Лайфкодинг_1

Начнем с того, что импортируем библиотеки) Есть мнение, что библиотеки лучше прописывать непосредственно перед использованием. Это правило больше подходит разработчикам, поскольку у них все очень жестко с оптимизацией - зачем грузить систему лишней библитекой, если программа до этого места даже может не доработать) Мы - аналитики, нас ресурсы не поджимают, можно прописывать все библиотеки в одном месте - так будет порядок)

In [1]:

```
# импортируем библиотеку PANDAS
import pandas as pd
```

Полезная конструкция для тех, кто делает проект на локалке - два вида пути к файлам - яндексовский и пользовательский. Где не запускай - будет работать)

In [2]:

```
%time
try:
    df = pd.read_csv('/datasets/data.csv') #яндексовский путь
except:
    df = pd.read_csv('data.csv') # если же ошибка - мой путь
```

```
Wall time: 0 ns
```

А еще можно вот так, но нужно учитывать. что так дольше)

In [3]:

```
%time
df = pd.read_csv('https://code.s3.yandex.net/datasets/data.csv')
```

```
Wall time: 0 ns
```

In [4]:

```
display(df.sample(15)) #посмотрим 15 случайных значений таблицы - это позволяет сделать м
етод Sample
```

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type
2995	0	NaN	47	высшее	0	женат / замужем	0	F	сотрудник
9948	0	-2261.863018	54	среднее	1	женат / замужем	0	M	сотрудник
6653	0	-122.532417	35	среднее	1	женат / замужем	0	F	госслужащий
2377	1	-1994.027385	50	СРЕДНЕЕ	1	женат / замужем	0	M	сотрудник
2955	0	-4803.192024	44	среднее	1	женат / замужем	0	F	сотрудник
2363	3	-1376.931175	30	среднее	1	женат / замужем	0	M	сотрудник
3237	0	-640.655545	23	неоконченное высшее	2	женат / замужем	0	F	сотрудник
9141	0	356975.960450	66	среднее	1	женат /	0	F	пенсионер

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type
3108	**0**	**333439.469777**	**45**	среднее	**1**	женат / замужем	**0**	**F**	пенсионер
8902	**2**	**-2550.908535**	**40**	среднее	**1**	в разводе	**3**	**M**	сотрудник
19515	**0**	**-1120.684427**	**34**	среднее	**1**	гражданский брак	**1**	**F**	сотрудник
4244	**1**	**381465.314757**	**63**	начальное	**3**	Не женат / не замужем	**4**	**F**	пенсионер
14525	**0**	**340420.288244**	**57**	среднее	**1**	женат / замужем	**0**	**F**	пенсионер
3690	**1**	**NaN**	**34**	ВЫСШЕЕ	**0**	гражданский брак	**1**	**F**	сотрудник
14691	**1**	**337072.710671**	**50**	высшее	**0**	женат / замужем	**0**	**M**	пенсионер

In [5]:

```
df.info() # информация о таблице, это круто) поработаем над пропусками в доходе
```

```
<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
```

In [6]:

```
pd.DataFrame(round((df.isna().mean()*100),2)).style.background_gradient('coolwarm')
```

Out[6]:

	0
children	0.000000
days_employed	10.100000
dob_years	0.000000
education	0.000000
education_id	0.000000
family_status	0.000000
family_status_id	0.000000
gender	0.000000
income_type	0.000000
debt	0.000000
total_income	10.100000

purpose	0.000000

Есть много вариантов проверки условий в датасете, вот один из них, для примера. Мы говорим - отобрази нам те столбцы, где значение **total_income** больше **200000**

In [7]:

```
df[df['total_income']>200000]['total_income']
```

Out[7]:

```
0         253875.639453
3         267628.550329
5         255763.565419
6         240525.971920
16        289202.704229
              ...
21513     250986.142309
21514     355988.407188
21516     322807.776603
21520     224791.862382
21523     244093.050500
Name: total_income, Length: 5066, dtype: float64
```

Если мы зададим вопрос напрямую, вот как в этом случае - то получим не сами значения, а логический "вывод" - **True** или **False**, в зависимости от того, как удовлетворяется наше условаие, результат можно посмотреть в новом столбце)

In [8]:

```
df['total_income_temp'] = df['total_income']>200000
df.sample(5)
```

Out[8]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	del
13250	1	-4691.034683	40	высшее	0	гражданский брак	1	F	сотрудник	
12057	0	348232.556988	65	среднее	1	женат / замужем	0	F	пенсионер	
14620	0	-327.399826	24	среднее	1	Не женат / не замужем	4	F	сотрудник	
240	0	-4973.641485	47	среднее	1	вдовец / вдова	2	F	компаньон	
14909	1	-941.943913	25	среднее	1	женат / замужем	0	M	сотрудник	

Но самый лучший вариант обьявления условия - это использование **loc**. Итак в первой части этого оператора мы выставляем условие, а во второй части - столбец, по которому нужно работать) Вот пример, напрямую поставим **0** и столбец **total_income**, как бы говоря, давай посмотрим на строку с нулевым индексом в столбце с заработком)

In [9]:

```
df.loc[0,'total_income']
```

Out[9]:

```
253875.6394525987
```

Вот еще пример - покажи все строки, где заработок больше ста тысяч в столбце с заработком.

In [10]:

```
df.loc[df['total_income']>100000, 'total_income'].mean()
```

Out[10]:

```
194724.68308460235
```

Переходим к непосредственному применению с нашими данными. Итак, давай покажи нам строки, где есть пропуски по столбцу **total_income** (а мы помним метод **isna** нам показывает **True** или **False**, в зависимости от того, есть пропуск в столбце или нет)

In [11]:

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

Out[11]:

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

Создадим копию датафрейма для работы с дальнейшими данными

In [12]:

```
df_temp = df.copy()
```

Теперь просто присвоим всем значениям в столбце с доходом медиану дохода - определяем условие, что в столбце есть пропуск, и заменяем его на медиану по этому столбцу)

In [13]:

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

Out[13]:

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

Все сработало, но нас так не устраивает, нам лучше уточнить замену пропуков. Это можно сделать, заменив на медиану в зависимости от категории, самая лучшая - это вид заработка. Мы уже знаем как делать условия, потому теперь пропишем их два - отобрази нам те строки в которых есть пропуск и где тип заработка равен "сотрудник".

In [14]:

```
df.loc[(df['total_income'].isna())&(df['income_type'] =='сотрудник'), 'total_income'] =
\
df.loc[df['income_type'] =='сотрудник', 'total_income'].median()
```

И так можно сделать для каждого вида заработка) Но нас так не утраивает - если бы видов заработка было **50?** Организуем небольшую автоматизацию) Для того, чтоб автоматизировать получение уникальных значений в виде заработка, воспользуемся методом ниже)

In [15]:

```
df['income_type'].unique()
```

Out[15]:

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

Круто, теперь эти значения можно по одному проработать - используем для этого цикл. **type_unique** - это переменная(она может называться как угодно) в неё по очереди будет вкладываться вид заработка. Распечатаем каждый вид заработка по очереди)

In [16]:

```
for type_unique in df['income_type'].unique():
    display(type_unique)
```

```
'сотрудник'

'пенсионер'

'компаньон'

'госслужащий'

'безработный'

'предприниматель'

'студент'

'в декрете'
```

In [17]:

```
df_temp = df.copy()
```

Теперь мы можем вместо прописывания напрямую условия, прописать в условие переменную из цикла, где все будет прорабатываться по очереди. Идем с автоматизацией дальше. Весь наш код впишем в фукцию. В качестве переменных пропишем наш датасет, столбец с доходом и вид заработка. Результат очевиден)

In [18]:

```
'''
for type_unique in df_temp['income_type'].unique():
    df_temp.loc[(df_temp['total_income'].isna())&(df_temp['income_type'] ==type_unique),
'total_income'] = \
    df_temp.loc[df_temp['income_type'] ==type_unique, 'total_income'].median()

'''

def no_non_value(df, value, category):
    for type_unique in df[category].unique():
        df.loc[(df[value].isna())&(df[category] ==type_unique), value] = \
        df.loc[df_temp[category] ==type_unique, value].median()
    return df

df_temp = no_non_value(df_temp, 'total_income', 'income_type')
df_temp.head(15)
```

Out[18]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	de
0	**1**	**-8437.673028**	**42**	высшее	**0**	женат / замужем	**0**	**F**	сотрудник	
1	**1**	**-4024.803754**	**36**	среднее	**1**	женат / замужем	**0**	**F**	сотрудник	
2	**0**	**-5623.422610**	**33**	Среднее	**1**	женат / замужем	**0**	**M**	сотрудник	
3	**3**	**-4124.747207**	**32**	среднее	**1**	женат / замужем	**0**	**M**	сотрудник	

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	de
4	0	340266.072047	53	среднее	1	гражданский брак	1	F	пенсионер	
5	0	-926.185831	27	высшее	0	гражданский брак	1	M	компаньон	
6	0	-2879.202052	43	высшее	0	женат / замужем	0	F	компаньон	
7	0	-152.779569	50	СРЕДНЕЕ	1	женат / замужем	0	M	сотрудник	
8	2	-6929.865299	35	ВЫСШЕЕ	0	гражданский брак	1	F	сотрудник	
9	0	-2188.756445	41	среднее	1	женат / замужем	0	M	сотрудник	
10	2	-4171.483647	36	высшее	0	женат / замужем	0	M	компаньон	
11	0	-792.701887	40	среднее	1	женат / замужем	0	F	сотрудник	
12	0	NaN	65	среднее	1	гражданский брак	1	M	пенсионер	
13	0	-1846.641941	54	неоконченное высшее	2	женат / замужем	0	F	сотрудник	
14	0	-1844.956182	56	высшее	0	гражданский брак	1	F	компаньон	

Функции могут быть использованы к столбцу, для каждой строки отдельно - для этого используется **apply.**

In [19]:

```
def temp_1(value):
    return value*3 #функция умножает значение на 3
```

In [20]:

```
df_temp['dob_years'] =  df_temp['dob_years'].apply(temp_1)#применяем функцию для каждого значения в dob_years
df_temp.head(15)
```

Out[20]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	de
0	1	-8437.673028	126	высшее	0	женат / замужем	0	F	сотрудник	
1	1	-4024.803754	108	среднее	1	женат / замужем	0	F	сотрудник	
2	0	-5623.422610	99	Среднее	1	женат / замужем	0	M	сотрудник	
3	3	-4124.747207	96	среднее	1	женат / замужем	0	M	сотрудник	
4	0	340266.072047	159	среднее	1	гражданский брак	1	F	пенсионер	
5	0	-926.185831	81	высшее	0	гражданский брак	1	M	компаньон	
6	0	-2879.202052	129	высшее	0	женат / замужем	0	F	компаньон	
7	0	-152.779569	150	СРЕДНЕЕ	1	женат / замужем	0	M	сотрудник	
8	2	-6929.865299	105	ВЫСШЕЕ	0	гражданский брак	1	F	сотрудник	
9	0	-2188.756445	123	среднее	1	женат /	0	M	сотрудник	

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	de
10	2	-4171.483647	108	высшее	0	женат / замужем	0	M	компаньон	
11	0	-792.701887	120	среднее	1	женат / замужем	0	F	сотрудник	
12	0	NaN	195	среднее	1	гражданский брак	1	M	пенсионер	
13	0	-1846.641941	162	неоконченное высшее	2	женат / замужем	0	F	сотрудник	
14	0	-1844.956182	168	высшее	0	гражданский брак	1	F	компаньон	

Автоматизируем процесс категоризации для дохода. Для этого применим квантили) Поместим результат в словарь, где в ключах будет категория а в значениях - максимальный заработок в данной категории)

In [21]:

```
income_quantile = df_temp['total_income'].quantile([.2, .4, .6, 0.8, 0.99]).round().to_d
ict()
income_quantile
```

Out[21]:

```
{0.2: 98662.0, 0.4: 132142.0, 0.6: 161152.0, 0.8: 214270.0, 0.99: 505069.0}
```

Теперь можно организовать функцию. На входе будет значение значение каждого значения(тавталогия) в столбце с доходом) Далее мы будем брать по очереди значения из словоря, укладывая их в переменные **k, v**, где **k** - это ключ, он же "имя" нашей будущей категории, а **v** - это максимальное значение по этой категории) Итак говорим в коде следующее - возьми из словоря по очереди ключ и соответсвующее ему значение, и посмотри, если число в столбце меньше чем значение в словаре, дай имя категории по ключу) И прогоняем весь столбец методом **apply.**

In [22]:

```
def income_category(value):
    for k, v in income_quantile.items():
        if value < v:
            return k

df_temp['income_category'] = df_temp['total_income'].apply(income_category)
df_temp.head(15)
```

Out[22]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	de
0	1	-8437.673028	126	высшее	0	женат / замужем	0	F	сотрудник	
1	1	-4024.803754	108	среднее	1	женат / замужем	0	F	сотрудник	
2	0	-5623.422610	99	Среднее	1	женат / замужем	0	M	сотрудник	
3	3	-4124.747207	96	среднее	1	женат / замужем	0	M	сотрудник	
4	0	340266.072047	159	среднее	1	гражданский брак	1	F	пенсионер	
5	0	-926.185831	81	высшее	0	гражданский брак	1	M	компаньон	
6	0	-2879.202052	129	высшее	0	женат / замужем	0	F	компаньон	
7	0	-152.779569	150	СРЕДНЕЕ	1	женат /	0	M	сотрудник	

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	de
7	0	-152.779569	150	СРЕДНЕЕ	1	замужем	0	M	сотрудник	
8	2	-6929.865299	105	ВЫСШЕЕ	0	гражданский брак	1	F	сотрудник	
9	0	-2188.756445	123	среднее	1	женат / замужем	0	M	сотрудник	
10	2	-4171.483647	108	высшее	0	женат / замужем	0	M	компаньон	
11	0	-792.701887	120	среднее	1	женат / замужем	0	F	сотрудник	
12	0	NaN	195	среднее	1	гражданский брак	1	M	пенсионер	
13	0	-1846.641941	162	неоконченное высшее	2	женат / замужем	0	F	сотрудник	
14	0	-1844.956182	168	высшее	0	гражданский брак	1	F	компаньон	

Теперь, применяя все предыдущие методики мы можем проработать очень быстро и почти (да,да - почти, можно еще сильнее автоматизировать процесс) автоматически вопрос с лемматизацией)

```
from pymystem3 import Mystem # импортируем библиотеку для лемматизации
m = Mystem()

def lemmatize_purpose(string):# создаем функцию для лемматизации отдельной строки
    lemma = m.lemmatize(string)
    return lemma

df['purpose'] = df['purpose'].apply(lemmatize_purpose)

dest_str = ['жилье', 'автомобиль', 'образование', 'недвижимость', 'свадьба']

def filter_purpose(srt_lem):
    """
    создаем функцию, которая отсеивает ненужные слова, возвращая только основные по
нятия
    """
    for filtr in dest_str:        # перебераем с помощью переменной filtr каждое по
нятие
        if filtr in srt_lem:      # сравниваем со строкой из таблицы
            return filtr          # в случае успеха функция возвращает основное пон
ятие

df['purpose'] = df['purpose'].apply(filter_purpose)
```

Кратенькое отступление - чем **if** отличется от **elif)**

In [23]:

```
a=10
if a>=10: print('Yes')
if a<=10: print('No')
if a==10: print('50/50')
```

```
Yes
No
50/50
```

In [24]:

```
a=10
if a>=10: print('Yes')
elif a<=10: print('No')
elif a==10: print('50/50')
```

```
Yes
```

In [25]:

```
def temp_func(a):
    if a>=10: return('Yes')
    if a<=10: return('No')
    if a==10: return('50/50')

temp_func(10)
```

Out[25]:

```
'Yes'
```

Вот простой пример группировки)

In [26]:

```
df_temp.mean()
```

Out[26]:

```
children                  0.538908
days_employed         63046.497661
dob_years               129.880139
education_id              0.817236
family_status_id          0.972544
debt                      0.080883
total_income         165225.324514
total_income_temp         0.235354
income_category           0.594026
dtype: float64
```

In [27]:

```
df_temp['children'].nunique()
```

Out[27]:

```
8
```

In [28]:

```
for col in df_temp.columns:
    display(df_temp[col].value_counts())
```

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

-327.685916     1
-1580.622577    1
-4122.460569    1
-2828.237691    1
-2636.090517    1
               ..
-7120.517564    1
-2146.884040    1
-881.454684     1
-794.666350     1
-3382.113891    1
```

Name: days_employed, Length: 19351, dtype: int64

```
105    617
120    609
123    607
102    603
114    598
126    597
99     581
117    573
93     560
108    555
132    547
87     545
90     540
144    538
111    537
150    514
129    513
96     510
147    508
84     503
135    497
81     493
168    487
156    484
141    480
162    479
138    475
174    461
171    460
159    459
153    448
177    444
165    443
78     408
180    377
75     357
183    355
186    352
189    269
192    265
72     264
69     254
195    194
198    183
66     183
201    167
63     111
0      101
204     99
207     85
210     65
213     58
60      51
216     33
57      14
219      8
222      6
225      1
```

Name: dob_years, dtype: int64

```
среднее                13750
высшее                  4718
СРЕДНЕЕ                  772
Среднее                  711
неоконченное высшее      668
ВЫСШЕЕ                   274
Высшее                   268
начальное                250
Неоконченное высшее       47
НЕОКОНЧЕННОЕ ВЫСШЕЕ       29
```

```
НАЧАЛЬНОЕ                17
Начальное                15
ученая степень            4
Ученая степень            1
УЧЕНАЯ СТЕПЕНЬ            1
Name: education, dtype: int64
```

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

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

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

```
F      14236
M       7288
XNA        1
Name: gender, dtype: int64
```

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

```
0    19784
1     1741
Name: debt, dtype: int64
```

```
142594.396847    1105
172357.950966     509
118514.486412     414
150447.935283     147
499163.144947       2
                  ...
101516.604975       1
239154.168013       1
165009.733021       1
94270.049769        1
189255.286637       1
Name: total_income, Length: 19353, dtype: int64
```

```
свадьба                                   797
на проведение свадьбы                     777
сыграть свадьбу                           774
операции с недвижимостью                  676
покупка коммерческой недвижимости         664
операции с жильем                         653
покупка жилья для сдачи                   653
операции с коммерческой недвижимостью     651
жилье                                     647
покупка жилья                             647
покупка жилья для семьи                   641
строительство собственной недвижимости    635
недвижимость                              634
операции со своей недвижимостью           630
```

```
строительство жилой недвижимости          626
покупка недвижимости                      624
покупка своего жилья                      620
строительство недвижимости                620
ремонт жилью                              612
покупка жилой недвижимости                607
на покупку своего автомобиля              505
заняться высшим образованием              496
автомобиль                                495
сделка с подержанным автомобилем          489
свой автомобиль                           480
на покупку подержанного автомобиля        479
автомобили                                478
на покупку автомобиля                     472
приобретение автомобиля                   462
дополнительное образование                462
сделка с автомобилем                      455
высшее образование                        453
образование                               447
получение дополнительного образования     447
получение образования                     443
профильное образование                    436
получение высшего образования             426
заняться образованием                     412
Name: purpose, dtype: int64
```

```
False    16459
True      5066
Name: total_income_temp, dtype: int64
```

```
0.20    4305
0.40    4305
0.80    4305
0.60    4305
0.99    4089
Name: income_category, dtype: int64
```

Сгруппируем данные по семейному положению и посмотрим средний заработок. Если мы поместим исследуемый столбец в двойные скобки, то на выходе получим тип **dataframe**, но не совсем обычный. В таком датафрейме все равно навсегда создается связь значения с категорией, потому, чтоб получить только значение нужно воспользоваться методом **item()**. Кроме того, можно сразу привести в порядок индексы сгруппированной таблицы используя параметр **as_index=False**.

In [29]:

```
df_gr = df_temp.groupby('family_status', as_index=False)[['total_income']].mean()
df_gr
```

Out[29]:

	family_status	total_income
0	Не женат / не замужем	**166292.460611**
1	в разводе	**167703.137092**
2	вдовец / вдова	**142565.982593**
3	гражданский брак	**164633.093342**
4	женат / замужем	**166700.597371**

In [30]:

```
df_gr[df_gr['family_status'] == 'в разводе']['total_income'].item()
```

Out[30]:

```
167703.13709181309
```

Можно ли в группировках применить несколько функций? Можно) Для этого есть метод **agg**

In [31]:

```
df_temp.groupby('family_status')[['total_income']].agg(['sum', 'mean'])
```

Out[31]:

	total_income	
	sum	mean
family_status		
Не женат / не замужем	4.677807e+08	166292.460611
в разводе	2.004052e+08	167703.137092
вдовец / вдова	1.368633e+08	142565.982593
гражданский брак	6.876724e+08	164633.093342
женат / замужем	2.063753e+09	166700.597371

Можно ли группировать по нескольким столбцам**?** можно**)**

In [32]:

```
df_temp.groupby(['gender','family_status'])[['total_income']].agg(['min', 'mean','median'])
```

Out[32]:

		total_income		
		min	mean	median
gender	family_status			
F	Не женат / не замужем	24457.666662	159830.476477	142594.396847
	в разводе	33767.814447	161553.693101	142594.396847
	вдовец / вдова	34024.426612	142661.318403	126955.559504
	гражданский брак	25308.586849	154276.760147	141638.488651
	женат / замужем	20667.263793	151274.284909	138117.209047
M	Не женат / не замужем	27907.836304	176645.981907	156674.032264
	в разводе	41090.087506	189926.610356	170736.546528
	вдовец / вдова	52180.320019	140997.275171	133650.596801
	гражданский брак	21205.280566	187310.992073	161460.489426
	женат / замужем	21367.648356	192927.010816	166323.689177
XNA	гражданский брак	203905.157261	203905.157261	203905.157261

Можно ли обрабатывать разные значения в группировках из разных столбцов**?** Можно**)**

In [33]:

```
df_temp.groupby(['gender','family_status']).agg({'children':'mean', 'total_income':['mean','median']})
```

Out[33]:

		children	total_income	
		mean	mean	median
gender	family_status			
F	Не женат / не замужем	0.313510	159830.476477	142594.396847
	в разводе	0.418803	161553.693101	142594.396847

	вдовец / вдова	0.236364	140997.275171	133650.596801
	гражданский брак	0.510703	187310.992073	161460.489426
	женат / замужем	0.708615	192927.010816	166323.689177
XNA	гражданский брак	0.000000	203905.157261	203905.157261

При этом мы можем сделать ограничения, при помощи **query**, например, отсеч аномалии) Создадим группировку, где ответим на вопрос, есть ли зависимость долгов от количества детей) Посчитаем всего заемщиков в категории, сумму "единичек" т.е. сумму должников в каждой категории, и долю должников, которая считается как и среднее)

In [34]:

```
df_temp.query('children !=-1 & children!=20').groupby('children')[['debt']].agg(['count'
,'sum','mean'])
```

Out[34]:

	debt		
	count	sum	mean
children			
0	14149	1063	0.075129
1	4818	444	0.092154
2	2055	194	0.094404
3	330	27	0.081818
4	41	4	0.097561
5	9	0	0.000000

А теперь сделаем то же самое при помощи сводных таблиц. В индексы таблицы положим "детей", в значения - долги, в функции списком положим необходимые функции. Обратите внимание, что параметр **columns** мы пока пропустили, так можно)

In [35]:

```
df_temp.query('children !=-1 & children!=20').pivot_table(index = 'children',
                                                          values = 'debt', aggfunc = ['
count','sum','mean'] )
```

Out[35]:

	count	sum	mean
	debt	debt	debt
children			
0	14149	1063	0.075129
1	4818	444	0.092154
2	2055	194	0.094404
3	330	27	0.081818
4	41	4	0.097561
5	9	0	0.000000

Вот теперь сделаем то, что не умеет группоровка) Добавим столбцы с полом, чтоб посмотреть зависимость от детей и пола)

In [36]:

```
df_temp.query('children !=-1 & children!=20 & gender!="XNA"').pivot_table(index = 'children', columns = 'gender',
                                                                          values = 'debt', aggfunc = 'mean')
```

Out[36]:

gender	F	M
children		
0	0.061860	0.102883
1	0.079186	0.115429
2	0.106434	0.075377
3	0.086735	0.074627
4	0.035714	0.230769
5	0.000000	0.000000

БОНУС

Коротенько про лямбды. Основная информация - тут https://faist.ru/post/2018/08/02/lambda-python

Помните функцию с умножение **dob_years** на **3.** Сделаем то же с этим столбцом, но разделим на **3 (**Двойное деление - деление без остатки**)**

In [37]:

```
df_temp['dob_years'] =  df_temp['dob_years'].apply(lambda x: x//3)
df_temp.head(5)
```

Out[37]:

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

А вот так лямбды можно применять с условием)

In [38]:

```
df_temp['days_employed_temp'] =  df_temp['days_employed'].apply(lambda x: x if x>0 else x*-1)
df_temp.head(5)
```

Out[38]:

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

А вот так можно сформировать красивый вывод в процентах для сводной таблицы, используя лямба-функции и форматирование)

In [39]:

```
df_temp.query('children !=-1 & children!=20 & gender!="XNA"').pivot_table(index = 'child
ren', columns = 'gender',
                                                          values = 'debt', aggfunc = la
mbda x: x.mean())
```

Out[39]:

gender	F	M
children		
0	0.061860	0.102883
1	0.079186	0.115429
2	0.106434	0.075377
3	0.086735	0.074627
4	0.035714	0.230769
5	0.000000	0.000000

In [40]:

```
df_temp.query('children !=-1 and children!=20 and gender!="XNA"').pivot_table(index = 'ch
ildren', columns = 'gender',
                                                          values = 'debt',
                                                        aggfunc = lambda x: '{:.2%}'.fo
rmat(x.mean()))
```

Out[40]:

gender	F	M
children		
0	6.19%	10.29%
1	7.92%	11.54%
2	10.64%	7.54%
3	8.67%	7.46%
4	3.57%	23.08%
5	0.00%	0.00%

Тепер посмотрим на очень интересный момент. Как я уже говорил при группировках получается датафрейм но со связями с категорией. Взгляните на вывод ниже. Вот такая конструкция

```
df_temp.groupby('income_type')['total_income']
```

практически ничем не отличется от

```
df_temp['total_income']
```

В этом случае сохраняется все - и индексы и значения, только данные сгруппированы - созданы связи) И этим можно воспользоваться)

In [41]:

```
df_temp.groupby('income_type')['total_income'].agg(display)
```

```
3133      59956.991984
14798    202722.511368
Name: total_income, dtype: float64
```

```
20845    53829.130729
Name: total_income, dtype: float64
```

```
26       150447.935283
41       150447.935283
47       356277.909345
62       435388.195272
70       207561.466998
             ...
21288    233316.781101
21349     81624.022602
21380    103028.288433
21459    150568.114869
21507     98180.279152
Name: total_income, Length: 1459, dtype: float64
```

```
5        255763.565419
6        240525.971920
10       113943.491460
14       165127.911772
33       157245.786233
             ...
21512    147301.457769
21514    355988.407188
21516    322807.776603
21517    178059.553491
21520    224791.862382
Name: total_income, Length: 5085, dtype: float64
```

```
4        158616.077870
12       118514.486412
18        56823.777243
24       290547.235997
25        55112.757732
             ...
21505     75439.993167
21508     72638.590915
21509     73029.059379
21518    153864.650328
21521    155999.806512
Name: total_income, Length: 3856, dtype: float64
```

```
5936     499163.144947
18697    499163.144947
Name: total_income, dtype: float64
```

```
0        253875.639453
1        112080.014102
2        145885.952297
3        267628.550329
7        135823.934197
             ...
21515    109486.327999
21519    115949.039788
21522     89672.561153
```

```
21523    244093.050500
21524     82047.418899
Name: total_income, Length: 11119, dtype: float64

9410    98201.625314
Name: total_income, dtype: float64
```

Out[41]:

```
income_type
безработный        None
в декрете          None
госслужащий        None
компаньон          None
пенсионер          None
предприниматель    None
сотрудник          None
студент            None
Name: total_income, dtype: object
```

In [42]:

```
df_temp['total_income']
```

Out[42]:

```
0        253875.639453
1        112080.014102
2        145885.952297
3        267628.550329
4        158616.077870
             ...
21520    224791.862382
21521    155999.806512
21522     89672.561153
21523    244093.050500
21524     82047.418899
Name: total_income, Length: 21525, dtype: float64
```

Итак, давайте снова проработаем пропуски в **total_income.** Теперь мы можем пробегаться по каждой категории в сгруппированной таблице и менять на медиану пропуски только в той категории, к которой выявлены связи)

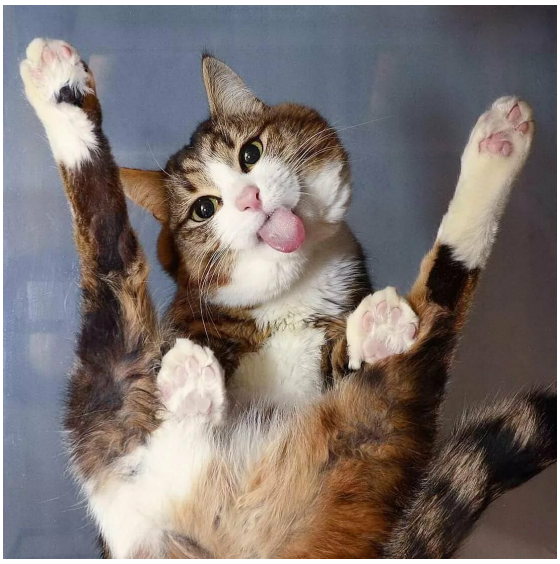
In [43]:

```
df_temp['total_income'] = df_temp.groupby('income_type')['total_income'].apply(lambda x:
x.fillna(x.median()))
df_temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21525 entries, 0 to 21524
Data columns (total 15 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
 12  total_income_temp   21525 non-null  bool
 13  income_category     21309 non-null  float64
 14  days_employed_temp  19351 non-null  float64
dtypes: bool(1), float64(4), int64(5), object(5)
memory usage: 2.3+ MB
```

ВСЕМ СПАСИБО)

ВСЕМ СПАСИБО)



In [ ]: