Исследование надежности заемщиков.

Заказчик — кредитный отдел банка. Нужно разобраться, влияет ли семейное положение и количество детей клиента на факт погашения кредита в срок.
Нам предстоит:

- познакомиться с данными
- обработать пропуски и аномалии
- преобразовать типы данных
- сформировать дополнительные датафреймы для составления словарей
- обработать категориональные данные \

И ответить на поставленные вопросы:

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

Шаг 1. Обзор данных

Импортируем необходимые библиотеки и взглянем на данные.

In [2]:
```
import pandas as pd

url = 'https://code.s3.yandex.net/datasets/data.csv'
try:
    df = pd.read_csv('/Users/smaloletnev/Desktop/Yandex_Practicum/2_Предобработка_дан
except:
    df = pd.read_csv(url)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21525 entries, 0 to 21524
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   children          21525 non-null  int64  
 1   days_employed     19351 non-null  float64
 2   dob_years         21525 non-null  int64  
 3   education         21525 non-null  object 
 4   education_id      21525 non-null  int64  
 5   family_status     21525 non-null  object 
 6   family_status_id  21525 non-null  int64  
 7   gender            21525 non-null  object 
 8   income_type       21525 non-null  object 
 9   debt              21525 non-null  int64  
 10  total_income      19351 non-null  float64
 11  purpose           21525 non-null  object 
dtypes: float64(2), int64(5), object(5)
memory usage: 2.0+ MB
```

Выведем на экран первые 10 строк таблицы для ознакомления с содержимым.

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

Out[3]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gend
0	1	-8437.673028	42	высшее	0	женат / замужем	0	
1	1	-4024.803754	36	среднее	1	женат / замужем	0	
2	0	-5623.422610	33	Среднее	1	женат / замужем	0	
3	3	-4124.747207	32	среднее	1	женат / замужем	0	
4	0	340266.072047	53	среднее	1	гражданский брак	1	
5	0	-926.185831	27	высшее	0	гражданский брак	1	
6	0	-2879.202052	43	высшее	0	женат / замужем	0	
7	0	-152.779569	50	СРЕДНЕЕ	1	женат / замужем	0	
8	2	-6929.865299	35	ВЫСШЕЕ	0	гражданский брак	1	
9	0	-2188.756445	41	среднее	1	женат / замужем	0	

Описание данных

- `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

Шаг 2 Заполнение пропусков

Посчитаем кол-во пропусков во всех данных.

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
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
```

In [5]:
```
round(df.isnull().mean()*100, 2)
```

Out[5]:
```
children             0.0
days_employed       10.1
dob_years            0.0
education            0.0
education_id         0.0
family_status        0.0
family_status_id     0.0
gender               0.0
income_type          0.0
debt                 0.0
total_income        10.1
purpose              0.0
dtype: float64
```

В колонках `days_employed` и `total_income` одинаковое кол-во пропусков. Познакомимся с пропусками поближе. Посмотрим на строки с пропусками в колокне `total_income`

In [6]:
```
df[df.total_income.isnull()].head()
```

Out[6]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gen
12	0	NaN	65	среднее	1	гражданский брак	1	
26	0	NaN	41	среднее	1	женат / замужем	0	
29	0	NaN	63	среднее	1	Не женат / не замужем	4	
41	0	NaN	50	среднее	1	женат / замужем	0	
55	0	NaN	54	среднее	1	гражданский брак	1	

Можно заметить, что в строках где пропущено значение для `total_income`, также отсутсвует значение и `days_employed`. Проверим: для всех ли строк с пустым значением в `total_income` будут, значения в `days_employed` будут также отсутсвовать
Для этого составим условие: Значение в `income_type` отсутвует и значение в `days_employed` НЕ отсутсвует

In [7]:
```
df[(df.total_income.isnull()) & (df.days_employed.notnull())]
```

Out[7]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gende

Мое предположение подтвердилось: Если значение в `total_income` отсутсвует, то и в колонке `days_employed` значение тоже пропущено
Можно сделать вывод, что данные в данных колонках отсутсвуют, тк замещик не указал сведения о своем доходе и месте работы.
Определим долю пропусков. Поскольку кол-во пропусков для двух столбцов одинаково, то не имеет значения на какое именно кол-во пропусков следует делить общее кол-во строк

In [8]:
```
NoN= round(df.isnull().mean()*100, 2)
NoN
```

Out[8]:
```
children            0.0
days_employed      10.1
dob_years           0.0
education           0.0
education_id        0.0
family_status       0.0
family_status_id    0.0
gender              0.0
income_type         0.0
debt                0.0
total_income       10.1
purpose             0.0
dtype: float64
```

Почти 10% пропусков. Достаточно большое количество, чтобы принебречь этими данными в целом.
Заменим пропуски в колонке `total_income` на медианное значение. Именно медиана в данном случае подходит больше, тк данные имеют сильные отклонения.

In [9]:
```
df['total_income'] = df['total_income'].fillna(df['total_income'].median())
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21525 entries, 0 to 21524
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   children          21525 non-null  int64  
 1   days_employed     19351 non-null  float64
 2   dob_years         21525 non-null  int64  
 3   education         21525 non-null  object 
 4   education_id      21525 non-null  int64  
 5   family_status     21525 non-null  object 
 6   family_status_id  21525 non-null  int64  
 7   gender            21525 non-null  object 
 8   income_type       21525 non-null  object 
 9   debt              21525 non-null  int64  
 10  total_income      21525 non-null  float64
 11  purpose           21525 non-null  object 
dtypes: float64(2), int64(5), object(5)
memory usage: 2.0+ MB
```

Отлично. С пропусками в `total_income` разобрались. Теперь подробнее познакомимся с данными из столбца `days_employed`
Взгляним на медиану трудового стажа, сгруппированную по групам типов занятости

Шаг 3 Проверка данных на аномалии и исправления.

Теперь подробнее познакомимся с данными из столбца `days_employed`
Взгляним на медиану трудового стажа, сгруппированную по групам типов занятости

In [10]:
```
df.groupby('income_type').days_employed.median()
```

Out[10]:
```
income_type
безработный        366413.652744
в декрете           -3296.759962
госслужащий         -2689.368353
компаньон           -1547.382223
пенсионер          365213.306266
предприниматель      -520.848083
сотрудник           -1574.202821
студент              -578.751554
Name: days_employed, dtype: float64
```

У большинства групп отрицательные значение трудового стажа, что наталкивает на мысль о ошибке в предоставлении данных. Могу предположить, что данные неверно считались и следует принять за верные модуль данных значений.
Заменим отрицательные данные. Для этого применим метод `abs()` к колонке `days_employed`

In [11]:
```
df['days_employed'] = df['days_employed'].abs()
```

Осталось разобраться с данными пенсионеров и безработных.
Проверим: У всех ли пенсионеров и безработных "кривые" данные
Для этого предлагаю взгялнуть на статистические показатели по этим двум группам клиентов.

In [12]:
```
df[(df['income_type'] == 'пенсионер') | (df['income_type'] == 'безработный')].describ
```

Out[12]:

	children	days_employed	dob_years	education_id	family_status_id	debt	tota
count	3858.000000	3445.000000	3858.000000	3858.000000	3858.000000	3858.000000	385
mean	0.132193	365004.309916	59.052100	0.913686	0.985485	0.056247	13796
std	1.014106	21075.016396	7.633294	0.510267	1.314957	0.230427	7586
min	-1.000000	328728.720605	0.000000	0.000000	0.000000	0.000000	2066
25%	0.000000	346639.413916	56.000000	1.000000	0.000000	0.000000	8716
50%	0.000000	365213.306266	60.000000	1.000000	0.000000	0.000000	12874
75%	0.000000	383246.444219	64.000000	1.000000	2.000000	0.000000	16214
max	20.000000	401755.400475	74.000000	4.000000	4.000000	1.000000	7351(

Как мы видим, трудовой стаж данных клиентов от 328728.720605 до 401755.4004755. Не думаю, что кто-то мог наработать трудовой стаж более 1000 лет.
Если честно, я теряюсь в догадках причин таких данных. Посколько наша задача состоит в том, чтобы проверить влияние семейного положения и кол-ва детей на платежеспособность клиентов, думаю, не стоит сейчас угулбляться в анализ трудового стажа и искать корень данной аномалии.
Чтобы избавиться от пропусков, предлагаю заменить пропущенные значения медианой.

In [13]:
```
df['days_employed'] = df['days_employed'].fillna(df['days_employed'].median())
```

In [14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21525 entries, 0 to 21524
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   children          21525 non-null  int64  
 1   days_employed     21525 non-null  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      21525 non-null  float64
 11  purpose           21525 non-null  object 
dtypes: float64(2), int64(5), object(5)
memory usage: 2.0+ MB
```

Теперь предлагаю познакомиться с колонкой `children`, кажется я встречал там странные значения.

Для начала выведем количество уникальных значений

In [15]:
```
df['children'].value_counts()
```

Out[15]:
```
 0     14149
 1      4818
 2      2055
 3       330
 20       76
-1        47
 4        41
 5         9
Name: children, dtype: int64
```

Хм, -1 ребенок - это как? А 20 детей? Кажется, здесь закрылась очредная ошибка.

Поменяю данные значения на 1 и 2 соответсвенно

In [16]:
```
# Фунция для корректировки значений кол-ва детей
def childer_correct(children):
    if children == -1:
        return 1
    elif children == 20:
        return 2
    else:
        return children
```

In [17]:
```
# Применим функцию к дата фрейму
df['children'] = df['children'].apply(childer_correct)
# Проверка
df['children'].value_counts()
```

Out[17]:
```
0    14149
1     4865
2     2131
3      330
4       41
5        9
Name: children, dtype: int64
```

С количеством детей тоже разобрались.

Теперь познакомимся с колонкой "пол"

In [18]:
```
df['gender'].value_counts()
```

Out[18]:
```
F      14236
M       7288
XNA        1
Name: gender, dtype: int64
```

Есть неизвестный пол XNA. Такое значение всего одно на весь датафрейм, поэтому предлагаю избавиться от него.

In [19]:
```
df = df.drop(df[df['gender'] == 'XNA'].index)
df['gender'].value_counts()
```

Out[19]:
```
F    14236
M     7288
Name: gender, dtype: int64
```

Ура! Мы избавились от всех пропусков и разобрались с аномалиями!

Шаг 4 Изменение типов данных.

Данные в колонке `income_type` выглядят странно. Вряд ли доход клиента может быть нецелочисленным.
Изменим тип данных на `int`
Также заменим типы данных float64 на float32 и с int64 на int32 для экономии памяти

In [20]:
```
df['total_income'] = df['total_income'].astype('int32')
df['days_employed'] = df['days_employed'].astype('float32')
df['children'] = df['children'].astype('int32')
df['dob_years'] = df['dob_years'].astype('int32')
df['education_id'] = df['education_id'].astype('int32')
df['family_status_id'] = df['family_status_id'].astype('int32')
df['debt'] = df['debt'].astype('int32')
```

In [21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21524 entries, 0 to 21524
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   children          21524 non-null  int32
 1   days_employed     21524 non-null  float32
 2   dob_years         21524 non-null  int32
 3   education         21524 non-null  object
 4   education_id      21524 non-null  int32
 5   family_status     21524 non-null  object
 6   family_status_id  21524 non-null  int32
 7   gender            21524 non-null  object
 8   income_type       21524 non-null  object
 9   debt              21524 non-null  int32
 10  total_income      21524 non-null  int32
 11  purpose           21524 non-null  object
dtypes: float32(1), int32(6), object(5)
memory usage: 1.6+ MB
```

In [22]:
```
df.head()
```

Out[22]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gend
0	1	8437.672852	42	высшее	0	женат / замужем	0	
1	1	4024.803711	36	среднее	1	женат / замужем	0	
2	0	5623.422852	33	Среднее	1	женат / замужем	0	
3	3	4124.747070	32	среднее	1	женат / замужем	0	
4	0	340266.062500	53	среднее	1	гражданский брак	1	

Отлично. Теперь данные похожи на человеческие

Шаг 5 Удаление дубликатов.

Посмотрим на общее количество дубликатов в данных.

In [23]:
```
df.duplicated().sum()
```

Out[23]:
```
54
```

Дубликаты в данных есть. Но что конкретно это за дибликаты?
Необходимо помнить, что мы заменяли пропуски в некоторых столбцах на медианные значения. Соответсвенно, такие значения как минимум должны попасть в дубликаты.
Для проверки создадим датафрейм, из которого исключим медианные значения

In [24]:
```
# создание проверочного датафрейма и проверка на дубликаты
test_dublicates = df[(df['days_employed'] != 2194.220567) & (df['total_income'] != 14
test_dublicates.duplicated().sum()
```

Out[24]:
```
0
```

Отличные новости! Явные дубликаты были вызваны заменой пропусков на первых этапах работы.

Теперь поищем неявные дубликаты в конкретных столбцах.
Уделим внимание следующим колонкам:

- Образование
- Семейное положение
- Тип занятости

В них могут скрываться быть дубликаты

In [25]:
```
df['education'].value_counts()
```

Out[25]:
```
среднее                13750
высшее                  4718
СРЕДНЕЕ                  772
Среднее                  711
неоконченное высшее      667
ВЫСШЕЕ                   274
Высшее                   268
начальное                250
Неоконченное высшее       47
НЕОКОНЧЕННОЕ ВЫСШЕЕ       29
НАЧАЛЬНОЕ                 17
Начальное                 15
ученая степень             4
Ученая степень             1
УЧЕНАЯ СТЕПЕНЬ             1
Name: education, dtype: int64
```

In [26]:
```
df['family_status'].value_counts()
```

Out[26]:
```
женат / замужем          12380
гражданский брак          4176
Не женат / не замужем     2813
в разводе                 1195
вдовец / вдова             960
Name: family_status, dtype: int64
```

In [27]:
```
df['income_type'].value_counts()
```

Out[27]:
```
сотрудник          11119
компаньон           5084
пенсионер           3856
госслужащий         1459
безработный            2
предприниматель        2
студент                1
в декрете              1
Name: income_type, dtype: int64
```

Дубликаты встречаются в колнке с образованием. Встречаются они из-за разного регистра. Это достаточно частая причина появления дубликатов в данных. Приведем данные к единому регистру.

In [28]:
```
df['education'] = df['education'].str.lower()
```

In [29]:
```
df['education'].value_counts()
```

Out[29]:
```
среднее                15233
высшее                  5260
неоконченное высшее      743
начальное                282
ученая степень             6
Name: education, dtype: int64
```

Проверим: как изменилось количество дупликатов после обработки данных

In [30]:
```
df.duplicated().sum()
```

Out[30]:
```
71
```

Логично, что кол-во дубликатов увеличилось. Теперь можно от них избавиться.

In [31]:
```
df = df.drop_duplicates()
```

In [32]:
```
df.duplicated().sum()
```

Out[32]:
```
0
```

Отлично. Избавились от неявных дубликатов.
Заемщиков со средним образованием намного больше. Интересно, влияет ли уровень образование на потребность в кредитах? Или у нас в стране в целом людей со средним образованием гораздо больше?
Думаю, на этот вопрос мы ответим в следующий раз

Шаг 6 Формирование дополнительных датафреймов словарей, декомпозиция исходного датафрейма.

В наших данных есть колонки, которые хранят одинаковую информацию, но в разном виде. Например, колонка `education` хранит информацию об образовании клиента, а колонка `education_id` хранит идентификатор образования. Тоже самое можно сказать про колонки `family_status` и `family_status_id`.
В работе проще и удобнее обращаться к данным по идентификатору, тк это позволяет избежать ошибок даже при бональной фильтрации. Также это упростит визуальную работу с данными и уменьшит размер файла.
Исходя из этого, получается что информацию в этих колонках так сказать дублирует друг друга, но в разных форматах. Предлагаю избавиться от "лишней" информации. Прежде чем просто удалить колонки `education` и `family_status`, создадим "словари", в которых каждому идентификатору будет соответсвовать расшифровка.

In [33]:
```
# Создадим "словарь" в которому каждому идентификатору будет соответсвовать образован
df_education_dict = df[['education', 'education_id']].drop_duplicates().reset_index(d
df_education_dict
```

Out[33]:

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

In [34]:
```
# Создадим "словарь" в которому каждому идентификатору будет соответсвовать семейный
df_family_status_dict = df[['family_status', 'family_status_id']].drop_duplicates().r
df_family_status_dict
```

Out[34]:

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

Теперь можно смело удалить колонки education и family_status

In [35]:
```
df = df.drop(columns=['education', 'family_status'])
```

Шаг 7 Категоризация дохода.

Остановимся подробнее на данных о доходах. Что и логично, доход у кажого клиента разный и отличаются они достаточно сильно. Рабоать с такими данными и делать из них выводы нельзя.
Поэтому предлагаю категоризировать доход клиентов.

In [36]:
```
# Функция считывает доход клиента и присваивает категорию исходя из условий.
def income_category(income):
    if income <= 30000:
        return 'E'
    elif income <= 50000:
        return 'D'
    elif income <= 200000:
        return 'C'
    elif income <= 1000000:
        return 'B'
    elif income > 1000000:
        return 'A'
```

In [37]:
```
# Добавим колонку с категорией дохода в наш датафрейм и взглянем на обновленный датаф
df['total_income_category'] = df['total_income'].apply(income_category)
df.head()
```

Out[37]:

	children	days_employed	dob_years	education_id	family_status_id	gender	income_type	debt	to
0	1	8437.672852	42	0	0	F	сотрудник	0	
1	1	4024.803711	36	1	0	F	сотрудник	0	
2	0	5623.422852	33	1	0	M	сотрудник	0	
3	3	4124.747070	32	1	0	M	сотрудник	0	
4	0	340266.062500	53	1	1	F	пенсионер	0	

Шаг 8 Категоризация целей кредита.

Теперь сделаем категоризацию для целей кредита.
Взглянем на уникальные значения колонки purpose

In [38]:
```
sorted(df['purpose'].unique())
```

Out[38]:
```
['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']
```

Сразу бросается в глаза, что глобально можно разделить все эти цели на 4 группы:

- Операции с автомобилем
- операции с недвижимостью
- Свадьбы
- Образование

In [39]:
```python
# функция ищет "ключевые" слова с колонке и присваивает категорию
def purpose_category(purpose):
    if 'автомоб' in purpose:
        return 'Операции с автомобилем'
    elif ('жил' in purpose) or ('недвиж' in purpose):
        return 'Операции с недвижимостью'
    elif 'свад' in purpose:
        return 'Свадьба'
    elif 'образован' in purpose:
        return 'Образование'

#Применим функцию к датафрейму
df['purpose_category'] = df['purpose'].apply(purpose_category)

# Проверим получишиеся категории.
sorted(df['purpose_category'].unique())
```

Out[39]:
```
['Образование',
 'Операции с автомобилем',
 'Операции с недвижимостью',
 'Свадьба']
```

Но верно ли распределились категории?

Можно проверить используя сводную таблицу

In [40]:
```
purpose_pivot = df.pivot_table(index=['purpose_category' , 'purpose'], columns='debt'
purpose_pivot
```

Out[40]:

	debt	0	1
purpose_category	purpose		
Образование	высшее образование	412	40
	дополнительное образование	422	38
	заняться высшим образованием	453	43
	заняться образованием	369	39
	образование	415	32
	получение высшего образования	380	46
	получение дополнительного образования	395	51
	получение образования	405	37
	профильное образование	392	44
Операции с автомобилем	автомобили	434	44
	автомобиль	452	42
	на покупку автомобиля	427	44
	на покупку подержанного автомобиля	442	36
	на покупку своего автомобиля	459	46
	приобретение автомобиля	419	42
	свой автомобиль	430	48
	сделка с автомобилем	405	50
	сделка с подержанным автомобилем	435	51
Операции с недвижимостью	жилье	600	46
	недвижимость	591	42
	операции с жильем	604	48
	операции с коммерческой недвижимостью	598	52
	операции с недвижимостью	620	55
	операции со своей недвижимостью	577	50
	покупка жилой недвижимости	565	41
	покупка жилья	598	48
	покупка жилья для сдачи	599	52
	покупка жилья для семьи	593	45
	покупка коммерческой недвижимости	614	47
	покупка недвижимости	577	43
	покупка своего жилья	586	34
	ремонт жилью	572	35
	строительство жилой недвижимости	576	48
	строительство недвижимости	565	54
	строительство собственной недвижимости	593	42
Свадьба	на проведение свадьбы	704	64
	свадьба	727	64
	сыграть свадьбу	707	58

Промахов не обнаружено

Ответы на вопросы

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

Взглянем на количество клиентов в зависимости от количества детей

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

Out[41]:
```
0    14090
1     4855
2     2128
3      330
4       41
5        9
Name: children, dtype: int64
```

Тепеь посмотрим сколько "должников" в каждой группе

In [42]:
```
df.groupby('children')['debt'].sum()
```

Out[42]:
```
children
0    1063
1     445
2     202
3      27
4       4
5       0
Name: debt, dtype: int32
```

Интересные данные у нас получились. Можно заметить, что у людей без детей гораздо чаще встречаются задолженности по кредитам.
Взгляним на это в процентном соотношении.

In [43]:
```
round((df.groupby('children')['debt'].sum() / df['debt'].sum() * 100), 2)
```

Out[43]:
```
children
0    61.06
1    25.56
2    11.60
3     1.55
4     0.23
5     0.00
Name: debt, dtype: float64
```

Ничего себе! 61% клиентов с задолженностями приходится на заемщиков без детей. Далее процент снижается. Но все ли так просто?
Что можно сказать о данных, которые мы получили? С одной стороны, вывод очевиден: чем больше детей у клиента, тем он исправнее платит по кредиту.
Но что, если мы взгялнем на данные с другой стороны? Прекрасно видно, что людей без детей значительно больше. Например, клиентов без детей 14149, а клиентов с 5 детьми всего 9. Поэтому считаю, что делать выводы рано. Давайте посчитаем долю клиентов с

задолженностями от общего количества клиентов в каждой группе.
Для удобства построим сводную таблицу и добавим в нее колонку с долей

In [44]:
```
data_pivot_children = df.pivot_table(index='children', values='debt', aggfunc=['sum',
data_pivot_children.columns=['sum','count','mean']
data_pivot_children.sort_values('mean')
```

Out[44]:

children	sum	count	mean
5	0	9	0.000000
0	1063	14090	0.075444
3	27	330	0.081818
1	445	4855	0.091658
2	202	2128	0.094925
4	4	41	0.097561

Вот это уже похоже на результат! Как я и предполагал, выводы делать было рано. Теперь данные говорят нам о другом: Да, для клиентов с 5 детьми картина не поменялась, но напомню, что таких клиентов всего 9. По остальным же картина существенно поменялась. Клиенты без детей, оказывается, не погошают кредит незначительно, но все же реже, чем клиенты с детьми.

> Можно сделать общий вывод: Клиенты без детей не платят по кредитам реже, чем клиенты с 3 детьми. Клиенты с 1, 2 или 4 детьми не платят чаще остальных. Клиенты с 5 детьми всегда платят по счетам, но таких клиентов очень мало, чтобы сделать однозначный вывод. В целом же, доля клиентов с задолженностями в зависимости от кол-ва детей примерно одинакова - от 7.5 до 9.7 процентов уместилось 5 категорий. Поэтому можно сказать смело: кол-во детей клиентов НЕСИЛЬНО влияет на платежеспособность клиента.

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

In [45]:
```
# оставим перед глазами наш "словарик" по семейному положению
df_family_status_dict
```

Out[45]:

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

Сначала посмотрим количество клиентов для каждой группы по семейному положению

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

Out[46]:
```
0    12339
1     4150
4     2810
3     1195
2      959
Name: family_status_id, dtype: int64
```

Теперь посмотрим как клиенты платят по кредитам

In [47]:
```
df.groupby('family_status_id')['debt'].sum()
```

Out[47]:
```
family_status_id
0    931
1    388
2     63
3     85
4    274
Name: debt, dtype: int32
```

А что с долей клиентов, которые не плаят?

In [48]:
```
round((df.groupby('family_status_id')['debt'].sum() / df['debt'].sum() * 100), 2)
```

Out[48]:
```
family_status_id
0    53.48
1    22.29
2     3.62
3     4.88
4    15.74
Name: debt, dtype: float64
```

Опять 25. Вроде бы количество должников среди женатых клиентов больше всего, но и их количество сильно больше остальных. Посчитаем долю должников по категориям от кол-ва клиентов в этих категориях
Для удобства построим сводную таблицу и добавим в нее колонку с долей

In [49]:
```
data_pivot_family_status = df.pivot_table(index='family_status_id', values='debt', a
data_pivot_family_status.columns=['sum','count','mean']
data_pivot_family_status.sort_values('mean')
```

Out[49]:

family_status_id	sum	count	mean
2	63	959	0.065693
3	85	1195	0.071130
0	931	12339	0.075452
1	388	4150	0.093494
4	274	2810	0.097509

Так гораздо лучше. Теперь можно сформировать вывод.

> Как мы видим, семейное положение оказывает некотрое влияние на платежеспособность клиентов. Холостые клиенты не платят по кредитам чаще остальных. А вот овдовевшие клиенты показывают себя более надежными заемщиками.

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

Посмотрим на количество клиентов каждой группе по уровню дохода

In [50]:
```
df['total_income_category'].value_counts()
```

Out[50]:
```
C    16016
B     5040
D      350
A       25
E       22
Name: total_income_category, dtype: int64
```

Больше всего клиентов среднего класса, у которых доход от 50 до 200 тысяч рублей
Посмотрим как платят клиенты в зависимости от своего дохода.

In [51]:
```
df.groupby('total_income_category')['debt'].sum()
```

Out[51]:
```
total_income_category
A       2
B     356
C    1360
D      21
E       2
Name: debt, dtype: int32
```

Любопытно. Даже клиенты с доходом более 1млн рублей иногда не платят по кредитам.
Хотя ведь и сумма кредита может быть большой.
Посмотрим долю в каждой категории. Для удобства построим сводную таблицу и добавим в нее колонку с долей

In [52]:
```
data_pivot_income = df.pivot_table(index='total_income_category', values='debt', ag
data_pivot_income.columns=['sum','count','mean']
data_pivot_income.sort_values('mean')
```

Out[52]:

total_income_category	sum	count	mean
D	21	350	0.060000
B	356	5040	0.070635
A	2	25	0.080000
C	1360	16016	0.084915
E	2	22	0.090909

Как мы прекрасно видим, клиенты с доходом от 30 до 50 тысяч платят по кредитам охотнее остальных. Хуже всех платят клиенты с доходом менее 30 тыс рублей

Как разные цели кредита влияют на его возврат в срок?

Поскольку действия в целом будут сходи с предыдущими шагами, предлага сразу перейти к постраению сводной таблицы с долей, по которой можно будет сделать вывод

In [60]:
```
#data_pivot_category = df.pivot_table(index='purpose_category', values='debt', ag
##data_pivot_category.columns=['sum','count','mean']
###data_pivot_category.sort_values('mean')
```

Клиенты взявшие деньги на операции с недвижимостью, возвращают средства охотнее остальных. Самыми неблагонадежными являются клиенты, которые взяли деньги на операции с автомобилем

Общий вывод:

В данном проекте был проделан большой объем работы. Мы изучили данные, разобрались с пропусками и заполнили их необходимыми значениями. Проверили данные на аномалии и обработали их значения. Изменили типы данных для расчетов и экономии памяти в системе. Удалили дубликаты из датафрейма. Избавились от "лишних" колонок, предварительно составив себе словари с данными. Категоризировали клиентов по доходу и целям кредита. Провели анализ данных и готовы сформировать общий вывод по работе.
Мы получили достаточно много интересных результатов. Кому выдавать кредит, а кому нет - непростой вопрос.
Хоть и не всегда выводы казались очевидными, нам удалось докапаться до истины с помощью цифр. А цифра не врут никогда.
Одназночно можно сказать лишь то, что чем больше будет информации о заемщике, тем точнее можно будет построить вывод о его надежности. Ведь, как мы уже доказали выше, каждый фактор: будь то количесвто детей или сумма дохода, влияют на платежеспособность клиента.
Судя по нашим данным и проведенному анализу, идеальный клиент - овдовевший заемщик без детей, с доходом 30-50 тысяч рублей, который планирует операции с недвижимостью. А вот одинокие клиенты с доходом до 30 тысяч рублей, занимающие деньги на операцию с автомобилем самые ненадежные.