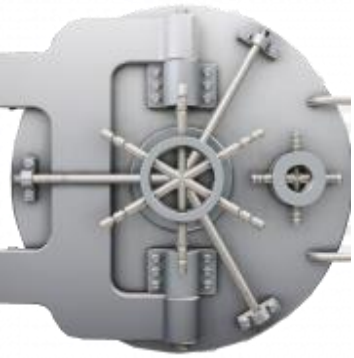


Bank Loan Case Study

AUTHORED BY

SUSHANT KARMAKAR

INDEX

1. Project Description

2. Problems and Planning

3. Preparation

4. Process and Analysis

5. Tech Stack Used

6. Insights

7. Conclusion

8. Drive Link

1. Project Description

This case study focuses on the practical application of exploratory data analysis (EDA) in a real-world business scenario, specifically within the context of risk analytics in the banking and financial services sector. The case study aims to demonstrate how EDA techniques can be used to analyze and visualize data, and how data can be leveraged to minimize the risk of losing money while lending to customers. Through this case study, you will gain a basic understanding of risk analytics and how data analysis can be used to inform decision-making in a business setting.

As an employee of a consumer finance company that provides loans to urban customers, you are aware of the challenge that many loan providers face - how to determine the creditworthiness of potential borrowers with little or no credit history. Unfortunately, some consumers take advantage of this situation by intentionally defaulting on loans. To address this issue, your company has tasked you with using exploratory data analysis (EDA) to analyze the patterns present in loan data. By conducting a thorough analysis of the data, you can identify factors that may predict an applicant's ability to repay a loan, thus ensuring that those who are capable of doing so are not unfairly rejected. This will ultimately lead to more responsible lending practices and a healthier lending portfolio for your company.

When a company receives a loan application, it needs to assess the applicant's profile to decide whether or not to approve the loan. The company faces two types of risks in this decision-making process. Firstly, if the applicant is likely to repay the loan, rejecting the loan may result in a loss of business for the company. On the other hand, if the applicant is not likely to repay the loan and is at risk of defaulting, approving the loan may lead to a financial loss for the company. Therefore, the company must carefully evaluate the applicant's creditworthiness and ability to repay the loan before making a decision on loan approval.

The data provided below pertains to loan applications at the time of application and can be categorized into two different scenarios.

- The first scenario pertains to clients who have had payment difficulties in the past. These clients have had late payments exceeding X days on at least one of the first Y installments of the loan in our sample. This data indicates that the client may have had trouble repaying loans in the past and may pose a higher risk for loan approval.

- The second scenario involves all other cases where the payments are made on time. This data indicates that the client has a good track record of timely repayments, and therefore, may be considered a lower risk for loan approval.

The information contained in this data can be used to assess the creditworthiness of the applicant and make informed decisions about loan approval. By carefully analyzing the payment history of the client, the lender can better evaluate the likelihood of timely repayment of the loan and determine whether to approve or reject the loan application.

When someone wants to borrow money from a company, there are four possible outcomes.

- The first is that the company approves the loan application, which means the person gets the money they need.
- The second is that the person cancels the application, either because they changed their mind about wanting the loan or because the company offered them a worse deal due to higher risk.
- The third is that the company refuses the loan, meaning the person does not get the money they wanted.
- The fourth possible outcome is that the loan is cancelled at some point during the approval process.

This case study aims to use Exploratory Data Analysis (EDA) to understand how different factors such as consumer attributes and loan attributes can affect the likelihood of defaulting on a loan. By analyzing these factors, we can gain insights into the risk of lending money to certain individuals and make more informed decisions about loan approval.

2. Problems & Planning

You are required to provide a detailed report for the below data record mentioning the answer to the questions that follows:

- **Task 1:** Present the overall approach of the analysis. Mention the problem statement and the analysis approach briefly
- **Task 2:** Identify the missing data and use appropriate method to deal with it. (Remove columns/or replace it with an appropriate value)

Hint: Note that in EDA, since it is not necessary to replace the missing value, but if you have to replace the missing value, what should be the approach. Clearly mention the approach.

- Identify if there are outliers in the dataset. Also, mention why do you think it is an outlier. Again, remember that for this exercise, it is not necessary to remove any data points.
- Identify if there is data imbalance in the data. Find the ratio of data imbalance.

Hint: Since there are a lot of columns, you can run your analysis in loops for the appropriate columns and find the insights.

- Explain the results of univariate, segmented univariate, bivariate analysis, etc. in business terms.
- Find the top 10 correlation for the Client with payment difficulties and all other cases (Target variable). Note that you have to find the top correlation by segmenting the data frame w.r.t to the target variable and then find the top correlation for each of the segmented data and find if any insight is there. Say, there are 5+1(target) variables in a dataset: Var1, Var2, Var3, Var4, Var5, Target. And if you have to find top 3 correlation, it can be: Var1 & Var2, Var2 & Var3, Var1 & Var3. Target variable will not feature in this correlation as it is a categorical variable and not a continuous variable which is increasing or decreasing.
- Include visualizations and summarize the most important results in the presentation. You are free to choose the graphs which explain the numerical/categorical variables. Insights should explain why the variable is important for differentiating the clients with payment difficulties with all other cases.

3. Preparation

A company that lends money to customers wants to understand the factors that contribute to loan default. These factors are known as driver variables, which are strong indicators that a borrower may not be able to repay their loan. By identifying these variables, the company can better assess the risk of lending to certain individuals and adjust their portfolio and risk assessment accordingly. By analyzing these variables, the company can develop a better understanding of the risk associated with each loan and make more informed decisions about loan approval. This can help the company minimize their losses and ensure a healthy portfolio of loans.

To develop the understanding of the domain, it has been advised to independently research a little about risk analytics – understanding the types of variables and their significance should be enough.

For analysis, the following dataset has been provided

- `application_data.csv` contains all the information of the client at the time of application. The data is about whether a client has payment difficulties.
- `previous_application.csv` contains information about the client's previous loan data. It contains the data whether the previous application had been Approved, Cancelled, Refused or Unused offer.
- `columns_descrption.csv` is data dictionary which describes the meaning of the variables.

4. Process & Analysis

Present the overall approach of the analysis. Mention the problem statement and the analysis approach briefly.

Problem Statement: This case study focuses on identifying patterns that can indicate whether a client may have difficulty paying their loan instalments. By identifying these patterns, the lender can take appropriate actions such as denying the loan, reducing the loan amount, or offering a higher interest rate to risky applicants. This approach can help ensure that clients who are capable of repaying the loan are not rejected and that the lender can minimize their risk of default.

The primary goal of this case study is to use Exploratory Data Analysis (EDA) to identify these patterns and determine which variables are strong indicators of loan default. By analyzing consumer and loan attributes, the study aims to develop insights into the risk associated with each loan application and make informed decisions about loan approval. This approach can help the lender better assess the creditworthiness of each applicant and reduce the risk of loan default.

Analysis Approach: The following steps involves the analysis approach

- Importing all the important libraries like numpy. pandas, matlplolib, seaborn and warnings to keep notebook clean.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

- Setting up the display properties for rows and columns

```
#setting up display properties
pd.set_option('display.max_columns', 200)
pd.set_option('display.max_rows', 200)
```

- Loading the dataset “application_data.csv”

```
#loading the dataset "application_data.csv"
app_data = pd.read_csv("application_data.csv")
```

- To get the better insight of the loaded dataset, I'll be using head() for a quick inspection

```
app_data.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREI
0	100002	1	Cash loans	M	N	Y	0	202500.0	40659
1	100003	0	Cash loans	F	N	N	0	270000.0	129350
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	13500
3	100006	0	Cash loans	F	N	Y	0	135000.0	31268
4	100007	0	Cash loans	M	N	Y	0	121500.0	51300

- To understand the structure and dimension, I'll be using shape attribute

```
#For understanding the structure and dimension
app_data.shape
```

```
(307511, 122)
```

Therefore, it can be seen that the dataframe contains 307511 rows and 122 columns.

- I'll be checking the data types of all the columns using info function

```
app_data.info("all")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 122 columns):
 #    Column                       Dtype
---   ------                       -----
 0    SK_ID_CURR                   int64
 1    TARGET                       int64
 2    NAME_CONTRACT_TYPE           object
 3    CODE_GENDER                  object
 4    FLAG_OWN_CAR                 object
 5    FLAG_OWN_REALTY              object
 6    CNT_CHILDREN                 int64
 7    AMT_INCOME_TOTAL             float64
 8    AMT_CREDIT                   float64
 9    AMT_ANNUITY                  float64
 10   AMT_GOODS_PRICE              float64
 11   NAME_TYPE_SUITE              object
 12   NAME_INCOME_TYPE             object
 13   NAME_EDUCATION_TYPE          object
```

Hence, there are total 122 columns that has dtypes like float64(65), int64(41) and object(16).

- To understand the descriptive statistics and numerical features of the dataframe, I'll be using the describe function

```
#For understanding the statistical and numerical features
app_data.describe()
```

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIV
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000	3.072330e+05	307511.0000
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573909	5.383962e+05	0.0208
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737315	3.694465e+05	0.0138
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e+04	0.0002
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05	0.0100
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05	0.0188
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	0.0286
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	0.0725

So, upon inspection I found that there were many columns that include negative as well as the positive values. Also, there were many columns with way much higher values that needed to be normalised.

- I'll be checking the percentage of null values of all the columns and sort them in descending order.

```
#converting the null values into percentage
def null_values(df):
    return round((df.isnull().sum()*100/len(df)).sort_values(ascending = False),2)
```

```
#checking the null values
null_values(app_data)
```

```
COMMONAREA_MEDI              69.87
COMMONAREA_AVG               69.87
COMMONAREA_MODE              69.87
NONLIVINGAPARTMENTS_MODE     69.43
NONLIVINGAPARTMENTS_AVG      69.43
NONLIVINGAPARTMENTS_MEDI     69.43
FONDKAPREMONT_MODE           68.39
LIVINGAPARTMENTS_MODE        68.35
LIVINGAPARTMENTS_AVG         68.35
LIVINGAPARTMENTS_MEDI        68.35
FLOORSMIN_AVG                67.85
FLOORSMIN_MODE               67.85
FLOORSMIN_MEDI               67.85
YEARS_BUILD_MEDI             66.50
YEARS_BUILD_MODE             66.50
YEARS_BUILD_AVG              66.50
OWN_CAR_AGE                  65.99
LANDAREA_MEDI                59.38
LANDAREA_MODE                59.38
```

Since, there are many columns that includes null values more than 50%, therefore I'll be separating them to a separate variable.

- Since, the maximum number of null values allowed is 25% to 30% and it varies from industry to industry, therefore I'll be removing all the columns with missing values more than 30% excluding the occupation type column which will be needed for further analysis. As occupation type column is excluded, therefore I'll take 40% as reference.
- For dropping off the columns, I will be using a new variable to the store the null columns.

```
#creating and storing null values with more than 40% of missing values in a separate variables
null_col = null_values(app_data)[null_values(app_data)>40]
```

- Before dropping off the columns, I did a recheck

```
#Doing a final check before dropping the columns
null_col.index
```

```
Index(['COMMONAREA_MEDI', 'COMMONAREA_AVG', 'COMMONAREA_MODE',
       'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAPARTMENTS_AVG',
       'NONLIVINGAPARTMENTS_MEDI', 'FONDKAPREMONT_MODE',
       'LIVINGAPARTMENTS_MODE', 'LIVINGAPARTMENTS_AVG',
       'LIVINGAPARTMENTS_MEDI', 'FLOORSMIN_AVG', 'FLOORSMIN_MODE',
       'FLOORSMIN_MEDI', 'YEARS_BUILD_MEDI', 'YEARS_BUILD_MODE',
       'YEARS_BUILD_AVG', 'OWN_CAR_AGE', 'LANDAREA_MEDI', 'LANDAREA_MODE',
       'LANDAREA_AVG', 'BASEMENTAREA_MEDI', 'BASEMENTAREA_AVG',
       'BASEMENTAREA_MODE', 'EXT_SOURCE_1', 'NONLIVINGAREA_MODE',
       'NONLIVINGAREA_AVG', 'NONLIVINGAREA_MEDI', 'ELEVATORS_MEDI',
       'ELEVATORS_AVG', 'ELEVATORS_MODE', 'WALLSMATERIAL_MODE',
       'APARTMENTS_MEDI', 'APARTMENTS_AVG', 'APARTMENTS_MODE',
       'ENTRANCES_MEDI', 'ENTRANCES_AVG', 'ENTRANCES_MODE', 'LIVINGAREA_AVG',
       'LIVINGAREA_MODE', 'LIVINGAREA_MEDI', 'HOUSETYPE_MODE',
       'FLOORSMAX_MODE', 'FLOORSMAX_MEDI', 'FLOORSMAX_AVG',
       'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BEGINEXPLUATATION_MEDI',
       'YEARS_BEGINEXPLUATATION_AVG', 'TOTALAREA_MODE', 'EMERGENCYSTATE_MODE'],
      dtype='object')
```

```
len(null_col)
```

```
49
```

- Now dropping off the null columns

```
#Dropping off the null columns
app_data.drop(columns = null_col.index, inplace = True)
```

- To confirm, I did a check on the main dataframe

```
#Checking for confirmation
app_data.shape
```

```
(307511, 73)
```

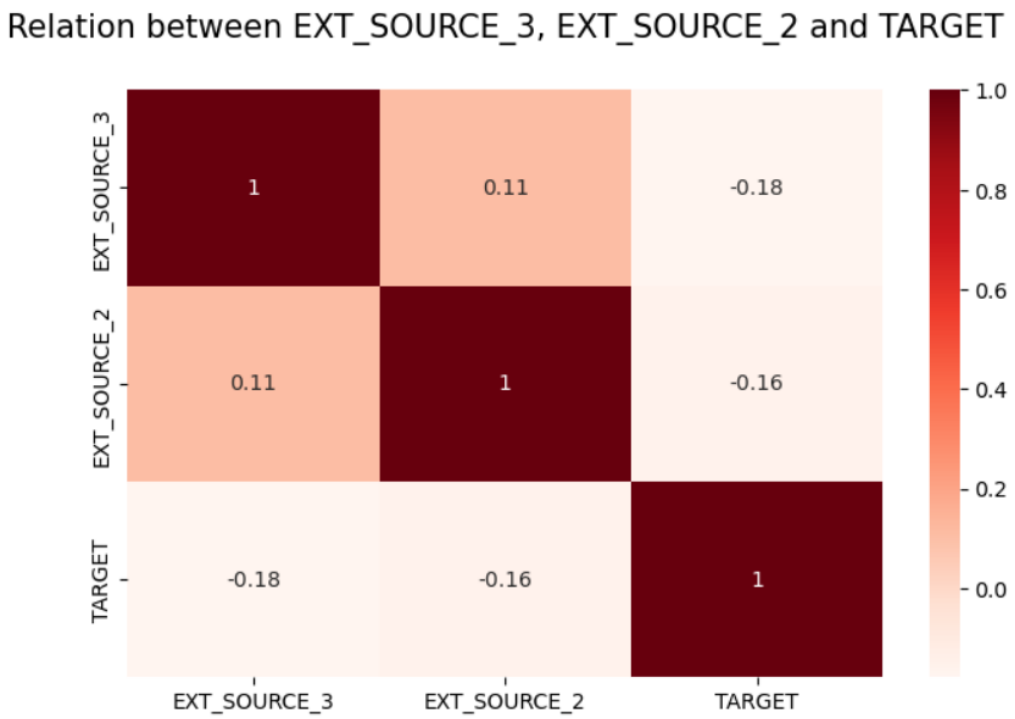
Therefore, after removal of null columns, only 73 columns were left

- Now, I'll be removing the irrelevant columns that won't be required for the analysis.
- To do this I will analyze the relationship between the columns EXT_SOURCE_3 and EXT_SOURCE_2, which have almost normalized values. To do this, I'll create a heatmap to visualize their correlation with the TARGET column.
- Importing the two columns to a new variable

```
#checking and removing irrelevent columns.
#importing columns to a new variable
irrelevant_col = ["EXT_SOURCE_3","EXT_SOURCE_2"]
```

- Plotting a heat map to check the relation between all the three columns

```
plt.figure(figsize= [8,5])
sns.heatmap(app_data[irrelevant_col+["TARGET"]].corr(), cmap="Reds",annot=True)
plt.title("Relation between EXT_SOURCE_3, EXT_SOURCE_2 and TARGET", fontdict={"fontsize":15}, pad=25)
plt.show()
```



From the above plot, it can be concluded that there happens to be no linear correlation among them. Hence, I'll be dropping these two columns.

- Dropping off the columns EXT_SOURCE_3 and EXT_SOURCE_2

```
#Since there is no correlation, therefore dropping off the columns
app_data.drop(irrelevant_col, axis=1, inplace= True)
```

- Checking for the total number of rows and columns after dropping off

```
#checking
app_data.shape
```

```
(307511, 71)
```

Therefore, after removing two columns, there are only 71 columns left.

- The following are the remaining columns left after dropping of the null data columns

```
app_data.columns
```

```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE',
       'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS',
       'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH',
       'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'FLAG_MOBIL',
       'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE',
       'FLAG_EMAIL', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS',
       'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY',
       'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
       'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
       'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
       'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY',
       'ORGANIZATION_TYPE', 'OBS_30_CNT_SOCIAL_CIRCLE',
       'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE',
       'DEF_60_CNT_SOCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2',
       'FLAG_DOCUMENT_3', 'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5',
       'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8',
       'FLAG_DOCUMENT_9', 'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11',
       'FLAG_DOCUMENT_12', 'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14',
       'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17',
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
       'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object')
```

- Now that I've removed all the unnecessary columns, next I'll be imputing the rest of the null cells with the appropriate values.

```
null_values(app_data).head(10)
```

```
OCCUPATION_TYPE               31.35
AMT_REQ_CREDIT_BUREAU_YEAR    13.50
AMT_REQ_CREDIT_BUREAU_QRT     13.50
AMT_REQ_CREDIT_BUREAU_MON     13.50
AMT_REQ_CREDIT_BUREAU_WEEK    13.50
AMT_REQ_CREDIT_BUREAU_DAY     13.50
AMT_REQ_CREDIT_BUREAU_HOUR    13.50
NAME_TYPE_SUITE                0.42
DEF_30_CNT_SOCIAL_CIRCLE       0.33
OBS_60_CNT_SOCIAL_CIRCLE       0.33
dtype: float64
```

Since there are only 7 columns left with null values more than 1%, therefore I'll only impute them for the further analysis.

- I'll be starting the imputation process with the OCCUPATION_TYPE column. At first I'll be checking the categories

```
#checking the categories before imputing for occupation_type column
app_data["OCCUPATION_TYPE"].value_counts(normalize=True)*100
```

```
Laborers                 26.139636
Sales staff              15.205570
Core staff               13.058924
Managers                 10.122679
Drivers                   8.811576
High skill tech staff     5.390299
Accountants               4.648067
Medicine staff            4.043672
Security staff            3.183498
Cooking staff             2.816408
Cleaning staff            2.203960
Private service staff     1.256158
Low-skill Laborers        0.991379
Waiters/barmen staff      0.638499
Secretaries               0.618132
Realty agents             0.355722
HR staff                  0.266673
IT staff                  0.249147
Name: OCCUPATION_TYPE, dtype: float64
```

- From the above details it can be stated that the column represent a categorical approach, therefore I'll be replacing the missing values with a new category such as "Unknown".

```
#imputing Unknown as a new category in place of null values
app_data["OCCUPATION_TYPE"] = app_data["OCCUPATION_TYPE"].fillna("Unknown")
```

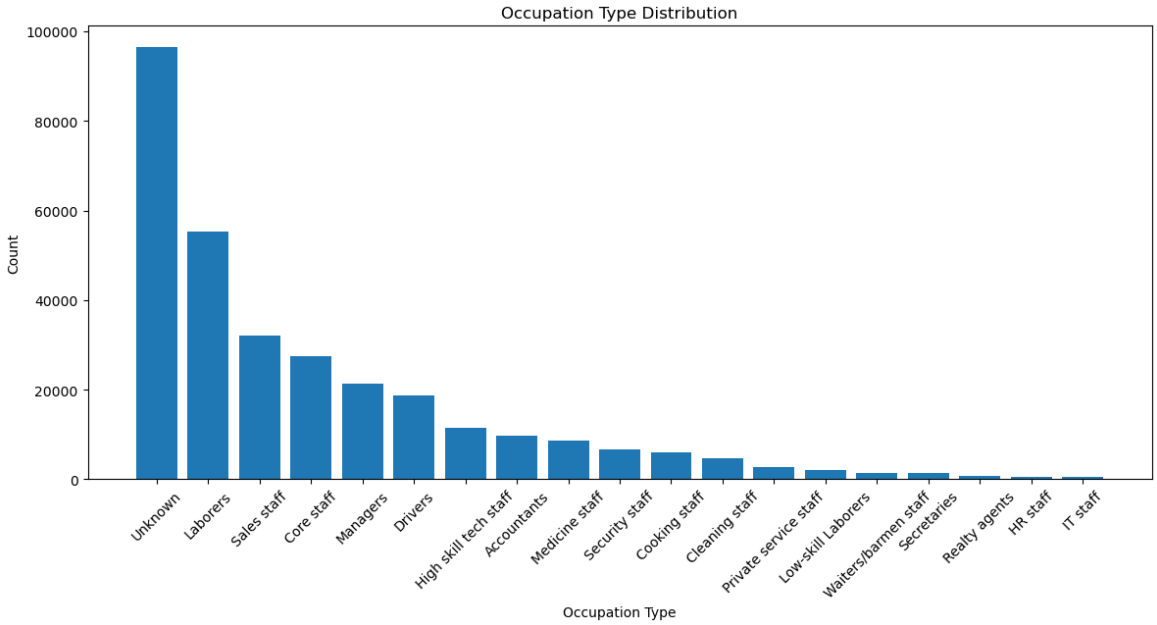
- After checking it has been seen that there were no null values left

```
#checking if any null values left
app_data["OCCUPATION_TYPE"].isnull().sum()
```

```
0
```

- Next, I'll be creating a plot to check the occupation type distribution

```
#Creating a plot to check occupation type distribution
occupation_data = app_data['OCCUPATION_TYPE']
occupation_counts = occupation_data.value_counts()
plt.figure(figsize=(14, 6))
fontdict={"fontsize":10}
plt.bar(occupation_counts.index, occupation_counts)
plt.xlabel('Occupation Type')
plt.ylabel('Count')
plt.title('Occupation Type Distribution')
plt.xticks(rotation=45)
plt.show()
```



- From the above plot it can be seen that the top three categories are Unknown, Laborers and Sales Staff.
- The next imputation process will be done with the columns having null values greater than 13%.

