

Исследование надёжности заёмщиков

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

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

Шаг 1. Откройте файл с данными и изучите общую информацию

Импортируем библиотеку Pandas. Официальная документация по библиотеке на английском языке [здесь](#).

Интересные материалы на русском языке:

- [Руководство по использованию pandas для анализа больших наборов данных](#)
- [Введение в анализ данных с помощью Pandas](#)

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

Откроем таблицу, загрузив её из файла с помощью функции `read_csv`. Файл можно также скачать через Интернет по [адресу](#).

```
In [2]: data = pd.read_csv('/datasets/data.csv')
```

Выведем **первые 10 записей** таблицы с помощью `head()`:

```
In [3]: data.head(10)
```

```
Out[3]:
```

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender
0	1	-8437.673028	42	высшее	0	женат / замужем	0	F
1	1	-4024.803754	36	среднее	1	женат / замужем	0	F
2	0	-5623.422610	33	Среднее	1	женат / замужем	0	M
3	3	-4124.747207	32	среднее	1	женат / замужем	0	M
4	0	340266.072047	53	среднее	1	гражданский брак	1	F
5	0	-926.185831	27	высшее	0	гражданский брак	1	M
6	0	-2879.202052	43	высшее	0	женат / замужем	0	F

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender
7	0	-152.779569	50	СРЕДНЕЕ	1	женат / замужем	0	M
8	2	-6929.865299	35	ВЫСШЕЕ	0	гражданский брак	1	F
9	0	-2188.756445	41	среднее	1	женат / замужем	0	M

Выведем **общую информацию** о наборе данных с помощью `info` :

In [4]: `data.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
```

Рассмотрим распределения числовых полей набора данных с помощью функции

`describe` .

In [5]: `data.describe()`

Out[5]:

	children	days_employed	dob_years	education_id	family_status_id	debt	total_i
count	21525.000000	19351.000000	21525.000000	21525.000000	21525.000000	21525.000000	19351.000000
mean	0.538908	63046.497661	43.293380	0.817236	0.972544	0.080883	1.674220
std	1.381587	140827.311974	12.574584	0.548138	1.420324	0.272661	1.029700
min	-1.000000	-18388.949901	0.000000	0.000000	0.000000	0.000000	2.066720
25%	0.000000	-2747.423625	33.000000	1.000000	0.000000	0.000000	1.030530
50%	0.000000	-1203.369529	42.000000	1.000000	0.000000	0.000000	1.450170
75%	1.000000	-291.095954	53.000000	1.000000	1.000000	0.000000	2.034350
max	20.000000	401755.400475	75.000000	4.000000	4.000000	1.000000	2.265600

Бегло изучим общую информацию о таблице. Проверим соответствие полей

спецификации задания. Результат проверки сведем в таблицу.

Поле	Данные	Соответствует спецификации	Замечания
children	количество детей в семье	да	Минимальное (-1) и максимальное (20) значения вызывают сомнения в корректности данных. Требуется дополнительная проверка.
days_employed	общий трудовой стаж в днях	нет	Встречаются пустые и отрицательные вещественные значения с большим разбросом , что противоречит смыслу (дни должны быть целыми). В реальной жизни следовало бы уточнить формат этого поля у поставщика данных для предотвращения ошибок его интерпритации и сокращения времени анализа.
dob_years	возраст клиента в годах	да	Поле пригодно для категоризации - получения новых полей, более удобных для анализа.
education	уровень образования клиента	да	Одинаковый по смыслу текст написан и строчными, и прописными буквами, что может затруднить анализ. Например, "среднее", "Среднее", "СРЕДНЕЕ".
education_id	идентификатор уровня образования	да	Возможно является числовой характеристикой уровня образования, т.е. дублирует предыдущее поле education_id.
family_status	семейное положение	да	
family_status_id	идентификатор семейного положения	да	Возможно является числовой характеристикой уровня образования, т.е. дублирует предыдущее поле family_status_id.
gender	пол клиента	да	F - женский, M - мужской. Возможна конвертация поля в логический тип.
income_type	тип занятости	да	
debt	имел ли задолженность по возврату кредитов	да	Возможна конвертация поля в логический тип.
total_income	ежемесячный доход	да	Встречаются пустые значения, которые могут означать, что клиент не имеет работы (в этом случае, возможно, поле days_employed тоже пустое), устроился недавно или не сообщил сведения. Требуется дополнительная проверка.
purpose	цель получения кредита	да	Поле содержит произвольные текстовые данные, анализ которых без лемматизации будет затруднен.

Оценим количество и долю дубликатов в массиве данных с помощью функции

`duplicated :`

```
In [6]: print('Количество записей, имеющих полные дубликаты:', data.duplicated().sum())
```

Количество записей, имеющих полные дубликаты: 54

```
In [7]: print('Доля полных дубликатов: {:.2%}'.format(data.duplicated().sum() / data.shape[0]))
```

Доля полных дубликатов: 0.25%

Вывод

- Полученный файл имеет корректный формат, удобный для обработки с помощью библиотеки Pandas.
- Названия столбцов не искажены и могут быть использованы для упрощенной адресации Pandas в виде `data.dob_years`, а не только `data['dob_years']`. Предобработка названий столбцов не требуется.
- Доля полных дубликатов строк невелика. Их наличие скорее не окажет негативное влияние на дальнейшую работу. Тем не менее, необходимо выбрать наиболее подходящий метод обработки дубликатов строк и повторяющихся значений в некоторых столбцах.
- Массив данных в целом пригоден для анализа, но требует предобработки, включая как минимум:
 - устранение пропусков в полях `days_employed` (общий трудовой стаж в днях) и `total_income` (ежемесячный доход);
 - изменение типа поля `days_employed` (общий трудовой стаж в днях);
 - фильтрацию отрицательных и слишком больших значений в поле `children` (количество детей в семье);
 - удаление дубликатов в столбце `education` (уровень образования клиента), который содержит одинаковый по смыслу текст, написанный строчными и прописными буквами;
 - категоризацию полей `dob_years` (возраст клиента в годах), `income_type` (тип занятости) и др.;
 - лемматизацию поля `purpose` (цель получения кредита);
 - проверку и удаление возможно дублирующих друг друга столбцов - `education` (уровень образования клиента) и `education_id` (идентификатор уровня образования), `family_status` (семейное положение) и `family_status_id` (идентификатор семейного положения).
- Целесообразно вычисление по полученным данным новых характеристик претендента на получение кредита для дальнейшего анализа, например, среднего дохода на члена семьи претендента (с учетом семейного положения и количества детей).

Шаг 2. Предобработка данных

Обработка пропусков

Поле `days_employed` (общий трудовой стаж в днях)

Поле `days_employed` выглядит наиболее запутанным. Его тип не соответствует спецификации. Должен быть целочисленным, реально в массиве данных содержатся положительные и отрицательные вещественные числа с большим разбросом и пропусками:

Проведем детальное изучение структуры полученных данных для определения оптимальных способов их предобработки в интересах облегчения последующего анализа. Используем

условную индексацию и функции `value_counts` и `describe`.

Проверим наличие связи поля `days_employed` с полем `income_type` (тип занятости).

Отрицательные значения `days_employed` только у работающих, включая находящихся в декрете (с сохранением заработной платы), предпринимателя и студента:

```
In [8]: data[data.days_employed < 0]['income_type'].value_counts()
```

```
Out[8]: сотрудник      10014
компаньон      4577
госслужащий    1312
в декрете      1
предприниматель 1
студент        1
Name: income_type, dtype: int64
```

```
In [9]: data[data.days_employed < 0]['days_employed'].describe()
```

```
Out[9]: count      15906.000000
mean      -2353.015932
std       2304.243851
min      -18388.949901
25%      -3157.480084
50%      -1630.019381
75%      -756.371964
max       -24.141633
Name: days_employed, dtype: float64
```

Положительные значения `days_employed` характерны только для пенсионеров и безработных. При этом среднее значение стремится к 365004 дням (примерно 1000 лет):

```
In [10]: data[data.days_employed > 0]['income_type'].value_counts()
```

```
Out[10]: пенсионер      3443
безработный      2
Name: income_type, dtype: int64
```

```
In [11]: data[data.days_employed > 0]['days_employed'].describe()
```

```
Out[11]: count      3445.000000
mean      365004.309916
std       21075.016396
min      328728.720605
25%      346639.413916
50%      365213.306266
75%      383246.444219
max      401755.400475
Name: days_employed, dtype: float64
```

Нулевые значения `days_employed` отсутствуют:

```
In [12]: data[data.days_employed == 0]['income_type'].value_counts().count()
```

```
Out[12]: 0
```

В `days_employed` имеются и пропуски. Наибольшее количество пропусков у "сотрудников". Есть пропуски и у пенсионеров:

```
In [13]: data[data.days_employed.isnull()]['income_type'].value_counts()
```

```
Out[13]: сотрудник      1105
компаньон      508
пенсионер      413
госслужащий    147
предприниматель 1
Name: income_type, dtype: int64
```

Пустые значения `days_employed` сопровождаются пропусками в `total_income` и наоборот. Эти два поля однозначно связаны: отсутствие данных в одном означает их отсутствие и в другом:

```
In [14]: print('Количество записей с пустыми полями days_employed И total_income:',
            data[data.days_employed.isnull() & data.total_income.isnull()]['children'].count())
print('Количество записей с пустыми полями days_employed ИЛИ total_income:',
            data[data.days_employed.isnull() | data.total_income.isnull()]['children'].count())
```

```
Количество записей с пустыми полями days_employed И total_income: 2174
Количество записей с пустыми полями days_employed ИЛИ total_income: 2174
```

Обращает на себя внимание большая **корреляция** между `days_employed` и `dob_years` (можно посчитать с помощью функций `corr` и `corrwith`).

```
In [15]: data[['days_employed', 'dob_years', 'total_income']].corr()
```

```
Out[15]:
```

	days_employed	dob_years	total_income
days_employed	1.000000	0.582643	-0.136648
dob_years	0.582643	1.000000	-0.052911
total_income	-0.136648	-0.052911	1.000000

Логично предположить, что чем больше возраст претендента, тем больше его общий трудовой стаж в днях (и следовательно в годах). Проверим эту гипотезу:

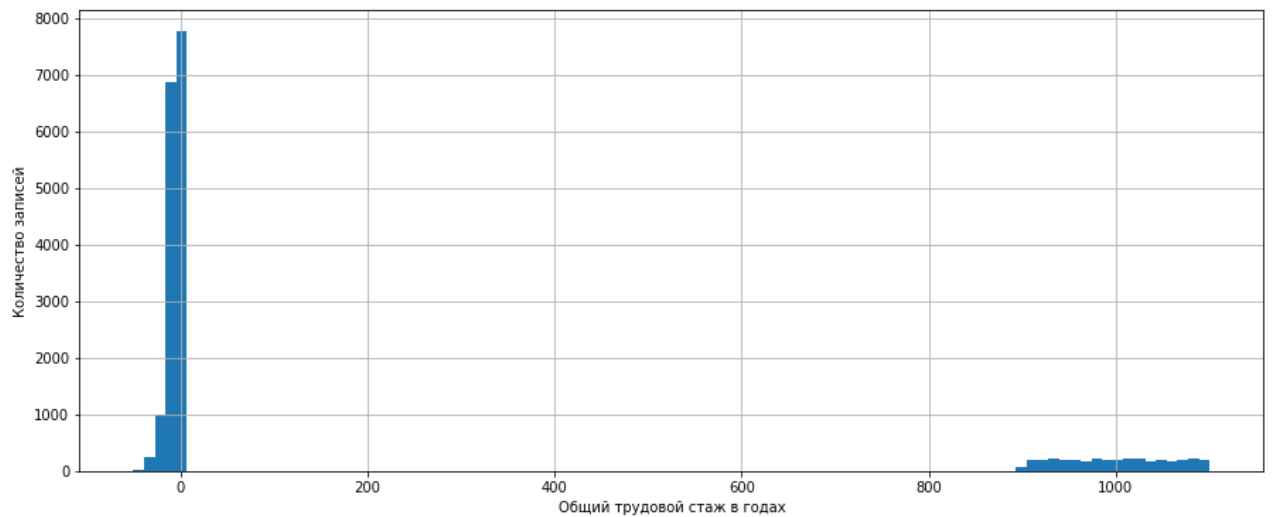
```
In [16]: # вычислим предполагаемый общий трудовой стаж в годах
years_employed = data.days_employed / 365
```

```
In [17]: # изучим характеристики распределения трудового стажа в годах у обратившихся за кредитом
years_employed.describe()
```

```
Out[17]: count      19351.000000
mean         172.730131
std          385.828252
min          -50.380685
25%          -7.527188
50%          -3.296903
75%          -0.797523
max          1100.699727
Name: days_employed, dtype: float64
```

Убеждаемся в большом разбросе общего трудового стажа "в годах" и наличии двух больших групп значений - положительных (как было установлено выше, характерных для пенсионеров) и отрицательных (для работающих):

```
In [18]: ax = years_employed.hist(bins=100, figsize = (15, 6))
ax.set_xlabel('Общий трудовой стаж в годах')
ax.set_ylabel('Количество записей');
```



Можно предположить два возможных варианта объяснения аномалии данных в поле `days_employed` :

1. В данные закралась ошибка. У пенсионеров и безработных в поле `days_employed` хранятся случайные значения, а у работающих изменен знак на противоположный.
2. В поле `days_employed` хранится не заявленный в спецификации общий трудовой стаж в днях, а значение какой-то функции многих переменных, характеризующей надежность источника дохода потенциального клиента. Данная функция вероятно зависит не только от трудового стажа, но и пенсионного статуса.

Не имея возможности связаться с поставщиком данных, будем далее исходить из первого варианта.

Скорректируем поле `days_employed` , сохранив его копию в `days_employed_backup` на будущее, следующим образом:

- для пенсионеров и безработных установим его в ноль;
- для остальных поменяем знак на противоположный;
- пропуски заполним нулями (Исходим из того, что клиент, не указавший доход, снижает свои шансы на получение кредита. В задаче машинного обучения, возможно, пришлось бы заменить пропуски на средние или медианные значения дохода для аналогичной группы клиентов, объединенным по схожести нескольких категориальных признаков).

```
In [19]: data['days_employed_backup'] = data['days_employed']
```

```
In [20]: data.loc[data['days_employed'] > 0, 'days_employed'] = 0
```

```
In [21]: data.loc[data['days_employed'] < 0, 'days_employed'] *= -1
```

```
In [22]: data.days_employed.fillna(0, inplace=True)
```

Убедимся в отсутствии пропусков в поле `days_employed` :

```
In [23]: data.days_employed.isna().sum()
```

```
Out[23]: 0
```

В дальнейшем можно пересчитать поле `days_employed`, если у банка изменится стратегия предоставления кредитов. Сейчас, обнулив пропуски в стаже, мы придерживаемся жестких правил, направленных на снижение риска невозврата займа. Если банк будет располагать финансовыми возможностями для игры на привлечение новых клиентов и повышение доходов, то мы будем рассматривать обращение потенциального заемщика с отсутствующими по каким-либо причинам данными о стаже, вычислив их на основе средних для похожих по некоторым категориям клиентов. Но для этого ниже необходимо определить эти категории (источник дохода, возраст и т.п.). А пока политика банка жесткая ограничимся обнулением пропусков в данных.

Проверка адекватности полученных данных по стажу работы

Проверим адекватность полученных данных, вычислив на их основе возраст начала трудовой деятельности клиентов. Переведем стаж в года и вычтем из возраста. Округлим до целых:

```
In [24]: work_start_age = (data.dob_years - data.days_employed / 365).round(decimals=0)
```

Посчитаем количество записей с неадекватным возрастом начала трудовой деятельности:

```
In [25]: # зададим адекватный возраст
appropriate_age = 14
print('Список неадекватных возрастов и их количество в наборе данных:')
display(work_start_age[work_start_age < appropriate_age].value_counts().sort_values(asc
```

Список неадекватных возрастов и их количество в наборе данных:

0.0	30
-3.0	12
-4.0	10
-1.0	9
-2.0	8
-5.0	7
-7.0	5
13.0	5
12.0	4
-12.0	3
-6.0	3
-13.0	3
-10.0	2
-9.0	2
-29.0	2
-14.0	2
11.0	2
-16.0	1
-22.0	1
-18.0	1

dtype: int64

```
In [26]: inadequate_count = work_start_age[work_start_age < appropriate_age].count()
print('Количество записей с неадекватным соотношением стажа и возраста:', inadequate_co
```

Количество записей с неадекватным соотношением стажа и возраста: 112

```
In [27]: print('Доля записей с неадекватным соотношением стажа и возраста: {:.2%}'.format(inadeq
```

Доля записей с неадекватным соотношением стажа и возраста: 0.52%

Встречаются даже отрицательные значения. Неадекватное соотношение стажа и возраста у небольшого количества записей может быть вызвано ошибками ввода данных операторами в

поля `dob_years` и `data.days_employed` или сообщением клиентами недостоверных сведений о себе.

Удаление этих записей не скажется на результатах дальнейшей работы (их доля меньше одного процента). Воспользуемся функцией `drop`:

```
In [28]: data.drop(work_start_age[work_start_age < appropriate_age].index, inplace=True)
```

Поле `total_income` (ежемесячный доход)

Оценим количество и долю пропусков в данных о ежемесячном доходе:

```
In [29]: print('Количество записей с пропусками в данных о ежемесячном доходе:', data.total_inco
```

Количество записей с пропусками в данных о ежемесячном доходе: 2164

```
In [30]: print('Доля записей с пропусками в данных о ежемесячном доходе: '
              '{:.2%}'.format(data.total_income.isnull().sum() / data.shape[0]))
```

Доля записей с пропусками в данных о ежемесячном доходе: 10.11%

Удаление десятой части данных может негативно сказаться на результатах решения задачи кредитного скоринга при реализации стратегии на привлечение новых клиентов. Мы можем упустить надежного клиента из-за отсутствия в его обращении некоторых данных.

В данном учебном проекте обнулим пропуски, но на будущее заведем в наборе данных признак `calculated_income`, который будет принимать следующие значения: `False` - данные о доходе были представлены в первоначальном наборе, `True` - данные о доходе получены из других признаков заемщика и статистики других клиентов.

