Hello Ameer!

My name is Elena. I'm happy to review your project today.

When I will see mistake at the first time, I will just point it out. I let you find it and fix it by yourself. I'm trying to prepare you to work as an Data Analyst. SO, at a real job, your team lead will do the same. But if you can't handle this task yet, I will give you a more accurate hint at the next iteration.

Below you will find my comments - please do not move, modify or delete them.

You can find my comments in green, yellow or red boxes like this:

Reviewer's comment Success. Everything is done succesfully.

Reviewer's comment Remarks. Some recommendations.

Reviewer's comment Needs fixing. The block requires some corrections. Work can't be accepted with the red comments.

You can answer me by using this:

Student answer.

Final reviewer's comment

The work is well structured, each step is described in detail. You've done a great job!

Ready:

1. There is an excellent structure with the addition of subtotals and commenting on your steps;
2. All available gaps are filled in, and duplicates are found and removed;
3. Well done data categorization;
4. The necessary replacements have been made in the "children" , "gender", "income_type" and "education" columns.

Final reviewer's comment

~~But there are a few remarks that need to be completed so that the work can be accepted: 1. Indicate possible reasons for data gaps and incorrect data; 2. Correct the data in the "days_employed" column (translate from hours to days); 3. Add your opinion about the presence of dependencies with logical reasoning to the conclusions in the "Hypothesis Test" block.~~

If you have any questions - write, I will definitely help to solve them.

I know you will succeed!

Student answer

1. Indicate possible reasons for data gaps and incorrect data; Done
2. Correct the data in the "days_employed" column (translate from hours to days); Done
3. Add your opinion about the presence of dependencies with logical reasoning to the conclusions in the "Hypothesis Test" block; Done

I appreciate your comments, really helped me understand my job way better and let me realize things I missed in my first submition. Thanks!

Final reviewer's comment 2

Thanks for fixing all the points. You've done well. You have a great job!

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 the **credit score** of a potential customer. The **credit score** is used to evaluate the ability of a potential borrower to repay their loan.

The purpose of this project is to check whether factors like; **number of children**, **family status**, **income level** and a couple other factors, have an impact on a customer defaulting a loan. Upon receiving the data about the bank customers' credit worthiness, we are going to analyze it bit by bit and hopefully conclude at the end which factors have the biggest impact on the bank customers on defaulting their loans.

Plan of work: We are going to start with preprocessing our data, looking for duplicates, errors in the data, missing values and any other problematic data that may be hiding in our dataset. Once we are done with the first step, we start with correcting the problematic data and then refill any missing values if there's any. when we are done with the preprocessing part, we move on into the analyzing part, in this part we process the data and compare it to one another, find certain dependencies and then conclude according to the results we achieved!

Reviewer's comment You did the right thing by adding a description of the projects and purpose. This is a good indicator of the quality work of the Data Analyst.

Reviewer's comment I would also suggest adding a Short Plan of Work. A short plan of work - here, highlight the points and, if necessary, sub-points, what actions you are going to perform. With it, it will be more convenient for you to complete the project yourself and know which stages you have already completed, and which ones you need to add. This will allow you to reach a new level and show future employers that you know what to do, you know how you will do it, and understand why. I am sure that you will succeed!

Student answer I actually had such list/plan on my tablet, checking (with a tick) every checkpoint I finish to make sure I covered everything. I didn't include it in the project because I thought it wasn't professional. I'll make sure to add such plan in my upcoming projects! (Added one here as well)

Reviewer's comment Well done! Glad to hear it!

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

In [1]:

```
import pandas as pd
from nltk.stem import SnowballStemmer

bank_data = pd.read_csv('/datasets/credit_scoring_eng.csv')
```

In [2]:

```
# Taking a look at the data
bank_data
```

Out[2]:

	children	days_employed	dob_years	education	education_id	family_status	family_stat
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	
...	...	...	...	...	...	...	
21520	1	-4529.316663	43	secondary education	1	civil partnership	
21521	0	343937.404131	67	secondary education	1	married	
21522	1	-2113.346888	38	secondary education	1	civil partnership	
21523	3	-3112.481705	38	secondary education	1	married	
21524	2	-1984.507589	40	secondary education	1	married	

21525 rows × 12 columns

Reviewer's comment It's good practice to split cells: 1st cell with library loading and 2nd cell with dataset reading. Please pay attention to this!

Student answer I am aware of that, it helps alot while still in testing phase since I don't need to run a-lot of code at once when everything is seperated in different cells. I should pay more attention to that.

Reviewer's comment If you need to restart all the above code, then it is most convenient to do this using the following method: find Kernel on the toolbar -> Restart & Run All

2 Task 1. Data exploration

Description of the data

- `children` - the number of children in the family
- `days_employed` - work experience in days
- `dob_years` - client's age in years
- `education` - client's education
- `education_id` - education identifier
- `family_status` - marital status
- `family_status_id` - marital status identifier
- `gender` - gender of the client
- `income_type` - type of employment
- `debt` - was there any debt on loan repayment
- `total_income` - monthly income
- `purpose` - the purpose of obtaining a loan

In [3]:

```
# Checking how many columns and rows does our data have
print('Our data has', bank_data.shape[0], 'rows and', bank_data.shape[1], 'columns!\n')
```

```
Our data has 21525 rows and 12 columns!
```

In [4]:

```
# Checking the first 20 rows in our dataset, to look for obvious errors/patterns
bank_data.head(20)
```

Out[4]:

	children	days_employed	dob_years	education	education_id	family_status	family_stat
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	
10	2	-4171.483647	36	bachelor's degree	0	married	
11	0	-792.701887	40	secondary education	1	married	
12	0	NaN	65	secondary education	1	civil partnership	
13	0	-1846.641941	54	some college	2	married	
14	0	-1844.956182	56	bachelor's degree	0	civil partnership	
15	1	-972.364419	26	secondary education	1	married	
16	0	-1719.934226	35	secondary education	1	married	
17	0	-2369.999720	33	bachelor's degree	0	civil partnership	
18	0	400281.136913	53	secondary education	1	widow / widower	
19	0	-10038.818549	48	SECONDARY EDUCATION	1	divorced	

Judging by looking at the first 20 rows in our data, I can already see errors in the data that will require further investigation. Some of these errors are; **missing values**, **negative time**, **different letter cases** (Look at 'education' column for instance, it has values of lower-case letters and other values of upper-case letters).

Below we have more information about the data we were given:

In [5]:

```
# Retreiving more information on our dataset
bank_data.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
```

If we look at the information that the **info()** method has provided, we can clearly see that most columns have 21525 values while 2 columns stand out for having less values. **days_employed** and **total_income** with only 19351 values, these columns have missing values obviously.