```
#Checking for the next columns needed to be imputed
null_values(app_data).head(10)
```

```
AMT_REQ_CREDIT_BUREAU_YEAR    13.50
AMT_REQ_CREDIT_BUREAU_HOUR    13.50
AMT_REQ_CREDIT_BUREAU_QRT     13.50
AMT_REQ_CREDIT_BUREAU_MON     13.50
AMT_REQ_CREDIT_BUREAU_WEEK    13.50
AMT_REQ_CREDIT_BUREAU_DAY     13.50
NAME_TYPE_SUITE                0.42
OBS_30_CNT_SOCIAL_CIRCLE       0.33
DEF_30_CNT_SOCIAL_CIRCLE       0.33
OBS_60_CNT_SOCIAL_CIRCLE       0.33
dtype: float64
```

- Checking the columns using describe function so that I can analyze the best possible way to impute

```
#checking the columns for best imputation process
app_data[["AMT_REQ_CREDIT_BUREAU_YEAR", "AMT_REQ_CREDIT_BUREAU_HOUR",
          "AMT_REQ_CREDIT_BUREAU_QRT", "AMT_REQ_CREDIT_BUREAU_MON",
          "AMT_REQ_CREDIT_BUREAU_WEEK", "AMT_REQ_CREDIT_BUREAU_DAY"]].describe()
```

- Since the above columns represent the details made for the customers that happens to be discreate data, therefore I'll be using mode to fill out the null values of those above columns.

- Replacing the null data with the mode for all the above mentioned columns

```
#Replacing the null data with the mode
app_data.AMT_REQ_CREDIT_BUREAU_YEAR.fillna(app_data.AMT_REQ_CREDIT_BUREAU_YEAR.mode()[0],inplace = True)
app_data.AMT_REQ_CREDIT_BUREAU_MON.fillna(app_data.AMT_REQ_CREDIT_BUREAU_MON.mode()[0],inplace = True)
app_data.AMT_REQ_CREDIT_BUREAU_WEEK.fillna(app_data.AMT_REQ_CREDIT_BUREAU_WEEK.mode()[0],inplace = True)
app_data.AMT_REQ_CREDIT_BUREAU_DAY.fillna(app_data.AMT_REQ_CREDIT_BUREAU_DAY.mode()[0],inplace = True)
app_data.AMT_REQ_CREDIT_BUREAU_HOUR.fillna(app_data.AMT_REQ_CREDIT_BUREAU_HOUR.mode()[0],inplace = True)
app_data.AMT_REQ_CREDIT_BUREAU_QRT.fillna(app_data.AMT_REQ_CREDIT_BUREAU_QRT.mode()[0],inplace = True)
```

- Again checking for null values

```
null_values(app_data).head(10)
```

```
NAME_TYPE_SUITE              0.42
OBS_30_CNT_SOCIAL_CIRCLE     0.33
DEF_30_CNT_SOCIAL_CIRCLE     0.33
OBS_60_CNT_SOCIAL_CIRCLE     0.33
DEF_60_CNT_SOCIAL_CIRCLE     0.33
AMT_GOODS_PRICE              0.09
AMT_ANNUITY                  0.00
CNT_FAM_MEMBERS              0.00
DAYS_LAST_PHONE_CHANGE       0.00
FLAG_DOCUMENT_5              0.00
dtype: float64
```

- Following the same process for the null values greater than 0.3%

```
#following the similar process for null values greater than 0.30%
app_data.NAME_TYPE_SUITE.fillna(app_data.NAME_TYPE_SUITE.mode()[0],inplace = True) #NAME_TYPE_SUITE
app_data.OBS_30_CNT_SOCIAL_CIRCLE.fillna(app_data.OBS_30_CNT_SOCIAL_CIRCLE.mode()[0],inplace = True) #OBS_30_CNT_SOCIAL_CIRCLE
app_data.DEF_30_CNT_SOCIAL_CIRCLE.fillna(app_data.DEF_30_CNT_SOCIAL_CIRCLE.mode()[0],inplace = True) #DEF_30_CNT_SOCIAL_CIRCLE
app_data.OBS_60_CNT_SOCIAL_CIRCLE.fillna(app_data.OBS_60_CNT_SOCIAL_CIRCLE.mode()[0],inplace = True) #OBS_60_CNT_SOCIAL_CIRCLE
app_data.DEF_60_CNT_SOCIAL_CIRCLE.fillna(app_data.DEF_60_CNT_SOCIAL_CIRCLE.mode()[0],inplace = True) #DEF_60_CNT_SOCIAL_CIRCLE
```

- Checking for null values

```
null_values(app_data).head(10)
```

```
AMT_GOODS_PRICE           0.09
AMT_ANNUITY               0.00
CNT_FAM_MEMBERS           0.00
DAYS_LAST_PHONE_CHANGE    0.00
FLAG_DOCUMENT_8           0.00
FLAG_DOCUMENT_7           0.00
FLAG_DOCUMENT_6           0.00
FLAG_DOCUMENT_5           0.00
FLAG_DOCUMENT_4           0.00
FLAG_DOCUMENT_3           0.00
dtype: float64
```

- Since all the major columns have been adjusted accordingly. I'll ignore the AMT_GOODS_PRICE column as it contains negligible number of null values.

- Now I'll be checking the columns using describe function.

```
app_data.describe()
```

	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE
	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000	3.072330e+05
	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573909	5.383962e+05
	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737315	3.694465e+05
	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e+04
	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05
	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05
	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05
	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06

From the above result it can be seen that the values in AMT_INCOME_TOTAL, AMT_CREDIT and AMT_GOODS_PRICE are very high in comparison to others.

```
app_data.describe()
```

IVE	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_REGISTRATION	DAYS_ID_PUBLISH	FLAG_MOBIL
000	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000
868	-16036.995067	63815.045904	-4986.120328	-2994.202373	0.999997
831	4363.988632	141275.766519	3522.886321	1509.450419	0.001803
290	-25229.000000	-17912.000000	-24672.000000	-7197.000000	0.000000
006	-19682.000000	-2760.000000	-7479.500000	-4299.000000	1.000000
850	-15750.000000	-1213.000000	-4504.000000	-3254.000000	1.000000
663	-12413.000000	-289.000000	-2010.000000	-1720.000000	1.000000
508	-7489.000000	365243.000000	0.000000	0.000000	1.000000

Also, it can be seen that columns like DAYS_BIRTH, DAYS_EMPLOYED, DAYS_REGISTRATION, DAY_ID_PUBLISH and DAYS_LAST_PHONE_CHANGE has many negative values. Therefore, after addressing the issues in those, I'll also convert and group DAYS_BIRTH values into various age under AGE GROUP and DAYS_EMPLOYED to YEARS_EMPLOYED for the convenience of analyzing data.

- At first, I'll be adding DAYS_BIRTH, DAYS_EMPLOYED, DAYS_REGISTRATION, DAY_ID_PUBLISH and DAYS_LAST_PHONE_CHANGE to a new variable.

```
# adding all the DAYS related column to a new variable
days_column = ["DAYS_BIRTH", "DAYS_EMPLOYED", "DAYS_REGISTRATION", "DAYS_ID_PUBLISH", "DAYS_LAST_PHONE_CHANGE"]

# checking if all the columns has been added
app_data[days_column].describe()
```

	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_REGISTRATION	DAYS_ID_PUBLISH	DAYS_LAST_PHONE_CHANGE
count	307511.000000	307511.000000	307511.000000	307511.000000	307510.000000
mean	-16036.995067	63815.045904	-4986.120328	-2994.202373	-962.858788
std	4363.988632	141275.766519	3522.886321	1509.450419	826.808487
min	-25229.000000	-17912.000000	-24672.000000	-7197.000000	-4292.000000
25%	-19682.000000	-2760.000000	-7479.500000	-4299.000000	-1570.000000
50%	-15750.000000	-1213.000000	-4504.000000	-3254.000000	-757.000000
75%	-12413.000000	-289.000000	-2010.000000	-1720.000000	-274.000000
max	-7489.000000	365243.000000	0.000000	0.000000	0.000000

- Now using the absolute function I'll be ensuring that the code handles values without considering their sign, focusing solely on their positive magnitude.

```
# using abs() function to correct the days values
app_data[days_column]= abs(app_data[days_column])

# checking if the convertion was successfull
app_data[days_column].describe()
```

	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_REGISTRATION	DAYS_ID_PUBLISH	DAYS_LAST_PHONE_CHANGE
count	307511.000000	307511.000000	307511.000000	307511.000000	307510.000000
mean	16036.995067	67724.742149	4986.120328	2994.202373	962.858788
std	4363.988632	139443.751806	3522.886321	1509.450419	826.808487
min	7489.000000	0.000000	0.000000	0.000000	0.000000
25%	12413.000000	933.000000	2010.000000	1720.000000	274.000000
50%	15750.000000	2219.000000	4504.000000	3254.000000	757.000000
75%	19682.000000	5707.000000	7479.500000	4299.000000	1570.000000
max	25229.000000	365243.000000	24672.000000	7197.000000	4292.000000

- Now using the binning method, I'll be converting the "DAYS_BIRTH" and "DAYS_EMPLOYED" columns to years and create two additional categorical columns, "AGE_GROUP" and "EMPLOYMENT_GROUP" for better understanding.

```
# creating categorical column from DAYS_BIRTH
app_data["AGE"] = app_data["DAYS_BIRTH"]/365
bins = [0,20,25,30,35,40,45,50,55,60,100]
slots = ["0-20","20-25","25-30","30-35","35-40","40-45","45-50","50-55","55-60","60 Above"]
app_data["AGE_GROUP"] = pd.cut(app_data["AGE"], bins=bins, labels=slots)
```

```
app_data["AGE_GROUP"].value_counts(normalize= True)*100
```

```
35-40       13.940314
40-45       13.464884
30-35       12.825557
60 Above    11.569993
45-50       11.425608
50-55       11.362846
55-60       10.770346
25-30       10.686447
20-25        3.954005
0-20         0.000000
Name: AGE_GROUP, dtype: float64
```

Using the above code, I have successfully created a new categorical column and named it as AGE_GROUP.

```
#creating categorical column from "DAYS_EMPLOYED
app_data["YEARS_EMPLOYED"] = app_data["DAYS_EMPLOYED"]/365
bins = [0,5,10,15,20,25,30,50]
slots = ["0-5","5-10","10-15","15-20","20-25","25-30","30 Above"]
app_data["EMPLOYEMENT_YEARS"] = pd.cut(app_data["YEARS_EMPLOYED"], bins=bins, labels=slots)
```

```
app_data["EMPLOYEMENT_YEARS"].value_counts(normalize= True)*100
```

```
0-5         54.061911
5-10        25.729074
10-15       10.926289
15-20        4.302854
20-25        2.476054
25-30        1.311996
30 Above     1.191822
Name: EMPLOYEMENT_YEARS, dtype: float64
```

Using the above code, I have successfully created a new column and named it EMPLOYEMENT_YEARS.

- Next, I'll be checking if the columns are added to the dataframe or not using app_data.info()

```
 64   FLAG_DOCUMENT_21              307511 non-null  int64
 65   AMT_REQ_CREDIT_BUREAU_HOUR    307511 non-null  float64
 66   AMT_REQ_CREDIT_BUREAU_DAY     307511 non-null  float64
 67   AMT_REQ_CREDIT_BUREAU_WEEK    307511 non-null  float64
 68   AMT_REQ_CREDIT_BUREAU_MON     307511 non-null  float64
 69   AMT_REQ_CREDIT_BUREAU_QRT     307511 non-null  float64
 70   AMT_REQ_CREDIT_BUREAU_YEAR    307511 non-null  float64
 71   AGE                           307511 non-null  float64
 72   AGE_GROUP                     307511 non-null  category
 73   YEARS_EMPLOYED                307511 non-null  float64
 74   EMPLOYEMENT_YEARS             252135 non-null  category
dtypes: category(2), float64(20), int64(41), object(12)
memory usage: 171.9+ MB
```

- Next, I'll be doing the outlier analysis. Outlier detection is crucial in data science and analysis. Removing outliers can enhance model performance, and can also provide valuable insights for analysis.
- I'll start the process by identifying the outliers

```
# identifying outliers
app_data.describe()
```

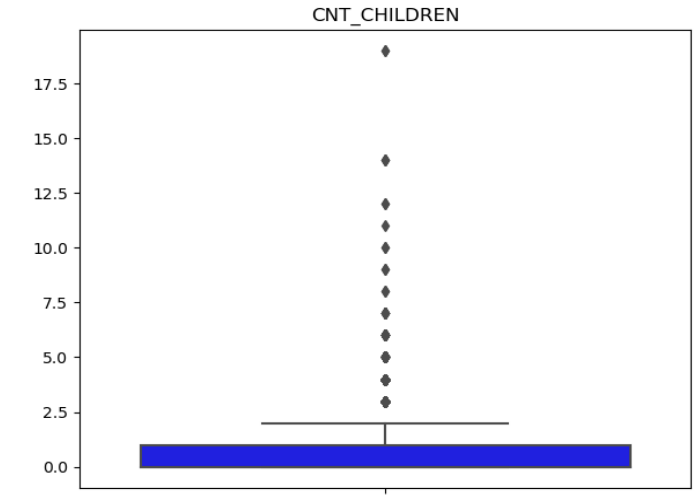
	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIV
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000	3.072330e+05	307511.0000
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573909	5.383962e+05	0.0208
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737315	3.694465e+05	0.0138
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e+04	0.0002
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05	0.0100
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05	0.0188
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	0.0286
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	0.0725

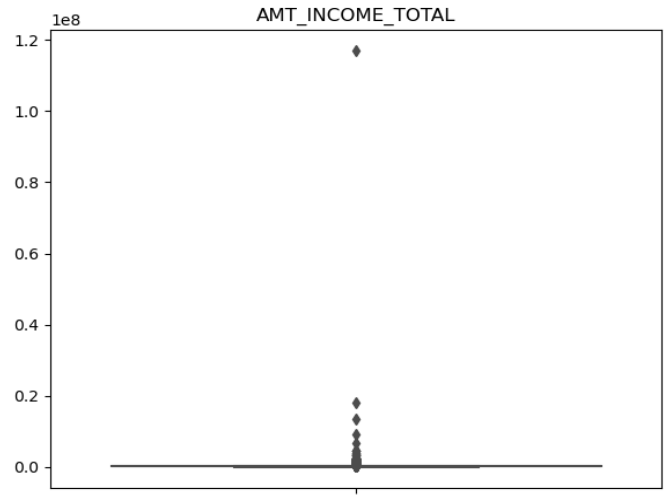
Using the describe function, it can be seen that there happens to be a huge difference between 75% value and maximum value. Therefore, I'll be separating them into a new variable.

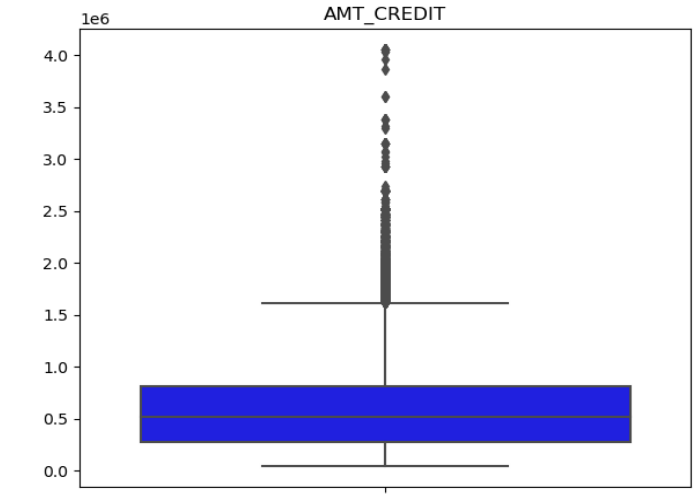
```
# adding the outlier columns to a new variable
outliers = ["CNT_CHILDREN","AMT_INCOME_TOTAL", "AMT_CREDIT", "AMT_ANNUITY", "AMT_GOODS_PRICE",
            "DAYS_BIRTH", "DAYS_EMPLOYED", "DAYS_REGISTRATION"]
```

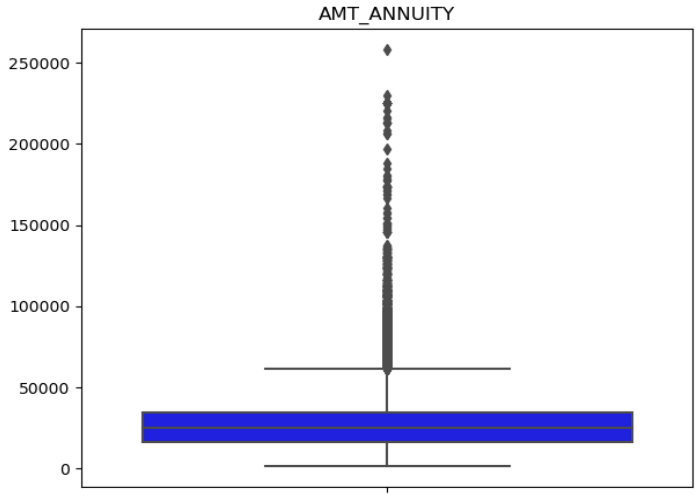
- Now I'll be checking for outliers for the columns present in outliers variable.

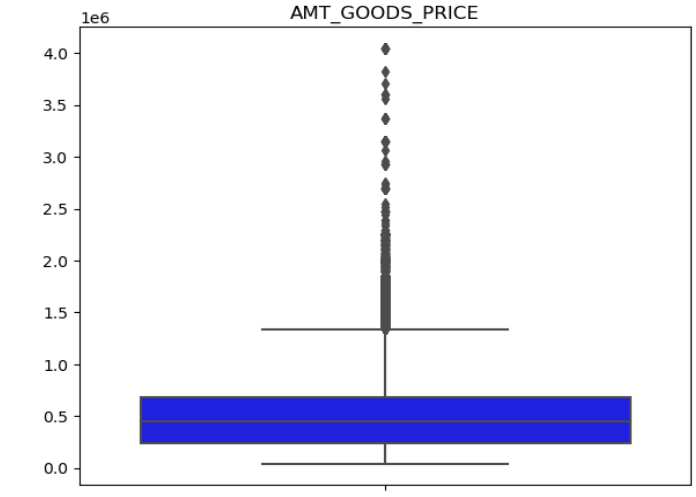
```
# checking for outliers
plt.figure(figsize=[15,25])
for i,j in itertools.zip_longest(outliers, range(len(outliers))):
    plt.subplot(4,2,j+1)
    sns.boxplot(y = app_data[i], orient = "h", color = "blue")
    #plt.yticks(fontsize=8)
    plt.xlabel("")
    plt.ylabel("")
    plt.title(i)
```

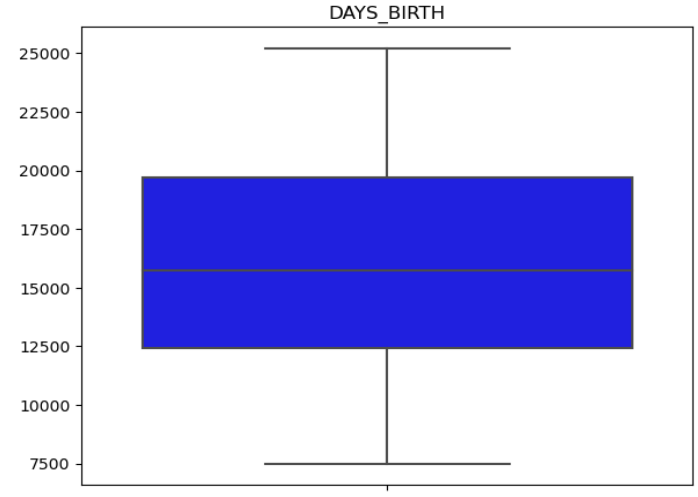

CNT_CHILDREN
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0


AMT_INCOME_TOTAL
1e8
1.2
1.0
0.8
0.6
0.4
0.2
0.0

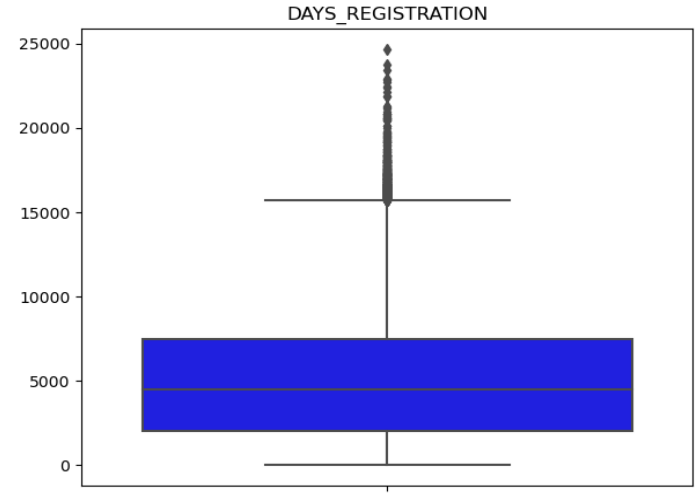

AMT_CREDIT
1e6
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0


AMT_ANNUITY
250000
200000
150000
100000
50000
0


AMT_GOODS_PRICE
1e6
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0


DAYS_BIRTH
25000
22500
20000
17500
15000
12500
10000
7500

DAYS_EMPLOYED
350000
300000
250000
200000
150000
100000
50000
0


DAYS_REGISTRATION
25000
20000
15000
10000
5000
0

So, from the above plot the following can be concluded

- The "AMT_INCOME_TOTAL" column contains a significant number of outliers, suggesting that some loan applicants have substantially higher incomes compared to others.
- DAYS_EMPLOYED has outlier values around 350000 which is quite impossible and hence this has to be incorrect entry.
- Some of the columns is showing an adequate number of outliers CNT_CHILDREN, AMT_ANNUITY, AMT_CREDIT and AMT_GOODS_PRICE and DAYS_REGISTRATION
- There are no outliers in DAYS_BIRTH column and hence the data is reliable.

- Next, I'll be converting some of the object-oriented columns into categorical columns. For that, I'll be checking the columns available in the dataframe app_data

```
app_data.columns
```

```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE',
       'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS',
       'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH',
       'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'FLAG_MOBIL',
       'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE',
       'FLAG_EMAIL', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS',
       'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY',
       'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
       'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
       'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
       'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY',
       'ORGANIZATION_TYPE', 'OBS_30_CNT_SOCIAL_CIRCLE',
       'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE',
       'DEF_60_CNT_SOCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2',
       'FLAG_DOCUMENT_3', 'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5',
       'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8',
       'FLAG_DOCUMENT_9', 'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11',
       'FLAG_DOCUMENT_12', 'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14',
       'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17',
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
       'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR', 'AGE', 'AGE_GROUP', 'YEARS_EMPLOYED',
       'EMPLOYEMENT_YEARS'],
      dtype='object')
```

- Then, I'll be creating a new variable named cat_col and store the columns, that needed to be transformed, into that variable. And then change the columns into categorical columns.

```
# taking the required columns to convert into categorical columns
cat_col = ['NAME_CONTRACT_TYPE','CODE_GENDER','NAME_TYPE_SUITE','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE',
                        'NAME_FAMILY_STATUS','NAME_HOUSING_TYPE','OCCUPATION_TYPE','WEEKDAY_APPR_PROCESS_START',
                        'ORGANIZATION_TYPE','FLAG_OWN_REALTY','LIVE_CITY_NOT_WORK_CITY',
                        'REG_CITY_NOT_LIVE_CITY','REG_CITY_NOT_WORK_CITY','REG_REGION_NOT_WORK_REGION',
                        'LIVE_REGION_NOT_WORK_REGION','REGION_RATING_CLIENT','WEEKDAY_APPR_PROCESS_START',
                        'REGION_RATING_CLIENT_W_CITY','CNT_CHILDREN','CNT_FAM_MEMBERS']

for col in cat_col:
    app_data[col] = pd.Categorical(app_data[col])
```

Therefore, using info() function, I'll check if the columns have been changed into categorical ones.

- From the above process, it has been observed that AMT_INCOME_TOTAL, AMT_CREDIT and AMT_GOODS_PRICE have very high values. Therefore, I'll be creating new categorical columns and group them according to their range.
- Creating a categorical new column for AMT_CREDIT and naming it as AMT_CREDIT_RANGE

```
# creating categorical columns for AMT_CREDIT
app_data['AMT_CREDIT']=app_data['AMT_CREDIT']/100000
bins = [0,1,2,3,4,5,6,7,8,9,10,100]
slots = ['0-1L','1L-2L', '2L-3L','3L-4L','4L-5L','5L-6L','6L-7L','7L-8L','8L-9L','9L-10L','10L Above']
app_data['AMT_CREDIT_RANGE']=pd.cut(app_data['AMT_CREDIT'],bins=bins,labels=slots)
```

```
round((app_data["AMT_CREDIT_RANGE"].value_counts(normalize = True)*100),2)
```

```
2L-3L        17.82
10L Above    16.25
5L-6L        11.13
4L-5L        10.42
1L-2L         9.80
3L-4L         8.56
6L-7L         7.82
8L-9L         7.09
7L-8L         6.24
9L-10L        2.90
0-1L          1.95
Name: AMT_CREDIT_RANGE, dtype: float64
```

- Creating a categorical new column for AMT_GOODS_PRICE and naming it as AMT_GOODS_PRICE_RANGE

```
# creating categorical column for AMT_GOODS_PRICE
app_data['AMT_GOODS_PRICE']=app_data['AMT_GOODS_PRICE']/100000

bins = [0,1,2,3,4,5,6,7,8,9,10,100]
slots = ['0-1L','1L-2L', '2L-3L','3L-4L','4L-5L','5L-6L','6L-7L','7L-8L','8L-9L','9L-10L','10L Above']

app_data['AMT_GOODS_PRICE_RANGE']=pd.cut(app_data['AMT_GOODS_PRICE'],bins=bins,labels=slots)
```

```
round((app_data["AMT_GOODS_PRICE_RANGE"].value_counts(normalize = True)*100),2)
```

```
2L-3L        20.43
4L-5L        18.54
6L-7L        13.03
10L Above    11.11
1L-2L        10.73
8L-9L         6.99
3L-4L         6.91
5L-6L         4.27
0-1L          2.83
7L-8L         2.64
9L-10L        2.53
Name: AMT_GOODS_PRICE_RANGE, dtype: float64
```

- Creating a categorical new column for AMT_INCOME_TOTAL and naming it as AMT_INCOME_TOTAL_RANGE

```
# creating categorical column for AMT_INCOME_TOTAL

app_data['AMT_INCOME_TOTAL']=app_data['AMT_INCOME_TOTAL']/100000

bins = [0,1,2,3,4,5,6,7,8,9,10,11]
slot = ['0-1L','1L-2L', '2L-3L','3L-4L','4L-5L','5L-6L','6L-7L','7L-8L','8L-9L','9L-10L','10L Above']

app_data['AMT_INCOME_TOTAL_RANGE']=pd.cut(app_data['AMT_INCOME_TOTAL'],bins,labels=slot)
```

```
round((app_data["AMT_INCOME_TOTAL_RANGE"].value_counts(normalize = True)*100),2)
```

```
1L-2L        50.73
2L-3L        21.21
0-1L         20.73
3L-4L         4.78
4L-5L         1.74
5L-6L         0.36
6L-7L         0.28
8L-9L         0.10
7L-8L         0.05
9L-10L        0.01
10L Above     0.01
Name: AMT_INCOME_TOTAL_RANGE, dtype: float64
```

Therefore, the process of data cleaning and data manipulation is complete with application data file. Next, I'll be doing the same process with the previous application file and after that I'll be moving towards analysis phase.

- Importing previous_application.csv to a new dataframe and named it as prev_app

```
# importing previous_application.csv
prev_app = pd.read_csv("previous_application.csv")
```

- I'll be using shape attribute to determine the total number of rows and columns

```
# checking total number of rows and columns
prev_app.shape
```

```
(1670214, 37)
```

Therefore, there are 1670214 rows and 37 columns in total.

- Next I'll be checking the type of entries present in the columns using head function

```
# checking the details present inside the columns
prev_app.head()
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN

- Then I'll be checking the data types of each and every columns using info() function

```
# getting all the details about the dataframe and the data types present
prev_app.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   SK_ID_PREV                   1670214 non-null  int64
 1   SK_ID_CURR                   1670214 non-null  int64
 2   NAME_CONTRACT_TYPE           1670214 non-null  object
 3   AMT_ANNUITY                  1297979 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_DOWN_PAYMENT             774370 non-null   float64
 7   AMT_GOODS_PRICE              1284699 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object
 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64
 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object
 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64
 12  RATE_DOWN_PAYMENT            774370 non-null   float64
 13  RATE_INTEREST_PRIMARY        5951 non-null     float64
 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64
 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object
 16  NAME_CONTRACT_STATUS         1670214 non-null  object
 17  DAYS_DECISION                1670214 non-null  int64
 18  NAME_PAYMENT_TYPE            1670214 non-null  object
 19  CODE_REJECT_REASON           1670214 non-null  object
 20  NAME_TYPE_SUITE              849809 non-null   object
 21  NAME_CLIENT_TYPE             1670214 non-null  object
 22  NAME_GOODS_CATEGORY          1670214 non-null  object
```

```
 23  NAME_PORTFOLIO              1670214 non-null  object
 24  NAME_PRODUCT_TYPE           1670214 non-null  object
 25  CHANNEL_TYPE                1670214 non-null  object
 26  SELLERPLACE_AREA            1670214 non-null  int64
 27  NAME_SELLER_INDUSTRY        1670214 non-null  object
 28  CNT_PAYMENT                 1297984 non-null  float64
 29  NAME_YIELD_GROUP            1670214 non-null  object
 30  PRODUCT_COMBINATION         1669868 non-null  object
 31  DAYS_FIRST_DRAWING          997149 non-null   float64
 32  DAYS_FIRST_DUE              997149 non-null   float64
 33  DAYS_LAST_DUE_1ST_VERSION   997149 non-null   float64
 34  DAYS_LAST_DUE               997149 non-null   float64
 35  DAYS_TERMINATION            997149 non-null   float64
 36  NFLAG_INSURED_ON_APPROVAL   997149 non-null   float64
dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
```

- Next I'll be checking for any null values present in the columns

```
# checking the null values present in the columns
null_values(prev_app)
```

```
RATE_INTEREST_PRIVILEGED       99.64
RATE_INTEREST_PRIMARY          99.64
AMT_DOWN_PAYMENT               53.64
RATE_DOWN_PAYMENT              53.64
NAME_TYPE_SUITE                49.12
NFLAG_INSURED_ON_APPROVAL      40.30
DAYS_TERMINATION               40.30
DAYS_LAST_DUE                  40.30
DAYS_LAST_DUE_1ST_VERSION      40.30
DAYS_FIRST_DUE                 40.30
DAYS_FIRST_DRAWING             40.30
AMT_GOODS_PRICE                23.08
AMT_ANNUITY                    22.29
CNT_PAYMENT                    22.29
PRODUCT_COMBINATION             0.02
```

- Creating a new variable to store columns with null values greater than 50%

```
# storing null value columns with more than 50% null values
null_50 = null_values(prev_app)[null_values(prev_app)>50]
```

- Dropping the columns after doing a check on the variable

```
null_50
```

```
RATE_INTEREST_PRIVILEGED    99.64
RATE_INTEREST_PRIMARY       99.64
AMT_DOWN_PAYMENT            53.64
RATE_DOWN_PAYMENT           53.64
dtype: float64
```

```
# dropping the columns
prev_app.drop(columns = null_50.index, inplace = True)
```

- Now creating a new variable that will be storing columns with null value more than 20%

```
# storing columns with more 20% null values
null_20 = null_values(prev_app)[null_values(prev_app)>20]
```

```
null_20
```

```
NAME_TYPE_SUITE              49.12
DAYS_FIRST_DRAWING           40.30
DAYS_TERMINATION             40.30
DAYS_LAST_DUE                40.30
DAYS_LAST_DUE_1ST_VERSION    40.30
DAYS_FIRST_DUE               40.30
NFLAG_INSURED_ON_APPROVAL    40.30
AMT_GOODS_PRICE              23.08
AMT_ANNUITY                  22.29
CNT_PAYMENT                  22.29
dtype: float64
```

- Checking the content details of all the columns mentioned above before taking any action.

```
# checking the contents of the columns before taking action
prev_app[null_20.index]
```

	NAME_TYPE_SUITE	DAYS_FIRST_DRAWING	DAYS_TERMINATION	DAYS_LAST_DUE	DAYS_LAST_DUE_1ST_VERSION	DAYS_FIRST_DUE	NF
0	NaN	365243.0	-37.0	-42.0	300.0	-42.0	
1	Unaccompanied	365243.0	365243.0	365243.0	916.0	-134.0	
2	Spouse, partner	365243.0	365243.0	365243.0	59.0	-271.0	
3	NaN	365243.0	-177.0	-182.0	-152.0	-482.0	
4	NaN	NaN	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	
1670209	NaN	365243.0	-351.0	-358.0	362.0	-508.0	
1670210	Unaccompanied	365243.0	-1297.0	-1304.0	-1274.0	-1604.0	
1670211	Spouse, partner	365243.0	-1181.0	-1187.0	-1187.0	-1457.0	
1670212	Family	365243.0	-817.0	-825.0	-825.0	-1155.0	
1670213	Family	365243.0	-423.0	-443.0	247.0	-1163.0	

- Since NAME_TYPE_SUIT column happens to be a categorical column, therefore, without removing it I will fill the blank an null cells with "Unknown" value.

```
# Filling NAME_TYPE_SUIT column blank values with "Unknown"
prev_app["NAME_TYPE_SUITE"] = prev_app["NAME_TYPE_SUITE"].fillna("Unknown")
```

- Checking the rest of the columns left for correction in null_20

```
# checking the rest of the columns
prev_app[null_20.index].describe()
```

	DAYS_FIRST_DRAWING	DAYS_TERMINATION	DAYS_LAST_DUE	DAYS_LAST_DUE_1ST_VERSION	DAYS_FIRST_DUE	NFLAG_
count	997149.000000	997149.000000	997149.000000	997149.000000	997149.000000	
mean	342209.855039	81992.343838	76582.403064	33767.774054	13826.269337	
std	88916.115834	153303.516729	149647.415123	106857.034789	72444.869708	
min	-2922.000000	-2874.000000	-2889.000000	-2801.000000	-2892.000000	
25%	365243.000000	-1270.000000	-1314.000000	-1242.000000	-1628.000000	
50%	365243.000000	-499.000000	-537.000000	-361.000000	-831.000000	
75%	365243.000000	-44.000000	-74.000000	129.000000	-411.000000	
max	365243.000000	365243.000000	365243.000000	365243.000000	365243.000000	

- Next, I'll be transforming the columns with negative values to positive ones using absolute function. And for that, I'll be creating a new variable and storing the columns with negative values in it.

```
# creating new variable to store negative value columns in it
negative_col = ['DAYS_FIRST_DRAWING','DAYS_TERMINATION', 'DAYS_LAST_DUE', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION', 'DAYS_DE

# checking before doing conversion
prev_app[negative_col].describe()
```

	DAYS_FIRST_DRAWING	DAYS_TERMINATION	DAYS_LAST_DUE	DAYS_FIRST_DUE	DAYS_LAST_DUE_1ST_VERSION	DAYS_DECISION
count	997149.000000	997149.000000	997149.000000	997149.000000	997149.000000	1.670214e+06
mean	342209.855039	81992.343838	76582.403064	13826.269337	33767.774054	-8.806797e+02
std	88916.115834	153303.516729	149647.415123	72444.869708	106857.034789	7.790997e+02
min	-2922.000000	-2874.000000	-2889.000000	-2892.000000	-2801.000000	-2.922000e+03
25%	365243.000000	-1270.000000	-1314.000000	-1628.000000	-1242.000000	-1.300000e+03
50%	365243.000000	-499.000000	-537.000000	-831.000000	-361.000000	-5.810000e+02
75%	365243.000000	-44.000000	-74.000000	-411.000000	129.000000	-2.800000e+02
max	365243.000000	365243.000000	365243.000000	365243.000000	365243.000000	-1.000000e+00

Here, DAYS_DECISION has been observed directly from prev_app dataframe. Since, it contained details regarding number of days in negative form, therefore it was necessary to convert the negative values into positive one.

- Converting the columns from negative to positive ones using the absolute function and then checking the contents of the columns using describe function

```
# converting negative columns to positive columns
prev_app[negative_col] = abs(prev_app[negative_col])
```

```
# checking after conversion
prev_app[negative_col].describe()
```

	DAYS_FIRST_DRAWING	DAYS_TERMINATION	DAYS_LAST_DUE	DAYS_FIRST_DUE	DAYS_LAST_DUE_1ST_VERSION	DAYS_DECISION
count	997149.000000	997149.000000	997149.000000	997149.000000	997149.000000	1.670214e+06
mean	342340.056543	83505.775017	78152.730207	15949.224065	35163.363265	8.806797e+02
std	88413.495220	152484.418802	148833.342466	72007.270877	106405.950190	7.790997e+02
min	2.000000	2.000000	2.000000	2.000000	0.000000	1.000000e+00
25%	365243.000000	447.000000	455.000000	475.000000	257.000000	2.800000e+02
50%	365243.000000	1171.000000	1155.000000	921.000000	741.000000	5.810000e+02
75%	365243.000000	2501.000000	2418.000000	1825.000000	1735.000000	1.300000e+03
max	365243.000000	365243.000000	365243.000000	365243.000000	365243.000000	2.922000e+03

- Now I'll be grouping the values of DAYS_DECISION into a new column as the current values are very high.

```
# grouping DAYS_DECISION into a new column for the ease of analysis
bins = [0,1*365,2*365,3*365,4*365,5*365,6*365,7*365,10*365]
slots = ["1","2","3","4","5","6","7","7 above"]
prev_app['YEARLY_DECISION'] = pd.cut(prev_app['DAYS_DECISION'],bins,labels=slots)
```

```
prev_app['YEARLY_DECISION'].value_counts(normalize=True)*100
```

```
1          34.351287
2          23.056806
3          12.855598
4           7.883181
5           6.128556
7           5.813806
7 above     5.060729
6           4.850037
Name: YEARLY_DECISION, dtype: float64
```

From the above result it can be concluded that most of the people have applied for a new loan within 1 year of previous loan decision.

- Checking the null value columns present in the dataframe

```
null_values(prev_app)
```

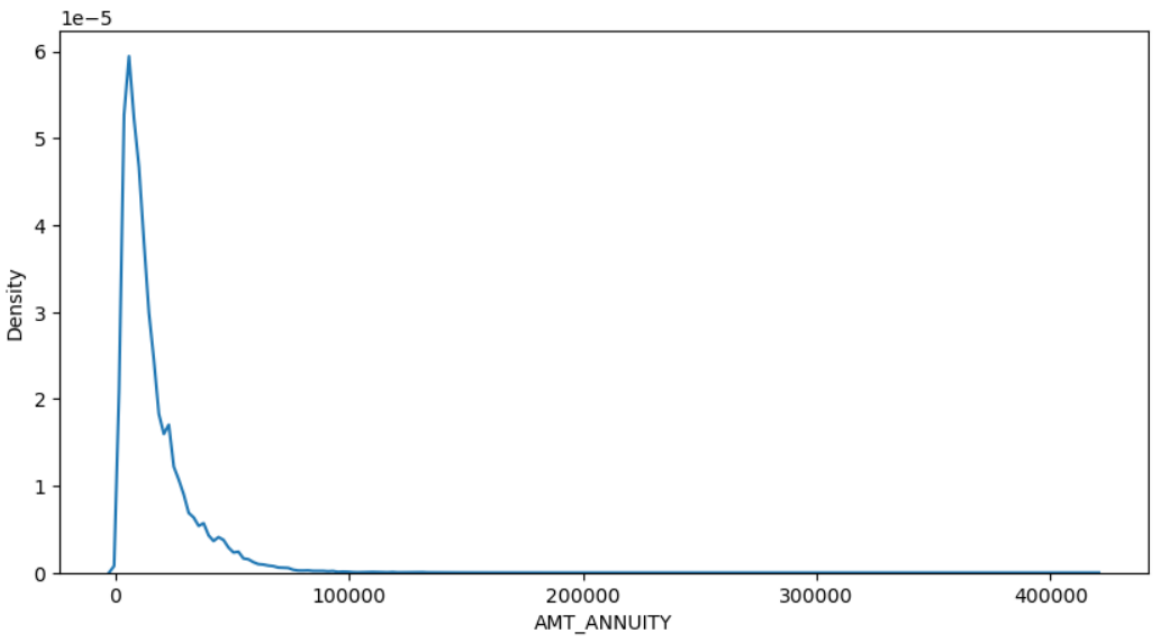
```
DAYS_FIRST_DUE                 40.30
NFLAG_INSURED_ON_APPROVAL      40.30
DAYS_TERMINATION               40.30
DAYS_LAST_DUE                  40.30
DAYS_LAST_DUE_1ST_VERSION      40.30
DAYS_FIRST_DRAWING             40.30
AMT_GOODS_PRICE                23.08
AMT_ANNUITY                    22.29
CNT_PAYMENT                    22.29
PRODUCT_COMBINATION             0.02
AMT_CREDIT                      0.00
```

The provided columns 'DAYS_FIRST_DUE', 'DAYS_TERMINATION', 'DAYS_FIRST_DRAWING', 'DAYS_LAST_DUE_1ST_VERSION', and 'DAYS_LAST_DUE' contain missing values. These columns represent counts of

days, and it is important to note that the null values in these columns should be retained as they are. Preserving the null values is necessary to maintain the integrity and accuracy of the data.

- In order to handle null values in continuous variables, I followed a two-step approach.
  - ✓ First, I plotted the distribution of the columns to analyze their shape and characteristics. If the distribution was found to be skewed, I chose to impute the null values using the median. The median is a measure of central tendency that is less sensitive to outliers, making it a suitable choice in skewed distributions.
  - ✓ On the other hand, if the distribution pattern appeared to be preserved, indicating a relatively symmetrical distribution, I opted to impute the null values using the mode. The mode represents the most commonly occurring value in the distribution and is useful in maintaining the overall pattern. By considering the distribution's shape and characteristics, I employed an appropriate imputation strategy to handle the null values in the continuous variables effectively.
- To understand the distribution of AMT_ANNUITY, I plotted a kdeplot

```
#plotting a kdeplot to understand the distribution of "AMT_ANNUITY"
plt.figure(figsize=(10,5))
sns.kdeplot(prev_app['AMT_ANNUITY'])
plt.show()
```

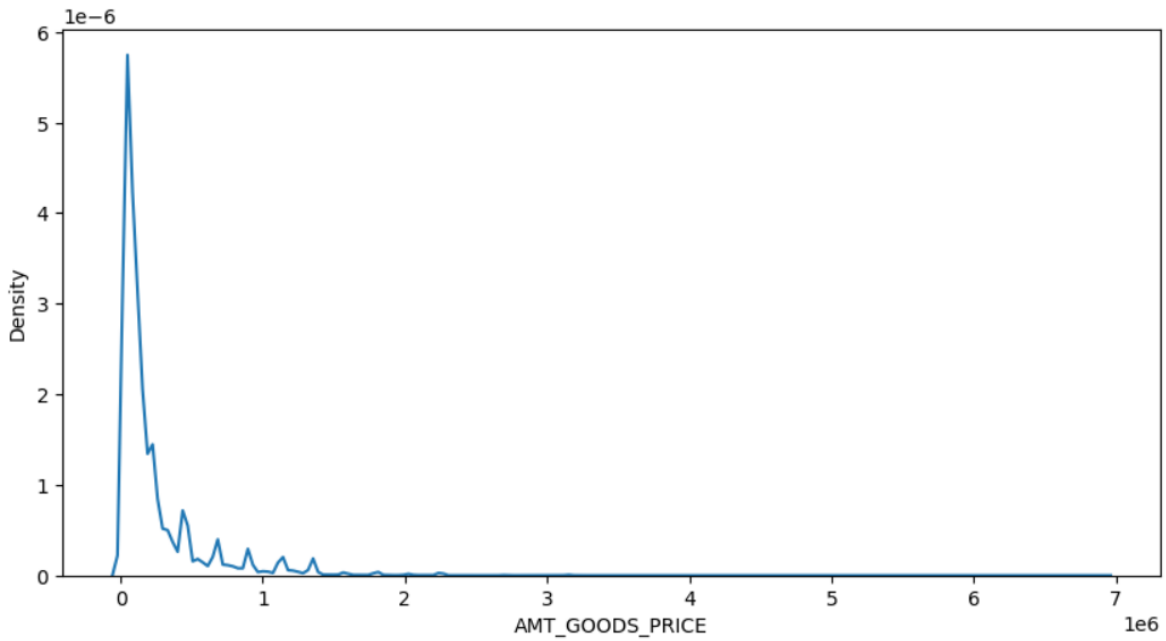


The distribution of the variable shows a single peak on the left side, suggesting the presence of outliers in the data. Due to the presence of these outliers,

imputing the missing values with the mean would not be an appropriate approach as it can be heavily influenced by the outliers and may not represent the central tendency accurately. Instead, imputing the missing values with the median is a better choice. The median is less sensitive to outliers and provides a more robust estimate of the central value. By imputing with the median, we can better account for the skewness and preserve the integrity of the data.

- Using the similar process that I did earlier for AMT_ANNUITY, I'll check the appropriate process for AMT_GOODS_PRICE as well. Hence, to understand the distribution, I'll plot a kde plot

```
#plotting a kdeplot to understand the distribution of "AMT_GOODS_PRICE"
plt.figure(figsize=(10,5))
sns.kdeplot(prev_app['AMT_GOODS_PRICE'])
plt.show()
```



When analyzing a distribution, I've encountered multiple peaks or clusters. To examine the impact of imputing missing values, I can utilize different measures such as the mode, mean, and median. By imputing missing values using these measures, I can assess whether the overall shape and characteristics of the distribution remain relatively unchanged.

By comparing the original distribution with the imputed versions using mode, mean, and median, I'll be able to evaluate the extent to which the peaks or clusters are preserved or altered. This analysis provides insights into the impact of imputation methods on the overall distribution and helps us understand the potential consequences of using different imputation techniques.

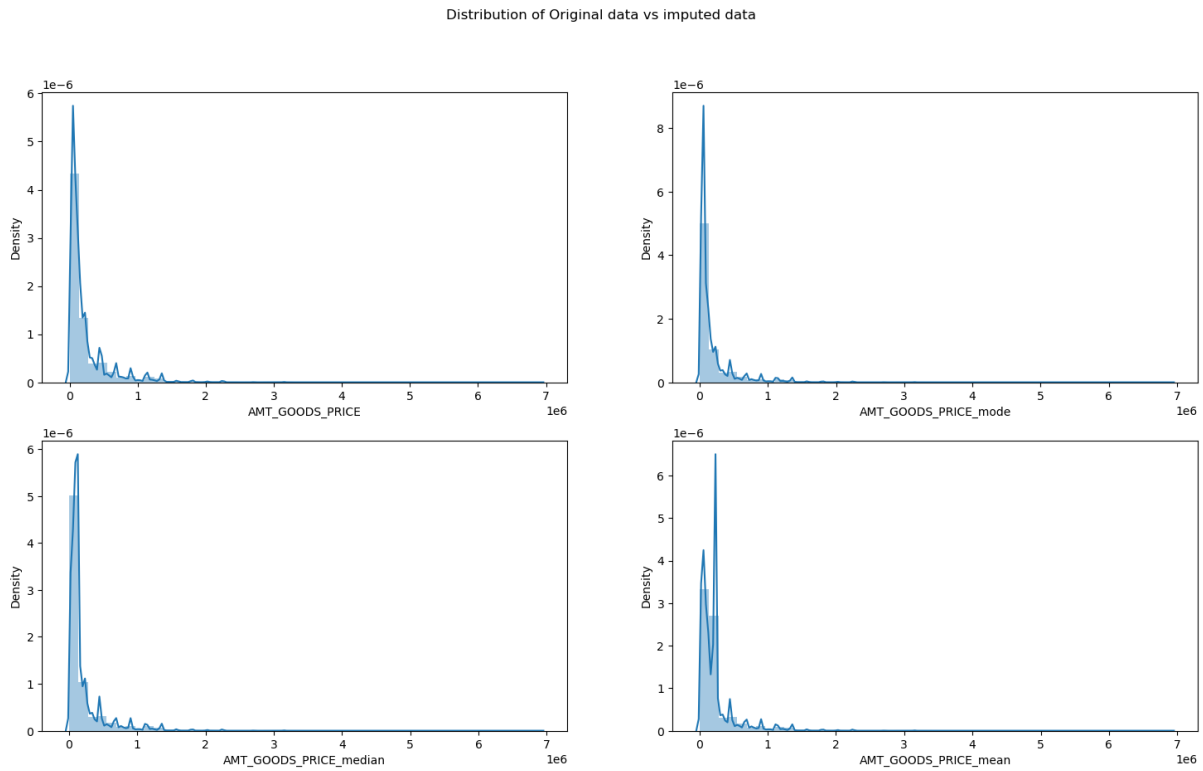
- By creating a new dataframe, it'll be easy to compare and analyze the impact of different imputation techniques on the "AMT_GOODS_PRICE"

```
# creating a new dataframe "AMT_GOODS_PRICE" with columns imputed with mode, median and mean

statsDF = pd.DataFrame()
statsDF['AMT_GOODS_PRICE_mode'] = prev_app['AMT_GOODS_PRICE'].fillna(prev_app['AMT_GOODS_PRICE'].mode()[0])
statsDF['AMT_GOODS_PRICE_median'] = prev_app['AMT_GOODS_PRICE'].fillna(prev_app['AMT_GOODS_PRICE'].median())
statsDF['AMT_GOODS_PRICE_mean'] = prev_app['AMT_GOODS_PRICE'].fillna(prev_app['AMT_GOODS_PRICE'].mean())

cols = ['AMT_GOODS_PRICE_mode', 'AMT_GOODS_PRICE_median','AMT_GOODS_PRICE_mean']

plt.figure(figsize=(18,10))
plt.suptitle('Distribution of Original data vs imputed data')
plt.subplot(221)
sns.distplot(prev_app['AMT_GOODS_PRICE'][pd.notnull(prev_app['AMT_GOODS_PRICE'])]);
for i in enumerate(cols):
    plt.subplot(2,2,i[0]+2)
    sns.distplot(statsDF[i[1]])
```



From the above distribution it can be observed that AMT_GOODS_PRICE_mode is much closer to the original distribution. Therefore, imputing the null values with mode.

- The 'CNT_PAYMENT' column likely represents the number of payments or installments for a loan. When there is a missing value in this column, it suggests that there is no information available regarding the number of payments associated with those specific loans. By imputing these missing values with 0, we are essentially assuming that these loans were not started or no payments were made. Therefore, when examining the

'NAME_CONTRACT_STATUS' values associated with these imputed records, we can infer that most of them would reflect a status indicating that the loan was not started or remains inactive.

```
# initializing values count for NAME_CONTRACT_STATUS categories where CNT_PAYMENT have null values.
prev_app.loc[prev_app['CNT_PAYMENT'].isnull(),'NAME_CONTRACT_STATUS'].value_counts()
```

```
Canceled        305805
Refused          40897
Unused offer     25524
Approved             4
Name: NAME_CONTRACT_STATUS, dtype: int64
```

```
#imputing null values with 0
prev_app['CNT_PAYMENT'].fillna(0,inplace = True)
```

In simple terms, imputing 'CNT_PAYMENT' with 0 suggests that the loans represented by these missing values were not initiated, and the 'NAME_CONTRACT_STATUS' values further support this assumption by indicating that the majority of these loans have a status indicating non-started or inactive status.

- Next, I'll be converting some of the object oriented columns to categorical ones

```
#Converting object oriented columns from Object to categorical

cat_col_2 = ['CHANNEL_TYPE','NAME_CONTRACT_STATUS','NAME_PAYMENT_TYPE','NAME_CASH_LOAN_PURPOSE',
             'CODE_REJECT_REASON','NAME_CLIENT_TYPE','NAME_GOODS_CATEGORY',
              'NAME_PORTFOLIO', 'NAME_PRODUCT_TYPE','NAME_SELLER_INDUSTRY',
              'NAME_YIELD_GROUP','PRODUCT_COMBINATION','NAME_CONTRACT_TYPE']

for col in cat_col_2:
    prev_app[col] =pd.Categorical(prev_app[col])
```

```
prev_app.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 34 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   SK_ID_PREV                   1670214 non-null  int64
 1   SK_ID_CURR                   1670214 non-null  int64
 2   NAME_CONTRACT_TYPE           1670214 non-null  category
 3   AMT_ANNUITY                  1670214 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_GOODS_PRICE              1670214 non-null  float64
 7   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object
 8   HOUR_APPR_PROCESS_START      1670214 non-null  int64
 9   FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object
 10  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64
 11  NAME_CASH_LOAN_PURPOSE       1670214 non-null  category
 12  NAME_CONTRACT_STATUS         1670214 non-null  category
 13  DAYS_DECISION                1670214 non-null  int64
 14  NAME_PAYMENT_TYPE            1670214 non-null  category
 15  CODE_REJECT_REASON           1670214 non-null  category
 16  NAME_TYPE_SUITE              1670214 non-null  object
 17  NAME_CLIENT_TYPE             1670214 non-null  category
 18  NAME_GOODS_CATEGORY          1670214 non-null  category
 19  NAME_PORTFOLIO               1670214 non-null  category
 20  NAME_PRODUCT_TYPE            1670214 non-null  category
 21  CHANNEL_TYPE                 1670214 non-null  category
 22  SELLERPLACE_AREA             1670214 non-null  int64
 23  NAME_SELLER_INDUSTRY         1670214 non-null  category
 24  CNT_PAYMENT                  1670214 non-null  float64
 25  NAME_YIELD_GROUP             1670214 non-null  category
 26  PRODUCT_COMBINATION          1669868 non-null  category
 27  DAYS_FIRST_DRAWING           997149 non-null   float64
 28  DAYS_FIRST_DUE               997149 non-null   float64
 29  DAYS_LAST_DUE_1ST_VERSION    997149 non-null   float64
 30  DAYS_LAST_DUE                997149 non-null   float64
 31  DAYS_TERMINATION             997149 non-null   float64
 32  NFLAG_INSURED_ON_APPROVAL    997149 non-null   float64
 33  YEARLY_DECISION              1670214 non-null  category
dtypes: category(14), float64(11), int64(6), object(3)
memory usage: 277.2+ MB
```

- The next process involves finding the outliers for prev_app dataframe. Therefore, before diving into finding outliers, first I'll check the details of the columns using describe function.

```
# checking for outliers
prev_app.describe()
```

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE	HOUR_APPR_PROCESS_START
count	1.670214e+06	1.670214e+06	1.670214e+06	1.670214e+06	1.670213e+06	1.670214e+06	1.670214e+06
mean	1.923089e+06	2.783572e+05	1.490651e+04	1.752339e+05	1.961140e+05	1.856429e+05	1.248418e+01
std	5.325980e+05	1.028148e+05	1.317751e+04	2.927798e+05	3.185746e+05	2.871413e+05	3.334028e+00
min	1.000001e+06	1.000010e+05	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	1.461857e+06	1.893290e+05	7.547096e+03	1.872000e+04	2.416050e+04	4.500000e+04	1.000000e+01
50%	1.923110e+06	2.787145e+05	1.125000e+04	7.104600e+04	8.054100e+04	7.105050e+04	1.200000e+01
75%	2.384280e+06	3.675140e+05	1.682403e+04	1.803600e+05	2.164185e+05	1.804050e+05	1.500000e+01
max	2.845382e+06	4.562550e+05	4.180581e+05	6.905160e+06	6.905160e+06	6.905160e+06	2.300000e+01

By analyzing the descriptive statistics of a dataset using the describe() function, we can identify columns that exhibit a significant difference between the maximum value and the 75th percentile. Additionally, we can identify columns where it seems illogical or impractical for the maximum value to be exceptionally high. The relevant information can be summarized as follows:

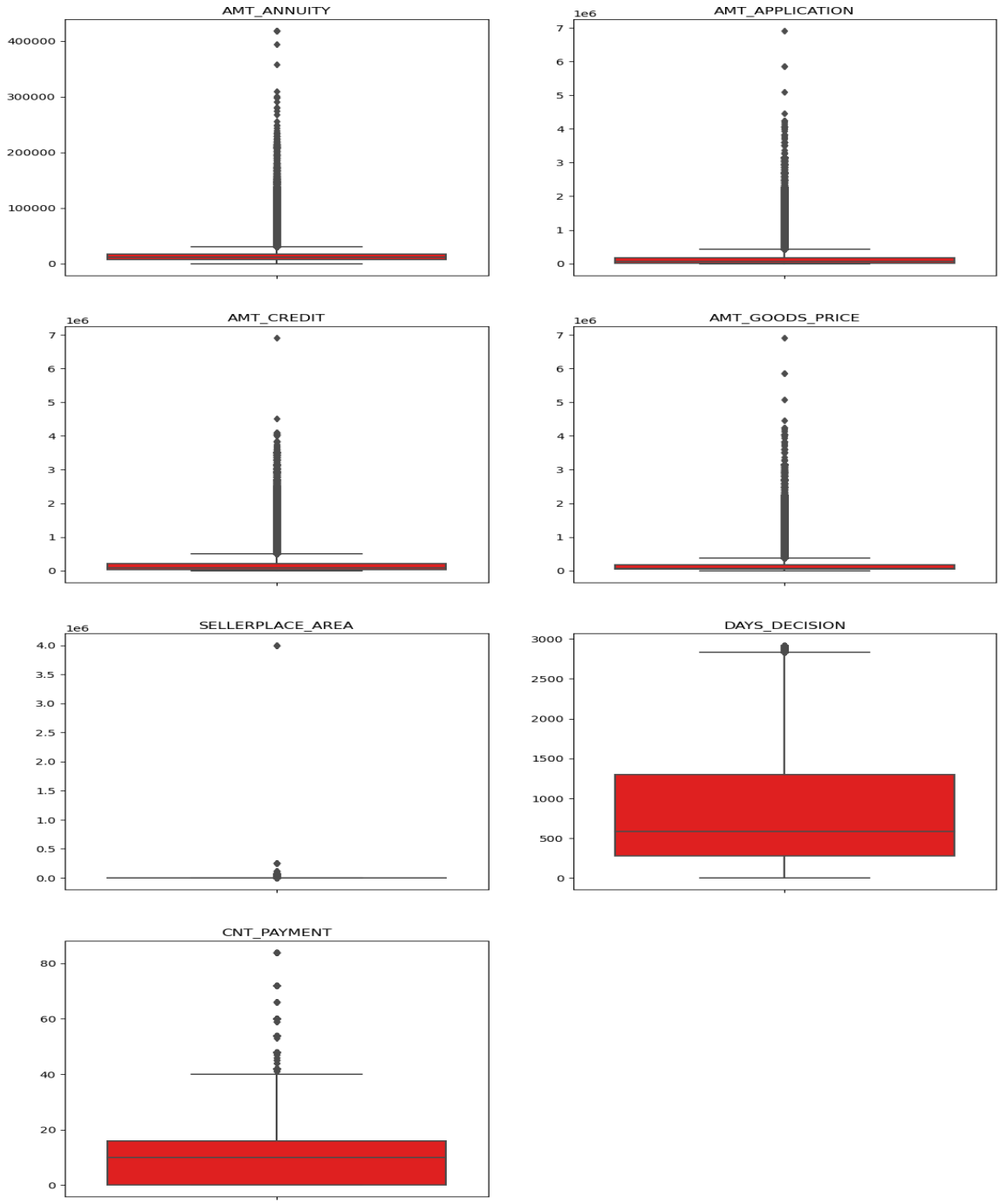
- ✓ From the describe() function, we can extract the columns that show a notable disparity between the maximum value and the 75th percentile. This discrepancy suggests that there might be extreme or outlier values present in these columns.
- ✓ Furthermore, we can identify specific columns where it seems unreasonable or nonsensical for the maximum value to be excessively high. Such values may be indicative of data entry errors, anomalies, or outliers.
- ✓ By examining these columns, we can focus our attention on understanding the reasons behind the high differences and evaluate whether any data cleaning or further analysis is required. Identifying these columns helps us pinpoint potential areas where data integrity issues or anomalies may exist, enabling us to take appropriate steps to address them.

- Adding the suspected columns to a new variable for the ease of analysis

```
# adding suspected columns to a new variable
outliers2 = ['AMT_ANNUITY','AMT_APPLICATION','AMT_CREDIT','AMT_GOODS_PRICE',
             'SELLERPLACE_AREA','DAYS_DECISION','CNT_PAYMENT']
```

- Plotting the figure to check the level of outliers present in the added columns

```
# operation to check the level of outliers present in the selected columns
plt.figure(figsize=[13,25])
for i,j in itertools.zip_longest(outliers2, range(len(outliers2))):
    plt.subplot(4,2,j+1)
    sns.boxplot(y = prev_app[i], orient = "h", color = "red")
    #plt.yticks(fontsize=6)
    plt.xlabel("")
    plt.ylabel("")
    plt.title(i)
```



Upon analyzing, several columns stand out due to the presence of a significant number of outliers. These outliers can provide valuable insights into the characteristics of the data. Here are the key observations:

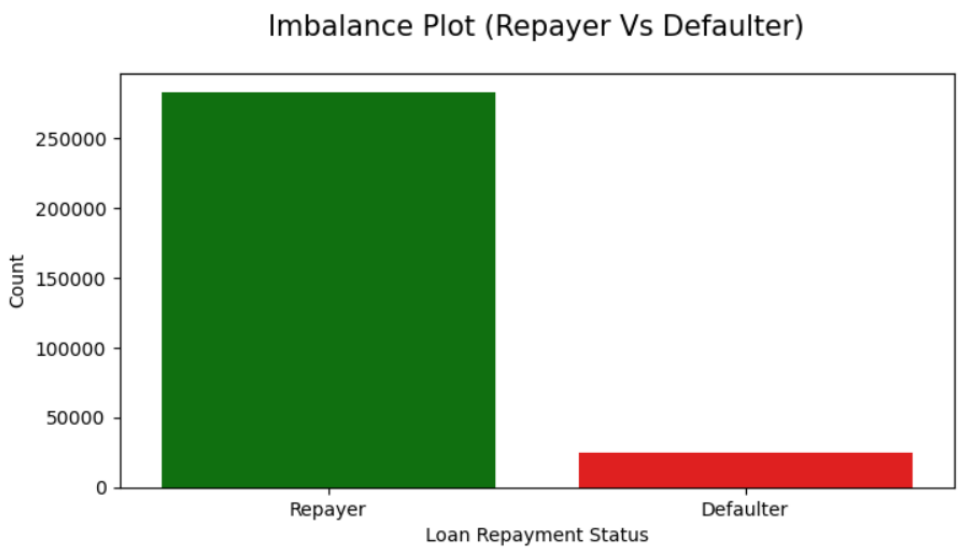
- ✓ AMT_ANNUITY, AMT_APPLICATION, AMT_CREDIT, AMT_GOODS_PRICE, and SELLERPLACE_AREA exhibit a substantial number of outliers. These columns likely represent financial aspects of the previous applications, such as the loan annuity, application amount, credit amount, goods price, and the area of the seller's location. The presence of numerous outliers suggests a wide range of values or exceptional cases in these financial measures.
- ✓ CNT_PAYMENT, on the other hand, has only a few outlier values. This column possibly represents the number of payments for the previous applications. The presence of outliers in this column indicates a few instances where the number of payments significantly deviates from the typical values.
- ✓ DAYS_DECISION shows a relatively small number of outliers. This column likely represents the number of days since the decision was made for each previous application. The limited number of outliers suggests that most of the decisions for these previous applications were made a long time ago, with only a few recent or unusual cases.

By identifying these outlier patterns, we gain insights into specific aspects of the previous application data. Understanding the characteristics and distribution of outliers can be crucial for data cleansing, anomaly detection, or gaining a deeper understanding of the underlying factors influencing previous application decisions.

- Now that the data processing, data cleaning and data manipulation has been done, next I'll be moving towards data analysis

- Data Imbalance: For observing the imbalance in data, I'll be plotting a figure depicting a relation between Repayer and Defaulter.

```
# imbalance in data
plt.figure(figsize=[8, 4])
sns.barplot(y=app_data["TARGET"].value_counts(), x=["Repayer", "Defaulter"], palette=["green", "red"], orient="v")
plt.xlabel("Loan Repayment Status", fontdict={"fontsize": 10})
plt.ylabel("Count", fontdict={"fontsize": 10})
plt.title("Imbalance Plot (Repayer Vs Defaulter)", fontdict={"fontsize": 15}, pad=20)
plt.show()
```




```
# ratio of imbalance percentage
repayer = round((app_data["TARGET"].value_counts()[0]/len(app_data)* 100),2)
print("Repayer Percentage is {}%".format(repayer))
defaluter = round((app_data["TARGET"].value_counts()[1]/len(app_data)* 100),2)
print("Defaulter Percentage is {}%".format(defaluter))
print("Ratio of imbalance for Repayer and Defaulter is given: {0:.2f}/1 (approx)".format(repayer/defaluter))
```

```
Repayer Percentage is 91.93%
Defaulter Percentage is 8.07%
Ratio of imbalance for Repayer and Defaulter is given: 11.39/1 (approx)
```

the code provides the percentage of "Repayer" and "Defaulter" occurrences in the TARGET column, and it calculates and prints the ratio of imbalance between these two categories i.e, 11.39/1

- Before moving on to categorical data analysis, it is important to create a function for plotting variables to do univariate analysis. Hence, will be creating two plots.
  - ✓ A count plot will be created to visualize the distribution of a given column with respect to the TARGET column.
  - ✓ The percentage of defaulters within that column will be calculated.

- I'll be creating a custom made function to find if the column is numerical or categorical using the following arguments
    - ✓ dataset: for the datasets we will be using
    - ✓ col: the name of the column
    - ✓ target_col: the name of the column that'll be used for comparing
    - ✓ If the plot becomes non readable in some cases then I'll be using ylog
    - ✓ x_label_angel: for the x-axis orientation
    - ✓ h_layout: for the horizontal layouts of subplots

```
# Creating a function to find if the column is categorical or numerical

def data_type(dataset,col):
    if dataset[col].dtype == np.int64 or dataset[col].dtype == np.float64:
        return "numerical"
    if dataset[col].dtype == "category":
        return "categorical"

# Creating a function "univariate" to perform analysis one single variable with respect to target variable

def univariate(dataset,col,target_col,ylog=False,x_label_angle=False,h_layout=True):
    if data_type(dataset,col) == "numerical":
        sns.distplot(dataset[col],hist=False)


    elif data_type(dataset,col) == "categorical":
        val_count = dataset[col].value_counts()
        df1 = pd.DataFrame({col: val_count.index,'count': val_count.values})


        target_1_percentage = dataset[[col, target_col]].groupby([col],as_index=False).mean()
        target_1_percentage[target_col] = target_1_percentage[target_col]*100
        target_1_percentage.sort_values(by=target_col,inplace = True)
```

```
# If the plot is not readable, use the log scale

        if(h_layout):
            fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(15,7))
        else:
            fig, (ax1, ax2) = plt.subplots(nrows=2, figsize=(25,35))


# 1. Subplot 1: Count plot of the column

        s = sns.countplot(ax=ax1, x=col, data=dataset, hue=target_col)
        ax1.set_title(col, fontsize = 18)
        ax1.legend(['Repayer','Defaulter'])
        ax1.set_xlabel(col,fontdict={'fontsize' : 12, 'fontweight' : 3})

        if(x_label_angle):
            s.set_xticklabels(s.get_xticklabels(),rotation=75)

# 2. Subplot 2: Percentage of defaulters within the column
        S
        s = sns.barplot(ax=ax2, x = col, y=target_col, data=target_1_percentage)
        ax2.set_title("Defaulters % in "+col, fontsize = 18)
        ax2.set_xlabel(col,fontdict={'fontsize' : 12, 'fontweight' : 3})
        ax2.set_ylabel(target_col,fontdict={'fontsize' : 12, 'fontweight' : 3})

        if(x_label_angle):
            s.set_xticklabels(s.get_xticklabels(),rotation=75)
```

```
# If the plot is not readable, use the log scale

        if ylog:
            ax1.set_yscale('log')
            ax1.set_ylabel("Count (log)",fontdict={'fontsize' : 12, 'fontweight' : 3})
        else:
            ax1.set_ylabel("Count",fontdict={'fontsize' : 12, 'fontweight' : 3})


        plt.show()
```

Understanding the above code:

- ✓ The data_type function is defined to determine whether a column in the dataset is categorical or numerical. It checks the data type of the column and returns "numerical" if it's either np.int64 or np.float64, and "categorical" if it's "category".
- ✓ The univariate function is created to perform analysis on a single variable with respect to the target variable.
- ✓ If the column is determined to be numerical, a distribution plot (sns.distplot) is plotted using sns.distplot(dataset[col], hist=False). This plot displays the distribution of values for the numerical column.
- ✓ If the column is categorical, the value counts for each category are calculated using dataset[col].value_counts(), and a dataframe (df1) is created to store the category and its corresponding count.
- ✓ The percentage of defaulters within each category is calculated by grouping the dataset by the column and taking the mean of the target column. The resulting dataframe (target_1_percentage) is sorted based on the target column values.
- ✓ Two subplots are created using plt.subplots. If h_layout is True, the subplots are arranged horizontally (ncols=2), otherwise they are arranged vertically (nrows=2).
- ✓ Subplot 1: A count plot (sns.countplot) is created using sns.countplot(ax=ax1, x=col, data=dataset, hue=target_col). It displays the count of each category in the column and differentiates the count based on the target variable. The title, legend, and labels for this subplot are set accordingly.
- ✓ Subplot 2: A bar plot (sns.barplot) is created using sns.barplot(ax=ax2, x=col, y=target_col, data=target_1_percentage). It represents the percentage of defaulters within each category. The title and labels for this subplot are set accordingly.

