Исследование надёжности заёмщиков

Заказчик — кредитный отдел банка. Нужно разобраться, влияет ли семейное положение и количество детей клиента на факт погашения кредита в срок. Входные данные от банка — статистика о платёжеспособности клиентов.

Результаты исследования будут учтены при построении модели **кредитного скоринга** — специальной системы, которая оценивает способность потенциального заёмщика вернуть кредит банку.

Шаг 1. Откройте файл с данными и изучите общую информацию.

In [1]:
```
import pandas as pd
import math
from IPython.display import display

data = pd.read_csv('/datasets/data.csv')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21525 entries, 0 to 21524
Data columns (total 12 columns):
children            21525 non-null int64
days_employed       19351 non-null float64
dob_years           21525 non-null int64
education           21525 non-null object
education_id        21525 non-null int64
family_status       21525 non-null object
family_status_id    21525 non-null int64
gender              21525 non-null object
income_type         21525 non-null object
debt                21525 non-null int64
total_income        19351 non-null float64
purpose             21525 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 2.0+ MB
```

Вывод

Дататасет состоит из 12 столбцов и 21525 строк. Названия столбцов не требуют изменений. Столбцы `days_employed` и `total_income` содержат 2174 нулевых элемента, что составляет 10% всех данных.

Шаг 2. Предобработка данных

Обработка пропусков

Найдём уникальные значения столбцов, заданных списком `data_headlines`, для выявления артефактов.

Создадим список `data_headlines` с интересующими столбцами и затем в цикле найдём уникальные значения по каждому из них. Закомментируем код после выполнения.

In [2]:
```
data_headlines = ['children', 'dob_years', 'education', 'family_status', 'gender', 'income_type', 'deb
t']
for headline in data_headlines:
    print(headline)
    print(data[headline].value_counts())
```

```
children
 0     14149
 1      4818
 2      2055
 3       330
 20       76
-1        47
 4        41
 5         9
Name: children, dtype: int64
dob_years
35    617
40    609
41    607
```

```
34    603
38    598
42    597
33    581
39    573
31    560
36    555
44    547
29    545
30    540
48    538
37    537
50    514
43    513
32    510
49    508
28    503
45    497
27    493
56    487
52    484
47    480
54    479
46    475
58    461
57    460
53    459
51    448
59    444
55    443
26    408
60    377
25    357
61    355
62    352
63    269
64    265
24    264
23    254
65    194
66    183
22    183
67    167
21    111
0     101
68     99
69     85
70     65
71     58
20     51
72     33
19     14
73      8
74      6
75      1
Name: dob_years, dtype: int64
education
среднее                13750
высшее                  4718
СРЕДНЕЕ                  772
Среднее                  711
неоконченное высшее       668
ВЫСШЕЕ                    274
Высшее                    268
начальное                 250
Неоконченное высшее        47
НЕОКОНЧЕННОЕ ВЫСШЕЕ        29
НАЧАЛЬНОЕ                  17
Начальное                  15
ученая степень              4
УЧЕНАЯ СТЕПЕНЬ              1
Ученая степень              1
Name: education, dtype: int64
family_status
женат / замужем          12380
гражданский брак          4177
Не женат / не замужем     2813
в разводе                 1195
вдовец / вдова             960
Name: family_status, dtype: int64
gender
```

```
F      14236
M       7288
XNA        1
Name: gender, dtype: int64
income_type
сотрудник          11119
компаньон           5085
пенсионер           3856
госслужащий         1459
безработный            2
предприниматель        2
студент                1
в декрете              1
Name: income_type, dtype: int64
debt
0    19784
1     1741
Name: debt, dtype: int64
```

Неадекватными мне кажутся количество детей в семье: -1, 20; возраст клиента в годах: 0.

Объясняю их присутствие человеческим фактором: -1 — лишним минусом, 20 — лишним нулем, 0 в возрасте — пропущенной перед нулем цифрой.

Произведём замену для `children`: с -1 на 1, с 20 на 2.

In [3]:
```
data.loc[data['children'] == -1, 'children'] = 1 # замена 'children' с -1 на 1
data.loc[data['children'] == 20, 'children'] = 2 # замена 'children' с 20 на 2

# print(data.loc[data['children']== 1]['children'].count()) # Проверка количества для сравнения с Excel
# print(data.loc[data['children']== 2]['children'].count()) # Проверка количества для сравнения с Excel
```

Заменим пенсионерам 0 в столбце `dob_years` на средний возраст пенсионеров. Остальным заменим 0 на средний возраст всех заёмщиков.

In [4]:
```
old_mean = int(data[(data['income_type'] == 'пенсионер')]['dob_years'].mean()) # Средний возраст пенсио
неров, целое число
# print(old_mean) # вывод среднего возраста всех пенсионеров
# Проверка количества пенсионеров с нулевым возрастом
# print(data.loc[(data['dob_years'] == 0) & (data['income_type'] == 'пенсионер')]['dob_years'].count())
```

Исправим пенсионеров с нулевым возрастом.

