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-dannyye/3-4-diaghramma-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 постронияния 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

В [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

В [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

В [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	Purpose	Term	Current Loan Amount	Current Credit Balance	Monthly Debt	Credit Score
0	Own Home	482087.0	NaN	0.0	11.0	26.3	685960.0	1.0	NaN	1.0	debt consolidation	Short Term	99999999.0	47386.0	7914.0	749.0
1	Own Home	1025487.0	10+ years	0.0	15.0	15.3	1181730.0	0.0	NaN	0.0	debt consolidation	Long Term	264968.0	394972.0	18373.0	737.0
2	Home Mortgage	751412.0	8 years	0.0	11.0	35.0	1182434.0	0.0	NaN	0.0	debt consolidation	Short Term	99999999.0	308389.0	13651.0	742.0
3	Own Home	805068.0	6 years	0.0	8.0	22.5	147400.0	1.0	NaN	1.0	debt consolidation	Short Term	121396.0	95855.0	11338.0	694.0
4	Rent	776264.0	8 years	0.0	13.0	13.6	385836.0	1.0	NaN	0.0	debt consolidation	Short Term	125840.0	93309.0	7180.0	719.0

В [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	Purpose	Term	Current Loan Amount	Current Credit Balance	Monthly Debt	Credit Score
0	Rent	NaN	4 years	0.0	9.0	12.5	220968.0	0.0	70.0	0.0	debt consolidation	Short Term	162470.0	105906.0	6813.0	NaN
1	Rent	231838.0	1 year	0.0	6.0	32.7	55946.0	0.0	8.0	0.0	educational expenses	Short Term	78298.0	46037.0	2318.0	699.0
2	Home Mortgage	1152540.0	3 years	0.0	10.0	13.7	204600.0	0.0	NaN	0.0	debt consolidation	Short Term	200178.0	146490.0	18729.0	7260.0
3	Home Mortgage	1220313.0	10+ years	0.0	16.0	17.0	456302.0	0.0	70.0	0.0	debt consolidation	Short Term	217382.0	213199.0	27559.0	739.0
4	Home Mortgage	2340952.0	6 years	0.0	11.0	23.6	1207272.0	0.0	NaN	0.0	debt consolidation	Long Term	777634.0	425391.0	42605.0	706.0

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

Out[6]:
```
(7500, 17)
```

В [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
```

В [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
```

