Исследование надежности заемщиков

Оглавление

Введение

Наш заказчик - кредитный отдел банка. Требуется выяснить, как влияет семейное положение и наличие детей на факт погашения кредита в срок. Результаты наших исследований будут учтены при построении модели кредитного скоринга.

Цель исследования

Ответить на следующие вопросы:

- Есть ли зависимость между количеством детей и возвратом кредита в срок?
- Есть ли зависимость между семейным положением и возвратом кредита в срок?
- Есть ли зависимость между уровнем дохода и возвратом кредита в срок?
- Как разные цели кредита влияют на его возврат в срок?
- Есть ли зависимость между образованием и возвратом кредита в срок?

Ход исследования

- Входные данные предоставил нам банк, это статистика о платежеспособности клиентов.
- Информации, какого они качества, у нас нет. Требуется самостоятельно проверить данные на пропуски, аномальные значения и дубликаты. И при необходимости исправить/дополнить данные
- Исследование пройдет в несколько этапов:
  - Обзор данных
  - Предобработка данных
  - Ответы на вопросы

1. Обзор данных

In [1]:
```
import pandas as pd

# для того чтобы код работал локально и на Практикуме применим конструкцию try-except
try:
    df = pd.read_csv('/datasets/data.csv')
except:
    df = pd.read_csv('datasets/data.csv')
```

In [2]:
```
# ознакомимся с таблицей визуально

display(df.head())
```

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_incc
0	1	-8437.673028	42	высшее	0	женат / замужем	0	F	сотрудник	0	253875.639
1	1	-4024.803754	36	среднее	1	женат / замужем	0	F	сотрудник	0	112080.014
2	0	-5623.422610	33	Среднее	1	женат / замужем	0	M	сотрудник	0	145885.952
3	3	-4124.747207	32	среднее	1	женат / замужем	0	M	сотрудник	0	267628.550
4	0	340266.072047	53	среднее	1	гражданский брак	1	F	пенсионер	0	158616.077

С ходу видно несколько проблем, возьмем их на заметку:

- `days_employed` — содержит отрицательные значения
- `education` — содержит одни и те же записи с разным регистром символов, это неявные дубликаты

Визуально с заголовками все в порядке, они написаны в нижнем регистре и используют **хороший_стиль**. На всякий случай проверим нет ли лишних пробелов.

In [3]:
```
# цикл обойдет каждый заголовок и проверит его на наличие пробела
# если пробел есть, проблемный заголовок отобразится в выводе

space_counter = 0
for heading in list(df.columns):
    if ' ' in heading: # если пробел есть в заголовке
        space_counter = 1
        print(f'В заголовке {heading} есть пробелы!') # выводим сообщение на экран вместе с этим заголовком

if space_counter == 0:
    print('В заголовках пробелов нет')
```

```
В заголовках пробелов нет
```

Все в порядке, действий с заголовками не требуется.

In [4]:
```
# изучим сводку о данных таблицы

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21525 entries, 0 to 21524
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   children          21525 non-null  int64
 1   days_employed     19351 non-null  float64
 2   dob_years         21525 non-null  int64
 3   education         21525 non-null  object
 4   education_id      21525 non-null  int64
 5   family_status     21525 non-null  object
```

```
 6   family_status_id  21525 non-null  int64
 7   gender            21525 non-null  object
 8   income_type       21525 non-null  object
 9   debt              21525 non-null  int64
 10  total_income      19351 non-null  float64
 11  purpose           21525 non-null  object
dtypes: float64(2), int64(5), object(5)
memory usage: 2.0+ MB
```

После осмотра можем сказать следующее:

- `days_employed` и `total_income` содержат пропуски, с этим будем разбираться
- Строковая информация соответсвует своему типу
- Числовая информация соответсвует своему типу
- Значения в столбцах `days_employed` и `total_income` вполне можно округлить до целых чисел, так как точность до десятых значений в трудовом стаже и зарплате для нас в текущей ситуации избыточная и будет только мешать.

Согласно документации в таблице записана следующая информация:

- `children` — количество детей в семье
- `days_employed` — общий трудовой стаж в днях
- `dob_years` — возраст клиента в годах
- `education` — уровень образования клиента
- `education_id` — идентификатор уровня образования
- `family_status` — семейное положение
- `family_status_id` — идентификатор семейного положения
- `gender` — пол клиента
- `income_type` — тип занятости
- `debt` — имел ли задолженность по возврату кредитов
- `total_income` — ежемесячный доход
- `purpose` — цель получения кредита

Выводы

- Количество данных для решения нашей задачи на первый взгляд достаточно.
- В таблице есть пропущенные значения и аномалии, которые необходимо исправить.
- Неплохо будет округлить часть числовых данных для удобства.
- Столбцы `education` и `family_status` имеют собственный `id`, возможно имеет смысл выделить эти данные в отдельные таблицы-справочники.

2. Предобработка данных

2.1 Заполнение пропусков

In [5]:
```
# ознакомимся с пропусками визуально

display(df[df['days_employed'].isna()].head())
```

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_inco
12	0	NaN	65	среднее	1	гражданский брак	1	M	пенсионер	0	I
26	0	NaN	41	среднее	1	женат / замужем	0	M	госслужащий	0	I
29	0	NaN	63	среднее	1	Не женат / не замужем	4	F	пенсионер	0	I
41	0	NaN	50	среднее	1	женат / замужем	0	F	госслужащий	0	I
55	0	NaN	54	среднее	1	гражданский брак	1	F	пенсионер	1	I

По всей видимости пропуски имеют `NaN` тип, что неплохо, так как это не будет нам крашить выполнение кода при математических операциях.

Еще в столбце `family_status` обнаружалась заглавная буква в написании. Неявный дубликат это или нет, пока непонятно. Возьмем этот момент на заметку и проверим позже.

In [6]:
```
# подсчитаем общее количество пропусков

df.isna().sum()
```

Out[6]:
```
children               0
days_employed       2174
dob_years              0
education              0
education_id           0
family_status          0
family_status_id       0
gender                 0
income_type            0
debt                   0
total_income        2174
purpose                0
dtype: int64
```

В столбце `days_employed` и `total_icome` обнаружено по **2174** пропущенных значений. По количеству они совпадают. Чтобы убедится, что пропуски находятся в одних и тех же строках проверим это.

In [7]:
```
# получаем индексы NaN значений в столбце 'days_employed'
# с помощью логической индексации подсчитываем количество NaN в столбце 'total_income'

print(
    'Количество NaN в столбце "total_income" по индексам NaN столбца "days_employed" =',
    df[df['days_employed'].isna()]['total_income'].isna().sum()
)
```

```
Количество NaN в столбце "total_income" по индексам NaN столбца "days_employed" = 2174
```

Количество значений `NaN` в столбце `total_income` по тем же индексам равно **2174**. Что подтверждает тот факт, что пропущенные значения столбцов `days_employed` и `total_icome` находятся в одинаковых строках

Пропущенные значения находятся только в столбцах отображающих трудовой стаж и заработную плату. Так как пропуски в одних и тех же строках, можем предположить, что данные не указанны по одной из возможных причин:

- клиенты намеренно не указали данные о своей трудовой деятельности
- данные о трудовой деятельности потерялись при переносе из других таблиц (например не подтянулся корректный id клиента)
- возможно это особенные клиенты и данные о их работе и доходах не указаны по какому-то соглашению

Найдем долю пропущенных значений. Так как количество пропущенных значений в двух столбцах одинаково, для рассчета можем использовать любой из них

In [8]:
```
all_values = df.shape[0] # найдем количество всех строк таблицы
nan_values = df['days_employed'].isna().sum() # найдем количество строк с NaN значениями
part_nan_values = nan_values / all_values # рассчитаем долю пропусков от общего количества

print(f'Общее количество строк в таблице: {all_values}')
print(f'Количество строк с пропущенными значениями: {nan_values}')
print(f'Доля пропущенных значений от общего числа: {part_nan_values:.0%}')
```

```
Общее количество строк в таблице: 21525
Количество строк с пропущенными значениями: 2174
Доля пропущенных значений от общего числа: 10%
```

Как видим доля `NaN` в стобцах `days_employed` и `total_icome` составляет целых **10%** от общего количества, что довольно весомо. Посмотрим как сильно это отразится на результатах исследования.

Для начала обработаем `NaN` значения в столбце `total_income` .

В этом столбце отображается доход клиента. Так как заработная плата может иметь сильное расхождение в значениях, например: **10** клиентов могут получать по **30.000 руб.**, а один клиент **1.000.000 руб.** то более реальную картину будет показывать **медианное** значение.

In [9]:
```
# Заполним медианным значением пропуски в 'total_income'
```

```
df.loc[df['total_income'].isna(), 'total_income'] = df['total_income'].median()

print('Новые значения столбца "total_income" вместо NaN:\n')
print(df[df['days_employed'].isna()]['total_income'].head())
```

```
Новые значения столбца "total_income" вместо NaN:

12    145017.937533
26    145017.937533
29    145017.937533
41    145017.937533
55    145017.937533
Name: total_income, dtype: float64
```

In [10]:
```
# Проверим что все пропуски столбца "total_income" заполнены

df.isna().sum()
```

Out[10]:
```
children               0
days_employed       2174
dob_years              0
education              0
education_id           0
family_status          0
family_status_id       0
gender                 0
income_type            0
debt                   0
total_income           0
purpose                0
dtype: int64
```

2.2 Проверка данных на аномалии и исправления

Мы уже выяснили, что в столбце **`days_employed`** присутствуют значения **`NaN`**. Но прежде чем заменим их, исследуем этот столбец на аномальные значения.

In [11]:
```
# для начала посмотрим на пограничные значения и общий срез

print('Минимальное значение в столбце "days_employed" =', df['days_employed'].min())
print('Максимальное значение в столбце "days_employed" =', df['days_employed'].max())
print('\nОбщий срез значений:')
print(df['days_employed'])
```

```
Минимальное значение в столбце "days_employed" = -18388.949900568383
Максимальное значение в столбце "days_employed" = 401755.40047533

Общий срез значений:
0         -8437.673028
1         -4024.803754
2         -5623.422610
3         -4124.747207
4        340266.072047
             ...
21520     -4529.316663
21521    343937.404131
21522     -2113.346888
21523     -3112.481705
21524     -1984.507589
Name: days_employed, Length: 21525, dtype: float64
```

В столбце присутствуют как **отрицательные** значения, так и **аномально большие**.

Для начала разберемся с **отрицательными** значениями.

Причина по которой появились отрицательные значения может заключаться в неверной конвертации текстовых данных. Например: данные о стаже хранились в текстовом виде с тире `-` перед значением. При конвертации в числовое значение это превратилось в отрицательное число.

In [12]:
```
# избавимся от отрицательных значений с помощью функции abs(), которая вернет нам модуль числа.
# для того чтобы наш код не крашнулся, если что-то пойдет не так, применим конструкцию try-except

try:
    df['days_employed'] = abs(df['days_employed'])
except:
    print('Код не отработал, в данных обнаружена ошибка!')
```

In [13]:
```
# Проверим результат выполнения запроса

print(df['days_employed'])
```

```
0          8437.673028
1          4024.803754
2          5623.422610
3          4124.747207
4        340266.072047
             ...
21520      4529.316663
21521    343937.404131
21522      2113.346888
21523      3112.481705
21524      1984.507589
Name: days_employed, Length: 21525, dtype: float64
```

Теперь займемся **аномально большими** значениями.

Попробуем выделить кластер проблемных значений. Для этого сделаем следующее:

- Предположим, что стаж может начинаться с 14 лет и не должен превышать разницу с текущим возрастом
- Итого расчет будет по формуле: $(\text{возраст} - 14) \cdot 365$
- Отсеим все значения таблицы где стаж превышает это значение

Так как в расчете будет участвовать столбец с возрастом, проверим нет ли там аномальных значений

In [14]:
```
# Проверим крайние значения возраста:

print(df['dob_years'].min())
print(df['dob_years'].max())
```

```
0
75
```

Как видим есть нулевой возраст. По сути это тоже пропущенные значения, вернемся к решению этого вопроса позже.

А пока исключим нули из расчета и на всякий случай проверим нет ли еще странных значений, отличных от нуля.

In [15]:
```
# узнаем минимальное не нулевое значение

print(df[df['dob_years'] != 0]['dob_years'].min())
```

```
19
```

Минимальный возраст в нашей анкете равен 19 годам, что вполне ложится в реальную картину.\ Выделим кластер проблемных значений и попробуем проанализировать его

In [16]:
```
# фильтр представляет из себя сдедующую логику:
# берем из таблицы только те значения, где возраст != 0 и стаж больше чем выражение: (возраст - 14) * 365

# чтобы сделать код более лаконичным, фильтрующую запись запишем в переменную

abnormal_filter = df[
    (df['dob_years'] != 0) &
    ((df['days_employed']) > ((df['dob_years'] - 14) * 365))]

display(abnormal_filter.head()) # ознакомимся с нашей выборкой
```

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_inc
4	0	340266.072047	53	среднее	1	гражданский брак	1	F	пенсионер	0	158616.07
18	0	400281.136913	53	среднее	1	вдовец / вдова	2	F	пенсионер	0	56823.77
24	1	338551.952911	57	среднее	1	Не женат / не замужем	4	F	пенсионер	0	290547.23
25	0	363548.489348	67	среднее	1	женат / замужем	0	M	пенсионер	0	55112.75

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_inc
30	1	335581.668515	62	среднее	1	женат / замужем	0	F	пенсионер	0	171456.06

В выборке почему то встречаются одни пенсионеры, посмотрим подробнее какие записи содержатся в этом столбце.

In [17]:
```
print(abnormal_filter['income_type'].value_counts())
```

```
пенсионер       3426
сотрудник         10
компаньон          5
безработный        2
госслужащий        1
Name: income_type, dtype: int64
```

Действительно, в выборке подавляющая часть пенсионеров, но встречаются и другие статусы, посмотрим подробнее на них. Так как значений не так много, выведем их все.

In [18]:
```
display(
    abnormal_filter[abnormal_filter['income_type'] != 'пенсионер'] # выведем всех Не пенсионеров
    .sort_values('days_employed', ascending=False)) # отсортируем по убыванию
```

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	tota
14798	0	395302.838654	45	Высшее	0	гражданский брак	1	F	безработный	0	20272
3133	1	337524.466835	31	среднее	1	женат / замужем	0	M	безработный	1	5995
16335	1	18388.949901	61	среднее	1	женат / замужем	0	F	сотрудник	0	18617
4299	0	17615.563266	61	среднее	1	женат / замужем	0	F	компаньон	0	12256
5581	0	15079.216069	55	среднее	1	женат / замужем	0	F	госслужащий	0	17876
8735	0	14240.932400	53	Среднее	1	женат / замужем	0	M	компаньон	0	17448
2492	0	13724.223884	50	высшее	0	женат / замужем	0	F	сотрудник	0	8815
5708	0	13210.485012	47	среднее	1	женат / замужем	0	F	сотрудник	0	14484
20061	0	12761.377792	48	среднее	1	женат / замужем	0	F	сотрудник	0	20002
397	0	12506.318296	46	среднее	1	женат / замужем	0	F	сотрудник	0	29284
3957	0	12111.680981	47	среднее	1	женат / замужем	0	F	компаньон	0	10840
15118	0	11037.198423	44	Среднее	1	женат / замужем	0	F	сотрудник	0	23465
5959	0	10939.299825	42	среднее	1	женат / замужем	0	F	сотрудник	0	7484
13531	0	9762.839918	39	среднее	1	женат / замужем	0	F	сотрудник	0	12985
12860	0	8946.243338	38	среднее	1	женат / замужем	0	F	компаньон	0	10441
19902	0	8436.151295	36	среднее	1	гражданский брак	1	F	сотрудник	0	9125

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	tota
14348	0	8076.147758	36	среднее	1	женат / замужем	0	F	компаньон	0	11187
3417	1	5673.867214	29	среднее	1	женат / замужем	0	M	сотрудник	0	28802

Аномальные значения есть только у двух безработных. Остальные товарищи каким-то образом смогли официально устроится на работу раньше чем в 14 лет. Что-ж, оставим это на их совести, а сами займемся нашими странными пенсионерами.

Выясним, все ли пенсионеры имеют аномальный стаж

In [19]:
```
print(df[df['income_type'] == 'пенсионер']['days_employed'].min()) # минимальный стаж пенсионера
print(df[df['income_type'] == 'пенсионер']['days_employed'].max()) # максимальный стаж пенсионера
```

```
328728.72060451825
401755.40047533
```

Бинго! Нарушители спокойствия: двое безработных и пенсионеры. У всех них стаж выше 300 тыс. дней. По тысяче лет люди не только не работают, но и не живут.

Гипотеза\ Возможно трудовой стаж данных клиентов попал в банковскую систему из учетной системы пенсионного фонда, где он хранится не в днях, а например в часах. Давайте проверим это. Разделим стаж на 24

In [20]:
```
# сделаем новый датафрейм где оставим только клиентов с аномальными значениями
abnormal_df = df[df['days_employed'] > 300000].copy()

# создадим столбец 'correct_days' с новыми значениями, для проверки гипотезы
abnormal_df['correct_days'] = abnormal_df['days_employed'] / 24

# создадим столбец с возрастом клиента в днях, для дополнительной проверки данных
abnormal_df['max_possible_days'] = abnormal_df['dob_years'] * 365

display(abnormal_df[['dob_years', 'days_employed', 'correct_days', 'max_possible_days']])
```

	dob_years	days_employed	correct_days	max_possible_days
4	53	340266.072047	14177.753002	19345
18	53	400281.136913	16678.380705	19345
24	57	338551.952911	14106.331371	20805
25	67	363548.489348	15147.853723	24455
30	62	335581.668515	13982.569521	22630
...	...	...	...	...
21505	53	338904.866406	14121.036100	19345
21508	62	386497.714078	16104.071420	22630
21509	59	362161.054124	15090.043922	21535
21518	59	373995.710838	15583.154618	21535
21521	67	343937.404131	14330.725172	24455

3445 rows × 4 columns

Результат похож на правду, новые данные выглядят вполне реальными. Для того, чтобы убедится, проверим не превышает ли трудовой стаж возраст самого клиента. Для этого воспользуемся нашим новым столбцом `max_possible_days`

In [21]:
```
# создадим переменную-фильтр, для более лаконичной записи кода
# Фильтр: (возраст !=0) And (возраст < трудовой стаж)

max_possible_filter = abnormal_df[
    (abnormal_df['dob_years'] !=0) &
    (abnormal_df['max_possible_days'] < abnormal_df['correct_days'])]

display(max_possible_filter[['dob_years', 'days_employed', 'correct_days', 'max_possible_days']].head())
```

	dob_years	days_employed	correct_days	max_possible_days
157	38	348414.028009	14517.251167	13870
751	41	390755.464054	16281.477669	14965
776	38	365336.560325	15222.356680	13870
1242	22	334764.259831	13948.510826	8030
1383	37	353802.811675	14741.783820	13505

Видно, что все-таки есть нереальные показатели. А счастье было так близко. Но не будем унывать, оценим масштаб проблемы.

In [22]:
```
print(f'Количество невозможных значений: {max_possible_filter.shape[0]}')
print(f'Количество всех скорректированных значений: {abnormal_df.shape[0]}')
print(f'Доля невозможных значений в корректировке: {max_possible_filter.shape[0] / abnormal_df.shape[0]:.1%}')
```

```
Количество невозможных значений: 54
Количество всех скорректированных значений: 3445
Доля невозможных значений в корректировке: 1.6%
```

Как видим, доля невозможных значений составляет всего **1.6%**. Все остальные значения выглядят вполне правдоподобно. Считаю что **гипотеза подтвердилась** и аномально большие значения в трудовом стаже вполне можно разделить на 24. Невозможными значениями пренебрежем.

Заменим аномальные значения в стаже, в нашем основном ДатаФрейме

In [23]:
```
# все что превышает 300.000 в 'days_employed' делим на 24

df.loc[df['days_employed'] > 300000, 'days_employed'] /= 24
```

In [24]:
```
# Проверим данные

print('Максимальное значение в столбце "days_employed":', df['days_employed'].max(), '\n')
print(df['days_employed'])
```

```
Максимальное значение в столбце "days_employed": 18388.949900568383

0         8437.673028
1         4024.803754
2         5623.422610
3         4124.747207
4        14177.753002
             ...
21520     4529.316663
21521    14330.725172
21522     2113.346888
21523     3112.481705
21524     1984.507589
Name: days_employed, Length: 21525, dtype: float64
```

Аномальные значения побеждены. Займемся **NaN**,ами

Так как показатели стажа могут очень сильно различаться по значениям, пропуски лучше заменить на **медианное** по столбцу.

In [25]:
```
df.loc[df['days_employed'].isna(), 'days_employed'] = df['days_employed'].median()
```

In [26]:
```
# проверим что пропущенных значений больше нет

df.isna().sum()
```

Out[26]:
```
children            0
days_employed       0
dob_years           0
education           0
education_id        0
family_status       0
family_status_id    0
gender              0
income_type         0
debt                0
total_income        0
```

```
purpose             0
dtype: int64
```

Ура! **NaN** значений в таблице больше нет!

Но остался **нулевой возраст**. Надо что-то с этим делать. Такое значение могло образоваться по причине того, что клиент не указал возраст и система автоматически проставила `0`.

Посмотрим на срез данных клиентов с нулевым возрастом:

In [27]:
```
display(df[df['dob_years'] == 0])
```

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	tota
99	0	14439.234121	0	Среднее	1	женат / замужем	0	F	пенсионер	0	7129
149	0	2664.273168	0	среднее	1	в разводе	3	F	сотрудник	0	7017
270	3	1872.663186	0	среднее	1	женат / замужем	0	F	сотрудник	0	10216
578	0	16577.356876	0	среднее	1	женат / замужем	0	F	пенсионер	0	9762
1040	0	1158.029561	0	высшее	0	в разводе	3	F	компаньон	0	30399
...	...	...	...	...	...	...	...	...	...	...	
19829	0	2194.220567	0	среднее	1	женат / замужем	0	F	сотрудник	0	14501
20462	0	14113.952856	0	среднее	1	женат / замужем	0	F	пенсионер	0	25919
20577	0	13822.552977	0	среднее	1	Не женат / не замужем	4	F	пенсионер	0	12978
21179	2	108.967042	0	высшее	0	женат / замужем	0	M	компаньон	0	24070
21313	0	1268.487728	0	среднее	1	Не женат / не замужем	4	M	сотрудник	0	15241

101 rows × 12 columns

Учитывая, что у нас есть данные о стаже. Можем заполнить пропуски по следующей формуле:

$$\frac{\text{Стаж}}{365} + 19$$

Таким образом мы получим более корректный возраст, чем просто среднее значение по столбцу

In [28]:
```
# для того чтобы заменить возраст по формуле напишем небольшую функцию
# функция будет обходить ДатаФрейм построчно

def dob_years_repair(row):
    if row['dob_years'] == 0: # если возраст равен нулю
        return int(row['days_employed'] / 365 + 19 ) # заменяем его на расчет формулы стаж/365 +19
    else:
        return row['dob_years'] # иначе оставляем старое значение
```

In [29]:
```
# вызовем функцию dob_years_repair. Для того чтобы код не крашнулся используем конструкцию try-except

try:
    df['dob_years'] = df.apply(dob_years_repair, axis=1)
except:
    print('Код не сработал, требуется проверить на ошибки!')
```

In [30]:
```
# Проверим, что нулевых значений больше нет
display(df[df['dob_years'] == 0])
```

```
# А так же выборочно проверим, как отработал код по строкам, где был нулевой возраст
display(df.loc[[99, 149, 270, 578, 1040]])
```

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income
99	0	14439.234121	58	Среднее	1	женат / замужем	0	F	пенсионер	0	71291
149	0	2664.273168	26	среднее	1	в разводе	3	F	сотрудник	0	70176
270	3	1872.663186	24	среднее	1	женат / замужем	0	F	сотрудник	0	102166
578	0	16577.356876	64	среднее	1	женат / замужем	0	F	пенсионер	0	97620
1040	0	1158.029561	22	высшее	0	в разводе	3	F	компаньон	0	303994

Получившиеся значения выглядят весьма правдоподобно.

Пройдемся по другим количественным показателям для поиска аномальных значений

In [31]:
```
# посмотрим какие значения есть в столбце "количество детей"

df['children'].value_counts()
```

Out[31]:
```
 0     14149
 1      4818
 2      2055
 3       330
 20       76
-1        47
 4        41
 5         9
Name: children, dtype: int64
```

Значения `-1` и `20` явно нездоровые. Подумаем как такое могло произойти и на что лучше их заменить.

- Значение `-1` могло появиться из-за ошибки в наборе, человек мол вбить `-` тире и `1`. Тут лучше всего просто заменить отрицательное значение на положительное
- Значение `20` могло появиться из-за случайного нажатия нуля на дополнительной клавиатуре. Как раз кнопка ноль расположена рядом с двойкой. Такая ошибка имеет место быть. Считаю что данную ошибку лучше всего заменить на `2`

In [32]:
```
abnomal_children = df[(df['children'] == -1) | (df['children'] == 20)]['children'].count()

print(f'Количество аномальных значений в столбце "children": {abnomal_children}')
print(f'Количество всех значений: {df.shape[0]}')
print(f'Доля аномальных значений в поле "children": {abnomal_children / df.shape[0]:.1%}')
```

```
Количество аномальных значений в столбце "children": 123
Количество всех значений: 21525
Доля аномальных значений в поле "children": 0.6%
```

Количество аномальных значений не превышает 1% можем избавиться от них.

In [33]:
```
# заменим все значения на модуль числа, чтобы избавится от отрицательных значений
# для защиты выполнения применим конструкцию try-except

try:
    df['children'] = abs(df['children'])
except:
    print('Код не отработал, в данных обнаружена ошибка!')

# все значения 20 заменим на 2
df.loc[df['children'] == 20, 'children'] = 2
```

In [34]:
```
# проверим результат наших запросов
```

```
df['children'].value_counts()
```

Out[34]:
```
0    14149
1     4865
2     2131
3      330
4       41
5        9
Name: children, dtype: int64
```

Пропуски и аномальные значения побеждены

Итоги работы с пропусками и аномальными значениями

- `total_income` - заменено **2174** `NaN` значений на **медианное** по столбцу
- `days_employed` - замена отрицательных значений на модуль числа
- `days_employed` - аномально большие значения (>300000) из часов переведены в дни путем деления на 24
- `days_employed` - заменено **2174** `NaN` значений на **медианное** по столбцу
- `dob_years` - заменили нулевой возраст по формуле: **стаж / 365 + 19**
- `children` - значения **-1** заменены на **1**, значения **20** заменены на **2**

2.3. Изменение типов данных

В столбцах `days_employed` и `total_income` используется **вещественный** тип данных. Для нашей задачи такая точность данных избыточна. По этому с чистой совестью можем сконвертировать эти значения в целые числа. Это облегчит нам визуальный анализ данных.

In [35]:
```
# применим функцию astype() чтобы сконвертировать наши значения

df['days_employed'] = df['days_employed'].astype('int32')
df['total_income'] = df['total_income'].astype('int32')
```

In [36]:
```
display(df.head(2))
```

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_incon
0	1	8437	42	высшее	0	женат / замужем	0	F	сотрудник	0	2538
1	1	4024	36	среднее	1	женат / замужем	0	F	сотрудник	0	1120

In [37]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21525 entries, 0 to 21524
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   children          21525 non-null  int64 
 1   days_employed     21525 non-null  int32 
 2   dob_years         21525 non-null  int64 
 3   education         21525 non-null  object
 4   education_id      21525 non-null  int64 
 5   family_status     21525 non-null  object
 6   family_status_id  21525 non-null  int64 
 7   gender            21525 non-null  object
 8   income_type       21525 non-null  object
 9   debt              21525 non-null  int64 
 10  total_income      21525 non-null  int32 
 11  purpose           21525 non-null  object
dtypes: int32(2), int64(5), object(5)
memory usage: 1.8+ MB
```

`int32` в данной ситуации нас вполне устроит. Вряд-ли у кого то найдется стаж или зарплата выше 2 млрд./мес.

2.4 Удаление дубликатов

Проведем разведку неявных дубликатов по столбцам с категориальными значениями.

Первым исследуем столбец "образование"

In [38]:
```
df['education'].value_counts()
```

Out[38]:
```
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
```

В столбце много схожих значений, но из заглавных и строчных символов целый винегрет. Возникновение подобных данных вполне естественный процесс, все люди разные и привыкли вбивать информацию по разному. Кто то с Большой буквы, а кто-то ВСЕ БОЛЬШИМИ. Как хорошо что для таких случаев у нас есть метод **`str.lower()`**

In [39]:
```
# приведм все в нижний регистр

df['education'] = df['education'].str.lower()
```

In [40]:
```
# опять посмотрим на данные

df['education'].value_counts()
```

Out[40]:
```
среднее                15233
высшее                  5260
неоконченное высшее      744
начальное                282
ученая степень             6
Name: education, dtype: int64
```

Теперь все в порядке.

Изучим столбец **`education_id`**

In [41]:
```
df['education_id'].value_counts()
```

Out[41]:
```
1    15233
0     5260
2      744
3      282
4        6
Name: education_id, dtype: int64
```

Количество уникальных значений полностью совпадает со столбцом **`education`**. Значит можно будет использовать связку этих данных, чтобы выделить отдельную таблицу-спарвочник. Вернемся к этому позже.

Следующий на очереди **`family_status`**

In [42]:
```
df['family_status'].value_counts()
```

Out[42]:
```
женат / замужем          12380
гражданский брак          4177
Не женат / не замужем     2813
в разводе                 1195
вдовец / вдова             960
Name: family_status, dtype: int64
```

В одном из значений обнаружилась заглавная буква. Не смотря на то, что это не создало нам неявный дубликат, от греха подальше привем все это в нижний регистр.

In [43]:
```
df['family_status'] = df['family_status'].str.lower()
```

In [44]:
```
df['family_status'].value_counts()
```

Out[44]:
```
женат / замужем          12380
гражданский брак          4177
не женат / не замужем     2813
в разводе                 1195
вдовец / вдова             960
Name: family_status, dtype: int64
```

Так-то лучше. Неявных дубликатов нет.

Теперь очередь **family_status_id**

In [45]:
```
df['family_status_id'].value_counts()
```

Out[45]:
```
0    12380
1     4177
4     2813
3     1195
2      960
Name: family_status_id, dtype: int64
```

Опять видим что **id** по количеству уникальных значений совпадают с одноименным столбцом. Сделаем из этой связки вторую таблицу-справочник.

Теперь проверим тип занятости **income_type**

In [46]:
```
df['income_type'].value_counts()
```

Out[46]:
```
сотрудник          11119
компаньон           5085
пенсионер           3856
госслужащий         1459
безработный            2
предприниматель        2
студент                1
в декрете              1
Name: income_type, dtype: int64
```

Тут все в полном порядке.

Далее цель кредита **purpose**

In [47]:
```
df['purpose'].value_counts()
```

Out[47]:
```
свадьба                                  797
на проведение свадьбы                    777
сыграть свадьбу                          774
операции с недвижимостью                 676
покупка коммерческой недвижимости        664
покупка жилья для сдачи                  653
операции с жильем                        653
операции с коммерческой недвижимостью    651
покупка жилья                            647
жилье                                    647
покупка жилья для семьи                  641
строительство собственной недвижимости   635
недвижимость                             634
операции со своей недвижимостью          630
строительство жилой недвижимости         626
покупка недвижимости                     624
строительство недвижимости               620
покупка своего жилья                     620
ремонт жилью                             612
покупка жилой недвижимости               607
на покупку своего автомобиля             505
заняться высшим образованием             496
автомобиль                               495
сделка с подержанным автомобилем         489
свой автомобиль                          480
на покупку подержанного автомобиля       479
автомобили                               478
на покупку автомобиля                    472
дополнительное образование               462
приобретение автомобиля                  462
сделка с автомобилем                     455
высшее образование                       453
образование                              447
получение дополнительного образования    447
получение образования                    443
профильное образование                   436
```

```
получение высшего образования          426
заняться образованием                  412
Name: purpose, dtype: int64
```

Очень много схожих значений, но неявных дубликатов тут нет. Чуть позже вернемся к этому столбцу для категоризации.

Далее пол **gender**

In [48]:
```
df['gender'].value_counts()
```

Out[48]:
```
F      14236
M       7288
XNA        1
Name: gender, dtype: int64
```

Обнаружилось какое-то странное значение **XNA**. Посмотрим на него по подробнее.

In [49]:
```
df[df['gender'] == 'XNA']
```

Out[49]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	to
10701	0	2358	24	неоконченное высшее	2	гражданский брак	1	XNA	компаньон	0	

Сложно сказать что это за ошибка и откуда она взялась. Но так как значение всего одно, давайте вручную исправим его. Проанализируем данные:

- 24 года
- неоконченное высшее
- гражданский брак
- поиск недвижимости

Скорей всего это мужчина. Исправим это. Не смотря на то, что значение всего одно и можно исправить ячейку адресно, все равно используем логическую индексацию, так как если по какой то причине это значение уедет на другой индекс, мы поменяем не то что нужно.

In [50]:
```
# прогоняем весть столбце на замену XNA -> M. По итогу меняется только одно значение.

df.loc[df['gender'] == 'XNA', 'gender'] = 'M'
```

In [51]:
```
# проверим

df['gender'].value_counts()
```

Out[51]:
```
F    14236
M     7289
Name: gender, dtype: int64
```

Все отлично, теперь только мужчины и женщины!

Изучим наличие задолжностей **debt**

In [52]:
```
df['debt'].value_counts()
```

Out[52]:
```
0    19784
1     1741
Name: debt, dtype: int64
```

Тут все в порядке.

Проверим наш ДатаФрейм на явные дубликаты.

In [53]:
```
df.duplicated().sum()
```

Out[53]:
```
71
```

Полных дубликатов строк 71 шутка. Они молгили появиться из-за того, что предоставленную нам таблицу заполняли из разных источников. Просто удалим их с перезаписью индексов.

In [54]:
```
df = df.drop_duplicates().reset_index(drop=True)
```

```
df.duplicated().sum()
```

Out[54]:
```
0
```

Явные дубликаты удалены.

Итоги работы с дубликатами

- удалено **71** строки явных дубликатов
- `education` - все значения приведены в нижний регистр
- `family_status` - все значения приведены в нижний регистр
- `gender` - замена странного значения `XNA` на `M`

2.5 Формирование дополнительных датафреймов словарей, декомпозиция исходного датафрейма

Как мы уже обнаружили ранее в таблице присутсвуют избыточные данные по категориям, которые мы можем поместить в отдельные таблицы-словари, для того, чтобы уменьшить размер нашей основной таблицы, а заодно упростить выбор категории.

Для начала создадим таблицу с образованием.

In [55]:
```
education_dict = df[['education_id', 'education']].copy() # копируем 2 нужных столбца в новый ДатаФрейм
education_dict = education_dict.drop_duplicates().reset_index(drop=True) # оставляем только уникальные записи
display(education_dict)
```

	education_id	education
0	0	высшее
1	1	среднее
2	2	неоконченное высшее
3	3	начальное
4	4	ученая степень

Теперь очередь семейного статуса.

In [56]:
```
family_status_dict = df[['family_status_id', 'family_status']].copy() # копируем 2 нужных столбца в новый ДатаФрей
family_status_dict = family_status_dict.drop_duplicates().reset_index(drop=True) # оставляем только уникальные зап
display(family_status_dict)
```

	family_status_id	family_status
0	0	женат / замужем
1	1	гражданский брак
2	2	вдовец / вдова
3	3	в разводе
4	4	не женат / не замужем

Таблицы-словари `education_dict` и `family_status_dict` готовы.

Теперь удалим из нашего основного ДатаФрейма лишние столбцы.

In [57]:
```
df.drop(['family_status', 'education'], axis=1, inplace=True)
```

In [58]:
```
display(df.head())
```

	children	days_employed	dob_years	education_id	family_status_id	gender	income_type	debt	total_income	purpose
0	1	8437	42	0	0	F	сотрудник	0	253875	покупка жилья
1	1	4024	36	1	0	F	сотрудник	0	112080	приобретение автомобиля
2	0	5623	33	1	0	M	сотрудник	0	145885	покупка жилья

	children	days_employed	dob_years	education_id	family_status_id	gender	income_type	debt	total_income	purpose
3	3	4124	32	1	0	M	сотрудник	0	267628	дополнительное образование
4	0	14177	53	1	1	F	пенсионер	0	158616	сыграть свадьбу

Красота да и только.

Итоги работы с ДатаФреймами

- Создан ДатаФрейм-словарь `education_dict` - содержит тип образования и id для него
- Создан ДатаФрейм-словарь `family_status_dict` - содержит семейное положение id для него
- Из основного ДатаФрейма `df` удалены столбцы `education` и `family_status`

2.6 Категоризация дохода

In [59]:
```
print('Количество уникальных значений в столбце "total_income":',
      len(list(df['total_income'].unique())))
```

```
Количество уникальных значений в столбце "total_income": 18606
```

В таблице есть информация о доходе, но там 18606 уникальных значний. Работать с такой пёстрой выборкой достаточно проблемно. Для того чтобы упростить нам задачу, категоризируем типы дохода и присвоим им статусы ABCDE, в следующем виде:

- `E` 0 – 30.000
- `D` 30.001 – 50.000
- `C` 50.001 – 200.000
- `B` 200.001 – 1.000.000
- `A` 1.000.001 и выше

In [60]:
```
# напишем функцию, которая будет проверять ячейку и возвращать значение по следующему алгоритму

def categorise_total_income(cell):
    if  0 <= cell <=30000:
        return 'E'
    elif 30001 <= cell <= 50000:
        return 'D'
    elif 50001 <= cell <= 200000:
        return 'C'
    elif 200001 <= cell <= 1000000:
        return 'B'
    elif 1000001 <= cell:
        return 'A'
```

In [61]:
```
# применим нашу функцию с столбцу "total_income".
# результат работы функции поместим в новый столбец "total_income_category"

df['total_income_category'] = df['total_income'].apply(categorise_total_income)
```

In [62]:
```
# посмотрим что вышло

df[['total_income', 'total_income_category']]
```

Out[62]:

	total_income	total_income_category
0	253875	B
1	112080	C
2	145885	C
3	267628	B
4	158616	C
...	...	...
21449	224791	B
21450	155999	C

	total_income	total_income_category
21451	89672	C
21452	244093	B
21453	82047	C

21454 rows × 2 columns

In [63]:
```
df['total_income_category'].value_counts()
```

Out[63]:
```
C    16016
B     5041
D      350
A       25
E       22
Name: total_income_category, dtype: int64
```

Все получилось. Теперь у нас есть столбец всего с пятью уникальными значениями вместо 18 тысяч. Ориентироваться по нему будет намного проще.

2.7 Категоризация целей кредита

В графе **`purpose`** записано много вариантов и комбинаций слов, которые в итоге сводятся всего к четырем категориям:

- операции с автомобилем
- операции с недвижимостью
- проведение свадьбы
- получение образования

Напишем функцию, которая анализирует информацию и сортирует все эти записи по четерем категориям

In [64]:
```
# посмотрим на формат записи

df['purpose'].value_counts()
```

Out[64]:
```
свадьба                                   791
на проведение свадьбы                     768
сыграть свадьбу                           765
операции с недвижимостью                  675
покупка коммерческой недвижимости         661
операции с жильем                         652
покупка жилья для сдачи                   651
операции с коммерческой недвижимостью     650
покупка жилья                             646
жилье                                     646
покупка жилья для семьи                   638
строительство собственной недвижимости    635
недвижимость                              633
операции со своей недвижимостью           627
строительство жилой недвижимости          624
покупка недвижимости                      621
покупка своего жилья                      620
строительство недвижимости                619
ремонт жилью                              607
покупка жилой недвижимости                606
на покупку своего автомобиля              505
заняться высшим образованием              496
автомобиль                                494
сделка с подержанным автомобилем          486
свой автомобиль                           478
на покупку подержанного автомобиля        478
автомобили                                478
на покупку автомобиля                     471
приобретение автомобиля                   461
дополнительное образование                460
сделка с автомобилем                      455
высшее образование                        452
образование                               447
получение дополнительного образования     446
получение образования                     442
профильное образование                    436
получение высшего образования             426
заняться образованием                     408
Name: purpose, dtype: int64
```

Видно, что по сути в цели кредита каждый раз встречаются одинаковые корни **авто** **образ** **свадь** **жиль** **недв** . И самое хорошее для нас, что они не конфликтуют друг с другом. Приступим к написанию функции.

In [65]:
```
# создадим 4 списка, для каждой из категорий и впишем туда необходимые корни слов

auto_list = ['авто'] # автомобильный
education_list = ['образ'] # образование
realty_list = ['жиль', 'недв'] # недвижимость
wedding_list = ['свадь'] # свадьба

# сначала создадим функцию-помощника, которая будет перебирать список корней
# и проверять вхождение этого корня в строковое значение

def str_in_val_list(str_val, words): # принимаем на вход строковое значение для проверки и список корней
    for word in words: # для каждого корня в списке корней
        if word in str_val: # если корень входит в строковое значение
            return True # возвращаем True
    return False # иначе False


# теперь создаем основную рабочую функцию

def categorise_purpose(cell): # принимаем на вход значение ячейки (это наше строковое значение)
    if str_in_val_list(cell, auto_list): # вызываем функцию-помощника, которой передаем значение ячейки и список н
        return 'операции с автомобилем' # если функция-помощник возвращает нам True, то основная возвращает нужную
    elif str_in_val_list(cell, realty_list):
        return 'операции с недвижимостью'
    elif str_in_val_list(cell, wedding_list):
        return 'проведение свадьбы'
    elif str_in_val_list(cell, education_list):
        return 'получение образования'
    else:
        return 'без категории' # если все вызовы функций-помощников вернули нам False, то получаем на выходе "без


# применим нашу функцию с столбцу "purpose".
# результат работы функции поместим в новый столбец "purpose_category"

df['purpose_category'] = df['purpose'].apply(categorise_purpose)
```

In [66]:
```
# проверим результат работы

print(df['purpose_category'].value_counts())
```

```
операции с недвижимостью    10811
операции с автомобилем       4306
получение образования        4013
проведение свадьбы           2324
Name: purpose_category, dtype: int64
```

Раз значения **без категории** нет, значит функция нашла все варианты значений и раскидала их по категориям. Это как раз то, что нам и было нужно.

In [67]:
```
# посмотрим новый столбец

df[['purpose', 'purpose_category']]
```

Out[67]:

	purpose	purpose_category
0	покупка жилья	операции с недвижимостью
1	приобретение автомобиля	операции с автомобилем
2	покупка жилья	операции с недвижимостью
3	дополнительное образование	получение образования
4	сыграть свадьбу	проведение свадьбы
...	...	...
21449	операции с жильем	операции с недвижимостью
21450	сделка с автомобилем	операции с автомобилем
21451	недвижимость	операции с недвижимостью
21452	на покупку своего автомобиля	операции с автомобилем

	purpose	purpose_category
21453	на покупку автомобиля	операции с автомобилем

21454 rows × 2 columns

Красота страшная сила. Теперь вместо кучи разнородных значений у нас всего 4 категории, которые передают основную суть цели кредита. Можно приступать к аналитической части проекта.

3. Ответы на вопросы

3.1 Есть ли зависимость между количеством детей и возвратом кредита в срок?

Требуется ответить на вопрос, как влияет наличие и количество детей на факт возврата кредита в срок.

Создадим сводную таблицу.

In [68]:
```
df_pivot_child = df.pivot_table(
    index=['children'], # индексы - информация о количестве детей у клиентов
    values=['debt'], # значения: 1 - имел задолженность, 0 - не имел задолженность
    aggfunc=['count', 'sum', 'mean'], # к значениям применим функции: количество, сумма, среднее значение
    fill_value=0)

display(df_pivot_child)
```

	count	sum	mean
	debt	debt	debt
children			
0	14091	1063	0.075438
1	4855	445	0.091658
2	2128	202	0.094925
3	330	27	0.081818
4	41	4	0.097561
5	9	0	0.000000

Вывод

Выборку клиентов с 5 детьми будем считать нерепрезентативной, так как в нашем наборе данных их слишком маленькое количество для составления статистических выводов.

По остальным клиентам четко видна закономерность, что клиенты без детей имеют наименьшее количество задолжностей. Причем чем детей больше, тем сильней ухудшается статистика.

Данную закономерность можно объяснить тем, что ребенок в семье это достаточно большая финансовая и временная нагрузка. И чем детей в семье больше, тем выше риск возникновения различных финансовых трудностей.

Особняком стоят клиенты с 3 детьми. Сложно сказать по какой причине тут статистика немного улучшилась. Возможно это аномалия чисто нашей выборки.

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

Требуется ответить на вопрос, как влияет семейное положение на факт возврата кредита в срок.

Создадим сводную таблицу.

In [69]:
```
df_pivot_family_status = df.pivot_table(
    index=['family_status_id'], # индексы - информация о семейном положение в виде id
    values=['debt'], # значения: 1 - имел задолженность, 0 - не имел задолженность
    aggfunc=['count', 'sum', 'mean'], # к значениям применим функции: количество, сумма, среднее значение
    fill_value=0)
```

```
display(df_pivot_family_status)
```

family_status_id	count debt	sum debt	mean debt
0	12339	931	0.075452
1	4151	388	0.093471
2	959	63	0.065693
3	1195	85	0.071130
4	2810	274	0.097509

Избаваимся от мультииндекса, чтобы не возникло проблем при сшивании таблиц. Так как мультииндек представляет из себя просто список кортежей, сошьем их с помощью метода `.join` и функции `map()`

In [70]:
```
# в качестве разделителя укажем "_" чтобы заголовки получились красивыми
df_pivot_family_status.columns = df_pivot_family_status.columns.map('_'.join)

# сбросим индекс, чтобы получить family_status_id в виде столбца
df_pivot_family_status = df_pivot_family_status.reset_index()

display(df_pivot_family_status)
```

	family_status_id	count_debt	sum_debt	mean_debt
0	0	12339	931	0.075452
1	1	4151	388	0.093471
2	2	959	63	0.065693
3	3	1195	85	0.071130
4	4	2810	274	0.097509

Так-то лучше, даже смотреть на таблицу в таком виде удобней.

In [71]:
```
# сошьем получившуюся таблицу с нашей таблицей-словерем "family_status_dict"
df_pivot_family_status_merged = df_pivot_family_status.merge(family_status_dict, on='family_status_id', how='left'

# выведем на экран в удобном виде
display(
    df_pivot_family_status_merged[['family_status_id', 'family_status', 'count_debt', 'sum_debt', 'mean_debt']]
    .sort_values('mean_debt', ascending=False))
```

	family_status_id	family_status	count_debt	sum_debt	mean_debt
4	4	не женат / не замужем	2810	274	0.097509
1	1	гражданский брак	4151	388	0.093471
0	0	женат / замужем	12339	931	0.075452
3	3	в разводе	1195	85	0.071130
2	2	вдовец / вдова	959	63	0.065693

Вывод

Самую плохую статистику показывают клиенты находящиеся в свободных отношениях и живущих в гражданском браке.

Ситуация резко меняется, если клиент находится в официальных отношениях. Видимо на это влияет совместный бюджет, а так же возросшая сознательность.

Далее клиенты находящиеся в разводе. Видимо в отличии от клиентов, которые находятся в свободных отношениях, у людей после официальных отношений остается повышенная сознательность. Как следствие они ответственнее относятся к финансовым обязательствам.

Лучше всего статистика у овдвовевших клиентов. Сложно сказать почему так получается. Возможно каким-то образом влияет унаследоованное имущество супруга/супруги.

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

Создадим сводную таблицу

In [72]:
```
df_pivot_income = df.pivot_table(
    index=['total_income_category'], # индексы - категория доходов
    values=['debt'], # значения: 1 - имел задолженность, 0 - не имел задолженность
    aggfunc=['count', 'sum', 'mean'], # к значениям применим функции: количество, сумма, среднее значение
    fill_value=0)

display(df_pivot_income)
```

	count	sum	mean
	debt	debt	debt
total_income_category			
A	25	2	0.080000
B	5041	356	0.070621
C	16016	1360	0.084915
D	350	21	0.060000
E	22	2	0.090909

Памятка, о распределении категорий:

- **A** 1.000.001 и выше
- **B** 200.001 – 1.000.000
- **C** 50.001 – 200.000
- **D** 30.001 – 50.000
- **E** 0 – 30.000

Вывод

Выборку с категорией доходов до 30.000 (E) и выше 1.000.000 (A) можно считать нерепрезентативной из-за недостаточного количества примеров.

Что же касается остальных категорий, лучше всего себя показывают клиенты с небольшим доходом 30-50 тыс.

Дальше идут более обеспеченные клиенты с доходом 50-200 тыс. где статистика резко ухудшается. Видимо с более высоким доходом быстро растут и аппетиты людей. Что плохо сказывается на возвращаемости долгов.

В катеории 200 тыс. - 1 млн. статистика опять выравнивается и выглядит намного лучше чем у клиентов с доходом 50-200 тыс.

Но по итогу самые беспроблемными заемщиками являются клиенты с невыским доходом 30-50 тыс.

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

In [73]:
```
df_pivot_purpose = df.pivot_table(
    index=['purpose_category'], # индексы - информация о цели кредита
    values=['debt'], # значения: 1 - имел задолженность, 0 - не имел задолженность
    aggfunc=['count', 'sum', 'mean'], # к значениям применим функции: количество, сумма, среднее значение
    fill_value=0)

display(df_pivot_purpose.sort_values(('mean', 'debt')))
```

	count	sum	mean
	debt	debt	debt
purpose_category			
операции с недвижимостью	10811	782	0.072334
проведение свадьбы	2324	186	0.080034

	count	sum	mean
	debt	debt	debt
purpose_category			
получение образования	4013	370	0.092200
операции с автомобилем	4306	403	0.093590

Вывод

Самая высокая возвращаемость долгов у клиентов с недвижимостью. Что вполне объяснимо, за долги банк пустит с молотка купленную квартиру и встанет вопрос где теперь жить.

Примерно такая же хорошая возвращаемость у клиентов, которые берут кредит на свадьбу. Видимо доход двух супругов положительно сказывается на возвращаемости долгов.

Резко статистика падает у клиентов бравших кредит на образование. Видимо связано это с тем, что учеба могла не оправдать ожиданий. Человека отчислили, либо он не получил желаемую работу, после чего сильно падает мотивация возвращать долг.

Самая плохая статистика у клиентов с автокредитом. Автомобиль вещь весьма финансово накладная. Видимо данные клиенты плохо прогнозируют во что может вылится покупка в перспективе, так как купить автомобиль мало, на обслуживание уходит солидная сумма денег.

3.5 Есть ли зависимость между образованием и возвратом кредита в срок?

In [74]:
```
df_pivot_education = df.pivot_table(
    index=['education_id'], # индексы - информация об образовании в виде id
    values=['debt'], # значения: 1 - имел задолженность, 0 - не имел задолженность
    aggfunc=['count', 'sum', 'mean'], # к значениям применим функции: количество, сумма, среднее значение
    fill_value=0)

display(df_pivot_education)
```

	count	sum	mean
	debt	debt	debt
education_id			
0	5250	278	0.052952
1	15172	1364	0.089902
2	744	68	0.091398
3	282	31	0.109929
4	6	0	0.000000

Избаваимся от мультииндекса, чтобы не возникло проблем при сшивании таблиц. Так как мультииндек представляет из себя просто список кортежей, сошьем их с помощью метода `.join` и функции `map()`

In [75]:
```
# в качестве разделителя укажем "_" чтобы заголовки получились красивыми
df_pivot_education.columns = df_pivot_education.columns.map('_'.join)

# сбросим индекс, чтобы получить education_id в виде столбца
df_pivot_education = df_pivot_education.reset_index()

display(df_pivot_education)
```

	education_id	count_debt	sum_debt	mean_debt
0	0	5250	278	0.052952
1	1	15172	1364	0.089902
2	2	744	68	0.091398
3	3	282	31	0.109929
4	4	6	0	0.000000

In [76]:
```
# сошьем получившуюся таблицу с нашей таблицей-словерем "education_dict"
df_pivot_education_merged = df_pivot_education.merge(education_dict, on='education_id', how='left')

# выведем на экран в удобном виде
display(
    df_pivot_education_merged[['education_id', 'education', 'count_debt', 'sum_debt', 'mean_debt']]
    .sort_values('mean_debt'))
```

	education_id	education	count_debt	sum_debt	mean_debt
4	4	ученая степень	6	0	0.000000
0	0	высшее	5250	278	0.052952
1	1	среднее	15172	1364	0.089902
2	2	неоконченное высшее	744	68	0.091398
3	3	начальное	282	31	0.109929

Вывод

Видим четкую закономерность **Уровень образования → Возвращаемость кредита**.

Хоть и выборка клиентов с **ученой степернью** очень мала для нормального участия в сравнении, думаю, что по итогу статистика таких клиентов будет либо равна либо лучше чем у клиентов с **высшим** образованием.

4. Итоги исследования

4.1 Общая информация

При осмотре предоставленного датасета выяснилось, что он содержит следубщие проблемы:

- Пропущенные значения
- Аномальные значения
- Дубли данных

В таком виде проводить исследование было неприемлемо, потребовалось выполнить предобработку данных.

4.2 Предобработка

Во время предобработки были решены следующие проблемы:

Пропуски и аномальные значения:

- `total_income` - заменено **2174** `NaN` значений на **медианное** по столбцу
- `days_employed` - замена отрицательных значений на модуль числа
- `days_employed` - аномально большие значения (>300000) из часов переведены в дни путем деления на 24
- `days_employed` - заменено **2174** `NaN` значений на **медианное** по столбцу
- `dob_years` - заменили нулевой возраст по формуле: **стаж / 365 + 19**
- `children` - значения **-1** заменены на **1**, значения **20** заменены на **2**

Дубликаты:

- удалено **71** строки явных дубликатов
- `education` - все значения приведены в нижний регистр
- `family_status` - все значения приведены в нижний регистр
- `gender` - замена странного значения `XNA` на `M`

4.3 Подготовка данных к исследованию

Для удобства исследования была проведена декомпозиция исходного ДатаФрейма, а так-же созданы два дополнительных ДатаФрейма-словаря:

- Создан ДатаФрейм-словарь `education_dict` - содержит тип образования и id для него

- Создан ДатаФрейм-словарь `family_status_dict` - содержит семейное положение id для него
- Из основного ДатаФрейма `df` удалены столбцы `education` и `family_status`

4.4 Результаты исследования:

- 1. Есть ли зависимость между количеством детей и возвратом кредита в срок?
  - Количество детей напрямую влияет на возвращаемость кредитов. Лучше всего возворащают кредиты клиенты без детей. Чем больше детей, тем хуже статистика возвращаемости кредитов.
  - Процент задолжавших:
  - Без детей - 7.5%
  - 1 ребенок - 9.1%
  - 2 ребенка - 9.4%
  - 3 ребенка - 8.1%
  - 4 ребенка - 9.7%
- 2. Есть ли зависимость между семейным положением и возвратом кредита в срок?
  - Семейное положение вляет на возвращаемость кредитов
  - Хуже всего статистика у людей в свободных отношениях. Процент задолжавших - 9.7%
  - Далее идут клиенты живущие в гражданском браке. Процент задолжавших - 9.3%
  - Средняя возвращаемость у людей находящихся в браке. Процент задолжавших - 7.5%
  - Хорошая возвращаемость у клиентов в разводе. Процент задолжавших - 7.1%
  - Лучшая возвращаемость у вдов/вдовцов. Процент задолжавших - 6.5%
- 3. Есть ли зависимость между уровнем дохода и возвратом кредита в срок?
  - Есть неявная зависимость
  - Лучше всего кредиты возвращают клиенты с небольшим доходом 30-50 тыс. Процент задолжавших - 6%
  - Средние показатели у хорошо обеспеченных клиентов, доход 200 тыс. - 1 млн. Процент задолжавших - 7%
  - Самая плохая статистика у средне обеспеченных клиентов, доход 50-200 тыс. Процент задолжавших - 8.4%
- 4. Как разные цели кредита влияют на его возврат в срок?
  - Лучшую возвращаемость показывают клиенты берущие кредит на недвижимость. Процент задолжавших - 7.2%
  - Примерно на том же уровне возвращаемость у клиентов ерущих кредит на свадьбу, Процент задолжавших - 7.9%
  - Резко хуже статистика у клиентов берущих кредиты на образование. Процент задолжавших - 9.2%
  - Так же в плачевном состоянии возвращаемость авто-кредитов. Процент задолжавших - 9.3%
- 5. Есть ли зависимость между образованием и возвратом кредита в срок?
  - Есть прямая зависимость от образования. Чем хуже образование, тем выше процент задолжавших клиентов.
  - Высшее - 5.2%
  - Среднее - 8.9%
  - Неоконченное высшее - 9.1%
  - Начальное - 10.9%