In [5]:
```
data.loc[(data['dob_years'] == 0) & (data['income_type'] == 'пенсионер'), 'dob_years'] = old_mean
# Проверка правильности замены
# print(data.loc[(data['dob_years'] == 0) & (data['income_type'] == 'пенсионер')]['dob_years'].count())
```

Найдём средний возраст всех заёмщиков, целое число.

In [6]:
```
dob_years_mean = int(data['dob_years'].mean())
# print(dob_years_mean) # вывод среднего возраста всех заемщиков
```

Исправим остальных заёмщиков с нулевым возрастом.

In [7]:
```
data.loc[data['dob_years'] == 0, 'dob_years'] = dob_years_mean
```

В столбце `days_employed` присутствуют отрицательные значения. Объясняю их наличие неправильной разностью дат (из меньшей вычитали большую), поэтому заменим на абсолютные значения.

In [8]:
```
data['days_employed'] = abs(data['days_employed'])
```

Датасет содержит 2174 нулевых элемента в столбцах `days_employed` и `total_income`. Пропуски для пенсионеров, студентов, безработных и находящихся в декрете можно было бы объяснить отсутствием дохода, однако таковых всего 413, что составляет менее четверти пропусков. Лиц младше 26 без дохода и трудового стажа ещё меньше — 121. Поэтому вместо поиска причин ограничимся заменой нулевых элементов средними значениями по столбцам.

In [9]:
```
# days_employed_mean = data['days_employed'].mean() # среднее значение 'days_employed'
# total_income_mean = data['total_income'].mean() # среднее значение 'total_income'
# print(days_employed_mean, total_income_mean) # вывод средних значений 'days_employed' и 'total_incom
e'
```

Заменим нулевые значения столбцов `days_employed` и `total_income`.

In [10]:
```
# data.loc[data['days_employed'].isnull(), 'days_employed'] = days_employed_mean
# data.loc[data['total_income'].isnull(), 'total_income'] = total_income_mean
```

Вывод

Артефакты исправлены. Пропущенные значения столбцов трудового стажа и ежемесячного дохода заменены средними значениями, что может быть некорректно, однако иной замены я не вижу. К тому же трудовой стаж не требуется для ответов на вопросы.

Сначала найдём дубликаты для проверки предположения, что дублируются строки с нулевыми элементами. Приведём строки в столбцах `education` и `family_status` к нижнему регистру вызовом метода `lower()`, подсчитаем количество дубликатов и выведем их на экран.

In [11]:
```
data['education'] = data['education'].str.lower() # приведение строк столбца 'education' к нижнему реги
стру
data['family_status'] = data['family_status'].str.lower() # приведение строк столбца 'family_status' к
 нижнему регистру

print('Количество дубликатов в датасете:', data.duplicated().sum())
display(data[data.duplicated()]) # вывод дубликатов с должниками
```

```
Количество дубликатов в датасете: 71
```

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_in
2849	0	NaN	41	среднее	1	женат / замужем	0	F	сотрудник	0	
3290	0	NaN	58	среднее	1	гражданский брак	1	F	пенсионер	0	
4182	1	NaN	34	высшее	0	гражданский брак	1	F	сотрудник	0	
4851	0	NaN	60	среднее	1	гражданский брак	1	F	пенсионер	0	
5557	0	NaN	58	среднее	1	гражданский брак	1	F	пенсионер	0	
...	...	...	...	...	...	...	...	...	...	...	
20702	0	NaN	64	среднее	1	женат / замужем	0	F	пенсионер	0	
21032	0	NaN	60	среднее	1	женат / замужем	0	F	пенсионер	0	
21132	0	NaN	47	среднее	1	женат / замужем	0	F	сотрудник	0	
21281	1	NaN	30	высшее	0	женат / замужем	0	F	сотрудник	0	
21415	0	NaN	54	среднее	1	женат / замужем	0	F	пенсионер	0	

71 rows × 12 columns

Видно, что дублируются строки с нулевыми элементами и среди заёмщиков должников нет.

Распределим пропуски по типам занятости.

In [12]:
```
display(data[data['total_income'].isnull()]['income_type'].value_counts())
```

```
сотрудник          1105
компаньон           508
пенсионер           413
госслужащий         147
предприниматель       1
Name: income_type, dtype: int64
```

Пропуски для пенсионеров, студентов, безработных и находящихся в декрете можно было бы объяснить отсутствием дохода, однако таковых всего 413, что составляет менее четверти пропусков. Подсчитаем количество лиц младше 26 без дохода и трудового стажа.

In [13]:
```
display(data[(data['total_income'].isnull()) & (data['dob_years'] < 26)]['dob_years'].count())
```

```
121
```

Их ещё меньше — 121. Таким образом, причина появления пропусков неясна.

Заполним пропуски в столбцах `days_employed` и `total_income` средними значениями по типу занятости. Для этого выделим уникальные значения столбца `income_type`.

In [14]:
```
income_type_unique = data['income_type'].unique()
print(income_type_unique)
```

```
['сотрудник' 'пенсионер' 'компаньон' 'госслужащий' 'безработный'
 'предприниматель' 'студент' 'в декрете']
```

В цикле для каждого типа занятости найдём средний трудовой стаж и средний доход и заполним пропущенные значения.

In [15]:
```
for i in income_type_unique:
    days_employed_mean = data[data['income_type'] == i]['days_employed'].mean() # среднее значение 'day
s_employed' по каждому типу
    total_income_mean = data[data['income_type'] == i]['total_income'].mean() # среднее значение 'total
_income' по каждому типу
    # Заменим нулевые значения средними по каждому типу
    data.loc[(data['days_employed'].isnull()) & (data['income_type'] == i), 'days_employed'] = days_emp
loyed_mean
    data.loc[(data['total_income'].isnull()) & (data['income_type'] == i), 'total_income'] = total_inco
me_mean
```

Замена типа данных

Столбцы `days_employed` и `total_income` имеют вещественный тип данных. Заменим его на целочисленный для удобства и потому что этого требует задание. Также столбец `debt` имеет всего два значения. Заменим его на логический, где 1 — `True` — наличие долга, 0 — `False` — отсутствие долга.

In [16]:
```
data['days_employed'] = data['days_employed'].astype('int') # замена типа данных столбцу 'days_employe
d' на целочисленный
data['total_income'] = data['total_income'].astype('int') # замена типа данных столбцу 'total_income' н
а целочисленный
data['debt'] = data['debt'].astype('bool') # замена типа данных столбцу 'debt' на логический, как более
 подходящий
```

Метод `to_numeric()` переводит числа в тип данных *float64*.

Вывод

Для удобства обработки произведены замены типов данных в следующих столбцах:

- `days_employed` и `total_income` — на целочисленный;
- `debt` — на логический.

Обработка дубликатов

Строки в столбцах `education` и `family_status` имеют разный регистр. Приведём их к нижнему регистру вызовом метода `lower()`.

In [17]:
```
data['education'] = data['education'].str.lower()
data['family_status'] = data['family_status'].str.lower()
```

Подсчитаем количество дубликатов.

In [18]:
```
print('Количество дубликатов в датасете:', data.duplicated().sum())
```

```
Количество дубликатов в датасете: 71
```

Предположение: дублируются те строки, в которых нулевые элементы заменили на средние значения. Проверим, есть ли среди них должники.

In [19]:
```
display(data[(data.duplicated()) & (data['debt'] == 1)]) # вывод дубликатов с должниками
```

children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income

Таблица пустая, удалим дубликаты.

In [20]:
```
data = data.drop_duplicates().reset_index(drop = True)
```

Вывод

После обработки данных в предыдущих заданиях обнаружен 71 дубликат. Ни один из них не принадлежит заёмщикам-должникам, поэтому можем удалить дубликаты без вреда для результатов исследования.

Для проверки найдём дубликаты до замены нулевых элементов на средние (сделано выше).

Видно, что дублируются строки с нулевыми элементами и среди заёмщиков должников нет.

Причина их появления может быть в том, что без id заёмщика и уникальных значений трудового стажа и ежемесячного дохода, строки совпадают друг с другом. Целесообразность удаления сомнительна (дубликаты составляют менее 0,33% всех данных), однако того требует задание.

Лемматизация

Импортируем библиотеку `pymystem3`.

In [21]:
```
from pymystem3 import Mystem
m = Mystem()
```

Вызовем контейнер `Counter` из модуля `collections` для подсчёта лемматизированных слов.

In [22]:
```
from collections import Counter
```

Найдём наиболее часто встречающиеся цели получения кредита. Для этого лемматизируем в цикле каждый элемент столбца `purpose` и объединим их в единый список `purpose_list`.

In [23]:
```
purpose_list = []
for i in range(data.shape[0]):
    purpose_list += m.lemmatize(data['purpose'][i])
```

Подсчитаем количество лемматизированных слов.

In [24]:
```
print(Counter(purpose_list))
```

```
Counter({' ': 33570, '\n': 21454, 'недвижимость': 6351, 'покупка': 5897, 'жилье': 4460, 'автомобиль':
4306, 'образование': 4013, 'с': 2918, 'операция': 2604, 'свадьба': 2324, 'свой': 2230, 'на': 2222, 'с
троительство': 1878, 'высокий': 1374, 'получение': 1314, 'коммерческий': 1311, 'для': 1289, 'жилой':
1230, 'сделка': 941, 'дополнительный': 906, 'заниматься': 904, 'проведение': 768, 'сыграть': 765, 'сд
ача': 651, 'семья': 638, 'собственный': 635, 'со': 627, 'ремонт': 607, 'подержанный': 486, 'подержат
ь': 478, 'приобретение': 461, 'профильный': 436})
```

Вручную выделим основные цели получения кредита по возрастанию упоминаний.

In [25]:
```
purpose_pattern = ['ремонт', 'свадьба', 'образование', 'автомобиль', 'жилье', 'недвижимость']
```

Поставим в соответствие каждому элементу столбца `purpose` единственную цель из эталонного списка `purpose_pattern`. Для этого создадим функцию, сравнивающую каждую строку с эталоном и возвращающую первый совпавший элемент.

In [26]:
```
def intersection(list_input):
    """
     Функция поэлементно сравнивает список list_input с эталонным списком purpose_pattern
     Возвращает первый совпавший элемент, иначе None
    """
    for i in list_input:
        for j in purpose_pattern:
            if i == j:
                return j
    return None
```

Создадим столбец `purpose_shortened` для вычленения цели кредита. В цикле каждая цель получения кредита сравнивается с эталонными при помощи функции `intersection`. Результат записывается в соответствующую строку стобца `purpose_shortened`.

In [27]:
```
for i in range(data.shape[0]):
    data.loc[i, 'purpose_shortened'] = intersection(m.lemmatize(data['purpose'][i]))
display(data.head(20)) # проверка
```

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_i
0	1	8437	42	высшее	0	женат / замужем	0	F	сотрудник	False	
1	1	4024	36	среднее	1	женат / замужем	0	F	сотрудник	False	
2	0	5623	33	среднее	1	женат / замужем	0	M	сотрудник	False	
3	3	4124	32	среднее	1	женат / замужем	0	M	сотрудник	False	
4	0	340266	53	среднее	1	гражданский брак	1	F	пенсионер	False	
5	0	926	27	высшее	0	гражданский брак	1	M	компаньон	False	
6	0	2879	43	высшее	0	женат / замужем	0	F	компаньон	False	
7	0	152	50	среднее	1	женат / замужем	0	M	сотрудник	False	
8	2	6929	35	высшее	0	гражданский брак	1	F	сотрудник	False	
9	0	2188	41	среднее	1	женат / замужем	0	M	сотрудник	False	
10	2	4171	36	высшее	0	женат / замужем	0	M	компаньон	False	
11	0	792	40	среднее	1	женат / замужем	0	F	сотрудник	False	
12	0	365003	65	среднее	1	гражданский брак	1	M	пенсионер	False	
13	0	1846	54	неоконченное высшее	2	женат / замужем	0	F	сотрудник	False	
14	0	1844	56	высшее	0	гражданский брак	1	F	компаньон	True	
15	1	972	26	среднее	1	женат / замужем	0	F	сотрудник	False	
16	0	1719	35	среднее	1	женат / замужем	0	F	сотрудник	False	
17	0	2369	33	высшее	0	гражданский брак	1	M	сотрудник	False	
18	0	400281	53	среднее	1	вдовец / вдова	2	F	пенсионер	False	
19	0	10038	48	среднее	1	в разводе	3	F	сотрудник	False	

Примечание: использование цикла для работы с `DataFrame` представляется нерациональным по времени. Как я понял, метод `apply()` не сочетается с `m.lemmatize()`. Какие ещё способы можно применить?

In [80]:
```
# data['purpose_shortened_2'] = data['purpose'].apply(lambda text: m.lemmatize(text))
# display(data.head(10))
data['purpose_shortened_2'] = data['purpose'].apply(lambda text: intersection(m.lemmatize(text)))
display(data.head(30))
```

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_i
0	1	8437	42	высшее	0	женат / замужем	0	F	сотрудник	False	
1	1	4024	36	среднее	1	женат / замужем	0	F	сотрудник	False	
2	0	5623	33	среднее	1	женат / замужем	0	M	сотрудник	False	
3	3	4124	32	среднее	1	женат / замужем	0	M	сотрудник	False	
4	0	340266	53	среднее	1	гражданский брак	1	F	пенсионер	False	
5	0	926	27	высшее	0	гражданский брак	1	M	компаньон	False	
6	0	2879	43	высшее	0	женат / замужем	0	F	компаньон	False	
7	0	152	50	среднее	1	женат / замужем	0	M	сотрудник	False	
8	2	6929	35	высшее	0	гражданский брак	1	F	сотрудник	False	
9	0	2188	41	среднее	1	женат / замужем	0	M	сотрудник	False	
10	2	4171	36	высшее	0	женат / замужем	0	M	компаньон	False	
11	0	792	40	среднее	1	женат / замужем	0	F	сотрудник	False	
12	0	365003	65	среднее	1	гражданский брак	1	M	пенсионер	False	
13	0	1846	54	неоконченное высшее	2	женат / замужем	0	F	сотрудник	False	
14	0	1844	56	высшее	0	гражданский брак	1	F	компаньон	True	
15	1	972	26	среднее	1	женат / замужем	0	F	сотрудник	False	
16	0	1719	35	среднее	1	женат / замужем	0	F	сотрудник	False	
17	0	2369	33	высшее	0	гражданский брак	1	M	сотрудник	False	
18	0	400281	53	среднее	1	вдовец / вдова	2	F	пенсионер	False	
19	0	10038	48	среднее	1	в разводе	3	F	сотрудник	False	
20	1	1311	36	среднее	1	женат / замужем	0	M	сотрудник	False	
21	1	253	33	среднее	1	гражданский брак	1	F	сотрудник	False	
22	1	1766	24	среднее	1	гражданский брак	1	F	сотрудник	False	
23	0	272	21	высшее	0	гражданский брак	1	M	сотрудник	False	
24	1	338551	57	среднее	1	не женат / не замужем	4	F	пенсионер	False	
25	0	363548	67	среднее	1	женат / замужем	0	M	пенсионер	False	
26	0	3399	41	среднее	1	женат / замужем	0	M	госслужащий	False	
27	0	529	28	высшее	0	женат / замужем	0	M	сотрудник	False	

28	1	717	26	высшее	0	женат / замужем	0	F	сотрудник	False
29	0	365003	63	среднее	1	не женат / не замужем	4	F	пенсионер	False

Жильё может означать как дом, так и квартиру. Недвижимость почти в трети случаев включает слово "строительство", т.е. недвижимость — это скорее частный дом. Перебравшись на юг, обнаружил, что многие предпочитают жить в частных домах.

Кроме того, разделение недвижимости и жилья не повлияло на вывод исследования.

Вывод

Выделены основные цели получения кредита, на их основе создан столбец `purpose_shortened`, однозначно определяющий цель каждого заёмщика.

Категоризация данных

Для ответов на вопросы 1 и 3 категоризируем столбцы `children` — количество детей в семье и `total_income` — ежемесячный доход.

Создадим функцию `having_children` для категоризации наличия детей.

In [28]:
```
def having_children(children):
    """
     Возвращает True, если дети есть
     Возвращает False, если детей нет
    """
    if children > 0:
        return True
    return False
```

Создадим столбец `having_children` для определения наличия детей.

In [29]:
```
data['having_children'] = data['children'].apply(having_children)
```

Создадим функцию `total_income_leveling` для категоризации уровня дохода. Пороги для каждой группы и их количество выбраны условно. Численность групп соразмерна.

In [30]:
```
def total_income_leveling(income):
    """
     Возвращает уровень дохода по значению дохода income, используя правила:
        - 'бедные' при значении income < 100000
        - 'ниже среднего' при значении income более 100000 и менее 150000, включая 150000
        - 'средние' при значении income более 150000 и менее 200000, включая 200000
        - 'выше среднего' при значении income более 200000 и менее 250000, включая 250000
        - 'богатые' во всех остальных случаях
    """
    if income < 100000:
        return 'бедные'
    if income <= 150000:
        return 'ниже среднего'
    if income <= 200000:
        return 'средние'
    if income <= 250000:
        return 'выше среднего'
    return 'богатые'
```

Создадим столбец `total_income_group` для определения уровня дохода.

In [31]:
```
data['total_income_group'] = data['total_income'].apply(total_income_leveling)
```

Вывод

Для ответов на вопросы 1 и 3 созданы следующие столбцы:

- `having_children` для распределения по двум категориям — заёмщики с детьми и бездетные;

- `total_income_group` для распределения по пяти категориям в зависимости от уровня дохода.

Границы для категорий с заработком были выбраны условно по критерию соразмерной численности после нескольких попыток.

Изменим категоризацию, разделив уровень дохода на семь приблизительно равных по количеству заёмщиков частей посредством функции `qcut()` и записав принадлежность в столбец `total_income_group_2`.

In [32]:
```
data['total_income_group_2'] = pd.qcut(data['total_income'], 7)
```

Выведем получившиеся категории на экран.

In [33]:
```
display(data['total_income_group_2'].value_counts())
```

```
(137468.0, 161380.0]        3535
(113605.571, 137468.0]      3066
(242019.571, 2265604.0]     3065
(191703.143, 242019.571]    3065
(87295.429, 113605.571]     3065
(20666.999, 87295.429]      3065
(161380.0, 191703.143]      2593
Name: total_income_group_2, dtype: int64
```

Шаг 3. Ответьте на вопросы

- Есть ли зависимость между наличием детей и возвратом кредита в срок?

Сгруппируем должников по наличию детей.

In [34]:
```
having_children_debt = data.groupby('having_children')['debt'].sum()
```

Подсчитаем общее количество заёмщиков по наличию детей.

In [35]:
```
having_children_total = data.groupby('having_children')['debt'].count()
```

Найдём относительное количество должников по наличию детей, поделив количество должников на общее количество.

In [36]:
```
having_children_relative = having_children_debt / having_children_total
```

Объединим получившиеся объекты `Series` при помощи функции `dict` и выведем на экран с сортировкой.

In [37]:
```
display(pd.DataFrame(dict(debt = having_children_debt, total = having_children_total, relative = having
_children_relative)).sort_values('relative', ascending = False))
```

having_children	debt	total	relative
True	678.0	7363	0.092082
False	1063.0	14091	0.075438

Проделаем тоже самое при помощи pivot_table()

Сгруппируем должников по наличию детей.

In [38]:
```
having_children_debt = data.pivot_table(index='having_children', values='debt', aggfunc='sum')
```

Подсчитаем общее количество заёмщиков по наличию детей.

In [39]:
```
having_children_total = data.pivot_table(index='having_children', values='debt', aggfunc='count')
```

Объединим получившиеся объекты методом `merge()`.

In [40]:
```
having_children_relative = having_children_debt.merge(having_children_total, on='having_children', how=
'right')
```

Изменим заголовки на `debt` и `total`.

In [41]:
```
having_children_relative.columns = ['debt','total']
```

Найдём относительное количество должников `relative` по наличию детей, поделив количество должников на общее количество заёмщиков, и выведем на экран с сортировкой.

In [42]:
```
having_children_relative['relative'] = having_children_relative['debt'] / having_children_relative['tot
al']
display(having_children_relative.sort_values('relative', ascending = False))
```

having_children	debt	total	relative
True	678.0	7363	0.092082
False	1063.0	14091	0.075438

Вывод

Отсутствие детей помогает вернуть кредит в срок.

- Есть ли зависимость между семейным положением и возвратом кредита в срок?

Сгруппируем должников по семейному положению.

In [43]:
```
family_status_debt = data.groupby('family_status')['debt'].sum()
```

Подсчитаем общее количество заёмщиков по семейному положению.

In [44]:
```
family_status_total = data.groupby('family_status')['debt'].count()
```

Подсчитаем средний возраст заёмщиков по семейному положению.

In [45]:
```
family_status_age = data.groupby('family_status')['dob_years'].mean()
```

Найдём относительное количество должников по семейному положению, поделив количество должников на общее количество.

In [46]:
```
family_status_relative = family_status_debt / family_status_total
```

Объединим получившиеся объекты `Series` при помощи функции `dict` и выведем на экран с сортировкой.

In [47]:
```
display(pd.DataFrame(dict(debt = family_status_debt, total = family_status_total, relative = family_status_relative, mean_age = family_status_age)).sort_values('relative', ascending = False))
```

family_status	debt	total	relative	mean_age
не женат / не замужем	274.0	2810	0.097509	38.631317
гражданский брак	388.0	4151	0.093471	42.290773
женат / замужем	931.0	12339	0.075452	43.728260
в разводе	85.0	1195	0.071130	45.904603
вдовец / вдова	63.0	959	0.065693	56.811262

Проделаем тоже самое при помощи pivot_table()

Сгруппируем должников по семейному положению.

In [48]:
```
family_status_debt = data.pivot_table(index='family_status', values='debt', aggfunc='sum')
```

Подсчитаем общее количество заёмщиков по семейному положению.

In [49]:
```
family_status_total = data.pivot_table(index='family_status', values='debt', aggfunc='count')
```

Подсчитаем средний возраст заёмщиков по семейному положению.

In [50]:
```
family_status_age = data.pivot_table(index='family_status', values='dob_years', aggfunc='mean')
```

Объединим получившиеся объекты методом `merge()`.

In [51]:
```
family_status_relative = family_status_debt.merge(family_status_total, on='family_status', how='right')
family_status_relative = family_status_relative.merge(family_status_age, on='family_status', how='right')
```

Изменим заголовки на `debt`, `total` и `mean_age`.

In [52]:
```
family_status_relative.columns = ['debt','total', 'mean_age']
```

Найдём относительное количество должников `relative` по семейному положению, поделив количество должников на общее количество, и выведем на экран с сортировкой.

In [53]:
```
family_status_relative['relative'] = family_status_relative['debt'] / family_status_relative['total']
display(family_status_relative.sort_values('relative', ascending = False))
```

family_status	debt	total	mean_age	relative
не женат / не замужем	274.0	2810	38.631317	0.097509
гражданский брак	388.0	4151	42.290773	0.093471
женат / замужем	931.0	12339	43.728260	0.075452
в разводе	85.0	1195	45.904603	0.071130
вдовец / вдова	63.0	959	56.811262	0.065693

Вывод

Чаще всего кредиты не возвращают холостяки и живущие в гражданском браке. Возможно, ввиду молодости заёмщиков?

Реже всего — вдовцы / вдовы и разведённые.

Дополнение: молодость действительно связана с семейным положением и просрочкой по кредиту.

- Есть ли зависимость между уровнем дохода и возвратом кредита в срок?

Сгруппируем должников по уровню дохода.

In [54]:
```
total_income_debt = data.groupby('total_income_group_2')['debt'].sum()
```

Подсчитаем общее количество заёмщиков по уровню дохода.

In [55]:
```
total_income_total = data.groupby('total_income_group_2')['debt'].count()
```

Найдём относительное количество должников `relative` по уровню дохода, поделив количество должников на общее количество.

In [56]:
```
total_income_relative = total_income_debt / total_income_total
```

Объединим получившиеся объекты `Series` со списком категорий `income_levels` при помощи функции `dict` и выведем на экран с сортировкой.

In [57]:
```
income_levels = ['нищие', 'бедные', 'ниже среднего', 'средние', 'выше среднего', 'состоятельные', 'богатые']
display(pd.DataFrame(dict(income_levels = income_levels, debt = total_income_debt, total = total_income_total, relative = total_income_relative)).sort_values('relative', ascending = False))
```

total_income_group_2	income_levels	debt	total	relative
(161380.0, 191703.143]	выше среднего	229.0	2593	0.088315
(137468.0, 161380.0]	средние	312.0	3535	0.088260
(113605.571, 137468.0]	ниже среднего	266.0	3066	0.086758
(87295.429, 113605.571]	бедные	262.0	3065	0.085481
(20666.999, 87295.429]	нищие	233.0	3065	0.076020
(191703.143, 242019.571]	состоятельные	228.0	3065	0.074388
(242019.571, 2265604.0]	богатые	211.0	3065	0.068842

Проделаем тоже самое при помощи pivot_table()

Сгруппируем должников по уровню дохода.

In [58]:
```
total_income_debt = data.pivot_table(index='total_income_group_2', values='debt', aggfunc='sum')
```

Подсчитаем общее количество заёмщиков по уровню дохода.

In [59]:
```
total_income_total = data.pivot_table(index='total_income_group_2', values='debt', aggfunc='count')
```

Объединим получившиеся объекты методом `merge()`.

In [60]:
```
total_income_relative = total_income_debt.merge(total_income_total, on='total_income_group_2', how='right')
```

Изменим заголовки на `debt` и `total`.

In [61]:
```
total_income_relative.columns = ['debt','total']
```

Добавим столбец со списком категорий `income_levels`.

In [62]:
```
income_levels = ['нищие', 'бедные', 'ниже среднего', 'средние', 'выше среднего', 'состоятельные', 'богатые']
total_income_relative['income_levels'] = income_levels
```

Найдём относительное количество должников `relative` по уровню дохода, поделив количество должников на общее количество, и выведем на экран с сортировкой.

In [63]:
```
total_income_relative['relative'] = total_income_relative['debt'] / total_income_relative['total']
display(total_income_relative.sort_values('relative', ascending = False))
```

total_income_group_2	debt	total	income_levels	relative
(161380.0, 191703.143]	229.0	2593	выше среднего	0.088315
(137468.0, 161380.0]	312.0	3535	средние	0.088260
(113605.571, 137468.0]	266.0	3066	ниже среднего	0.086758
(87295.429, 113605.571]	262.0	3065	бедные	0.085481
(20666.999, 87295.429]	233.0	3065	нищие	0.076020
(191703.143, 242019.571]	228.0	3065	состоятельные	0.074388
(242019.571, 2265604.0]	211.0	3065	богатые	0.068842

Вывод

Чаще всего возвращают в срок богатые заёмщики. Реже всего — заёмщики со средним доходом. Бедность напротив не препятствует возврату кредита.

- Как разные цели кредита влияют на его возврат в срок?

Сгруппируем должников в зависимости от целей кредита.

In [64]:
```
purpose_shortened_debt = data.groupby('purpose_shortened')['debt'].sum()
```

Подсчитаем общее количество заёмщиков в зависимости от целей кредита.

In [65]:
```
purpose_shortened_total = data.groupby('purpose_shortened')['debt'].count()
```

Подсчитаем средний возраст заёмщиков в зависимости от целей кредита.

In [66]:
```
purpose_shortened_age = data.groupby('purpose_shortened')['dob_years'].mean()
```

Найдём относительное количество должников в зависимости от целей кредита, поделив количество должников на общее количество.

In [67]:
```
purpose_shortened_relative = purpose_shortened_debt / purpose_shortened_total
```

Объединим получившиеся объекты `Series` при помощи функции `dict` и выведем на экран с сортировкой.

In [68]:
```
display(pd.DataFrame(dict(debt = purpose_shortened_debt, total = purpose_shortened_total, relative = purpose_shortened_relative, mean_age = purpose_shortened_age)).sort_values('relative', ascending = False))
```

purpose_shortened	debt	total	relative	mean_age
автомобиль	403.0	4306	0.093590	43.705759
образование	370.0	4013	0.092200	43.619736
свадьба	186.0	2324	0.080034	43.393287
недвижимость	474.0	6351	0.074634	43.513935
жилье	273.0	3853	0.070854	43.256164
ремонт	35.0	607	0.057661	42.655684

Проделаем тоже самое при помощи pivot_table()

Сгруппируем должников в зависимости от целей кредита.

In [69]:
```
purpose_shortened_debt = data.pivot_table(index='purpose_shortened', values='debt', aggfunc='sum')
```

Посчитаем общее количество заёмщиков в зависимости от целей кредита.

In [70]:
```
purpose_shortened_total = data.pivot_table(index='purpose_shortened', values='debt', aggfunc='count')
```

Подсчитаем средний возраст заёмщиков в зависимости от целей кредита.

In [71]:
```
purpose_shortened_age = data.pivot_table(index='purpose_shortened', values='dob_years', aggfunc='mean')
```

Объединим получившиеся объекты методом `merge()`.

In [72]:
```
purpose_shortened_relative = purpose_shortened_debt.merge(purpose_shortened_total, on='purpose_shortened', how='right')
purpose_shortened_relative = purpose_shortened_relative.merge(purpose_shortened_age, on='purpose_shortened', how='right')
```

Изменим заголовки на `debt`, `total` и `mean_age`.

In [73]:
```
purpose_shortened_relative.columns = ['debt','total', 'mean_age']
```

Найдём относительное количество должников `relative` в зависимости от целей кредита, поделив количество должников на общее количество, и выведем на экран с сортировкой.

In [74]:
```
purpose_shortened_relative['relative'] = purpose_shortened_relative['debt'] / purpose_shortened_relative['total']
display(purpose_shortened_relative.sort_values('relative', ascending = False))
```

purpose_shortened	debt	total	mean_age	relative
автомобиль	403.0	4306	43.705759	0.093590
образование	370.0	4013	43.619736	0.092200
свадьба	186.0	2324	43.393287	0.080034
недвижимость	474.0	6351	43.513935	0.074634
жилье	273.0	3853	43.256164	0.070854
ремонт	35.0	607	42.655684	0.057661

Вывод

Наиболее рискованными являются кредиты на автомобиль, образование и свадьбу. Возможно, ввиду молодости заёмщиков?

Наиболее безопасными — кредиты на ремонт.

Дополнение: молодость заёмщиков не связана с целями кредита.

Шаг 4. Общий вывод

В датасете обнаружены и исправлены следующие артефакты: количество детей в семье: -1, 20; возраст клиента в годах: 0.

Объясняю их присутствие человеческим фактором: -1 — лишним минусом, 20 — лишним нулем, 0 в возрасте — пропущенной перед нулем цифрой.

Обнаружено 2174 пропуска в стобцах трудового стажа и ежемесячного дохода. Причины появления пропусков определить не удалось. Они были заменены средними значениями в зависимости от типа занятости.

Обнаружено и удалено незначительное (менее 0,33% всех данных) количество дубликатов — 71.

Выделены шесть основных целей получения кредита: ремонт, свадьба, образование, автомобиль, жилье, недвижимость. С их помощью однозначно определены цели каждого заёмщика.

Заёмщики распределены по двум категориям — по наличию детей и по семи соразмерным категориям — в зависимости от уровня дохода.

Даны ответы на следующие вопросы:

1. Есть ли зависимость между наличием детей и возвратом кредита в срок? *Да. Отсутствие детей помогает вернуть кредит в срок.*
2. Есть ли зависимость между семейным положением и возвратом кредита в срок? *Да. Чаще всего кредиты не возвращают холостяки и живущие в гражданском браке. Реже всего — вдовцы / вдовы и разведённые.*
3. Есть ли зависимость между уровнем дохода и возвратом кредита в срок? *Да. Чаще всего возвращают в срок богатые заёмщики. Реже всего — заёмщики со средним доходом. Бедность напротив не препятствует возврату кредита.*
4. Как разные цели кредита влияют на его возврат в срок? *Наиболее рискованными являются кредиты на автомобиль, образование и свадьбу. Наиболее безопасными — кредиты на ремонт.*

Итоговый вывод:

Молодость и дети осложняют возврат кредита в срок. Бедность напротив не препятствует возврату кредита.