Analyzing borrowers' risk of defaulting

Your project is to prepare a report for a bank's loan division. You'll need to find out if a customer's marital status and number of children has an impact on whether they will default on a loan. The bank already has some data on customers' credit worthiness.

Your report will be considered when building a **credit scoring** of a potential customer. A **credit scoring** is used to evaluate the ability of a potential borrower to repay their loan.

Step 1. Open the data file and have a look at the general information.

In [299]:

```
import pandas as pd
credit_scoring = pd.read_csv('/datasets/credit_scoring_eng.csv')

credit_scoring.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21525 entries, 0 to 21524
Data columns (total 12 columns):
children            21525 non-null int64
days_employed       19351 non-null float64
dob_years           21525 non-null int64
education           21525 non-null object
education_id        21525 non-null int64
family_status       21525 non-null object
family_status_id    21525 non-null int64
gender              21525 non-null object
income_type         21525 non-null object
debt                21525 non-null int64
total_income        19351 non-null float64
purpose             21525 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 2.0+ MB
```

In [300]:

```
credit_scoring.head()
```

Out[300]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income	p
0	1	-8437.673028	42	bachelor's degree	0	married	0	F	employee	0	40620.102	pur th
1	1	-4024.803754	36	secondary education	1	married	0	F	employee	0	17932.802	car p
2	0	-5623.422610	33	Secondary Education	1	married	0	M	employee	0	23341.752	pur th
3	3	-4124.747207	32	secondary education	1	married	0	M	employee	0	42820.568	supple e
4	0	340266.072047	53	secondary education	1	civil partnership	1	F	retiree	0	25378.572	t

In [301]:

```
credit_scoring.describe()
```

Out[301]:

	children	days_employed	dob_years	education_id	family_status_id	debt	total_income
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	26787.568355
std	1.381587	140827.311974	12.574584	0.548138	1.420324	0.272661	16475.450632
min	-1.000000	-18388.949901	0.000000	0.000000	0.000000	0.000000	3306.762000
25%	0.000000	-2747.423625	33.000000	1.000000	0.000000	0.000000	16488.504500
50%	0.000000	-1203.369529	42.000000	1.000000	0.000000	0.000000	23202.870000
75%	1.000000	-291.095954	53.000000	1.000000	1.000000	0.000000	32549.611000
max	20.000000	401755.400475	75.000000	4.000000	4.000000	1.000000	362496.645000

Conclusion

After first look at the data, we can already see that the data is not perfect:

1. Not all the data is in the right format;
2. Some data has fewer rows than other;
3. There are strange values in some columns:
   - Some suspicious amount of children (20);
   - Really strange values in amount of days employed.

So there is a some amount of data prepocessing needed to be made.

Step 2. Data preprocessing

Processing missing values

Firstly we need to find out what is going on with columns that have less values than others.

In [302]:
```
#credit_scoring['days_employed'].value_counts().sort_values()

unemployed = credit_scoring[credit_scoring['days_employed'].isnull()]
unemployed.head()
```

Out[302]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income	pur
12	0	NaN	65	secondary education	1	civil partnership	1	M	retiree	0	NaN	to ha wec
26	0	NaN	41	secondary education	1	married	0	M	civil servant	0	NaN	educ
29	0	NaN	63	secondary education	1	unmarried	4	F	retiree	0	NaN	build e
41	0	NaN	50	secondary education	1	married	0	F	civil servant	0	NaN	sec han purc
55	0	NaN	54	secondary education	1	civil partnership	1	F	retiree	1	NaN	to ha wec

Here we can see a surtain correlation between unemployed people and people with 0 income. Lets check that these are the same people.

In [303]:
```
#Find out if all missing values in unemployed column mean that these people are unemployed
unemployed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2174 entries, 12 to 21510
Data columns (total 12 columns):
children            2174 non-null int64
days_employed       0 non-null float64
dob_years           2174 non-null int64
education           2174 non-null object
education_id        2174 non-null int64
family_status       2174 non-null object
family_status_id    2174 non-null int64
gender              2174 non-null object
income_type         2174 non-null object
debt                2174 non-null int64
total_income        0 non-null float64
purpose             2174 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 220.8+ KB
```

In [304]:
```
#Now it's time to fill this missing data with some values.
#Because we know that this empty data means that they are unemployed and have no income we will fill this data with 0
credit_scoring['days_employed'] = credit_scoring['days_employed'].fillna('0')
credit_scoring['total_income'] = credit_scoring['total_income'].fillna('0')
```

So now we have changed the data in that columns to good zeroes that suit us. Let's move on and check if there is anything wrong in other columns. We will check them applying 'value_columns' for each column.

```
In [305]: credit_scoring['gender'].value_counts()
```

```
Out[305]: F      14236
          M       7288
          XNA        1
          Name: gender, dtype: int64
```

Here we can see that gender column has some strange data that doesn't fit there. let's have a look at this row.

Who knows, may be that person is still not certain yet. 😃

```
In [306]: credit_scoring[credit_scoring['gender'] == 'XNA']
```

Out[306]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income	p
10701	0	-2358.6	24	some college	2	civil partnership	1	XNA	business	0	32624.8	I

That is a woman, men don't use 'civil partnership' as a family_status, what do you think?

In other columns this row looks alright, so let's keep it for a while and see if we'll need to do anything with it later.

```
In [307]: credit_scoring['dob_years'].describe()
```

```
Out[307]: count    21525.000000
          mean        43.293380
          std         12.574584
          min          0.000000
          25%         33.000000
          50%         42.000000
          75%         53.000000
          max         75.000000
          Name: dob_years, dtype: float64
```

Here I also see that data is corrupted with some suspicious '0' in age column. Even though '0' were suitable for us in other columns, here it's completely unacceptable. To process it let''s firstly have a look at what's going on in these rows.

```
In [308]: credit_scoring[credit_scoring['dob_years'] == 0].head()
```

Out[308]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income	
99	0	346542	0	Secondary Education	1	married	0	F	retiree	0	11406.6	
149	0	-2664.27	0	secondary education	1	divorced	3	F	employee	0	11228.2	tra
270	3	-1872.66	0	secondary education	1	married	0	F	employee	0	16346.6	r
578	0	397857	0	secondary education	1	married	0	F	retiree	0	15619.3	co
1040	0	-1158.03	0	bachelor's degree	0	divorced	3	F	business	0	48639.1	

Looks like a proper data with valueable information in it. So from here we have 2 ways of action: delete these rows or replace the age in these rows with average value. If we were investegating correlation between income and repaing loan on time, than maybe it would have been better for us to drop this rows. But for us clients' age is not essential so I'm going to replace it with average age value.

```
In [309]: age_mean = credit_scoring['dob_years'].mean()
          credit_scoring.loc[credit_scoring['dob_years'] == 0, 'dob_years'] = age_mean.round(0).astype(int)
          #changing missing age values to average age for the whole data
```

After that I've decided to analize what is going on with amount of children.

In [310]:
```
credit_scoring.children.value_counts()
```

Out[310]:
```
 0     14149
 1      4818
 2      2055
 3       330
 20       76
-1        47
 4        41
 5         9
Name: children, dtype: int64
```

We have two strange type of values here: 20 children and -1 child. Let's look at them separatly.

In [311]:
```
#check family status of people with 20 children
credit_scoring[credit_scoring['children'] == 20].family_status.value_counts()
```

Out[311]:
```
married              49
civil partnership    12
unmarried             9
widow / widower       4
divorced              2
Name: family_status, dtype: int64
```

In [312]:
```
#check values of family status column for the whole table
credit_scoring.family_status.value_counts()
```

Out[312]:
```
married              12380
civil partnership     4177
unmarried             2813
divorced              1195
widow / widower        960
Name: family_status, dtype: int64
```

I didn't see any sort of connection here. Then I tried to check connection with other columns.

In [313]:
```
#connection to gender
credit_scoring[credit_scoring['children'] == 20].gender.value_counts()
```

Out[313]:
```
F    47
M    29
Name: gender, dtype: int64
```

In [314]:
```
#connection to debt
credit_scoring[credit_scoring['children'] == 20].debt.value_counts()
```

Out[314]:
```
0    68
1     8
Name: debt, dtype: int64
```

In [315]:
```
#connection to purpose
credit_scoring[credit_scoring['children'] == 20].income_type.value_counts()
```

Out[315]:
```
employee         43
business         22
retiree           9
civil servant     2
Name: income_type, dtype: int64
```

I didn't find out any specific reason for this mistake to be in there. Maybe it was some programming error, but maybe it was human error. I decided not to drop these rows just yet, because they could be useful in other analysis, but when I get to analysis of how number of children affect default on a loan I will drop that rows.

After that I checked the rows that had -1 child.

In [316]:
```
credit_scoring[credit_scoring['children'] == -1].head(5)
```

Out[316]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income	
291	-1	-4417.7	46	secondary education	1	civil partnership	1	F	employee	0	16450.6	
705	-1	-902.085	50	secondary education	1	married	0	F	civil servant	0	22061.3	ca
742	-1	-3174.46	57	secondary education	1	married	0	F	employee	0	10282.9	sup
800	-1	349988	54	secondary education	1	unmarried	4	F	retiree	0	13807	sup
941	-1	0	57	Secondary Education	1	married	0	F	retiree	0	0	

In [317]:
```
credit_scoring[credit_scoring['children'] == -1].family_status.value_counts()
```

Out[317]:
```
married              29
unmarried             5
civil partnership     5
divorced              4
widow / widower       4
Name: family_status, dtype: int64
```

Seems to me that here a human mistake took place, when some bank worker has put in -1 instead of 1. Therefore I will change these values to 1.

It might be just 2, but it also might be something else:) I'm not sure that here we have enough information to make these conclusion.

In [318]:
```
credit_scoring.loc[credit_scoring['children'] == -1, 'children'] = 1

credit_scoring.children.value_counts()
```

Out[318]:
```
0     14149
1      4865
2      2055
3       330
20       76
4        41
5         9
Name: children, dtype: int64
```

Conclusion

I have found out two types of missing values.

1. The data had some NaN values in columns days_employed and total_income. These were connected to unemployed clients, so we couldn't delete them or change them, we need to keep them, so I have changed these values to '0', so thy would be more suitable for us. These missing values came to our data because of specifics of writing this data down: it has processed no income as an empty data, and it wasn't useful for us.
2. In the age column I have found out that some custumers are having 0 age. This data is completely wrong and needs to be changed. For reasons described in that section I have decided to change the values in these rows to average age. It won't affect our analysis, but it's crutial to remember that we've done it. Most likely this column was corrupted with zeroes due to human errors.

Data type replacement

In [319]:

```
credit_scoring.info()
credit_scoring.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21525 entries, 0 to 21524
Data columns (total 12 columns):
children            21525 non-null int64
days_employed       21525 non-null object
dob_years           21525 non-null int64
education           21525 non-null object
education_id        21525 non-null int64
family_status       21525 non-null object
family_status_id    21525 non-null int64
gender              21525 non-null object
income_type         21525 non-null object
debt                21525 non-null int64
total_income        21525 non-null object
purpose             21525 non-null object
dtypes: int64(5), object(7)
memory usage: 2.0+ MB
```

Out[319]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income	p
0	1	-8437.67	42	bachelor's degree	0	married	0	F	employee	0	40620.1	pur th
1	1	-4024.8	36	secondary education	1	married	0	F	employee	0	17932.8	car p
2	0	-5623.42	33	Secondary Education	1	married	0	M	employee	0	23341.8	pur th
3	3	-4124.75	32	secondary education	1	married	0	M	employee	0	42820.6	supple e
4	0	340266	53	secondary education	1	civil partnership	1	F	retiree	0	25378.6	t

It's easy to see that here we have some columns that have data in format that is deffinetly not right for us.