В [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
```

В [10]:
```
#df_train.dtypes
```

representing null/NaN values using seaborn plotting techniques

representing using heatmap()

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

...

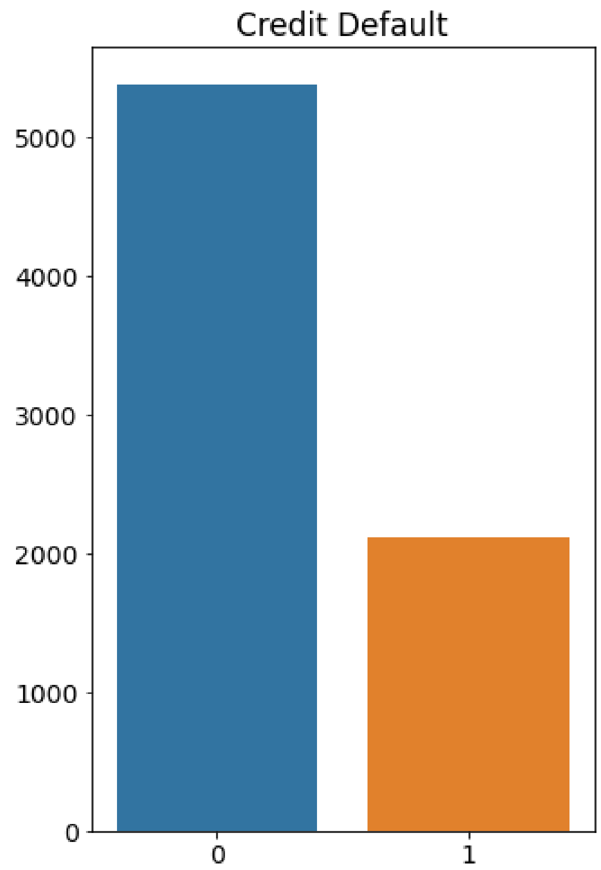
1. Обзор данных (Обзор обучающего датасета)

Обзор целевой переменной

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

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

В [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

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

В [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

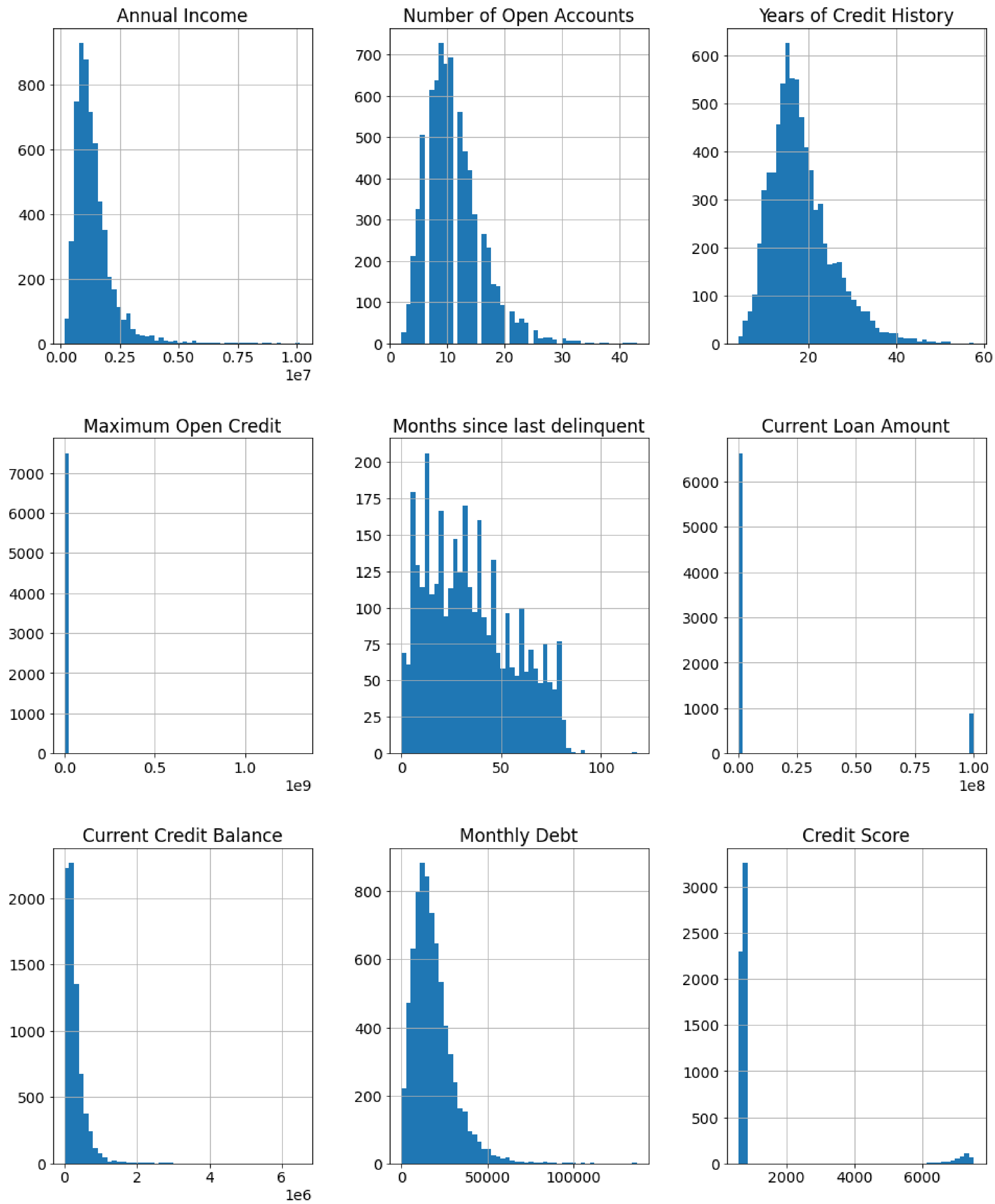
В [16]:
```
df_train.describe().T  # Анализ количественные признаки
```

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

В [17]:
```
df_num_features = df_train.select_dtypes(include=['float32', 'float64', 'int8', 'int16', 'int32'])

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.

В [18]:
```
def plot_feature(feature_name, df_train, feature_value_max, feature_value_min, data_type, count_sort=0):
    '''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(ascending=False))
    #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.shape[0]))
        df = df_train.loc[(df_train[feature_name] > feature_value_max)]
        print('Количество записей в датасете > {0}: {1}'.format(feature_value_max, df.shape[0]))
        print('_' * 50)

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

        # 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)

В [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)
```

...

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

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

В [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)

В [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)
```

...

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)
```

...

7. Maximum Open Credit - наибольший открытый кредит

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

В [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

В [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)
```

...

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

10. Bankruptcies - банкротства (категориальные данные)

В [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)

В [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)

В [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, ..., 1*10^8]
```

...

```
**14. Current Credit Balance** - текущий кредитный баланс
```

В [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)
```

...

15. Monthly Debt - ежемесячный долг

В [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)
```

...

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 - [6500, ..., 7500]
# Считаем выбросами значения 'Monthly Debt' < 585
```

...

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 * 10^7], 2 - [85 * 10^7, ..., 1 * 10^8]
- Проверить kooреляцию 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 kooреляцию с Current Loan Amount - текущая сумма кредита

В [ ]:

Анализ признакового пространства¶

Корреляция с базовыми признаками

В [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']
```

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

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

В [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, cmap='GnBu')

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**).

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

В [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
```

В [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

В [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' + '*' * 100 + '\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
5.0       2
6.0       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 * 10^7], 2 - [85 * 10^7, ..., 1 * 10^8]
- Проверить кореляцию 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 кореляцию с Current Loan Amount - текущая сумма кредита

3. Обработка пропусков

В [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	Purpose	Term	Current Loan Amount	Current Credit Balance	Monthly Debt	Credit Score	Cre Defa
0	False	False	True	False	False	False	False	False	True	False	False	False	False	False	False	False	Fa
1	False	False	False	False	False	False	False	False	True	False	False	False	False	False	False	False	Fa
2	False	False	False	False	False	False	False	False	True	False	False	False	False	False	False	False	Fa
3	False	False	False	False	False	False	False	False	True	False	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	False	False	True	False	False	False	False	False	False	False	Fa
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
7495	False	False	False	False	False	False	False	False	True	False	False	False	False	False	False	False	Fa
7496	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
7497	False	False	False	False	False	False	False	False	True	False	False	False	False	False	False	False	Fa
7498	False	True	True	False	False	False	False	False	True	False	False	False	False	False	False	True	Fa
7499	False	True	False	False	False	False	False	False	True	False	False	False	False	False	False	True	Fa

7500 rows × 17 columns

В [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"**

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

Years in current job - количество лет на текущем месте работы

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

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

В [46]:
```
cat_colname = 'Years in current job'
df_train[cat_colname] = df_train[cat_colname].replace(to_replace = np.nan, value = 'неизвестно')
```

В [47]:
```
print(str(cat_colname) + '\n\n' + str(df_train[cat_colname].value_counts()) + '\n' + '*' * 100 + '\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
****************************************************************************************************
```

В [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

В [49]:
```
# Считаем выбросами Годовой доход 'Annual Income' > 4 000 000 (91 значения) и Annual Income < 165000
# Считаем выбросами Количество лет кредитной истории 'Years of Credit History' > 40 (83 значения)
# Считаем выбросами Наибольший открытый кредит 'Maximum Open Credit' > 4 000 000 (64 значений)
# и 'Maximum Open Credit' < 50 000 (125 значений)
# Считаем выбросами Количество месяцев с последней просрочки платежа Months since last delinquent > 83 (5 значений)
# Считаем выбросами Текущий кредитный баланс 'Current Credit Balance' > 1300000 (106 значений)
# 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 выбрасов"""

        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, ['Current Loan Amount']]
        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']].fillna(self.medians)

        # 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 = 'неизвестно')

        # 2. Выбросы (outliers)

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

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

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

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

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

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

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

        # 3. Обработка категорий
        colname = 'Bankruptcies'
        df[colname] = df[colname].replace(to_replace = 'nan', value = '0.0')
        # (создание дами-переменных)
        #df = pd.concat([df, pd.get_dummies(df['Tax Liens'], prefix='Tax Liens', dtype='int8')], axis=1)
        #df = pd.concat([df, pd.get_dummies(df['Number of Credit Problems'], prefix='Number of Credit Problems', dtype='int8')], axis=1,
        #df = pd.concat([df, pd.get_dummies(df['Bankruptcies'], prefix='Bankruptcies', dtype='int8')], axis=1)

        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_int']
        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', 'Term_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

В [51]:
```
data_pl = DataPipeLine()

# тренировочные данные
data_pl.fit(df_train)

df = data_pl.transform(df_train)
```

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

В [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
```

В [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' + '*' * 100 + '\n')
```

...

В [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)
```

...

В [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)
```

...

4. Анализ данных

см. выше

5. Отбор признаков

В [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']
```

В [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	Purpose	...	Credit Default	Home_Ownership_int	Years_in_current_jc
0	Own Home	482087.0	неизвестно	0.0	11.0	26.3	685960.0	1.0	1.0	debt consolidation	...	0	1	
1	Own Home	1025487.0	10+ years	0.0	15.0	15.3	1181730.0	0.0	0.0	debt consolidation	...	1	1	

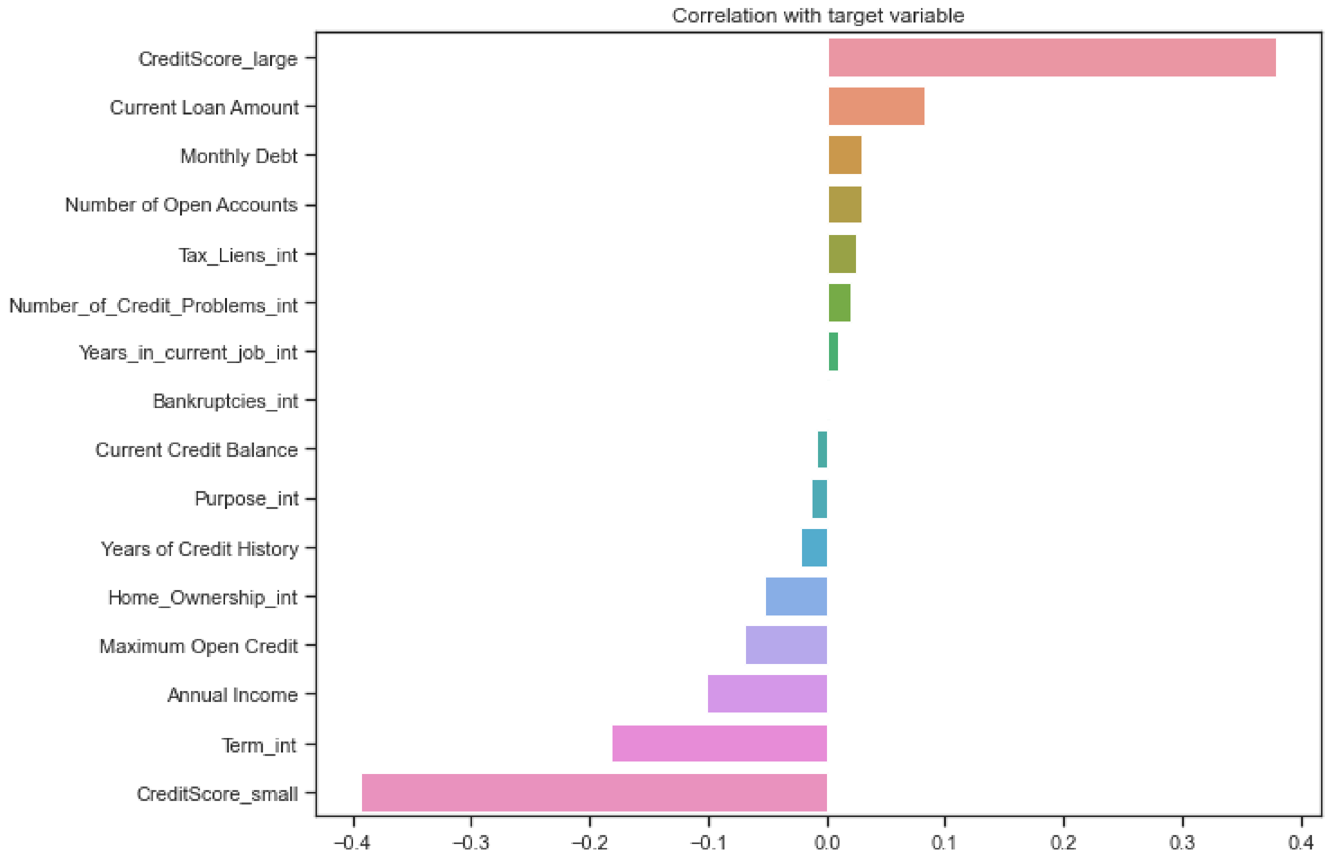
2 rows × 25 columns

В [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'
```

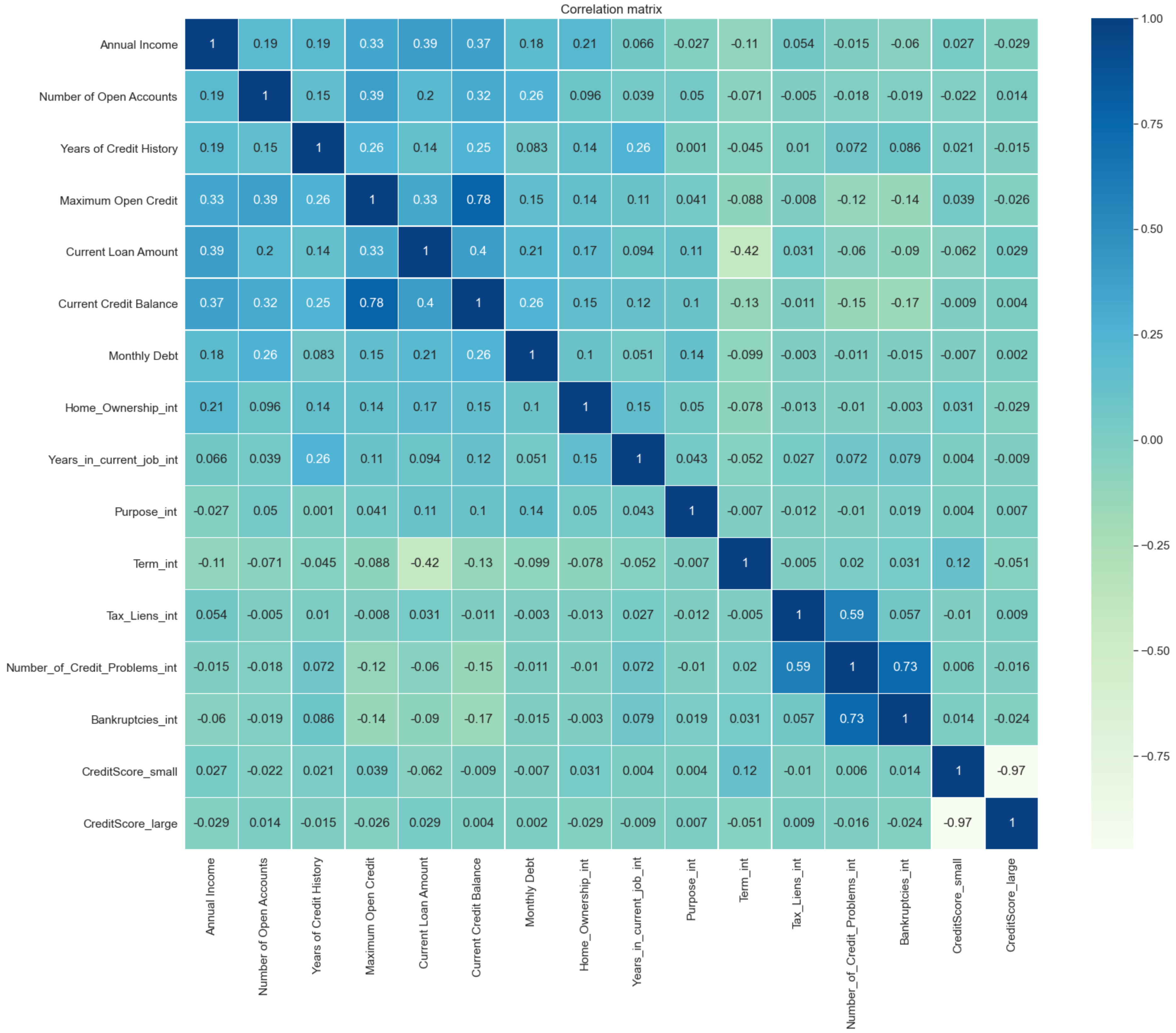
В [60]:
```
TARGET_NAME = 'Credit Default'
BASE_FEATURE_NAMES = feature_names
BASE_FEATURE_NAMES
```

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']
```

В [61]:
```
corr_with_target = df[BASE_FEATURE_NAMES + [TARGET_NAME]].corr().iloc[:-1, -1].sort_values(ascending=False)

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()
```



В [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='GnBu')

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.

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

В [ ]:
```

```

6. Балансировка классов

7. Подбор моделей, получение бейзлана

В [63]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, ShuffleSplit, cross_val_score, learning_curve
from sklearn.model_selection import StratifiedKFold, GridSearchCV, RandomizedSearchCV
from sklearn.metrics import classification_report, f1_score#, precision_score, recall_score

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 [64]:
```
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('CONFUSION MATRIX\n')
    print(pd.crosstab(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 [65]:
```
sklearn.metrics.classification_report()
```

```
Object `sklearn.metrics.classification_report()` not found.
```

В [66]:
```
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

В [67]:
```
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
```

Масштабрование данных

В [68]:
```
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

В [69]:
```
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

В [70]:
```
#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'
```

В [71]:
```
train = pd.concat([X_train, y_train], axis=1)
test = pd.concat([X_test, y_test], axis=1)
```

В [72]:
```
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

В [73]:
```
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      0.99      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

CONFUSION MATRIX

col_0              0    1
Credit Default
0               3715   56
1               1110  369
TEST

              precision    recall  f1-score   support

           0       0.77      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                488  146
```

Метод опорных векторов

В [74]:
```
import sklearn.svm as svm
import matplotlib.pyplot as plt

import numpy as np
import pandas as pd
import sklearn
import sklearn.datasets as ds
import sklearn.model_selection as ms
import sklearn.svm as svm
import matplotlib.pyplot as plt
%matplotlib inline
```

В [75]:
```
model_knn = svm.LinearSVC()
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       0.80      0.89      0.84      3771
           1       0.60      0.43      0.50      1479

    accuracy                           0.76      5250
   macro avg       0.70      0.66      0.67      5250
weighted avg       0.74      0.76      0.74      5250

CONFUSION MATRIX

col_0              0    1
Credit Default
0               3338  433
1                839  640
TEST

              precision    recall  f1-score   support

           0       0.79      0.88      0.83      1616
           1       0.56      0.40      0.47       634

    accuracy                           0.74      2250
   macro avg       0.67      0.64      0.65      2250
weighted avg       0.72      0.74      0.73      2250

CONFUSION MATRIX

col_0              0    1
Credit Default
0               1414  202
1                378  256
```

В [76]:
```
# We train the classifier.
est = svm.LinearSVC()
#est.fit(X, y)
est.fit(X_train, y_train)
```

Out[76]:
```
LinearSVC()
```

В [77]:
```
# We generate a grid in the square [-3,3 ]^2.
xx, yy = np.meshgrid(np.linspace(-3, 3, 500),
                     np.linspace(-3, 3, 500))

# This function takes a SVM estimator as input.


def plot_decision_function(est, title):
    # We evaluate the decision function on the grid.
    Z = est.decision_function(np.c_[xx.ravel(),
                                    yy.ravel()])
    Z = Z.reshape(xx.shape)
    cmap = plt.cm.Blues

    # We display the decision function on the grid.
    fig, ax = plt.subplots(1, 1, figsize=(5, 5))
    ax.imshow(Z,
              extent=(xx.min(), xx.max(),
                      yy.min(), yy.max()),
              aspect='auto',
              origin='lower',
              cmap=cmap)

    # We display the boundaries.
    ax.contour(xx, yy, Z, levels=[0],
               linewidths=2,
               colors='k')

    # We display the points with their true labels.
    ax.scatter(X[:, 0], X[:, 1],
               s=50, c=.5 + .5 * y,
               edgecolors='k',
               lw=1, cmap=cmap,
               vmin=0, vmax=1)
    ax.axhline(0, color='k', ls='--')
    ax.axvline(0, color='k', ls='--')
    ax.axis([-3, 3, -3, 3])
    ax.set_axis_off()
    ax.set_title(title)
```

В [78]:
```
#ax = plot_decision_function(est, "Linearly separable, linear SVC")
```

k ближайших соседей

В [79]:
```
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      0.95      0.88      3771
           1       0.79      0.45      0.57      1479

    accuracy                           0.81      5250
   macro avg       0.80      0.70      0.72      5250
weighted avg       0.81      0.81      0.79      5250

CONFUSION MATRIX

col_0              0    1
Credit Default
0               3590  181
1                817  662
TEST

              precision    recall  f1-score   support

           0       0.77      0.92      0.84      1616
           1       0.61      0.30      0.40       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               1494  122
1                443  191
```

Дерево решений

В [80]:
```
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

CONFUSION MATRIX

col_0              0     1
Credit Default
0               3771     0
1                  0  1479
TEST

              precision    recall  f1-score   support

           0       0.79      0.79      0.79      1616
           1       0.46      0.46      0.46       634

    accuracy                           0.70      2250
   macro avg       0.63      0.63      0.63      2250
weighted avg       0.70      0.70      0.70      2250

CONFUSION MATRIX

col_0              0    1
Credit Default
0               1276  340
1                342  292
```

В [81]:
```
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.neural_network import MLPClassifier
```

Случайный лес

В [82]:
```
model_tree = RandomForestClassifier(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

CONFUSION MATRIX

col_0              0     1
Credit Default
0               3771     0
1                  0  1479
TEST

              precision    recall  f1-score   support

           0       0.77      0.97      0.86      1616
           1       0.77      0.26      0.39       634

    accuracy                           0.77      2250
   macro avg       0.77      0.61      0.62      2250
weighted avg       0.77      0.77      0.73      2250

CONFUSION MATRIX

col_0              0    1
Credit Default
0               1566   50
1                469  165
```

MLP - классификатор

В [83]:
```
model_tree = MLPClassifier(random_state=21)
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       0.79      0.98      0.87      3771
           1       0.85      0.34      0.48      1479

    accuracy                           0.80      5250
   macro avg       0.82      0.66      0.68      5250
weighted avg       0.81      0.80      0.76      5250

CONFUSION MATRIX

col_0              0    1
Credit Default
0               3686   85
1                983  496
TEST

              precision    recall  f1-score   support

           0       0.77      0.96      0.85      1616
           1       0.73      0.27      0.39       634

    accuracy                           0.76      2250
   macro avg       0.75      0.61      0.62      2250
weighted avg       0.76      0.76      0.72      2250

CONFUSION MATRIX

col_0              0    1
Credit Default
0               1552   64
1                465  169
```

В [84]:
```
# from https://www.kaggle.com/krishnaharish/titanic1

from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

models = [
    #KNeighborsClassifier(3),
    #SVC(kernel="linear", C=0.025),
    #SVC(gamma=2, C=1),
    #DecisionTreeClassifier(max_depth=10),
    RandomForestClassifier(n_estimators=100),
    MLPClassifier(),
    #AdaBoostClassifier(),
    #GaussianNB(),
    #QuadraticDiscriminantAnalysis()
]

for model in models:
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    print(score)
```

```
0.7702222222222223
0.7684444444444445
```

В [85]:
```
"""if FINAL:

    models = [
        RandomForestClassifier(n_estimators=100),
        MLPClassifier(),
    ]

    i=1
    for model in models:
        model.fit(training_data, survived)
        prediction = model.predict(testing_data)
        np.savetxt('submission{}.csv'.format(i), prediction, delimiter=",")
        i += 1"""
```

Out[85]:
```
'if FINAL:\n\n    models = [\n        RandomForestClassifier(n_estimators=100),\n        MLPClassifier(),\n    ]\n\n    i=1\n    for m
odel in models:\n        model.fit(training_data, survived)\n        prediction = model.predict(testing_data)\n        np.savetxt(\'su
bmission{}.csv\'.format(i), prediction, delimiter=",")\n        i += 1'
```

В [ ]:
```

```

Бустинговые алгоритмы

XGBoost

В [86]:
```
%%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)
```

```
[23:54:57] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the d
efault evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric i
f 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

CONFUSION MATRIX

col_0              0     1
Credit Default
0               3764     7
1                195  1284
TEST

              precision    recall  f1-score   support

           0       0.78      0.91      0.84      1616
           1       0.60      0.35      0.44       634

    accuracy                           0.75      2250
   macro avg       0.69      0.63      0.64      2250
weighted avg       0.73      0.75      0.73      2250

CONFUSION MATRIX

col_0              0    1
Credit Default
0               1471  145
1                413  221
Wall time: 537 ms
```

LightGBM

В [87]:
```
%%time
model_lgbm = lgbm.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      0.84      0.91      3771
           1       0.71      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

CONFUSION MATRIX

col_0              0     1
Credit Default
0               3185   586
1                 28  1451
TEST

              precision    recall  f1-score   support

           0       0.82      0.70      0.75      1616
           1       0.44      0.60      0.51       634

    accuracy                           0.67      2250
   macro avg       0.63      0.65      0.63      2250
weighted avg       0.71      0.67      0.68      2250

CONFUSION MATRIX

col_0              0    1
Credit Default
0               1126  490
1                252  382
Wall time: 173 ms
```

CatBoost

В [88]:
```
%%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

CONFUSION MATRIX

col_0             0    1
Credit Default
0              3745   26
1               647  832
TEST

              precision    recall  f1-score   support

           0       0.78      0.94      0.86      1616
           1       0.70      0.33      0.45       634

    accuracy                           0.77      2250
   macro avg       0.74      0.64      0.65      2250
weighted avg       0.76      0.77      0.74      2250

CONFUSION MATRIX

col_0             0    1
Credit Default
0              1526   90
1               424  210
Wall time: 15.2 s
```

В [89]:
```
BASE_FEATURE_NAMES
```

Out[89]:
```
['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']
```

В [90]:
```
NEW_FEATURE_NAMES
```

Out[90]:
```
['Home_Ownership_int',
 'Years_in_current_job_int',
 'Purpose_int',
 'Term_int',
 'Tax_Liens_int',
 'Number_of_Credit_Problems_int']
```

В [91]:
```
CAT_FEATURE_NAMES
```

Out[91]:
```
['Home Ownership',
 'Years in current job',
 'Tax Liens',
 'Number of Credit Problems',
 'Bankruptcies',
 'Purpose',
 'Term']
```

В [92]:
```
SELECTED_FEATURE_NAMES
```

Out[92]:
```
['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']
```

В [93]:
```
# 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)
```

В [94]:
```
%%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

CONFUSION MATRIX

col_0              0    1
Credit Default
0               3745   26
1                647  832
TEST

              precision    recall  f1-score   support

           0       0.78      0.94      0.86      1616
           1       0.70      0.33      0.45       634

    accuracy                           0.77      2250
   macro avg       0.74      0.64      0.65      2250
weighted avg       0.76      0.77      0.74      2250

CONFUSION MATRIX

col_0              0    1
Credit Default
0               1526   90
1                424  210
Wall time: 15.3 s
```

В [95]:
```
%%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       0.83      0.99      0.90      3771
           1       0.95      0.48      0.64      1479

    accuracy                           0.85      5250
   macro avg       0.89      0.74      0.77      5250
weighted avg       0.86      0.85      0.83      5250

CONFUSION MATRIX

col_0             0    1
Credit Default
0              3732   39
1               762  717
TEST

              precision    recall  f1-score   support

           0       0.78      0.96      0.86      1616
           1       0.74      0.31      0.44       634

    accuracy                           0.78      2250
   macro avg       0.76      0.63      0.65      2250
weighted avg       0.77      0.78      0.74      2250

CONFUSION MATRIX

col_0             0    1
Credit Default
0              1546   70
1               436  198
Wall time: 33.1 s
```

В [96]:
```
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[96]:
```
2.5496957403651117
```

В [97]:
```
%%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       0.70      0.83      0.76      1479

    accuracy                           0.85      5250
   macro avg       0.81      0.84      0.82      5250
weighted avg       0.86      0.85      0.85      5250

CONFUSION MATRIX

col_0              0     1
Credit Default
0               3234   537
1                252  1227
TEST

              precision    recall  f1-score   support

           0       0.82      0.78      0.80      1616
           1       0.50      0.56      0.53       634

    accuracy                           0.72      2250
   macro avg       0.66      0.67      0.66      2250
weighted avg       0.73      0.72      0.72      2250

CONFUSION MATRIX

col_0              0    1
Credit Default
0               1255  361
1                279  355
Wall time: 40.2 s
```

В [98]:
```
%%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))
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
Wall time: 4.26 s
```

Out[98]:
```
<catboost.core.CatBoostClassifier at 0xd008b38eb0>
```

В [99]:
```
model_catb.best_score_
```

Out[99]:
```
{'learn': {'Recall:use_weights=false': 0.6511156186612576,
  'Logloss': 0.5276628005059759,
  'F1': 0.6992971156471378,
  'Precision:use_weights=false': 0.5583277140930546,
  'Precision:use_weights=true': 0.7632086263279774,
  'Recall:use_weights=true': 0.6511156186612576},
 'validation': {'Recall:use_weights=false': 0.5615141955835962,
  'Logloss': 0.563324853924661,
  'F1': 0.6277433289104256,
  'Precision:use_weights=false': 0.5204582651391162,
  'Precision:use_weights=true': 0.7345541408138336,
  'Recall:use_weights=true': 0.5615141955835962}}
```

В [100]:
```
evaluate_preds(model_catb, X_train, X_test, y_train, y_test)
```

```
TRAIN

              precision    recall  f1-score   support

           0       0.84      0.80      0.82      3771
           1       0.55      0.61      0.58      1479

    accuracy                           0.75      5250
   macro avg       0.69      0.71      0.70      5250
weighted avg       0.76      0.75      0.75      5250

CONFUSION MATRIX

col_0              0    1
Credit Default
0               3020  751
1                578  901
TEST

              precision    recall  f1-score   support

           0       0.82      0.78      0.80      1616
           1       0.50      0.56      0.53       634

    accuracy                           0.72      2250
   macro avg       0.66      0.67      0.66      2250
weighted avg       0.73      0.72      0.72      2250

CONFUSION MATRIX

col_0              0    1
Credit Default
0               1265  351
1                281  353
```

Выбор лучшей модели и подбор гиперпараметров

В [101]:
```
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

В [102]:
```
params = {'iterations':[50, 200, 500, 700, 1500],
          'max_depth':[3, 5, 7]}
```

В [103]:
```
cv = StratifiedKFold(n_splits=3, random_state=21, shuffle=True)
```

В [104]:
```
grid_search = model_catb.grid_search(params, X_train, y_train, cv=cv, stratified=True, plot=True, refit=True)
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
bestTest = 0.5916758615
bestIteration = 44

0:	loss: 0.5916759 best: 0.5916759 (0)	total: 1.08s	remaining: 15.2s
Stopped by overfitting detector  (20 iterations wait)

bestTest = 0.5887683815
bestIteration = 3

1:	loss: 0.5887684 best: 0.5916759 (0)	total: 1.64s	remaining: 10.6s
Stopped by overfitting detector  (20 iterations wait)

bestTest = 0.5887683815
bestIteration = 3

2:	loss: 0.5887684 best: 0.5916759 (0)	total: 2.24s	remaining: 8.97s
Stopped by overfitting detector  (20 iterations wait)

bestTest = 0.5887683815
bestIteration = 3

3:	loss: 0.5887684 best: 0.5916759 (0)	total: 2.84s	remaining: 7.82s
Stopped by overfitting detector  (20 iterations wait)

bestTest = 0.5887683815
bestIteration = 3

4:	loss: 0.5887684 best: 0.5916759 (0)	total: 3.35s	remaining: 6.7s
Stopped by overfitting detector  (20 iterations wait)

bestTest = 0.6137925144
bestIteration = 3

5:	loss: 0.6137925 best: 0.6137925 (5)	total: 3.95s	remaining: 5.93s
Stopped by overfitting detector  (20 iterations wait)

bestTest = 0.6248919924
bestIteration = 121

6:	loss: 0.6248920 best: 0.6248920 (6)	total: 8.25s	remaining: 9.43s
Stopped by overfitting detector  (20 iterations wait)

bestTest = 0.6248919924
bestIteration = 121

7:	loss: 0.6248920 best: 0.6248920 (6)	total: 12.5s	remaining: 11s
Stopped by overfitting detector  (20 iterations wait)

bestTest = 0.6248919924
bestIteration = 121

8:	loss: 0.6248920 best: 0.6248920 (6)	total: 16.8s	remaining: 11.2s
Stopped by overfitting detector  (20 iterations wait)

bestTest = 0.6248919924
bestIteration = 121

9:	loss: 0.6248920 best: 0.6248920 (6)	total: 21s	remaining: 10.5s

bestTest = 0.5975214061
bestIteration = 42

10:	loss: 0.5975214 best: 0.6248920 (6)	total: 22.7s	remaining: 8.25s
Stopped by overfitting detector  (20 iterations wait)

bestTest = 0.5930382268
bestIteration = 47

11:	loss: 0.5930382 best: 0.6248920 (6)	total: 25.4s	remaining: 6.35s
Stopped by overfitting detector  (20 iterations wait)

bestTest = 0.5930382268
bestIteration = 47

12:	loss: 0.5930382 best: 0.6248920 (6)	total: 28.3s	remaining: 4.35s
Stopped by overfitting detector  (20 iterations wait)

bestTest = 0.5930382268
bestIteration = 47

13:	loss: 0.5930382 best: 0.6248920 (6)	total: 31s	remaining: 2.21s
Stopped by overfitting detector  (20 iterations wait)

bestTest = 0.5930382268
bestIteration = 47

14:	loss: 0.5930382 best: 0.6248920 (6)	total: 34s	remaining: 0us
Estimating final quality...
Stopped by overfitting detector  (20 iterations wait)
```

В [105]:
```
grid_search
```

Out[105]:
```
{'params': {'depth': 5, 'iterations': 200},
 'cv_results': defaultdict(list,
             {'iterations': [0,
               1,
               2,
               3,
               4,
               5,
               6,
               7,
               8,
               9,
               10,
               11,
               12,
               13,
               14,
               15,
               16,
```

В [106]:
```
pd.DataFrame(grid_search['cv_results']).sort_values('test-F1-mean', ascending=False).head()
```

Out[106]:

	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
145	145	0.649027	0.030971	0.705988	0.008094	0.560737	0.008034	0.532006	0.006982
152	152	0.647941	0.033698	0.708889	0.009675	0.560206	0.007832	0.530414	0.007344
148	148	0.647862	0.032611	0.705834	0.010774	0.560406	0.007883	0.531348	0.007333
153	153	0.647752	0.033839	0.709647	0.009159	0.560217	0.007896	0.530192	0.007263
146	146	0.647746	0.031108	0.705869	0.008667	0.560642	0.008083	0.531804	0.006948

Обучение и оценка финальной модели

В [107]:
```
%%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)
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
TRAIN

              precision    recall  f1-score   support

           0       0.84      0.80      0.82      3771
           1       0.55      0.63      0.59      1479

    accuracy                           0.75      5250
   macro avg       0.70      0.71      0.70      5250
weighted avg       0.76      0.75      0.75      5250

CONFUSION MATRIX

col_0              0    1
Credit Default
0               3014  757
1                554  925
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      0.66      2250
weighted avg       0.73      0.72      0.72      2250

CONFUSION MATRIX

col_0              0    1
Credit Default
0               1257  359
1                275  359
Wall time: 3.31 s
```

В [3]:
```
np.ceil?
```

```
Object `ceil` not found.
```

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.

Опциона́льный - 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 постронияния 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 [4]:

```
Object `numpy.ceil` not found.
```

В [ ]: