Data preprocessing project

Credit worthiness analysis

Table of Contents

- 1 Goal
- 2 Hypothesis
- 3 Description of the data
- 4 Imports
- 5 Input data
- 6 Descriptive statistics
- 7 Unique values
  - 7.1 Education
  - 7.2 Family status
- 8 Artifacts
  - 8.1 Age (dob_years)
  - 8.2 Days_employed
  - 8.3 Children
  - 8.4 Gender
- 9 Missing values
  - 9.1 Total income
  - 9.2 Days employed
- 10 Data type change
- 11 Duplicates
- 12 Categorization
  - 12.1 Income type
  - 12.2 Total income
  - 12.3 Purpose
  - 12.4 Days employed bins
- 13 Final df
- 14 Link between children and debt
- 15 Link between family status and debt
- 16 Link between income level and debt
- 17 Link between education and debt
- 18 Link between loan purpose and debt
- 19 Link between age and debt
- 20 Link between gender and debt

Goal

Prepare a report for a bank's loan division to find out if a customer's marital status, number of children, income and education level, loan purpose, gender and age have an impact on whether they will default on a loan.

Hypothesis

Based on common sense, we predict the following tendencies:

1. The more children a client has, the higher the default rate as they have more expenses;
2. Married people have lower default rates as the spouses support each other in case of difficulties;
3. The higher the income level, the lower the default rate because the financial situation in this case is more stable;
4. The higher the education level, the lower the default rate because there is a higher chance to find a well-paid job;
5. The more expensive a purpose of a loan, the higher the chances that a loan will be paid off;
6. Younger people have the highest default rates because of the less stable financial situation (on average), while senior people (in terms of work experience) have the lowest default rates;
7. Men default less often on a loan as on average they have higher income level.

Description of the data

- `children` : the number of children in the family
- `days_employed` : how long the customer has been working
- `dob_years` : the customer’s age
- `education` : the customer’s education level
- `education_id` : identifier for the customer’s education
- `family_status` : the customer’s marital status
- `family_status_id` : identifier for the customer’s marital status
- `gender` : the customer’s gender
- `income_type` : the customer’s income type
- `debt` : whether the customer has ever defaulted on a loan
- `total_income` : monthly income
- `purpose` : reason for taking out a loan

Imports

In [1085]:

```
import pandas as pd
import seaborn as sns
import numpy as np

from nltk.stem import SnowballStemmer
english_stemmer = SnowballStemmer('english')
import matplotlib.pyplot as plt
%matplotlib inline

print("Setup Complete")
```

```
Setup Complete
```

Input data

In [1086]:

```
# Read the csv file with the input data and examine the first 10 rows of the dat
a
df = pd.read_csv('credit_scoring_eng.csv')
df.head(10)
```

Out[1086]:

	children	days_employed	dob_years	education	education_id	family_status	family_statu
0	1	-8437.673028	42	bachelor's degree	0	married	
1	1	-4024.803754	36	secondary education	1	married	
2	0	-5623.422610	33	Secondary Education	1	married	
3	3	-4124.747207	32	secondary education	1	married	
4	0	340266.072047	53	secondary education	1	civil partnership	
5	0	-926.185831	27	bachelor's degree	0	civil partnership	
6	0	-2879.202052	43	bachelor's degree	0	married	
7	0	-152.779569	50	SECONDARY EDUCATION	1	married	
8	2	-6929.865299	35	BACHELOR'S DEGREE	0	civil partnership	
9	0	-2188.756445	41	secondary education	1	married	

Notes based on the table above:

- Check if 'education_id' correctly reflects the 'education' column and if so, get rid of the 'education' column, since duplicate columns make analysis less transparent; Same situation with 'family_status' but we will keep both columns for now because the id representation in this case is not as intuitive as with the education;
- 'Days_employed' column has negative values, it needs to be corrected;
- 'Purpose' column -> reduce the number of categories (and make them more clear: purchase of the house -> house)

Descriptive statistics

In [1087]:

```
# Check the general information about this df
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
```

From the table above we can make the following conclusions:

- The data set includes 21525 observations and 12 columns;
- The "debt" column contains the numerical labels and the other 11 columns are features;
- 6 features are numerical, the other 5 are categorical;
- Float data type should be changed to integer;
- Values are missing in the 'days_employed' and 'total_income columns'; The number of missing values (2174) is the same in both columns. Since the data is based on the credit history of customers and people are required to provide this information in order to get a loan, both these variables must have been filled. Probably, they are missing due to some technical errors;
- The column names seem to be correct.

In [1088]:

```
# Summarize the central tendency, dispersion and shape of the dataset's distribu
tion
df.describe()
```

Out[1088]:

	children	days_employed	dob_years	education_id	family_status_id	de
count	21525.000000	19351.000000	21525.000000	21525.000000	21525.000000	21525.0000
mean	0.538908	63046.497661	43.293380	0.817236	0.972544	0.0808
std	1.381587	140827.311974	12.574584	0.548138	1.420324	0.2726
min	-1.000000	-18388.949901	0.000000	0.000000	0.000000	0.0000
25%	0.000000	-2747.423625	33.000000	1.000000	0.000000	0.0000
50%	0.000000	-1203.369529	42.000000	1.000000	0.000000	0.0000
75%	1.000000	-291.095954	53.000000	1.000000	1.000000	0.0000
max	20.000000	401755.400475	75.000000	4.000000	4.000000	1.0000

From the table above we can make the following conclusions:

- The minimum value of -1 in 'children' column is an error that should be fixed. On average people in this data set have 0-1 children with the maximum of 20 (maybe check this number too);
- The min 'dob_years' is 0 which is also an error, need to find a way to fix it. On average, people in their 40s are taking a loan;
- Most people in the data set did not default on a loan as the mean value of the 'debt' column is much closer to 0 than to 1 (0.08);

Unique values

Education

In [1089]:

```
df['education'].str.lower().value_counts()
```

Out[1089]:

```
secondary education    15233
bachelor's degree       5260
some college             744
primary education        282
graduate degree            6
Name: education, dtype: int64
```

In [1090]:

```
df['education_id'].value_counts()
```

Out[1090]:

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

After having changed all strings in the 'education' column to lower case, we see that the numbers for each group are exactly the same, meaning no error in data. We will omit the 'education' column from our final data frame.

Family status

In [1091]:

```
df['family_status'].str.lower().value_counts()
```

Out[1091]:

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

In [1092]:

```
df['family_status_id'].value_counts()
```

Out[1092]:

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

We can see the same situation, the numbers are matching, we don't need both columns as they have the same data. In our final df we will keep the 'family_status' column as it is more descriptive.

Artifacts

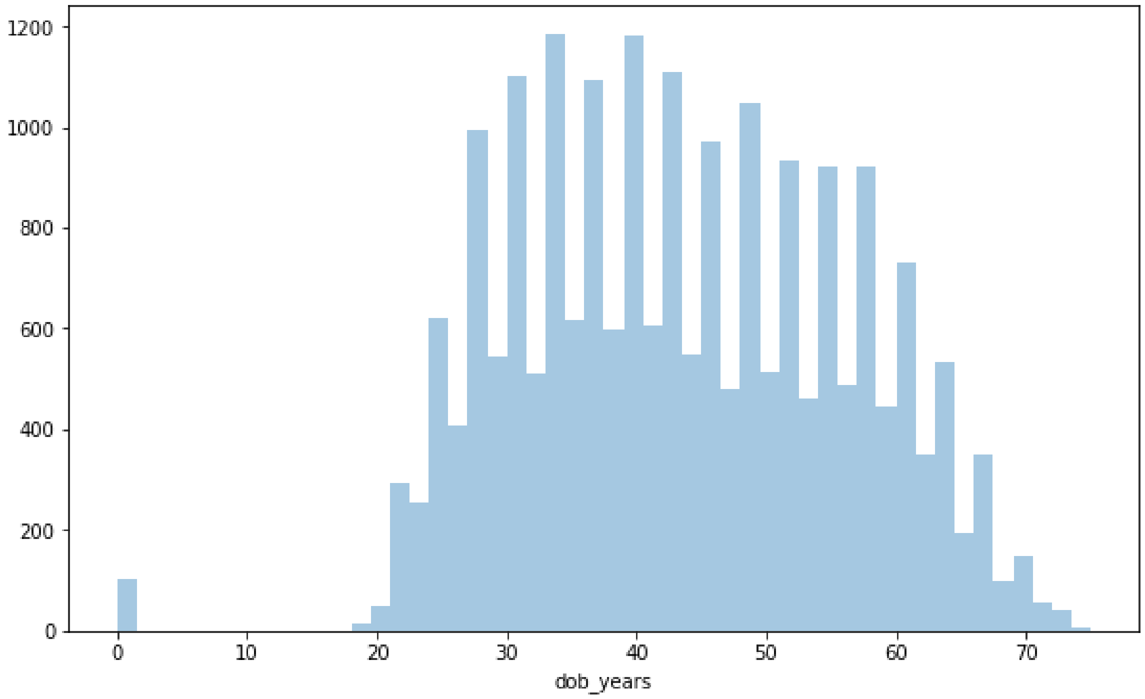
Age (dob_years)