In [6]:

```
# Let's look at the filtered table with missing values in the the first column with missing
bank_data[bank_data['days_employed'].isna()].head(30)
```

Out[6]:

	children	days_employed	dob_years	education	education_id	family_status	family_sta
12	0	NaN	65	secondary education	1	civil partnership	
26	0	NaN	41	secondary education	1	married	
29	0	NaN	63	secondary education	1	unmarried	
41	0	NaN	50	secondary education	1	married	
55	0	NaN	54	secondary education	1	civil partnership	
65	0	NaN	21	secondary education	1	unmarried	
67	0	NaN	52	bachelor's degree	0	married	
72	1	NaN	32	bachelor's degree	0	married	
82	2	NaN	50	bachelor's degree	0	married	
83	0	NaN	52	secondary education	1	married	
90	2	NaN	35	bachelor's degree	0	married	
94	1	NaN	34	bachelor's degree	0	civil partnership	
96	0	NaN	44	SECONDARY EDUCATION	1	married	
97	0	NaN	47	bachelor's degree	0	married	
120	0	NaN	46	secondary education	1	married	
121	0	NaN	29	bachelor's degree	0	married	
135	0	NaN	27	secondary education	1	married	
141	0	NaN	39	secondary education	1	civil partnership	
145	0	NaN	62	secondary education	1	married	

	children	days_employed	dob_years	education	education_id	family_status	family_sta
174	0	NaN	55	bachelor's degree	0	widow / widower	
181	0	NaN	26	secondary education	1	civil partnership	
189	1	NaN	30	secondary education	1	unmarried	
205	1	NaN	31	bachelor's degree	0	married	
220	1	NaN	23	some college	2	civil partnership	
241	0	NaN	47	secondary education	1	married	
242	0	NaN	58	secondary education	1	married	
247	1	NaN	60	bachelor's degree	0	married	
250	0	NaN	54	bachelor's degree	0	married	
264	2	NaN	40	secondary education	1	divorced	
278	1	NaN	23	Secondary Education	1	civil partnership	

After filtering the data we have, keeping only rows with missing values in the **days_employed** column, we can notice symmetry between the **days_employed** column's missing values and the **total_income** column's missing values. But to prove that this symmetry is running through all the dataset we have to filter the dataset completely.

Please check the table below:

In [7]:

```
#Let's apply multiple conditions for filtering data and look at the number of rows in the f
bank_data[(bank_data['days_employed'].isna()) & (bank_data['total_income'].isna())]
```

Out[7]:

	children	days_employed	dob_years	education	education_id	family_status	family_s
12	0	NaN	65	secondary education	1	civil partnership	
26	0	NaN	41	secondary education	1	married	
29	0	NaN	63	secondary education	1	unmarried	
41	0	NaN	50	secondary education	1	married	
55	0	NaN	54	secondary education	1	civil partnership	
...	...	...	...	...	...	...	
21489	2	NaN	47	Secondary Education	1	married	
21495	1	NaN	50	secondary education	1	civil partnership	
21497	0	NaN	48	BACHELOR'S DEGREE	0	married	
21502	1	NaN	42	secondary education	1	married	
21510	2	NaN	28	secondary education	1	married	

2174 rows × 12 columns

In the table above (The table filtered according to missing values in both **days_employed** and **total_income**) we got the same results of data we got in the first table (The one filtered only according to **one of the conditions**). This means that the symmetry between **days_employed** and **total_income** runs through the whole dataset. In my opinion, this makes perfect sense because most of the people who do not work also do not have a reported income, this possibily means that the missing values are missing for a reason and it is not due to a mistake.

In [8]:

```
#Checking the percentage of the missing data among all data
percentage = bank_data[(bank_data['days_employed'].isna()) & (bank_data['total_income'].isn
print('The percentage of the missing data is {:.2%}'.format(percentage))
```

```
The percentage of the missing data is 10.10%
```

Intermediate conclusion

Does the number of rows in the filtered table match the number of missing values? What conclusion can we make from this?

According to the results above, the number of rows in the filtered table, the table with missing values in columns **days_employed** and **total_income**, is 2174 which is exactly the number of missing values in each column. This means that there's a complete symmetry between missing values in the **days_employed** and **total_income** columns.

After comparing the missing values with the whole dataset, we found that their percentage is ~10% which is actually a big piece of data. Therefore, we should fill those missing values. But first of all, we need to consider whether the missing data is due to a specific client characterisitic. Other than that, we are going to check any dependencies between missing values and other client characterisitics.

Next: We continue to check the correlation between the missing values and other values in our dataset to see if those missing values are missing for a specific reason like we said earlier or just randomly.

Reviewer's comment Great start! You're absolutely right about 10% - that's too much data to delete. That's right

In [9]:

```
#Below we print more information about the data we have, the distribution of missing values
print((bank_data[bank_data['total_income'].isnull()]['education']).value_counts(),'\n')
print((bank_data[bank_data['total_income'].isnull()]['income_type']).value_counts(),'\n')
print((bank_data[bank_data['total_income'].isnull()]['family_status']).value_counts(),'\n')
```

```
secondary education    1408
bachelor's degree       496
SECONDARY EDUCATION      67
Secondary Education      65
some college             55
Bachelor's Degree        25
BACHELOR'S DEGREE        23
primary education        19
SOME COLLEGE              7
Some College              7
Primary Education         1
PRIMARY EDUCATION         1
Name: education, dtype: int64

employee         1105
business          508
retiree           413
civil servant     147
entrepreneur        1
Name: income_type, dtype: int64

married              1237
civil partnership     442
unmarried             288
divorced              112
widow / widower        95
Name: family_status, dtype: int64
```

In [10]:

```
#Here we calculate the percentages of every characteristic with missing values of the total
total_missing_rows = bank_data[bank_data['total_income'].isna()].shape[0]
education_missing = (bank_data[bank_data['total_income'].isna()]['education']).value_counts
income_missing = (bank_data[bank_data['total_income'].isna()]['income_type']).value_counts(
family_missing = (bank_data[bank_data['total_income'].isna()]['family_status']).value_count

education_missing_dist = education_missing / total_missing_rows
income_missing_dist = income_missing / total_missing_rows
family_missing_dist = family_missing / total_missing_rows

print(education_missing_dist, '\n')
print(income_missing_dist,'\n')
print(family_missing_dist,'\n')
```

```
secondary education    0.647654
bachelor's degree      0.228151
SECONDARY EDUCATION    0.030819
Secondary Education    0.029899
some college           0.025299
Bachelor's Degree      0.011500
BACHELOR'S DEGREE      0.010580
primary education      0.008740
SOME COLLEGE           0.003220
Some College           0.003220
Primary Education      0.000460
PRIMARY EDUCATION      0.000460
Name: education, dtype: float64

employee         0.508280
business         0.233671
retiree          0.189972
civil servant    0.067617
entrepreneur     0.000460
```

Looking at the results we have above, we don't see any obvious relation between missing values and other characteristics. We actually expected that rows with missing values should have had **retiree** or **unemployed** as an **income_type** but what we actually got is around **70%** of rows had either **employee or buisness**, which adds up to the possibility that these missing values were random due to a human or a system error. This opposes to the assumption we assumed earlier that the missing values could've been due to a specific reason.

Possible reasons for missing values in data

In my opinion, the reason is most probably by humans collecting the data or simply a systematic error that caused values to disappear. But we cannot conclude on that yet, therefore we have to conduct yet further investigations, dig deep into our data.

Reviewer's comment You are on the right track.

In [11]:

```
# calculating the percentages of missing values according different specific characteristic
education_missing_dist_whole = education_missing / len(bank_data)
income_missing_dist_whole = income_missing / len(bank_data)
family_missing_dist_whole = family_missing / len(bank_data)

print(education_missing_dist_whole, '\n')
print(income_missing_dist_whole, '\n')
print(family_missing_dist_whole, '\n')
```

```
secondary education    0.065412
bachelor's degree      0.023043
SECONDARY EDUCATION    0.003113
Secondary Education    0.003020
some college           0.002555
Bachelor's Degree      0.001161
BACHELOR'S DEGREE      0.001069
primary education      0.000883
SOME COLLEGE           0.000325
Some College           0.000325
Primary Education      0.000046
PRIMARY EDUCATION      0.000046
Name: education, dtype: float64

employee         0.051336
business         0.023600
retiree          0.019187
civil servant    0.006829
entrepreneur     0.000046
Name: income_type, dtype: float64

married              0.057468
civil partnership    0.020534
unmarried            0.013380
divorced             0.005203
widow / widower      0.004413
Name: family_status, dtype: float64
```

Intermediate conclusion

The distribution in the original dataset is pretty similar to the distribution of the filtered data, this adds to the possibilty that these mistakes were more of random mistakes than anything else, but we still have to consider other factors and parameters in order to confirm this.

In order to confirm that the missing values are random, we will want to look for every parameter that could lead to errors if it was inserted incorrectly. One factor that comes to mind is zeros in the column **dob_years** because it doesn't make sense for a customer to have the age of 0. While we check whether the **dob_years** column has any zeros, we can also check for zeros on other columns where it doesn't make sense to be zero just to make sure (Check the code below)

In [12]:

```
mylist  = ['dob_years', 'days_employed', 'education', 'family_status', 'gender', 'income_ty

for value in mylist:
    if bank_data[bank_data[value] == 0].shape[0] > 0:
        print(value)
```

```
dob_years
```

In [13]:

```
bank_data[(bank_data['total_income'].isna()) & (bank_data['dob_years'] == 0)]
```

Out[13]:

	children	days_employed	dob_years	education	education_id	family_status	family_stat
1890	0	NaN	0	bachelor's degree	0	unmarried	
2284	0	NaN	0	secondary education	1	widow / widower	
4064	1	NaN	0	secondary education	1	civil partnership	
5014	0	NaN	0	secondary education	1	married	
6411	0	NaN	0	bachelor's degree	0	civil partnership	
6670	0	NaN	0	Bachelor's Degree	0	divorced	
8574	0	NaN	0	secondary education	1	married	
12403	3	NaN	0	secondary education	1	married	
13741	0	NaN	0	secondary education	1	civil partnership	
19829	0	NaN	0	secondary education	1	married	

We only found 10 records of dob_year as 0 which is a very very small portion of our dataset and despite that all 10 records have missing values on the columns **days_employed and total_income** which is what we expected, but the fact that this is a very small portion of our dataset means we can completely neglect this information as it doesn't explain the reason why the rest of the missing values exist.

Intermediate conclusion After the initial investigations we already conducted, we can say that the majority of missing values are not dependat on other values in our dataset and they exist due to random mistake and not due to a specific reason.

Conclusions To sum up, we already checked the correlation between the missing values with several other factors and we haven't reached to a conclusion where these missing values happened for a specific reason. Evenmore, with every check we conducted, we arrived to the conclusion that these missing values occured

randomly, *This means that the missing values are possibily due to a human error (when filling the data) or to a systematic error (bugs in the programs that process the dataset)*.

Since the rows with missing values comprise a very large piece of our dataset, therefore we are going to have to refill them instead of dropping them entirely. To do so, with **days_employed** we can refill them according to specific age categories, by this I mean that we will split the customers's ages into several categories and refill missing values with mean/median of the category it belongs to. When it comes to **total_income** we can refill them according to several other factors combined like **Education**, **Income Type** in a similar way to the way we are going to fill the missing data for **days_employed**.

Our next steps:

1. Check for duplicates
2. Transform exisitng values accordingly
3. Drop unneccasisry values
4. Refill missing values

Reviewer's comment I am very pleased to check your work. You structured it well, adding intermediate conclusions and commenting on your steps. This is a very important quality of Analytics given. Thank you. Keep it up!

Reviewer's comment

~~And what are your assumptions about the possible reasons for the presence of missing values? Please complete the project with your assumptions about the possible reasons for the missed readings.~~

Student answer I added another line, explaining why would the data be missing in our case.

Reviewer's comment Thanks for updating your answer.

3 Data transformation

Our first step is to check for **duplicates**. Unfortunately, the data we were given doesn't include any identifying characterisitc for every unique customer which means we do not have customer names or ids in order to identify them by these characterisitcs. Therefore, we can not assume we have any duplicates just according to rows with similar values because it is very likely that there are many customers with very similar data since our dataset is quite large.

Next we are going to check the **education** column for any errors that might be:

In [14]:

```
# Let's see the distribution of values in the `education` column
bank_data['education'].value_counts()
```

Out[14]:

```
secondary education    13750
bachelor's degree       4718
SECONDARY EDUCATION      772
Secondary Education      711
some college             668
BACHELOR'S DEGREE        274
Bachelor's Degree        268
primary education        250
Some College              47
SOME COLLEGE              29
PRIMARY EDUCATION         17
Primary Education         15
graduate degree            4
GRADUATE DEGREE            1
Graduate Degree            1
Name: education, dtype: int64
```

In [15]:

```
# Transforming the education column to all lower case letters
bank_data['education'] = bank_data['education'].str.lower()
```

In [16]:

```
bank_data['education'].value_counts()
```

Out[16]:

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

We noticed that the **education** column had similar values with different letter cases so we had to transform these values all into one form/letter case in order to have a better understanding of the data we have and for easier work with it later on.

Next we are going to check the **children** column for any errors that might be:

In [17]:

```
# Let's see the distribution of values in the `children` column
bank_data['children'].value_counts()
```

Out[17]:

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

Here we see that we have the values 20 and -1, which don't make alot of sense. These are most likely mistakes that should be dealt with. If we consider these mistakes as human mistakes, then it is reasonable to assume that these numbers have been 1 and 2 instead of -1 and 20. On the other hand, those could be systematic mistakes and we could replace them with the mean/median. Eitherway, the percentage of these values is very low that choosing either method won't really have a big impact on the conclusion. Therefore, we chose to replace them with the values 2 for 20 and 1 for -1:

Reviewer's comment Great guess and correct solution. Well done!

In [18]:

```
# [fix the data based on your decision]
bank_data.loc[bank_data['children'] == 20, 'children'] = 2
bank_data.loc[bank_data['children'] == -1, 'children'] = 1
```

In [19]:

```
# Checking the `children` column again to make sure it's all fixed
bank_data['children'].value_counts()
```

Out[19]:

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

Next we are going to check the **days_employed** column for any errors that might be:

In [20]:

```
# Find problematic data in `days_employed`, if they exist, and calculate the percentage
negative_values = bank_data[bank_data['days_employed'] < 0].shape[0]
neg_per = negative_values/bank_data.shape[0]

print('Negative values: {} rows, their percentage: {:.2%}'.format(negative_values, neg_per)
```

```
Negative values: 15906 rows, their percentage: 73.90%
```

We found that the problematic values in the **days_employed** column is very large. The problem is negative number of days which makes no sense. Since this portion of problematic values is very large, we need to find a way to deal with it. A very reasonable assumption here is to assume (We should first check this with the sources for better results) that the problematic/negative data is accutally right in the **absolute** form and change all negative values into positive values:

In [21]:

```
# Address the problematic values, if they exist
bank_data['days_employed'] = bank_data['days_employed'].abs()
```

In [22]:

```
# Check the result - make sure it's fixed
bank_data[bank_data['days_employed'] < 0].shape[0]
```

Out[22]:

```
0
```

In [23]:

```
print(bank_data['days_employed'].min())
print(bank_data['days_employed'].max())
bank_data[bank_data['days_employed'] >= 5000].shape[0]
```

```
24.14163324048118
401755.40047533
```

Out[23]:

```
5215
```

Looking at this range of values, we can see that there is something wrong in the **days_employed** column. If we divide any value above 15,000 with 365 (to count how many years each customer worked) we end up with 50 to 1000 years of work, which obviously doesn't make alot of sense. With so many cells of that type, we can only assume one of two assumptions;

1. The data we got is completely incorrect (we ask our sources for correct data)
2. The data we got is partly incorrect (we received the **days_employed** data as hours and not as days, we can also check with our sources here) We'll assume that in our situation we have case #2, therefore we should fix the values in the **days_employed** column by dividing it by 24.

In [24]:

```
bank_data['days_employed'] = bank_data['days_employed']/24
```

In [25]:

```
bank_data['days_employed'].max()
```

Out[25]:

```
16739.80835313875
```

Now the data makes more sense, with our max as 16739, this means the maximum number of years a customer has worked is 45 years instead of 1000 years as was previously.

Reviewer's comment Yes, you have solved this problem.

Next we are going to check the **dob_years** column for any errors that might be:

In [26]:

```
# Check the `dob_years` for suspicious values and count the percentage
bank_data['dob_years'].value_counts()
```

Out[26]:

```
35    617
40    609
41    607
34    603
38    598
42    597
33    581
39    573
31    560
36    555
44    547
29    545
30    540
48    538
37    537
50    514
43    513
32    510
```

The dataset is about a bank's customers' credit worthiness and it doesn't make sense that some customers are of the age 0. Therefore, since the number of cases with 0 **dob_years** is close to nothing, we decided to drop them of the table.

In [27]:

```
# Address the issues in the `dob_years` column, if they exist
bank_data.loc[bank_data['dob_years'] == 0] = None
bank_data = bank_data.dropna(how='all').reset_index(drop=True)
```

In [28]:

```
# Check the result - make sure it's fixed
print(bank_data[bank_data['dob_years'].isna()])
bank_data['dob_years'].value_counts()
```

```
Empty DataFrame
Columns: [children, days_employed, dob_years, education, education_id, famil
y_status, family_status_id, gender, income_type, debt, total_income, purpos
e]
Index: []
```

Out[28]:

```
35.0    617
40.0    609
41.0    607
34.0    603
38.0    598
42.0    597
33.0    581
39.0    573
31.0    560
36.0    555
44.0    547
29.0    545
30.0    540
48.0    538
37.0    537
50.0    514
43.0    513
32.0    510
49.0    508
28.0    503
45.0    497
27.0    493
56.0    487
52.0    484
47.0    480
54.0    479
46.0    475
58.0    461
57.0    460
53.0    459
51.0    448
59.0    444
55.0    443
26.0    408
60.0    377
25.0    357
61.0    355
62.0    352
63.0    269
64.0    265
24.0    264
23.0    254
65.0    194
66.0    183
22.0    183
67.0    167
21.0    111
```

```
68.0     99
69.0     85
70.0     65
71.0     58
20.0     51
72.0     33
19.0     14
73.0      8
74.0      6
75.0      1
Name: dob_years, dtype: int64
```

Next we are going to check the **family_status** column for any errors that might be:

In [29]:

```
# Let's see the values for the column
bank_data['family_status'].value_counts()
```

Out[29]:

```
married              12331
civil partnership     4156
unmarried             2797
divorced              1185
widow / widower        955
Name: family_status, dtype: int64
```

family_status doesn't seem to have any errors or problematic values.

Next we are going to check the **gender** column for any errors that might be:

In [30]:

```
# Check for gender counts
bank_data['gender'].value_counts()
```

Out[30]:

```
F      14164
M       7259
XNA        1
Name: gender, dtype: int64
```

In [31]:

```
# Address the problematic values, if they exist
bank_data.loc[bank_data['gender'] == 'XNA', 'gender'] = 'F'
```

In [32]:

```
# Check the result - make sure it's fixed
bank_data['gender'].value_counts()
```

Out[32]:

```
F    14165
M     7259
Name: gender, dtype: int64
```

gender column had one error. It was one row with the gender **XNA**. We dealt with it by transforming it into **F**, the bigger audience among the customers. **Next** we are going to check the **income_type** column for any errors that might be:

In [33]:

```
# Let's see the values in the column'
bank_data['income_type'].value_counts()
```

Out[33]:

```
employee                       11064
business                        5065
retiree                         3836
civil servant                   1453
entrepreneur                       2
unemployed                         2
student                            1
paternity / maternity leave        1
Name: income_type, dtype: int64
```

We didn't find any problematic values among the **income_type** values.

In this part of the project, we basically transformed a very small part of the dataset in order to make it reasonable and less problematic. Most of the dataset values were accurate and we didn't have to change alot in the general dataset.

Next: we move on to refilling the missing values!

Reviewer's comment You did a great job with this section. Made all the necessary changes.

Working with missing values

3.1 Restoring missing values in `total_income`

As we mentioned earlier, now we have to address missing values in the **days_employed** and **total_income** columns. To do so, with **days_employed** we can refill them according to specific age categories, by this I mean that we will split the customers's ages into several categories and refill missing values with mean/median of the

category it belongs to. When it comes to **total_income** we can refill them according to several other factors combined like **Education**, **Income Type**, **Age category** in a similar way to the way we are going to fill the missing data for **days_employed**.

First we want to create the **age_category** column:

In [34]:

```
# Let's write a function that calculates the age category
def get_age_category(age):
    if age < 25:
        return 'Age Group 1'
    elif age >= 25 and age < 35:
        return 'Age Group 2'
    elif age >= 35 and age < 50:
        return 'Age Group 3'
    else:
        return 'Age Group 4'
```

In [35]:

```
# Test if the function works
print(get_age_category(1))
print(get_age_category(120))
print(get_age_category(51))
print(get_age_category(49))
print(get_age_category(29))
```

```
Age Group 1
Age Group 4
Age Group 4
Age Group 3
Age Group 2
```

In [36]:

```
# Creating new column based on function
bank_data['age_category'] = bank_data['dob_years'].apply(get_age_category)
```

In [37]:

```
# Checking how values in the new column
bank_data.head(30)
```

Out[37]:

	children	days_employed	dob_years	education	education_id	family_status	family_status_i
0	1.0	351.569709	42.0	bachelor's degree	0.0	married	0
1	1.0	167.700156	36.0	secondary education	1.0	married	0
2	0.0	234.309275	33.0	secondary education	1.0	married	0
3	3.0	171.864467	32.0	secondary education	1.0	married	0
4	0.0	14177.753002	53.0	secondary education	1.0	civil partnership	1
5	0.0	38.591076	27.0	bachelor's degree	0.0	civil partnership	1
6	0.0	119.966752	43.0	bachelor's degree	0.0	married	0
7	0.0	6.365815	50.0	secondary education	1.0	married	0
8	2.0	288.744387	35.0	bachelor's degree	0.0	civil partnership	1
9	0.0	91.198185	41.0	secondary education	1.0	married	0
10	2.0	173.811819	36.0	bachelor's degree	0.0	married	0
11	0.0	33.029245	40.0	secondary education	1.0	married	0
12	0.0	NaN	65.0	secondary education	1.0	civil partnership	1
13	0.0	76.943414	54.0	some college	2.0	married	0
14	0.0	76.873174	56.0	bachelor's degree	0.0	civil partnership	1
15	1.0	40.515184	26.0	secondary education	1.0	married	0
16	0.0	71.663926	35.0	secondary education	1.0	married	0
17	0.0	98.749988	33.0	bachelor's degree	0.0	civil partnership	1
18	0.0	16678.380705	53.0	secondary education	1.0	widow / widower	2
19	0.0	418.284106	48.0	secondary education	1.0	divorced	3

	children	days_employed	dob_years	education	education_id	family_status	family_status_i
20	1.0	54.650174	36.0	secondary education	1.0	married	0
21	1.0	10.570215	33.0	secondary education	1.0	civil partnership	1
22	1.0	73.610172	24.0	secondary education	1.0	civil partnership	1
23	0.0	11.374224	21.0	bachelor's degree	0.0	civil partnership	1
24	1.0	14106.331371	57.0	secondary education	1.0	unmarried	4
25	0.0	15147.853723	67.0	secondary education	1.0	married	0
26	0.0	NaN	41.0	secondary education	1.0	married	0
27	0.0	22.049651	28.0	bachelor's degree	0.0	married	0
28	1.0	29.886430	26.0	bachelor's degree	0.0	married	0
29	0.0	NaN	63.0	secondary education	1.0	unmarried	4

We created the new column **age_category**, now we can check whether age categorization is a good factor to use in order to refill our missing values in the **total_income** column.

In [38]:

```
# Create a table without missing values and print a few of its rows to make sure it looks f
no_missing_data = bank_data[bank_data['total_income'].notna()]
```

In [39]:

```
# Look at the mean values for income based on your identified factors
no_missing_data.groupby('age_category')['total_income'].mean()
```

Out[39]:

```
age_category
Age Group 1    22703.351103
Age Group 2    27337.934929
Age Group 3    28583.378718
Age Group 4    24833.682909
Name: total_income, dtype: float64
```

In [40]:

```
# Look at the median values for income based on your identified factors
no_missing_data.groupby('age_category')['total_income'].median()
```

Out[40]:

```
age_category
Age Group 1    20572.209
Age Group 2    23990.901
Age Group 3    24795.835
Age Group 4    21387.208
Name: total_income, dtype: float64
```

We also want to consider other factors to refill the missing values in the **total_income** column like; **education** and **income_type**.

In [41]:

```
# Look at the median/mean values for income based on your identified factors
no_missing_data.groupby('education').agg({'total_income' : ['mean', 'median']})
```

Out[41]:

	total_income	
	mean	median
education		
bachelor's degree	33172.428387	28054.5310
graduate degree	27960.024667	25161.5835
primary education	21144.882211	18741.9760
secondary education	24600.353617	21839.4075
some college	29040.391842	25618.4640

In [42]:

```
# Look at the median/mean values for income based on your identified factors
no_missing_data.groupby('income_type').agg({'total_income' : ['mean', 'median']})
```

Out[42]:

	total_income	
	mean	median
income_type		
business	32397.357125	27564.8930
civil servant	27361.316126	24083.5065
employee	25824.679592	22815.1035
entrepreneur	79866.103000	79866.1030
paternity / maternity leave	8612.661000	8612.6610
retiree	21939.310393	18969.1490
student	15712.260000	15712.2600
unemployed	21014.360500	21014.3605

We decided to choose the **income_type** in order to refill the missing values in **total_income** because we think that the income type is most representitative over an individual income. Besides, the **income_type** is the more versatile column in terms of unique values and it covers more categories than other factors. We also decided to favor the use of **median** over **mean** in this case because in most cases mean is >10% bigger than the median which means there are some customers with very high incomes that push the mean up thus using the mean here doesn't reflect the data in the best way.

Next we want to refill the data according to the conclusion we reach to, first we start with writing a refill function:

In [43]:

```
#  Write a function that we will use for filling in missing values
def fill_missing_incomes(row):
    if not row['total_income'] >= 0:
        median = no_missing_data.groupby('income_type')['total_income'].median()
        return median[row['income_type']]
    else:
        return row['total_income']
```

In [44]:

```
# Check if it works
print(fill_missing_incomes(bank_data.loc[12]))
```

```
18969.148999999998
```

In [45]:

```
# Apply it to every row
bank_data['total_income'] = bank_data.apply(fill_missing_incomes, axis=1)
```

In [46]:

```
# Check if we got any errors
bank_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21424 entries, 0 to 21423
Data columns (total 13 columns):
children            21424 non-null float64
days_employed       19260 non-null float64
dob_years           21424 non-null float64
education           21424 non-null object
education_id        21424 non-null float64
family_status       21424 non-null object
family_status_id    21424 non-null float64
gender              21424 non-null object
income_type         21424 non-null object
debt                21424 non-null float64
total_income        21424 non-null float64
purpose             21424 non-null object
age_category        21424 non-null object
dtypes: float64(7), object(6)
memory usage: 2.1+ MB
```

We obviously can see that now we have 21424/21424 values in the **total_income** column which means there is no missing values no more.

Reviewer's comment Great job. Missing in the "total_income" column are filled in correctly and logically. You're doing great!

3.2 Restoring values in `days_employed`

Below we will refill missing values in the **days_employed** column according to the factors we mentioned earlier. But first lets check the correlation between **days_employed** and several other factors in order to decide which factor is the most representitative to be used to refill the missing values.

In [47]:

```
# Distribution of `days_employed` medians based on your identified parameters
no_missing_data = bank_data[bank_data['total_income'].notna()]
no_missing_data.groupby(['income_type', 'age_category']).agg({'days_employed' : ['mean', 'm
```

Out[47]:

		days_employed	
		mean	median
income_type	age_category		
business	Age Group 1	33.064881	28.998706
	Age Group 2	61.705389	48.687276
	Age Group 3	96.457813	75.790021
	Age Group 4	122.131050	86.089392
civil servant	Age Group 1	40.639499	38.432189
	Age Group 2	83.494639	77.704239
	Age Group 3	157.204027	140.430529
	Age Group 4	197.230892	156.132698
employee	Age Group 1	36.411008	32.073291
	Age Group 2	65.946306	53.086425
	Age Group 3	104.843795	75.916553
	Age Group 4	139.465015	95.581961
entrepreneur	Age Group 2	21.702003	21.702003
	Age Group 4	NaN	NaN
paternity / maternity leave	Age Group 3	137.364998	137.364998
retiree	Age Group 1	13948.510826	13948.510826
	Age Group 2	15171.566528	15268.837661
	Age Group 3	15285.831115	15285.042446
	Age Group 4	15206.050157	15207.727471
student	Age Group 1	24.114648	24.114648
unemployed	Age Group 2	14063.519451	14063.519451
	Age Group 3	16470.951611	16470.951611

In [48]:

```
no_missing_data.groupby('age_category').agg({'days_employed' : ['mean', 'median']})
```

Out[48]:

	days_employed	
	mean	median
age_category		
Age Group 1	53.443768	31.022589
Age Group 2	114.937151	53.842542
Age Group 3	411.425535	82.684296
Age Group 4	7781.781237	13739.269426

In [49]:

```
no_missing_data.groupby('income_type').agg({'days_employed' : ['mean', 'median']})
```

Out[49]:

	days_employed	
	mean	median
income_type		
business	88.033264	64.526564
civil servant	141.187856	111.391873
employee	97.025155	65.669487
entrepreneur	21.702003	21.702003
paternity / maternity leave	137.364998	137.364998
retiree	15208.988648	15215.680699
student	24.114648	24.114648
unemployed	15267.235531	15267.235531

According to the data we gathered so far, we decided to refill missing values in the **days_employed** column according the **age_category** because it represents it more than **income_type**. **income_type** doesn't really give perfect insight about **days_employed**, while **age_category** could be a better representatitive because most people of the similar age groups tend to work similar work hours/days since this is how our world works. We also decided to favor the **mean** this time due to the close results of mean and median.

In [50]:

```
# Let's write a function that calculates means or medians (depending on your decision) base
def fill_missing_days(row):
    if not row['days_employed'] >= 0:
        mean = no_missing_data.groupby('age_category')['days_employed'].mean()
        return mean[row['age_category']]
    else:
        return row['days_employed']
```

In [51]:

```
# Check that the function works
fill_missing_days(bank_data.loc[26])
```

Out[51]:

```
411.42553533149845
```

In [52]:

```
# Replacing missing values
bank_data['days_employed'] = bank_data.apply(fill_missing_days, axis=1)
```

In [53]:

```
# Check the entries in all columns - make sure we fixed all missing values
bank_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21424 entries, 0 to 21423
Data columns (total 13 columns):
children            21424 non-null float64
days_employed       21424 non-null float64
dob_years           21424 non-null float64
education           21424 non-null object
education_id        21424 non-null float64
family_status       21424 non-null object
family_status_id    21424 non-null float64
gender              21424 non-null object
income_type         21424 non-null object
debt                21424 non-null float64
total_income        21424 non-null float64
purpose             21424 non-null object
age_category        21424 non-null object
dtypes: float64(7), object(6)
memory usage: 2.1+ MB
```

Obviously, also here, we can see that we have 21424/21424 values on the **days_employed** column and this means that all missing values were filled (with the mean of each customer's age category)

Reviewer's comment

~~A very important quality of data analytics is the desire to understand why such data is presented in the data set and if there is a lot of it, is it possible to correct the situation on our own in order to answer questions of interest. There was a situation with the "days_employed" column when the data is presented not in the form of days that is convenient for us, but in the format of hours. Therefore, to get the correct numbers, you need to convert hours to days. Please complete this step.~~

Student answer I fixed this issue in the data transformation section rather in this one, thanks for the comment!

Reviewer's comment Thanks for correction. Please note that sometimes you have to find out of all the data only those that correspond to reality and correct only them :) This is because, ideally, it was necessary to make an adjustment not for the entire column, but only for a certain category.

4 Categorization of data

To answer the questions and test the hypotheses, we will want to work with categorized data. This way we can be more specific when testing our hypotheses on a very versatile column.

Among the very versatile datas are **purpose** and **total_income** therefore we are going to categorize these data into:

purpose - several categories; Wedding, Housing, Education and Car purchase. **total_income** - several levels; Very high, High, Normal, Average, Below average and Low.

In [54]:

```
# Print the values for your selected data for categorization
bank_data['purpose']
```

Out[54]:

```
0          purchase of the house
1                    car purchase
2          purchase of the house
3        supplementary education
4              to have a wedding
                   ...
21419       housing transactions
21420          purchase of a car
21421                   property
21422          buying my own car
21423               to buy a car
Name: purpose, Length: 21424, dtype: object
```

In [55]:

```
# Check the unique values
bank_data['purpose'].unique()
```

Out[55]:

```
array(['purchase of the house', 'car purchase', 'supplementary education',
       'to have a wedding', 'housing transactions', 'education',
       'having a wedding', 'purchase of the house for my family',
       'buy real estate', 'buy commercial real estate',
       'buy residential real estate', 'construction of own property',
       'property', 'building a property', 'buying a second-hand car',
       'buying my own car', 'transactions with commercial real estate',
       'building a real estate', 'housing',
       'transactions with my real estate', 'cars', 'to become educated',
       'second-hand car purchase', 'getting an education', 'car',
       'wedding ceremony', 'to get a supplementary education',
       'purchase of my own house', 'real estate transactions',
       'getting higher education', 'to own a car', 'purchase of a car',
       'profile education', 'university education',
       'buying property for renting out', 'to buy a car',
       'housing renovation', 'going to university'], dtype=object)
```

In [56]:

```
# Getting the stems of each purpose in order to use later
english_stemmer = SnowballStemmer('english')
list_of_purposes = ['wedding', 'house', 'car', 'property', 'estate', 'education', 'universi
for word in list_of_purposes:
    print(english_stemmer.stem(word))
```

```
wed
hous
car
properti
estat
educ
univers
```

In [57]:

```
# Let's write a function to categorize the data based on common topics
def get_purpose_category(purpose):
    list_of_stems = {'wed': 'Wedding', 'car': 'Car purchase', 'properti': 'Housing', 'estat
    eng_stem = SnowballStemmer('english')
    words = purpose.split(' ')
    for word in words:
        word = eng_stem.stem(word)
        if word in list_of_stems:
            return list_of_stems[word]
```

In [58]:

```
# Test the function
print(get_purpose_category('for the wedding'))
print(get_purpose_category('to buy a house'))
print(get_purpose_category('going for university'))
```

```
Wedding
Housing
Education
```

In [59]:

```
# Create a column with the categories and count the values for them
bank_data['purpose_category'] = bank_data['purpose'].apply(get_purpose_category)
```

In [60]:

```
bank_data['purpose_category'].value_counts()
```

Out[60]:

```
Housing         10793
Car purchase     4293
Education        4004
Wedding          2334
Name: purpose_category, dtype: int64
```

Reviewer's comment All right!

Next we categorize the **total_income** into several **income levels**:

In [61]:

```
# Looking through all the numerical data in your selected column for categorization
def get_income_level(income):
    if income <= 2500:
        return 'Low'
    if income <= 5000:
        return 'Below Average'
    if income <= 10000:
        return 'Average'
    if income <= 20000:
        return 'Normal'
    if income <= 50000:
        return 'High'
    else:
        return 'Very high'
```

In [62]:

```
# Test the function
print(get_income_level(2499))
print(get_income_level(10000))
print(get_income_level(45000))
print(get_income_level(350000))
```

```
Low
Average
High
Very high
```

In [63]:

```
# Applying the function to every row
bank_data['income_level'] = bank_data['total_income'].apply(get_income_level)
```

In [64]:

```
bank_data.head(30)
```

Out[64]:

	children	days_employed	dob_years	education	education_id	family_status	family_statu
0	1.0	351.569709	42.0	bachelor's degree	0.0	married	
1	1.0	167.700156	36.0	secondary education	1.0	married	
2	0.0	234.309275	33.0	secondary education	1.0	married	
3	3.0	171.864467	32.0	secondary education	1.0	married	
4	0.0	14177.753002	53.0	secondary education	1.0	civil partnership	
5	0.0	38.591076	27.0	bachelor's degree	0.0	civil partnership	
6	0.0	119.966752	43.0	bachelor's degree	0.0	married	
7	0.0	6.365815	50.0	secondary education	1.0	married	
8	2.0	288.744387	35.0	bachelor's degree	0.0	civil partnership	
9	0.0	91.198185	41.0	secondary education	1.0	married	
10	2.0	173.811819	36.0	bachelor's degree	0.0	married	
11	0.0	33.029245	40.0	secondary education	1.0	married	
12	0.0	7781.781237	65.0	secondary education	1.0	civil partnership	
13	0.0	76.943414	54.0	some college	2.0	married	
14	0.0	76.873174	56.0	bachelor's degree	0.0	civil partnership	
15	1.0	40.515184	26.0	secondary education	1.0	married	
16	0.0	71.663926	35.0	secondary education	1.0	married	
17	0.0	98.749988	33.0	bachelor's degree	0.0	civil partnership	
18	0.0	16678.380705	53.0	secondary education	1.0	widow / widower	
19	0.0	418.284106	48.0	secondary education	1.0	divorced	
20	1.0	54.650174	36.0	secondary education	1.0	married	

	children	days_employed	dob_years	education	education_id	family_status	family_statu
21	1.0	10.570215	33.0	secondary education	1.0	civil partnership	
22	1.0	73.610172	24.0	secondary education	1.0	civil partnership	
23	0.0	11.374224	21.0	bachelor's degree	0.0	civil partnership	
24	1.0	14106.331371	57.0	secondary education	1.0	unmarried	
25	0.0	15147.853723	67.0	secondary education	1.0	married	
26	0.0	411.425535	41.0	secondary education	1.0	married	
27	0.0	22.049651	28.0	bachelor's degree	0.0	married	
28	1.0	29.886430	26.0	bachelor's degree	0.0	married	
29	0.0	7781.781237	63.0	secondary	1.0	unmarried	

Reviewer's comment You correctly performed the categorization of the data.

5 Checking the Hypotheses

Reviewer's comment

~~One of the components of the work of a data analyst is the interpretation of the results obtained. The statement of facts is only part of this point. Please supplement each of the intermediate conclusions of this section with your opinion - why such results were obtained in the context of the existing groups.~~ ~~Please note that for big data, a deviation of 1% is of great importance.~~

Student answer Added my personal opinion on each and every one of the conclusions, hope they are good enough!

Regarding the deviation of 1%, I completely understand this fact now. I didn't look at the results from this POV first but now I understand that even a deviation of 1% is important when looking at the whole data. (Great example is the the 'married' group in the family_status category, while it was 0.5% percent below general percent, it actually comprised 50% of all those in debt. Imagine it was 1% greater than the general percentage, it would've comrpised much more than 50% of all those in debt and I would've actually thought about considering it one of the factors of our research!

Is there a correlation between having children and paying back on time?

In [65]:

```
# Check the children data and paying back on time
child_loan_data = bank_data.groupby('children')
child_loan_data['debt'].value_counts()
```

Out[65]:

```
children  debt
0.0       0.0     13022
          1.0      1058
1.0       0.0      4407
          1.0       442
2.0       0.0      1915
          1.0       202
3.0       0.0       301
          1.0        27
4.0       0.0        37
          1.0         4
5.0       0.0         9
Name: debt, dtype: int64
```

In [66]:

```
# Checking the precentages of people in debt according to children number
for i in range(5):
    people_with_debt = bank_data[(bank_data['debt'] == 1) & (bank_data['children'] == i)].s
    people_without_debt = bank_data[bank_data['children'] == i].shape[0]
    print('People with {} children debt ratio: {:.2%}'.format(i,people_with_debt/people_wit
```

```
People with 0 children debt ratio: 7.51%
People with 1 children debt ratio: 9.12%
People with 2 children debt ratio: 9.54%
People with 3 children debt ratio: 8.23%
People with 4 children debt ratio: 9.76%
```

In [67]:

```
# Calculating default-rate based on the number of children
people_on_debt = bank_data[bank_data['debt'] == 1].shape[0]
data_shape = bank_data.shape[0]
print('The percentage of people on debt from the overall is {:.2%}'.format(people_on_debt/d
```

```
The percentage of people on debt from the overall is 8.09%
```

Conclusion First we checked the percentage of people having debt compared to overall people with the same number of children and got the results: People with 0 children debt ratio: 7.51% People with 1 children debt ratio: 9.12% People with 2 children debt ratio: 9.54% People with 3 children debt ratio: 8.23% People with 4 children debt ratio: 9.76% (The number of customers that defaulted on a loan with x children / the number of customers with x children)

What we notice immediately here is that customers with 0 children had the least percentage, which makes perfect sense since they have to spend less. On the other hand, people with 4 children had the highest percentage among all of the groups, that's one thing that we should've also expected, since they have to spend the most. But despite these facts, we expected the percentages to have a wider range, a range of at least 10% instead of just 2.03% given the fact that raising kids nowadays isn't an easy task and is very expensive.

Later we calculated the overall percentage for people on debt and found it to be 8.09% which is very close to each and every one of other percentages. This means that the number of children doesn't really have an impact on a customer defaulting on a loan or not.

Reviewer's comment Excellent findings. Thank you for completing the research.

Is there a correlation between family status and paying back on time?

In [68]:

```
# Check the family status data and paying back on time
child_loan_data = bank_data.groupby('family_status')
child_loan_data['debt'].value_counts()
```

Out[68]:

```
family_status      debt
civil partnership  0.0      3770
                   1.0       386
divorced           0.0      1100
                   1.0        85
married            0.0     11404
                   1.0       927
unmarried          0.0      2524
                   1.0       273
widow / widower    0.0       893
                   1.0        62
Name: debt, dtype: int64
```

In [69]:

```
# Checking the precentages of people in debt according to family status
family_statuses = bank_data['family_status'].unique()
for i in family_statuses:
    people_with_debt = bank_data[(bank_data['debt'] == 1) & (bank_data['family_status'] ==
    people_without_debt = bank_data[bank_data['family_status'] == i].shape[0]
    print('People who are {} debt ratio: {:.2%}'.format(i,people_with_debt/people_without_d
```

```
People who are married debt ratio: 7.52%
People who are civil partnership debt ratio: 9.29%
People who are widow / widower debt ratio: 6.49%
People who are divorced debt ratio: 7.17%
People who are unmarried debt ratio: 9.76%
```

Conclusion

First we checked the percentage of people having debt that belong to a specific family status compared to the overall people with the same family status and got the results: People who are married debt ratio: 7.52% People who are civil partnership debt ratio: 9.29% People who are widow / widower debt ratio: 6.49% People who are divorced debt ratio: 7.17% People who are unmarried debt ratio: 9.76% (The number of customers that defaulted on a loan and are of x family status / the number of customers with x family status)

What we first notice here is that we have a wider range of percentages (3.27%) than previously (~2%). But still, no group really stands out to be elected as a possible factor for customers to default on loan. Why? If we look at the group with:

1. Highest percentage (unmarried), it turns out that in numbers it is 273 customers out of 1733 (All those in debt), which is only 15% of all in debt.
2. Highest number of customers in debt (married), with 927 of 1733 , around 50% of all those in debt. #2 shows that married people comprise around 50% of those in debt, more than unmarried people according to #1 (in perc.), but due to the fact that married people are actually around 57% of all the customers therefore it only makes sense that they comprise 50% of all of those in debt. (In my opinion, if the married group had 3%-4% higher ratio at least, only then we can consider it as a main factor of impact on customers defaulting their loans, but in the current situation it is not correct to confirm such thing)

Reviewer's comment Excellent!

Is there a correlation between income level and paying back on time?

In [70]:

```
# Check the income level data and paying back on time
child_loan_data = bank_data.groupby('income_level')
child_loan_data['debt'].value_counts()
```

Out[70]:

```
income_level   debt
Average        0.0       839
               1.0        56
Below Average  0.0        24
               1.0         2
High           0.0     11362
               1.0      1001
Normal         0.0      6239
               1.0       582
Very high      0.0      1227
               1.0        92
Name: debt, dtype: int64
```

In [71]:

```
# Checking the precentages of people in debt according to income level
income_levels = bank_data['income_level'].unique()
for i in income_levels:
    people_with_debt = bank_data[(bank_data['debt'] == 1) & (bank_data['income_level'] == i
    people_without_debt = bank_data[bank_data['income_level'] == i].shape[0]
    print('People with {} income level debt ratio: {:.2%}'.format(i,people_with_debt/people
```

```
People with High income level debt ratio: 8.10%
People with Normal income level debt ratio: 8.53%
People with Average income level debt ratio: 6.26%
People with Very high income level debt ratio: 6.97%
People with Below Average income level debt ratio: 7.69%
```

Conclusion

First we checked the percentage of people having debt that belong to a specific income level compared to the overall people with the same income level and got the results: People with High income level debt ratio: 8.10% People with Normal income level debt ratio: 8.53% People with Average income level debt ratio: 6.26% People

with Very high income level debt ratio: 6.97% People with Below Average income level debt ratio: 7.69% (The number of customers that defaulted on a loan of the x income level / the number of customers with x in come level)

At this part of our research, we expect people with Average and Below average incomes to have the highest percentages and/or numbers. For our surprise, people with Very high/High/Normal incomes comprise more than 80% of all those in debt (Due to the fact that they also comprise more than 70% of the whole customer base). We expect that this group of people should have the least amount of debts due to their high incomes but this isn't the case here. Yet, High/Normal income groups ratios are 8.10%/8.53% which is very close to the total ratio. Meaning, we cannot confirm that the income level really have an impact on our customers defaulting on a loan.

Reviewer's comment Good

How does credit purpose affect the default rate?

In [72]:

```
# Check the percentages for default rate for each credit purpose and analyze them
child_loan_data = bank_data.groupby('purpose_category')
child_loan_data['debt'].value_counts()
```

Out[72]:

```
purpose_category  debt
Car purchase      0.0      3893
                  1.0       400
Education         0.0      3634
                  1.0       370
Housing           0.0     10014
                  1.0       779
Wedding           0.0      2150
                  1.0       184
Name: debt, dtype: int64
```

In [73]:

```
purpose_cats = bank_data['purpose_category'].unique()
for i in purpose_cats:
    people_with_debt = bank_data[(bank_data['debt'] == 1) & (bank_data['purpose_category']
    people_without_debt = bank_data[bank_data['purpose_category'] == i].shape[0]
    print('People in {} purpose category debt ratio: {:.2%}'.format(i,people_with_debt/peop
```

```
People in Housing purpose category debt ratio: 7.22%
People in Car purchase purpose category debt ratio: 9.32%
People in Education purpose category debt ratio: 9.24%
People in Wedding purpose category debt ratio: 7.88%
```

Conclusion

First we checked the percentage of people having debt that belong to a specific purpose category compared to the overall people with the same purpose category and got the results: People in Housing purpose category debt ratio: 7.22% People in Car purchase purpose category debt ratio: 9.32% People in Education purpose

category debt ratio: 9.24% People in Wedding purpose category debt ratio: 7.88% (The number of customers that defaulted on a loan of the x purpose category / the number of customers with x in the purpose category)

Looking at the numbers, the Housing purpose group include 779 out of 1733 people in debt (due to the fact that there's ~10000 in this group/category) yet it's ratio is only 7.22% the lowest among all the other groups in this category. The Car purchase purpose group and the Education purpose group have the highest percentages with 400 and 370 customers in debt in each category respectively (23% and 21% of all people in debt respectively), together not even comprising 50% of all people in debt. None of the groups in this category stands out among the rest in this matter, therefore we can't consider any of them of high impact on the bank customers defaulting on their loans.

Reviewer's comment Fine. Thank you for adding each of the intermediate conclusions!

General Conclusion

To conclude generally I would like to sum up all conclusions mentioned above here: *In the following conclusion we may refer to our customers as either customers or people*

When we compared whether a customer would default on a loan according to the number of children, we ended up with those results: People with; 0 children: 7.51% had debt among all people with 0 children and its the lowest among this categories, 1 children: 9.12% had debt among all people with 1 children, 2 children: 9.54% had debt among all people with 2 children, 3 children: 8.23% had debt among all people with 3 children, 4 children: 9.76% had debt among all people with 4 children. While the overall percentage of people on debt was 8.09%. This means the percentages in each category were very close to the overall category and no category stands out in this matter (We actaully expected that people with more children would struggle on paying the loans therefore we would have higher percentages but this isn't the case here). Therefore, the number of children doesn't really have an impact on customers defaulting on their loans.

Next we compared whether a customer would default on a loan according to their family status, this part of the project we got these results: Married people had 7.52% percent in debt, people who were in a civil partnership with 9.29% people in debt, widow / widowers had 6.49% and the lowest percent of people on debt, divorced people had 7.17% people on debt and lastly unmarried people had 9.76% of the unmarried people in debt compared to all of the unmarried people. For this data we can conclude that family status doesn't really impact on customers defaulting on their loans since the overall percentage of people deafulting was, again, 8.09% which is very close to the rest of the percentages here which means no specific category stands out on customers defaulting loans.

In the third part we compared whether a customer would default on a loan according to their income level. The results we got were: People with a very high income had 6.97% of the people in this category in debt, while people with high income had 8.10% of the people in this category in debt. We expect that those categories have low people in debt ratio due to their high incomes and that is the case here. People with normal incomes had 8.53% of the people on this category in debt which completely makes sense, since we are still talking about people with relatively high incomes. People with average and below average incomes we got 6.26% and 7.69% respectively, the lowest among the percentages in these categories, which isn't really expected. Again, the percentages in these categories turn out to be very close to the overall percentage of people in debt (8.09%) and therefore we can not say that this factor has impact on customers defaulting on loans.

The last part, we compared whether a customer would default on a loan according to their initial purpose when they took the loan. The results we got were: People in Housing purpose category debt ratio was 7.22%, people in Car purchase purpose category debt ratio was 9.32%, people in Education purpose category debt ratio was

9.24% and people in Wedding purpose category debt ratio was 7.88%. We didn't expect to have any sub-category stand out here since all categories can be very costy in terms of loans. Our expectations were in place and all sub-categories had similar and close percentages, and those percentages were also very close to the overall percentage of people in debt which means that this factor didn't really have an impact on customers defaulting on their loans.

Going through all the conclusions we found out, we got some expected results on some data, on other data we got unexpected results. But despite all that, all conclusions shared the same theme, they **didn't have a direct impact on the customers defaulting on loans**. Perhaps, as the conclusions assume, there isn't a specific factor that really affects customer to default on loan or not, or maybe perhaps there is, but to find out, we'll have to conduct more investigations, include more data and take into consideration even more factors that could lead to that.

Final reviewer's comment

The work is well structured, each step is described in detail. You've done a great job!

Ready:

1. There is an excellent structure with the addition of subtotals and commenting on your steps;
2. All available gaps are filled in, and duplicates are found and removed;
3. Well done data categorization;
4. The necessary replacements have been made in the "children" , "gender", "income_type" and "education" columns.

Final reviewer's comment

~~But there are a few remarks that need to be completed so that the work can be accepted: 1. Indicate possible reasons for data gaps and incorrect data; 2. Correct the data in the "days_employed" column (translate from hours to days); 3. Add your opinion about the presence of dependencies with logical reasoning to the conclusions in the "Hypothesis Test" block.~~

If you have any questions - write, I will definitely help to solve them.

I know you will succeed!

Student answer Thanks for your time!

Final reviewer's comment 2

Thanks for fixing all the points. You've done well. You have a great job!

In [ ]: