, ncols=1)
sns.boxplot(df[feature_name]);

ax.xaxis.grid(True)
ax.set(ylabel='')
sns.despine(trim=True, left=False)

# 4-й график
fig, ax = plt.subplots(nrows=1, ncols=1)
sns.violinplot(df[feature_name], palette='rainbow');

ax.xaxis.grid(True)
ax.set(ylabel='')
sns.despine(trim=True, left=False)
```

Рассмотрим признаки подробнее

1. Home Ownership - домовладение (категориальные данные)

```
B [19]: feature_name = 'Home Ownership'
feature_value_max = 10
feature_value_min = 0
data_type = 2
# plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type)
```

2. Annual Income - годовой доход

```

В [20]: feature_name = 'Annual Income'
feature_value_max = 4000000
feature_value_min = 164597
data_type = 0
plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type)

# Считаем выбросами Annual Income > 4 000 000 (91 значения) и Annual Income < 164597
# Считаем выбросами Annual Income > 5 000 000 (44 значения)

```

```

feature_name = Annual Income
feature_value_max = 4000000
feature_value_min = 164597

```

Количество

2083825.0	1
785954.0	1
266000.0	1
1177411.0	1
1539152.0	1
..	
969475.0	4
1043651.0	4
1338113.0	4
1058376.0	4
1161660.0	4

Name: Annual Income, Length: 5478, dtype: int64

Отсортированные записи

4240	164597.0
4485	175845.0
3946	177251.0
3310	191577.0
1114	192223.0
...	
7482	NaN
7492	NaN
7494	NaN
7498	NaN
7499	NaN

Name: Annual Income, Length: 7500, dtype: float64

Первичный датасет

Мода датасета: 969475.0
 Медиана датасета: 1168386.0
 Среднее значение датасета: 1366391.7201749957
 Максимальное значение датасета: 10149344.0
 Минимальное значение датасета: 164597.0

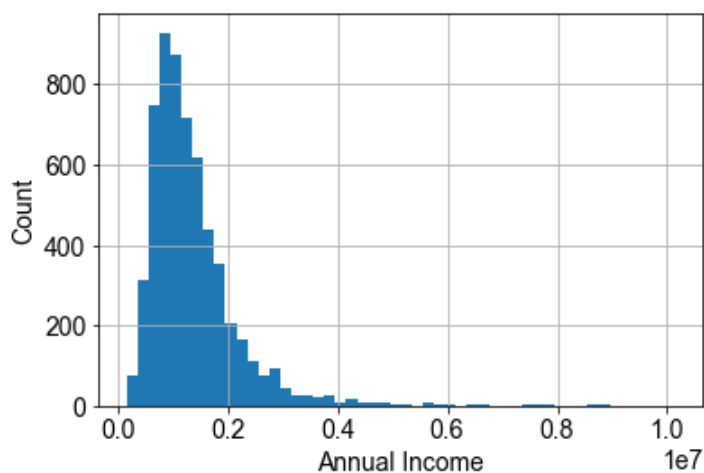
Количество записей в датасете: 7500
 Количество записей в датасете < 164597: 0
 Количество записей в датасете > 4000000: 91

Обработанный датасет

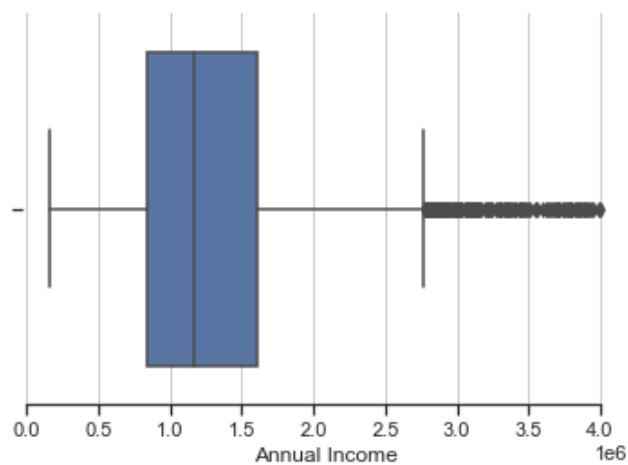
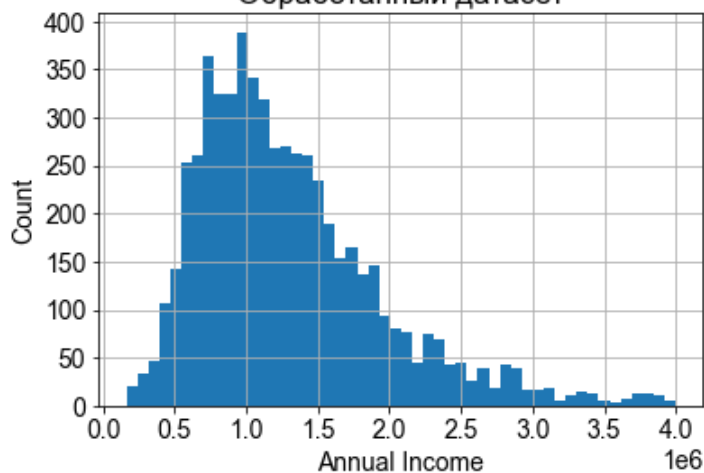
Мода датасета: 969475.0
 Медиана датасета: 1161432.0
 Среднее значение датасета: 1301564.1461038962
 Максимальное значение датасета: 3997334.0

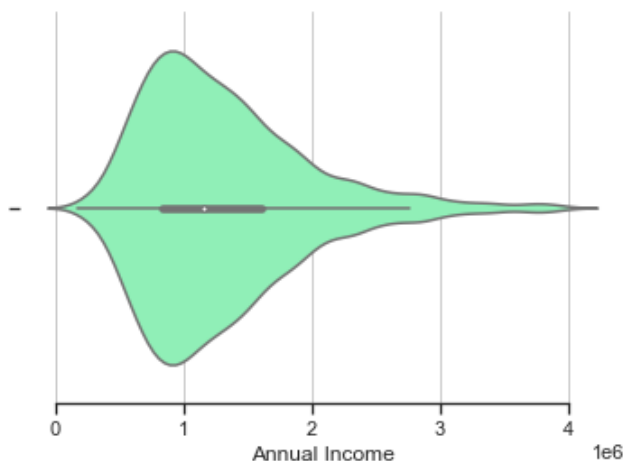
Минимальное значение датасета: 164597.0

Первичный датасет



Обработанный датасет





Считаем выбросами **Annual Income** > 5 000 000 (44 значения)

3. Years in current job - количество лет на текущем месте работы (категориальные данные)

```
B [21]: feature_name = 'Years in current job'
feature_value_max = 5000000
feature_value_min = 0
data_type = 2
# plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type)
```

4. Tax Liens - налоговые обременения (категориальные данные)

```
B [22]: feature_name = 'Tax Liens'
feature_value_max = 4000000
feature_value_min = 0
data_type = 1
# plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type)
```

5. Number of Open Accounts - количество открытых счетов

```

В [23]: feature_name = 'Number of Open Accounts'
feature_value_max = 33
feature_value_min = 0
data_type = 0
plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type)

# Считаем выбросами Number of Open Accounts > 33 (9 значений)

```

```

feature_name = Number of Open Accounts
feature_value_max = 33
feature_value_min = 0

```

Количество

42.0	1
43.0	1
38.0	1
41.0	1
35.0	1
37.0	2
34.0	2
31.0	6
33.0	6
32.0	6
29.0	10
30.0	11
26.0	12
27.0	14
28.0	14
2.0	28
25.0	32
22.0	49
24.0	50
23.0	59
21.0	78
20.0	93
3.0	95
19.0	139
18.0	143
4.0	212
17.0	232
16.0	265
15.0	313
5.0	325
14.0	420
13.0	465
6.0	504
12.0	562
7.0	613
8.0	638
10.0	677
11.0	692
9.0	728

Name: Number of Open Accounts, dtype: int64

Отсортированные записи

3271	2.0
3768	2.0
2321	2.0
2325	2.0
1743	2.0
...	
6868	37.0
3475	38.0


```
2840    41.0
5738    42.0
1769    43.0
```

Name: Number of Open Accounts, Length: 7500, dtype: float64

Первичный датасет

Мода датасета: 9.0

Медиана датасета: 10.0

Среднее значение датасета: 11.130933333333333

Максимальное значение датасета: 43.0

Минимальное значение датасета: 2.0

Количество записей в датасете: 7500

Количество записей в датасете < 0: 0

Количество записей в датасете > 33: 9

Обработанный датасет

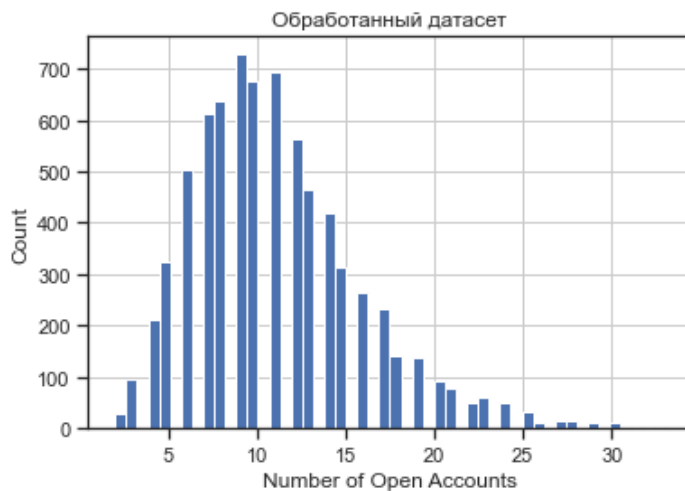
Мода датасета: 9.0

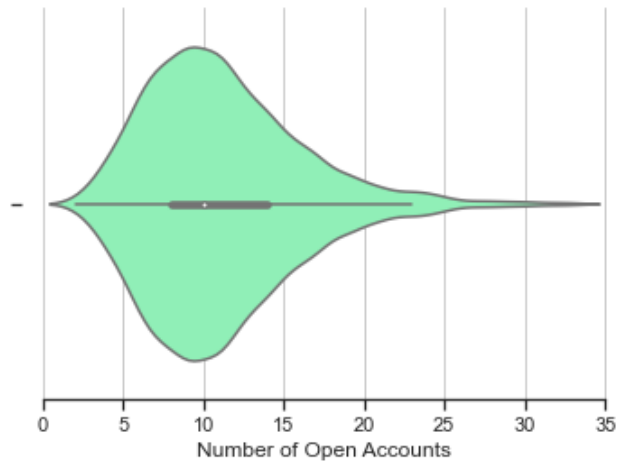
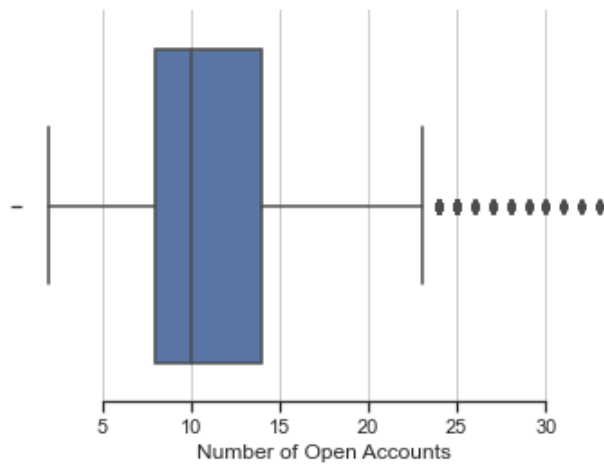
Медиана датасета: 10.0

Среднее значение датасета: 11.098785208917368

Максимальное значение датасета: 33.0

Минимальное значение датасета: 2.0





6. Years of Credit History - количество лет кредитной истории

```

В [24]: feature_name = 'Years of Credit History'
feature_value_max = 40
feature_value_min = 0
data_type = 0
plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type)

# Считаем выбросами Years of Credit History > 40 (83 значения)
# Считаем выбросами Years of Credit History > 50 (8 значения)

```

```

feature_name = Years of Credit History
feature_value_max = 40
feature_value_min = 0

```

Количество

39.8	1
41.8	1
46.3	1
6.2	1
36.3	1

...

17.5	83
17.0	86
16.5	91
16.0	99
15.0	104

Name: Years of Credit History, Length: 408, dtype: int64

Отсортированные записи

324	4.0
5497	4.3
3784	4.5
2560	4.5
6633	4.7

...

3628	51.3
4716	51.5
4301	51.9
247	52.2
476	57.7

Name: Years of Credit History, Length: 7500, dtype: float64

Первичный датасет

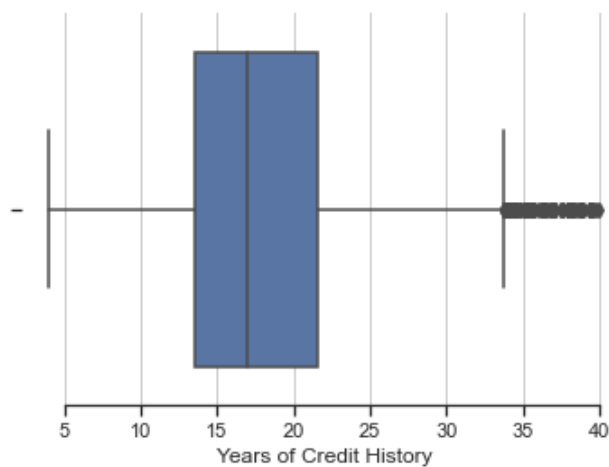
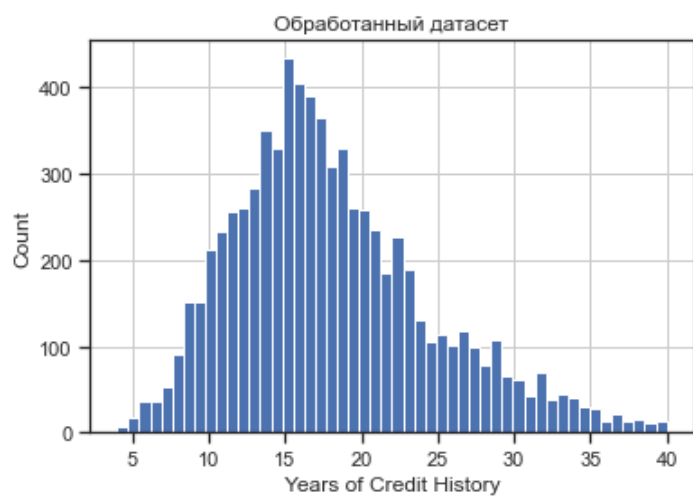
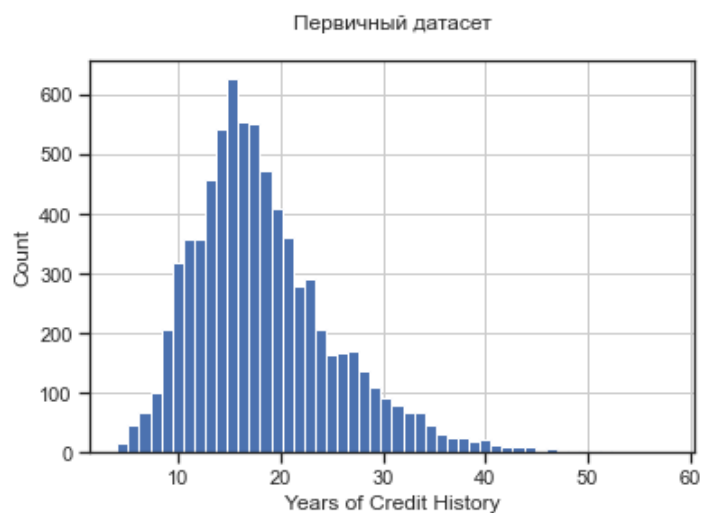
Мода датасета: 15.0
 Медиана датасета: 17.0
 Среднее значение датасета: 18.3174666666666647
 Максимальное значение датасета: 57.7
 Минимальное значение датасета: 4.0

Количество записей в датасете: 7500
 Количество записей в датасете < 0: 0
 Количество записей в датасете > 40: 83

Обработанный датасет

Мода датасета: 15.0
 Медиана датасета: 17.0
 Среднее значение датасета: 18.027747067547494
 Максимальное значение датасета: 40.0

Минимальное значение датасета: 4.0





7. Maximum Open Credit - наибольший открытый кредит

```

B [25]: feature_name = 'Maximum Open Credit'
feature_value_max = 2000000
feature_value_min = 0 # 50000
data_type = 0
plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type, 0)

# Считаем выбросами значения 'Maximum Open Credit' > 4 000 000 (64 значений) 'Maximum Op
# Считаем выбросами значения 'Maximum Open Credit' > 2 000 000 (249 значений) 'Maximum O

```

```

feature_name = Maximum Open Credit
feature_value_max = 2000000
feature_value_min = 0

```

Количество

804958.0	1
653488.0	1
368192.0	1
3007136.0	1
243166.0	1
..	
323312.0	3
615714.0	3
349140.0	3
319110.0	5
0.0	65

Name: Maximum Open Credit, Length: 6963, dtype: int64

Отсортированные записи

2297	0.000000e+00
319	0.000000e+00
611	0.000000e+00
1427	0.000000e+00
294	0.000000e+00
...	
2763	4.092389e+07
2023	5.756256e+07
2617	2.655129e+08
44	3.800523e+08
617	1.304726e+09

Name: Maximum Open Credit, Length: 7500, dtype: float64

Первичный датасет

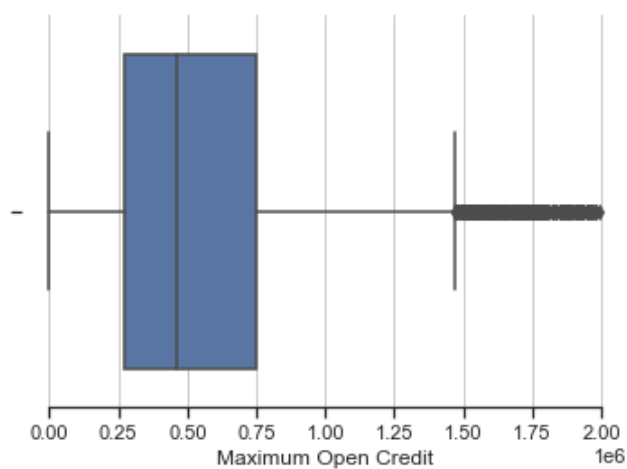
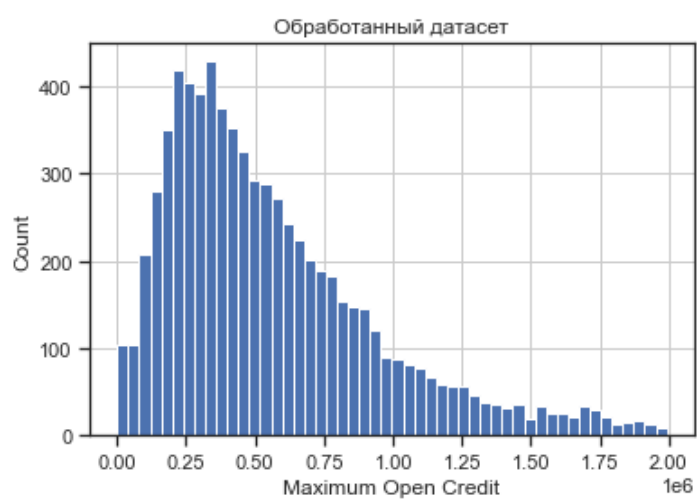
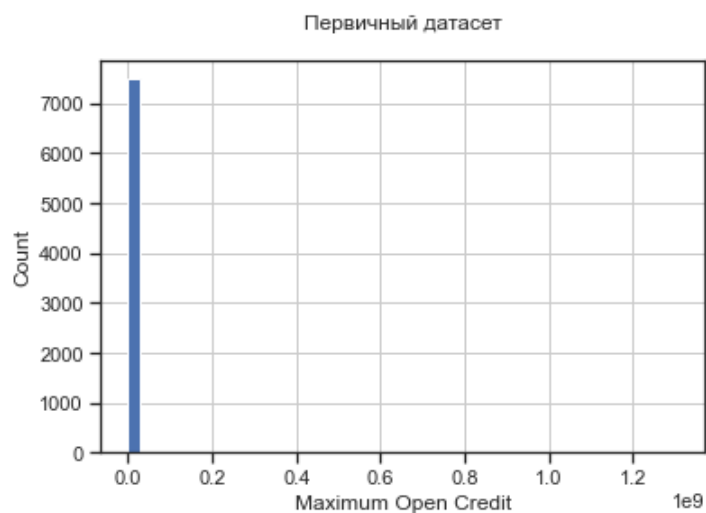
Мода датасета: 0.0
 Медиана датасета: 478159.0
 Среднее значение датасета: 945153.7274666667
 Максимальное значение датасета: 1304726170.0
 Минимальное значение датасета: 0.0

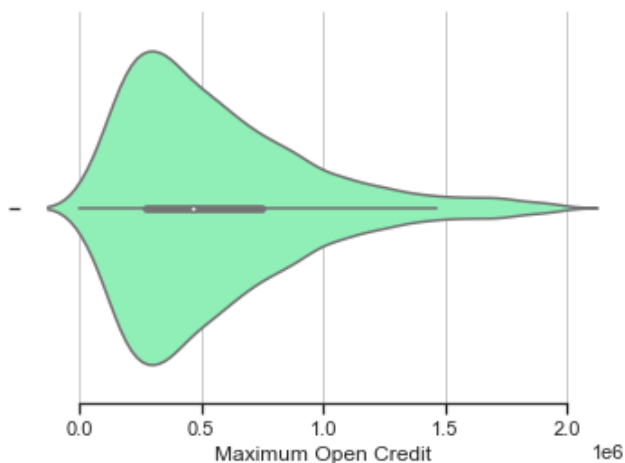
Количество записей в датасете: 7500
 Количество записей в датасете < 0: 0
 Количество записей в датасете > 2000000: 249

Обработанный датасет

Мода датасета: 0.0
 Медиана датасета: 463452.0
 Среднее значение датасета: 559819.0845400634
 Максимальное значение датасета: 1992298.0

Минимальное значение датасета: 0.0





Считаем выбросами значения 'Maximum Open Credit' > 2 000 000 (249 значений) 'Maximum Open Credit' < 50 000 (125 значений)

8. Number of Credit Problems - количество проблем с кредитом (категориальные данные)

```
B [26]: feature_name = 'Number of Credit Problems'
feature_value_max = 7
feature_value_min = 0
data_type = 1
# plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type)
```

9. Months since last delinquent - количество месяцев с последней просрочки платежа


```

B [27]: feature_name = 'Months since last delinquent'
feature_value_max = 83
feature_value_min = 0
data_type = 0
print(np.sort(df_train['Months since last delinquent'].unique()))

plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type)

# Считаем выбросами Months since last delinquent > 83 (5 значений)

```

```

[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13.
 14. 15. 16. 17. 18. 19. 20. 21. 22. 23. 24. 25. 26. 27.
 28. 29. 30. 31. 32. 33. 34. 35. 36. 37. 38. 39. 40. 41.
 42. 43. 44. 45. 46. 47. 48. 49. 50. 51. 52. 53. 54. 55.
 56. 57. 58. 59. 60. 61. 62. 63. 64. 65. 66. 67. 68. 69.
 70. 71. 72. 73. 74. 75. 76. 77. 78. 79. 80. 81. 82. 83.
 84. 86. 91. 92. 118. nan]
feature_name = Months since last delinquent
feature_value_max = 83
feature_value_min = 0

```

Количество

91.0	1
86.0	1
84.0	1
118.0	1
92.0	1
..	
13.0	65
33.0	68
8.0	68
29.0	71
14.0	76

Name: Months since last delinquent, Length: 89, dtype: int64

Отсортированные записи

5705	0.0
4995	0.0
4938	0.0
3063	0.0
257	0.0
...	
7494	NaN
7495	NaN
7497	NaN
7498	NaN
7499	NaN

Name: Months since last delinquent, Length: 7500, dtype: float64

Первичный датасет

Мода датасета: 14.0
 Медиана датасета: 32.0
 Среднее значение датасета: 34.69260017548991
 Максимальное значение датасета: 118.0
 Минимальное значение датасета: 0.0

Количество записей в датасете: 7500
 Количество записей в датасете < 0: 0
 Количество записей в датасете > 83: 5

Обработанный датасет

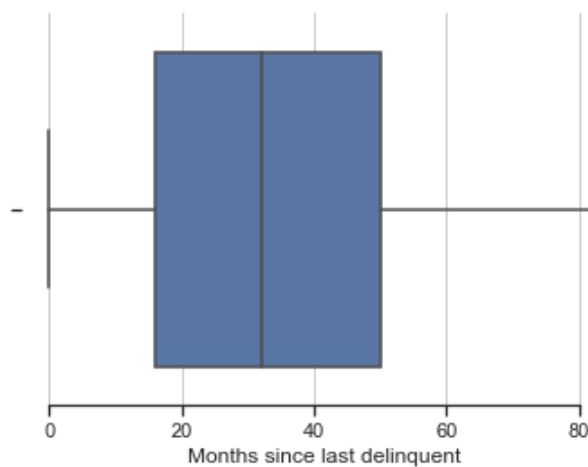
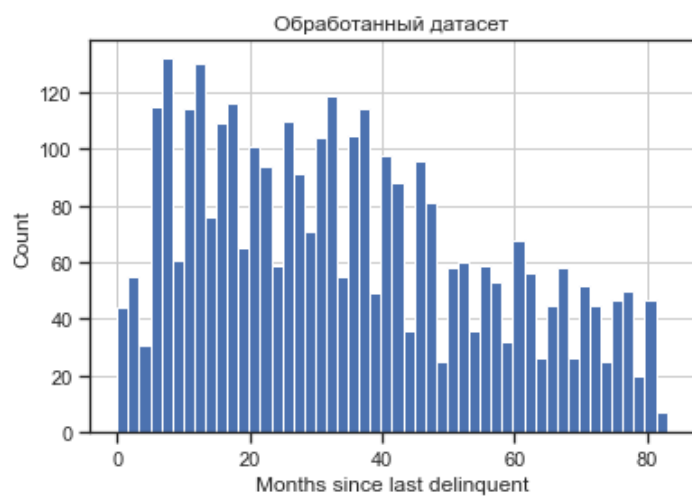
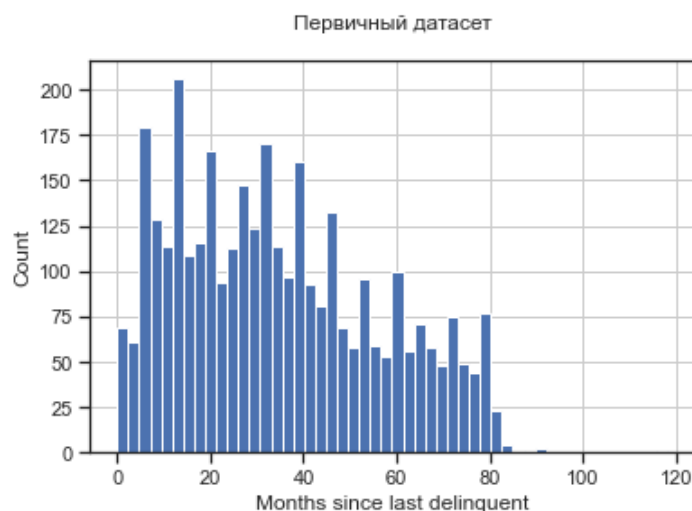
Мода датасета: 14.0

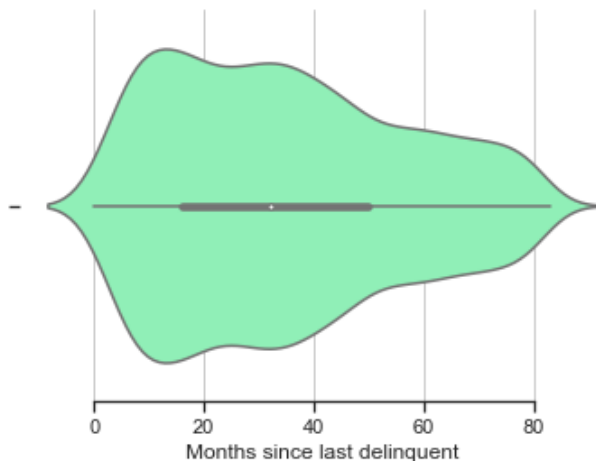
Медиана датасета: 32.0

Среднее значение датасета: 34.605448154657296

Максимальное значение датасета: 83.0

Минимальное значение датасета: 0.0





Считаем выбросами **Months since last delinquent** > 83

10. Bankruptcies - банкротства (категориальные данные)

```
B [28]: feature_name = 'Bankruptcies'
feature_value_max = 4
feature_value_min = 0
data_type = 1
# plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type)
```

11. Purpose - цель кредита (категориальные данные)

```
B [29]: feature_name = 'Purpose'
feature_value_max = 136679
feature_value_min = 0
data_type = 2
# plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type)
```

12. Term - срок кредита (категориальные данные)

```
B [30]: feature_name = 'Term'
feature_value_max = 136679
feature_value_min = 0
data_type = 2
# plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type)
```

13. Current Loan Amount - текущая сумма кредита

```

В [31]: feature_name = 'Current Loan Amount'
feature_value_max = 99999999
feature_value_min = 0
data_type = 0
plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type, 1)

# выбросы 99999999.0 (870 записей)

# Набор данных надо разбивать на два по сумме кредита: 1 - [0, ..., 2*10^7], 2 - [85*10^7

```

```

feature_name = Current Loan Amount
feature_value_max = 99999999
feature_value_min = 0

```

Количество

11242.0	1
21472.0	2
21516.0	1
21560.0	1
21582.0	1
...	
788634.0	2
788788.0	1
788942.0	1
789030.0	1
99999999.0	870

Name: Current Loan Amount, Length: 5386, dtype: int64

Отсортированные записи

1404	11242.0
4467	21472.0
2735	21472.0
7144	21516.0
5861	21560.0
...	
4384	99999999.0
732	99999999.0
4374	99999999.0
4555	99999999.0
0	99999999.0

Name: Current Loan Amount, Length: 7500, dtype: float64

Первичный датасет

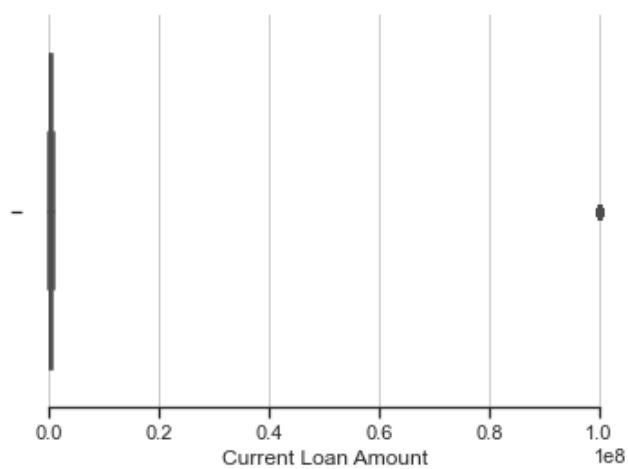
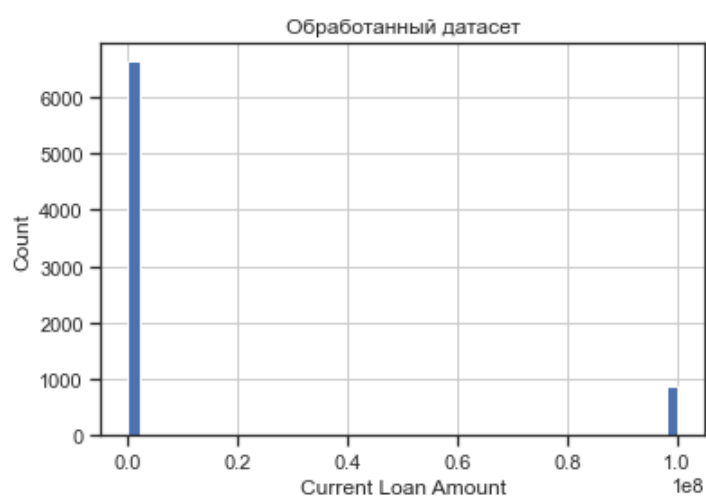
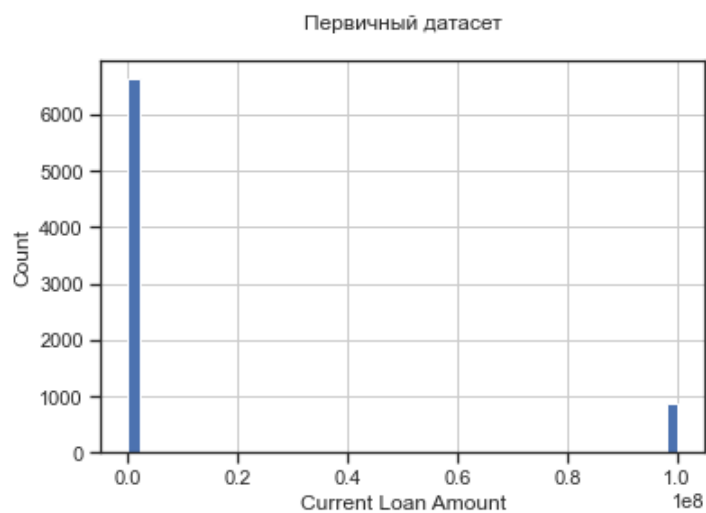
Мода датасета: 99999999.0
 Медиана датасета: 309573.0
 Среднее значение датасета: 11873177.445066666
 Максимальное значение датасета: 99999999.0
 Минимальное значение датасета: 11242.0

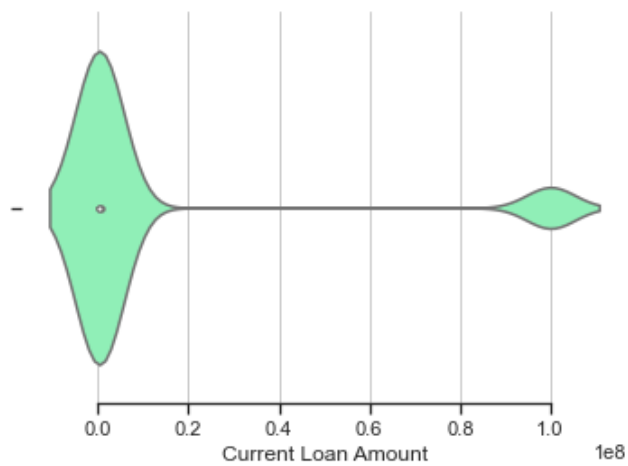
Количество записей в датасете: 7500
 Количество записей в датасете < 0: 0
 Количество записей в датасете > 99999999: 0

Обработанный датасет

Мода датасета: 99999999.0
 Медиана датасета: 309573.0
 Среднее значение датасета: 11873177.445066666
 Максимальное значение датасета: 99999999.0

Минимальное значение датасета: 11242.0





14. Current Credit Balance - текущий кредитный баланс

```

B [32]: feature_name = 'Current Credit Balance'
feature_value_max = 1300000
feature_value_min = 0
data_type = 0

plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type)

# Считаем выбросами значения 'Current Credit Balance' > 1300000 (106 значений)
# Считаем выбросами значения 'Current Credit Balance' > 2500000 (21 значений)

```

```

feature_name = Current Credit Balance
feature_value_max = 1300000
feature_value_min = 0

```

Количество

250477.0	1
474601.0	1
134900.0	1
150366.0	1
153026.0	1
..	
198911.0	4
136401.0	4
82289.0	4
191710.0	5
0.0	53

Name: Current Credit Balance, Length: 6592, dtype: int64

Отсортированные записи

4405	0.0
4274	0.0
1802	0.0
1464	0.0
2276	0.0
...	
7278	4209659.0
1580	4249673.0
4602	4367245.0
4745	4720132.0
4769	6506797.0

Name: Current Credit Balance, Length: 7500, dtype: float64

Первичный датасет

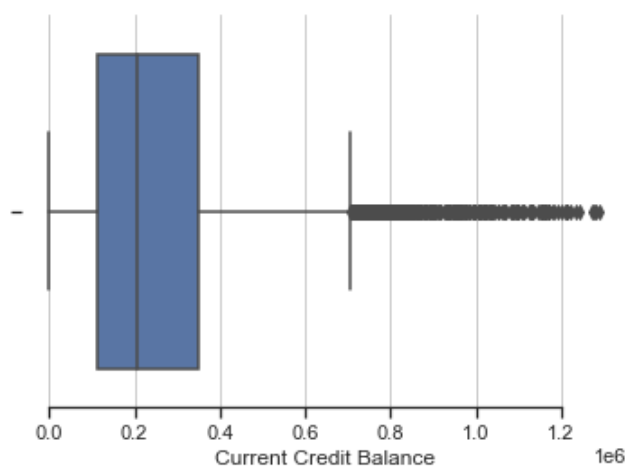
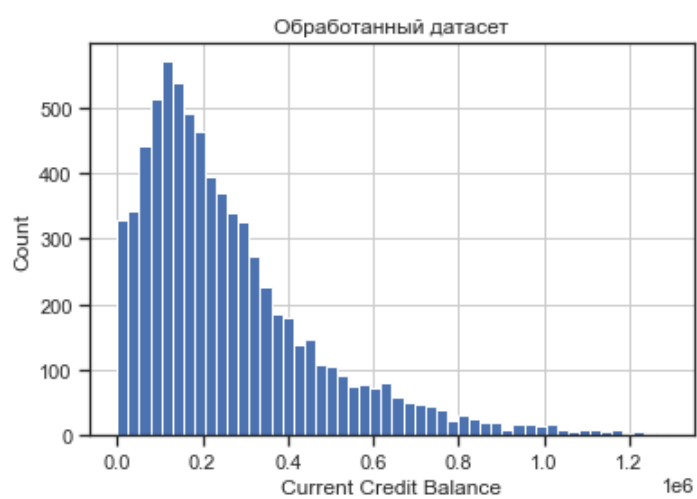
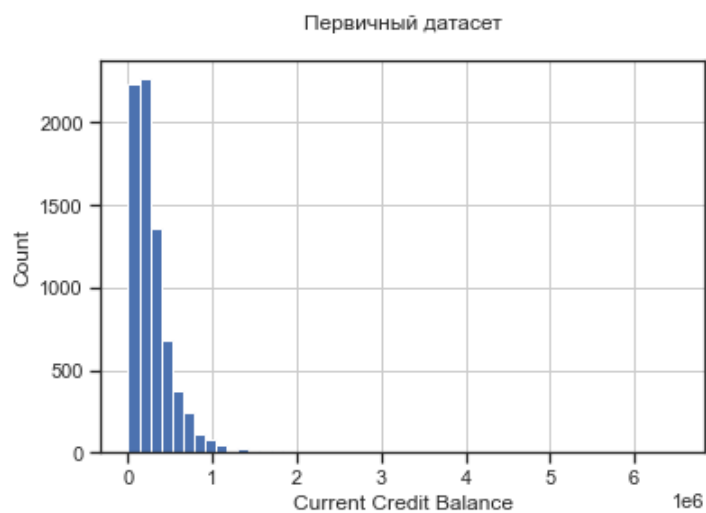
Мода датасета: 0.0
 Медиана датасета: 209323.0
 Среднее значение датасета: 289833.2352
 Максимальное значение датасета: 6506797.0
 Минимальное значение датасета: 0.0

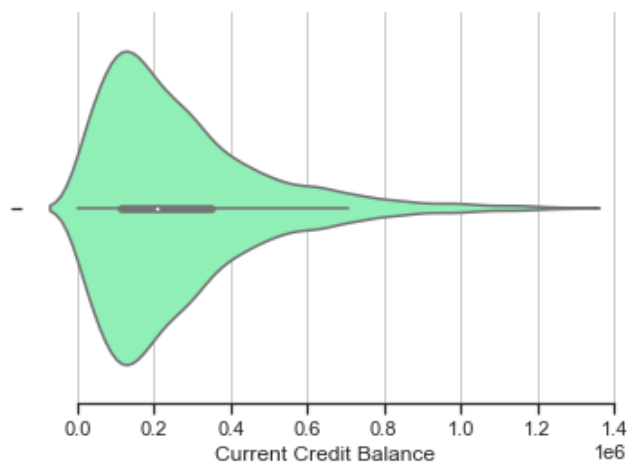
Количество записей в датасете: 7500
 Количество записей в датасете < 0: 0
 Количество записей в датасете > 1300000: 106

Обработанный датасет

Мода датасета: 0.0
 Медиана датасета: 206188.0
 Среднее значение датасета: 264468.74749797134
 Максимальное значение датасета: 1288504.0

Минимальное значение датасета: 0.0





15. Monthly Debt - ежемесячный долг

```

B [33]: feature_name = 'Monthly Debt'
# feature_value_max = 136679
feature_value_max = 55000
feature_value_min = 0 # 236
data_type = 0

plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type)

# Считаем выбросами значения 'Monthly Debt' > 55 000 (98 значений)
# Считаем выбросами значения 'Monthly Debt' > 80 000 (17 значений)

```

```

feature_name = Monthly Debt
feature_value_max = 55000
feature_value_min = 0

```

Количество

22292.0	1
23287.0	1
14015.0	1
21381.0	1
8390.0	1
..	
14848.0	3
11659.0	3
19667.0	4
19222.0	4
0.0	6

Name: Monthly Debt, Length: 6716, dtype: int64

Отсортированные записи

780	0.0
1643	0.0
7124	0.0
4165	0.0
3219	0.0
...	
6253	96177.0
6946	100091.0
2535	104036.0
1615	110311.0
4745	136679.0

Name: Monthly Debt, Length: 7500, dtype: float64

Первичный датасет

Мода датасета: 0.0
 Медиана датасета: 16076.5
 Среднее значение датасета: 18314.454133333333
 Максимальное значение датасета: 136679.0
 Минимальное значение датасета: 0.0

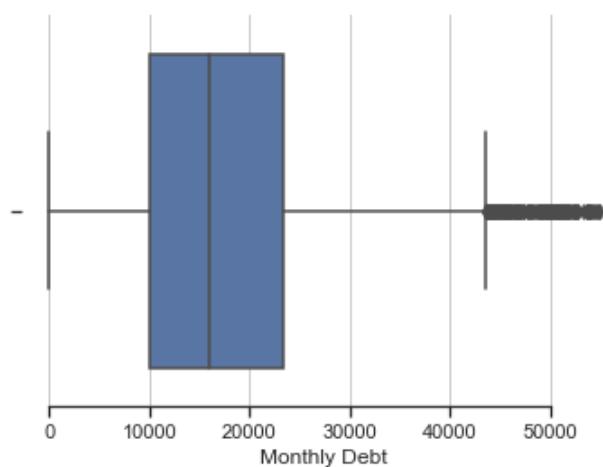
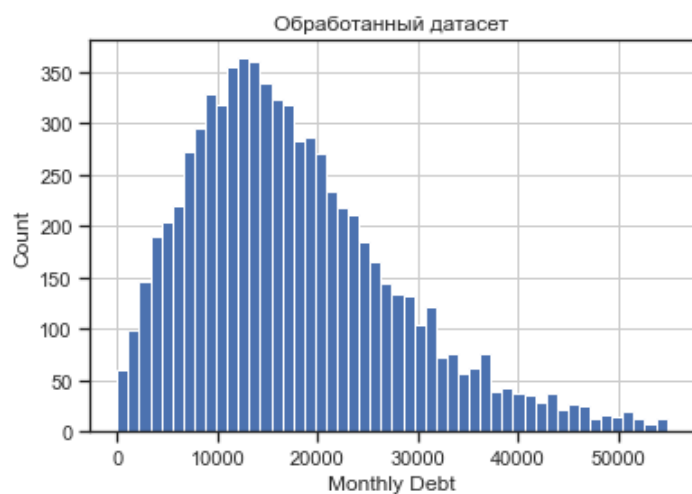
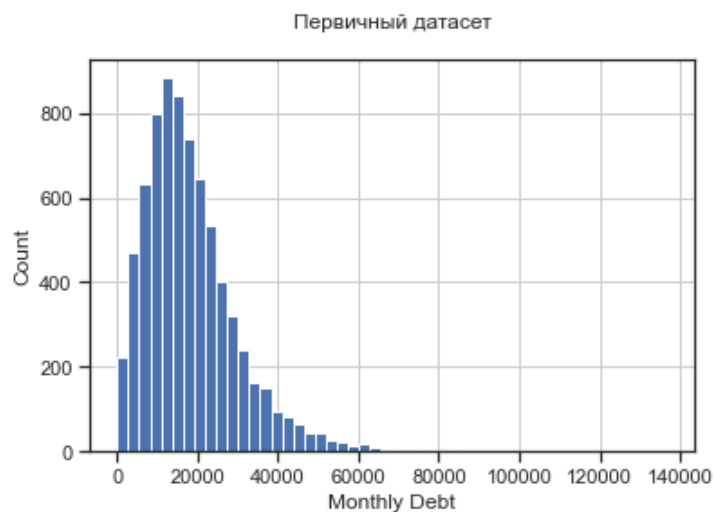
Количество записей в датасете: 7500
 Количество записей в датасете < 0: 0
 Количество записей в датасете > 55000: 98

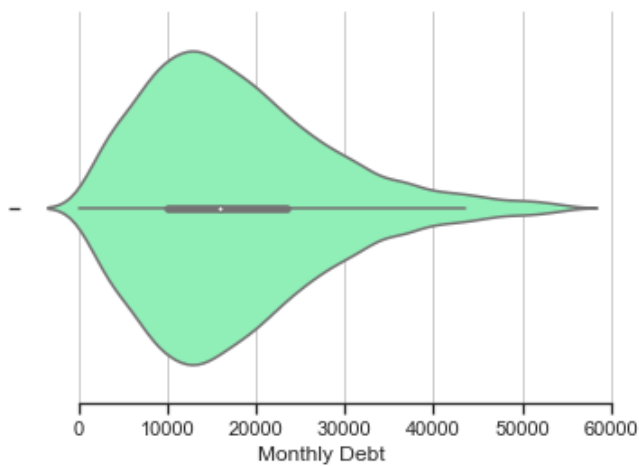
Обработанный датасет

Мода датасета: 0.0
 Медиана датасета: 15901.0
 Среднее значение датасета: 17654.054579843287

Максимальное значение датасета: 54882.0

Минимальное значение датасета: 0.0





Считаем выбросами значения 'Monthly Debt' > 80 000 (17 значений)

16. Credit Score - Кредитный рейтинг?

```

В [34]: feature_name = 'Credit Score'
feature_value_max = 1000
feature_value_min = 585
data_type = 0
plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type)

# Набор данных надо разбивать на два по Кредитному рейтингу: 1 - [585, ..., 800], 2 - [65
# Считаем выбросами значения 'Monthly Debt' < 585

```

```

feature_name = Credit Score
feature_value_max = 1000
feature_value_min = 585

```

Количество

7010.0	1
6150.0	1
604.0	1
629.0	1
6600.0	1

...

741.0	151
745.0	152
748.0	157
747.0	168
740.0	169

Name: Credit Score, Length: 268, dtype: int64

Отсортированные записи

599	585.0
6114	586.0
1455	588.0
3475	589.0
3491	590.0

...

7482	NaN
7492	NaN
7494	NaN
7498	NaN
7499	NaN

Name: Credit Score, Length: 7500, dtype: float64

Первичный датасет

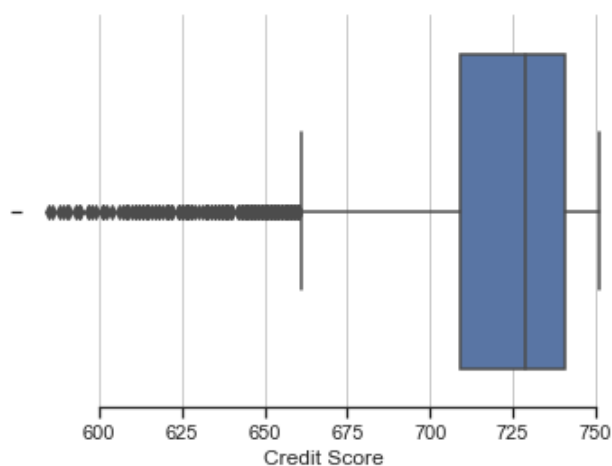
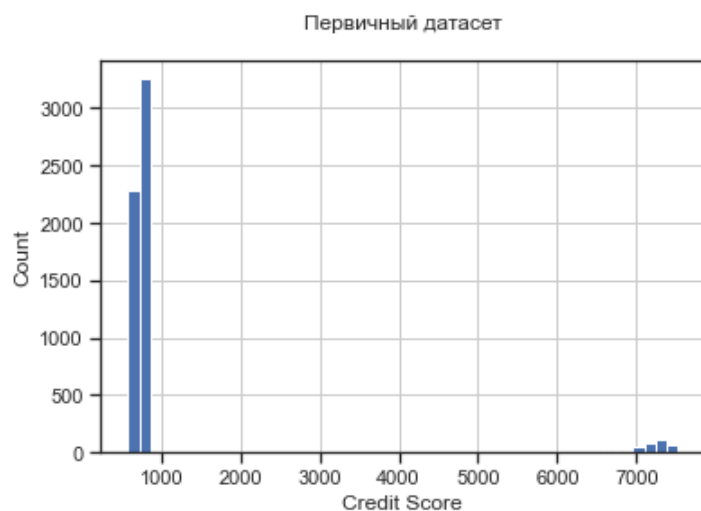
Мода датасета: 740.0
 Медиана датасета: 731.0
 Среднее значение датасета: 1151.0874978966851
 Максимальное значение датасета: 7510.0
 Минимальное значение датасета: 585.0

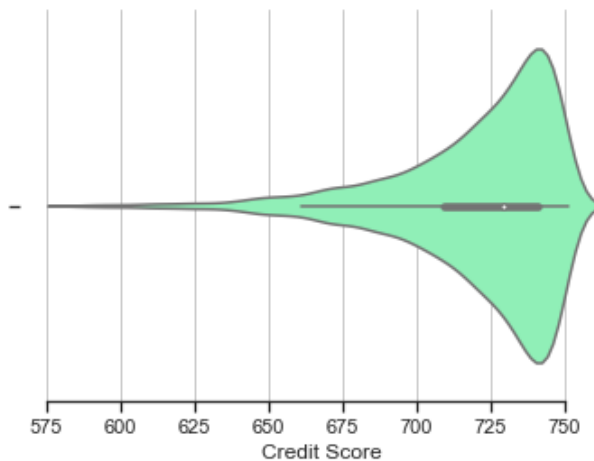
Количество записей в датасете: 7500
 Количество записей в датасете < 585: 0
 Количество записей в датасете > 1000: 400

Обработанный датасет

Мода датасета: 740.0
 Медиана датасета: 729.0
 Среднее значение датасета: 720.7059354140357
 Максимальное значение датасета: 751.0

Минимальное значение датасета: 585.0





1. **Home Ownership** - домовладение (категориальные данные)
2. **Annual Income** - годовой доход
 - **Считаем выбросами Annual Income > 4 000 000 (91 значения)**
 - Считаем выбросами Annual Income > 5 000 000 (44 значения)
3. **Years in current job** - количество лет на текущем месте работы (категориальные данные)
4. **Tax Liens** - налоговые обременения (категориальные данные)
5. **Number of Open Accounts** - количество открытых счетов
6. **Years of Credit History** - количество лет кредитной истории
 - **Считаем выбросами Years of Credit History > 40 (83 значения)**
 - Считаем выбросами Years of Credit History > 50 (8 значения)
7. **Maximum Open Credit** - наибольший открытый кредит
 - **Считаем выбросами значения 'Maximum Open Credit' > 4 000 000 (64 значений) 'Maximum Open Credit' < 50 000 (125 значений)**
 - Считаем выбросами значения 'Maximum Open Credit' > 2 000 000 (249 значений) 'Maximum Open Credit' < 50 000 (125 значений)
8. **Number of Credit Problems** - количество проблем с кредитом (категориальные данные)
9. **Months since last delinquent** - количество месяцев с последней просрочки платежа
 - **Считаем выбросами Months since last delinquent > 83 (5 значений)**
10. **Bankruptcies** - банкротства (категориальные данные)
11. **Purpose** - цель кредита (категориальные данные)
12. **Term** - срок кредита (категориальные данные)
13. **Current Loan Amount** - текущая сумма кредита
 - Набор данных надо разбивать на два по сумме кредита: 1 - [0, ..., 2 * 10⁷], 2 - [85 * 10⁷, ..., 1 * 10⁸]
 - Проверить коореляцию с Credit Score - Кредитный рейтинг
14. **Current Credit Balance** - текущий кредитный баланс
 - **Считаем выбросами значения 'Current Credit Balance' > 1300000 (106 значений)**
 - Считаем выбросами значения 'Current Credit Balance' > 2500000 (21 значений)
15. **Monthly Debt** - ежемесячный долг
 - **Считаем выбросами значения 'Monthly Debt' > 55 000 (98 значений)**

- Считаем выбросами значения 'Monthly Debt' > 80 000 (17 значений)

16. Credit Score - Кредитный рейтинг?

- Считаем выбросами значения 'Monthly Debt' < 585 и 'Monthly Debt' > 7510
- Набор данных надо разбивать на два по Кредитному рейтингу: 1 - [585, ..., 800], 2 - [6500, ..., 7500]
- Проверить корреляцию с Current Loan Amount - текущая сумма кредита

Анализ признакового пространства

Корреляция с базовыми признаками

```
In [35]: TARGET_NAME = 'Credit Default'
BASE_FEATURE_NAMES = df_train.columns.drop(TARGET_NAME).tolist()
BASE_FEATURE_NAMES
```

```
Out[35]: ['Home Ownership',
          'Annual Income',
          'Years in current job',
          'Tax Liens',
          'Number of Open Accounts',
          'Years of Credit History',
          'Maximum Open Credit',
          'Number of Credit Problems',
          'Months since last delinquent',
          'Bankruptcies',
          'Purpose',
          'Term',
          'Current Loan Amount',
          'Current Credit Balance',
          'Monthly Debt',
          'Credit Score']
```

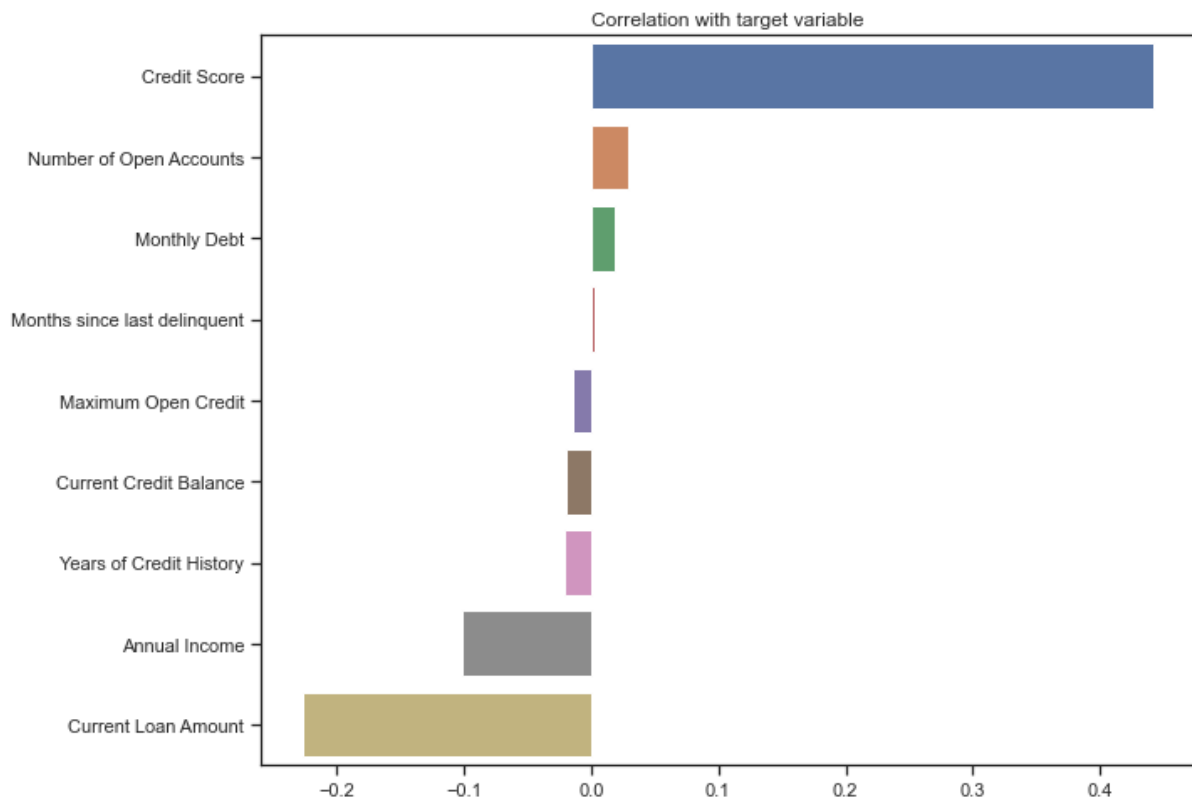


```
B [36]: corr_with_target = df_train[BASE_FEATURE_NAMES + [TARGET_NAME]].corr().iloc[:,-1].sort_values(ascending=True)

plt.figure(figsize=(10, 8))

sns.barplot(x=corr_with_target.values, y=corr_with_target.index)

plt.title('Correlation with target variable')
plt.show()
```



Матрица корреляций

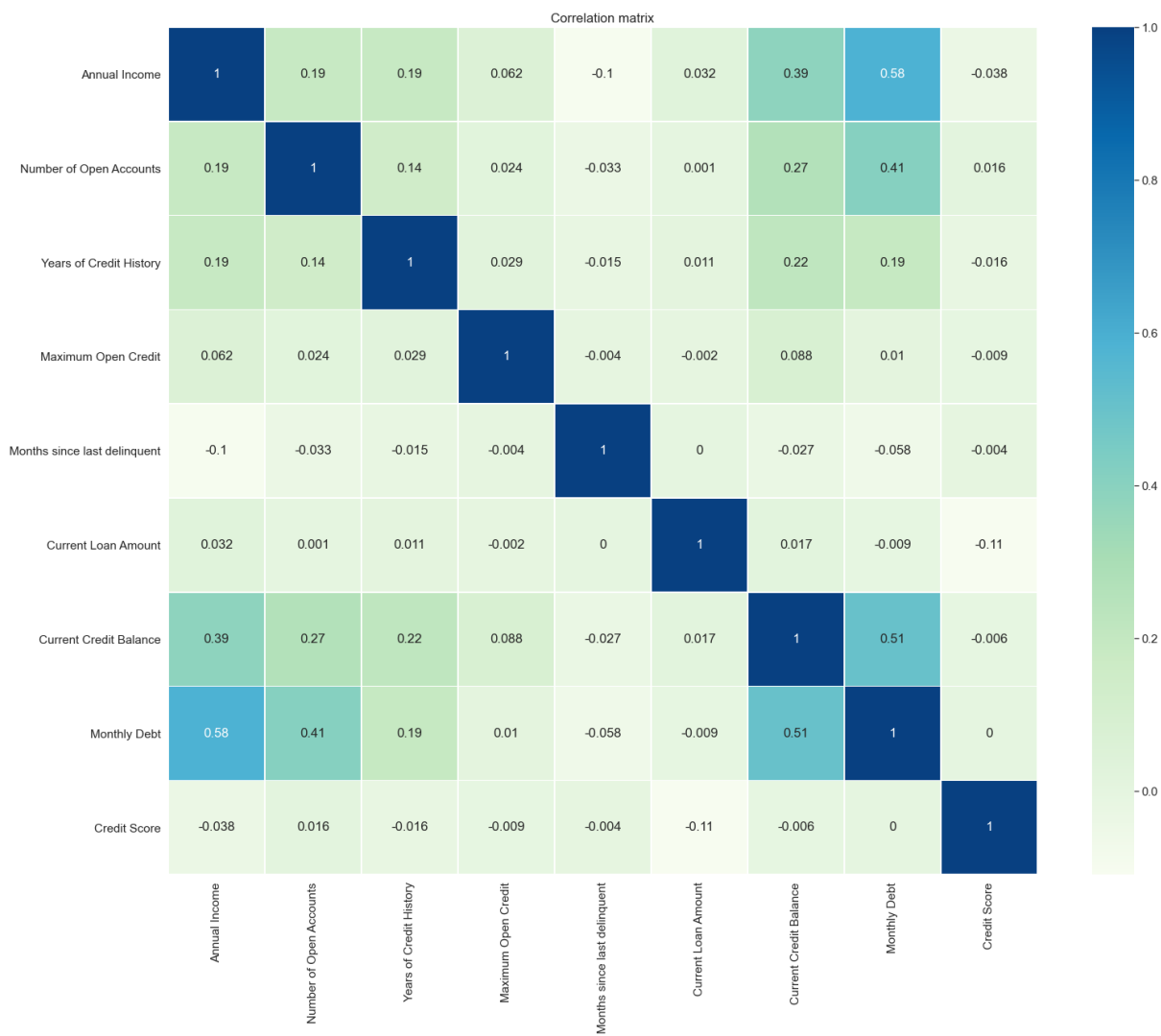
```

B [37]: plt.figure(figsize = (25,20))

sns.set(font_scale=1.4)
sns.heatmap(df_train[BASE_FEATURE_NAMES].corr().round(3), annot=True, linewidths=.5, cma

plt.title('Correlation matrix')
plt.show()

```



1. Наблюдается сильная положительная корреляция (**0.78**) между полями **'Current Loan Amount'** и **'Maximum Open Credit'**. Поэтому исключим из рассмотрения поле **'Maximum Open Credit'**
2. Наблюдается средняя положительная корреляция (**0.39**) между полями **'Number of Open Accounts'** и **'Maximum Open Credit'**.
3. Наблюдается средняя положительная корреляция (**0.37**) между полями **'Annual Income'** и **'Current Credit Balance'**.
4. Корреляции между **'Credit Score'** и **'Current Loan Amount'** слабая, отрицательная (**-0.084**).

В []:

Приведение типов

```
В [38]: for colname in ['Tax Liens', 'Number of Credit Problems', 'Bankruptcies']:
        df_train[colname] = df_train[colname].astype(str)
```

Обзор категориальных (номинативных, порядковых) признаков

Категориальные данные:

1. 'Home Ownership' (порядковые данные)

- Have Mortgage (ипотека) 12
 - Own Home 647
 - Rent 3204
 - Home Mortgage 3637
 - -
 - Name: Home Ownership, dtype: int64
-

3. 'Years in current job' (порядковые данные)

- 9 years 259
 - 8 years 339
 - 7 years 396
 - 6 years 426
 - 4 years 469
 - 1 year 504
 - 5 years 516
 - < 1 year 563
 - 3 years 620
 - 2 years 705
 - 10+ years 2332
 - -
 - Name: Years in current job, dtype: int64
-

4. 'Tax Liens' - налоговые обременения (порядковые данные)

- 7.0 1
 - 5.0 2
 - 6.0 2
 - 4.0 6
 - 3.0 10
 - 2.0 30
 - 1.0 83
 - 0.0 7366
 - -
 - Name: Tax Liens, dtype: int64
-

8. 'Number of Credit Problems' - количество проблем с кредитом (порядковые данные)

- 7.0 1
 - 6.0 4
 - 5.0 7
 - 4.0 9
 - 3.0 35
 - 2.0 93
 - 1.0 882
 - 0.0 6469
 - -
 - Name: Number of Credit Problems, dtype: int64
-

10. 'Bankruptcies' - банкротства (порядковые данные)

- 4.0 2
 - 3.0 7
 - 2.0 31
 - 1.0 786
 - 0.0 6660
 - -
 - Name: Bankruptcies, dtype: int64
-

11. Purpose - цель кредита (порядковые данные)

- renewable energy (Возобновляемая энергия) 2
- vacation (отпуск) 8
- educational expenses (расходы на образование) 10
- moving (переезд?) 11
- wedding (свадьба) 15
- small business 26
- buy house 34
- take a trip (отправиться в путешествие) 37
- major purchase (крупная покупка) 40
- medical bills (Медицинские счета) 71
- buy a car 96
- business loan (бизнес-кредит) 129
- home improvements (Домашние улучшения) 412
- other 665
- debt consolidation (консолидация долгов) 5944
- -
- Name: Purpose, dtype: int64

12. Term - срок кредита (номинативные данные)

- Long Term 1944
- Short Term 5556
-
- Name: Term, dtype: int64

```
B [39]: df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7500 entries, 0 to 7499
Data columns (total 17 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Home Ownership                        7500 non-null   object
1   Annual Income                        5943 non-null   float64
2   Years in current job                 7129 non-null   object
3   Tax Liens                            7500 non-null   object
4   Number of Open Accounts             7500 non-null   float64
5   Years of Credit History              7500 non-null   float64
6   Maximum Open Credit                 7500 non-null   float64
7   Number of Credit Problems            7500 non-null   object
8   Months since last delinquent         3419 non-null   float64
9   Bankruptcies                        7500 non-null   object
10  Purpose                              7500 non-null   object
11  Term                                7500 non-null   object
12  Current Loan Amount                 7500 non-null   float64
13  Current Credit Balance              7500 non-null   float64
14  Monthly Debt                       7500 non-null   float64
15  Credit Score                        5943 non-null   float64
16  Credit Default                      7500 non-null   int64
dtypes: float64(9), int64(1), object(7)
memory usage: 996.2+ KB
```

```
B [40]: df_train.select_dtypes(include='object').columns
```

```
Out[40]: Index(['Home Ownership', 'Years in current job', 'Tax Liens',
               'Number of Credit Problems', 'Bankruptcies', 'Purpose', 'Term'],
              dtype='object')
```

Обзор значений категориальных признаков

```
B [41]: for cat_colname in df_train.select_dtypes(include='object').columns:
        print(str(cat_colname) + '\n\n' + str(df_train[cat_colname].value_counts()) + '\n' +
              # Bankruptcies имеет странное значение 'nan' (14 значений), нужно заменить на 0
```

Home Ownership

```
Home Mortgage    3637
Rent              3204
Own Home          647
Have Mortgage     12
```

Name: Home Ownership, dtype: int64

```
*****
*****
```

Years in current job

```
10+ years    2332
2 years       705
3 years       620
< 1 year     563
5 years       516
1 year        504
4 years       469
6 years       426
7 years       396
8 years       339
9 years       259
```

Name: Years in current job, dtype: int64

```
*****
*****
```

Tax Liens

```
0.0    7366
1.0     83
2.0     30
3.0     10
4.0      6
6.0      2
5.0      2
7.0      1
```

Name: Tax Liens, dtype: int64

```
*****
*****
```

Number of Credit Problems

```
0.0    6469
1.0     882
2.0     93
3.0     35
4.0      9
5.0      7
6.0      4
7.0      1
```

Name: Number of Credit Problems, dtype: int64

```
*****
*****
```

Bankruptcies

```
0.0    6660
1.0     786
2.0     31
nan     14
3.0      7
```

4.0 2

Name: Bankruptcies, dtype: int64

```
*****
*****
```

Purpose

```
debt consolidation      5944
other                   665
home improvements      412
business loan          129
buy a car               96
medical bills           71
major purchase          40
take a trip             37
buy house               34
small business          26
wedding                 15
moving                  11
educational expenses    10
vacation                 8
renewable energy        2
```

Name: Purpose, dtype: int64

```
*****
*****
```

Term

```
Short Term      5556
Long Term       1944
```

Name: Term, dtype: int64

```
*****
*****
```

2. Обработка выбросов

2. Annual Income - годовой доход

- Считаем выбросами Annual Income > 4 000 000 (91 значения) и Annual Income < 164597
- Считаем выбросами Annual Income > 5 000 000 (44 значения) и Annual Income < 164597

6. Years of Credit History - количество лет кредитной истории

- Считаем выбросами Years of Credit History > 40 (83 значения)
- Считаем выбросами Years of Credit History > 50 (8 значения)

7. Maximum Open Credit - наибольший открытый кредит

- Считаем выбросами значения 'Maximum Open Credit' > 4 000 000 (64 значений) 'Maximum Open Credit' < 50 000 (125 значений)
- Считаем выбросами значения 'Maximum Open Credit' > 2 000 000 (249 значений) 'Maximum Open Credit' < 50 000 (125 значений)

9. Months since last delinquent - количество месяцев с последней просрочки платежа

- Более 3500 null значений - удаляем столбец
- Считаем выбросами Months since last delinquent > 83 (5 значений)

13. Current Loan Amount - текущая сумма кредита

- Набор данных надо разбивать на два по сумме кредита: 1 - [0, ..., 2 * 10⁷], 2 - [85 * 10⁷, ..., 1 * 10⁸]
- Проверить коореляцию с Credit Score - Кредитный рейтинг

14. Current Credit Balance - текущий кредитный баланс

- Считаем выбросами значения 'Current Credit Balance' > 1300000 (106 значений)
- Считаем выбросами значения 'Current Credit Balance' > 2500000 (21 значений)

15. Monthly Debt - ежемесячный долг

- Считаем выбросами значения 'Monthly Debt' > 55 000 (98 значений)
- Считаем выбросами значения 'Monthly Debt' > 80 000 (17 значений)

16. Credit Score - Кредитный рейтинг?

- Считаем выбросами значения 'Monthly Debt' < 585 и 'Monthly Debt' > 7510
- Набор данных надо разбивать на два по Кредитному рейтингу: 1 - [585, ..., 800], 2 - [6500, ..., 7500]
- Проверить коореляцию с Current Loan Amount - текущая сумма кредита

3. Обработка пропусков

```
In [42]: df_train.isnull()
#df_example.notnull()
```

Out[42]:

	Home Ownership	Annual Income	Years in current job	Tax Liens	Number of Open Accounts	Years of Credit History	Maximum Open Credit	Number of Credit Problems	Months since last delinquent	Bankruptcies
0	False	False	True	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	True	False
2	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	True	False
4	False	False	False	False	False	False	False	False	True	False
...
7495	False	False	False	False	False	False	False	False	True	False
7496	False	False	False	False	False	False	False	False	False	False
7497	False	False	False	False	False	False	False	False	True	False
7498	False	True	True	False	False	False	False	False	True	False
7499	False	True	False	False	False	False	False	False	True	False

7500 rows × 17 columns




```
B [43]: #len(df_train) - df_train.count()
df_train.isna().sum() # просматриваем пропуски
```

```
Out[43]: Home Ownership      0
Annual Income      1557
Years in current job    371
Tax Liens          0
Number of Open Accounts  0
Years of Credit History  0
Maximum Open Credit    0
Number of Credit Problems  0
Months since last delinquent  4081
Bankruptcies         0
Purpose             0
Term                0
Current Loan Amount    0
Current Credit Balance  0
Monthly Debt          0
Credit Score          1557
Credit Default         0
dtype: int64
```

Нулевые значения имеются в столбцах "Annual Income", "Years in current job", "Months since last delinquent" и "Credit Score"

```
B [44]: #df_train.info()
#df_train = df_train.fillna(median)
```

Years in current job - количество лет на текущем месте работы

```
B [45]: # количество пропусков
df_train['Years in current job'].isnull().sum()
```

```
Out[45]: 371
```

```
B [46]: df_train[cat_colname] = df_train[cat_colname].replace(to_replace = np.nan, value = 'неиз')
```

```
B [47]: cat_colname = 'Years in current job'
print(str(cat_colname) + '\n\n' + str(df_train[cat_colname].value_counts()) + '\n' + '*')
```

Years in current job

```
10+ years    2332
2 years      705
3 years      620
< 1 year     563
5 years      516
1 year       504
4 years      469
6 years      426
7 years      396
8 years      339
9 years      259
```

Name: Years in current job, dtype: int64

```
*****
*****
```

```
In [48]: df_train.isna().sum() # просматриваем пропуски
```

```
Out[48]: Home Ownership          0
Annual Income          1557
Years in current job    371
Tax Liens              0
Number of Open Accounts 0
Years of Credit History 0
Maximum Open Credit     0
Number of Credit Problems 0
Months since last delinquent 4081
Bankruptcies           0
Purpose                0
Term                   0
Current Loan Amount     0
Current Credit Balance   0
Monthly Debt            0
Credit Score           1557
Credit Default          0
dtype: int64
```

Очистка данных

Класс с подготовкой данных

```
In [49]: # Считаем выбросами Годовой доход 'Annual Income' > 4 000 000 (91 значения) и Annual Inc
# Считаем выбросами Количество лет кредитной истории 'Years of Credit History' > 40 (83
# Считаем выбросами Набольший открытый кредит 'Maximum Open Credit' > 4 000 000 (64 зна
# и 'Maximum Open Credit' < 50 000 (125 значений)
# Считаем выбросами Количество месяцев с последней просрочки платежа Months since last d
# Считаем выбросами Текущий кредитный баланс 'Current Credit Balance' > 1300000 (106 зна
# Считаем выбросами Ежемесячный долг 'Monthly Debt' > 55 000 (98 значений)
# Считаем выбросами Кредитный рейтинг 'Monthly Debt' < 585 и 'Monthly Debt' > 7510
```

```

В [50]: class DataPipeline:
        """Подготовка исходных данных"""

        def __init__(self):
            """Параметры класса:
            Константы для обработки выбросов"""

            self.medians = None
            self.modes = None

            self.AnnualIncome_min = 165000
            self.AnnualIncome_max = 4000000

            self.YearsofCreditHistory_max = 40

            self.MaximumOpenCredit_min = 50000
            self.MaximumOpenCredit_max = 4000000

            self.MonthsSinceLastDelinquent_max = 83
            self.CurrentLoanAmount_max = 1000000
            self.CurrentCreditBalance_max = 1300000
            self.MonthlyDebt_max = 55000

            self.MonthlyDebt_min = 585
            self.MonthlyDebt_max = 7510

        def fit(self, df):
            """Сохранение статистик"""

            # Расчёт медиан
            self.medians = df_train[['Annual Income', 'Credit Score']].median()
            df = df_train.loc[df_train['Current Loan Amount'] < self.CurrentLoanAmount_max,
            self.modes = df[['Current Loan Amount']].median()

        def transform(self, df):
            """Трансформация данных"""

            # 1. Обработка пропусков
            #df_train = df_train.fillna(median)

            df[['Annual Income', 'Credit Score']] = df[['Annual Income', 'Credit Score']].fi

            # Months since last delinquent
            # 3581 пропущенное значение из 7500 - удаляем
            if 'Months since last delinquent' in df.columns:
                # df = df.drop(['Months since last delinquent'], axis=1)
                df.drop('Months since last delinquent', axis=1, inplace=True)

            # Years in current job
            cat_colname = 'Years in current job'
            df[cat_colname] = df[cat_colname].replace(to_replace = np.nan, value = 'неизвест

            # 2. Выбросы (outliers)

            # Annual Income - годовой доход
            df.loc[df['Annual Income'] < self.AnnualIncome_min, 'Annual Income'] = self.Annu
            df.loc[df['Annual Income'] >= self.AnnualIncome_max, 'Annual Income'] = self.Ann

            # Years of Credit History - Количество лет кредитной истории
            df.loc[df['Years of Credit History'] >= self.YearsofCreditHistory_max, 'Years of

            # Maximum Open Credit - наибольший открытый кредит
            df.loc[df['Maximum Open Credit'] < self.MaximumOpenCredit_min, 'Maximum Open Cre
            df.loc[df['Maximum Open Credit'] >= self.MaximumOpenCredit_max, 'Maximum Open Cr

```

```

# Current Loan Amount - текущая сумма кредита
df.loc[df['Current Loan Amount'] >= self.CurrentLoanAmount_max, 'Current Loan Am

# Current Credit Balance - текущий кредитный баланс
df.loc[df['Current Credit Balance'] >= self.CurrentCreditBalance_max, 'Current C

# Monthly Debt - Ежемесячный долг
df.loc[df['Monthly Debt'] >= self.MonthlyDebt_max, 'Monthly Debt'] = self.Monthl

# Monthly Debt - Кредитный рейтинг
df.loc[df['Monthly Debt'] < self.MonthlyDebt_min, 'Monthly Debt'] = self.Monthly
df.loc[df['Monthly Debt'] >= self.MonthlyDebt_max, 'Monthly Debt'] = self.Monthl

# 3. Обработка категорий
colname = 'Bankruptcies'
df[colname] = df[colname].replace(to_replace = 'nan', value = '0.0')
# (создание дамми-переменных)
#df = pd.concat([df, pd.get_dummies(df['Tax Liens'], prefix='Tax Liens', dtype='
#df = pd.concat([df, pd.get_dummies(df['Number of Credit Problems'], prefix='Num
#df = pd.concat([df, pd.get_dummies(df['Bankruptcies'], prefix='Bankruptcies', d

return df

def features(self, df):
    """4. Feature engineering
        Генерация новых фич"""

    # 1. Home Ownership - домовладение
    cat_colname = 'Home_Ownership_int'

    df[cat_colname] = df['Home Ownership']
    df.loc[df[cat_colname] == 'Have Mortgage', cat_colname] = 0
    df.loc[df[cat_colname] == 'Own Home', cat_colname] = 1
    df.loc[df[cat_colname] == 'Rent', cat_colname] = 2
    df.loc[df[cat_colname] == 'Home Mortgage', cat_colname] = 3

    # 3. 'Years in current job' (порядковые данные)
    cat_colname = 'Years_in_current_job_int'

    df[cat_colname] = df['Years in current job']
    df.loc[df[cat_colname] == '< 1 year', cat_colname] = 0
    df.loc[df[cat_colname] == '1 year', cat_colname] = 1
    df.loc[df[cat_colname] == '2 years', cat_colname] = 2
    df.loc[df[cat_colname] == '3 years', cat_colname] = 3
    df.loc[df[cat_colname] == '4 years', cat_colname] = 4
    df.loc[df[cat_colname] == '5 years', cat_colname] = 5
    df.loc[df[cat_colname] == '6 years', cat_colname] = 6
    df.loc[df[cat_colname] == '7 years', cat_colname] = 7
    df.loc[df[cat_colname] == '8 years', cat_colname] = 8
    df.loc[df[cat_colname] == '9 years', cat_colname] = 9
    df.loc[df[cat_colname] == '10+ years', cat_colname] = 10
    df.loc[df[cat_colname] == 'неизвестно', cat_colname] = 11

    # 11. Purpose - цель кредита (порядковые данные)
    cat_colname = 'Purpose_int'

    df[cat_colname] = df['Purpose']
    df.loc[df[cat_colname] == 'renewable energy', cat_colname] = 0
    df.loc[df[cat_colname] == 'vacation', cat_colname] = 1
    df.loc[df[cat_colname] == 'educational expenses', cat_colname] = 2
    df.loc[df[cat_colname] == 'moving', cat_colname] = 3
    df.loc[df[cat_colname] == 'wedding', cat_colname] = 4
    df.loc[df[cat_colname] == 'small business', cat_colname] = 5
    df.loc[df[cat_colname] == 'buy house', cat_colname] = 6
    df.loc[df[cat_colname] == 'take a trip', cat_colname] = 7
    df.loc[df[cat_colname] == 'major purchase', cat_colname] = 8
    df.loc[df[cat_colname] == 'medical bills', cat_colname] = 9

```

```

df.loc[df[cat_colname] == 'buy a car', cat_colname] = 10
df.loc[df[cat_colname] == 'business loan', cat_colname] = 11
df.loc[df[cat_colname] == 'home improvements', cat_colname] = 12
df.loc[df[cat_colname] == 'other', cat_colname] = 13
df.loc[df[cat_colname] == 'debt consolidation', cat_colname] = 14

# 12. Term - срок кредита (номинативные данные)
cat_colname = 'Term_int'

df[cat_colname] = df['Term']
df.loc[df[cat_colname] == 'Long Term', cat_colname] = 0
df.loc[df[cat_colname] == 'Short Term', cat_colname] = 1

numbers = ['0.0', '1.0', '2.0', '3.0', '4.0', '5.0', '6.0', '7.0', '8.0', '9.0']
numbers_int = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Добавление признаков
colnames_new = ['Tax_Liens_int', 'Number_of_Credit_Problems_int', 'Bankruptcies_
colnames = ['Tax Liens', 'Number of Credit Problems', 'Bankruptcies']

for i in range(len(colnames_new)):
    df[colnames_new[i]] = df[colnames[i]]
    for j in range(len(numbers)):
        df.loc[df[colnames_new[i]] == numbers[j], colnames_new[i]] = numbers_int[j]

# Обработка категорий
for colname in ['Home_Ownership_int', 'Years_in_current_job_int', 'Purpose_int',
df_train[colname] = df_train[colname].astype('int8')
for colname in colnames_new:
    df_train[colname] = df_train[colname].astype('int8')

# 16. Credit Score - Кредитный рейтинг
df['CreditScore_small'] = df['Credit Score']
df['CreditScore_large'] = df['Credit Score']

df.loc[df['Credit Score'] > 2000, 'CreditScore_small'] = 0.0
df.loc[df['Credit Score'] < 600, 'CreditScore_small'] = 0.0

df.loc[df['Credit Score'] < 3000, 'CreditScore_large'] = 0.0
df.loc[df['Credit Score'] > 9000, 'CreditScore_large'] = 0.0

return df

```

Инициализируем класс

```

B [51]: data_pl = DataPipeLine()

# тренировочные данные
data_pl.fit(df_train)

df = data_pl.transform(df_train)

```

```

B [52]: df = data_pl.features(df_train)

```

```
B [53]: #df.columns
#df.describe()
df.info() # Рассмотрим типы признаков
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7500 entries, 0 to 7499
Data columns (total 25 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Home Ownership                        7500 non-null   object
1   Annual Income                        7500 non-null   float64
2   Years in current job                 7500 non-null   object
3   Tax Liens                            7500 non-null   object
4   Number of Open Accounts              7500 non-null   float64
5   Years of Credit History              7500 non-null   float64
6   Maximum Open Credit                 7500 non-null   float64
7   Number of Credit Problems            7500 non-null   object
8   Bankruptcies                        7500 non-null   object
9   Purpose                             7500 non-null   object
10  Term                                 7500 non-null   object
11  Current Loan Amount                  7500 non-null   float64
12  Current Credit Balance               7500 non-null   float64
13  Monthly Debt                        7500 non-null   float64
14  Credit Score                        7500 non-null   float64
15  Credit Default                      7500 non-null   int64
16  Home_Ownership_int                  7500 non-null   int8
17  Years_in_current_job_int            7500 non-null   int8
18  Purpose_int                         7500 non-null   int8
19  Term_int                           7500 non-null   int8
20  Tax_Liens_int                      7500 non-null   int8
21  Number_of_Credit_Problems_int       7500 non-null   int8
22  Bankruptcies_int                   7500 non-null   int8
23  CreditScore_small                   7500 non-null   float64
24  CreditScore_large                   7500 non-null   float64
dtypes: float64(10), int64(1), int8(7), object(7)
memory usage: 1.1+ MB
```

```
B [54]: colname = 'Bankruptcies'
df[colname] = df[colname].replace(to_replace = 'nan', value = '0.0')

#for cat_colname in df.select_dtypes(include='object').columns:
for cat_colname in df.select_dtypes(include='int8').columns:
    print(str(cat_colname) + '\n\n' + str(df[cat_colname].value_counts()) + '\n' + '*' *

```

Home_Ownership_int

```
3    3637
2    3204
1     647
0      12
```

Name: Home_Ownership_int, dtype: int64

```
*****
*****
```

Years_in_current_job_int

```
10    2332
2      705
3      620
0      563
5      516
1      504
4      469
6      426
7      396
11     371
8      339
9      259
```

Name: Years_in_current_job_int, dtype: int64

```
*****
*****
```

Purpose_int

```
14    5944
13     665
12     412
11     129
10      96
9       71
8       40
7       37
6       34
5       26
4       15
3       11
2       10
1        8
0        2
```

Name: Purpose_int, dtype: int64

```
*****
*****
```

Term_int

```
1    5556
0    1944
```

Name: Term_int, dtype: int64

```
*****
*****
```

Tax_Liens_int

```
0    7366
```

1	83
2	30
3	10
4	6
6	2
5	2
7	1

Name: Tax_Liens_int, dtype: int64

Number_of_Credit_Problems_int

0	6469
1	882
2	93
3	35
4	9
5	7
6	4
7	1

Name: Number_of_Credit_Problems_int, dtype: int64

Bankruptcies_int

0	6674
1	786
2	31
3	7
4	2

Name: Bankruptcies_int, dtype: int64


```

B [55]: feature_name = 'CreditScore_small'
feature_value_max = 1000
feature_value_min = 600
data_type = 0
#plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type)
plot_feature(feature_name, df, feature_value_max, feature_value_min, data_type)

```

```

feature_name = CreditScore_small
feature_value_max = 1000
feature_value_min = 600

```

Количество

629.0	1
604.0	1
602.0	1
619.0	1
620.0	1
...	
748.0	157
747.0	168
740.0	169
0.0	414
731.0	1651

Name: CreditScore_small, Length: 148, dtype: int64

Отсортированные записи

3749	0.0
325	0.0
1862	0.0
5899	0.0
1875	0.0
...	
1849	751.0
873	751.0
1957	751.0
5795	751.0
6584	751.0

Name: CreditScore_small, Length: 7500, dtype: float64

Первичный датасет

Мода датасета: 731.0
 Медиана датасета: 731.0
 Среднее значение датасета: 683.2993333333334
 Максимальное значение датасета: 751.0
 Минимальное значение датасета: 0.0

Количество записей в датасете: 7500
 Количество записей в датасете < 600: 414
 Количество записей в датасете > 1000: 0

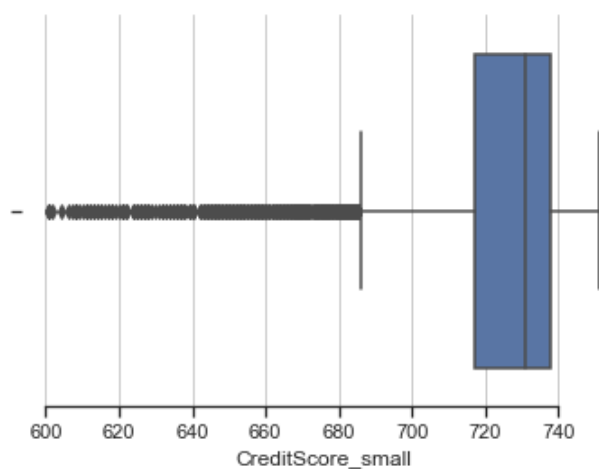
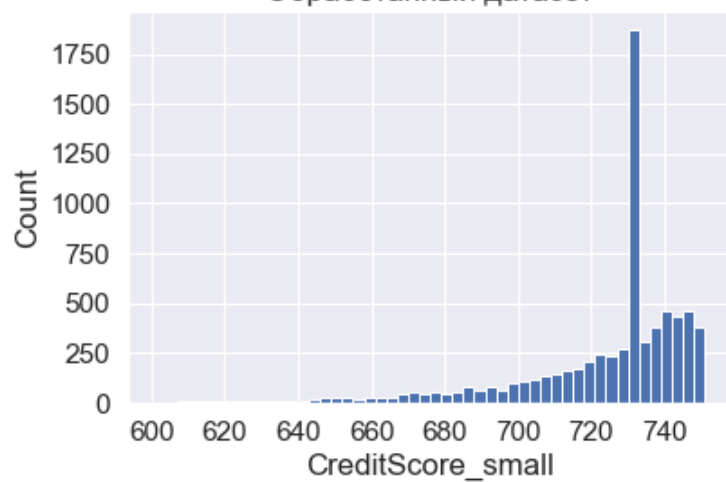
Обработанный датасет

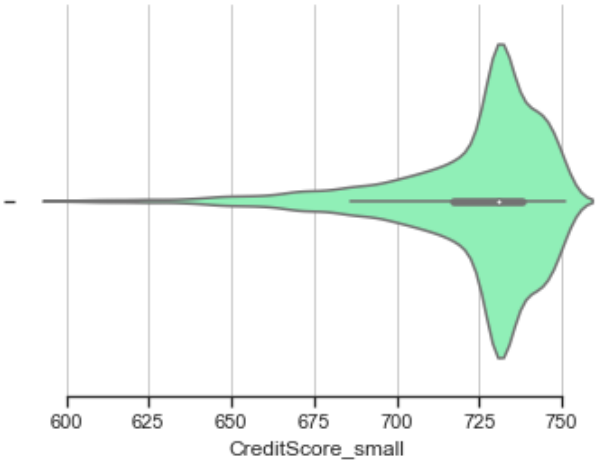
Мода датасета: 731.0
 Медиана датасета: 731.0
 Среднее значение датасета: 723.2211402766018
 Максимальное значение датасета: 751.0
 Минимальное значение датасета: 601.0

Первичный датасет



Обработанный датасет





```
B [56]: feature_name = 'CreditScore_large'
feature_value_max = 10000
feature_value_min = 3000
data_type = 0
#plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type)
plot_feature(feature_name, df, feature_value_max, feature_value_min, data_type)
```

```
feature_name = CreditScore_large
feature_value_max = 10000
feature_value_min = 3000
```

Количество

6670.0	1
6710.0	1
6170.0	1
6770.0	1
6570.0	1
...	
7370.0	12
7330.0	13
7300.0	13
7400.0	15
0.0	7100

Name: CreditScore_large, Length: 111, dtype: int64

Отсортированные записи

0	0.0
4922	0.0
4921	0.0
4920	0.0
4919	0.0
...	
3063	7490.0
355	7500.0
2408	7500.0
2213	7510.0
3688	7510.0

Name: CreditScore_large, Length: 7500, dtype: float64

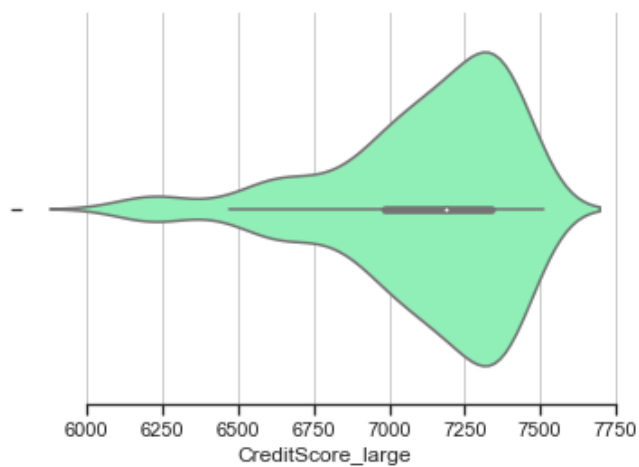
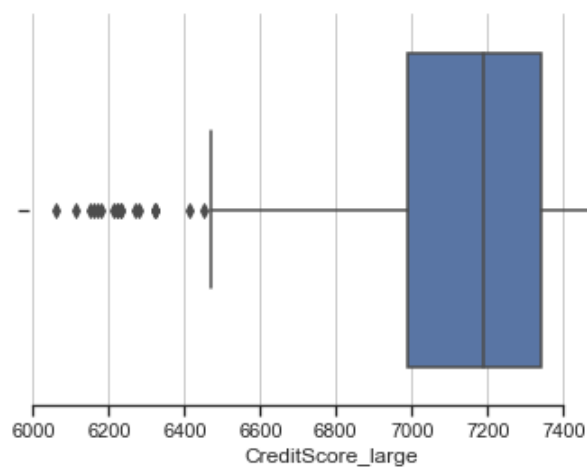
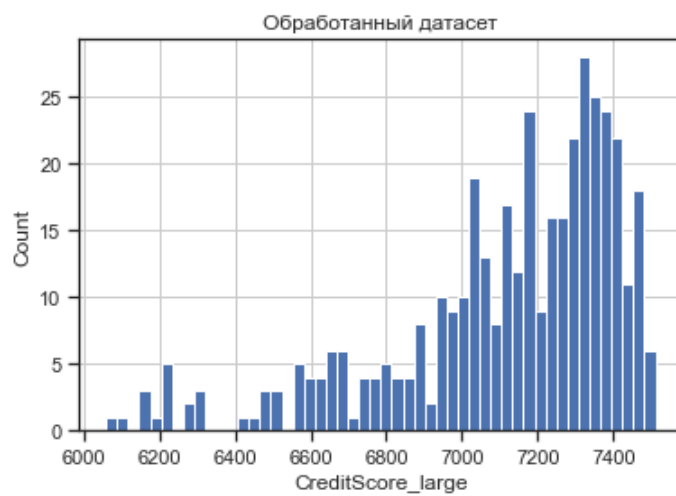
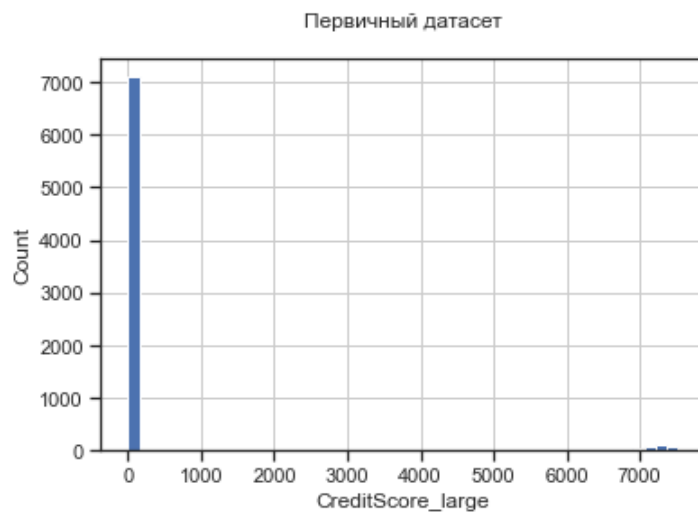
Первичный датасет

Мода датасета: 0.0
Медиана датасета: 0.0
Среднее значение датасета: 379.472
Максимальное значение датасета: 7510.0
Минимальное значение датасета: 0.0

Количество записей в датасете: 7500
Количество записей в датасете < 3000: 7100
Количество записей в датасете > 10000: 0

Обработанный датасет

Мода датасета: 7400.0
Медиана датасета: 7190.0
Среднее значение датасета: 7115.1
Максимальное значение датасета: 7510.0
Минимальное значение датасета: 6060.0



4. Анализ данных

СМ. ВЫШЕ

5. Отбор признаков

```
B [57]: df.columns.tolist()
```

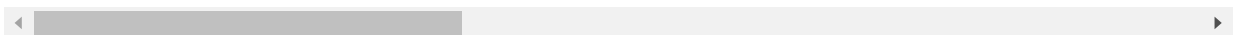
```
Out[57]: ['Home Ownership',
          'Annual Income',
          'Years in current job',
          'Tax Liens',
          'Number of Open Accounts',
          'Years of Credit History',
          'Maximum Open Credit',
          'Number of Credit Problems',
          'Bankruptcies',
          'Purpose',
          'Term',
          'Current Loan Amount',
          'Current Credit Balance',
          'Monthly Debt',
          'Credit Score',
          'Credit Default',
          'Home_Ownership_int',
          'Years_in_current_job_int',
          'Purpose_int',
          'Term_int',
          'Tax_Liens_int',
          'Number_of_Credit_Problems_int',
          'Bankruptcies_int',
          'CreditScore_small',
          'CreditScore_large']
```

```
B [58]: df.head(2)
```

Out[58]:

	Home Ownership	Annual Income	Years in current job	Tax Liens	Number of Open Accounts	Years of Credit History	Maximum Open Credit	Number of Credit Problems	Bankruptcies	Pur
0	Own Home	482087.0	неизвестно	0.0	11.0	26.3	685960.0	1.0	1.0	consolic
1	Own Home	1025487.0	10+ years	0.0	15.0	15.3	1181730.0	0.0	0.0	consolic

2 rows × 25 columns



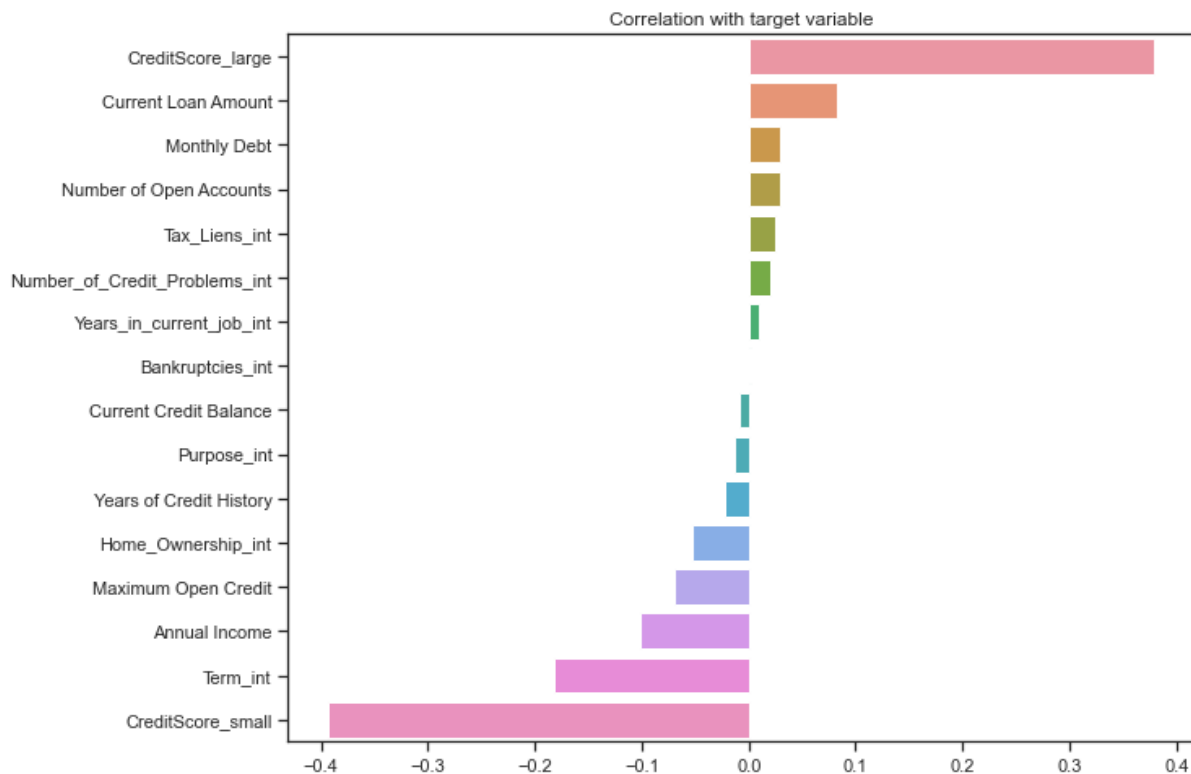
```
B [59]: feature_names = ['#Home Ownership',
                        'Annual Income',
                        # 'Years in current job',
                        # 'Tax Liens',
                        'Number of Open Accounts',
                        'Years of Credit History',
                        'Maximum Open Credit',
                        # 'Number of Credit Problems',
                        # 'Bankruptcies',
                        # 'Purpose',
                        # 'Term',
                        'Current Loan Amount',
                        'Current Credit Balance',
                        'Monthly Debt',
                        # 'Credit Score',
                        # 'Credit Default',
                        'Home_Ownership_int',
                        'Years_in_current_job_int',
                        'Purpose_int',
                        'Term_int',
                        'Tax_Liens_int',
                        'Number_of_Credit_Problems_int',
                        'Bankruptcies_int',
                        'CreditScore_small',
                        'CreditScore_large']

target_name = 'Credit Default'
```

```
B [60]: TARGET_NAME = 'Credit Default'
BASE_FEATURE_NAMES = feature_names
BASE_FEATURE_NAMES
```

```
Out[60]: ['Annual Income',
          'Number of Open Accounts',
          'Years of Credit History',
          'Maximum Open Credit',
          'Current Loan Amount',
          'Current Credit Balance',
          'Monthly Debt',
          'Home_Ownership_int',
          'Years_in_current_job_int',
          'Purpose_int',
          'Term_int',
          'Tax_Liens_int',
          'Number_of_Credit_Problems_int',
          'Bankruptcies_int',
          'CreditScore_small',
          'CreditScore_large']
```

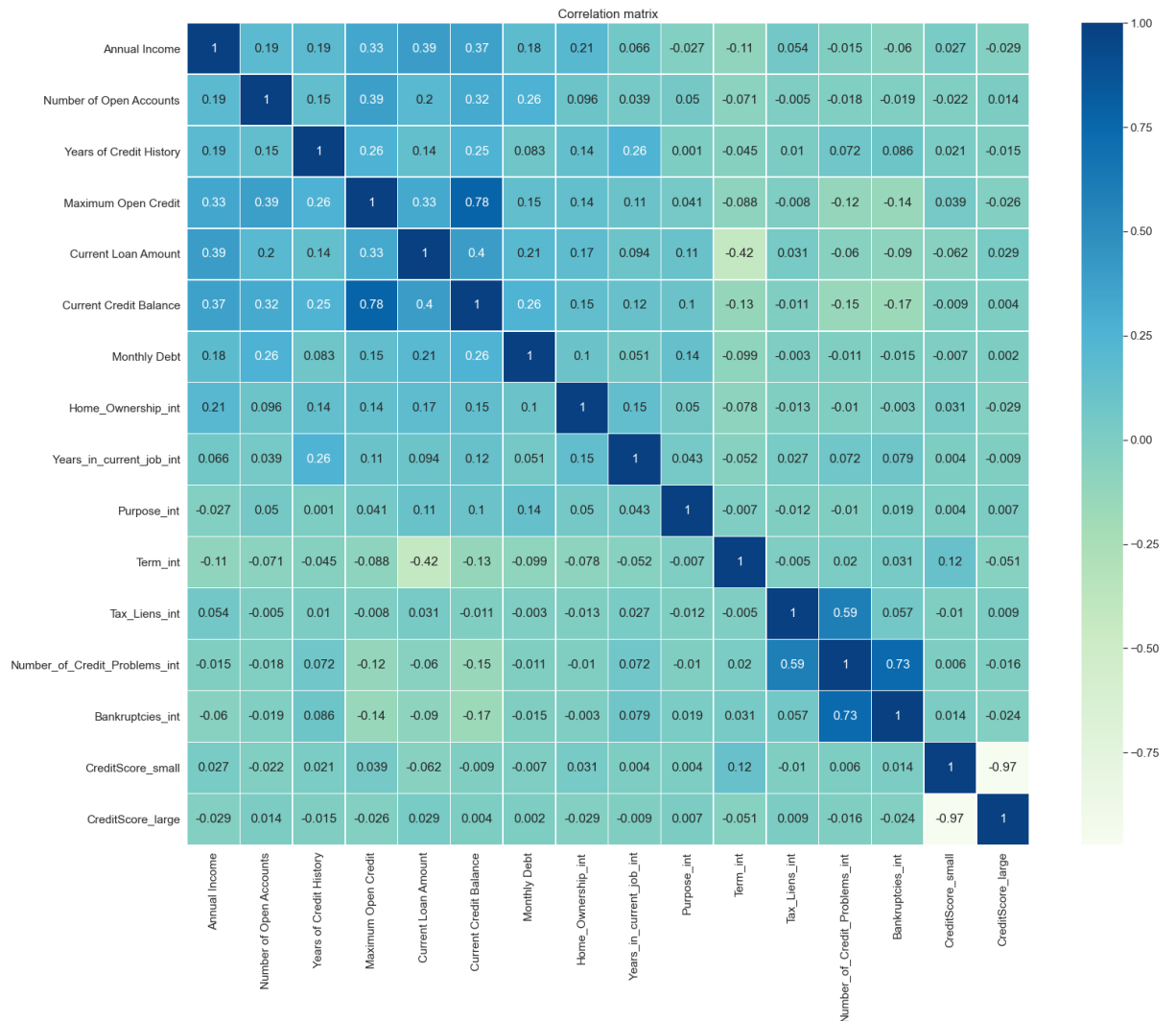
```
B [61]: corr_with_target = df[BASE_FEATURE_NAMES + [TARGET_NAME]].corr().iloc[:,-1].sort_valu
plt.figure(figsize=(10, 8))
sns.barplot(x=corr_with_target.values, y=corr_with_target.index)
plt.title('Correlation with target variable')
plt.show()
```




```
B [62]: plt.figure(figsize = (25,20))

sns.set(font_scale=1.4)
sns.heatmap(df[BASE_FEATURE_NAMES].corr().round(3), annot=True, linewidths=.5, cmap='GnB')

plt.title('Correlation matrix')
plt.show()
```



1. Наблюдается сильная положительная корреляция (**0.78**) между признаками '**Current Loan Amount**' и '**Maximum Open Credit**'. Оба признака сильно влияют на целевой показатель. Оставляем оба признака.
2. Наблюдается сильная положительная корреляция (**0.73**) между признаками '**Bankruptcies_int**' и '**Number_of_Credit_Problems_int**'. При этом '**Bankruptcies_int**' слабо влияет на целевой показатель, данный признак можно исключить из анализа.

3. Наблюдается средняя положительная корреляция (**0.59**) между признаками **'Number_of_Credit_Problems_int'** и **'Tax_Liens_int'**. При этом **'Number_of_Credit_Problems_int'** слабо влияет на целевой показатель. Но **'Number_of_Credit_Problems_int'** сильно связан с признаком **'Bankruptcies_int'**, который мы исключили. Поэтому **'Number_of_Credit_Problems_int'** оставляем.
4. Наблюдается сильная отрицательная корреляция (**-0.97**) между признаками **'CreditScore_small'** и **'CreditScore_large'**. При этом оба признака сильно влияют на целевой показатель. Оставляем оба признака.

Что дальше

1. Нужно подобрать правильную комбинацию модели+список признаков.
2. Сделать список признаков, которые вы точно хотите включить на основании анализа, и опциональный список.
3. И сделать grid search между моделями и признаками.
4. Балансировку классов пока не трогайте.
5. Также можно поиграться с weights.

Опциональный - не входящий в основной комплект и устанавливаемый по желанию заказчика за отдельную плату

В []:

6. Балансировка классов

7. Подбор моделей, получение бейзлана

```
В [136]: from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, ShuffleSplit, cross_val_score, lea
from sklearn.model_selection import StratifiedKFold, GridSearchCV, RandomizedSearchCV
from sklearn.metrics import classification_report, f1_score#, precision_score, recall_sc

from sklearn.linear_model import LogisticRegression # Логистическая регрессия
from sklearn.neighbors import KNeighborsClassifier # k ближайших соседей
from sklearn.tree import DecisionTreeClassifier # Дерево решений
import xgboost as xgb
import lightgbm as lgbm
import catboost as catb
```

```
В [91]: def get_classification_report(y_train_true, y_train_pred, y_test_true, y_test_pred):
    print('TRAIN\n\n' + classification_report(y_train_true, y_train_pred))
    print('TEST\n\n' + classification_report(y_test_true, y_test_pred))
    print('CONFUSION MATRIX\n')
    print(pd.crosstab(y_test_true, y_test_pred))
```

```
В [92]: def evaluate_preds(model, X_train, X_test, y_train, y_test):
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    get_classification_report(y_train, y_train_pred, y_test, y_test_pred)
```

Отбор признаков

```
B [93]: NUM_FEATURE_NAMES = [  
        'Annual Income',  
        'Number of Open Accounts',  
        'Years of Credit History',  
        'Maximum Open Credit',  
        'Current Loan Amount',  
        'Current Credit Balance',  
        'Monthly Debt',  
        # 'Credit Score',  
        # 'Credit Default',  
        'CreditScore_small',  
        'CreditScore_large']  
  
CAT_FEATURE_NAMES = [  
        'Home Ownership',  
        'Years in current job',  
        'Tax Liens',  
        'Number of Credit Problems',  
        'Bankruptcies',  
        'Purpose',  
        'Term']  
  
NEW_FEATURE_NAMES = [  
        'Home_Ownership_int',  
        'Years_in_current_job_int',  
        'Purpose_int',  
        'Term_int',  
        'Tax_Liens_int',  
        'Number_of_Credit_Problems_int',  
        'Bankruptcies_int']  
  
TARGET_NAME = 'Credit Default'  
  
# SELECTED_FEATURE_NAMES = NUM_FEATURE_NAMES + CAT_FEATURE_NAMES + NEW_FEATURE_NAMES  
SELECTED_FEATURE_NAMES = NUM_FEATURE_NAMES + NEW_FEATURE_NAMES
```

Масштабирование данных

```
B [94]: scaler = StandardScaler()  
  
df_norm = df.copy()  
df_norm[NUM_FEATURE_NAMES] = scaler.fit_transform(df_norm[NUM_FEATURE_NAMES])  
  
df = df_norm.copy()
```

Разбиение на train и test

```
B [95]: X = df[SELECTED_FEATURE_NAMES]
        y = df[TARGET_NAME]

        X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                            shuffle=True,
                                                            test_size=0.3,
                                                            random_state=21,
                                                            stratify=y)

        display(y_train.value_counts(normalize=True), y_test.value_counts(normalize=True))

0    0.718286
1    0.281714
Name: Credit Default, dtype: float64

0    0.718222
1    0.281778
Name: Credit Default, dtype: float64
```

Сохранение обучающего и тестового датасетов

```
B [96]: #DATA_ROOT = Path('./data/training_project/')
        DATA_ROOT = './data/training_project/'

        # output
        TRAIN_FULL_PATH = DATA_ROOT + 'training_project_train_full.csv'
        TRAIN_PART_PATH = DATA_ROOT + 'training_project_train_part_b.csv'
        TEST_PART_PATH = DATA_ROOT + 'training_project_test_part.csv'

B [97]: train = pd.concat([X_train, y_train], axis=1)
        test = pd.concat([X_test, y_test], axis=1)

B [98]: df.to_csv(TRAIN_FULL_PATH, index=False, encoding='utf-8')
        train.to_csv(TRAIN_PART_PATH, index=False, encoding='utf-8')
        test.to_csv(TEST_PART_PATH, index=False, encoding='utf-8')
```

Построение и оценка базовых моделей

Логистическая регрессия

```
B [99]: model_lr = LogisticRegression()
model_lr.fit(X_train, y_train)

evaluate_preds(model_lr, X_train, X_test, y_train, y_test)
```

TRAIN

	precision	recall	f1-score	support
0	0.77	0.98	0.86	3771
1	0.87	0.25	0.39	1479
accuracy			0.78	5250
macro avg	0.82	0.62	0.63	5250
weighted avg	0.80	0.78	0.73	5250

TEST

	precision	recall	f1-score	support
0	0.76	0.98	0.86	1616
1	0.85	0.23	0.36	634
accuracy			0.77	2250
macro avg	0.81	0.61	0.61	2250
weighted avg	0.79	0.77	0.72	2250

CONFUSION MATRIX

col_0	0	1
Credit Default		
0	1590	26
1	490	144

к ближайших соседей

```
B [100]: model_knn = KNeighborsClassifier()
model_knn.fit(X_train, y_train)

evaluate_preds(model_knn, X_train, X_test, y_train, y_test)
```

TRAIN

	precision	recall	f1-score	support
0	0.81	0.96	0.88	3771
1	0.80	0.45	0.57	1479
accuracy			0.81	5250
macro avg	0.81	0.70	0.73	5250
weighted avg	0.81	0.81	0.79	5250

TEST

	precision	recall	f1-score	support
0	0.77	0.92	0.84	1616
1	0.61	0.30	0.41	634
accuracy			0.75	2250
macro avg	0.69	0.61	0.62	2250
weighted avg	0.73	0.75	0.72	2250

CONFUSION MATRIX

col_0	0	1
Credit Default		
0	1492	124
1	441	193

Дерево решений

```
B [101]: model_tree = DecisionTreeClassifier(random_state=21,
                                           class_weight={0:1, 1:3.6},
                                           max_depth=100
                                           )
model_tree.fit(X_train, y_train)
evaluate_preds(model_tree, X_train, X_test, y_train, y_test)
```

TRAIN

	precision	recall	f1-score	support
0	1.00	1.00	1.00	3771
1	1.00	1.00	1.00	1479
accuracy			1.00	5250
macro avg	1.00	1.00	1.00	5250
weighted avg	1.00	1.00	1.00	5250

TEST

	precision	recall	f1-score	support
0	0.78	0.78	0.78	1616
1	0.45	0.44	0.45	634
accuracy			0.69	2250
macro avg	0.62	0.61	0.61	2250
weighted avg	0.69	0.69	0.69	2250

CONFUSION MATRIX

col_0	0	1
Credit Default		
0	1268	348
1	352	282

Бустинговые алгоритмы*XGBoost*

```

B [102]: %%time
model_xgb = xgb.XGBClassifier(random_state=21,
#                               n_estimators=100
)
model_xgb.fit(X_train, y_train)

evaluate_preds(model_xgb, X_train, X_test, y_train, y_test)

```

[21:18:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

TRAIN

	precision	recall	f1-score	support
0	0.95	1.00	0.97	3771
1	0.99	0.87	0.93	1479
accuracy			0.96	5250
macro avg	0.97	0.93	0.95	5250
weighted avg	0.96	0.96	0.96	5250

TEST

	precision	recall	f1-score	support
0	0.78	0.90	0.84	1616
1	0.58	0.37	0.45	634
accuracy			0.75	2250
macro avg	0.68	0.63	0.64	2250
weighted avg	0.73	0.75	0.73	2250

CONFUSION MATRIX

col_0	0	1
Credit Default		
0	1447	169
1	402	232

Wall time: 917 ms

LightGBM


```

B [103]: %%time
model_lgbm = lgbl.LGBMClassifier(random_state=21,
                                class_weight={0:1, 1:3.6},
                                # n_estimators=100
                                )
model_lgbm.fit(X_train, y_train)

evaluate_preds(model_lgbm, X_train, X_test, y_train, y_test)

```

TRAIN

	precision	recall	f1-score	support
0	0.99	0.85	0.91	3771
1	0.72	0.98	0.83	1479
accuracy			0.88	5250
macro avg	0.85	0.91	0.87	5250
weighted avg	0.91	0.88	0.89	5250

TEST

	precision	recall	f1-score	support
0	0.82	0.69	0.75	1616
1	0.44	0.62	0.52	634
accuracy			0.67	2250
macro avg	0.63	0.66	0.63	2250
weighted avg	0.72	0.67	0.68	2250

CONFUSION MATRIX

col_0	0	1
Credit Default		
0	1114	502
1	239	395

Wall time: 471 ms

CatBoost

```
B [105]: %%time
model_catb = catb.CatBoostClassifier(silent=True, random_state=21)
model_catb.fit(X_train, y_train)

evaluate_preds(model_catb, X_train, X_test, y_train, y_test)
```

TRAIN

	precision	recall	f1-score	support
0	0.85	0.99	0.92	3771
1	0.97	0.56	0.71	1479
accuracy			0.87	5250
macro avg	0.91	0.78	0.81	5250
weighted avg	0.89	0.87	0.86	5250

TEST

	precision	recall	f1-score	support
0	0.78	0.95	0.86	1616
1	0.72	0.32	0.44	634
accuracy			0.77	2250
macro avg	0.75	0.64	0.65	2250
weighted avg	0.76	0.77	0.74	2250

CONFUSION MATRIX

```
col_0      0    1
Credit Default
0          1536  80
1           431 203
Wall time: 15.4 s
```

```
B [106]: BASE_FEATURE_NAMES
```

```
Out[106]: ['Home_Ownership_int',
'Years_in_current_job_int',
'Purpose_int',
'Term_int',
'Tax_Liens_int',
'Number_of_Credit_Problems_int',
'Bankruptcies_int']
```

```
B [107]: NEW_FEATURE_NAMES
```

```
Out[107]: ['Home_Ownership_int',
'Years_in_current_job_int',
'Purpose_int',
'Term_int',
'Tax_Liens_int',
'Number_of_Credit_Problems_int',
'Bankruptcies_int']
```

```
B [114]: CAT_FEATURE_NAMES
```

```
Out[114]: ['Home_Ownership',
'Years in current job',
'Tax Liens',
'Number of Credit Problems',
'Bankruptcies',
'Purpose',
'Term']
```

B [115]: SELECTED_FEATURE_NAMES

```
Out[115]: ['Annual Income',
           'Number of Open Accounts',
           'Years of Credit History',
           'Maximum Open Credit',
           'Current Loan Amount',
           'Current Credit Balance',
           'Monthly Debt',
           'CreditScore_small',
           'CreditScore_large',
           'Home_Ownership_int',
           'Years_in_current_job_int',
           'Purpose_int',
           'Term_int',
           'Tax_Liens_int',
           'Number_of_Credit_Problems_int',
           'Bankruptcies_int']
```

```
B [119]: # X = df[BASE_FEATURE_NAMES]
X = df[SELECTED_FEATURE_NAMES]
y = df[TARGET_NAME]

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    shuffle=True,
                                                    test_size=0.3,
                                                    random_state=21,
                                                    stratify=y)
```

```
B [120]: %%time
model_catb = catb.CatBoostClassifier(silent=True, random_state=21)
model_catb.fit(X_train, y_train)

evaluate_preds(model_catb, X_train, X_test, y_train, y_test)
```

TRAIN

	precision	recall	f1-score	support
0	0.85	0.99	0.92	3771
1	0.97	0.56	0.71	1479
accuracy			0.87	5250
macro avg	0.91	0.78	0.81	5250
weighted avg	0.89	0.87	0.86	5250

TEST

	precision	recall	f1-score	support
0	0.78	0.95	0.86	1616
1	0.72	0.32	0.44	634
accuracy			0.77	2250
macro avg	0.75	0.64	0.65	2250
weighted avg	0.76	0.77	0.74	2250

CONFUSION MATRIX

col_0	0	1
Credit Default		
0	1536	80
1	431	203

Wall time: 15.6 s

```

B [121]: %%time
model_catb = catb.CatBoostClassifier(silent=True, random_state=21,
                                     cat_features=NEW_FEATURE_NAMES,
                                     # one_hot_max_size=10
                                     )
model_catb.fit(X_train, y_train)

evaluate_preds(model_catb, X_train, X_test, y_train, y_test)

```

TRAIN

	precision	recall	f1-score	support
0	0.82	0.99	0.90	3771
1	0.94	0.45	0.61	1479
accuracy			0.84	5250
macro avg	0.88	0.72	0.75	5250
weighted avg	0.85	0.84	0.81	5250

TEST

	precision	recall	f1-score	support
0	0.78	0.95	0.86	1616
1	0.72	0.31	0.43	634
accuracy			0.77	2250
macro avg	0.75	0.63	0.65	2250
weighted avg	0.76	0.77	0.74	2250

CONFUSION MATRIX

col_0	0	1
Credit Default		
0	1541	75
1	437	197

```

B [123]: disbalance = y_train.value_counts()[0] / y_train.value_counts()[1]
print(y_train.value_counts()[0])
print(y_train.value_counts()[1])
disbalance

```

3771

1479

Out[123]: 2.5496957403651117

```
B [124]: %%time
model_catb = catb.CatBoostClassifier(silent=True, random_state=21,
                                     cat_features=NEW_FEATURE_NAMES,
                                     class_weights=[1, disbalance]
                                     )
model_catb.fit(X_train, y_train)

evaluate_preds(model_catb, X_train, X_test, y_train, y_test)
```

TRAIN

	precision	recall	f1-score	support
0	0.93	0.86	0.89	3771
1	0.69	0.82	0.75	1479
accuracy			0.85	5250
macro avg	0.81	0.84	0.82	5250
weighted avg	0.86	0.85	0.85	5250

TEST

	precision	recall	f1-score	support
0	0.82	0.78	0.80	1616
1	0.50	0.57	0.53	634
accuracy			0.72	2250
macro avg	0.66	0.67	0.67	2250
weighted avg	0.73	0.72	0.72	2250

CONFUSION MATRIX

col_0	0	1
Credit Default		
0	1255	361
1	271	363

```

B [127]: %%time
model_catb = catb.CatBoostClassifier(silent=True, random_state=21,
                                     class_weights=[1, disbalance],
                                     eval_metric='F1',
                                     cat_features=NEW_FEATURE_NAMES,
                                     early_stopping_rounds=20,
                                     use_best_model=True,
                                     custom_metric=['Precision', 'Recall'])
model_catb.fit(X_train, y_train, plot=True, eval_set=(X_test, y_test))

```

☒ --- Learn ☒ — Eval

☒ catboost_info ~34s 499ms 2s 358ms

--- learn — test

curr --- 0.6794894... — 0.6252523... 63

best 0.6289835... 43

F1 Logloss Precision:use_weights=true

Precision:use_weights=false Recall:use_weights=true

Recall:use_weights=false

☐ Click Mode ☐ Logarithm

☐ Smooth 0



Wall time: 2.87 s

Out[127]: <catboost.core.CatBoostClassifier at 0x39deff7c10>

```

B [129]: model_catb.best_score_

```

```

Out[129]: {'learn': {'Recall:use_weights=false': 0.6166328600405679,
                    'Logloss': 0.5434115428867086,
                    'F1': 0.6795887550331505,
                    'Precision:use_weights=false': 0.5998142989786444,
                    'Precision:use_weights=true': 0.7925993522034408,
                    'Recall:use_weights=true': 0.6166328600405679},
          'validation': {'Recall:use_weights=false': 0.5504731861198738,
                        'Logloss': 0.5703927367497929,
                        'F1': 0.628983519474006,
                        'Precision:use_weights=false': 0.5668103448275862,
                        'Precision:use_weights=true': 0.7693816844386225,
                        'Recall:use_weights=true': 0.5504731861198738}}

```

```
B [130]: evaluate_preds(model_catb, X_train, X_test, y_train, y_test)
```

TRAIN

	precision	recall	f1-score	support
0	0.83	0.81	0.82	3771
1	0.55	0.57	0.56	1479
accuracy			0.75	5250
macro avg	0.69	0.69	0.69	5250
weighted avg	0.75	0.75	0.75	5250

TEST

	precision	recall	f1-score	support
0	0.82	0.81	0.81	1616
1	0.53	0.55	0.54	634
accuracy			0.73	2250
macro avg	0.67	0.68	0.67	2250
weighted avg	0.74	0.73	0.74	2250

CONFUSION MATRIX

col_0	0	1
Credit Default		
0	1304	312
1	287	347

Выбор лучшей модели и подбор гиперпараметров

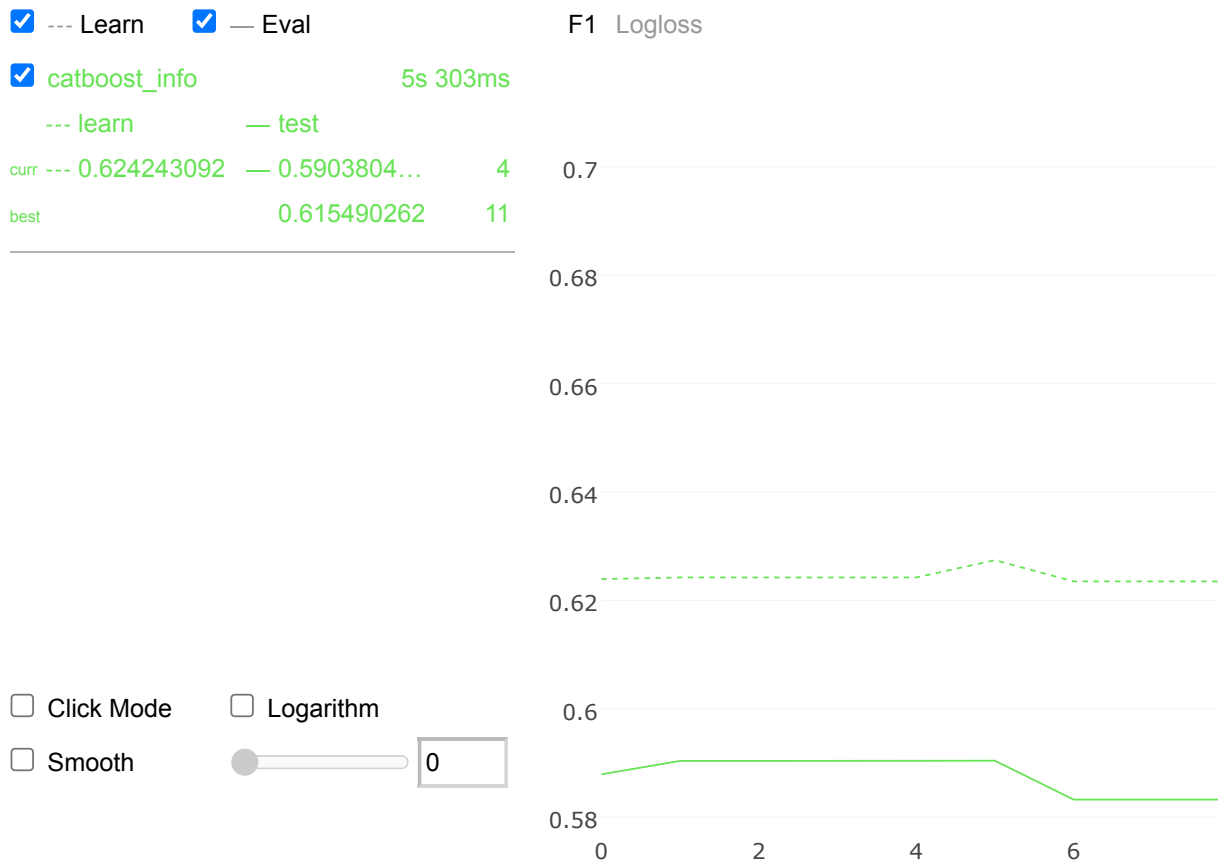
```
B [131]: frozen_params = {
    'class_weights':[1, disbalance],
    'silent':True,
    'random_state':21,
    'cat_features':NEW_FEATURE_NAMES,
    'eval_metric':'F1',
    'early_stopping_rounds':20
}
model_catb = catb.CatBoostClassifier(**frozen_params)
```

Подбор гиперпараметров

```
B [132]: params = {'iterations':[50, 200, 500, 700, 1500],
    'max_depth':[3, 5, 7]}
```

```
B [137]: cv = StratifiedKFold(n_splits=3, random_state=21, shuffle=True)
```

B [138]: `grid_search = model_catb.grid_search(params, X_train, y_train, cv=cv, stratified=True, p`



Stopped by overfitting detector (20 iterations wait)

bestTest = 0.5879091518
bestIteration = 3

0: loss: 0.5879092 best: 0.5879092 (0) total: 518ms remaining: 7.25s
Stopped by overfitting detector (20 iterations wait)

bestTest = 0.5903804662
bestIteration = 38

1: loss: 0.5903805 best: 0.5903805 (1) total: 1.79s remaining: 11.6s
Stopped by overfitting detector (20 iterations wait)

bestTest = 0.5903804662
bestIteration = 38

2: loss: 0.5903805 best: 0.5903805 (1) total: 3.06s remaining: 12.2s
Stopped by overfitting detector (20 iterations wait)

bestTest = 0.5903804662
bestIteration = 38

3: loss: 0.5903805 best: 0.5903805 (1) total: 4.2s remaining: 11.6s
Stopped by overfitting detector (20 iterations wait)

bestTest = 0.5903804662
bestIteration = 38

4: loss: 0.5903805 best: 0.5903805 (1) total: 5.3s remaining: 10.6s
Stopped by overfitting detector (20 iterations wait)

bestTest = 0.5904189252
bestIteration = 15


```
5:      loss: 0.5904189 best: 0.5904189 (5)      total: 6.2s      remaining: 9.3s
Stopped by overfitting detector (20 iterations wait)

bestTest = 0.5832415955
bestIteration = 5

6:      loss: 0.5832416 best: 0.5904189 (5)      total: 7s      remaining: 8s
Stopped by overfitting detector (20 iterations wait)

bestTest = 0.5832415955
bestIteration = 5

7:      loss: 0.5832416 best: 0.5904189 (5)      total: 7.84s      remaining: 6.86s
Stopped by overfitting detector (20 iterations wait)

bestTest = 0.5832415955
bestIteration = 5

8:      loss: 0.5832416 best: 0.5904189 (5)      total: 8.66s      remaining: 5.77s
Stopped by overfitting detector (20 iterations wait)

bestTest = 0.5832415955
bestIteration = 5

9:      loss: 0.5832416 best: 0.5904189 (5)      total: 9.56s      remaining: 4.78s
Stopped by overfitting detector (20 iterations wait)

bestTest = 0.598388377
bestIteration = 12

10:     loss: 0.5983884 best: 0.5983884 (10)     total: 10.8s      remaining: 3.92s
Stopped by overfitting detector (20 iterations wait)

bestTest = 0.615490262
bestIteration = 90

11:     loss: 0.6154903 best: 0.6154903 (11)     total: 15.6s      remaining: 3.91s
Stopped by overfitting detector (20 iterations wait)

bestTest = 0.615490262
bestIteration = 90

12:     loss: 0.6154903 best: 0.6154903 (11)     total: 20.7s      remaining: 3.18s
Stopped by overfitting detector (20 iterations wait)

bestTest = 0.615490262
bestIteration = 90

13:     loss: 0.6154903 best: 0.6154903 (11)     total: 25.6s      remaining: 1.83s
Stopped by overfitting detector (20 iterations wait)

bestTest = 0.615490262
bestIteration = 90

14:     loss: 0.6154903 best: 0.6154903 (11)     total: 29.3s      remaining: 0us
Estimating final quality...
Stopped by overfitting detector (20 iterations wait)
```

```
B [139]: grid_search
```

```
Out[139]: {'params': {'depth': 7, 'iterations': 200},
          'cv_results': defaultdict(list,
          {'iterations': [0,
          1,
          2,
          3,
          4,
          5,
          6,
          7,
          8,
          9,
          10,
          11,
          12,
          13,
          14,
          15,
          16,
          17,
          18,
          19,
          20,
          21,
          22,
          23,
          24,
          25,
          26,
          27,
          28,
          29,
          30,
          31],
          'test-F1-mean': [0.565053045984036,
          0.6080829119531548,
          0.5936727539312572,
          0.5958168544858995,
          0.6058812917142293,
          0.6069622556015847,
          0.6023345377899423,
          0.6099063416390396,
          0.6130438990024384,
          0.6101740067633473,
          0.6128800101227315,
          0.6152608466731703,
          0.6112307110498217,
          0.6115575749812278,
          0.6119990466740425,
          0.6097561483993955,
          0.6122418297950469,
          0.6097898241519241,
          0.6105649484392268,
          0.6092193757391043,
          0.6116165729514874,
          0.6133240276587211,
          0.614609093257558,
          0.612358796370509,
          0.6105985545864794,
          0.6112965343700415,
          0.6127653134795553,
          0.6090663989661959,
          0.6106066143307608,
          0.6140507037730952,
          0.613419478144733,
```

```
0.6132130313990846],  
'test-F1-std': [0.023476258740599245,  
0.03377387500287111,  
0.01856956965340093,  
0.020147515454626504,  
0.026965626642379264,  
0.03096549735793214,  
0.033326306326742265,  
0.02583324170210395,  
0.02580336371422832,  
0.023394943526516467,  
0.020108116520056216,  
0.020702132180351985,  
0.02178499219921111,  
0.021069826428625094,  
0.022767129461271962,  
0.02603226482707921,  
0.02898648369473419,  
0.029707090050317218,  
0.029018927290094858,  
0.02882316577627969,  
0.027870526174356383,  
0.024222247742537388,  
0.022523038713453354,  
0.023592585353208906,  
0.022363276290686136,  
0.02575178732995261,  
0.02304604854825822,  
0.019746705541835337,  
0.021065015899160888,  
0.021903034352209663,  
0.021394432545226806,  
0.01867372711781397],  
'train-F1-mean': [0.578645499965734,  
0.6190984982686528,  
0.6099429295137647,  
0.6084349776200567,  
0.6160334447577515,  
0.6245383585626243,  
0.6196172429739043,  
0.6258639185734977,  
0.6227117437106607,  
0.626322895448034,  
0.623383122776298,  
0.6253775827366487,  
0.6257618228431775,  
0.6252570776160774,  
0.625226966280224,  
0.6259204200043444,  
0.6245421872410716,  
0.624116122902031,  
0.6247006777700989,  
0.6243501409826994,  
0.6249859705911166,  
0.6256626953210972,  
0.6277003097696724,  
0.6292103298288839,  
0.6318280798526182,  
0.6339110698648919,  
0.6328115510777584,  
0.633145948792669,  
0.6333123897440943,  
0.6352596044504577,  
0.6377911771485524,  
0.6382400229967365],  
'train-F1-std': [0.017969901162306092,  
0.017213462243283965,  
0.012482783908194938,
```

```
0.01266945278601381,  
0.008164781202077483,  
0.00820202976151169,  
0.005015781707363664,  
0.010298106080064484,  
0.010959931567011402,  
0.011152990060124376,  
0.013695223796266726,  
0.012816742940679436,  
0.01419575323726824,  
0.015979748575214333,  
0.017532111297411414,  
0.013010302633227028,  
0.012827899302864279,  
0.01294482556087138,  
0.013718415462337433,  
0.012815817619869075,  
0.014061909439719588,  
0.016667183617640826,  
0.01633564925493691,  
0.012163966106773068,  
0.01208370974771544,  
0.01376550581843644,  
0.015005054928088435,  
0.017130437050245726,  
0.01801582295169821,  
0.019213437581824835,  
0.021497757902826337,  
0.021367545376171592],  
'test-Logloss-mean': [0.6852649019630591,  
0.6779574472011239,  
0.6709820473959994,  
0.664737537853905,  
0.6608301200158712,  
0.6562256947827987,  
0.6517172664913082,  
0.6442032785009998,  
0.6389425452167319,  
0.6324597262374336,  
0.6290107673085733,  
0.626783912690224,  
0.6228604186668482,  
0.6189750480686318,  
0.6164789186614502,  
0.6138157176763747,  
0.6111802966945395,  
0.6090816590769526,  
0.6064161977919101,  
0.6043813123997842,  
0.6032769458841586,  
0.601052133738582,  
0.5996271132438779,  
0.5978890434145822,  
0.5963986298947619,  
0.5956532422374355,  
0.5947798316823095,  
0.5930381282244779,  
0.5919635251192393,  
0.5911023719045261,  
0.5903502358618496,  
0.5885809648724764],  
'test-Logloss-std': [0.0004170549728106711,  
0.0011228658889144466,  
0.0016051252583686607,  
0.0026850755992455827,  
0.0010954985305193531,  
0.0002561606155349594,  
0.0009477468304150264,
```

```
0.000746861769473496,  
0.004056353518273976,  
0.0020154366882104667,  
0.002933357905378315,  
0.0040402000404220935,  
0.004052036168102251,  
0.0022905931148248762,  
0.003126735168011619,  
0.003043607852225985,  
0.003151110956313971,  
0.0024584417130585905,  
0.003352955044669737,  
0.004529056847593291,  
0.004386275421607698,  
0.004557924807834787,  
0.004398080848257322,  
0.0039951024812173086,  
0.004395015555555645,  
0.004897264565659418,  
0.005065411598963004,  
0.0047211173946031994,  
0.004718288494056993,  
0.004714129701029383,  
0.004475788348561851,  
0.0037393006279722533],  
'train-Logloss-mean': [0.6845016664632007,  
0.6767428041840979,  
0.6692235172824995,  
0.662755234443485,  
0.658436171373721,  
0.6531374110628122,  
0.6483900181378636,  
0.6407694017859793,  
0.6352887189973718,  
0.6283175913455117,  
0.6244809146475573,  
0.6220248577763613,  
0.617704335533909,  
0.6136188182677164,  
0.6108463368193138,  
0.6078593525865175,  
0.6049496275695745,  
0.6025171166679708,  
0.5994996640095068,  
0.5970115394475359,  
0.5955434997516665,  
0.5930450472823389,  
0.5908725967121279,  
0.5885613802790611,  
0.5866075287555982,  
0.5854726464847904,  
0.5841569971733106,  
0.5821155795986849,  
0.5806253850343793,  
0.5794033462682391,  
0.5781343771371225,  
0.5760692094809224],  
'train-Logloss-std': [0.0008710668082168189,  
0.0013333614435391908,  
0.0015455229911579135,  
0.0012774974714599532,  
0.003672029634739806,  
0.00448825537876399,  
0.004826717371468374,  
0.006234645103790485,  
0.007757550348727876,  
0.006795269478806936,  
0.00656732843122855,
```

```

0.007768981389233945,
0.004901673113654485,
0.00564073468853716,
0.005007708802749501,
0.0057608740491216235,
0.006248944292200985,
0.006452478289729805,
0.005939996831112383,
0.005507300064589872,
0.006526456115558345,
0.006542336087204999,
0.006630939336315184,
0.006575038194864474,
0.0064635313942695265,
0.006203626269314894,
0.006245762291023964,
0.006679042346515379,
0.006992977024171358,
0.006672354262273469,
0.007385136131927896,
0.008099876781221092]]}}

```

```
B [140]: pd.DataFrame(grid_search['cv_results']).sort_values('test-F1-mean', ascending=False).head(10)
```

Out[140]:

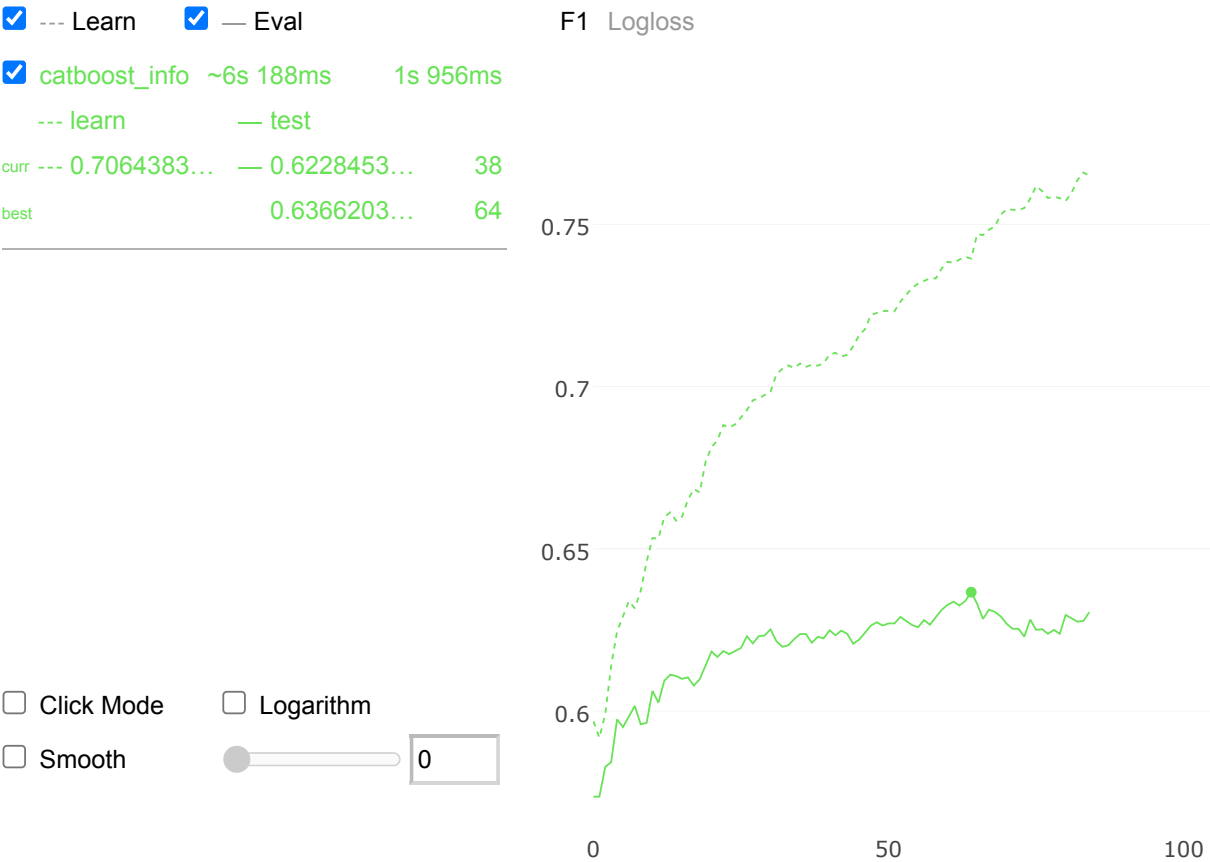
	iterations	test-F1-mean	test-F1-std	train-F1-mean	train-F1-std	test-Logloss-mean	test-Logloss-std	train-Logloss-mean	train-Logloss-std
11	11	0.615261	0.020702	0.625378	0.012817	0.626784	0.004040	0.622025	0.007769
22	22	0.614609	0.022523	0.627700	0.016336	0.599627	0.004398	0.590873	0.006631
29	29	0.614051	0.021903	0.635260	0.019213	0.591102	0.004714	0.579403	0.006672
30	30	0.613419	0.021394	0.637791	0.021498	0.590350	0.004476	0.578134	0.007385
21	21	0.613324	0.024222	0.625663	0.016667	0.601052	0.004558	0.593045	0.006542

Обучение и оценка финальной модели

```
B [141]: %%time

final_model = catb.CatBoostClassifier(**frozen_params, iterations=200, max_depth=7)
final_model.fit(X_train, y_train, plot=True, eval_set=(X_test, y_test))

evaluate_preds(final_model, X_train, X_test, y_train, y_test)
```



TRAIN

	precision	recall	f1-score	support
0	0.86	0.81	0.83	3771
1	0.57	0.66	0.61	1479
accuracy			0.76	5250
macro avg	0.71	0.73	0.72	5250
weighted avg	0.78	0.76	0.77	5250

TEST

	precision	recall	f1-score	support
0	0.82	0.76	0.79	1616
1	0.49	0.58	0.53	634
accuracy			0.71	2250
macro avg	0.66	0.67	0.66	2250
weighted avg	0.73	0.71	0.72	2250

CONFUSION MATRIX

col_0	0	1
Credit Default		
0	1234	382
1	268	366
Wall time: 5.58 s		

В []:

В []:

1. Нужно подобрать правильную комбинацию модели+список признаков.
2. Сделать список признаков, которые вы точно хотите включить на основании анализа, и опциональный список.
3. И сделать grid search между моделями и признаками.
4. Балансировку классов пока не трогайте.
5. Также можно поиграться с weights.

Опциональный - не входящий в основной комплект и устанавливаемый по желанию заказчика за отдельную плату

8. Выбор наилучшей модели, настройка гиперпараметров
9. Проверка качества, борьба с переобучением
10. Интерпретация результатов

Прогнозирование на тестовом датасете

1. Выполнить для тестового датасета те же этапы обработки и построения признаков
2. Спрогнозировать целевую переменную, используя модель, построенную на обучающем датасете
3. Прогнозы должны быть для всех примеров из тестового датасета (для всех строк)
4. Соблюдать исходный порядок примеров из тестового датасета

В []:

В []: