Исследование надежности заемщиков

Содержание

Заказчиком исследования является кредитный отдел банка.

На основе полученной статистики предполагается выявить взаимосвязь между платёжеспособностью клиентов и их семейным положением, наличием и количеством количеством их детей. Результаты исследования заказчик предполагает учитывать при построении модели кредитного скоринга.

Цель исследования - проверка четырёх гипотез:

1. Количество детей влияет на возврат кредита в срок.
2. Между семейным положением и возвратом кредита в срок существует зависимость.
3. Уровень дохода влияет на возврат кредита в срок.
4. Существует зависимость между целью кредита и его возвратом в срок.

Ход исследования

Предоставленные банком данные - статистика платёжеспособности клиентов - хранятся в файле `data.csv`. О качестве данных ничего не известно, поэтому перед проверкой гипотез потребуется обзор данных.

Необходимо проверить данные на ошибки, оценить их влияние на исследование. Затем, на этапе предобработки следует попытаться исправить самые критичные ошибки в данных.

Примерный план исследования выглядит следующим образом:

1. Обзор данных.
2. Предобработка данных.
3. Проверка гипотез.
4. Выводы исследования.

Обзор данных

Составим первое представление о данных, предоставленных банком. Для этого импортируем библиотеку `pandas`, прочитаем данные из файла `data.csv` в папке `/datasets` в датафрейм `borrowers` (заёмщики) и выведем его первые 10 строк.

Замечание. Дополнительно импортируем библиотеку `math` для работы с числами.

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

try:
    borrowers = pd.read_csv('/datasets/data.csv')
except:
    borrowers = pd.read_csv('data.csv')

borrowers.head(10)
```

Out[1]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income	purpose
0	1	-8437.673028	42	высшее	0	женат / замужем	0	F	сотрудник	0	253875.639453	покупка жилья
1	1	-4024.803754	36	среднее	1	женат / замужем	0	F	сотрудник	0	112080.014102	приобретение автомобиля
2	0	-5623.422610	33	Среднее	1	женат / замужем	0	M	сотрудник	0	145885.952297	покупка жилья
3	3	-4124.747207	32	среднее	1	женат / замужем	0	M	сотрудник	0	267628.550329	дополнительное образование
4	0	340266.072047	53	среднее	1	гражданский брак	1	F	пенсионер	0	158616.077870	сыграть свадьбу
5	0	-926.185831	27	высшее	0	гражданский брак	1	M	компаньон	0	255763.565419	покупка жилья
6	0	-2879.202052	43	высшее	0	женат / замужем	0	F	компаньон	0	240525.971920	операции с жильем
7	0	-152.779569	50	СРЕДНЕЕ	1	женат / замужем	0	M	сотрудник	0	135823.934197	образование
8	2	-6929.865299	35	ВЫСШЕЕ	0	гражданский брак	1	F	сотрудник	0	95856.832424	на проведение свадьбы
9	0	-2188.756445	41	среднее	1	женат / замужем	0	M	сотрудник	0	144425.938277	покупка жилья для семьи

Таблица содержит в себе достаточно большое количество данных о заёмщиках. Рассмотрим общую информацию о ней:

In [2]:
```
# Получение общей информации о данных в таблице borrowers
borrowers.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
```

Из полученных данных видно, что в таблице 12 столбцов, 21 525 строк. Встречаются следующие типы данных: `int64`, `float64`, `object`.

Столбцы поименованы корректно, в едином стиле. Количество непустых значений в столбцах разнится: в столбцах `'days_employed'` и `'total_income'` наблюдаются пропуски данных.

Согласно полученной документации к данным:

- `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.

Определим типы данных в различных столбцах:

- количественные данные используются в столбцах
  - `children` - целое число;
  - `days_employed` - число с плавающей точкой;
  - `dob_years` - целое число;
  - `total_income` - число с плавающей точкой;
- категорийные данные используются в столбцах
  - `education` - строковое значение;
  - `education_id` - выраженный целым числом идентификатор уровня образования;
  - `family_status` - строковое значение;
  - `family_status_id` - выраженный целым числом идентификатор семейного положения;
  - `gender` - строковое значение;
  - `income_type` - строковое значение;
  - `debt` - выраженный целым числом, предположительно, булевский идентификатор наличия задолженности по возврату кредитов;
  - `purpose` - строковое значение.

Выясним, какие множества значений могут принимать перечисленные идентификаторы, чтобы подтвердить предположение об их типе.

In [3]:
```
# Уникальные значения в столбце 'education_id'
borrowers['education_id'].unique()
```

Out[3]:
```
array([0, 1, 2, 3, 4])
```

In [4]:
```
# Уникальные значения в столбце 'family_status_id'
borrowers['family_status_id'].unique()
```

Out[4]:
```
array([0, 1, 2, 3, 4])
```

In [5]:
```
# Уникальные значения в столбце 'debt'
borrowers['debt'].unique()
```

Out[5]:
```
array([0, 1])
```

Итак, в таблице представлены 5 категорий уровня образования, 5 категорий семейного положения и булевская категория - наличие задолженности.

Выводы

В каждой строке представленной таблицы данных содержится запись о клиенте банка. Часть колонок описывает самого клиента, часть носит избыточный, вспомогательный характер (например, колонки `'education'` и `'education_id'`, а также `'family_status'` и `'family_status_id'` по сути содержат одинаковую информацию), часть описывает характеристики кредитной истории заёмщика.

Избыточность представляется целесообразным удалить из таблицы, сформировав словари категорий. Это может ускорить последующую обработку данных.

Предварительно можно утверждать, что данных достаточно для проверки гипотез. Но встречаются пропуски, которые необходимо устранить.

Пропуски в данных наблюдаются только в колонках, описывающих клиента (`'days_employed'` и `'total_income'`), значения которых выражены числами с плавающей точкой.

Кроме того, в колонке `'days_employed'` уже в начале датафрейма наблюдаются положительные и отрицательные числа, что противоречит смыслу значений данной колонки. Данные наблюдения могут свидетельствовать о технологических ошибках выборки и преобразования данных. *Следует обратить на это внимание инженеров по данным.*

Также нельзя сделать однозначный вывод о соответствии булевых значений в столбце `'debt'` фактам наличия и отсутствия задолженности. ***Из общих соображений, для целей настоящего исследования будем предполагать, что значением `0` кодируется отсутствие, а значением `1` - наличие задолженности.*** *Тем не менее, на данный недостаток следует обратить внимание инженеров по данным.*

Столбец `'education'` содержит различные стили написания одинаковых категорий сведений об образовании (использование букв в различном регистре). Представляется целесообразным устранить его на этапе предобработки, а также проверить столбцы `'family_status'`, `'income_type'`, `'purpose'` на наличие неявных дубликатов указанного свойства.

Чтобы двигаться дальше, устраним найденные проблемы в данных.

Предобработка данных

Явными проблемами в данных, обнаруженными на этапе обзора, являются пропуски в столбцах `'days_employed'` и `'total_income'`, а также наличие отрицательных значений в солбце `'days_employed'`.

Данные столбцы содержат вещественные значения и являются количественными, поскольку значения в них можно сравнивать. Обычно пропуски в количественных данных заполняют средним или медианным значением. Среднее лучше подходит для величин, разброс значений которых невелик. Для величин с большим разбросом возможных значений, когда единичный выброс может исказить среднее значение большого количества наблюдений, лучше определять медиану - величину, разделяющую наблюдения на 2 равномощных множества: со значениями больше и меньше медианного.

Оценим разброс значений в столбцах `'days_employed'` и `'total_income'`.

In [6]:
```
# Оценка разброса значений в столбце 'days_employed'
print("MAX значение в 'days_employed': {0}\nMIN значение в 'days_employed': {1}".
      format(borrowers['days_employed'].max(), borrowers['days_employed'].min()))
```

```
MAX значение в 'days_employed': 401755.40047533
MIN значение в 'days_employed': -18388.949900568383
```

In [7]:
```
# Оценка разброса значений в столбце 'total_income'
print("MAX значение в 'total_income': {0}\nMIN значение в 'total_income': {1}".
      format(borrowers['total_income'].max(), borrowers['total_income'].min()))
```