In [1093]:

```
plt.figure(figsize=(10,6))
sns.distplot(df["dob_years"], kde=False)
```

Out[1093]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2aab4090>
```



The age in the dataset ranges from around 20 to 70 years old. We see an error where age = 0, let's fix that first. We will replace 0 with the median age of the whole column.

In [1094]:

```
df.loc[df['dob_years'] == 0, 'dob_years'] = df['dob_years'].median()
```

Let's also calculate the maximum age of this dataset. We will use it to identify the maximum possible value in the 'days_employed' column.

In [1095]:

```
df['dob_years'].max()
```

Out[1095]:

```
75.0
```

In [1096]:

```
df['dob_years'] = df['dob_years'].astype(int)
```

Days_employed

First of all, let's get rid of negative values. The values themselves are reasonable, so it looks like there has been an error with the sign of the value. We will correct it by multiplying all the negative values by -1.

In [1097]:

```
df.loc[df['days_employed'] < 0, 'days_employed'] = df.loc[df['days_employed'] <
0, 'days_employed']*(-1)
```

As we have calculated before, the maximum age of this dataset is 75. If we multiply it by 252 (assumed average working days a year) we get 18 900 days - the maximum possible number of working days (even less actually as one doesn't start working right after he was born but let's keep this value to have some kind of a baseline). We will thus replace all the values exceeding 18 900 first with 'NaN' and then all the missing values together will be replaced by the median value for each age group because on average the older a person gets the more days he has been working.

In [1098]:

```
df.loc[df['days_employed'] > 18900, 'days_employed'] = np.nan
```

Children

In [1099]:

```
df['children'].value_counts()
```

Out[1099]:

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

We see 47 rows with a negative value of -1 in the 'children' column. It could be either a human or a technical error. As the number of missing observations is insignificant (0.002) we will remove these values. The values of 20 in the children column is also suspicious but technically possible (considering foster children). As we don't have any additional information about it and it's also just a fraction of a dataset, let's keep them.

In [1100]:

```
df = df[df['children'] != -1]
```

Gender

In [1101]:

```
df['gender'].value_counts()
```

Out[1101]:

```
F      14201
M       7276
XNA        1
Name: gender, dtype: int64
```

In [1102]:

```
df[df['gender'] == 'XNA']
```

Out[1102]:

	children	days_employed	dob_years	education	education_id	family_status	family_stat
10701	0	2358.600502	24	some college	2	civil partnership	

Interestingly, twice as many female than male clients!

We see one row has a values of 'XNA'. Most likely an error. All other columns are filled, so in order not to loose valuable information let's fill it with the median value for this column.

In [1103]:

```
df.loc[df['gender'] == 'XNA', 'gender'] = 'F'
```

Missing values

Total income

Missing income values will be filled with the median value of a group based on the income type and education level. It is only logical that people with the same education (e.g. secondary) but different income types (e.g. business and retiree) have different income levels.

In [1104]:

```
df["total_income"] = df.groupby(['income_type','education_id'])["total_income"].
apply(lambda x: x.fillna(x.median()))
```

Days employed

Missing 'days_employed' values will be filled with the median value of a group based on the age and education level. The logic behind is that, on average, the older a person is, the more years he has been working and also, the higher his education level, the higher his chances to get a job.

To do that, first, we will categorize the 'dob_years' column into the following groups to make further analysis more transparent:

- dob_years < 30 -> '0-29'
- 30 <= dob_years < 40 -> '30-39'
- 40 <= dob_years < 50 -> '40-49'
- 50 <= dob_years < 60 -> '50-59'
- dob_years >= 60 -> '60+'

In [1105]:

```
def age_group(dob_years):
    """
    The function returns the age group according to the age value, using the fol
lowing rules:
    - dob_years < 30 -> '0-29'
    - 30 <= dob_years < 40 -> '30-39'
    - 40 <= dob_years < 50 -> '40-49'
    - 50 <= dob_years < 60 -> '50-59'
    - dob_years >= 60 -> '60+'
    """

    if dob_years < 30:
        return '0-29'
    if 30 <= dob_years < 40:
        return '30-39'
    if 40 <= dob_years < 40:
        return '40-49'
    if 50 <= dob_years < 60:
        return '50-59'
    return '60+'
```

In [1106]:

```
df['age_bins'] = df['dob_years'].apply(age_group)
df['age_bins'].value_counts().sort_values()
```

Out[1106]:

```
0-29     3178
50-59    4669
30-39    5659
60+      7972
Name: age_bins, dtype: int64
```

Next, we fill NaN values with the groups' median.

In [1107]:

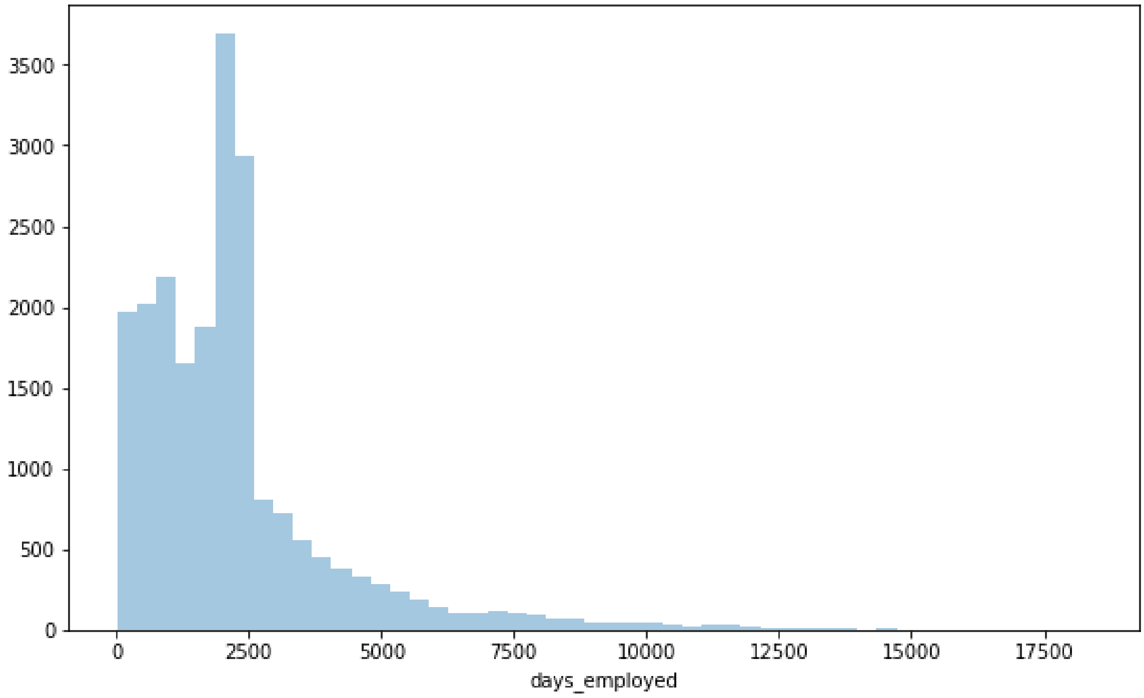
```
df["days_employed"] = df.groupby(['age_bins','education_id'])["days_employed"].a
pply(lambda x: x.fillna(x.median()))
```

In [1108]:

```
plt.figure(figsize=(10,6))
sns.distplot(df["days_employed"], kde=False)
```

Out[1108]:

<matplotlib.axes._subplots.AxesSubplot at 0x1a2b614210>



The final distribution of 'days_employed' now seems correct and logical, ranging from less than a year of work to up to 70 years, the distribution is positively skewed.

In [1109]:

```
df['days_employed'].skew()
```

Out[1109]:

2.39507280372904

Data type change

Data type 'days_employed' and 'total_income' columns should be changed to integer type. The first column contains total working days, so they must be integers and not floats. The second column could be a float type but we will still convert it into integers to make further analysis clearer, besides, exact income amount is not important as we will create categories later on.

In [1111]:

```
df[['days_employed','total_income']] = df[['days_employed','total_income']].asty
pe('int')
```

In [1113]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21478 entries, 0 to 21524
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   children          21478 non-null  int64 
 1   days_employed     21478 non-null  int64 
 2   dob_years         21478 non-null  int64 
 3   education         21478 non-null  object
 4   education_id      21478 non-null  int64 
 5   family_status     21478 non-null  object
 6   family_status_id  21478 non-null  int64 
 7   gender            21478 non-null  object
 8   income_type       21478 non-null  object
 9   debt              21478 non-null  int64 
 10  total_income      21478 non-null  int64 
 11  purpose           21478 non-null  object
 12  age_bins          21478 non-null  object
dtypes: int64(7), object(6)
memory usage: 2.3+ MB
```

As we can see from the above table, no more missing data and the data type is correct as well. Next step is removing duplicates.