1. Total income is in 'object' format but should be a **floating point number**.
2. Days employed should also be a whole number, but there is something wrong with this column. I'm going to try to look into it and to find out what is possible to do to make it correct and than I will transform it to absolute values (we have some strange negative numbers there due to possibly human error) and change it to **int** format.

For changing format of total income column and days employed column i have decided to use different method, that is more suitable for working with strings and object data. I made it raise all the errors that may come up, so in case there were any errors i would have seen them and the change of format wouldn't have proceeded.

In [320]:

```
credit_scoring['total_income'] = pd.to_numeric(credit_scoring['total_income'], errors='raise')
#check if format of data was successfully changed
if credit_scoring['total_income'].dtypes =='float64':
    print ('Success!')
else: print ('Failure')
```

```
Success!
```

After that i have tried to find out what is wrong with days_employed column. Firstly i have looked at maximum, minimum and average amount of years employed from this table.

In [321]:

```
#Firstly I'hve temporally transform days employed column to floating point
credit_scoring['days_employed'] = pd.to_numeric(credit_scoring['days_employed'], errors='raise')
if credit_scoring['days_employed'].dtypes =='float64':
    print ('Success!')
else: print ('Failure')
```

```
Success!
```

In [322]:

```
#And now I've looked at what kind of numbers we have here, but transformed it from days to years.
(credit_scoring['days_employed'] / 365).describe()
```

Out[322]:

```
count    21525.000000
mean       155.284588
std        369.508940
min        -50.380685
25%         -6.899093
50%         -2.691868
75%          0.000000
max       1100.699727
Name: days_employed, dtype: float64
```

Here I found out that we have two kinds of corrupted data:

- Negative data;
- Unrealistically hign value of days employed.

In [323]:
```
#Let's have a look at what kind of data do we have here. We will devide the data we have here in 4 categories:
extreme_worker = credit_scoring[credit_scoring['days_employed'] >= 29200] # more than amount of days in 80 years
normal_worker = credit_scoring[(credit_scoring['days_employed'] < 29200) & (credit_scoring['days_employed'] > 0)]
negative_worker =  credit_scoring[credit_scoring['days_employed'] < 0]

extreme_worker_amount = extreme_worker['days_employed'].count()
normal_worker_amount = normal_worker['days_employed'].count()
unemployed_amount = unemployed['family_status'].count()
negative_worker_amount = negative_worker['days_employed'].count()
print ('Amount of people employed for more than 80 years:', extreme_worker_amount)
print ('Amount of people employed for less than 80 years, but not less than 0:', normal_worker_amount)
print ('Amount of unemployed people (who have worked 0 days):', unemployed_amount)
print ('Amount of people employed for less than 0 days:', negative_worker_amount)
print ('Total amount of workers:', extreme_worker_amount + normal_worker_amount + negative_worker_amount + unemployed_
amount)
```

```
Amount of people employed for more than 80 years: 3445
Amount of people employed for less than 80 years, but not less than 0: 0
Amount of unemployed people (who have worked 0 days): 2174
Amount of people employed for less than 0 days: 15906
Total amount of workers: 21525
```

So fisrstly here I saw that we don't have any clients, who have normal infromation about their amount of days employed. After that i tried to find out what is going on in each of these groups.

In [324]:
```
extreme_worker.head()
```

Out[324]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income	pı
4	0	340266.072047	53	secondary education	1	civil partnership	1	F	retiree	0	25378.572	to w
18	0	400281.136913	53	secondary education	1	widow / widower	2	F	retiree	0	9091.804	bı s ha
24	1	338551.952911	57	secondary education	1	unmarried	4	F	retiree	0	46487.558	trans comı real
25	0	363548.489348	67	secondary education	1	married	0	M	retiree	0	8818.041	b
30	1	335581.668515	62	secondary education	1	married	0	F	retiree	0	27432.971	trans comı real

In [325]:
```
#Here I had a theory that all this mess with very big numbers is happening only with clients who are currently retire
d. I checked that.
extreme_worker['income_type'].value_counts()
```

Out[325]:
```
retiree       3443
unemployed       2
Name: income_type, dtype: int64
```

My theory has almost confirmed. Most of the clients, who have extremely high value are retired and two are unemployed. Because for these group of people banks wouldn't really care for amount of days employed, we can change these values to '0'.

In [326]:
```
credit_scoring.loc[credit_scoring['days_employed'] >= 29200, 'days_employed'] = 0
```

After that I had a look at all the negative numbers.

In [327]:
```
negative_worker.head(5)
```

Out[327]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income	
0	1	-8437.673028	42	bachelor's degree	0	married	0	F	employee	0	40620.102	pur th
1	1	-4024.803754	36	secondary education	1	married	0	F	employee	0	17932.802	car p
2	0	-5623.422610	33	Secondary Education	1	married	0	M	employee	0	23341.752	pur th
3	3	-4124.747207	32	secondary education	1	married	0	M	employee	0	42820.568	supple e
5	0	-926.185831	27	bachelor's degree	0	civil partnership	1	M	business	0	40922.170	pur th

In [328]:
```
negative_worker['income_type'].value_counts()
```

Out[328]:
```
employee                     10014
business                      4577
civil servant                 1312
entrepreneur                     1
student                          1
paternity / maternity leave      1
Name: income_type, dtype: int64
```

In [329]:
```
negative_worker.describe()
```

Out[329]:

	children	days_employed	dob_years	education_id	family_status_id	debt	total_income
count	15906.000000	15906.000000	15906.000000	15906.000000	15906.000000	15906.000000	15906.000000
mean	0.630643	-2353.015932	40.018295	0.798378	0.969634	0.087326	27837.509634
std	1.428524	2304.243851	10.311818	0.554845	1.442263	0.282320	16980.846677
min	0.000000	-18388.949901	19.000000	0.000000	0.000000	0.000000	3418.824000
25%	0.000000	-3157.480084	32.000000	0.000000	0.000000	0.000000	17323.415000
50%	0.000000	-1630.019381	39.000000	1.000000	0.000000	0.000000	24181.535000
75%	1.000000	-756.371964	48.000000	1.000000	1.000000	0.000000	33839.106500
max	20.000000	-24.141633	75.000000	4.000000	4.000000	1.000000	362496.645000

In [330]:
```
#check if there are any people with negative income, who have worked longer than they lived
negative_worker[negative_worker['days_employed']/(-365) > negative_worker['dob_years']]
```

Out[330]:

children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income	purpos

After conducting this analysis I have found out that all the negative numbers in 'days_employed' column should be there because of systematic error that automaticly makes these numbers negative. I made these conclusions because this data is:

- Maximum amount of years employed is not longer than possible real data;
- No client has worked for longer than he has lived.

Therefore I expect this information to be relevant and I have decided to change this values to positive for them to be usable.

In [331]:
```
credit_scoring.loc[credit_scoring['days_employed'] < 0 , 'days_employed'] = credit_scoring.loc[credit_scoring['days_employed'] < 0 , 'days_employed'].abs()
#credit_scoring[credit_scoring['days_employed'] < 0] = credit_scoring[credit_scoring['days_employed'] < 0].abs()
credit_scoring.describe()
```

Out[331]:

	children	days_employed	dob_years	education_id	family_status_id	debt	total_income
count	21525.000000	21525.000000	21525.000000	21525.000000	21525.000000	21525.000000	21525.000000
mean	0.543275	1738.772191	43.495145	0.817236	0.972544	0.080883	24082.055063
std	1.379876	2234.171998	12.218213	0.548138	1.420324	0.272661	17583.554088
min	0.000000	0.000000	19.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	34.000000	1.000000	0.000000	0.000000	14178.053000
50%	0.000000	982.531720	43.000000	1.000000	0.000000	0.000000	21682.354000
75%	1.000000	2518.168900	53.000000	1.000000	1.000000	0.000000	31286.979000
max	20.000000	18388.949901	75.000000	4.000000	4.000000	1.000000	362496.645000

Then I have transformed the 'days_employed' column format to integrers.

In [332]:
```
credit_scoring['days_employed'] = credit_scoring['days_employed'].round(0).astype(int)
credit_scoring.info()
credit_scoring.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21525 entries, 0 to 21524
Data columns (total 12 columns):
children            21525 non-null int64
days_employed       21525 non-null int64
dob_years           21525 non-null int64
education           21525 non-null object
education_id        21525 non-null int64
family_status       21525 non-null object
family_status_id    21525 non-null int64
gender              21525 non-null object
income_type         21525 non-null object
debt                21525 non-null int64
total_income        21525 non-null float64
purpose             21525 non-null object
dtypes: float64(1), int64(6), object(5)
memory usage: 2.0+ MB
```

Out[332]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income	I
0	1	8438	42	bachelor's degree	0	married	0	F	employee	0	40620.102	pur th
1	1	4025	36	secondary education	1	married	0	F	employee	0	17932.802	car p
2	0	5623	33	Secondary Education	1	married	0	M	employee	0	23341.752	pur th
3	3	4125	32	secondary education	1	married	0	M	employee	0	42820.568	supple e
4	0	0	53	secondary education	1	civil partnership	1	F	retiree	0	25378.572	t

Conclusion

While conducting change of data types i have applied different methods of changing data type.

- For total_income column I have used *to_numeric* method, and have also added a check there for me to be sure that all the transformation has gone well.
- For days_employed I have firstly applied the same principle that I have used with total_income column. After that I have analysed all the data in there, dropped some corrupted extremely hign values (changed them to 0) and changed negative values to positive (they were negative due to some programic or system error). In the end I changed these values to integrer type by applying *astype* method.

Processing duplicates

The first thing that is needed to do when processing duplicates is to check if data has any completely simillar rows.

In [333]:
```
print ('Duplicate entries in the data:', credit_scoring.duplicated().sum())
print ('Total amount of entries:', credit_scoring.shape[0])
```

```
Duplicate entries in the data: 54
Total amount of entries: 21525
```

In [334]:
```
# examine duplicated rows
credit_scoring.loc[credit_scoring.duplicated(), :].head()
```

Out[334]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income
2849	0	0	41	secondary education	1	married	0	F	employee	0	0.0
4182	1	0	34	BACHELOR'S DEGREE	0	civil partnership	1	F	employee	0	0.0
4851	0	0	60	secondary education	1	civil partnership	1	F	retiree	0	0.0
5557	0	0	58	secondary education	1	civil partnership	1	F	retiree	0	0.0
7808	0	0	57	secondary education	1	civil partnership	1	F	retiree	0	0.0

In [335]:
```
credit_scoring = credit_scoring.drop_duplicates(keep='first') #drop duplicated rows, but keep one of each unique row.
credit_scoring = credit_scoring.dropna().reset_index(drop=True)
print ('Duplicate entries in the data:',credit_scoring.duplicated().sum())
print ('Total amount of entries:', credit_scoring.shape[0])
```

```
Duplicate entries in the data: 0
Total amount of entries: 21471
```

Next step is to check if data has any case relative duplicates. For that I change all the values in string columns to lower-case and check what kind of duplicates are there.

In [336]:
```
credit_scoring['education'] = credit_scoring['education'].str.lower()
credit_scoring['education'].value_counts()
```

Out[336]:
```
secondary education    15188
bachelor's degree       5251
some college             744
primary education        282
graduate degree            6
Name: education, dtype: int64
```

In [337]:
```
credit_scoring['income_type'].value_counts() #no need to change case here
```

Out[337]:
```
employee                       11091
business                        5080
retiree                         3837
civil servant                   1457
entrepreneur                       2
unemployed                         2
paternity / maternity leave        1
student                            1
Name: income_type, dtype: int64
```

In [338]:
```
credit_scoring['purpose'].value_counts()
```

Out[338]:
```
wedding ceremony                            793
having a wedding                            773
to have a wedding                           769
real estate transactions                    675
buy commercial real estate                  662
buying property for renting out             652
housing transactions                        652
transactions with commercial real estate    650
purchase of the house                       646
housing                                     646
purchase of the house for my family         638
construction of own property                635
property                                    633
transactions with my real estate            627
building a real estate                      625
buy real estate                             621
purchase of my own house                    620
building a property                         619
housing renovation                          607
buy residential real estate                 606
buying my own car                           505
going to university                         496
car                                         494
second-hand car purchase                    486
to own a car                                479
buying a second-hand car                    478
cars                                        478
to buy a car                                472
car purchase                                461
supplementary education                     460
purchase of a car                           455
university education                        452
education                                   447
to get a supplementary education            447
getting an education                        442
profile education                           436
getting higher education                    426
to become educated                          408
Name: purpose, dtype: int64
```

Also no need to change case here. But there are some values that need to be categorized in next chapter.

Conclusion

After analyzing the data we have found several dublicated rows. They could have appeared due to program error, that could have occured at any point of working with clients before the data was given to us. To solve this I've applied drop_duplicates method with keeping one of each duplicates in our table for us not to lose anything. Also the table indexing has been reset. In addition to that I have processed all case relative dublicates to make the data more usable.

Categorizing Data

First group that I've decided to categorize was *purpose*. I had a look at what information was there.

In [339]:
```
credit_scoring['purpose'].value_counts()
```

Out[339]:
```
wedding ceremony                           793
having a wedding                           773
to have a wedding                          769
real estate transactions                   675
buy commercial real estate                 662
buying property for renting out            652
housing transactions                       652
transactions with commercial real estate   650
purchase of the house                      646
housing                                    646
purchase of the house for my family        638
construction of own property               635
property                                   633
transactions with my real estate           627
building a real estate                     625
buy real estate                            621
purchase of my own house                   620
building a property                        619
housing renovation                         607
buy residential real estate                606
buying my own car                          505
going to university                        496
car                                        494
second-hand car purchase                   486
to own a car                               479
buying a second-hand car                   478
cars                                       478
to buy a car                               472
car purchase                               461
supplementary education                    460
purchase of a car                          455
university education                       452
education                                  447
to get a supplementary education           447
getting an education                       442
profile education                          436
getting higher education                   426
to become educated                         408
Name: purpose, dtype: int64
```

In [340]:
```
credit_scoring['purpose'].value_counts().shape[0]
```

Out[340]:
```
38
```

There is a lot of different data, that could have been combined in from 4 to 6 groups. After haveing a thorough look at it I have found out that there are some values that aren't specific enough. This values are: housing transactions, housing, property, real estate transactions, building a property, building a real estate, buy real estate, etc. It would have been good to have different groups for buying commertial real estate, buying private real estate, building commercial real estate, building private real estate. In real life we could have requested more information, maybe something that would have helped us to specify which goes where. But unfortunatly I didn't have this option here, therefore I have decided to put all of these purposes in a single group, called **real estate**.

Also for this groupping i could have used some stemming or lemminisation, but due to the fact that there are only 38 types of purposes and they have very different words in them, it's much quicker to sort them to group manually.

In [341]:
```
import nltk
from nltk.stem import SnowballStemmer
from nltk.stem import WordNetLemmatizer
wordnet_lemma = WordNetLemmatizer()
english_stemmer = SnowballStemmer('english')
def general_purpose (row):
    #Program to apply surtain general purpose, depending on what purpose is in the argument.

    #turn the purpose value into a list, made out of words in it
    words = nltk.word_tokenize(row)
    #make lemmas out of the nouns that are in this list
    purpose_lemma_nouns = [wordnet_lemma.lemmatize(w, pos=('n')) for w in words]
    #make lemmas out of verbs also
    purpose_lemma_nouns_verbs = [wordnet_lemma.lemmatize(w, pos=('v')) for w in purpose_lemma_nouns]

    #define the words that will be used to apply to the list one general purpose or another
    real_estate = ['estate', 'house', 'construction', 'property']
    wedding = ['wedding', 'wed']
    car = ['car', 'vehicle']
    education = ['university', 'educate', 'education']
    #process the data
    for word in purpose_lemma_nouns_verbs:
        if word in real_estate:
            return 'real estate'
        elif word in wedding:
            return 'wedding'
        elif word in car:
            return 'car'
        elif word in education:
            return 'education'

#check the function
print (general_purpose ('real estate transactions'))
print (general_purpose ('to become educated'))
print (general_purpose ('to have a wedding'))
print (general_purpose ('second-hand car purchase'))
```

```
real estate
education
wedding
car
```

Works fine!

In [342]:
```
credit_scoring['purpose_grouped'] = credit_scoring['purpose'].apply(general_purpose)
#create new column, called 'purpose_grouped' and put in it general purpose values using the general_purpose function
credit_scoring['purpose_grouped'].value_counts()
```

Out[342]:
```
real estate    10814
car             4308
education       4014
wedding         2335
Name: purpose_grouped, dtype: int64
```

Next step is to group the clients depending on their level of income.

In [343]:
```
print ('Maximum income:', credit_scoring.total_income.max())
print ('Minimum income:', credit_scoring.total_income.min())
print ('Average income:', credit_scoring.total_income.mean())
print ('Median income:', credit_scoring.total_income.median())
```

```
Maximum income: 362496.645
Minimum income: 0.0
Average income: 24142.62191937963
Median income: 21714.7
```

In [344]:
```
#same analysys, but without clients with no income
print ('Maximum income:', credit_scoring[credit_scoring['total_income']!=0].total_income.max())
print ('Minimum income:', credit_scoring[credit_scoring['total_income']!=0].total_income.min())
print ('Average income:', credit_scoring[credit_scoring['total_income']!=0].total_income.mean())
print ('Median income:', credit_scoring[credit_scoring['total_income']!=0].total_income.median())
```

```
Maximum income: 362496.645
Minimum income: 3306.762
Average income: 26787.568354658673
Median income: 23202.87
```

After reading several articles about separation in classes, I have decided to split all client in 5 groups. This will be done partly according to Pew sociological research (this is the best method to devide these groups, but this is the one I found out to be the most representative):

1. No income
2. Lower income: total income lower than 67% of median income (excluding 0 income);
3. Lower medium income: from 67% of median income (excluding 0 income) to 200% of median income;
4. Upper medium income: from 200% of median income (excluding 0 income) to 500% of median income;
5. Wealthy: from 500% of median income.

In [345]:
```
#create median income and maximum income variables to the data:
median_income = credit_scoring[credit_scoring['total_income']!=0].total_income.median()
max_income = credit_scoring[credit_scoring['total_income']!=0].total_income.max()

def income_class(row):
    """
The income class is returned according to the "income" value and the "median income" value, by using the following rul
es:
1. No income: for "income" value = 0
2. Lower income:  "income" value lower than 67% of median income;
3. Lower medium income: from 67% of median income to 200% of median income;
4. Upper medium income: from 200% of median income (excluding 0 income) to 500% of median income;
5. Wealthy: from 500% of median income.
"""
    #median_income = row['median_income']
    income = row ['total_income']
    if income == 0:
        return 'no income'

    elif income < median_income * 0.67:
        return 'lower income'

    elif income < median_income * 2:
        return 'lower medium income'

    elif income < median_income * 5:
        return 'upper medium income'

    else: return 'wealthy'

#Check the function:
row_values = [13000, 10000] #income and median income
row_columns = ['total_income', 'median_income'] #column names
row = pd.Series(data=row_values, index=row_columns)
income_class(row)
```

Out[345]: 'lower income'

In [346]:
```
credit_scoring['income_class'] = credit_scoring.apply(income_class, axis=1)
credit_scoring['income_class'].value_counts()
```

Out[346]:
```
lower medium income    13568
lower income            4128
no income               2120
upper medium income     1603
wealthy                   52
Name: income_class, dtype: int64
```

In [347]:
```
#create description table of what values do these incomes represent
income_description_values = [
    ['no income', 0],
    ['lower income', median_income * 0.67],
    ['lower medium income', median_income * 2],
    ['upper medium income', median_income * 5],
    ['wealthy', max_income]
]
income_description_columns = ['grouped income', 'total income lower than']
income_description = pd.DataFrame(data = income_description_values, columns = income_description_columns).round(1)
income_description
```

Out[347]:

	grouped income	total income lower than
0	no income	0.0
1	lower income	15545.9
2	lower medium income	46405.7
3	upper medium income	116014.4
4	wealthy	362496.6

In [348]:
```
credit_scoring.head()
```
Out[348]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income	p
0	1	8438	42	bachelor's degree	0	married	0	F	employee	0	40620.102	pur th
1	1	4025	36	secondary education	1	married	0	F	employee	0	17932.802	car p
2	0	5623	33	secondary education	1	married	0	M	employee	0	23341.752	pur th
3	3	4125	32	secondary education	1	married	0	M	employee	0	42820.568	supple e
4	0	0	53	secondary education	1	civil partnership	1	F	retiree	0	25378.572	t

After that I have decided to drop the columns that we won't be needing in our further analysis and save this to the new table, called ***credit_scoring_clear***.

In [349]:
```
credit_scoring_clear = credit_scoring.drop(columns=['education_id', 'family_status_id', 'gender', 'income_type', 'tota
l_income', 'purpose', 'days_employed', 'dob_years'])

credit_scoring_clear.head()
```
Out[349]:

	children	education	family_status	debt	purpose_grouped	income_class
0	1	bachelor's degree	married	0	real estate	lower medium income
1	1	secondary education	married	0	car	lower medium income
2	0	secondary education	married	0	real estate	lower medium income
3	3	secondary education	married	0	education	lower medium income
4	0	secondary education	civil partnership	0	wedding	lower medium income

Conclusion

In this step I have categorised to columns of data, according to the information we had in there:

1. For column *purpose* I have decided to group information in it manually, due to differences in the way purposes were written. Also there was some unclear information, that I would have asked to specify if I were working with real client. But unfortunatly I had to work with the information I had, so Ive categorised all the purposes here only in 4 groups, but I hope that it would be good enough for final analysis.
2. For column *'total_income* I have decided to categorize all data to 5 groups mostly according to the common perception of lower, middle, upper-middle and wealthy class. Also there was a group for clients with no income. Using these 5 groups of income we would be able to analyze the relation between income and repaing loan on time.

Step 3. Answer these questions

- Is there a relation between having kids and repaying a loan on time?

In [350]:
```
# agregathe amount of people who have dept, according to their number of children
#(don't forget to drop row with 20 children)

children_scoring_grouped = credit_scoring_clear[credit_scoring_clear['children'] != 20].groupby('children').agg({'deb
t': ['sum', 'count']})

#calculate percentage of people not returning their loan on time
children_scoring_grouped['convertion'] = (children_scoring_grouped['debt']['sum'] / children_scoring_grouped['debt'][
'count'] * 100).round(2)
children_scoring_grouped.sort_values(by = 'convertion', ascending = False)
```
Out[350]:

	debt		convertion
	sum	count	
children			
4	4	41	9.76
2	194	2052	9.45
1	445	4856	9.16
3	27	330	8.18
0	1063	14107	7.54
5	0	9	0.00

There isn't such huge amount of people who have 4 or 5 children, who asked for a loan, so it's better to group them together for more consistent data. And run it all again.

In [351]:
```
# agregate the amount of people who had dept, according to their number of children
#(don't forget to drop row with 20 children)
children_scoring_grouped = credit_scoring_clear[credit_scoring_clear['children'] != 20].groupby('children').agg({'deb
t': ['sum', 'count']})

#calculate percentage of people not returning their loan on time
children_scoring_grouped['convertion'] = (children_scoring_grouped['debt']['sum'] / children_scoring_grouped['debt'][
'count'] * 100).round(2)
children_scoring_grouped.sort_values(by = 'convertion', ascending = False)
```

Out[351]:

	debt		convertion
	sum	count	
children			
4	4	41	9.76
2	194	2052	9.45
1	445	4856	9.16
3	27	330	8.18
0	1063	14107	7.54
5	0	9	0.00

In [352]:
```
loan_without_children = credit_scoring_clear[credit_scoring_clear['children'] == 0]['children'].count()
loan_with_children = credit_scoring_clear[(credit_scoring_clear['children'] != 0) &
                                          (credit_scoring_clear['children'] != 20)]['children'].count()

print ('Amount of people without children, who took a loan:', loan_without_children)
print ('Amount of people with children, who took a loan:', loan_with_children)
```

```
Amount of people without children, who took a loan: 14107
Amount of people with children, who took a loan: 7288
```

Conclusion

Finally, looking at this data we can see definite connection between having kids and paying a loan on time.

1. People, who don't have children are more likely to pay their loan on time. They have only 7,5% rate of defaulting on their loan.
2. People who have 2 children are more likely to default on their loan that people who only have 1 child.
3. Due to our data people who have 3 and more children are even less likely not to pay their loan on time. But this cunclusion is not to be percepted as universal, because in our data amount of people who have 3 and more children is much less than the amount of people who don't have children or have one or two.
4. The last conclution that we can make here is that people who don't have any children are almost twice more likely to take a loan, that people who have children.

- Is there a relation between marital status and repaying a loan on time?

In [353]:
```
# agregate amount of people who have dept, according to their family status
family_status_scoring_grouped = credit_scoring_clear.groupby('family_status').agg({'debt': ['sum', 'count']})

#calculate percentage of people not returning their loan on time
family_status_scoring_grouped['convertion'] = (family_status_scoring_grouped['debt']['sum'] /
                                               family_status_scoring_grouped['debt']['count'] * 100).round(2)

family_status_scoring_grouped.sort_values(by = 'convertion', ascending = False)
```

Out[353]:

	debt		convertion
	sum	count	
family_status			
unmarried	274	2810	9.75
civil partnership	388	4163	9.32
married	931	12344	7.54
divorced	85	1195	7.11
widow / widower	63	959	6.57

Conclusion

From this analysis it's possible to make several assumptions:

1. Unmarried people and people who are currently in civil partnership have the biggest chance not to return the loan on time.
2. Widows/ers and divorced people are less likely to default on a loan than other groups.
3. Most of people who took a loan are married.

Extra analysis

Now let's combine maritial status with number of children to see how these statystics corespond.

In [354]:
```
credit_scoring_clear_less_children = credit_scoring_clear[credit_scoring_clear['children'] != 20]
#make a table without people with 20 children

#create pivot table that contains maritial status with number of children as columns,
#family status as rows and conversion in percentages for each group as values

family_state_pivot = credit_scoring_clear_less_children.pivot_table(index = ['family_status'],
                                columns = 'children', values = 'debt', aggfunc =lambda x: sum(x)*100/x.count()).reset_
index().round(2)
family_state_pivot
```

Out[354]:

children	family_status	0	1	2	3	4	5
0	civil partnership	8.35	11.79	8.75	14.29	0.00	0.0
1	divorced	7.02	6.65	8.64	9.09	0.00	NaN
2	married	6.90	8.22	9.46	6.83	10.34	0.0
3	unmarried	9.28	11.45	12.00	12.50	50.00	NaN
4	widow / widower	6.26	8.64	15.00	0.00	0.00	NaN

In this analysis I consider data for people with 4 or more children not reliable enough, because we really don't have this many clients with 4 or more children.

According to this pivot table we see that the hist risk of defaulting goes to widows with 2 children, single people with 2 or more children and for people in civil partnership with more than 3 children.

Divorsed people with one child or single people with one child or widows/ers without children have the lowest risk of defaulting on their loan. Let's check the amount of rows, matching these conditions for us to see if this data is reliable.

In [355]:
```
print ('Amount of widows/ers with 2 children:',
       credit_scoring_clear_less_children[(credit_scoring_clear_less_children['children'] == 2) &
      (credit_scoring_clear_less_children['family_status'] == 'widow / widower')]['children'].count())
print ('Amount of singles with 2 or more children:',
       credit_scoring_clear_less_children[(credit_scoring_clear_less_children['children'] >= 2) &
      (credit_scoring_clear_less_children['family_status'] == 'unmarried')]['children'].count())

print ('Amount of people in civil partnership with 3 or more children:',
      credit_scoring_clear_less_children[(credit_scoring_clear_less_children['children'] >= 3) &
    (credit_scoring_clear_less_children['family_status'] == 'civil partnership')]['children'].count())

print ('Amount of divorced people with 1 child:',
      credit_scoring_clear_less_children[(credit_scoring_clear_less_children['children'] == 1) &
      (credit_scoring_clear_less_children['family_status'] == 'divorced')]['children'].count())

print ('Amount of married people without children:',
      credit_scoring_clear_less_children[(credit_scoring_clear_less_children['children'] == 0) &
      (credit_scoring_clear_less_children['family_status'] == 'married')]['children'].count())

print ('Amount of widows/ers without children:',
      credit_scoring_clear_less_children[(credit_scoring_clear_less_children['children'] == 0) &
      (credit_scoring_clear_less_children['family_status'] == 'widow / widower')]['children'].count())
```

```
Amount of widows/ers with 2 children: 20
Amount of singles with 2 or more children: 85
Amount of people in civil partnership with 3 or more children: 66
Amount of divorced people with 1 child: 316
Amount of married people without children: 7473
Amount of widows/ers without children: 847
```

In [356]:
```
print('True or False is', True or False)
print('True and False is', True and False)
print('True == 1 is', True == 1.0)
print('True == 0 is', True == 0.0)
print('True * 2 is', True * 2)
print('0 == False is', 0 == False)
print('0 or 1 is', 0 or 1)
```

```
True or False is True
True and False is False
True == 1 is True
True == 0 is False
True * 2 is 2
0 == False is True
0 or 1 is 1
```

So here we see that data for widowers with 2 children, for singles with 2 or more children and for people in civil partnership with more than 3 children is not reliable enough, therefore we can't make any definite conclusions about groups that will be more likely to defoault on their loan.

Conclusion

All the data about correlation between amount of children and maritial status shows us some differences in statystics but not all of it is reliable enough, but there is one conclusion that we can deffinetly make here:

- Married people and widowers without children, and divorced people with only 1 child are much less likely to default on their loan than all other groups.

- Is there a relation between income level and repaying a loan on time?

In [357]:
```
# agregate amount of people who have dept, according to their income class
income_level_aggregated = credit_scoring_clear.groupby('income_class').agg({'debt': ['sum', 'count']})


#calculate percentage of people not returning their loan on time
income_level_aggregated['conversion'] = (income_level_aggregated['debt']['sum']* 100 /
                                         income_level_aggregated['debt']['count']).round(2)
income_level_aggregated
```

Out[357]:

	debt		conversion
	sum	count	
income_class			
lower income	331	4128	8.02
lower medium income	1120	13568	8.25
no income	170	2120	8.02
upper medium income	116	1603	7.24
wealthy	4	52	7.69

In [358]:
```
#print description of income class values
income_description
```

Out[358]:

	grouped income	total income lower than
0	no income	0.0
1	lower income	15545.9
2	lower medium income	46405.7
3	upper medium income	116014.4
4	wealthy	362496.6

Conclusion

So here we can make 3 conclusions:

1. Despite common perception that people with low income are more likely to pay their loan, acctually it turned out that people who have their income from 15546 to 46405 in a year are most likely to default on their loan.
2. People who earn more than 116000 a year have the lowest percentage of defaulting on a loan.
3. All the difference between these groups are acctually really small, relation here is negligible, so level of income actually should't be defining criteria for making decision if client is going to have debt on his loan.

- How do different loan purposes affect on-time repayment of the loan?

In [359]:
```
# agregate amount of people who have dept, according to their income class
purpose_aggregated = credit_scoring_clear.groupby('purpose_grouped').agg({'debt': ['sum', 'count']})


#calculate percentage of people not returning their loan on time
purpose_aggregated['conversion'] = (purpose_aggregated['debt']['sum']* 100 /
                                    purpose_aggregated['debt']['count']).round(2)
purpose_aggregated.reset_index().sort_values('conversion')
```

Out[359]:

	purpose_grouped	debt		conversion
		sum	count	
2	real estate	782	10814	7.23
3	wedding	186	2335	7.97
1	education	370	4014	9.22
0	car	403	4308	9.35

Conclusion

1. Most amount of loans were taken for different operations with real estate;
2. People who take loans for buying a car or getting education are more likely to default on their loans, than people buying or constructing real estate.
3. This data isn't conclusive enough, to make it more conclusive we need to request extra information about purposes for people to take their loans.

Step 4. General conclusion

After preposessing categorizing this data I can make several conclusions:

1. There's definite connection between amount of children and possibility to default on a loan. People who don't have children or have only one child are more likely to pay their loan on time.
2. People who are married or were married are also more likely to pay their loan on time.
3. There actually isn't this much connection between people level of income and chance that they will pay their loan on time.
4. People who take real estate loans have have the lowest chance to defaul on their loan (actually this common assumptions caused economic crysis in 2008, but it's not the point of our analysis). On the other hand giving loans on education or car purchase have the highest risk for banks.

Project Readiness Checklist

Put 'x' in the completed points. Then press Shift + Enter.

- [x] file open;
- [X] file examined;
- [X] missing values defined;
- [X] missing values are filled;
- [X] an explanation of which missing value types were detected;
- [X] explanation for the possible causes of missing values;
- [X] an explanation of how the blanks are filled;
- [X] replaced the real data type with an integer;
- [X] an explanation of which method is used to change the data type and why;
- [X] duplicates deleted;
- [X] an explanation of which method is used to find and remove duplicates;
- [X] description of the possible reasons for the appearance of duplicates in the data;
- [X] data is categorized;
- [X] an explanation of the principle of data categorization;
- [X] an answer to the question "Is there a relation between having kids and repaying a loan on time?";
- [X] an answer to the question " Is there a relation between marital status and repaying a loan on time?";
- [X] an answer to the question " Is there a relation between income level and repaying a loan on time?";
- [X] an answer to the question " How do different loan purposes affect on-time repayment of the loan?"
- [X] conclusions are present on each stage;
- [X] a general conclusion is made.