```
MAX значение в 'total_income': 2265604.028722744
MIN значение в 'total_income': 20667.26379327158
```

Значения в столбце `'total_income'` различаются в 100 раз. Это большой разброс, поэтому для заполнения пропусков в нём будем использовать медианное значение.

Большие отрицательные значения в столбце `'days_employed'`, полученные, вероятно, вследствие технической ошибки, либо ошибки машинного представления данных, могут существенным образом исказить значение медианы, поэтому перед заполнением пропусков в нём необходимо избавиться от отрицательных значений и повторно оценить разброс.

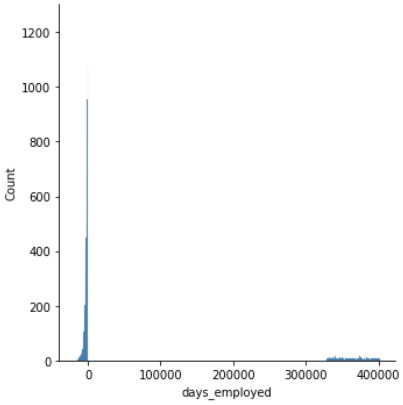
Также не понятно происхождение больших положительных значений 400000 дней - это более 1095 лет. Это не нормально. Необходимо определить количество и локализацию этих "выбросов".

Устранение аномалий данных

Начнём с анализа значений в столбце `'days_employed'`. С помощью библиотеки `seaborn` построим гистограмму значений столбца:

In [8]:
```
# Импорт библиотеки seaborn
import seaborn as sbn
# Построение количественной гистограммы значений столбца 'days_employed'
sbn.displot(x='days_employed', data=borrowers)
```

Out[8]:
```
<seaborn.axisgrid.FacetGrid at 0x7fa399392d90>
```



Полученная гистограмма свидетельствует о том, что укладывающиеся в рамки здравого смысла значения стажа являются отрицательными, а положительные сгруппированы в диапазоне свыше 300 000 дней (примерно 822 года). Наиболее вероятно, данная ошибка носит технический характер. Оценим долю таких аномальных значений:

In [9]:
```
# Оценка доли значений > 300 000 в столбце 'days_employed'
borrowers[borrowers['days_employed'] > 300000]['days_employed'].count() / len(borrowers)
```

Out[9]:
```
0.16004645760743322
```

Аномально большие значения составляют 16% от общего количества записей в таблице. Их надо исправить. Наиболее целесообразно для этого использовать данные о возрасте и образовании. Проверим значения столбцов `'dob_years'` и `'education'`.

In [10]:
```
# Список уникальных значений столбца 'dob_years'
```

```
borrowers['dob_years'].sort_values().unique()
```

Out[10]:
```
array([ 0, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
       69, 70, 71, 72, 73, 74, 75])
```

Наблюдается аномальное значение 0, которое никак не может соответствовать возрасту клиента в годах. Проверим, насколько часто встречается 0 в столбце `'dob_years'` с использованием метода value_counts().

In [11]:
```
# Подсчёт уникальных значений столбца 'dob_years'
borrowers['dob_years'].value_counts()
```

Out[11]:
```
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
```

Значение 0 встречается 101 раз. Не так уж и редко. Характер возникновения данной ошибки предположить сложно. Попробуем восстановить данные по информации из других столбцов. Например, увяжем возраст и значения столбца `'education'` в предположении, что по мере освоения различных ступеней образования человек взрослеет. Рассмотрим, какие уровни образования характерны для клиентов с возрастом 0:

In [12]:
```
# Определение уровня образования для клиентов с возрастом 0
borrowers[borrowers['dob_years'] == 0]['education'].unique()
```

Out[12]:
```
array(['Среднее', 'среднее', 'высшее', 'Высшее', 'неоконченное высшее',
       'СРЕДНЕЕ', 'ВЫСШЕЕ'], dtype=object)
```

В столбце `'education'` наблюдаются неявные дубликаты. Удалять дубликаты в категорийных столбцах предполагается ниже, в разделе "Удаление дубликатов". Тем не менее, для упрощения работы по исправлению аномального нулевого возраста клиентов проведём очистку столбца `'education'` от неявных дубликатов в текущем разделе.

Из обзора данных нам известно, что в таблице представлены 5 категорий уровня образования. Рассмотрим список уникальных значений столбца `'education'`:

In [13]:
```
# Список уникальных значений столбца 'education'
borrowers['education'].unique()
```

Out[13]:
```
array(['высшее', 'среднее', 'Среднее', 'СРЕДНЕЕ', 'ВЫСШЕЕ',
       'неоконченное высшее', 'начальное', 'Высшее',
       'НЕОКОНЧЕННОЕ ВЫСШЕЕ', 'Неоконченное высшее', 'НАЧАЛЬНОЕ',
       'Начальное', 'Ученая степень', 'УЧЕНАЯ СТЕПЕНЬ', 'ученая степень'],
      dtype=object)
```

Действительно, похоже, что одинаковые значения записаны по-разному и их общее количество явно больше 5. Приведём значения в `'education'` к нижнему регистру методом `.str.lower()` и выведем новый список полученных уникальных:

In [14]:
```
# Приведение к нижнему регистру значений столбца 'education'
borrowers['education'] = borrowers['education'].str.lower()
# Список уникальных значений столбца 'education'
borrowers['education'].unique()
```

Out[14]:
```
array(['высшее', 'среднее', 'неоконченное высшее', 'начальное',
       'ученая степень'], dtype=object)
```

Теперь в столбце `'education'` порядок. Рассмотрим ещё раз, какие уровни образования характерны для клиентов с возрастом 0:

In [15]:
```
# Определение уровня образования для клиентов с возрастом 0
borrowers[borrowers['dob_years'] == 0]['education'].unique()
```

Out[15]:
```
array(['среднее', 'высшее', 'неоконченное высшее'], dtype=object)
```

Аномальное значение возраста встречается в трёх категориях уровня образования. Как мы уже отмечали ранее, исправлять его предпочтительнее всего на медианное значение, но вычислять это медианное значение нужно в рамках каждой упомянутой категории:

In [16]:
```
# Вычисление медианного возраста в каждой категории образования
secondary_education_years_median = int(borrowers.loc[(borrowers['education'] == 'среднее') &
                                                     (borrowers['dob_years'] != 0), 'dob_years'].
                                       median())
incomplete_high_education_median = int(borrowers.loc[(borrowers['education'] == 'неоконченное высшее') &
                                                     (borrowers['dob_years'] != 0), 'dob_years'].
                                       median())
high_education_median = int(borrowers.loc[(borrowers['education'] == 'высшее') &
                                          (borrowers['dob_years'] != 0), 'dob_years'].
                            median())
print(f"Медианное значение возраста в категории 'среднее' равно {secondary_education_years_median}")
print(f"Медианное значение возраста в категории 'неоконченное высшее' равно {incomplete_high_education_median}")
print(f"Медианное значение возраста в категории 'высшее' равно {high_education_median}")
```

```
Медианное значение возраста в категории 'среднее' равно 44
Медианное значение возраста в категории 'неоконченное высшее' равно 31
Медианное значение возраста в категории 'высшее' равно 39
```

Полученные значения медиан не противоречат здравому смыслу - люди, не получившие высшее образование, тоже берут кредиты. Причём не всегда в молодом возрасте.

Исправим нулевые значения возраста в таблице `borrowers`:

In [17]:
```
# Исправление нулевых значений в столбце 'dob_years'
# Зададим словарь исправлений
correction_categories = {
    'среднее' : secondary_education_years_median,
    'неоконченное высшее' : incomplete_high_education_median,
    'высшее' : high_education_median
}
# В цикле применим исправления
for education_cat in correction_categories:
    borrowers.loc[(borrowers['education'] == education_cat) & (borrowers['dob_years'] == 0),
                  'dob_years'] = correction_categories[education_cat]
```

Проверим результаты корректировки значений возраста:

In [18]:
```
# Список уникальных значений столбца 'dob_years'
borrowers['dob_years'].sort_values().unique()
```

Out[18]:
```
array([19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
       36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52,
       53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69,
       70, 71, 72, 73, 74, 75])
```

Аномальных значений в столбце `'dob_years'` больше нет. Одновременно мы откорректировали названия категорий образования. Можно переходить к исправлению аномальных выбросов в столбце `'days_employed'`. Определим для этого функцию одной строки:

In [19]:
```
# Определение функции employment_correction для корректировки больших значений стажа
#=================================================================================
# На вход функции подаётся строка таблицы, содержащая:
#   - текущее значение стажа 'days_employed'
#   - возраст 'dob_years'
#   - уровень образования 'education'
# Функция возвращает откорректированное значение стажа
def employment_correction(row):
    # Сформируем возрастов получения образования
    education_age = {
        'начальное' : 16,    # примерный возраст законного начала трудовой деятельности
        'среднее' : 18,    # примерный возраст окончания средней школы
        'неоконченное высшее' : 21,    # примерный возраст окончания института минус 2
        'высшее' : 23,    # примерный возраст окончания института
        'ученая степень' : 26    # примерный возраст окончания аспирантуры
    }

    days_employed = row['days_employed']
    age = row['dob_years']
    education = row['education']

    # Если наблюдаем аномальное значение
    if days_employed > 300000:
        # Заменим его на разницу возрастов в днях
        days_employed = (age - education_age[education]) * 365
    # В противном случае вернём само значение
    return days_employed
```

Протестируем функцию:

In [20]:
```
# Зададим тестовый набор
row_values = [33, 'среднее', 400000] #значения возраста? уровня образования и стажа
row_columns = ['dob_years', 'education', 'days_employed'] #названия столбцов
# Сформируем строку
row = pd.Series(data=row_values, index=row_columns)
# Вызовем функцию
employment_correction(row)
```

Out[20]:
```
5475
```

Функция отработала корректно. Применим её для исправления столбца `'days_employed'`, после чего найдём его максимальное значение:

In [21]:
```
# Применение функции employment_correction к строкам таблицы borrowers
borrowers['days_employed'] = borrowers.apply(employment_correction, axis=1)
# Оценка максимального значения в столбце 'days_employed'
borrowers['days_employed'].max()
```

Out[21]:
```
20805.0
```

Значения свыше 300 000 исчезли. Максимальное значение 20 805 соответствует 57 годам стажа и, судя по всему, это результат работы функции. Проверим возраст человека с этим стажем:

In [22]:
```
borrowers[borrowers['days_employed'] == borrowers['days_employed'].max()]['dob_years']
```

Out[22]:
```
19385    73
Name: dob_years, dtype: int64
```

В 73 года человек может иметь стаж 57 лет. Продолжим устранение аномалий в столбце `'days_employed'`. Разберёмся с отрицательными значениями в нём при помощи метода `abs()` библиотеки `pandas` и повторно оценим разброс величин:

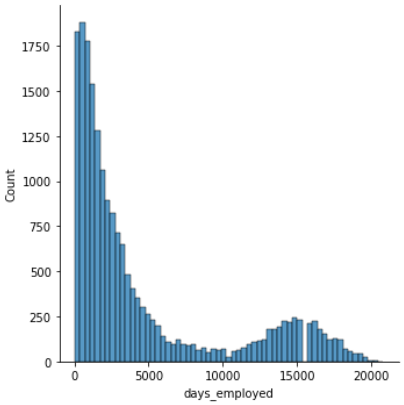
In [23]:
```
# Устранение отрицательных значений в столбце 'days_employed'
borrowers['days_employed'] = borrowers['days_employed'].abs()
# Оценка разброса значений в столбце 'days_employed'
print("MAX значение в 'days_employed': {0}\nMIN значение в 'days_employed': {1}".
      format(borrowers['days_employed'].max(), borrowers['days_employed'].min()))
```

```
MAX значение в 'days_employed': 20805.0
MIN значение в 'days_employed': 24.14163324048118
```

Отрицательных значений в столбце `'days_employed'` больше нет. Снова построим гистограмму и проверим распределение минимальных значений стажа:

In [24]:
```
# Построение количественной гистограммы значений столбца 'days_employed'
sbn.displot(x='days_employed', data=borrowers)
```

Out[24]:
```
<seaborn.axisgrid.FacetGrid at 0x7fa38f903970>
```



Гистограмма выглядит значительно лучше. Выдача банком кредитов людям, работающим меньше месяца, не типична, однако в целом левая часть гистограммы выглядит нормально, поэтому данный факт можно *отметить вопросом к клиентской службе и далее не корректировать*.

Количественный столбец `'children'` не содержит пропусков по результатам обзора данных, но вполне может содержать аномалии. Проверим это.

In [25]:
```
# Список уникальных значений столбца 'children'
borrowers['children'].sort_values().unique()
```

Out[25]:
```
array([-1,  0,  1,  2,  3,  4,  5, 20])
```

Неожиданно в ряду объяснимых значений 0, 1, 2, 3, 4, 5 встречаются аномальные -1 и 20. Проверим, насколько часто встречаются аномалии в таблице с использованием метода `value_counts()`.

In [26]:
```
# Подсчёт уникальных значений столбца 'children'
borrowers['children'].value_counts()
```

Out[26]:
```
 0     14149
 1      4818
 2      2055
 3       330
 20       76
-1        47
 4        41
 5         9
Name: children, dtype: int64
```

Аномальные значения встречаются не так уж и часто. При этом природа данных аномалий не ясна. *Целесообразно проконсультироваться со специалистами клиентского отдела и инженерами по данным.*

Однако, в настоящий момент это не возможно. Поэтому для исправления аномалий ***выдвинем предположение*** - наиболее вероятным источником данных ошибок представляются опечатки оператора:

- вместо 1 с цифровой клавиатуры было введено значение -1;
- вместо 2 и последующего нажатия Enter было введено 20.

Исправим данные аномалии присваиванием новых значений с использованием полной индексации:

In [27]:
```
# Исправление аномалий в столбце 'children'
borrowers.loc[borrowers['children'] == -1, 'children'] = 1
borrowers.loc[borrowers['children'] == 20, 'children'] = 2
# Проверим результат исправления
borrowers['children'].sort_values().unique()
```

Out[27]:
```
array([0, 1, 2, 3, 4, 5])
```

Аномалии в столбце `'children'` исправлены.

Перейдём к заполнению пропусков данных в таблице.

Заполнение пропусков

Устраним пропуски данных в столбцах таблицы `borrowers`. Для поиска используем комбинацию методов `isna()` и `sum()`:

In [28]:
```
# Подсчет количества пропусков в столбцах
borrowers.isna().sum()
```

Out[28]:
```
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
```

Пропуски, как и предполагалось ранее, наблюдаются в двух столбцах: `'days_employed'` и `'total_income'`. Интересно, какую долю составляют пропуски от общего количества записей?

In [29]:
```
# Определение доли пропусков в столбце 'days_employed' в процентах
print("Доля пропусков в столбце 'days_employed' составляет {:.0%}".
      format(borrowers['days_employed'].isna().sum() / len(borrowers['days_employed'])))
# Определение доли пропусков в столбце 'total_income' в процентах
print("Доля пропусков в столбце 'total_income' составляет {:.0%}".
      format(borrowers['total_income'].isna().sum() / len(borrowers['total_income'])))
```

```
Доля пропусков в столбце 'days_employed' составляет 10%
Доля пропусков в столбце 'total_income' составляет 10%
```

Итак, пропуски составляют 10% от общего количества записей, поэтому их необходимо восполнять. Проверим, связаны ли пропуски в столбце `'total_income'` с пропусками в `'days_employed'`.

In [30]:
```
nan_rows = borrowers[borrowers['total_income'].isna()]
print("Общее количество пропусков в 'total_income':", len(nan_rows))
print("Среди них пропусков в 'days_employed':", nan_rows['days_employed'].isna().sum())
```

```
Общее количество пропусков в 'total_income': 2174
Среди них пропусков в 'days_employed': 2174
```

Очевидно, пропуски в столбцах `'days_employed'` и `'total_income'` сделаны в одних и тех же строках. Это свидетельствует о человеческом факторе, как о возможной причине возникновения пропусков. Скорее всего, данные о стаже и доходе не были указаны клиентами, или по какой-то причине не введены оператором в систему в 2174 случаях.

Заполним пропуски в столбце `'total_income'` с использованием функции для строки.

In [31]:
```
# Создадим вектор медианных значений дохода по типам занятости
total_income_medians = borrowers.groupby('income_type')['total_income'].median()
display(total_income_medians) # выведем полученный вектор

# Определение функции total_income_fill_nans для замены пропусков в столбце 'total_income'
#=============================================================================================
# На вход функции подаётся строка таблицы, содержащая:
#   - текущее значение дохода 'total_income'
#   - тип занятости 'income_type'
# Функция возвращает откорректированное значение дохода
def total_income_fill_nans(row):
    total_income = row['total_income']
    income_type = row['income_type']

    if math.isnan(total_income):
        total_income = total_income_medians[income_type]
    return total_income

# Протестируем функцию total_income_fill_nans
# 1. Зададим тестовый набор
row_values = [math.nan, 'госслужащий'] # значения дохода и типа занятости
row_columns = ['total_income', 'income_type'] #названия столбцов
# 2. Сформируем строку
r = pd.Series(data=row_values, index=row_columns)
# Вызовем функцию
total_income_fill_nans(r)
```

```
income_type
безработный        131339.751676
в декрете           53829.130729
госслужащий        150447.935283
компаньон          172357.950966
пенсионер          118514.486412
предприниматель    499163.144947
сотрудник          142594.396847
студент             98201.625314
Name: total_income, dtype: float64
```

Out[31]:
```
150447.9352830068
```

Функция работает корректно. Применим её для заполнения пропусков в столбце `'total_income'`:

In [32]:
```
# Заполнение пропусков в 'total_income'
borrowers['total_income'] = borrowers.apply(total_income_fill_nans, axis=1)
# Проверка результатов заполнения пропусков в 'total_income'
print("Количество пропусков в 'total_income' равно:", borrowers['total_income'].isna().sum())
```

```
Количество пропусков в 'total_income' равно: 0
```

Пропуски в `'total_income'` заполнены. Для заполнения пропусков в столбце `'days_employed'` можно было бы по аналогии со столбцом `'total_income'` использовать медианные значения. Однако выше мы уже корректировали слишком большие значения в `'days_employed'` прогнозными на основе возраста и уровня образования. В контексте данного исследования такая корректировка представляется не хуже заполнения пропусков медианой. Поэтому заполним их заведомо большими значениями (> 300 000) и вызовем функцию корректировки `employment_correction`:

In [33]:
```
# Заполнение пропусков в 'days_employed' заведомо большими значениями
borrowers['days_employed'] = borrowers['days_employed'].fillna(400000)
# Проверка результатов заполнения пропусков в 'days_employed'
print("Количество пропусков в 'days_employed' равно:", borrowers['days_employed'].isna().sum())
```

```
# Применение функции employment_correction к строкам таблицы borrowers
borrowers['days_employed'] = borrowers.apply(employment_correction, axis=1)
# Оценка максимального значения в столбце 'days_employed'
print("Максимальное значение в 'days_employed' равно:", borrowers['days_employed'].max())
```

```
Количество пропусков в 'days_employed' равно: 0
Максимальное значение в 'days_employed' равно: 20805.0
```

Все пропуски в количественных данных устранены. Однако мы до сих пор не понимаем, почему значения стажа выражены вещественным числом, а также в каких единицах представлен доход.

Изменение типов данных

Если природа вещественных значений в столбце дохода может быть объяснена усреднением дохода по месяцам (среднее имеет вещественный тип, поскольку при его вычислении используется обычная операция деления целых чисел), то использование значений с плавающей точкой для стажа не характерно. Обычно учитывают стаж на последнем месте работы.

Вероятно, в таблицу `borrowers` вещественные значения в столбце `'days_employed'` попали в результате технической ошибки, связанной с извлечением данных, или их машинным представлением.

В любом случае, для дальнейшего исследования целесообразнее привести столбцы `'total_income'` и `'days_employed'` к целому типу значений. Выполним это применением метода `.astype('int')` к соответствующим столбцам датафрейма:

In [34]:
```
# Приведение столбца 'total_income' к целочисленным значениям
borrowers['total_income'] = borrowers['total_income'].astype('int')
# Приведение столбца 'days_employed' к целочисленным значениям
borrowers['days_employed'] = borrowers['days_employed'].astype('int')
# Проверка результата
borrowers.head()
```

Out[34]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income	purpose
0	1	8437	42	высшее	0	женат / замужем	0	F	сотрудник	0	253875	покупка жилья
1	1	4024	36	среднее	1	женат / замужем	0	F	сотрудник	0	112080	приобретение автомобиля
2	0	5623	33	среднее	1	женат / замужем	0	M	сотрудник	0	145885	покупка жилья
3	3	4124	32	среднее	1	женат / замужем	0	M	сотрудник	0	267628	дополнительное образование
4	0	12775	53	среднее	1	гражданский брак	1	F	пенсионер	0	158616	сыграть свадьбу

Теперь в столбцах `'total_income'` и `'days_employed'` целые числа.

Отметим, что на самом деле для нашего исследования не важна природа значений `'days_employed'`, поскольку данная информация не будет использоваться при проверке сформулированных гипотез.

Также не важны единицы измерения значений (рубли, доллары, копейки) столбца `'total_income'`. В дальнейшем для удобства анализа мы разобъём заёмщиков на несколько категорий по уровню дохода. Попутно узнаем количество миллионеров среди клиентов банка.

Удаление дубликатов

Для начала удалим явные дубликаты, если такие есть в таблице, поскольку маловероятно, что два и более различных заёмщика имели 12 полностью одинаковых параметров в таблице `borrowers`:

In [35]:
```
# Удаление явных дубликатов (с удалением индексов и формированием новых)
borrowers = borrowers.drop_duplicates()
# Проверка на отсутствие явных дубликатов
borrowers.duplicated().sum()
```

Out[35]:
```
0
```

Явных дубликатов нет. Рассмотрим теперь неявные.

Часть категориальных столбцов, а именно - числовые и булевский идентификаторы, мы рассмотрели на этапе обзора данных. Теперь мы знаем, что в таблице представлены 5 категорий уровня образования, 5 категорий семейного положения. Учитывая эти знания, перейдем к рассмотрению категории `'family_status'`, поскольку от дубликатов в столбце `'education'` мы избавились ранее:

In [36]:
```
# Список уникальных значений столбца 'education'
borrowers['education'].unique()
```

Out[36]:
```
array(['высшее', 'среднее', 'неоконченное высшее', 'начальное',
       'ученая степень'], dtype=object)
```

Проделаем то же самое с `'family_status'`:

In [37]:
```
# Список уникальных значений столбца 'family_status'
borrowers['family_status'].unique()
```

Out[37]:
```
array(['женат / замужем', 'гражданский брак', 'вдовец / вдова',
       'в разводе', 'Не женат / не замужем'], dtype=object)
```

В столбце `'family_status'` в точности 5 значений, но присутствуют символы в верхнем регистре. Приведём значения в `'family_status'` к нижнему регистру и выведем новый список полученных уникальных:

In [38]:
```
# Приведение к нижнему регистру значений столбца 'family_status'
borrowers['family_status'] = borrowers['family_status'].str.lower()
# Список уникальных значений столбца 'education'
borrowers['family_status'].unique()
```

Out[38]:
```
array(['женат / замужем', 'гражданский брак', 'вдовец / вдова',
       'в разводе', 'не женат / не замужем'], dtype=object)
```

Теперь значения столбца `'family_status'` соответствуют "хорошему стилю" анализируемых данных.

У нас остались нерассмотрены три категории: `'gender'`, `'income_type'` и `'purpose'`. Количество значений в них нам пока не известно. Рассмотрим список уникальных значений цели кредита. Для удобства восприятия отсортируем значения в лексикографическом порядке:

In [39]:
```
# Список уникальных значений столбца 'purpose'
borrowers['purpose'].sort_values().unique()
```

Out[39]:
```
array(['автомобили', 'автомобиль', 'высшее образование',
       'дополнительное образование', 'жилье',
       'заняться высшим образованием', 'заняться образованием',
       'на покупку автомобиля', 'на покупку подержанного автомобиля',
       'на покупку своего автомобиля', 'на проведение свадьбы',
       'недвижимость', 'образование', 'операции с жильем',
       'операции с коммерческой недвижимостью',
       'операции с недвижимостью', 'операции со своей недвижимостью',
       'покупка жилой недвижимости', 'покупка жилья',
       'покупка жилья для сдачи', 'покупка жилья для семьи',
       'покупка коммерческой недвижимости', 'покупка недвижимости',
       'покупка своего жилья', 'получение высшего образования',
       'получение дополнительного образования', 'получение образования',
       'приобретение автомобиля', 'профильное образование',
       'ремонт жилью', 'свадьба', 'свой автомобиль',
       'сделка с автомобилем', 'сделка с подержанным автомобилем',
       'строительство жилой недвижимости', 'строительство недвижимости',
       'строительство собственной недвижимости', 'сыграть свадьбу'],
      dtype=object)
```

Анализ полученного множества позволяет выявить следующие неявные дубликаты:

- 'автомобили', 'автомобиль' - некие общие категории, о сути которых ничего не известно, можно объединить;
- 'высшее образование', 'заняться высшим образованием', 'получение высшего образования' - по сути одна категория;
- 'дополнительное образование', 'получение дополнительного образования' - по сути одна категория;
- 'заняться образованием', 'образование', 'получение образования' - по сути одна категория без уточнения специфики;

Также в списке значений присутствуют различные виды операций с неовижимостью, включая жилую, причём последняя представлена в виде просторечного выражения "жилье". Предлагается заменить "жилье" на "жилая недвижимость" для упрощения дальнейшей классификации: 'жилье', 'операции с жильем', 'покупка жилья', 'покупка жилья для сдачи', 'покупка жилья для семьи', 'покупка своего жилья', 'ремонт жилью'.

Данные неявные дубликаты в `'purpose'` появились по причине отсутствия формализованных правил описания категорий в банковском программном обеспечении. Клиенты в произвольной форме заполняют цель кредита в заявке, а операторы дословно переносят данные в базу. Такой путь может привести к бесконтрольному разрастанию количества используемых категорий и трудностям в анализе и классификации клиентов по цели кредита.

Напишем функцию, которая будет заменять значения в столбцах:

In [40]:
```
# Функция замены неявных дубликатов
#==================================
# На вход функции подаются:
#   - название столбца (column)
#   - список значений, которые надо заменить (wrong_values)
#   - значение, на которое надо заменить (correct_value)
# Замена происходит непосредственно в таблице borrowers
# Функция не возвращает значений
def replace_wrong_values(column, wrong_values, correct_value):
    for wrong_value in wrong_values:
        borrowers[column] = borrowers[column].replace(wrong_value, correct_value)
```

Устраним неявные дубликаты в столбце `'purpose'`:

In [41]:
```
# 1. Сформируем словарь неявных дубликатов в формате
# 'correct_value' : [wrong_value_list]
values_to_replace = {
    'автомобиль' : ['автомобили'],
    'высшее образование' : ['заняться высшим образованием', 'получение высшего образования'],
    'дополнительное образование' : ['получение дополнительного образования'],
    'образование' : ['заняться образованием', 'получение образования'],
    'жилая недвижимость' : ['жилье'],
    'операции с жилой недвижимостью' : ['операции с жильем'],
    'покупка жилой недвижимости' : ['покупка жилья'],
    'покупка жилой недвижимости для сдачи' : ['покупка жилья для сдачи'],
    'покупка жилой недвижимости для семьи' : ['покупка жилья для семьи'],
    'покупка своей жилой недвижимости' : ['покупка своего жилья'],
    'ремонт жилой недвижимости' : ['ремонт жилью']
}
# 2. В цикле пройдём словарь и осуществим вызов функции replace_wrong_values()
for corr_value in values_to_replace:
    replace_wrong_values('purpose', values_to_replace[corr_value], corr_value)
# 3. Проверим результат устранения дубликатов
borrowers['purpose'].sort_values().unique()
```

Out[41]:
```
array(['автомобиль', 'высшее образование', 'дополнительное образование',
       'жилая недвижимость', 'на покупку автомобиля',
       'на покупку подержанного автомобиля',
       'на покупку своего автомобиля', 'на проведение свадьбы',
       'недвижимость', 'образование', 'операции с жилой недвижимостью',
       'операции с коммерческой недвижимостью',
       'операции с недвижимостью', 'операции со своей недвижимостью',
       'покупка жилой недвижимости',
       'покупка жилой недвижимости для сдачи',
       'покупка жилой недвижимости для семьи',
       'покупка коммерческой недвижимости', 'покупка недвижимости',
       'покупка своей жилой недвижимости', 'приобретение автомобиля',
       'профильное образование', 'ремонт жилой недвижимости', 'свадьба',
       'свой автомобиль', 'сделка с автомобилем',
       'сделка с подержанным автомобилем',
       'строительство жилой недвижимости', 'строительство недвижимости',
       'строительство собственной недвижимости', 'сыграть свадьбу'],
      dtype=object)
```

Список целей кредита выглядит аккуратнее и единообразнее. Количество категорий сократилось. Однако по-прежнему сохраняется определенное количество схожих по смыслу значений.

Для целей данного исследования в дальнейшем мы проведём укрупнение категорий целей кредита путём введения групп категорий. *Однако, следует обсудить с клиентским отделом вопрос сокращения и приведения к единому стандарту списка целей*. Это может потребоваться в случае проведения более тонкого анализа по целям.

Отметим, что функция `replace_wrong_values()`, реализованная в данном методе устранения неявных дубликатов, универсальна и может быть при необходимости применена к любому столбцу таблицы `'borrowers'`.

Рассмотрим список уникальных значений типов занятости. Для удобства восприятия отсортируем значения в лексикографическом порядке:

In [42]:
```
# Список уникальных значений столбца 'income_type'
borrowers['income_type'].sort_values().unique()
```

Out[42]:
```
array(['безработный', 'в декрете', 'госслужащий', 'компаньон',
       'пенсионер', 'предприниматель', 'сотрудник', 'студент'],
      dtype=object)
```

Дубликатов в столбце `'income_type'` не выявлено.

Рассмотрим список уникальных значений пола клиентов. Для удобства восприятия отсортируем значения в лексикографическом порядке:

In [43]:
```
# Список уникальных значений столбца 'gender'
borrowers['gender'].sort_values().unique()
```

Out[43]:
```
array(['F', 'M', 'XNA'], dtype=object)
```

В столбце `'gender'` дубликатов также не обнаружено. Значение `'XNA'` соответствует случаям, когда клиент отказался предоставлять информацию о поле. Такая ситуация является нормальной.

Итак, дубликаты в таблице устранены. Перейдём к устранению избыточности хранящейся в ней информации путем её декомпозиции и выделения словарей категорий.

Формирование дополнительных датафреймов словарей, декомпозиция исходного датафрейма

Выделим словари семейного положения `family_statuses` и уровня образования `educations` из исходного датафрейма путём выделения нужных столбцов в новую таблицу, удаления дубликатов значений и сброса индексации.

In [44]:
```
# Выделим столбцы 'family_status_id' и 'family_status', удалим дубликаты и сбросим индексацию
family_statuses = borrowers[['family_status_id', 'family_status']].drop_duplicates().reset_index(drop=True)
family_statuses
```

Out[44]:

	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

In [45]:
```
# Выделим столбцы 'education_id' и 'education', удалим дубликаты и сбросим индексацию
educations = borrowers[['education_id', 'education']].drop_duplicates().reset_index(drop=True)
educations
```

Out[45]:

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

Итак, мы создали словари семейного положения и уровня образования. Теперь можно устранить избыточность в таблице `borrowers`, удалив столбцы `'family_status'` и `'education'`. Результат проверим, выведя первые 5 строк таблицы.

In [46]:
```
# удаление столбцов 'family_status' и 'education'
borrowers = borrowers.drop(['family_status', 'education'], axis='columns')
# проверка результата
borrowers.head()
```

Out[46]:

	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 жилой недвижимости
1	1	4024	36	1	0	F	сотрудник	0	112080	приобретение автомобиля
2	0	5623	33	1	0	M	сотрудник	0	145885	покупка жилой недвижимости
3	3	4124	32	1	0	M	сотрудник	0	267628	дополнительное образование
4	0	12775	53	1	1	F	пенсионер	0	158616	сыграть свадьбу

Избыточность таблицы устранена.

Категоризация дохода

Оперировать количественными значениями при ответе на вопросы исследования может быть не удобно, поскольку разброс значений, как мы помним, велик. Поэтому представляется целесообразным сгруппировать клиентов по уровню дохода:

- `A` - 1000001 и выше;
- `B` - 200001–1000000;
- `C` - 50001–200000;
- `D` - 30001–50000;
- `E` - 0–30000.

Для этого напишем функцию, которая на основе принятого целочисленного значения возвращает одну из перечисленных групп доходов:

In [47]:
```
# Определение функции категоризации доходов income_category(income)
#================================================================
# На вход функции подаётся целочисленное значение дохода
# Функция возвращает:
#   - в случае неотрицательного значения на входе - одну из 5 категорий дохлда
#   - в случае отрицательного значения на входе - сообщение об ошибке
# Поскольку на вход может поступать не числовое значение, обеспечим безопасность функции
```

```
# с использованием конструкции try - except
def income_category(income):
    try:
        if income > 1000000:
            return 'A'
        elif income > 200000:
            return 'B'
        elif income > 50000:
            return 'C'
        elif income > 30000:
            return 'D'
        elif income > 0:
            return 'E'
        else:
            ret = 'Ошибка! Отрицательный доход!'
    except:
        ret = 'Ошибка! Не число на входе!'
    return ret
```

Проверим функцию `income_category()` на тестовом наборе данных:

In [48]:
```
# Тестирование функции income_category(income)
# 1. Зададим тестовые данные
test_data = [2500000, 35000, 60000, 10001, 700200, -24, '500']
# 2. В цикле проверим работу функции
for val in test_data:
    print(f'Значению {val} соответствует выход: {income_category(val)}')
```

```
Значению 2500000 соответствует выход: A
Значению 35000 соответствует выход: D
Значению 60000 соответствует выход: C
Значению 10001 соответствует выход: E
Значению 700200 соответствует выход: B
Значению -24 соответствует выход: Ошибка! Отрицательный доход!
Значению 500 соответствует выход: Ошибка! Не число на входе!
```

Функция работает. Применим её для добавления к таблице `borrowers` столбца `'total_income_category'` с категорией доходов и выведем 5 строк результата:

In [49]:
```
# Добавление столбца 'total_income_category'
borrowers['total_income_category'] = borrowers['total_income'].apply(income_category)
# Проверим результат
borrowers.head()
```

Out[49]:

	children	days_employed	dob_years	education_id	family_status_id	gender	income_type	debt	total_income	purpose	total_income_category
0	1	8437	42	0	0	F	сотрудник	0	253875	покупка жилой недвижимости	B
1	1	4024	36	1	0	F	сотрудник	0	112080	приобретение автомобиля	C
2	0	5623	33	1	0	M	сотрудник	0	145885	покупка жилой недвижимости	C
3	3	4124	32	1	0	M	сотрудник	0	267628	дополнительное образование	B
4	0	12775	53	1	1	F	пенсионер	0	158616	сыграть свадьбу	C

Мы присвоили всем заёмщикам категории дохода. Теперь данные таблицы `borrowers` можно группировать по данной категории.

Категоризация целей кредита

Проведём укрупнение категорий целей кредита путем задания групп категорий `'purpose_category'` в следующем составе: `'операции с автомобилем'`, `'операции с недвижимостью'`, `'проведение свадьбы'`, `'получение образования'`.

Создадим функцию, которая на основании данных из столбца `'purpose'` сформирует новый столбец `'purpose_category'`:

In [50]:
```
# Определение функции категоризации целей кредита purpose_category(purpose)
#=============================================================================
# На вход функции подаётся строковое значение цели кредита
# Функция возвращает:
#   - в случае наличия в строке одного из ключевых слов - одну из 4 категорий цели кредита
#   - в случае отсутствия ключевых слов - значение 'undefined'
# Поскольку на вход может поступать не строковое значение, обеспечим безопасность функции
# с использованием конструкции try - except
def purpose_category(purpose):
    try:
        if purpose.find('авто') != -1:
            return 'операции с автомобилем'
        elif purpose.find('недвиж') != -1:
            return 'операции с недвижимостью'
        elif purpose.find('свадьб') != -1:
            return 'проведение свадьбы'
        elif purpose.find('образован') != -1:
            return 'получение образования'
        else:
            ret = 'undefined'
    except:
        ret = 'Ошибка! Не строка на входе!'
    return ret
```

Протестируем функцию `purpose_category()` на тестовом наборе данных:

In [51]:
```
# Тестирование функции purpose_category()
# 1. Зададим тестовые данные
test_data = ['автомобиль', 'высшее образование', 'дополнительное образование',
             'жилая недвижимость', 'на покупку автомобиля',
             'на покупку подержанного автомобиля', 'пусто',
             'на покупку своего автомобиля', 'на проведение свадьбы',
             'недвижимость', 'образование', 'операции с жилой недвижимостью']
# 2. В цикле проверим работу функции
for val in test_data:
    print(f'Значению {val} соответствует выход: {purpose_category(val)}')
```

```
Значению автомобиль соответствует выход: операции с автомобилем
Значению высшее образование соответствует выход: получение образования
Значению дополнительное образование соответствует выход: получение образования
Значению жилая недвижимость соответствует выход: операции с недвижимостью
Значению на покупку автомобиля соответствует выход: операции с автомобилем
Значению на покупку подержанного автомобиля соответствует выход: операции с автомобилем
Значению пусто соответствует выход: undefined
Значению на покупку своего автомобиля соответствует выход: операции с автомобилем
Значению на проведение свадьбы соответствует выход: проведение свадьбы
Значению недвижимость соответствует выход: операции с недвижимостью
Значению образование соответствует выход: получение образования
Значению операции с жилой недвижимостью соответствует выход: операции с недвижимостью
```

Функция работает. Добавим в таблицу `borrowers` столбец `'purpose_category'`:

In [52]:
```
# Добавление столбца 'purpose_category'
borrowers['purpose_category'] = borrowers['purpose'].apply(purpose_category)
# Проверим результат
borrowers['purpose_category'].unique()
```

Out[52]:
```
array(['операции с недвижимостью', 'операции с автомобилем',
       'получение образования', 'проведение свадьбы'], dtype=object)
```

Мы укрупнили категории цели кредита, причём учли все возможные значения в столбце `'purpose'`, не получив в `'purpose_category'` ни одного значения `undefined`. Теперь данные таблицы borrowers можно группировать по `'purpose_category'`.

Перейдём к ответам на основные вопросы исследования.

Ответы на вопросы исследования (проверка гипотез)

В описании контекста исследования мы сформулировали 4 гипотезы, которые надо проверить:

1. Количество детей влияет на возврат кредита в срок.
2. Между семейным положением и возвратом кредита в срок существует зависимость.
3. Уровень дохода влияет на возврат кредита в срок.
4. Существует зависимость между целью кредита и его возвратом в срок.

Напомним, что мы действуем в предположении, что ***значением `0` кодируется отсутствие, а значением `1` - наличие задолженности***.

Оценка влияния количества детей на возврат кредита в срок

Необходимо проверить гипотезу: *Количество детей влияет на возврат кредита в срок*.

Для ответа на первый вопрос создадим сводную таблицу `borrowers_children_pivot` с индексами из столбца `'children'` таблицы `borrowers` и столбцами-категориями из значений индикаторов наличия задолженности `'debt'`. В качестве функции-аггрегатора используем подсчёт количества значений (count). В этом случае столбцом значений может быть любой отличный от `'children'` и `'debt'`, например - `'total_income'`:

In [53]:
```
# Построение сводной таблицы borrowers_children_pivot
borrowers_children_pivot = borrowers.pivot_table(index='children', columns='debt',
                                                 values='total_income', aggfunc='count')
borrowers_children_pivot
```

Out[53]:

debt	0	1
children		
0	13028.0	1063.0
1	4410.0	445.0
2	1926.0	202.0
3	303.0	27.0
4	37.0	4.0
5	9.0	NaN

В таблице `borrowers_children_pivot` присутствует пропуск на пересечении строки 5 и колонки 1. Вероятно, он появился из-за отсутствия задолженностей у заёмщиков с 5 детьми. Проверим эту гопотезу, сгруппировав таблицу `borrowers` по столбцу `'children'` и просуммировав сгруппированные индикаторы наличия задолженности `'debt'`:

In [54]:
```
# Определение количества просрочек в зависимости от уровня дохода
borrowers.groupby('children')['debt'].sum()
```

Out[54]:
```
children
0    1063
1     445
2     202
3      27
4       4
5       0
Name: debt, dtype: int64
```

Действительно, количество просрочек у клиентов с 5 детьми равно 0. Устраним пропуск в сводной таблице.

Кроме того, добавим к ней столбец `'total'`, содержащий общее количество клиентов, имеющих заданное количество детей. Результат отсортируем по убыванию значений в столбце `1` (наличие задолженности):

In [55]:
```
# Заполнение пропуска в таблице borrowers_children_pivot
borrowers_children_pivot = borrowers_children_pivot.fillna(0)
# Добавление столбца 'total'
borrowers_children_pivot['total'] = borrowers_children_pivot[1] + borrowers_children_pivot[0]
# Сортировка таблицы
borrowers_children_pivot.sort_values(1, ascending=False)
```

Out[55]:

children	debt	0	1	total
0		13028.0	1063.0	14091.0
1		4410.0	445.0	4855.0
2		1926.0	202.0	2128.0
3		303.0	27.0	330.0
4		37.0	4.0	41.0
5		9.0	0.0	9.0

Данные сводной таблицы, с одной стороны, показывают, что чем больше у заёмщиков детей, тем меньше у них просрочек (выше платёжная дисциплина). В то же время, с ростом количества детей уменьшается и общее количество таких клиентов. Поэтому по абсолютному количеству задолженностей нельзя судить о платёжной дисциплине клиентов, имеющих детей.

Добавим в таблицу столбец `'percent'`, отражающий процентное отношение количества клиентов, имеющих задолженность, к общему количеству клиентов, имеющих заданное количество детей. Результат отсортируем по убыванию значений в столбце `'percent'`:

In [56]:
```
# Добавление столбца 'percent'
borrowers_children_pivot['percent'] = round(borrowers_children_pivot[1] * 100 / \
                                      borrowers_children_pivot['total'], 2)
# Сортировка таблицы
borrowers_children_pivot.sort_values('percent', ascending=False)
```

Out[56]:

children	debt	0	1	total	percent
4		37.0	4.0	41.0	9.76
2		1926.0	202.0	2128.0	9.49
1		4410.0	445.0	4855.0	9.17
3		303.0	27.0	330.0	8.18
0		13028.0	1063.0	14091.0	7.54
5		9.0	0.0	9.0	0.00

Из полученной таблицы очевидно, что наибольшей платёжной дисциплиной обладают многодетные клиенты с 5-ю детьми, за ними следуют клиенты без детей. Средний результат - у многодетных с 3-мя детьми. Клиенты, имеющие 1, 2 или 4 детей имеют примерно одинаковую платёжную дисциплину.

Следует отметить, что результат для клиентов с 5 детьми можно считать спорным ввиду их малочисленной выборки, представленной в данных. С учётом этого замечания можно сделать следующий вывод.

Вывод

Гипотеза подтвердилась частично. Если не учитывать непредставительную выборку клиентов с 5 детьми, то бездетные клиенты допускают меньше задолженностей, чем имеющие детей.

Однозначный характер зависимости количества просрочек у клиентов с 1, 2, 3 и 4 детьми установить по имеющимся данным затруднительно. Наиболее дисциплинированными являются клиенты банка, у которых по 3 ребёнка. В остальном, с ростом количества детей незначительно увеличивается процент задолженностей.

Оценка взаимосвязи между семейным положением и возвратом кредита в срок

Необходимо проверить гипотезу: *Между семейным положением и возвратом кредита в срок существует зависимость*.

Для ответа на данный вопрос объединим таблицу `borrowers` со словарём `family_statuses` исоздадим сводную таблицу `borrowers_family_statuses_pivot` с индексами из столбца `'family_statuses'` и столбцами-категориями из значений индикаторов наличия задолженности `'debt'`. В качестве функции-аггрегатора используем подсчёт количества значений (count). В этом случае столбцом значений может быть любой отличный от `'family_statuses'` и `'debt'`, например - `'total_income'`.

Кроме того, добавим к ней столбец `'total'`, содержащий общее количество клиентов, имеющих заданное семейное положение. Результат отсортируем по убыванию значений в столбце `1` (наличие задолженности):

In [57]:
```
# Объединение таблиц borrowers и family_statuses
borrowers_family_statuses = borrowers.merge(family_statuses, on='family_status_id', how='right')
# Построение сводной таблицы borrowers_family_statuses_pivot
borrowers_family_statuses_pivot = \
    borrowers_family_statuses.pivot_table(index='family_status', columns='debt',
                                          values='total_income', aggfunc='count')
# Добавление столбца 'total'
borrowers_family_statuses_pivot['total'] = borrowers_family_statuses_pivot[1] + \
                                           borrowers_family_statuses_pivot[0]
# Сортировка таблицы
borrowers_family_statuses_pivot.sort_values(1, ascending=False)
```

Out[57]:

family_status	debt	0	1	total
женат / замужем		11408	931	12339
гражданский брак		3763	388	4151
не женат / не замужем		2536	274	2810
в разводе		1110	85	1195
вдовец / вдова		896	63	959

Как и в предыдущем случае, наблюдается одновременное убывание как количества задолженностей, так и общего количества клиентов по каждой категории семейного положения. Поэтому добавим в таблицу столбец `'percent'`, отражающий процентное отношение количества клиентов, имеющих задолженность, к общему количеству клиентов, имеющих заданную категорию семейного положения. Результат отсортируем по убыванию значений в столбце `'percent'`:

In [58]:
```
# Добавление столбца 'percent'
borrowers_family_statuses_pivot['percent'] = round(borrowers_family_statuses_pivot[1] * 100 / \
                                             borrowers_family_statuses_pivot['total'], 2)
# Сортировка таблицы
borrowers_family_statuses_pivot.sort_values('percent', ascending=False)
```

Out[58]:

debt	0	1	total	percent
family_status				
не женат / не замужем	2536	274	2810	9.75
гражданский брак	3763	388	4151	9.35
женат / замужем	11408	931	12339	7.55
в разводе	1110	85	1195	7.11
вдовец / вдова	896	63	959	6.57

Из данной таблицы следует, что меньше всего просрочек допускают овдовевшие клиенты, больше всего - холостые или живущие гражданским браком. Разведенные и семейные клиенты имеют среднюю платёжную дисциплину.

Вывод

Гипотеза о том, что между семейным положением и возвратом кредита в срок существует зависимость, полностью подтвердилась.

Оценка влияния уровня дохода на возврат кредита в срок

Необходимо проверить гипотезу: *Уровень дохода влияет на возврат кредита в срок*.

Ранее мы ввели категоризацию клиентов по уровню дохода:

- `A` - 1000001 и выше;
- `B` - 200001–1000000;
- `C` - 50001–200000;
- `D` - 30001–50000;
- `E` - 0–30000.

Для ответа на данный вопрос создадим сводную таблицу `borrowers_total_income_pivot` с индексами из столбца `'total_income_category'` таблицы `borrowers` (анализировать по значениям дохода невозможно в силу их большого количества, сравнимого с общим числом строк в таблице `borrowers`) и столбцами-категориями из значений индикаторов наличия задолженности `'debt'`. В качестве функции-аггрегатора используем подсчёт количества значений (count). В этом случае столбцом значений может быть любой отличный от `'total_income_category'` и `'debt'`, например - `'total_income'`.

Кроме того, добавим к ней столбец `'total'`, содержащий общее количество клиентов, имеющих заданную категорию дохода. Результат отсортируем по убыванию значений в столбце `1` (наличие задолженности):

In [59]:
```
# Определение количества просрочек в зависимости от уровня дохода
borrowers_total_income_pivot = \
    borrowers.pivot_table(index='total_income_category', columns='debt',
                                    values='total_income', aggfunc='count')
# Добавление столбца 'total'
borrowers_total_income_pivot['total'] = borrowers_total_income_pivot[1] + \
                                        borrowers_total_income_pivot[0]
# Сортировка таблицы
borrowers_total_income_pivot.sort_values(1, ascending=False)
```

Out[59]:

debt	0	1	total
total_income_category			
C	14655	1360	16015
B	4686	356	5042
D	329	21	350
A	23	2	25
E	20	2	22

Как и раньше, наблюдается одновременное убывание как количества задолженностей, так и общего количества клиентов по каждой категории дохода. Поэтому добавим в таблицу столбец `'percent'`, отражающий процентное отношение количества клиентов, имеющих задолженность, к общему количеству клиентов, имеющих заданную категорию дохода. Результат отсортируем по убыванию значений в столбце `'percent'`:

In [60]:
```
# Добавление столбца 'percent'
borrowers_total_income_pivot['percent'] = round(borrowers_total_income_pivot[1] * 100 / \
                                          borrowers_total_income_pivot['total'], 2)
# Сортировка таблицы
borrowers_total_income_pivot.sort_values('percent', ascending=False)
```

Out[60]:

debt	0	1	total	percent
total_income_category				
E	20	2	22	9.09
C	14655	1360	16015	8.49
A	23	2	25	8.00
B	4686	356	5042	7.06
D	329	21	350	6.00

Из полученной сводной таблицы следует, что меньше всего просрочек допускают клиенты с доходом от 30 до 50 тысяч, больше всего - клиенты с минимальными доходами (до 30 тысяч). Наиболее богатые клиенты (доход свыше миллиона) а также так называемый "средний класс" с доходом от 50 до 200 тысяч находятся в нижней части рейтинга платёжной дисциплины и уступают по этому параметру клиентам с доходом "выше среднего" (от 200 тысяч до миллиона).

Вывод

Гипотеза о том, что уровень дохода влияет на возврат кредита в срок, полностью подтвердилась. При этом худший результат оказался ожидаемым - наиболее бедные клиенты чаще допускают просрочки. А вот категория наиболее богатых заёмщиков показала себя не самыми дисциплинированными плательщиками.

Оценка взаимосвязи между целью кредита и его возвратом в срок

Необходимо проверить гипотезу: *Существует зависимость между целью кредита и его возвратом в срок*.

Для ответа на данный вопрос создадим сводную таблицу `borrowers_purpose_pivot` с индексами из столбцов `'purpose_category'` и `'purpose'` таблицы `borrowers` и столбцами-категориями из значений индикаторов наличия задолженности `'debt'`. В качестве функции-аггрегатора используем подсчёт количества значений (count). В этом случае столбцом значений может быть любой отличный от `'purpose_category'`, `'purpose'` и `'debt'`, например - `'total_income'`.

Кроме того, добавим к ней столбец `'total'`, содержащий общее количество клиентов, взявших кредит для конкретной цели. Результат отсортируем по убыванию значений в столбце `1` (наличие задолженности):

In [61]:
```
# Определение количества просрочек в зависимости от уровня дохода
borrowers_purpose_pivot = \
    borrowers.pivot_table(index=['purpose_category', 'purpose'], columns='debt',
                          values='total_income', aggfunc='count')
# Добавление столбца 'total'
borrowers_purpose_pivot['total'] = borrowers_purpose_pivot[1] + borrowers_purpose_pivot[0]
# Сортировка таблицы
borrowers_purpose_pivot.sort_values(1, ascending=False)
```

Out[61]:

	debt	0	1	total
purpose_category	purpose			
получение образования	высшее образование	1245	129	1374
	образование	1189	108	1297
операции с недвижимостью	покупка жилой недвижимости	1163	89	1252
получение образования	дополнительное образование	817	89	906
операции с автомобилем	автомобиль	886	86	972
проведение свадьбы	свадьба	727	64	791
	на проведение свадьбы	704	64	768
	сыграть свадьбу	707	58	765
операции с недвижимостью	операции с недвижимостью	620	55	675
	строительство недвижимости	565	54	619
	покупка жилой недвижимости для сдачи	599	52	651
	операции с коммерческой недвижимостью	598	52	650
операции с автомобилем	сделка с подержанным автомобилем	435	51	486
операции с недвижимостью	операции со своей недвижимостью	577	50	627
операции с автомобилем	сделка с автомобилем	405	50	455
операции с недвижимостью	строительство жилой недвижимости	576	48	624
	операции с жилой недвижимостью	604	48	652
операции с автомобилем	свой автомобиль	430	48	478
операции с недвижимостью	покупка коммерческой недвижимости	614	47	661
операции с автомобилем	на покупку своего автомобиля	459	46	505
операции с недвижимостью	жилая недвижимость	600	46	646
	покупка жилой недвижимости для семьи	593	45	638
получение образования	профильное образование	392	44	436
операции с автомобилем	на покупку автомобиля	427	44	471
операции с недвижимостью	покупка недвижимости	578	43	621
	недвижимость	591	42	633
	строительство собственной недвижимости	593	42	635
операции с автомобилем	приобретение автомобиля	419	42	461
	на покупку подержанного автомобиля	442	36	478
операции с недвижимостью	ремонт жилой недвижимости	572	35	607
	покупка своей жилой недвижимости	586	34	620

Анализ полученной таблицы затруднён ввиду большого количества категорий цели кредита (`'purpose'`), оставленных нами на этапе удаления неявных дубликатов. Он станет возможным только после проработки со специалистами клиентского отдела стандартизованного списка допустимых значений данной категории и внесения операторами исправлений в таблицу заёмщиков.

Тем не менее, количество категорий `'purpose_category'` невелико. Поэтому в рамках данного исследования возможно построение оценки верхнего уровня.

Изменим сводную таблицу `borrowers_purpose_pivot`: индексы возьмём из столбца `'purpose_category'` таблицы `borrowers`, столбцы - из значений индикаторов наличия задолженности `'debt'`. Остальные настройки таблицы оставим прежними.

Кроме того, добавим к ней столбец `'total'`, содержащий общее количество клиентов, взявших кредит для конкретной цели. Результат отсортируем по убыванию значений в столбце `1` (наличие задолженности):

In [62]:
```
# Определение количества просрочек в зависимости от уровня дохода
borrowers_purpose_pivot = \
    borrowers.pivot_table(index='purpose_category', columns='debt',
                          values='total_income', aggfunc='count')
# Добавление столбца 'total'
borrowers_purpose_pivot['total'] = borrowers_purpose_pivot[1] + borrowers_purpose_pivot[0]
# Сортировка таблицы
borrowers_purpose_pivot.sort_values(1, ascending=False)
```

Out[62]:

debt	0	1	total
purpose_category			
операции с недвижимостью	10029	782	10811
операции с автомобилем	3903	403	4306
получение образования	3643	370	4013
проведение свадьбы	2138	186	2324

В полученной таблице наблюдается ситуация, похожая на рассмотренные выше случаи. Однако разброс значений по разным категориям не столь велик, что затрудняет анализ.

Добавим в таблицу столбец `'percent'`, отражающий процентное отношение количества клиентов, имеющих задолженность, к общему количеству клиентов, относящихся к заданной категории цели кредита. Результат отсортируем по убыванию значений в столбце `'percent'`:

In [63]:
```
# Добавление столбца 'percent'
borrowers_purpose_pivot['percent'] = round(borrowers_purpose_pivot[1] * 100 / \
                                           borrowers_purpose_pivot['total'], 2)
# Сортировка таблицы
borrowers_purpose_pivot.sort_values('percent', ascending=False)
```

Out[63]:

debt	0	1	total	percent
purpose_category				
операции с автомобилем	3903	403	4306	9.36
получение образования	3643	370	4013	9.22
проведение свадьбы	2138	186	2324	8.00
операции с недвижимостью	10029	782	10811	7.23

Теперь картина прояснилась. Больше всего просрочек происходит по автокредитам и кредитам на образование. Более всего дисциплинированы клиенты, приобретающие недвижимость. Клиенты, взявшие потребкредиты на проведение свадьбы занимают вторую позицию по дисциплинированности.

Вывод

Гипотеза о том, что существует зависимость между целью кредита и его возвратом в срок, подтвердилась. При этом отмечено, что на имеющихся данных затруднительно делать выводы по подкатегориям цели кредитования (например, по различным видам недвижимости или образования). Для устранения этих затруднений необходима дополнительная работа по очистке данных с привлечением специалистов клиентского отдела банка.

Выводы исследования

Мы проверили четыре гипотезы и установили:

1. Количество детей влияет на возврат кредита в срок.

Гипотеза подтвердилась частично. Бездетные клиенты допускают меньше задолженностей, чем имеющие детей.

Нулевое количество задолженностей для клиентов с 5 детьми можно считать спорным ввиду малой выборки заёмщиков данной категории, представленной в данных.

Наиболее дисциплинированными являются клиенты банка, у которых по 3 ребёнка. В остальном, с ростом количества детей незначительно увеличивается процент задолженностей.

2. Между семейным положением и возвратом кредита в срок существует зависимость.

Гипотеза полностью подтвердилась.

Меньше всего просрочек допускают овдовевшие клиенты, больше всего - холостые или живущие гражданским браком. Разведенные и семейные клиенты имеют среднюю платёжную дисциплину.

3. Уровень дохода влияет на возврат кредита в срок.

Гипотеза полностью подтвердилась.

Худший результат оказался ожидаемым - наиболее бедные клиенты (до 30 тасяч) чаще допускают просрочки. А вот категория наиболее богатых заёмщиков (свыше 1 миллиона) а также "средний класс" (50-200тысяч) показали себя не самыми дисциплинированными плательщиками. Лидерами рейтинга платёжной дисциплины являются клиенты с доходом 30-50 тысяч, за ними следуют клиенты доходом "выше среднего" (от 200 тысяч до миллиона).

4. Существует зависимость между целью кредита и его возвратом в срок.

Гипотеза подтвердилась.

Больше всего просрочек происходит по автокредитам и кредитам на образование. Наиболее дисциплинированы клиенты, приобретающие недвижимость. Клиенты, взявшие потребкредиты на проведение свадьбы занимают вторую позицию по дисциплинированности.

Замечания

1. Исследование проводилось в предположении, что значением 0 кодируется отсутствие, а значением 1 - наличие задолженности, поскольку документация к данным не содержит информации о соответствии булевых значений в столбце 'debt' фактам наличия и отсутствия задолженности. - *Вниманию инженеров по данным.*
2. В ходе исследования отмечено, что на имеющихся данных затруднительно делать выводы по подкатегориям цели кредитования (например, по различным видам недвижимости или образования). Для устранения этих затруднений необходима дополнительная работа по очистке данных с привлечением специалистов клиентского отдела банка. - *Вниманию клиентского отдела.*