Duplicates

We are going to look for full duplicates, meaning that every column value of the two rows must be the same because in this data set we don't have any column that could fully identify a customer (such as a personal id number for example).

In [1114]:

```
df.duplicated().sum()
```

Out[1114]:

```
56
```

In [1115]:

```
df.drop_duplicates(inplace=True, ignore_index=True)
```

Categorization

Income type

In [1116]:

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

Out[1116]:

```
employee                       11064
business                        5071
retiree                         3828
civil servant                   1453
unemployed                         2
entrepreneur                       2
student                            1
paternity / maternity leave        1
Name: income_type, dtype: int64
```

We have some outliers among income types. For further categorization, let's combine those values with bigger groups:

- For the purposes of this report a student and both unemployed people will be merged with the employee category because keeping a separate group for only 3 observations is meaningless but important columns for us are filled, so we don't want to loose valuable information. We will add them to the biggest group, so it won't change the distribution of the data;
- paternity / maternity leave goes to the employee category;
- entrepreneur goes to the business category;

In [1117]:

```
df.loc[df['income_type'] == 'student', 'income_type'] = 'employee'
df.loc[df['income_type'] == 'unemployed', 'income_type'] = 'employee'
df.loc[df['income_type'] == 'entrepreneur', 'income_type'] = 'business'
df.loc[df['income_type'] == 'paternity / maternity leave', 'income_type'] = 'emp
loyee'
```

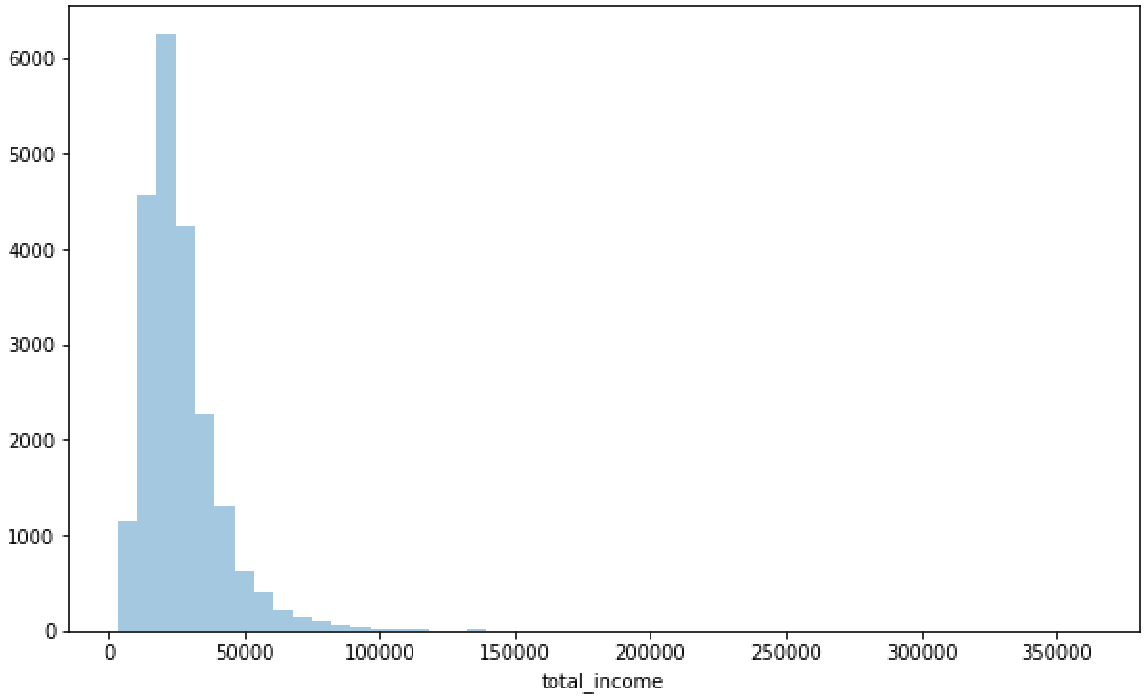
Total income

In [1118]:

```
plt.figure(figsize=(10,6))
sns.distplot(df["total_income"], kde=False)
```

Out[1118]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2b887310>
```



The 'total_income' column values range from around 3 000 to 300 000 (a difference of almost a 100 times) with an average of around 30 000, the income distribution is positively skewed.

In [1119]:

```
df['total_income'].skew()
```

Out[1119]:

```
4.149277216996072
```

To simplify the analysis we will categorize income data into 3 groups:

- below average
- average
- above average

We will consider average income to be a range from (mean - std) up to (mean + std). Below average are values below that range and above average - the values above that range.

In [1120]:

```
def income_group(total_income):
    """
    The function returns an income bin according to the total income value, usin
g the following rules:
    - total_income < (mean - std)  -> 'below average'
    - (mean - std) <= total_income <= (mean + std) -> 'average'
    - total_income > (mean + std) -> 'above average'
    """
    mean = df['total_income'].mean()
    std = df['total_income'].std()

    if total_income < (mean - std):
        return 'below average'
    if (mean - std) <= total_income <= (mean + std):
        return 'average'
    return 'above average'
```

In [1121]:

```
df['income_bins'] = df['total_income'].apply(income_group)
df['income_bins'].value_counts().sort_values()
```

Out[1121]:

```
below average     1277
above average     2317
average          17828
Name: income_bins, dtype: int64
```

Purpose

We can identify 4 main purposes for taking a loan:

- wedding
- real estate
- car
- education

Let's combine the data into these categories to make it easier to analysis.

In [1122]:

```
purpose_list = ['wedding', 'estate', 'property', 'house', 'car', 'education', 'u
niversity']
purpose_stem_list = [english_stemmer.stem(word) for word in purpose_list]
purpose_dict = {'wedding':'wed', 'real estate':['estat', 'properti', 'hous'], 'c
ar':'car', 'education': ['educ', 'univers']}
```

In [1123]:

```
def get_key(val):
    """
    If the val can be found in the dictinary.values() list,
    returns the key of the dictionary item in which the val was found.
    """
    wrong_val = []

    for key, value in purpose_dict.items():
        try:
            if val in value:
                return key
        except:
            wrong_val.append(key, value)
```

In [1124]:

```
def flatten(A):
    """
    Given a list of lists, returns the flattened list.
    """
    rt = []
    for i in A:
        if isinstance(i,list): rt.extend(flatten(i))
        else: rt.append(i)
    return rt
```

In [1125]:

```
def purpose_group(row):
    """
    Takes in a row of a data frame, finds the 'purpose' column, splits it into s
trings.
    Then for each word of that string, searches if the stem of the word is found
in the 'purpose_dict':
    purpose_dict = {'wedding':'wed', 'real estate':['estat', 'properti', 'hou
s'], 'car':'car', 'education': ['educ', 'univers']}
    Returns the key of the dictionary item in which the stem was found.
    """

    purpose = row['purpose']
    purpose_split = purpose.split(' ')
    wrong_content = []

    for word in purpose_split:
        try:
            stem_word = english_stemmer.stem(word)
            if stem_word in flatten(purpose_dict.values()):
                return get_key(stem_word)
        except:
            wrong_content.append(word)
```

In [1126]:

```
df['purpose_group'] = df.apply(purpose_group, axis=1)
```

In [1127]:

```
df['purpose_group'].value_counts()
```

Out[1127]:

```
real estate    10790
car             4296
education       4004
wedding         2332
Name: purpose_group, dtype: int64
```

Days employed bins

To categorize 'days_employed', we will assume that there are 252 working days a year and group them the following way:

- 0 to 3 years -> 'junior'
- 3 to 10 years -> 'experienced'
- more than 10 years -> 'senior'

In [1128]:

```
def work_experience(row):
    """
    The function returns a workdays bin according to the days_employed value, us
ing the following rules:
    - days_employed < 3*252  -> 'junior'
    - 3*252 <= days_employed <= 10*252 -> 'experienced'
    - days_employed > 10*252 -> 'senior'
    """

    days_employed = row['days_employed']
    if days_employed < 3*252:
        return 'junior'
    if days_employed > 10*252:
        return 'senior'
    return 'experienced'
```

In [1129]:

```
df['work_experience'] = df.apply(work_experience, axis=1)
```

Final df

Let's create a final data frame with only those columns that we are going to use for further analysis.

In [1130]:

```
final_col = ['children', 'work_experience', 'age_bins', 'education_id', 'family_
status', 'gender', 'income_type', 'income_bins', 'purpose_group', 'debt']
df_final = df[final_col]
df_final.head()
```

Out[1130]:

	children	work_experience	age_bins	education_id	family_status	gender	income_type	inc
0	1	senior	60+	0	married	F	employee	
1	1	senior	30-39	1	married	F	employee	
2	0	senior	30-39	1	married	M	employee	
3	3	senior	30-39	1	married	M	employee	
4	0	experienced	50-59	1	civil partnership	F	retiree	

Let's calculate an average debt percentage of the whole dataset:

In [1131]:

```
df_final['debt'].mean()
```

Out[1131]:

```
0.08122490897208477
```

Link between children and debt

In [1132]:

```
df_final.groupby('children')['debt'].mean()
```

Out[1132]:

```
children
0     0.075358
1     0.092346
2     0.094542
3     0.081818
4     0.097561
5     0.000000
20    0.105263
Name: debt, dtype: float64
```

We can see that the average debt percentage of each group is almost the same as the overall average (8%). As we predicted, people without children are doing a little bit better but the difference is not significant (0.6 to 3%). People with 20 children are most likely to fail on a loan. It can be explained by the much increased daily expenses and all the unexpected costs associated with children. People having 5 children in this dataset never failed on a loan but their number is very limited (9 people), so this result can be viewed as more of an exception.

Link between family status and debt

In [1133]:

```
df_final.groupby('family_status')['debt'].mean()
```

Out[1133]:

```
family_status
civil partnership    0.093337
divorced             0.071369
married              0.075524
unmarried            0.097683
widow / widower      0.065969
Name: debt, dtype: float64
```

Considering family status apart from other variables, we see that unmarried people along with people in a civil partnership have the highest default rates (9.7% and 9.3% respectively). In contrast, married people's rate is 7.5%. It could be explained by the fact that spouses usually support each other in difficult times. Let's have a look at the same data split by the number of children in a family.

In [1134]:

```
df_pivot_family = df_final.pivot_table(index=['children'],
                                       columns='family_status', values='debt', aggf
unc='mean')
df_pivot_family
```

Out[1134]:

family_status children	civil partnership	divorced	married	unmarried	widow / widower
0	0.083577	0.070153	0.069049	0.092838	0.062574
1	0.118474	0.067308	0.082717	0.115813	0.090909
2	0.087464	0.086420	0.094586	0.120000	0.150000
3	0.142857	0.090909	0.068273	0.125000	0.000000
4	0.000000	0.000000	0.103448	0.500000	0.000000
5	0.000000	NaN	0.000000	NaN	NaN
20	0.250000	0.500000	0.061224	0.111111	0.000000

As before, we see that overall, the more children a family has, the higher is the default rate, regardless of their family status. In this view, unmarried people default on a loan much more often that married ones, with an up to 40% difference. Interestingly, widowers don't have many children (up to 2), so that might be one of the reasons why they default less often.

Link between income level and debt

In [1135]:

```
df_pivot_income = df_final.pivot_table(index=['income_bins'],
                                       columns='income_type', values='debt', aggfun
c='mean')
df_pivot_income
```

Out[1135]:

income_type income_bins	business	civil servant	employee	retiree
above average	0.062165	0.035088	0.087879	0.044843
average	0.077228	0.064220	0.097227	0.057308
below average	0.060000	0.036145	0.089076	0.056112

Overall, retirees tend to default less often than others. Besides, people with average income default more often that those with income below or above average. The least risky are loans to civil servants with low or high income.

Link between education and debt

In [1136]:

```
df_final.groupby('education_id')['debt'].mean()
```

Out[1136]:

```
education_id
0    0.053043
1    0.089967
2    0.091521
3    0.109929
4    0.000000
Name: debt, dtype: float64
```

Interestingly, the higher the education level, the higher the default rate (with the exception of people with graduate degree who never defaulted on a loan but there are only 6 of them in the dataset, which is not significant enough to make a conclusion).

Link between loan purpose and debt

In [1137]:

```
df_final.groupby('purpose_group')['debt'].mean()
```

Out[1137]:

```
purpose_group
car            0.093575
education      0.092408
real estate    0.072475
wedding        0.079760
Name: debt, dtype: float64
```

The difference between groups is not very significant - around 2%. People who are planning a wedding or buying some property tend to be slightly more credit worthy.

Link between age and debt

In [1138]:

```
df_final.groupby('age_bins')['debt'].mean()
```

Out[1138]:

```
age_bins
0-29     0.109887
30-39    0.097557
50-59    0.065563
60+      0.067329
Name: debt, dtype: float64
```

Younger people tend to default more often on a loan. Let's see if we see the same tendency if we split this table by work experience.

In [1139]:

```
df_pivot_age = df_final.pivot_table(index=['work_experience'],
                                    columns='age_bins', values='debt', aggfunc=
'mean')
df_pivot_age
```

Out[1139]:

age_bins	0-29	30-39	50-59	60+
work_experience				
experienced	0.105030	0.099964	0.069392	0.063951
junior	0.129477	0.107286	0.090734	0.106822
senior	0.064748	0.085642	0.045789	0.054722

Overall senior people (in terms of work experience) default less often which can be explained by a more stable financial situation and higher wage. The same tendency can be observed age wise with an exception for young senior people who are on average more credit worthy than middle-aged seniors.

Link between gender and debt

First of all, let's examine the overall difference between men and women in terms of their credit worthiness.

In [1140]:

```
df_final.groupby('gender')['debt'].mean()
```

Out[1140]:

```
gender
F    0.070162
M    0.102765
Name: debt, dtype: float64
```

We see that women are 3.2% more likely to payoff a loan. It could probably be explained by the finding that has been replicated in a variety of studies over the years that men are more inclined to take risks than women. Now, let's compare these two groups split by income level and type.

In [1141]:

```
df_pivot_gender = df_final.pivot_table(index=['income_bins','income_type'],
                                        columns='gender', values='debt', aggfunc='me
an')
df_pivot_gender
```

Out[1141]:

	gender	F	M
income_bins	**income_type**		
above average	**business**	0.056723	0.067834
	civil servant	0.043011	0.025641
	employee	0.079498	0.095703
	retiree	0.053254	0.018519
average	**business**	0.069661	0.091295
	civil servant	0.057906	0.083056
	employee	0.082472	0.120544
	retiree	0.051799	0.081456
below average	**business**	0.011905	0.312500
	civil servant	0.025641	0.200000
	employee	0.087683	0.094828
	retiree	0.057604	0.046154

This overall tendency continues to be present in the split data: almost in every case females have a lower default rate than males, except for female retirees with income below and above average. The biggest difference between the two groups is for business people and civil servants with income below average (males are 30 and 10 times less credit worthy than females, respectively).

Conclusion

In this report we have analyzed data on customers' credit worthiness in order to discover connections between various client's characteristics and their ability to payoff loans.

First of all, we have familiarized ourselves with the data by performing the descriptive statistics. Then, we noticed that data is duplicated for 2 characteristics in the data set, so after making sure the data is identical in both columns for each characteristic, we made a decision to keep one of these columns.

We found a few artifacts in the data set that we have corrected: rows with age = 0, negative values for 'days_employed' and number of children, rows with gender value = 'XNA'.

Next step was to deal with missing values. We have missing values in two columns - 'total_income' and 'days_employed':

- Missing income values were filled with the median value of a group based on the income type and education level. We assumed that people with the same education (e.g. secondary) but different income types (e.g. business and retiree) have different income levels.
- Missing 'days_employed' values were filled with the median value of a group based on the age and education level. On average, the older a person is, the more years he has been working and also, the higher his education level, the higher his chances to get a job.

After that we have changed data type from 'float' to 'integer' for two columns - 'days_employed' and 'total_income' - to make further analysis clearer.

Next, we got rid of duplicate rows in the data set. We considered only fully identical rows to be duplicates because we don't have any column that could fully identify a customer (such as a personal id number for example).

Lastly, we have performed categorization of 4 columns: 'income type', 'total income', 'purpose', 'days employed':

- For 'income_type' we have identified 4 main groups and placed a few observations from other types into one of these main groups;
- To simplify the analysis we categorized 'total_income' data into 3 groups: below average, average, above average. We considered average income to be a range from (mean - std) up to (mean + std). Below average are values below that range and above average - the values above that range;
- We identified 4 main purposes for taking a loan: wedding, real estate, car, education. Then for each row we checked if a stem of one of those groups can be found in a 'purpose' column and replaced the value with it;
- We have grouped 'days_employed' column into 3 categories - junior, experienced and senior - based on the number of working years.

In the end, we created a final data frame with only those columns that we used for our final analysis. Based on our analysis, some of the predicted tendencies were correct while others were not:

1. Correct, the more children a client has, the higher the default rate;
2. Correct, married people have lower default rate;
3. Incorrect, people with average income default more often that those with income below or above average. Overall, retirees tend to default less often than others. The least risky are loans to civil servants with low or high income;
4. Incorrect, the higher the education level, the higher the default rate;
5. Incorrect. People do least often fail on a loan for the purpose of 'real estate' which is probably the most expensive among the 4 purposes, but the second most expensive must be education and people fail on it almost as often as on buying a car (which is on average the least expansive of all). It would be useful to see the amount of a loan in this case to make better conclusions;

6. Correct, younger people tend to default more often on a loan while senior people (in terms of work experience) have the lowest default rates;
7. Incorrect, women are more likely to payoff a loan. It could probably be explained by the finding that has been replicated in a variety of studies over the years that men are more inclined to take risks than women.