```
In [31]: data['calculated_income'] = data.total_income.isnull()
```

```
In [32]: data.total_income.fillna(0, inplace=True)
```

Поле `children` (количество детей в семье)

В поле `children` (количество детей в семье) обнаружены недопустимое (`-1`) и выбивающееся из ряда (`20`) значения:

```
In [33]: data.children.value_counts().sort_index()
```

```
Out[33]: -1      47
          0    14071
          1    4800
          2    2042
          3     328
          4     41
          5       9
          20     75
          Name: children, dtype: int64
```

Изучим распределение возраста клиентов с разным количеством детей:

```
In [34]: data.pivot_table(index = 'children', values = 'dob_years', aggfunc=['min', 'mean', 'max
```

```
Out[34]:
```

	min	mean	max
dob_years	dob_years	dob_years	dob_years

children	min	mean	max
dob_years	dob_years	dob_years	dob_years
children			
-1	23	42.574468	69
0	19	46.488949	75
1	20	38.491250	73
2	20	35.998041	64
3	23	36.509146	63
4	24	36.048780	51
5	31	38.777778	59
20	21	42.373333	69

Обнаружено подтверждение искажения некоторых данных - самому молодому родителю двадцати детей исполнился лишь 21 год. Исходя из оптимистичного здравого смысла, такого выдающегося результата можно в лучшем случае достичь годам к 30.

В учебных целях будем считать, что мы проверили ряд гипотез о причинах искажения данных. Было установлено, что -1 - неправильно распознанные сканером четверки 4 (автор программы не закончил в своё время курсы [Яндекс.Практикум](#), а 20 - ошибки операторов при вводе с клавиатуры (при нажатии на клавишу 2 в правой части 105-клавишной клавиатуры они иногда случайно задевали ещё и 0, расположенный чуть ниже).



Исправим обнаруженные опечатки:

```
In [35]: data.loc[data.children == -1, 'children'] = 4
```

```
In [36]: data.loc[data.children == 20, 'children'] = 2
```

Теперь поле `children` (количество детей в семье) не содержит аномалий:

```
In [37]: data.children.value_counts().sort_index()
```

```
Out[37]: 0    14071
         1     4800
         2     2117
         3       328
```

```
4      88
5      9
Name: children, dtype: int64
```

Вывод

1. Значения в поле `days_employed` (общий трудовой стаж в днях) пересчитаны в соответствии с его предназначением. Исправлены искажения в данных. Первоначальные данные сохранены в `days_employed_backup` для возможного использования в дальнейшем.
2. После восстановления пропусков удалены записи с неадекватным возрастом начала трудовой деятельности.
3. Пропуски в полях `days_employed` (общий трудовой стаж в днях) и `total_income` (ежемесячный доход) заполнены нулевыми значениями.
4. Заведено поле `calculated_income` с признаком обнуления поля `total_income` для возможности при необходимости заменить нули на оценку дохода по статистике других клиентов.
5. Исправлены опечатки в полях с информацией о количестве детей (`children`) в анкетах кандидатов.

Замена типа данных

Поле `days_employed` (общий трудовой стаж в днях)

Изменим тип поля на целочисленный в соответствии с его математическим смыслом (общий трудовой стаж в днях не должен быть вещественным) с помощью функции `astype` и проконтролируем выполнение:

```
In [38]: data.astype({'days_employed': 'int64'}).dtypes
```

```
Out[38]: children          int64
days_employed          int64
dob_years              int64
education              object
education_id          int64
family_status          object
family_status_id      int64
gender                 object
income_type            object
debt                   int64
total_income           float64
purpose                object
days_employed_backup  float64
calculated_income      bool
dtype: object
```

Поле `gender` (пол клиента)

В наборе данных присутствуют записи с отклонением от естественных вариантов пола.

Помимо мужского (М) и женского (F) обнаружена одна запись с отметкой XNA, о значении которой можно только догадываться (возможно, "третий пол"):


```
In [39]: data.gender.value_counts()
```

```
Out[39]: F      14154
M      7258
XNA      1
Name: gender, dtype: int64
```

```
In [40]: data[data.gender == 'XNA']
```

```
Out[40]:
```

	children	days_employed	dob_years	education	education_id	family_status	family_status_id
10701	0	2358.600502	24	неоконченное высшее	2	гражданский брак	1



В учебных целях не будем принимать во внимание толерантность и удалим эту запись:

```
In [41]: data.drop(data[data.gender == 'XNA'].index, inplace=True)
```

```
In [42]: data[data.gender == 'XNA'].count()
```

```
Out[42]: 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
days_employed_backup  0
calculated_income     0
dtype: int64
```

Оперировать данными логического типа, как представляется, удобнее. Создадим новое поле `gender_male` логического типа со значениями `True` для мужского пола и `False` - для женского. Воспользуемся функцией `astype` :

```
In [43]: data['gender_male'] = (data.gender == 'M').astype('bool')
```

Избыточное поле `gender` теперь можно удалить из набора данных с помощью функции `drop` :

```
In [44]: data.drop(columns='gender', inplace=True, errors='ignore')
```

Поле `debt` (имел ли задолженность по возврату кредитов)

Изменим тип поля `debt` (имел ли задолженность по возврату кредитов) на логический. Проконтролируем выполнение:

```
In [45]: data.astype({'debt': 'bool'}).dtypes
```

```
Out[45]: children          int64
days_employed        float64
dob_years             int64
education             object
education_id         int64
```

```
family_status      object
family_status_id   int64
income_type        object
debt               bool
total_income       float64
purpose            object
days_employed_backup float64
calculated_income  bool
gender_male        bool
dtype: object
```

Вывод

Изменены типы полей на подходящие под их описание или более удобные с точки зрения дальнейшей обработки.

Обработка дубликатов

Удаление полных дубликатов

Изучим образцы дубликатов и убедимся в их ненужности для дальнейшей работы:

```
In [46]: data[data.duplicated(keep=False)].head(5)
```

```
Out[46]:
```

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	income
120	0	0.0	46	среднее	1	женат / замужем	0	сс
520	0	0.0	35	среднее	1	гражданский брак	1	сс
541	0	0.0	57	среднее	1	женат / замужем	0	сс
554	0	0.0	60	среднее	1	женат / замужем	0	сс
680	1	0.0	30	высшее	0	женат / замужем	0	госсл

Возможные причины появления полных дубликатов:

- неоднократное обращение одного клиента за кредитом (в разное время и/или в разных отделениях банка);
- ввод данных одного заявления несколько раз из-за ошибки оператора;
- за кредитом могли обращаться люди с некоторыми одинаковыми данными.
Идентифицировать их можно только по отсутствующим персональным данным, которые были скрыты от аналитика в соответствии с требованиями законодательства.

Удалим полные дубликаты с помощью функции `drop_duplicates`, оставив в наборе данных только уникальные строки:

```
In [47]: data.drop_duplicates(inplace=True)
```

Убедимся в отсутствии дубликатов:

```
In [48]: data.duplicated().sum()
```

```
Out[48]: 0
```

Поля `education` (уровень образования клиента) и `education_id` (идентификатор уровня образования клиента)

Поля `education` (уровень образования клиента) и `education_id` (идентификатор уровня образования клиента) дублируют друг друга по смысловому содержанию. Только в поле `education` встречаются одинаковые строки, записанные в разной форме. Возможная причина - массив готовили разные люди, которые по разному вводили текстовые данные, а их контроль в программе не осуществлялся:

```
In [49]: data.groupby(by=['education_id', 'education'], sort=True)['education_id'].count()
```

```
Out[49]: education_id  education
0      ВЫСШЕЕ          272
      Высшее          266
      высшее         4677
1      СРЕДНЕЕ          770
      Среднее          708
      среднее        13636
2      НЕОКОНЧЕННОЕ ВЫСШЕЕ    29
      Неоконченное высшее    47
      неоконченное высшее    665
3      НАЧАЛЬНОЕ           17
      Начальное            15
      начальное           250
4      УЧЕНАЯ СТЕПЕНЬ         1
      Ученая степень         1
      ученая степень         4
Name: education_id, dtype: int64
```

Для упрощения обработки можно преобразовать оба поля в словарь, переведя `education` в нижний регистр. Применим функции `groupby`, `first`, `str.lower` и `to_dict`:

```
In [50]: education_dict = data.groupby(by='education_id', sort=True)['education'].first().str.lower()
print('Словарь образования education_dict:', education_dict)
```

```
Словарь образования education_dict: {0: 'высшее', 1: 'среднее', 2: 'неоконченное высшее', 3: 'начальное', 4: 'ученая степень'}
```

Если в задании не накладывается ограничений на поле `education_id` (идентификатор уровня образования клиента), то ему можно придать большее смысловое значение, сопоставив с отсортированным по значимости набором уровней образования ('начальное', 'среднее', 'неоконченное высшее', 'высшее', 'ученая степень') - чем выше образование, тем больше `education_id`:

```
In [51]: # опишем уровни образования в новом порядке
educations = ('начальное', 'среднее', 'неоконченное высшее', 'высшее', 'ученая степень')
```

```
In [52]: # обновим поле в соответствии с новым порядком уровней образования
data['education_id'] = data['education'].str.lower().apply(func=lambda e: educations.index(e))
```

Перестроим словарь `education_dict` и убедимся в его корректности:

```
In [53]: education_dict = data.groupby(by='education_id', sort=True)['education'].first().str.lower()
print('Словарь образования education_dict:', education_dict)
```

Словарь образования education_dict: {0: 'начальное', 1: 'среднее', 2: 'неоконченное высшее', 3: 'высшее', 4: 'ученая степень'}

```
In [54]: print('Список ключей в словаре образования:', list(education_dict.keys()))
print('Список значений поля education_id:', sorted(list(data['education_id'].unique())))
```

Список ключей в словаре образования: [0, 1, 2, 3, 4]

Список значений поля education_id: [0, 1, 2, 3, 4]

Избыточное поле `education` теперь можно удалить из набора данных с помощью функции `drop`:

```
In [55]: data.drop(columns='education', inplace=True, errors='ignore')
```

Поля `family_status` (семейное положение) и `family_status_id` (идентификатор семейного положения)

Поля `family_status` (семейное положение) и `family_status_id` (идентификатор семейного положения) дублируют друг друга по смысловому содержанию:

```
In [56]: data.groupby(by=['family_status_id', 'family_status'], sort=True)['family_status_id'].count()
```

```
Out[56]: family_status_id  family_status
0                женат / замужем      12285
1                гражданский брак      4140
2                вдовец / вдова        954
3                в разводе            1185
4                Не женат / не замужем  2794
Name: family_status_id, dtype: int64
```

Для упрощения обработки можно преобразовать оба поля в словарь, переводя

`family_status` в нижний регистр. Применим функции `groupby`, `first`, `str.lower` и `to_dict`:

```
In [57]: family_status_dict = data.groupby(by='family_status_id', sort=True)['family_status'].first().str.lower()
print('Словарь семейного положения family_status_dict:')
print(family_status_dict)
```

Словарь семейного положения family_status_dict:

{0: 'женат / замужем', 1: 'гражданский брак', 2: 'вдовец / вдова', 3: 'в разводе', 4: 'не женат / не замужем'}

```
In [58]: family_status_to_id_dict = dict(zip(family_status_dict.values(), family_status_dict.keys()))
print('Словарь идентификаторов семейного положения family_status_to_id_dict:')
print(family_status_to_id_dict)
```

Словарь идентификаторов семейного положения family_status_to_id_dict:

{'женат / замужем': 0, 'гражданский брак': 1, 'вдовец / вдова': 2, 'в разводе': 3, 'не женат / не замужем': 4}

Избыточное поле `family_status` теперь можно удалить из набора данных с помощью функции `drop`:

```
In [59]: data.drop(columns='family_status', inplace=True, errors='ignore')
```

Поле `income_type` (тип занятости)

Поле `income_type` (тип занятости) содержит много текстовых дубликатов:

```
In [60]: data.income_type.value_counts()
```

```
Out[60]: сотрудник      11027
компаньон      5057
пенсионер      3817
госслужащий    1451
безработный     2
предприниматель 2
студент         1
в декрете       1
Name: income_type, dtype: int64
```

Полученный массив данных не содержит смысловых ошибок, по крайней мере, мужчин в декрете не обнаружено:

```
In [61]: data.loc[(data.income_type == 'в декрете') & (data.gender_male)]['gender_male'].count()
```

```
Out[61]: 0
```

Целесообразно избавиться от избыточности дублированных текстовых данных и перейти к словарю, который будет одновременно классифицировать клиентов по типу занятости:

```
In [62]: items = list(data.income_type.unique())
keys = list(range(len(items)))

income_type_dict = dict(zip(keys, items))
print('Словарь типов занятости income_type_dict:')
display(income_type_dict)
```

Словарь типов занятости `income_type_dict`:

```
{0: 'сотрудник',
1: 'пенсионер',
2: 'компаньон',
3: 'госслужащий',
4: 'безработный',
5: 'предприниматель',
6: 'студент',
7: 'в декрете'}
```

```
In [63]: income_type_to_id_dict = dict(zip(items, keys))
print('Словарь идентификаторов типов занятости income_type_to_id_dict:')
display(income_type_to_id_dict)
```

Словарь идентификаторов типов занятости `income_type_to_id_dict`:

```
{'сотрудник': 0,
'пенсионер': 1,
'компаньон': 2,
'госслужащий': 3,
'безработный': 4,
'предприниматель': 5,
'студент': 6,
'в декрете': 7}
```

Создадим в наборе данных новое поле `income_type_id` (идентификатор типа занятости):

```
In [64]: data['income_type_id'] = data.income_type.apply(func=lambda t: dict(zip(items, keys))[t])
```

Избыточное поле `income_type` теперь можно удалить из набора данных с помощью функции `drop`:


```
In [65]: data.drop(columns='income_type', inplace=True, errors='ignore')
```

Вывод

1. Удалены полные дубликаты строк массива данных, т.к. в данной задаче их обработка привела бы к лишним операциям без улучшения качества решения.
2. Исправлены написанные разным регистром дубликаты в полях, что избавит от ошибок обработки данных в дальнейшем анализе.

В соответствии с полученным советом **оценим количество и долю дубликатов** в массиве данных с помощью функции `data.duplicated`:

```
In [66]: print('Количество записей, имеющих полные дубликаты:', data.duplicated().sum())
```

Количество записей, имеющих полные дубликаты: 17

```
In [67]: print('Доля полных дубликатов: {:.2%}'.format(data.duplicated().sum() / data.shape[0]))
```

Доля полных дубликатов: 0.08%

Удалим полные дубликаты с помощью функции `data.drop_duplicates`, оставив в наборе данных только уникальные строки:

```
In [68]: data.drop_duplicates(inplace=True)
```

Убедимся в отсутствии дубликатов:

```
In [69]: data.duplicated().sum()
```

```
Out[69]: 0
```

Вывод по совету: поиск полных дубликатов лучше производить после нормализации и категоризации отдельных полей.

Лемматизация

Проведем [лемматизацию](#) текста в поле `purpose` (цель получения кредита) с помощью библиотеки [pymorphy2](#). Её скорости вполне достаточно для данной учебной задачи. А качество [pymystem3](#) прибережем для будущих серьезных проектов. В учебных целях разработаем свои функции и не будем обращаться к библиотеке [Natural Language Toolkit](#), которая могла бы существенно упростить работу.

```
In [70]: import pymorphy2
```

Используем счетчик [Counter](#) из библиотеки [collections](#) для подсчета частоты встречаемости слов в текстовом описании целей получения кредита:

```
In [71]: from collections import Counter
```

Для разделения слов используем [регулярные выражения](#) и библиотеку [re](#):

```
In [72]: import re
```

Опишем функцию лемматизации одной строки:

```
In [73]: def lemmatize_text(text, min_len=1, pymorph=None, global_counter=None, words_count=3, s
        """
        Функция производит лемматизацию строки text с помощью лемматизатора pymorph (pymorph
        Во внимание принимаются только русские буквы. Все остальные символы считаются разде
        Если pymorph не задан использует свой экземпляр pymorphy2.MorphAnalyzer.
        stemmer целесообразно задавать при обработке больших массивов Pandas или в цикле дл
        времени выполнения.
        Если задан счетчик global_counter (collections.Counter), то в него добавляются корт
        words_count, состоящие из найденных рядом нормальных форм и их частота упоминания.

        Возвращает строку с нормальными формами слов из строки text, разделенными пробелами
        Нормальные формы длиной меньше min_len опускаются.
        """
        # создадим лемматизатор, если он не задан
        if pymorph is None:
            pymorph = pymorphy2.MorphAnalyzer()

        # получим список нормальных форм всех слов из русских букв
        words = [pymorph.parse(w)[0].normal_form for w in re.findall(r'[а-яё]+', text.lower

        # оставим только нормальные формы длиной не менее min_len
        words = [w for w in words if len(w) >= min_len]

        # при необходимости заменяем синонимы
        if synonyms is not None:
            words = [synonyms[w] if w in synonyms else w for w in words]
            words = [w for w in words if w is not None]

        # обновим внешний счетчик
        if global_counter is not None:
            # построим кортежи слов подряд длиной words_count (или меньше, если нормальных
            word_tuples = [tuple(words[i : i + words_count]) for i in range(max(1, len(word
            # увеличим частоты встречаемости кортежей слов
            global_counter += Counter(word_tuples)

        # вернем нормальные формы через пробел
        return ' '.join(words)
```

Проверим её работоспособность:

```
In [74]: test_text = 'Ехали медведи на велосипеде, а за ними мы с Питоном, Пандами и Юпитером
        'почитывая Notion, и официальную документацию!'
```

```
In [75]: lemmatize_text(test_text)
```

```
Out[75]: 'ехать медведь на велосипед а за они мы с питон панда и юпитер разговаривать в слэк и по
        читывать и официальный документация'
```

А теперь проведем лемматизацию с заменой синонимов, исключая слово официальный и принимая нормы длиной не меньше 3 символов:

```
In [76]: synonyms = {'медведь': 'животное', 'питон': 'язык', 'панда': 'библиотека', 'юпитер': 'п
```

```
In [77]: lemmatize_text(test_text, min_len=3, synonyms=synonyms)
```

```
Out[77]: 'ехать животное велосипед они язык библиотека программа разговаривать слэк почитать до
```

кументация'

Опишем функцию лемматизации столбца таблицы Pandas:

```
In [78]: def lemmatize_pandas_column(data, column, min_len=1, words_count=3, synonyms=None):
        """
        Сканирует столбец с именем column таблицы data.
        Возвращает лемматизированный столбец Pandas и счетчик упоминаемости нормальных форм
        длиной не более min_len символов.
        """
        # создадим экземпляр лемматизатора
        pymorph = pymorphy2.MorphAnalyzer()

        # обнулим счетчик
        purposes_dict = Counter()

        # возвратим лемматизированный столбец и частоты фраз в нем
        return data[column].apply(func=lambda x :
                                   lemmatize_text(x, min_len, pymorph, purposes_dict, words_
                                   purposes_dict
```

Проведем лемматизацию, игнорируя слова короче трёх символов. Нормализованный текст с описанием цели кредита поместим в новое поле `purpose_norm` :

```
In [79]: data['purpose_norm'], purposes_dict_3_1 = lemmatize_pandas_column(data, 'purpose', 3, 1
```

Видим, что нормализованный словарь целей кредита имеет вполне конечный размер:

```
In [80]: sorted(purposes_dict_3_1.items(), key=lambda x: x[1], reverse=True)
```

```
Out[80]: [ (('недвижимость',), 6321),
  (('покупка',), 5867),
  (('жильё',), 4435),
  (('автомобиль',), 4283),
  (('образование',), 3993),
  (('операция',), 2591),
  (('свадьба',), 2309),
  (('свой',), 2223),
  (('строительство',), 1871),
  (('высокий',), 1365),
  (('получение',), 1307),
  (('коммерческий',), 1305),
  (('для',), 1286),
  (('жила',), 1223),
  (('поддержать',), 962),
  (('сделка',), 938),
  (('дополнительный',), 901),
  (('заняться',), 900),
  (('проведение',), 764),
  (('сыграть',), 760),
  (('сдача',), 649),
  (('семья',), 637),
  (('собственный',), 632),
  (('ремонт',), 605),
  (('приобретение',), 459),
  (('профильный',), 435)]
```

В нём встречаются синонимы, например "покупка" и "приобретение".

А вот так выглядит текст после нормализации:

```
In [81]: display(data['purpose_norm'])
```

```

0                покупка жильё
1    приобретение автомобиль
2                покупка жильё
3    дополнительное образование
4                сыграть свадьба
...
21520            операция жильё
21521            сделка автомобиль
21522            недвижимость
21523    покупка свой автомобиль
21524            покупка автомобиль
Name: purpose_norm, Length: 21341, dtype: object

```

Проведем лемматизацию, игнорируя слова короче трёх символов и объединяя их по возможности в биграммы:

Объединим близкие по смыслу слова и удалим незначащие, используя словарь синонимов:

```
In [82]: purpose_synonyms = {'приобретение': 'покупка', 'операция': 'сделка', 'сдача': 'сделка',
                             'проведение': None, 'сыграть': None, 'поддержать': None, 'свой': None,
                             'заняться': None, 'получение': None}
```

```
In [83]: data['purpose_norm'], purposes_dict_3_2 = lemmatize_pandas_column(data, 'purpose', 3, 2)
```

Размер словаря существенно не увеличился:

```
In [84]: print('Размер словаря фраз из одного слова:', len(purposes_dict_3_1))
          print('Размер словаря фраз до двух слов:   ', len(purposes_dict_3_2))
```

```

Размер словаря фраз из одного слова: 26
Размер словаря фраз до двух слов:    24

```

```
In [85]: sorted(purposes_dict_3_2.items(), key=lambda x: x[1], reverse=True)
```

```

Out[85]: [ (('покупка', 'жильё'), 3147),
           (('свадьба',), 2309),
           (('покупка', 'автомобиль'), 1905),
           (('автомобиль',), 1440),
           (('высокий', 'образование'), 1365),
           (('коммерческий', 'недвижимость'), 1305),
           (('сделка', 'недвижимость'), 1298),
           (('образование',), 1292),
           (('жильё', 'для'), 1286),
           (('строительство', 'недвижимость'), 1251),
           (('жильё', 'недвижимость'), 1223),
           (('сделка', 'автомобиль'), 938),
           (('дополнительный', 'образование'), 901),
           (('покупка', 'коммерческий'), 658),
           (('для', 'сделка'), 649),
           (('сделка', 'коммерческий'), 647),
           (('сделка', 'жильё'), 646),
           (('жильё',), 640),
           (('для', 'семья'), 637),
           (('недвижимость',), 628),
           (('строительство', 'жильё'), 620),
           (('покупка', 'недвижимость'), 616),
           (('ремонт', 'жильё'), 605),
           (('профильный', 'образование'), 435)]

```

Проведем лемматизацию, игнорируя слова короче трёх символов и объединяя их по возможности в триграммы:

```
data['purpose_norm'], purposes_dict_3_3 = lemmatize_pandas_column(data, 'purpose', 3, 3)
```

In [86]:

Размер словаря существенно не изменился:

In [87]:

```
print('Размер словаря фраз из одного слова:', len(purposes_dict_3_1))
print('Размер словаря фраз до двух слов:   ', len(purposes_dict_3_2))
print('Размер словаря фраз до трех слов:   ', len(purposes_dict_3_3))
```

Размер словаря фраз из одного слова: 26

Размер словаря фраз до двух слов: 24

Размер словаря фраз до трех слов: 23

Изучим словарь фраз лемматизированных целей кредита, отсортировав их по популярности:

In [88]:

```
sorted(purposes_dict_3_3.items(), key=lambda x: x[1], reverse=True)
```

Out[88]:

```
[(('свадьба',), 2309),
 (('покупка', 'автомобиль'), 1905),
 (('автомобиль',), 1440),
 (('высокий', 'образование'), 1365),
 (('сделка', 'недвижимость'), 1298),
 (('образование',), 1292),
 (('покупка', 'жильё', 'для'), 1286),
 (('покупка', 'жильё'), 1258),
 (('строительство', 'недвижимость'), 1251),
 (('сделка', 'автомобиль'), 938),
 (('дополнительный', 'образование'), 901),
 (('покупка', 'коммерческий', 'недвижимость'), 658),
 (('жильё', 'для', 'сделка'), 649),
 (('сделка', 'коммерческий', 'недвижимость'), 647),
 (('сделка', 'жильё'), 646),
 (('жильё',), 640),
 (('жильё', 'для', 'семья'), 637),
 (('недвижимость',), 628),
 (('строительство', 'жильё', 'недвижимость'), 620),
 (('покупка', 'недвижимость'), 616),
 (('ремонт', 'жильё'), 605),
 (('покупка', 'жильё', 'недвижимость'), 603),
 (('профильный', 'образование'), 435)]
```

Малое увеличение словаря при увеличении размера картежа от одного до трех слов позволяет предположить, что категоризацию цели кредита можно осуществить по небольшому списку ключевых слов, наиболее точно описывающих ту или иную категорию. В данной работе не будем применять более эффективные, но сложные алгоритмы, оставив их на предстоящие специальные курсы.

Вывод

Проведена лемматизация текста в поле с описанием цели кредита. Создан новый столбец с нормальными формами слов. В дальнейшем можно проводить категоризацию целей, выполняя поиск по ключевым словам в новом поле.

Категоризация данных

Категоризация по цели кредита

Просмотрев [словарь лемматизированных целей кредита](#) сформулируем принцип классификации, выделив в учебном проекте только наиболее значимые причины обращения

за займом и их признаки. Оформим поясняющую таблицу:

Категория	Признаки и значения категории	Название нового поля
Займ для сделки	True - если кредит берется на обеспечение сделки с участием других лиц (т.е. возврат зависит не только от заемщика, но и в некотором смысле от его партнеров по сделке). Такие случаи могут возникать, например, при обмене квартир.	for_contract
	False - клиент берет займ на приобретение чего-либо самостоятельно.	
Объект займа	Будем исходить из того, что основная цель займа всегда одна и может принимать следующие значения:	purpose_detail_id
	Проведение свадьбы.	
	Покупка автомобиля.	
	Получение образования без учета его уровня (хотя в реальной задаче кредитного скоринга эту информацию можно было бы учитывать, в том числе для оценки правдивости клиента, сопоставляя цель с другими полями - возраст, имеющееся образование, количество детей).	
	Покупка жилья.	
	Покупка коммерческой недвижимости - будем предполагать, что этот класс заемщиков существенно отличается от людей, желающих справить новоселье.	
	Покупка прочей недвижимости , которая не описана как жилая или коммерческая. Сюда могут относиться гаражи, места в подземных стоянках и т.п.	
	Строительство новых объектов.	
	Ремонт собственности.	
	Прочая цель , не попавшая в классы выше.	

Выполним категоризацию по типу займа, разделив их на два класса **Займ для сделки** и **Займ для себя**, в зависимости от упоминания в цели слова 'сделка':

```
In [89]: def is_for_contract(s):
         return s.find('сделка') != -1

In [90]: data['for_contract'] = data['purpose_norm'].apply(is_for_contract)

In [91]: data[['purpose_norm', 'for_contract']]

Out[91]:
```

	purpose_norm	for_contract
0	покупка жильё	False
1	покупка автомобиль	False
2	покупка жильё	False
3	дополнительный образование	False

	purpose_norm	for_contract
4	свадьба	False
...
21520	сделка жильё	True
21521	сделка автомобиль	True
21522	недвижимость	False
21523	покупка автомобиль	False
21524	покупка автомобиль	False

21341 rows × 2 columns

Выполним категоризацию **по типу объекту займа**, разделив их на несколько классов, в зависимости от упоминания в цели соответствующих ключевых слов.

Зададим словарь описаний классов целей и лямбда-функций проверки принадлежности цели этим классам:

```
In [92]: object_criteria_dict = {
    'проведение свадьбы': lambda s : s.find('свадьба') != -1,
    'покупка автомобиля': lambda s : s.find('автомоб') != -1,
    'получение образования': lambda s : s.find('образов') != -1,
    'строительство': lambda s : s.find('строит') != -1,
    'ремонт': lambda s : s.find('ремонт') != -1,
    'покупка жилья': lambda s : s.find('жильё') != -1,
    'покупка коммерческой недвижимости': lambda s : s.find('недвиж') != -1 and s.find('
    'покупка прочей недвижимости': lambda s : s.find('недвиж') != -1,
    'прочая цель': True }
```

Зададим словарь кодов описаний классов для их последующего сохранения в поле `purpose_detail_id` набора данных:

```
In [93]: object_type_dict = dict(enumerate(object_criteria_dict.keys()))
object_type_dict
```

```
Out[93]: {0: 'проведение свадьбы',
1: 'покупка автомобиля',
2: 'получение образования',
3: 'строительство',
4: 'ремонт',
5: 'покупка жилья',
6: 'покупка коммерческой недвижимости',
7: 'покупка прочей недвижимости',
8: 'прочая цель'}
```

Опишем функцию категоризации в соответствии с критериями, определенными выше:

```
In [94]: def get_object_type_id(s):
    # возвращаем индекс первого выполненного условия из словаря критериев object_criter
    for i, criteria in enumerate(object_criteria_dict.items()):
        if criteria[1](s):
            return i
    return None
```

Выполним категоризацию по основной цели займа. Значение кодов класса занесем в поле `purpose_detail_id` набора данных:

```
In [95]: data['purpose_detail_id'] = data['purpose_norm'].apply(get_object_type_id)
```

```
In [96]: data[['purpose_norm', 'purpose_detail_id']]
```

```
Out[96]:
```

	purpose_norm	purpose_detail_id
0	покупка жильё	5
1	покупка автомобиль	1
2	покупка жильё	5
3	дополнительный образование	2
4	свадьба	0
...
21520	сделка жильё	5
21521	сделка автомобиль	1
21522	недвижимость	7
21523	покупка автомобиль	1
21524	покупка автомобиль	1

21341 rows × 2 columns

Категоризация по цели кредита выполнена. Набор данных теперь можно анализировать по категориям, используя группировку. Посмотрим количество обращений с различными целями:

```
In [97]: data.groupby(['for_contract', 'purpose_detail_id']).debt.count()
```

```
Out[97]:
```

for_contract	purpose_detail_id	
False	0	2309
	1	3345
	2	3993
	3	1871
	4	605
	5	3138
	6	658
	7	1244
True	1	938
	5	1295
	6	647
	7	1298

Name: debt, dtype: int64

Категоризация по статусу "многодетная семья"

В Москве **многодетной признается семья**, в которой родились и (или) воспитываются **трое и более детей** (в том числе усыновленные, а также пасынки и падчерицы) до достижения

младшим из них возраста 16 лет, а обучающимся в образовательной организации, реализующей основные общеобразовательные программы, — 18 лет.

Поместим значение `True` в новое поле `'big_family'` для многодетных семей:

```
In [98]: data['big_family'] = data['children'].apply(lambda n: None if n == 0 else n >= 3)

In [99]: print('Количество запросов от многодетных семей:', data.big_family.sum())
Количество запросов от многодетных семей: 425

In [100]: print('Доля запросов от многодетных семей: {:.2%}'.format(data.big_family.sum() / data.
Доля запросов от многодетных семей: 1.99%
```

Другие категории

При удалении дубликатов выше была фактически проведена категоризация по некоторым признакам:

- [уровень образования](#);
- [семейный статус](#);
- [тип занятости](#).

Проводить категоризацию по уровню дохода на данном этапе не имеет смысла. Это можно делать динамически с помощью функции `cut`. Воспользуемся ею при оценке зависимости возврата кредита от зарплаты ([см. ниже](#)).

Вывод

Осуществлена категоризация данных. В результате получены новые данные, которые упрощают обработку. Например, неформализованное поле цели получения кредита теперь пригодно для использования численными методами для решения задачи кредитного скоринга.

Шаг 3. Ответьте на вопросы

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

Изучим изменение доли просроченных кредитов в зависимости от количества детей у клиента.

```
In [101]: agg_columns = (('обратились', 'count'), ('не вернули в срок', 'sum'), ('доля просрочки',
bebt_by_children = data.groupby('children')['debt'].agg(agg_columns)
bebt_by_children
```

Out[101]:

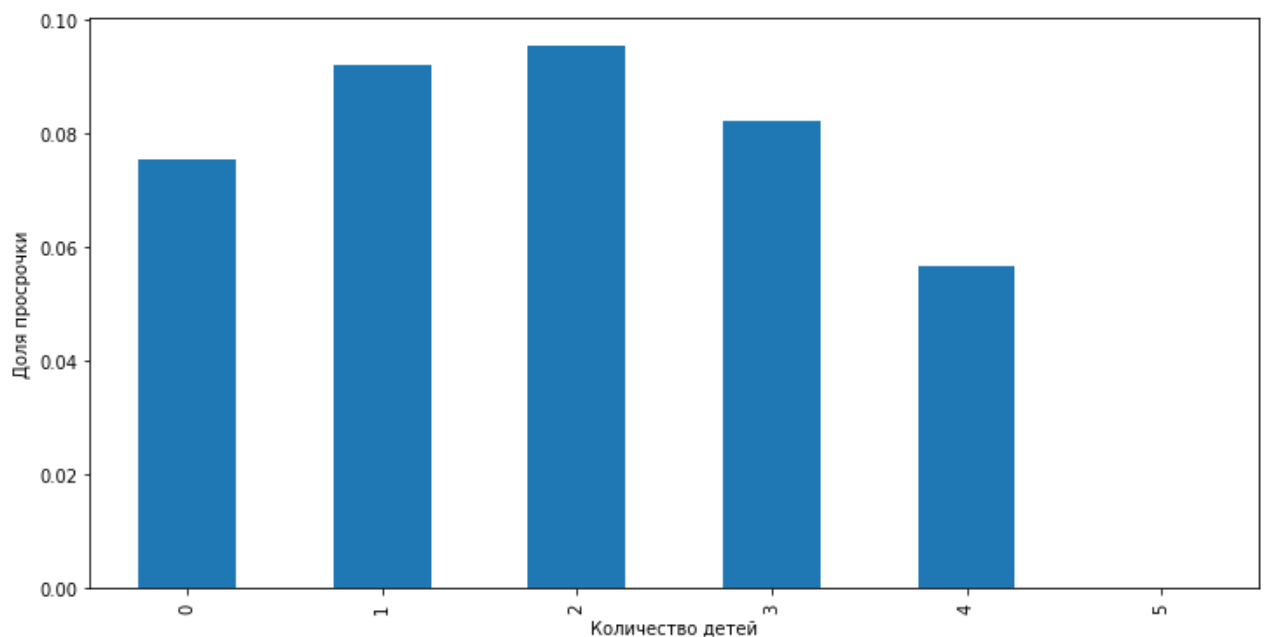
	обратились	не вернули в срок	доля просрочки
--	------------	-------------------	----------------

children			
0	14012	1058	0.075507
1	4790	441	0.092067

	обратились	не вернули в срок	доля просрочки
children			
2	2114	202	0.095553
3	328	27	0.082317
4	88	5	0.056818
5	9	0	0.000000

Наблюдается нелинейная зависимость возврата кредита в срок от количества детей у клиента. Больше всего страдают от коллекторов семьи с двумя детьми.

```
In [102...] ax = bebt_by_children['доля просрочки'].plot.bar(figsize = (12, 6))
ax.set_xlabel('Количество детей')
ax.set_ylabel('Доля просрочки');
```



Отсортируем клиентов по надежности в зависимости от количества детей в их семье:

```
In [103...] for n, i in enumerate(data.groupby('children')['debt'].mean().sort_values().index.tolist()):
    print('{:d}. Количество детей - {:d}'.format(n + 1, i))
```

1. Количество детей - 5
2. Количество детей - 4
3. Количество детей - 0
4. Количество детей - 3
5. Количество детей - 1
6. Количество детей - 2

Изучим изменение доли просроченных кредитов по категории "многодетная семья".

```
In [104...] agg_columns = (('обратились', 'count'), ('не вернули в срок', 'sum'), ('доля просрочки', 'mean'))
bebt_by_family_size = data.groupby('big_family')['debt'].agg(agg_columns)
bebt_by_family_size.index = ['1-2 ребенка', 'многодетные']
bebt_by_family_size
```

```
Out[104...]      обратился  не вернули в срок  доля просрочки
```

	обратились	не вернули в срок	доля просрочки
1-2 ребенка	6904	643	0.093134
многодетные	425	32	0.075294

Вывод

1. Наблюдается нелинейная зависимость возврата кредита в срок от количества детей у клиента. Больше всего страдают от коллекторов семьи с двумя детьми.
2. На надежность многодетных семей несомненно влияет их ответственность и государственная поддержка. Но результат не должен нас успокаивать. Семей с большим количеством детей гораздо меньше, чем с 1-2 детьми. Данные по многодетным чувствительны к каждому очередному просроченному платежу этой категории.

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

Изучим изменение доли просроченных кредитов в зависимости от семейного положения.

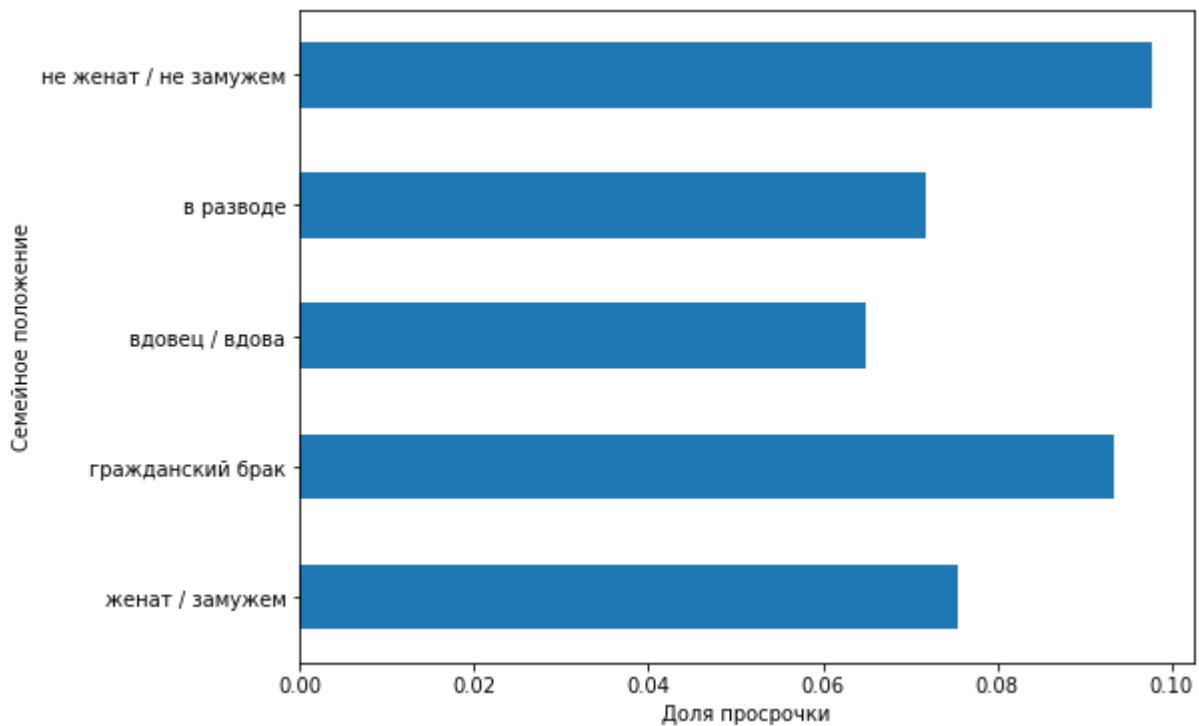
```
In [105... bebt_by_family_status = data.groupby('family_status_id')['debt'].agg(agg_columns)
bebt_by_family_status.index = family_status_to_id_dict.keys()
bebt_by_family_status
```

```
Out[105... 
```

	обратились	не вернули в срок	доля просрочки
женат / замужем	12280	927	0.075489
гражданский брак	4128	386	0.093508
вдовец / вдова	954	62	0.064990
в разводе	1185	85	0.071730
не женат / не замужем	2794	273	0.097709

Наблюдается зависимость возврата кредита в срок от семейного положения клиента. Чем меньше штампов в паспорте, тем ненадежнее клиент. Холостяки и наслаждающиеся так называемым "гражданским браком" не спешат выплачивать долги.

```
In [106... ax = bebt_by_family_status['доля просрочки'].plot.barh(figsize = (8, 6))
ax.set_ylabel('Семейное положение')
ax.set_xlabel('Доля просрочки');
```



Отсортируем клиентов по надежности в зависимости от их семейного положения:

```
In [107... for n, i in enumerate(data.groupby('family_status_id')['debt'].mean().sort_values().index):
    print('{:d}. {:s}'.format(n + 1, family_status_dict[i]))
```

1. вдовец / вдова
2. в разводе
3. женат / замужем
4. гражданский брак
5. не женат / не замужем

Вывод

Наблюдается зависимость возврата кредита в срок от семейного положения клиента. Чем меньше штампов в паспорте, тем ненадежнее клиент. Холостяки и наслаждающиеся так называемым "гражданским браком" не спешат выплачивать долги.

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

Разобьем клиентов на отдельные группы по уровню дохода с помощью функции `cut` и попытаемся найти зависимость. Метод разбиения зададим произвольно (на глаз):

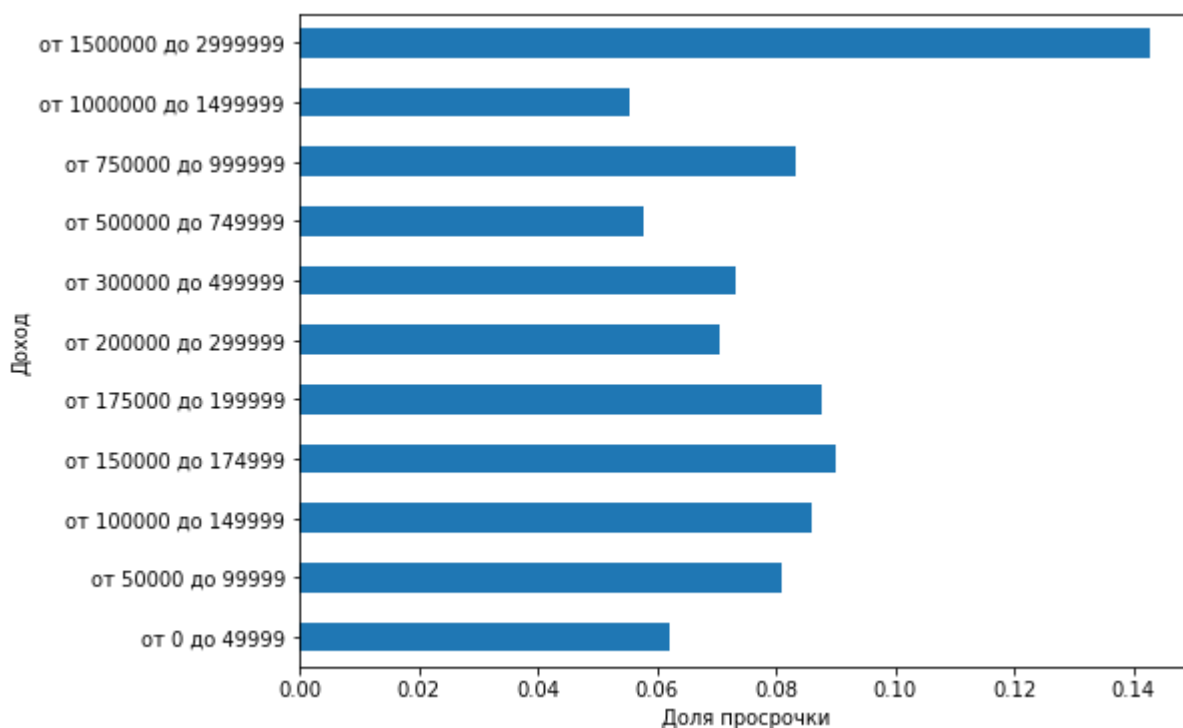
```
In [108... bins = [0, 50000, 100000, 150000, 175000, 200000, 300000, 500000, 750000, 1000000, 1500000]
labels = ['от {} до {}'.format(bins[i], bins[i+1]-1) for i in range(len(bins)-1)]
bebt_by_income = data.groupby(pd.cut(data.total_income, bins=bins, labels=labels))['debt'].mean()
bebt_by_income.index.name = 'доход'
bebt_by_income
```

```
Out[108...      обратился  не вернули в срок  доля просрочки
доход
от 0 до 49999      370             23      0.062162
```

	обратились	не вернули в срок	доля просрочки
доход			
от 50000 до 99999	4067	330	0.081141
от 100000 до 149999	5678	488	0.085946
от 150000 до 174999	2395	216	0.090188
от 175000 до 199999	1702	149	0.087544
от 200000 до 299999	3558	251	0.070545
от 300000 до 499999	1256	92	0.073248
от 500000 до 749999	173	10	0.057803
от 750000 до 999999	24	2	0.083333
от 1000000 до 1499999	18	1	0.055556
от 1500000 до 2999999	7	1	0.142857

Может показаться, что люди со высоким доходом чаще не возвращают кредиты в срок. Но богатых мало и их статистика может сильно измениться со временем.

```
In [109... ax = bebt_by_income['доля просрочки'].plot.barh(figsize = (8, 6))
ax.set_ylabel('Доход')
ax.set_xlabel('Доля просрочки');
```



Ошибки кредитного скоринга для многочисленного среднего класса может принести гораздо большие убытки банку. Давайте взглянем на статистику, разделив выборку на четыре примерно одинаковые по размеру группы с помощью функции `quantile` по квантилям:

```
In [110... p = [0, 0.25, 0.5, 0.75, 1.]
bins = data.total_income.quantile(q=p).astype(int).tolist()
```

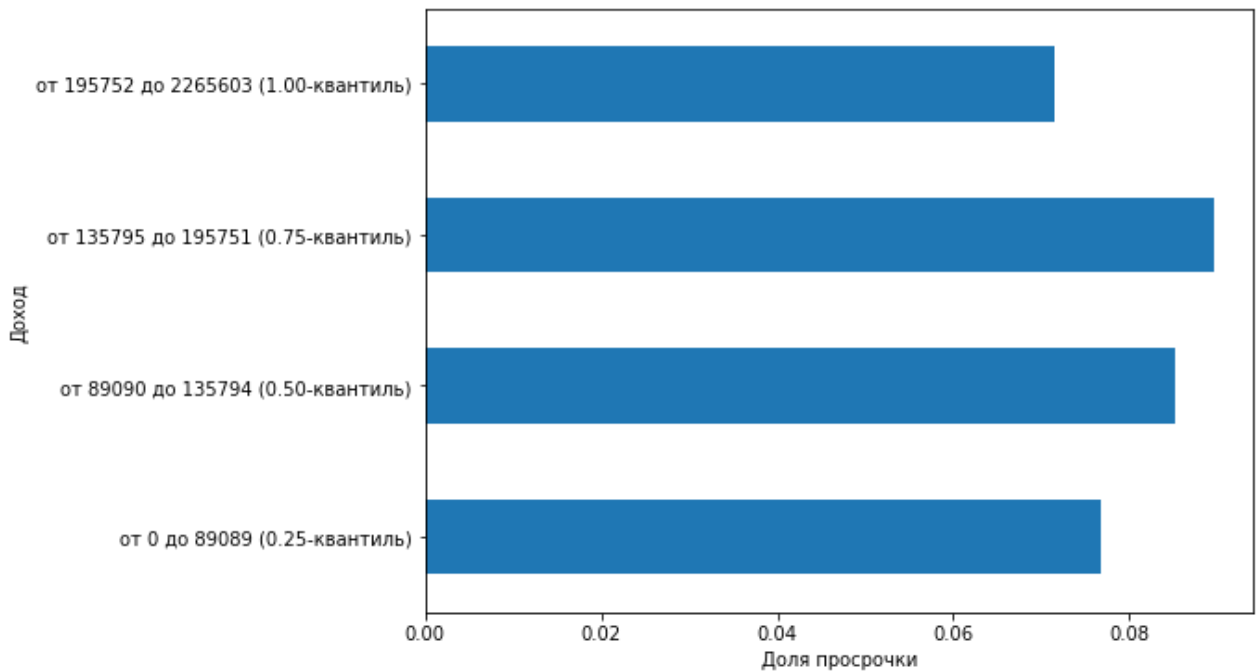
```
labels = ['от {} до {} ( {:.02f}-квантиль)'.format(bins[i], bins[i+1]-1, p[i+1]) for i in range(len(bins)-1)]
bebt_by_income = data.groupby(pd.cut(data.total_income, bins=bins, labels=labels))['debt']
bebt_by_income.index.name = 'доход'
bebt_by_income
```

Out[110]...

	обратились	не вернули в срок	доля просрочки
доход			
от 0 до 89089 (0.25-квантиль)	3242	249	0.076804
от 89090 до 135794 (0.50-квантиль)	5335	454	0.085098
от 135795 до 195751 (0.75-квантиль)	5335	478	0.089597
от 195752 до 2265603 (1.00-квантиль)	5335	382	0.071603

In [111]...

```
ax = bebt_by_income['доля просрочки'].plot.barh(figsize = (8, 6))
ax.set_ylabel('Доход')
ax.set_xlabel('Доля просрочки');
```



Вывод

Имеется зависимость между уровнем дохода и возвратом кредита в срок. Наибольшее количество просрочек допускают представители среднего класса.

Как разные цели кредита влияют на его возврат в срок?

Изучим изменение доли просроченных кредитов в зависимости от их основной цели (целевого объекта).

In [112]...

```
bebt_by_object = data.groupby(['purpose_detail_id'])['debt'].agg(agg_columns)
bebt_by_object.index = pd.MultiIndex.from_tuples(
    [(object_type_dict[i], ) for i in bebt_by_object.index.tolist()],
```

```
names=['целевой объект'])
bebt_by_object
```

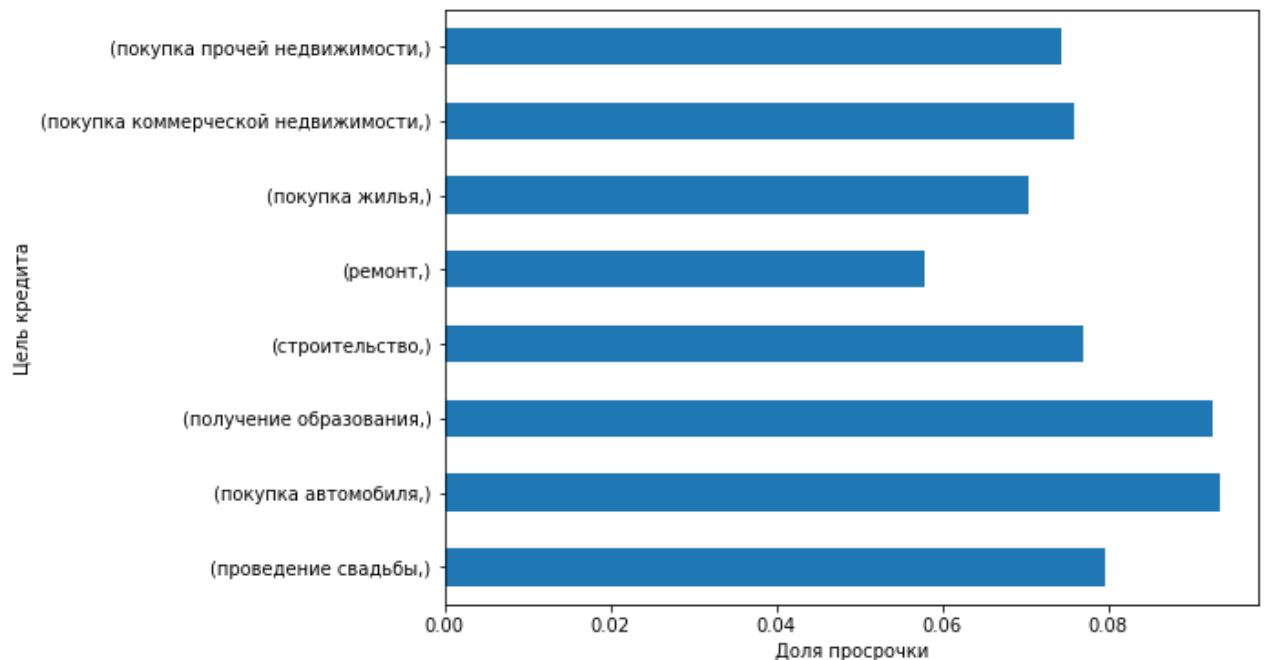
Out[112...

	обратились	не вернули в срок	доля просрочки
целевой объект			
проведение свадьбы	2309	184	0.079688
покупка автомобиля	4283	400	0.093392
получение образования	3993	370	0.092662
строительство	1871	144	0.076964
ремонт	605	35	0.057851
покупка жилья	4433	312	0.070381
покупка коммерческой недвижимости	1305	99	0.075862
покупка прочей недвижимости	2542	189	0.074351

Самыми рискованными для банка выглядят займы на обучение и покупку автомобиля, самыми возвращаемыми в срок - кредиты на ремонт:

In [113...

```
ax = bebt_by_object['доля просрочки'].plot.barh(figsize = (8, 6))
ax.set_ylabel('Цель кредита')
ax.set_xlabel('Доля просрочки');
```



Взглянем на проблему под другим углом. Проанализируем риски, если реализация цели кредита зависит ещё и от третьих лиц. Проанализируем сделки отдельно:

In [114...

```
bebt_by_purpose = data.groupby(['for_contract', 'purpose_detail_id'])['debt'].agg(agg_c
bebt_by_purpose.index = pd.MultiIndex.from_tuples(
    [('сделка' if i[0] else 'без сделки', object_type_dict[i[1]]) for i in bebt_by_purp
    names=['сделка', 'целевой объект'])
bebt_by_purpose
```

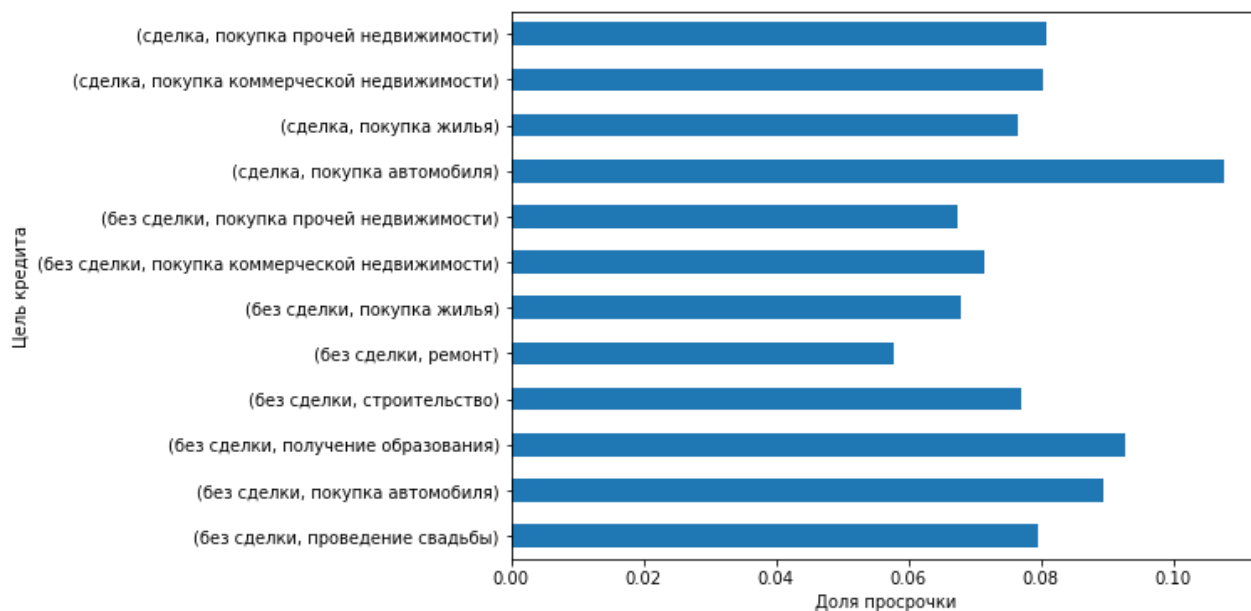
Out[114...

		обратились	не вернули в срок	доля просрочки
сделка	целевой объект			
без сделки	проведение свадьбы	2309	184	0.079688
	покупка автомобиля	3345	299	0.089387
	получение образования	3993	370	0.092662
	строительство	1871	144	0.076964
	ремонт	605	35	0.057851
	покупка жилья	3138	213	0.067878
	покупка коммерческой недвижимости	658	47	0.071429
сделка	покупка прочей недвижимости	1244	84	0.067524
	покупка автомобиля	938	101	0.107676
	покупка жилья	1295	99	0.076448
	покупка коммерческой недвижимости	647	52	0.080371
	покупка прочей недвижимости	1298	105	0.080894

Наблюдается изменение зависимости возврата кредита от цели, на которую его хочет потратить клиент, при рассмотрении сделок и обычных займов отдельно.

In [115...

```
ax = bebt_by_purpose['доля просрочки'].plot.barh(figsize = (8, 6))
ax.set_ylabel('Цель кредита')
ax.set_xlabel('Доля просрочки');
```



Самым рискованным оказывается кредитование сделок с автомобилями.

Вывод

1. Наблюдается зависимость возврата кредита от цели, на которую его хочет потратить клиент. При этом качество анализа рисков будет существенно зависеть от качества лемматизации текста с описанием цели займа, а также правильности выбора принципов категоризации.
2. Самыми рискованными для банка выглядят займы на обучение и покупку автомобиля, самыми возвращаемыми в срок - кредиты на ремонт.
3. Кредиты на прямую покупку автомобилей возвращаются в срок чаще, чем на сделки с автомобилями.

Шаг 4. Общий вывод

1. В данной работе на практике закреплены теоретические знания по предобработке данных и выполнены её типовые этапы на примере задачи кредитного скоринга.
2. Установлено, что возврат кредита зависит от многих факторов. Эту зависимость можно обнаружить с помощью простых инструментов обработки данных. Однако точно восстановить зависимость надежности клиента от различных параметров не представляется возможным без специальных численных методов. Их результативность будет во многом зависеть от качества предобработки данных, их лемматизации и категоризации.

Чек-лист готовности проекта

Поставьте 'x' в выполненных пунктах. Далее нажмите Shift+Enter.

- [x] [открыт файл](#);
- [x] [файл изучен](#);
- [x] [определены пропущенные значения](#);
- [x] [заполнены пропущенные значения](#);
- [x] [есть пояснение, какие пропущенные значения обнаружены](#);
- [x] [описаны возможные причины появления пропусков в данных](#);
- [x] [объяснено, по какому принципу заполнены пропуски](#);
- [x] [заменен вещественный тип данных на целочисленный](#);
- [x] [есть пояснение, какой метод используется для изменения типа данных и почему](#);
- [x] [удалены дубликаты](#);
- [x] [есть пояснение, какой метод используется для поиска и удаления дубликатов](#);
- [x] [описаны возможные причины появления дубликатов в данных \(причина 1, причина 2\)](#);
- [x] [выделены леммы](#) в значениях столбца с целями получения кредита;
- [x] [описан процесс лемматизации](#);
- [x] [данные категоризированы](#);
- [x] [есть объяснение принципа категоризации данных \(принцип 1, принцип 2, другие принципы\)](#);
- [x] [есть ответ на вопрос](#): "Есть ли зависимость между наличием детей и возвратом кредита в срок?"

- [x] есть **ответ** на **вопрос**: "Есть ли зависимость между семейным положением и возвратом кредита в срок?";
- [x] есть **ответ** на **вопрос**: "Есть ли зависимость между уровнем дохода и возвратом кредита в срок?";
- [x] есть **ответ** на **вопрос**: "Как разные цели кредита влияют на его возврат в срок?";
- [x] в каждом этапе есть выводы;
- [x] **есть общий вывод**.