- ✓ If x_label_angle is True, the x-axis labels in both subplots are rotated by 75 degrees for better readability.
- ✓ If ylog is True, the y-axis scale of the count plot is set to logarithmic using ax1.set_yscale('log'). This is useful when the count values have a wide range and a logarithmic scale helps in visualizing the distribution better.
- ✓ Finally, the plots are displayed using plt.show().

The code will allow me to analyze a single variable in the dataset and compare it with the target variable, providing insights into the distribution of numerical variables and the percentage of defaulters within categorical variables.

- Creating a code for plotting repetitive barplots in bivariate numerical and categorical analysis

```
# function for plotting repetitive plots in bivaritae numerical analysis

def bivariate_n(x,y,df,hue,kind,labels):
    plt.figure(figsize=[15,15])
    sns.relplot(x=x, y=y, data=df, hue=hue,kind=kind,legend = False)
    plt.legend(labels=labels)
    plt.xticks(rotation=45, ha='right')
    plt.show()
```

Understanding the above code :

- ✓ The function creates a figure with a size of 15x15 using plt.figure(figsize=[15,15]).
- ✓ It uses sns.relplot() to create the bivariate plot. The x and y parameters specify the columns to be plotted on the x-axis and y-axis, respectively. The data parameter specifies the dataframe from which the data is taken. The hue parameter is used to differentiate the data points or lines based on a specific column. The kind parameter specifies the type of plot to be created.
- ✓ The legend parameter is set to False to hide the default legend created by sns.relplot().
- ✓ plt.legend(labels=labels) is used to create a legend with the specified labels.
- ✓ plt.xticks(rotation=45, ha='right') is used to rotate the x-axis tick labels by 45 degrees and align them to the right for better readability.

- ✓ Finally, plt.show() is called to display the plot.

This function will allow me to quickly create bivariate plots with customizable features such as colour differentiation, plot type, and legend labels.

```
# function for plotting repetitive barplots in bivariate categorical analysis

def bivariate_c(x,y,df,hue,figsize,labels):

    plt.figure(figsize=figsize)
    sns.barplot(x=x,y=y,data=df, hue=hue)

    # Defining aesthetics of Labels and Title of the plot using style dictionaries
    plt.xlabel(x,fontsize = 12)
    plt.ylabel(y,fontsize = 12)
    plt.title(col,fontsize = 18)
    plt.xticks(rotation=45, ha='right')
    plt.legend(labels = labels )
    plt.show()
```

Understanding the above code: The code defines a function called bivariate_c which is used to create a bivariate analysis plot using a bar plot.

- ✓ The function creates a figure with the specified size using plt.figure(figsize=figsize).
- ✓ It uses sns.barplot() to create a bar plot. The x and y parameters specify the columns to be plotted on the x-axis and y-axis, respectively. The data parameter specifies the dataframe from which the data is taken. The hue parameter is used to differentiate the bars based on a specific column.
- ✓ The function defines the aesthetics of the plot, such as x-label, y-label, and title, using plt.xlabel(), plt.ylabel(), and plt.title() respectively. The specified values for x-label, y-label, and title are taken from the respective function parameters.
- ✓ plt.xticks(rotation=45, ha='right') is used to rotate the x-axis tick labels by 45 degrees and align them to the right for better readability.
- ✓ plt.legend(labels=labels) is used to create a legend with the specified labels.
- ✓ Finally, plt.show() is called to display the plot.

This function will allow me to create a bivariate bar plot with customizable features such as colour differentiation, plot size, and labels.

Next code involves creation of a function for plotting repetitive countplots in univariate categorical analysis for merged categories

```
#function for plotting repetitive countplots in univariate categorical analysis on the merged df

def univariate_c_merged(col,df,hue,palette,ylog,figsize):
    plt.figure(figsize=figsize)
    ax=sns.countplot(x=col, data=df,hue= hue,palette= palette,order=df[col].value_counts().index)


    if ylog:
        plt.yscale('log')
        plt.ylabel("Count (log)",fontsize=12)
    else:
        plt.ylabel("Count",fontsize=15)

    plt.title(col , fontsize=18)
    plt.legend(loc = "upper right")
    plt.xticks(rotation=45, ha='right')

    plt.show()
```

Understanding the code: The code defines a function called univariate_c_merged which is used to create a univariate analysis plot using a count plot with merged categories.

- ✓ The function creates a figure with the specified size using plt.figure(figsize=figsize).
- ✓ It uses sns.countplot() to create a count plot. The x parameter specifies the column to be plotted on the x-axis, and the data parameter specifies the dataframe from which the data is taken. The hue parameter is used to differentiate the bars based on a specific column. The palette parameter specifies the color palette to be used for different categories.
- ✓ If ylog is True, the function sets the y-axis scale to logarithmic using plt.yscale('log') and sets the y-label to "Count (log)". Otherwise, it sets the y-label to "Count".
- ✓ The function sets the plot title to the value of col using plt.title().
- ✓ plt.legend(loc = "upper right") is used to create a legend in the upper right corner of the plot.
- ✓ plt.xticks(rotation=45, ha='right') is used to rotate the x-axis tick labels by 45 degrees and align them to the right for better readability.
- ✓ Finally, plt.show() is called to display the plot.

This function will allow me to create a univariate count plot with merged categories, customizable features such as color differentiation, plot size, and the option to use a logarithmic scale on the y-axis.

And lastly, I'll create a function to plot the point plots

```
# Function to plot point plots

def pointplot(df,hue,x,y):
    plt.figure(figsize=(10,4))
    sns.pointplot(x=x, y=y, hue=hue, data=df)
    plt.title(x+" VS "+y,fontsize = 12)
```

Understanding the code: The function pointplot is used to create a point plot using the Seaborn library in Python.

- ✓ The function sets the figure size to (10, 4) using plt.figure(figsize=(10,4)).
- ✓ Then, it uses sns.pointplot to create the point plot, where x and y are specified along with the DataFrame df and hue to differentiate the points by color.
- ✓ Finally, it sets the title of the plot using the concatenation of x and y variables.

The point plot displays the relationship between two variables (x and y) by showing the average value of y for each unique value of x, and the hue parameter allows further differentiation based on a third variable.

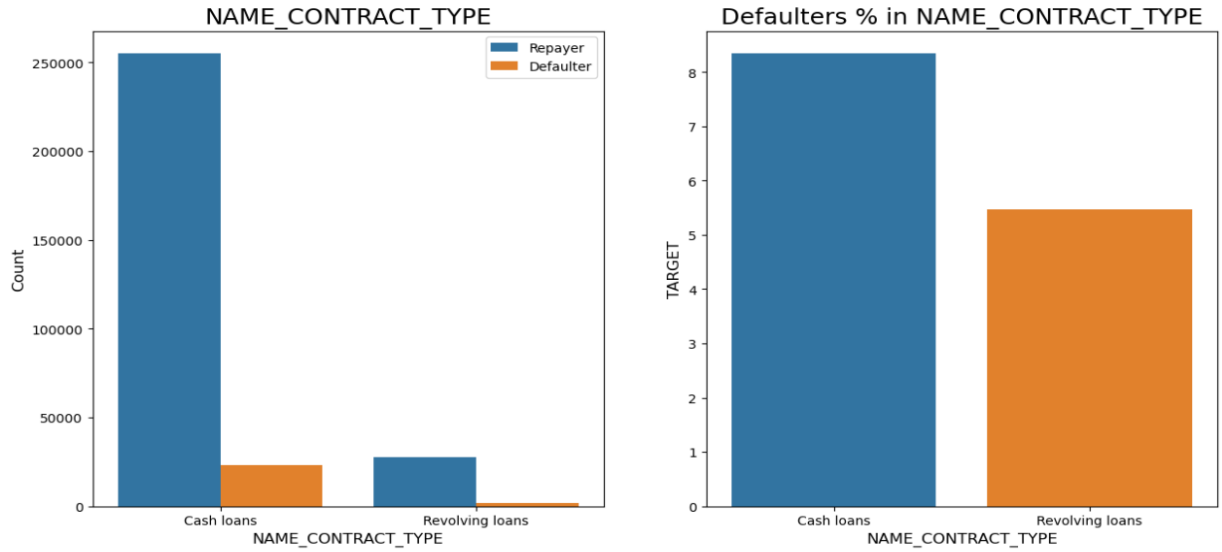
- For storing the list of categorical as well as numerical column list, I used two new variables col_cat and col_num.

```
# for storing categorical column list
col_cat = list(app_data.select_dtypes(["category"]).columns)

#for storing numerical column list
col_num = list(app_data.select_dtypes(["int","float"]).columns)
```

- For segmented univariate analysis, I'll be checking the contract type based on loan repayment status

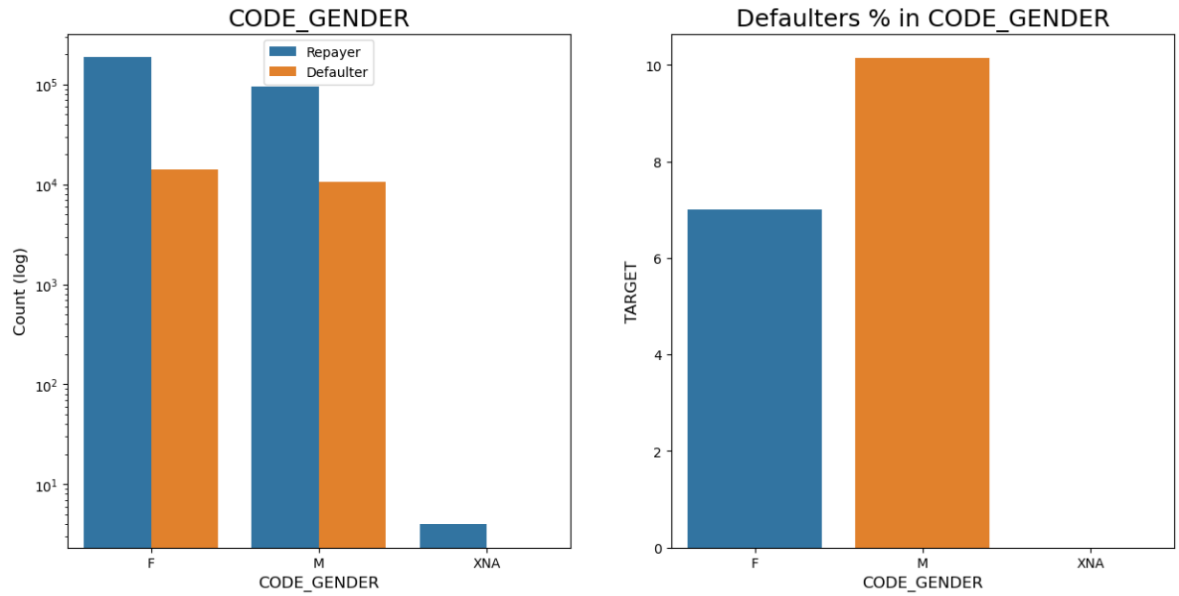
```
# checking the contract type on loan repayment status
univariate(app_data,"NAME_CONTRACT_TYPE","TARGET",False,False,True)
```



The above plot suggests that revolving loans constitute only a small portion, approximately 10%, of the total number of loans. It also states that around 8-9% of cash loan applicants and 5-6% of revolving loan applicants are classified as defaulters.

- Next, I'll be checking the type of gender on loan repayment status

```
#2 Checking the type of Gender on loan repayment status
univariate(app_data,"CODE_GENDER","TARGET",True,False,True)
```

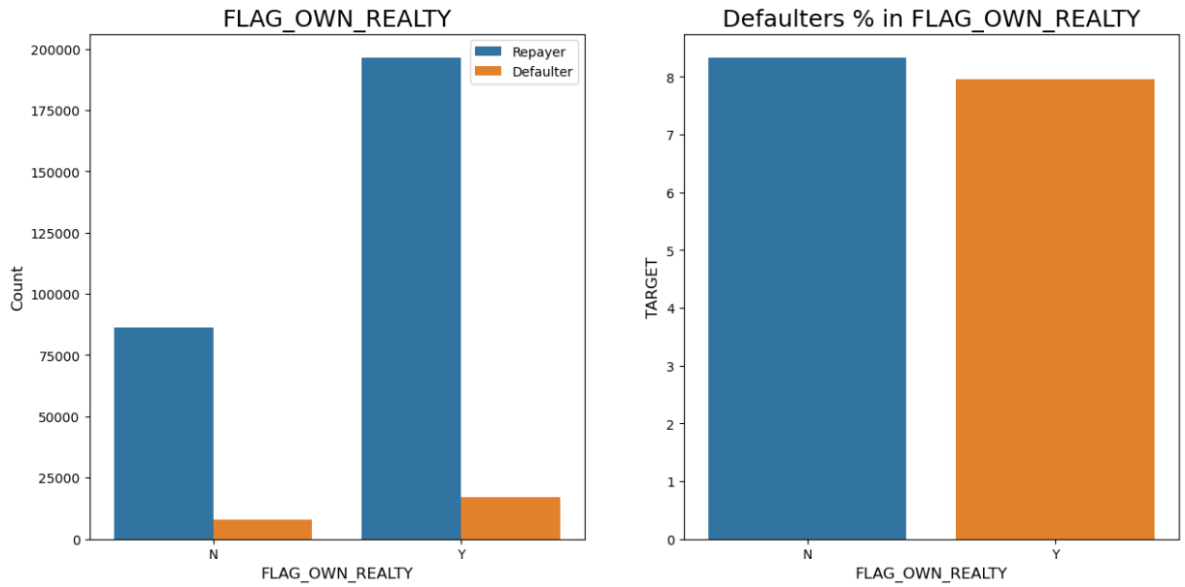


The above polt indicates that the number of female clients is nearly twice the number of male clients. Additionally, based on the percentage of defaulted

credits, males have a higher likelihood, approximately 10%, of not repaying their loans compared to women, who have a default rate of around 7%.

- Next, I'll be checking whether owning a real estate is related to loan repayment status

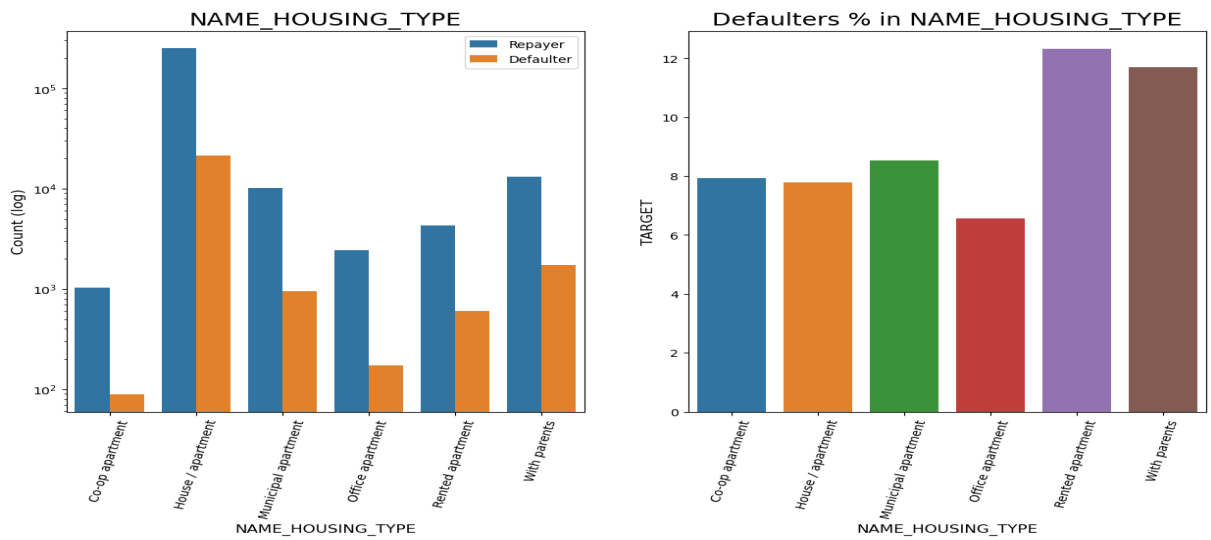
```
# Checking whether owning a real estate is related to loan repayment status
univariate(app_data,"FLAG_OWN_REALTY","TARGET",False,False,True)
```



The above plott highlights that the number of clients who own real estate is more than double the number of clients who do not own real estate. Despite this difference, both categories have a similar defaulting rate, approximately 8%. This suggests that there is no significant correlation between owning real estate and the likelihood of defaulting on a loan.

- Next, I'll be analyzing the housing type on loan repayment status

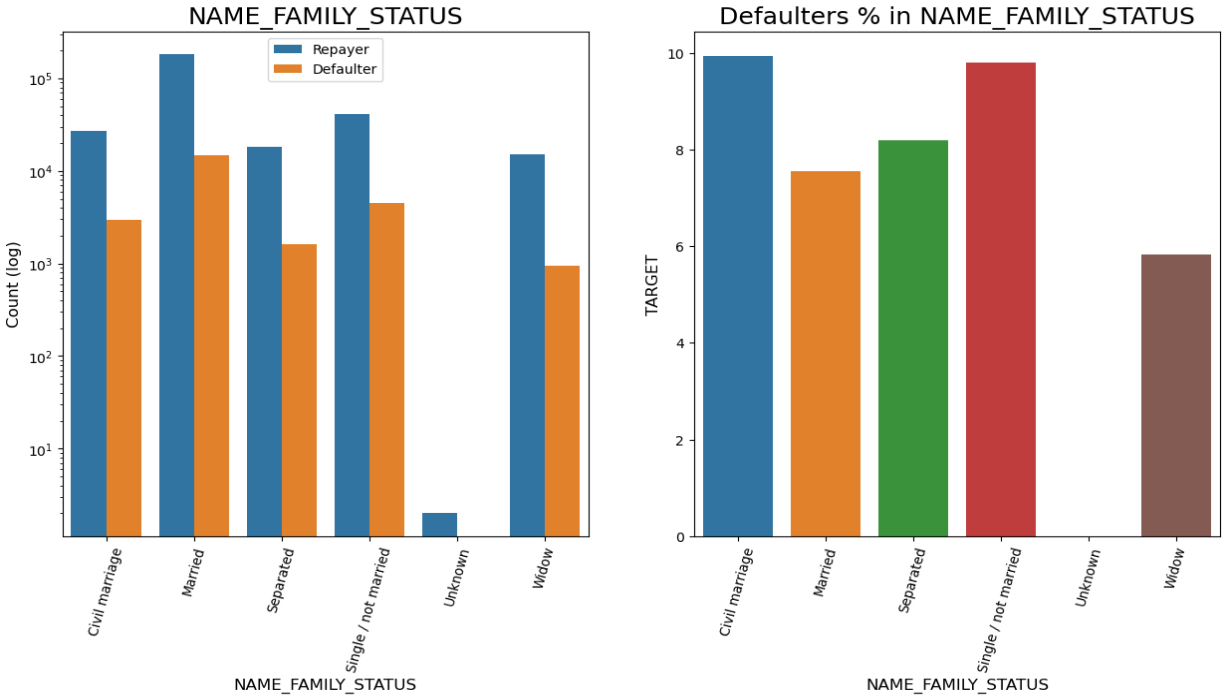
```
# checking if housing type is based on loan repayment status
univariate(app_data,"NAME_HOUSING_TYPE","TARGET",True,True,True)
```



The above plot reveals that the majority of individuals in the dataset are living in a house or an apartment, indicating that this is the most common living arrangement. Interestingly, among different types of living arrangements, individuals residing in office apartments exhibit the lowest default rate. On the other hand, individuals living with their parents have a default rate of approximately 11.5%, and those living in rented apartments have a higher probability of defaulting, exceeding 12%. This suggests that living with parents or renting an apartment may be associated with a higher risk of loan default compared to other living arrangements.

- Next, I'll be checking family status based on loan repayment status

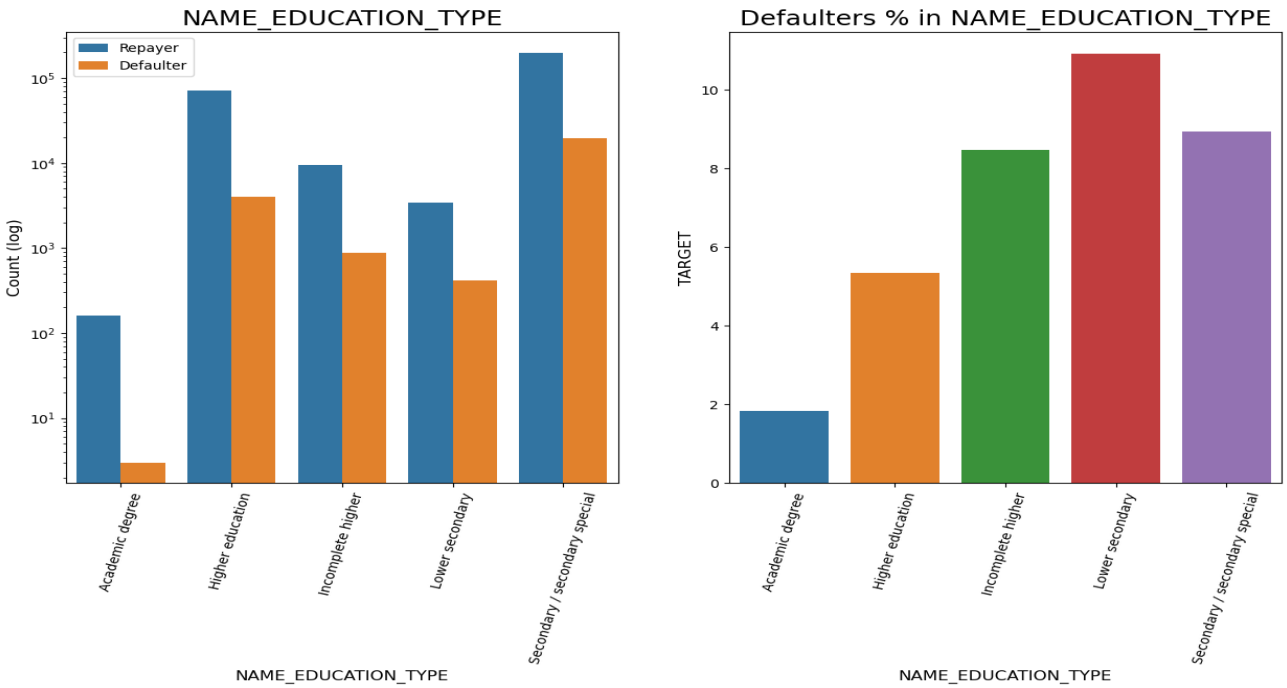
```
# checking family status based on loan repayment status
univariate(app_data,"NAME_FAMILY_STATUS","TARGET",True,True,True)
```



The above plot suggests that the majority of individuals who have taken a loan are married, followed by those who are single or not married, and individuals in civil marriages. When considering the percentage of defaulters, individuals in civil marriages have the highest percentage of defaults, accounting for approximately 10% of the defaulting group. On the other hand, individuals who are widowed have the lowest percentage of defaults, with a rate of around 6%, excluding the category of "Unknown." This suggests that marital status, particularly civil marriage, may play a role in the likelihood of loan default, with civil marriages exhibiting a higher default rate compared to other categories.

- Next, I'll be analyzing education type based on loan repayment status

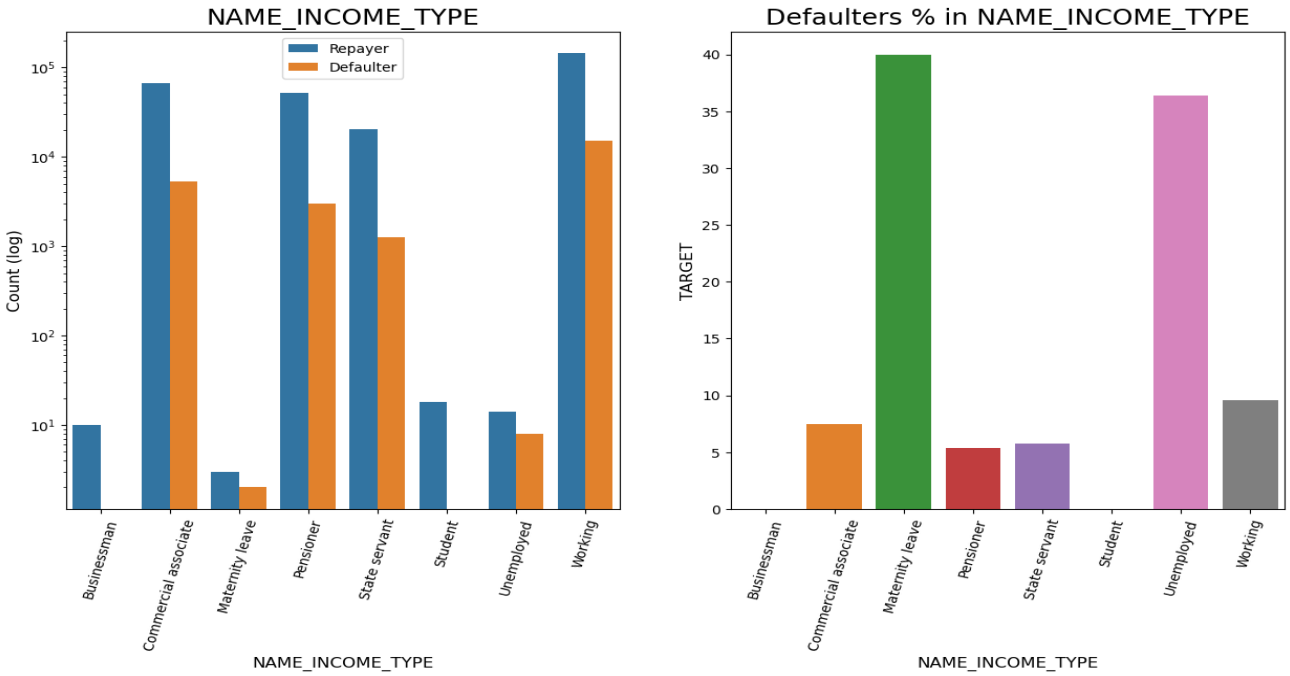
```
# analyzing Education Type based on loan repayment status
univariate(app_data,"NAME_EDUCATION_TYPE","TARGET",True,True,True)
```



The above plot shows that the majority of clients have completed secondary or secondary special education, with a higher number of clients having higher education qualifications. However, there are very few clients who hold an academic degree. Among these education categories, clients with lower secondary education have the highest rate of defaulting, with approximately 11% of this group defaulting on their loans. On the other hand, clients with academic degrees are the least likely to default, indicating a lower risk of loan default among individuals with higher educational qualifications.

- Next, I'll be analyzing the income type based on loan repayment status

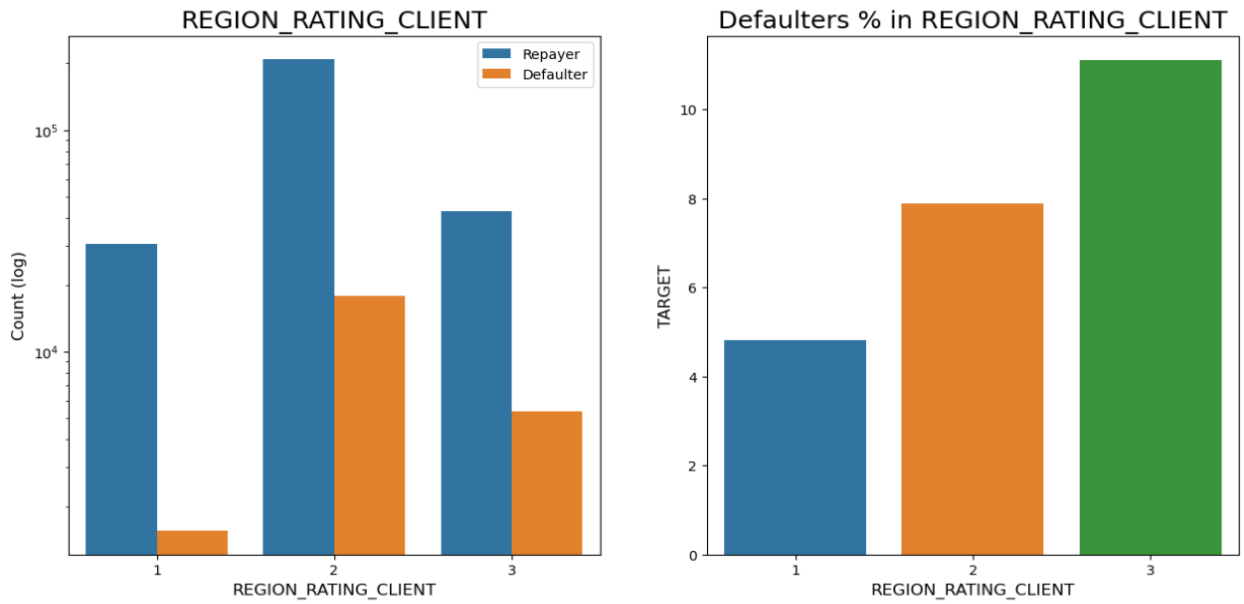
```
# analyzing income type based on loan repayment status
univariate(app_data,"NAME_INCOME_TYPE","TARGET",True,True,True)
```



The above plot shows that the majority of loan applicants have income from working, followed by commercial associates, pensioners, and state servants. Among the various income types, applicants on maternity leave have the highest defaulting percentage, with 40% of this group defaulting on their loans. The unemployed category follows closely with a defaulting percentage of 37%. The remaining income types have an average defaulting rate of around 10%. Interestingly, applicants who identify as students or businessmen, although they are fewer in numbers, have no records of defaulting. This makes them the safest categories for providing loans, as they have demonstrated a reliable repayment behavior.

- Next, I'll be analyzing the region rating where applicants live based on loan repayment status

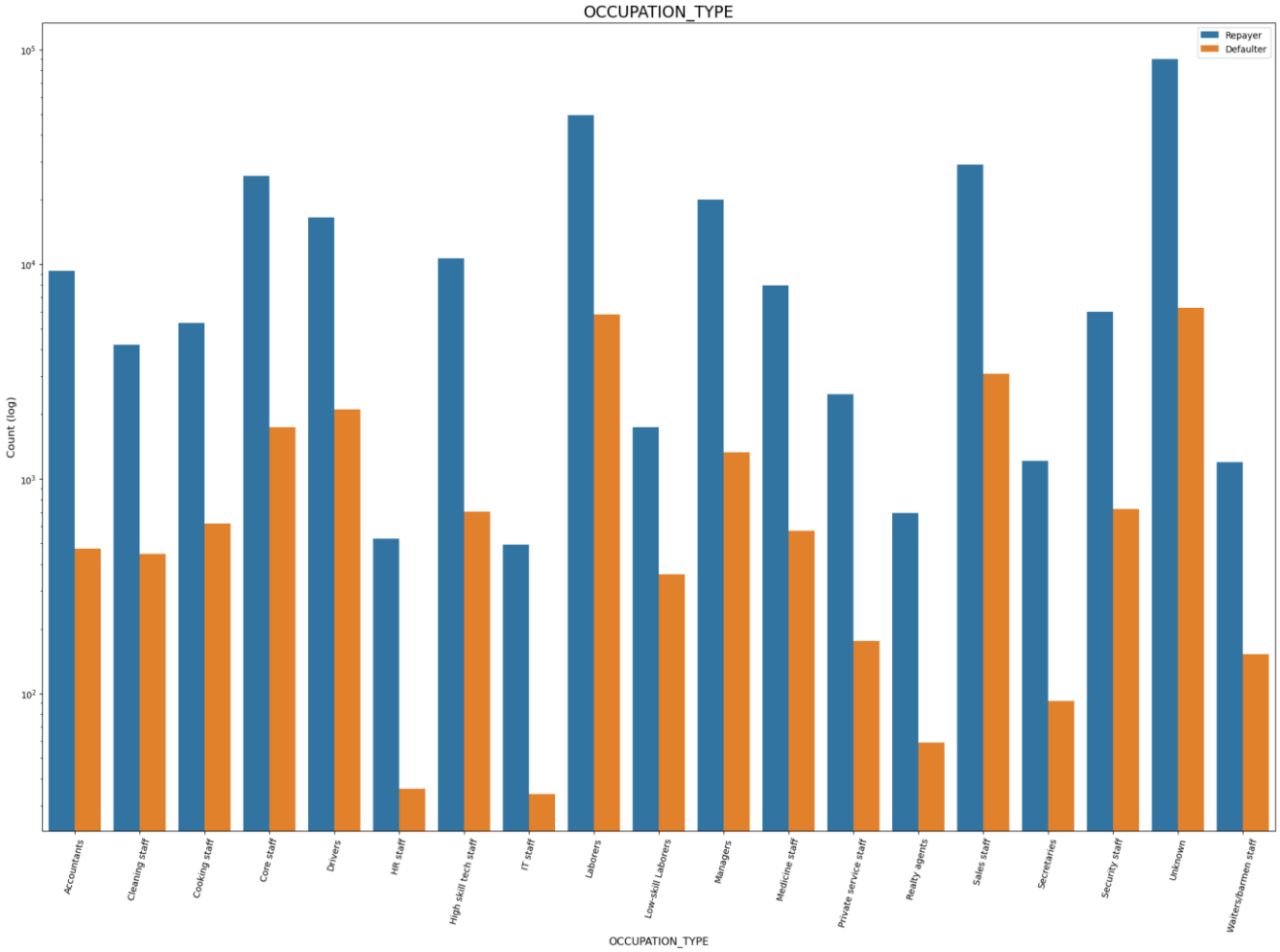
```
# analyzing region rating where applicant lives based on loan repayment status
univariate(app_data,"REGION_RATING_CLIENT","TARGET",True,False,True)
```

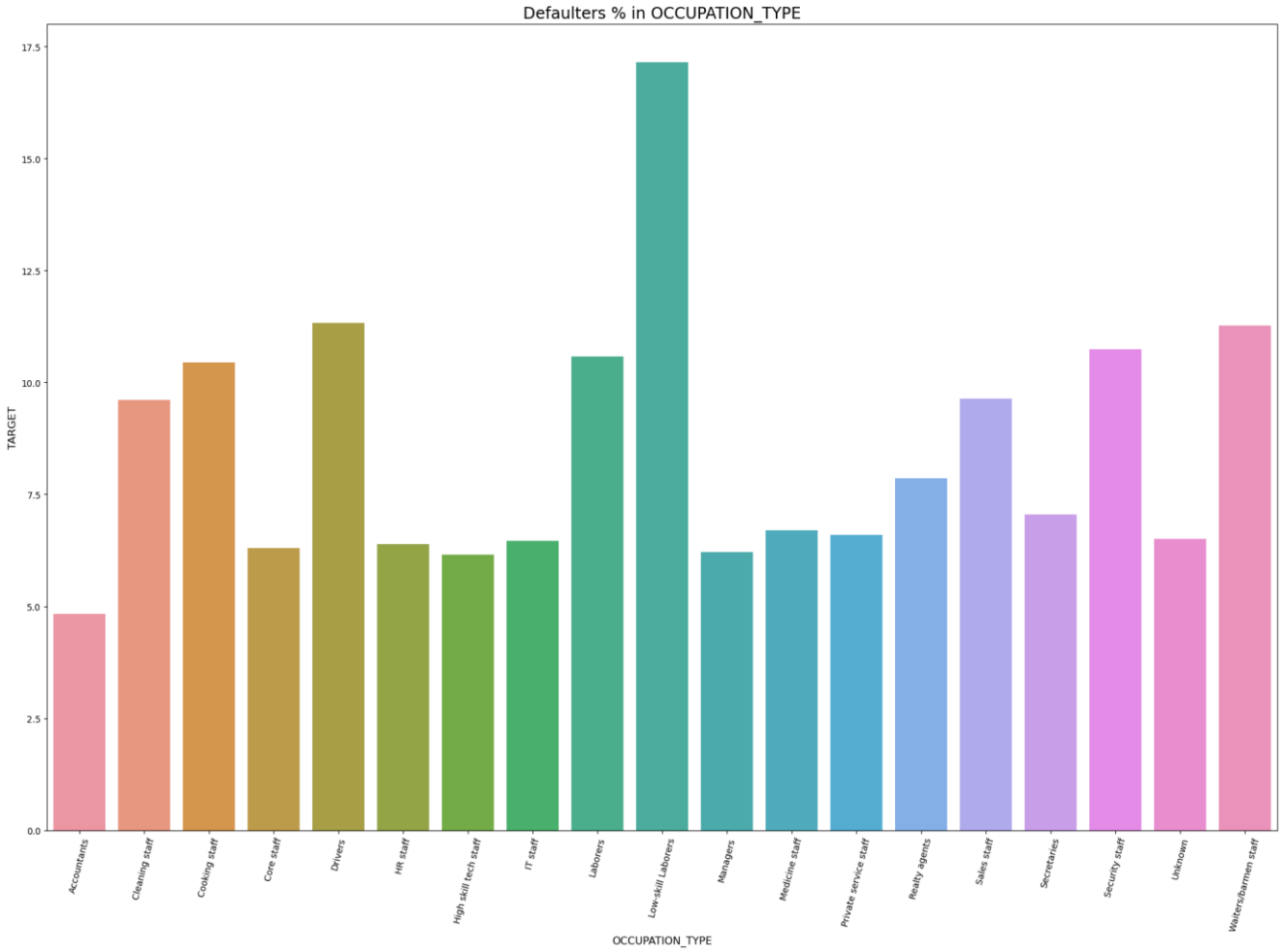


The majority of loan applicants are residing in regions with a rating of 2. However, it is worth noting that applicants living in regions with a rating of 3 have the highest default rate, with approximately 11% of them defaulting on their loans. On the other hand, applicants residing in regions with a rating of 1 have the lowest probability of defaulting, indicating that these regions are safer for approving loans.

- Next, I'll be analyzing the occupation type where applicants are based on loan repayment status

```
# analyzing occupation type where applicant are based on loan repayment status
univariate(app_data,"OCCUPATION_TYPE","TARGET",True,True,False)
```


OCCUPATION_TYPE
Repayer
Defaulter
Count (log)
10^5
10^4
10^3
10^2
Accountants
Cleaning staff
Cooking staff
Core staff
Drivers
HR staff
High skill tech staff
IT staff
Laborers
Low-skill Laborers
Managers
Medicine staff
Private service staff
Realty agents
Sales staff
Secretaries
Security staff
Unknown
Waiters/barmen staff
OCCUPATION_TYPE


Defaulters % in OCCUPATION_TYPE
TARGET
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Accountants
Cleaning staff
Cooking staff
Core staff
Drivers
HR staff
High skill tech staff
IT staff
Laborers
Low-skill Laborers
Managers
Medicine staff
Private service staff
Realty agents
Sales staff
Secretaries
Security staff
Unknown
Waiters/barmen staff
OCCUPATION_TYPE

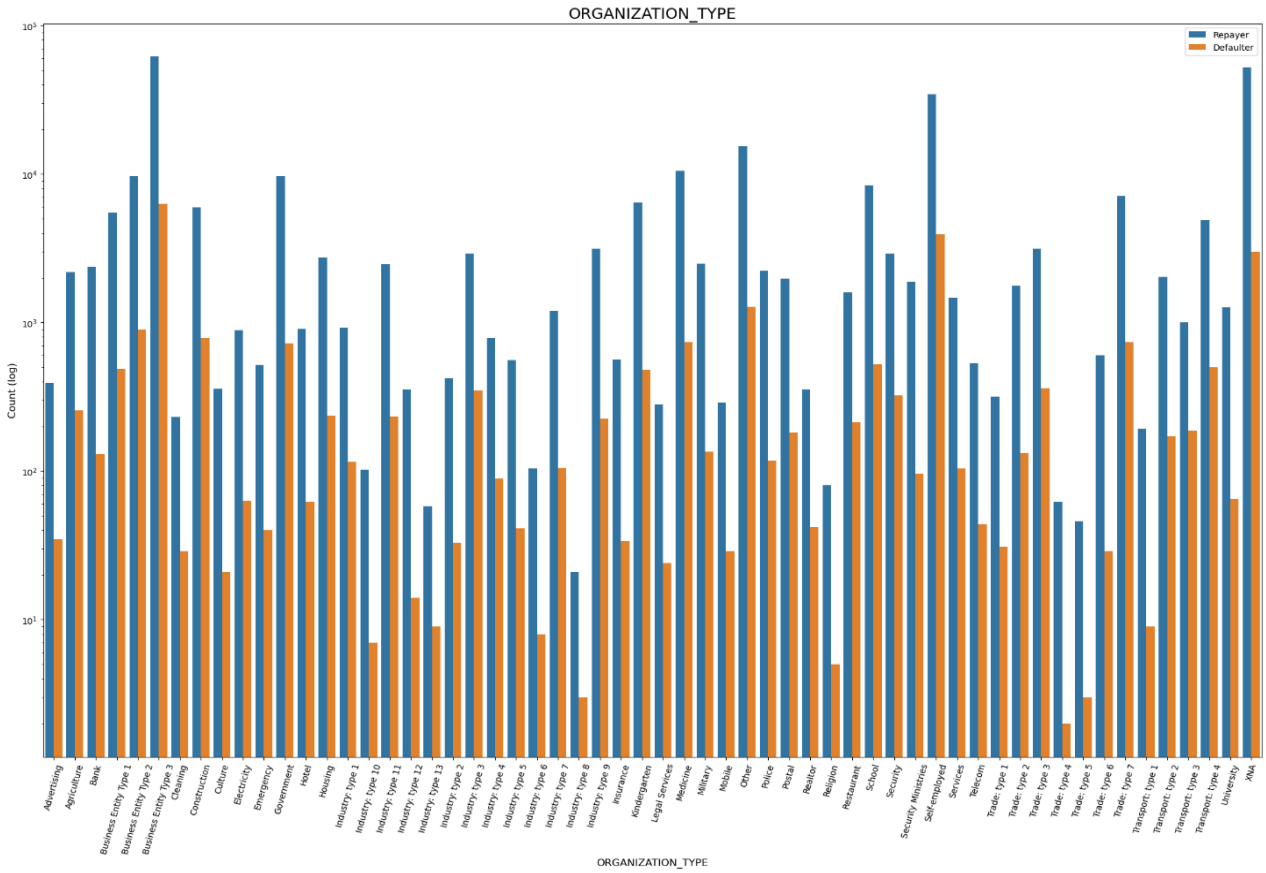
The majority of loans are taken by laborers, with sales staff being the second-largest group of loan applicants. In contrast, IT staff members are less likely to apply for loans compared to other professions.

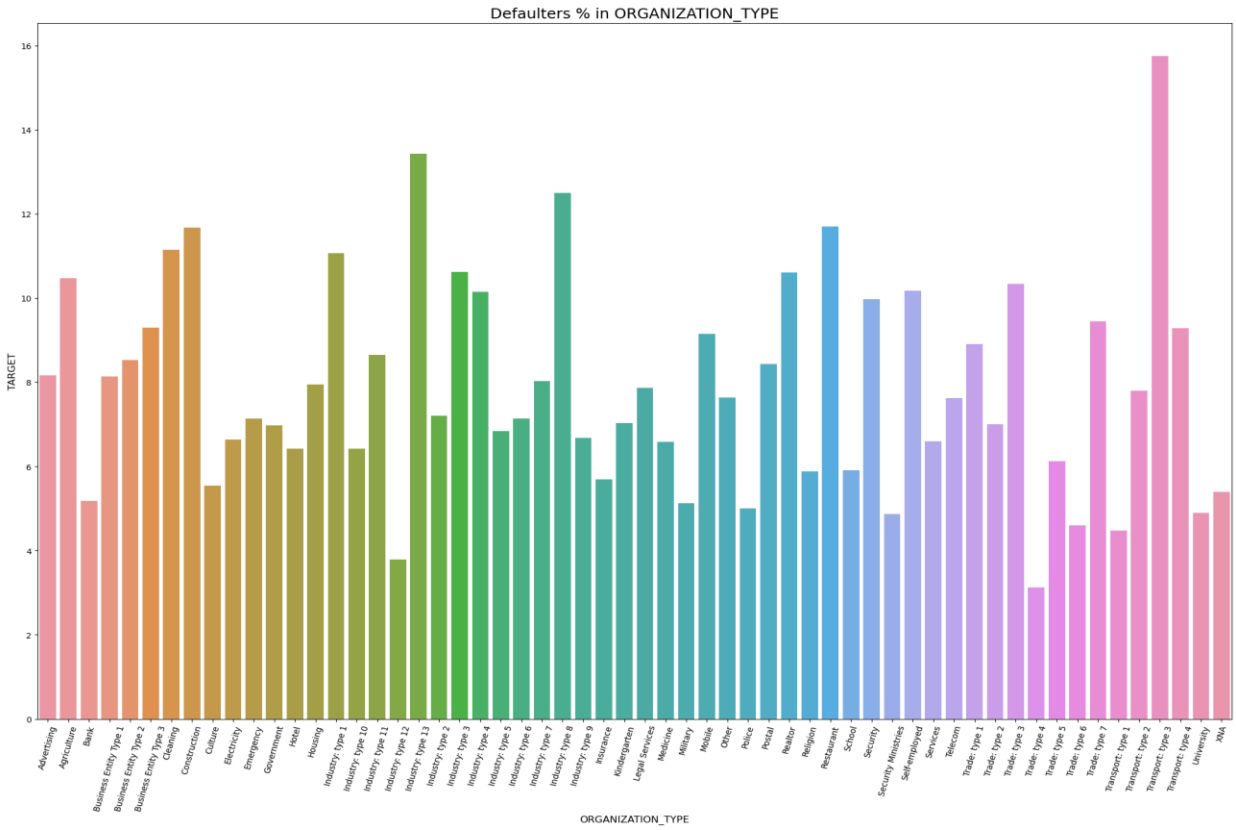
When examining the default rates, the category with the highest percentage of defaulters is low-skill laborers, with a rate exceeding 17%. This is followed by drivers, waiters/barmen staff, security staff, general laborers, and cooking staff. These categories have a relatively higher likelihood of defaulting on their loans.

- Next, I'll be analyzing loan repayment status based on organization type

```
# analyzing loan repayment status based on Organization type

univariate(app_data,"ORGANIZATION_TYPE","TARGET",True,True,False)
```





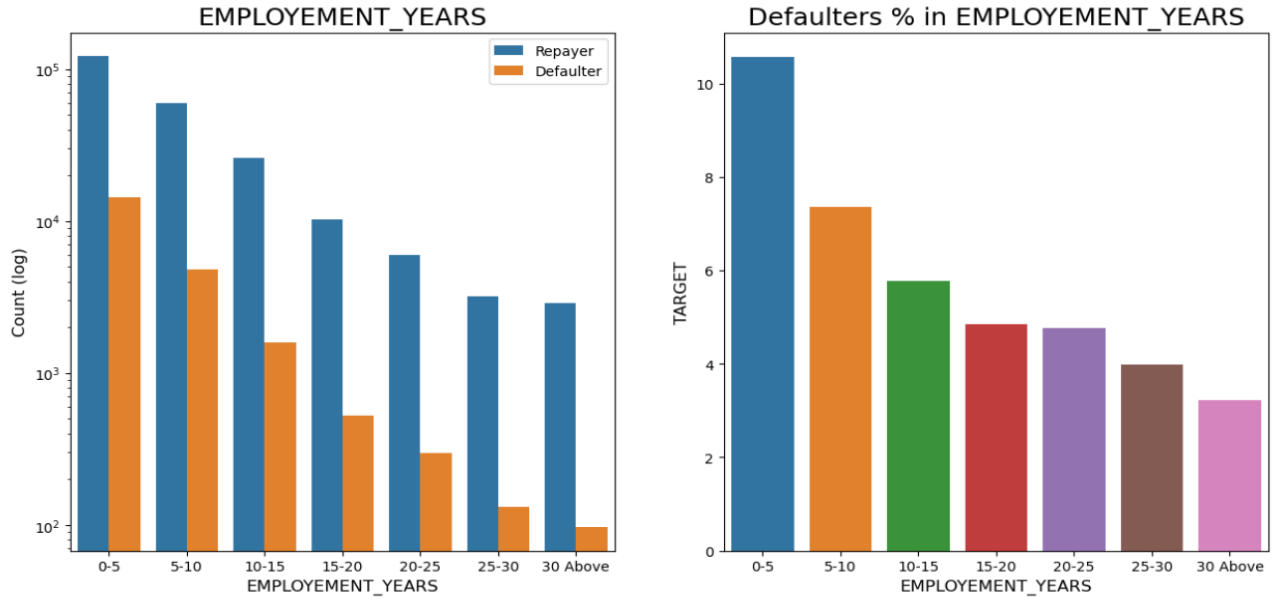
The above plots shows that the organizations with the highest percentage of defaulters include Transport: type 3 with a rate of 16%, Industry: type 13 with a rate of 13.5%, Industry: type 8 with a rate of 12.5%, and the Restaurant category with a rate below 12%.

Self-employed individuals tend to have a relatively higher defaulting rate, indicating that caution should be exercised when disbursing loans to this group. One approach to mitigate the risk of defaulting is to either avoid providing loans to self-employed individuals or offer loans with higher interest rates.

A significant number of loan applications come from Business Entity Type 3. However, there is a large proportion of applications where the organization type information is unavailable (XNA). To ensure a safer lending environment, it is advisable to focus on categories of organization types with lower default rates. Specifically, Trade Type 4 and 5, as well as Industry type 8, have fewer defaulters and can be considered safer options for loan disbursement.

- Next, I'll be checking employment years based on loan repayment status

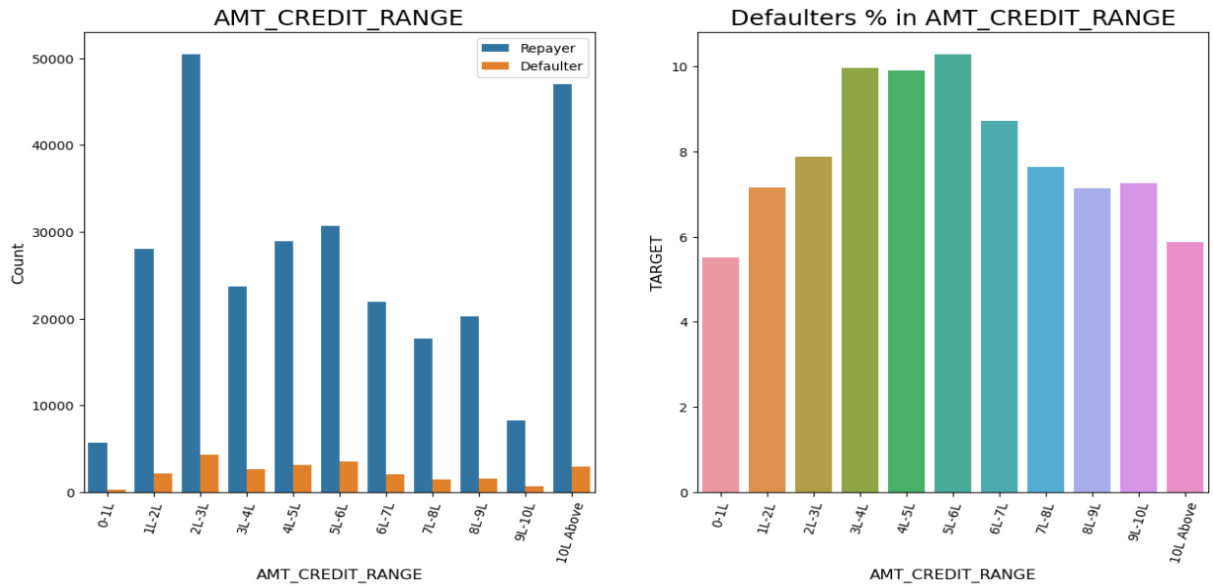
```
# checking employment_year based on loan repayment status
univariate(app_data,"EMPLOYEMENT_YEARS","TARGET",True,False,True)
```



The above plot shows that the majority of loan applicants with working experience between 0-5 years are defaulters, and this group has the highest defaulting rate at around 10%. However, as the number of employment years increases, the defaulting rate gradually decreases. Interestingly, individuals with 40+ years of working experience have a default rate of less than 1%, indicating a higher level of reliability in loan repayment for this group.

- Next, I'll be checking amount credit based on loan repayment status

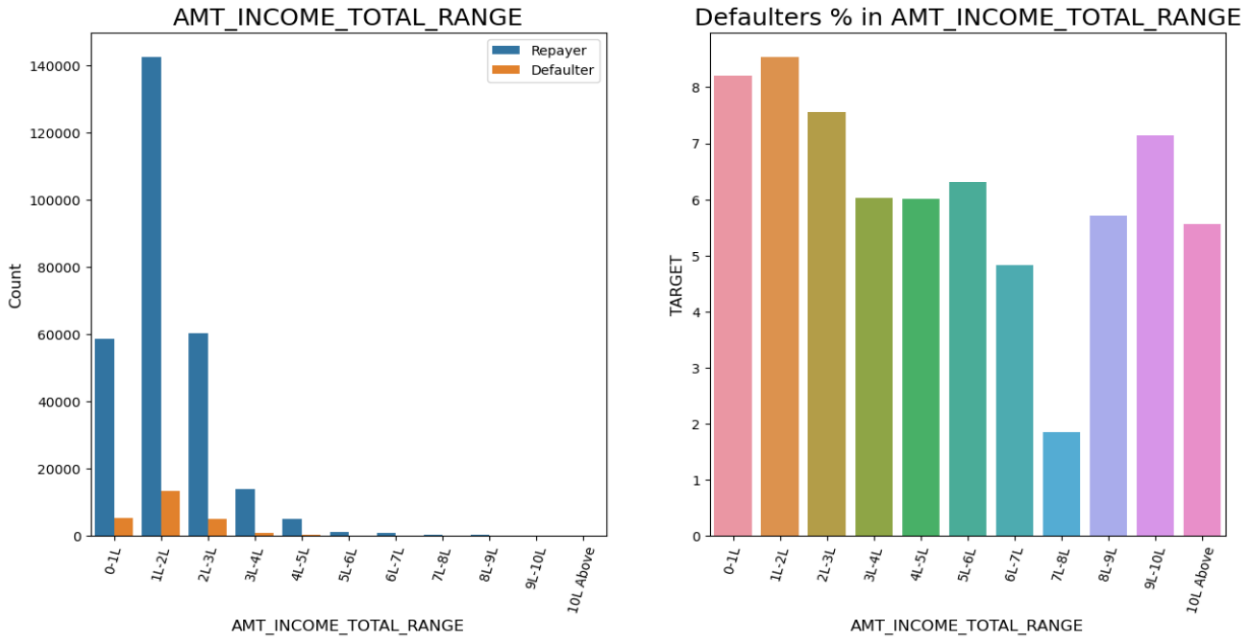
```
# analyzing amount credit based on loan repayment status
univariate(app_data,"AMT_CREDIT_RANGE","TARGET",False,True,True)
```



The above plot stated that a significant number of loan applicants have loans in the range of 2-3 lakhs, followed by loans above 10 lakhs. However, it is noteworthy that applicants who obtained loans in the range of 3-6 lakhs have the highest number of defaulters compared to other loan ranges. This suggests that loans within this specific range may carry a higher risk of default.

- Next, I'll be checking amount income range based on loan repayment status

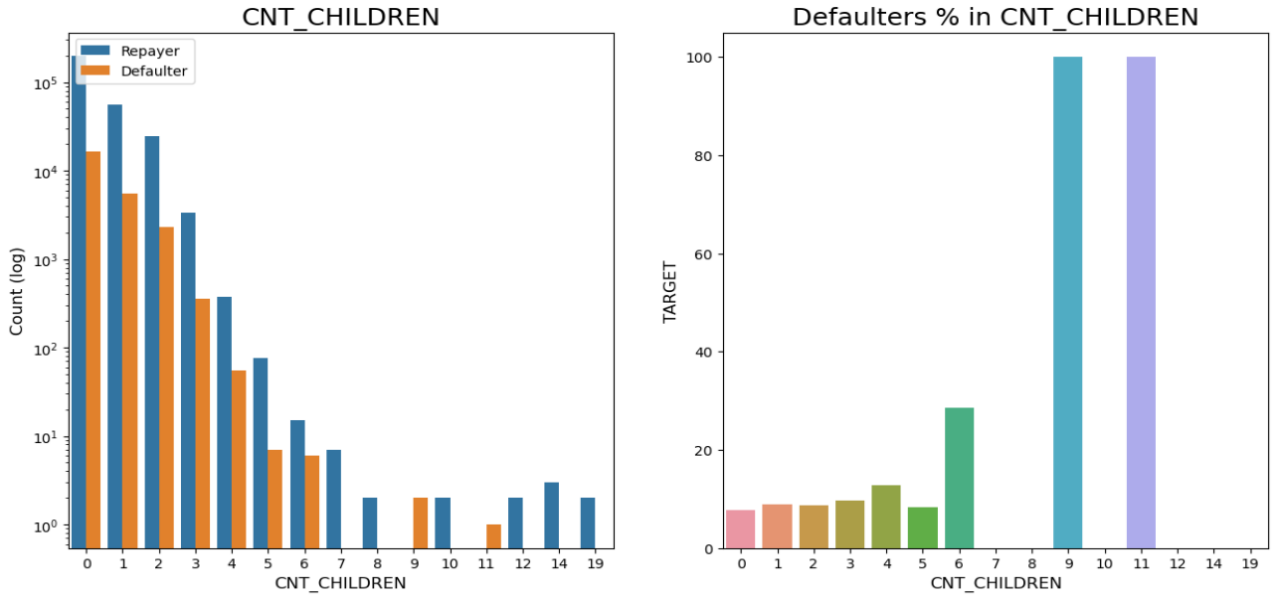
```
# checking amount income range based on loan repayment status
univariate(app_data,"AMT_INCOME_TOTAL_RANGE","TARGET",False,True,True)
```



The majority of loan applications have an income total of less than 3 lakhs. It is observed that applicants with an income below this threshold have a higher probability of defaulting on their loans. On the other hand, applicants with an income in the range of 7-8 lakhs are less likely to default, indicating a lower risk associated with this income bracket.

- Next, I'll be checking number of children based on loan repayment status

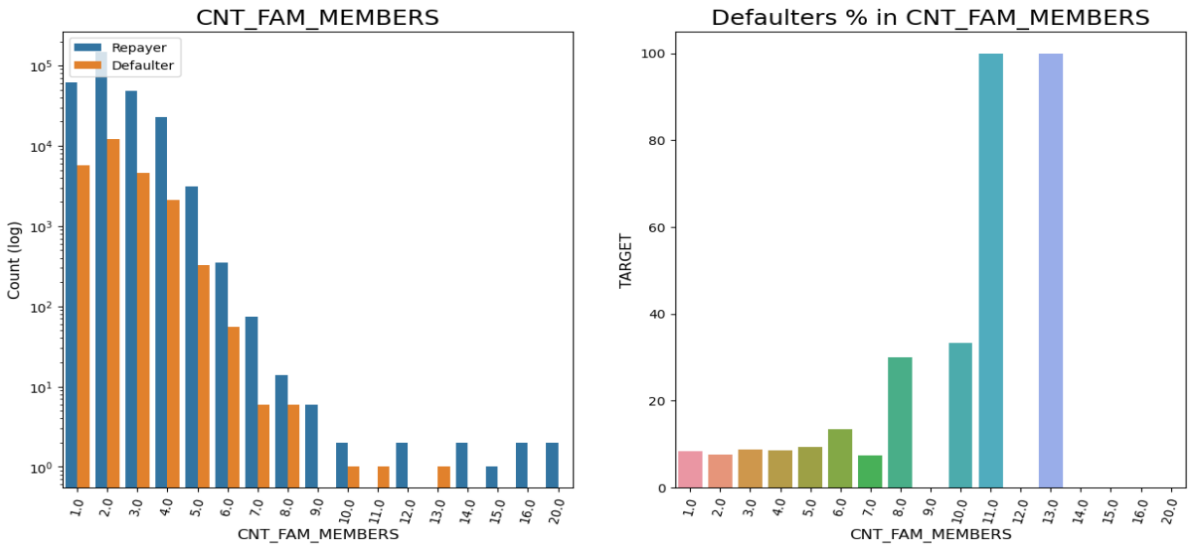
```
# checking number of children based on loan repayment status
univariate(app_data,"CNT_CHILDREN","TARGET",True,False,True)
```



The above plot shows that the majority of loan applicants do not have any children, while a small number of clients have more than three children. Notably, clients with more than four children have an alarmingly high default rate, with instances of 100% default observed for clients with child counts of 9 and 11. This suggests that having a large number of children may significantly impact a client's ability to repay their loans.

- Next, I'll be analyzing number of family members based on loan repayment status

```
# analyzing number of family members based on loan repayment status
univariate(app_data,"CNT_FAM_MEMBERS","TARGET",True,True,True)
```



The above plot stated that the the number of family members follows a similar pattern as children, where having more family members increases the risk of defaulting on loans. Clients with larger families tend to have a higher probability of defaulting compared to those with smaller families. This indicates that the size of the client's family can be a contributing factor to their loan repayment capacity.

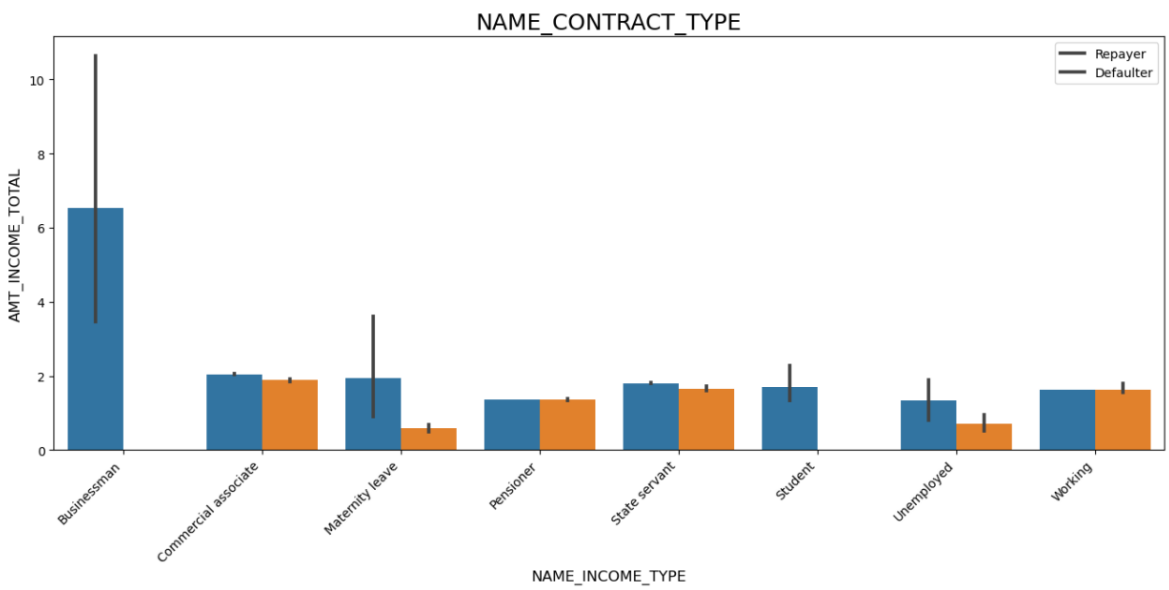
- Bivariate or Multivariate Analysis (categorical): The following code performs a group-by operation on the "app_data" DataFrame based on the "NAME_INCOME_TYPE" column. It then selects the "AMT_INCOME_TOTAL" column and calculates descriptive statistics for each group.

```
# categorical bivariate analysis
app_data.groupby('NAME_INCOME_TYPE')['AMT_INCOME_TOTAL'].describe()
```

NAME_INCOME_TYPE	count	mean	std	min	25%	50%	75%	max
Businessman	10.0	6.525000	6.272260	1.8000	2.250	4.9500	8.43750	22.5000
Commercial associate	71617.0	2.029553	1.479742	0.2655	1.350	1.8000	2.25000	180.0009
Maternity leave	5.0	1.404000	1.268569	0.4950	0.675	0.9000	1.35000	3.6000
Pensioner	55362.0	1.364013	0.766503	0.2565	0.900	1.1700	1.66500	22.5000
State servant	21703.0	1.797380	1.008806	0.2700	1.125	1.5750	2.25000	31.5000
Student	18.0	1.705000	1.066447	0.8100	1.125	1.5750	1.78875	5.6250
Unemployed	22.0	1.105364	0.880551	0.2655	0.540	0.7875	1.35000	3.3750
Working	158774.0	1.631699	3.075777	0.2565	1.125	1.3500	2.02500	1170.0000

Then using the above table, I'll plot a barplot for income type vs income amount range

```
# creating a barplot for income type vs income amount range
bivariate_c("NAME_INCOME_TYPE","AMT_INCOME_TOTAL",app_data,"TARGET",(16,6),['Repayer','Defaulter'])
```



Based on the data analysis, Businessmen tend to have the highest income compared to other income types. The estimated income range with a 95% confidence level suggests that the income of a Businessman could be somewhere around 4 lakhs or slightly above 10 lakhs. However, please note that this estimation is based on the available data and statistical calculations, and it's important to consider the limitations and potential variability in the data when interpreting these findings.

- Numerical Analysis: Bisecting the app_data dataframe based on the target value of 0 and 1 involves splitting the data into two separate subsets based on the target variable. The target variable typically represents the outcome or the variable of interest in a predictive or classification problem. In this case, the target variable has two distinct values: 0 and 1. The goal of bisecting the data is to separate the observations into two groups based on their target values. This separation allows for further analysis and comparison between the two groups.
- Firstly, I'll be listing all the columns present in app_data dataframe

```
# listing all the columnns of dataframe "app_data"
app_data.columns
```

```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE',
       'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS',
       'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH',
       'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'FLAG_MOBIL',
       'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE',
       'FLAG_EMAIL', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS',
       'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY',
       'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
       'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
       'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
       'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY',
       'ORGANIZATION_TYPE', 'OBS_30_CNT_SOCIAL_CIRCLE',
       'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE',
       'DEF_60_CNT_SOCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2',
       'FLAG_DOCUMENT_3', 'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5',
       'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8',
       'FLAG_DOCUMENT_9', 'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11',
       'FLAG_DOCUMENT_12', 'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14',
       'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17',
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
       'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR', 'AGE', 'AGE_GROUP', 'YEARS_EMPLOYED',
       'EMPLOYEMENT_YEARS', 'AMT_CREDIT_RANGE', 'AMT_GOODS_PRICE_RANGE',
       'AMT_INCOME_TOTAL_RANGE'],
      dtype='object')
```

- Next, I'll be bisecting the dataframe. The following code will be bisecting the app_data dataframe into two separate dataframes based on the target value of 0 and 1.

```
# bisecting the dataframe based on Target value 0 and 1 for correlation and analysis

correlation_col = ['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_REALTY',
                        'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
                        'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS',
                        'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED',
                        'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT',
                        'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
                        'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION',
                        'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'ORGANIZATION_TYPE',
                        'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_3',
                        'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
                        'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR']


# Repayers dataframe
df_repayer = app_data.loc[app_data['TARGET']==0, correlation_col]

# Defaulters dataframe
df_defaulter = app_data.loc[app_data['TARGET']==1, correlation_col]
```

The correlation_col variable contains a list of column names that will be used for the correlation and analysis. These columns represent various features or variables in the app_data dataframe.

The code then creates two dataframes: df_repayer and df_defaulter.

- ✓ df_repayer is the dataframe that contains observations where the target value is 0, representing the group of borrowers who have successfully repaid their loans.
- ✓ df_defaulter is the dataframe that contains observations where the target value is 1, representing the group of borrowers who have defaulted on their loans.

By splitting the data into these two separate dataframes, it becomes possible to analyze and compare the variables' relationship with the target variable (repayment status) within each group separately. This allows for a more focused analysis to understand how different variables may influence the repayment behavior differently for repayers and defaulters.

The subsequent analysis can include calculating correlation between the selected columns in each dataframe, examining descriptive statistics, visualizations, or applying machine learning models separately for repayers and defaulters.

Overall, this bisecting of the dataframe based on the target value allows for a more targeted and insightful analysis of the relationship between the variables and the repayment status of the borrowers.

- Next, I'll be calculating the top 10 correlation for the repayers data

```
# getting  top 10 correlation for the repayers dataframe

corr_repayer = df_repayer.corr()
corr_df_repayer = corr_repayer.where(np.triu(np.ones(corr_repayer.shape),k=1).astype(np.bool)).unstack().reset_index()
corr_df_repayer.columns =['VAR1','VAR2','Correlation']
corr_df_repayer.dropna(subset = ["Correlation"], inplace = True)
corr_df_repayer["Correlation"]=corr_df_repayer["Correlation"].abs()
corr_df_repayer.sort_values(by='Correlation', ascending=False, inplace=True)
corr_df_repayer.head(10)
```

	VAR1	VAR2	Correlation
64	AMT_GOODS_PRICE	AMT_CREDIT	0.987250
65	AMT_GOODS_PRICE	AMT_ANNUITY	0.776686
43	AMT_ANNUITY	AMT_CREDIT	0.771309
131	DAYS_EMPLOYED	DAYS_BIRTH	0.626114
42	AMT_ANNUITY	AMT_INCOME_TOTAL	0.418953
63	AMT_GOODS_PRICE	AMT_INCOME_TOTAL	0.349462
21	AMT_CREDIT	AMT_INCOME_TOTAL	0.342799
152	DAYS_REGISTRATION	DAYS_BIRTH	0.333151
174	DAYS_ID_PUBLISH	DAYS_EMPLOYED	0.276663
173	DAYS_ID_PUBLISH	DAYS_BIRTH	0.271314

The above performed the following operations:

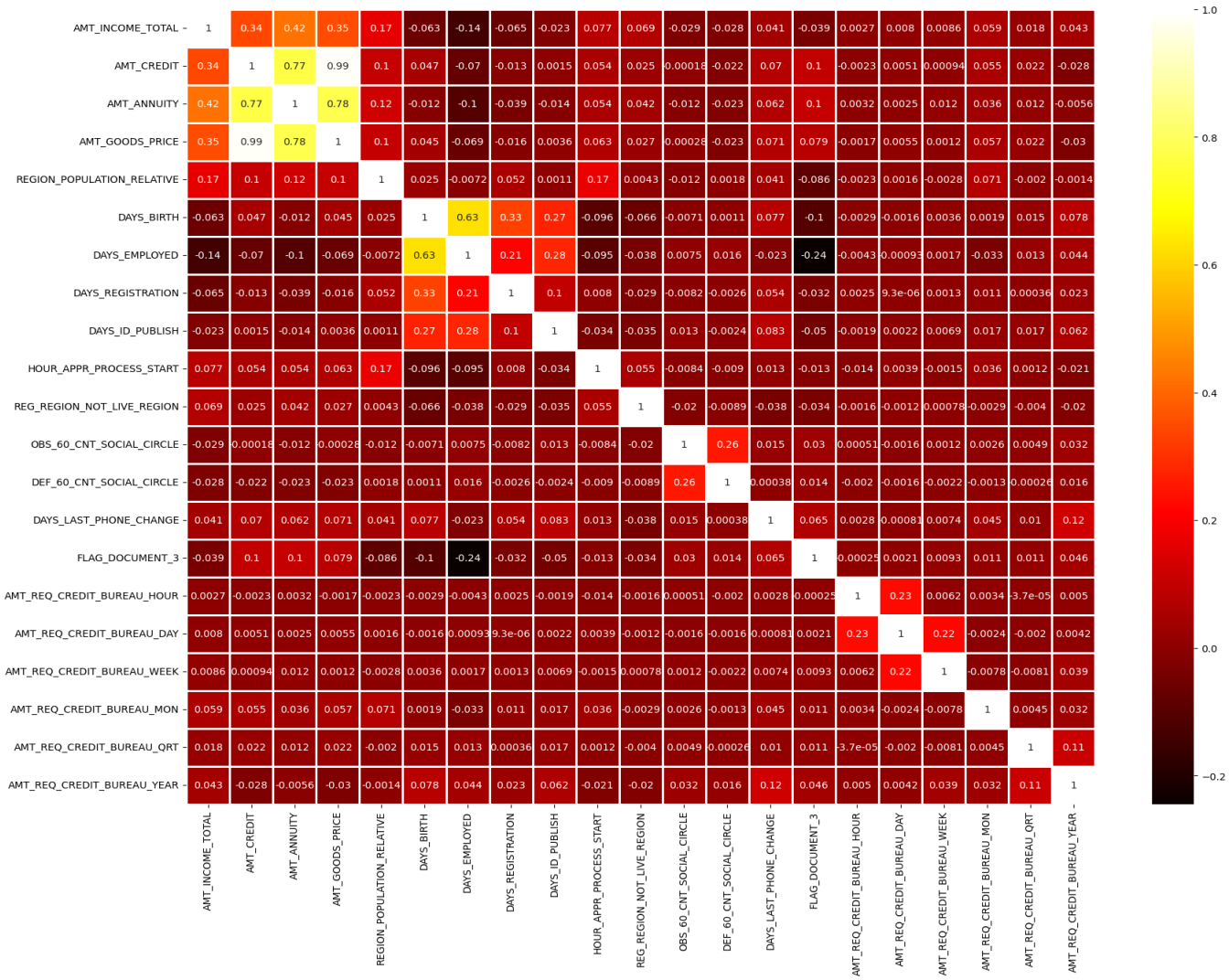
- ✓ corr_repayer = df_repayer.corr(): Calculates the correlation matrix for the dataframe df_repayer, which contains data for the repayers. The correlation matrix measures the linear relationship between pairs of variables in the dataframe.
- ✓ corr_df_repayer = corr_repayer.where(np.triu(np.ones(corr_repayer.shape),k=1).astype(np.bool)).unstack().reset_index(): Creates a new dataframe corr_df_repayer by extracting the upper triangle of the correlation matrix. The np.triu function generates a boolean mask to select the upper triangle, and unstack converts the matrix into a series. The resulting series is then reset as a dataframe with three columns: VAR1, VAR2, and Correlation.
- ✓ corr_df_repayer.columns =['VAR1','VAR2','Correlation']: Renames the columns of corr_df_repayer to 'VAR1', 'VAR2', and 'Correlation' to reflect the variables and their correlation values.
- ✓ corr_df_repayer.dropna(subset = ["Correlation"], inplace = True): Removes any rows from corr_df_repayer where the 'Correlation' column contains missing values (NaN).
- ✓ corr_df_repayer["Correlation"]=corr_df_repayer["Correlation"].abs(): Converts the correlation values in the 'Correlation' column of corr_df_repayer to absolute values, ensuring that all correlations are positive.

- ✓ corr_df_repayer.sort_values(by='Correlation', ascending=False, inplace=True): Sorts the rows of corr_df_repayer in descending order based on the 'Correlation' column, arranging the variables with the highest absolute correlation values at the top.
- ✓ corr_df_repayer.head(10): Displays the top 10 rows of corr_df_repayer, showing the variables with the highest absolute correlation values.

Overall, the code calculates and presents the top correlated pairs of variables in the df_repayer dataframe, providing insights into the relationships between different variables in the context of repayers.

- Next, I'll be plotting a heatmap to check linear correlation among the repayers

```
# plotting heatmap to see linear correlation amoung Repayers
fig = plt.figure(figsize=(20,15))
ax = sns.heatmap(df_repayer.corr(), cmap="hot",annot=True,linewidth =1)
```



In the analysis of repayers, several factors showed strong correlations

Credit amount: The credit amount taken by repayers showed a high positive correlation with the following factors:

- ✓ Goods price amount: The price of goods for which the credit was taken.
- ✓ Loan annuity: The regular installment amount to be paid towards the loan.
- ✓ Loan annuity: The regular installment amount to be paid towards the loan.

These correlations suggest that repayers who took higher credit amounts tend to have higher goods price amounts, loan annuities, and total incomes.

In the analysis of repayers, another notable correlation was observed in the number of days employed. This factor showed a strong positive correlation among repayers.

This suggests that repayers who have been employed for a longer duration tend to have a higher likelihood of repaying their loans. It indicates a potential relationship between stable employment and loan repayment behavior.

- Next, I'll be calculating top 10 correlation for the defaulter data

```
# getting the top 10 correlation for the defaulter data
corr_defaulter = df_defaulter.corr()
corr_defaulter = corr_defaulter.where(np.triu(np.ones(corr_defaulter.shape),k=1).astype(np.bool))
corr_df_defaulter = corr_defaulter.unstack().reset_index()
corr_df_defaulter.columns =['VAR1','VAR2','Correlation']
corr_df_defaulter.dropna(subset = ["Correlation"], inplace = True)
corr_df_defaulter["Correlation"]=corr_df_defaulter["Correlation"].abs()
corr_df_defaulter.sort_values(by='Correlation', ascending=False, inplace=True)
corr_df_defaulter.head(10)
```

	VAR1	VAR2	Correlation
64	AMT_GOODS_PRICE	AMT_CREDIT	0.983103
65	AMT_GOODS_PRICE	AMT_ANNUITY	0.752699
43	AMT_ANNUITY	AMT_CREDIT	0.752195
131	DAYS_EMPLOYED	DAYS_BIRTH	0.582185
152	DAYS_REGISTRATION	DAYS_BIRTH	0.289114
300	FLAG_DOCUMENT_3	DAYS_EMPLOYED	0.272169
263	DEF_60_CNT_SOCIAL_CIRCLE	OBS_60_CNT_SOCIAL_CIRCLE	0.264357
173	DAYS_ID_PUBLISH	DAYS_BIRTH	0.252863
351	AMT_REQ_CREDIT_BUREAU_DAY	AMT_REQ_CREDIT_BUREAU_HOUR	0.247511
174	DAYS_ID_PUBLISH	DAYS_EMPLOYED	0.229090

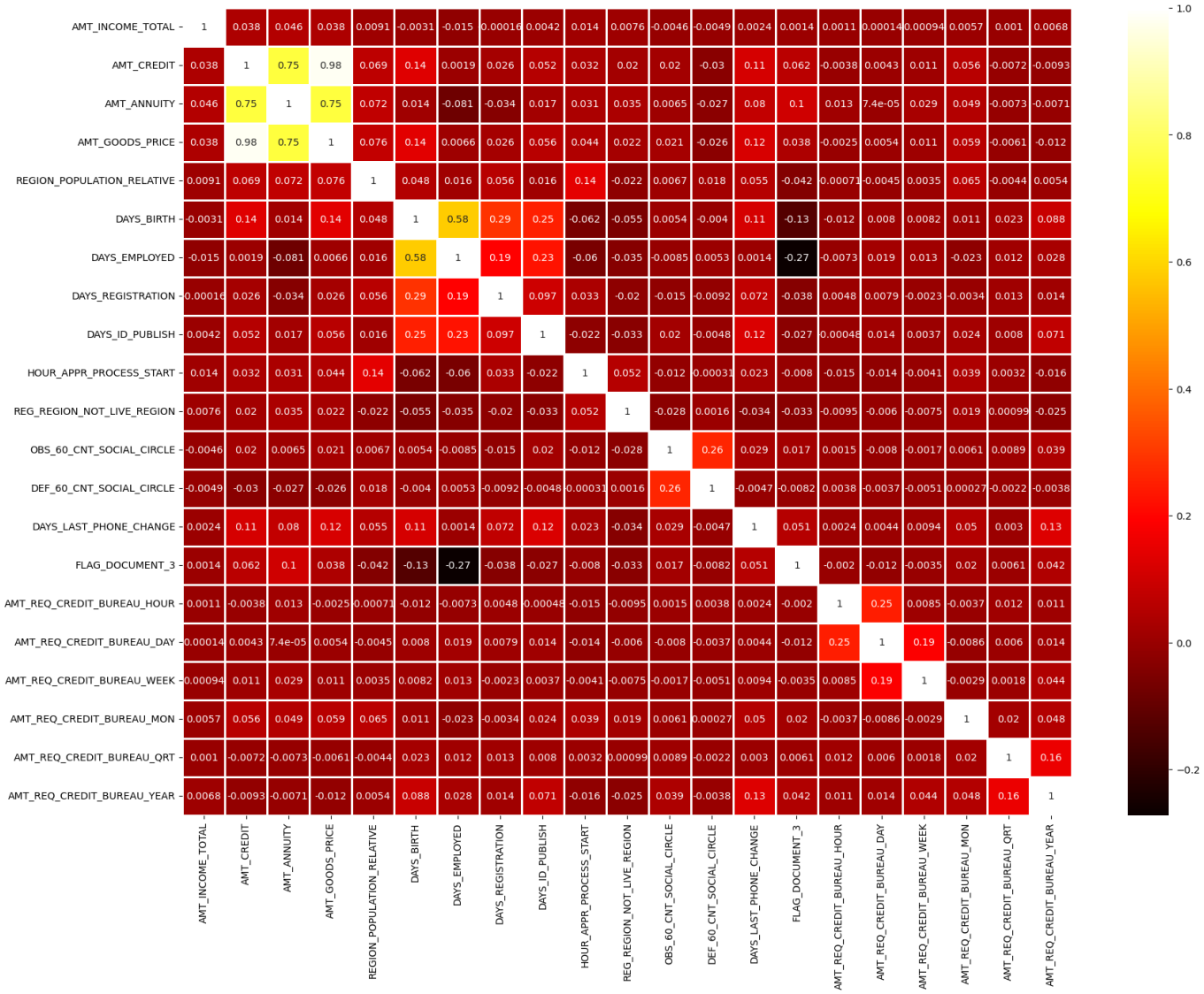
The code mentioned above is performing correlation analysis among the defaulters.

- ✓ the correlation matrix is calculated for the variables in the df_defaulter DataFrame using the corr() function.
- ✓ Created a boolean mask using np.triu() to select the upper triangular part of the correlation matrix. This is done to avoid duplicate correlation values since the correlation matrix is symmetric.
- ✓ Applied the boolean mask to the correlation matrix using where() to set the lower triangular part to NaN.
- ✓ Reshaped the correlation matrix using unstack() to convert it into a series.
- ✓ Removed rows with missing values in the 'Correlation' column using dropna().
- ✓ Took the absolute values of the 'Correlation' column using abs() to consider the magnitude of correlations regardless of their direction.
- ✓ Retrieved the top 10 rows of the sorted DataFrame using head(10).

This code allowed us to identify the pairs of variables among the defaulters that have the highest absolute correlation values. It helps in understanding the relationships and potential dependencies between different variables in the dataset.

- Next, I'll be plotting a heatmap to see correlation among the defaulters

```
# plotting heatmap to see correlation amoung defaulters
fig = plt.figure(figsize=(20,15))
ax = sns.heatmap(df_defaulter.corr(), cmap="hot",annot=True,linewidth =1)
```



The correlation analysis among the defaulters reveals the following findings:

- ✓ Credit amount is highly correlated with the goods price amount, similar to repayers.
- ✓ The correlation between loan annuity and credit amount has slightly reduced in defaulters (0.75) compared to repayers (0.77).
- ✓ Repayers have a higher correlation in the number of days employed (0.62) compared to defaulters (0.58).
- ✓ There is a significant drop in the correlation between the total income of the client and the credit amount (0.038) among defaulters, whereas it is 0.342 among repayers.

- ✓ The correlation between days_birth and the number of children has reduced to 0.259 in defaulters compared to 0.337 in repayers.
- ✓ There is a slight increase in the correlation between the defaulted and observed count in the social circle among defaulters (0.264) compared to repayers (0.254).

These correlations provide insights into the relationships between different variables for defaulters and highlight potential factors that contribute to loan default.

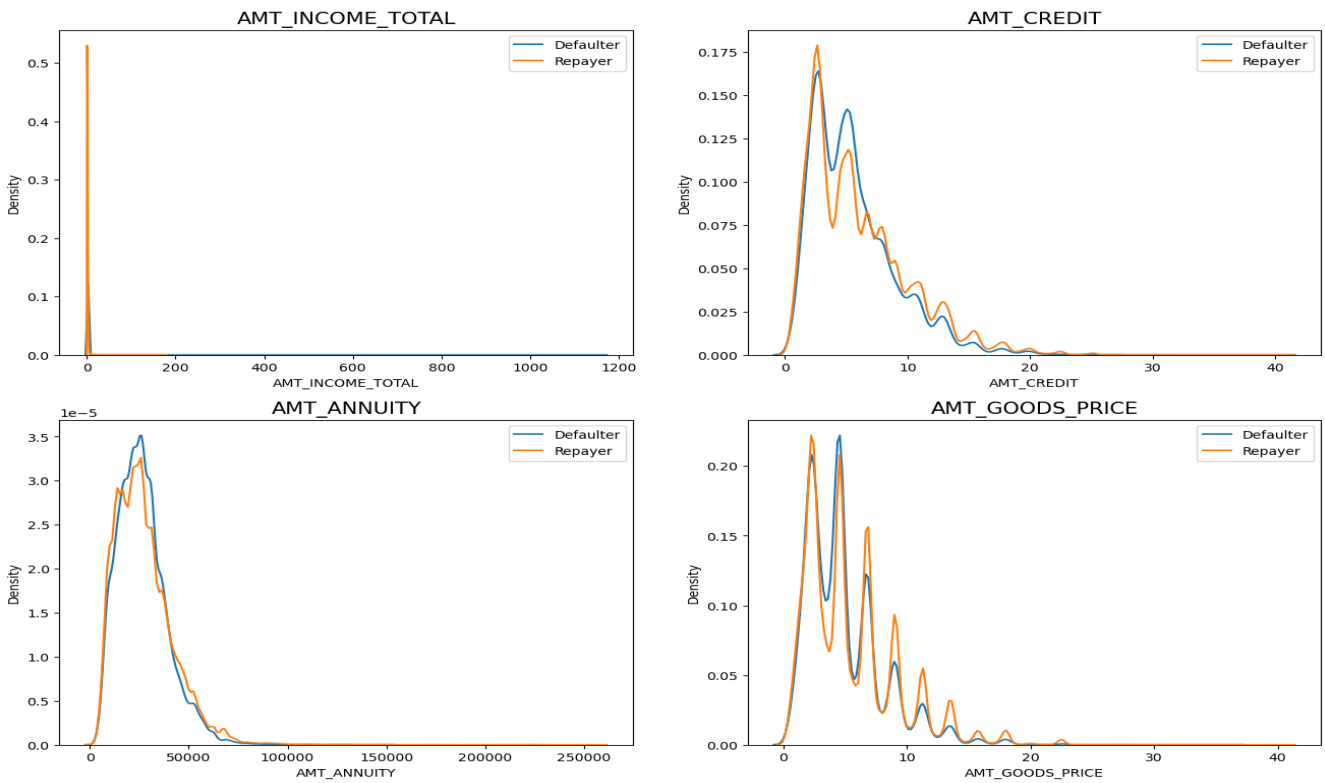
- Numerical Univariate Analysis: Numerical univariate analysis involves examining individual numerical variables in a dataset to gain insights into their distributions, central tendencies, and other statistical properties. Next, I'll be plotting the numerical columns related to amount as distribution plot to see density.

```
# plotting the numerical columns related to amount as distribution plot
amount = app_data[[ 'AMT_INCOME_TOTAL','AMT_CREDIT','AMT_ANNUITY', 'AMT_GOODS_PRICE']]

fig = plt.figure(figsize=(16,12))

for i in enumerate(amount):
    plt.subplot(2,2,i[0]+1)
    sns.distplot(df_defaulter[i[1]], hist=False,label ="Defaulter")
    sns.distplot(df_repayer[i[1]], hist=False, label ="Repayer")
    plt.title(i[1], fontdict={'fontsize' : 15, 'fontweight' : 5})
    plt.legend()


plt.show()
```

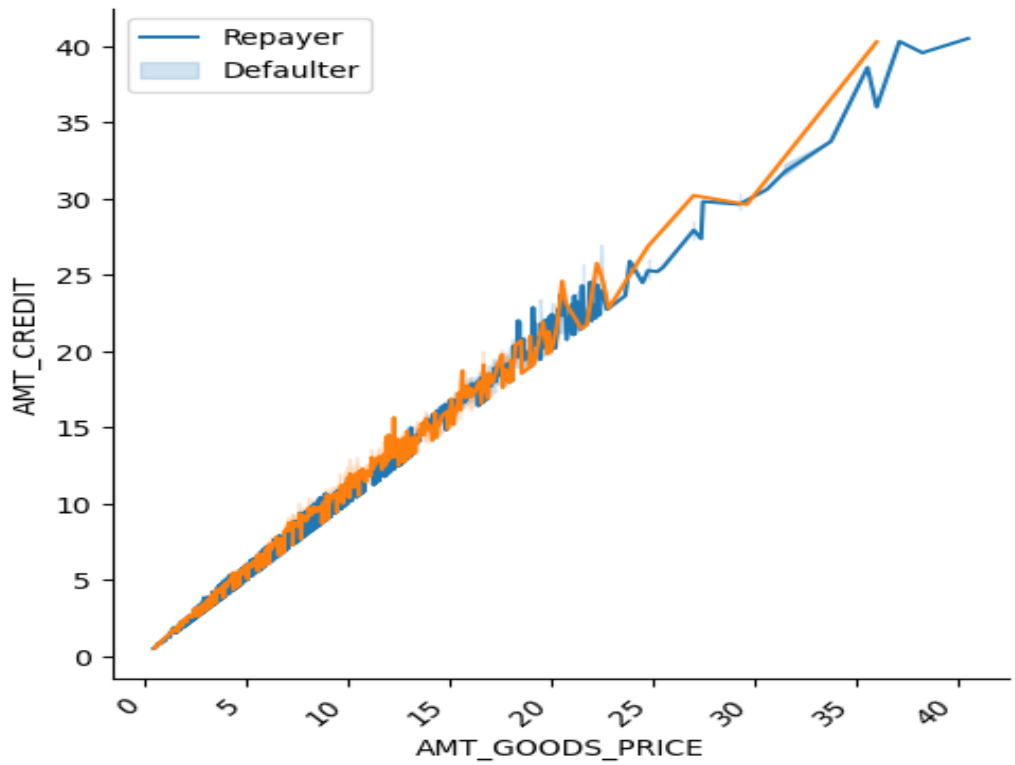


Based on the analysis, it is observed that a majority of loans are provided for goods with a price below 10 lakhs. Additionally, most people pay an annuity amount below 50,000 for their credit loans. The credit amount of the loans is predominantly less than 10 lakhs.

However, when examining the distributions of repayers and defaulters, it is noticed that they overlap in all the plots. This indicates that using these variables (goods price, annuity amount, and credit amount) in isolation is not sufficient to make accurate predictions or decisions regarding loan repayment. Other factors or variables should be considered to gain a more comprehensive understanding of loan repayment likelihood.

- Numerical Bivariate Analysis: Numerical bivariate analysis involves examining the relationship between two numerical variables. It helps us understand how changes in one variable are related to changes in another variable. This analysis can provide insights into correlations, trends, patterns, and dependencies between the variables.
- To check the relationship between the Goods price and credit amount, and compare it with the loan repayment status, we can perform a bivariate analysis. This analysis will help us understand if there is any association between these variables and how it relates to the loan repayment status.

```
# Numerical Bivariate Analysis
# Checking the relationship between Goods price and credit and comparing with loan repayment staus
bivariate_n('AMT_GOODS_PRICE','AMT_CREDIT',app_data,"TARGET", "line",['Repayer','Defaulter'])
```



In the bivariate analysis of credit amount and loan repayment status, it was observed that when the credit amount exceeds 30 Lakhs, there is an increase in the number of defaulters. This suggests that higher loan amounts may pose a higher risk of defaulting on the loan.

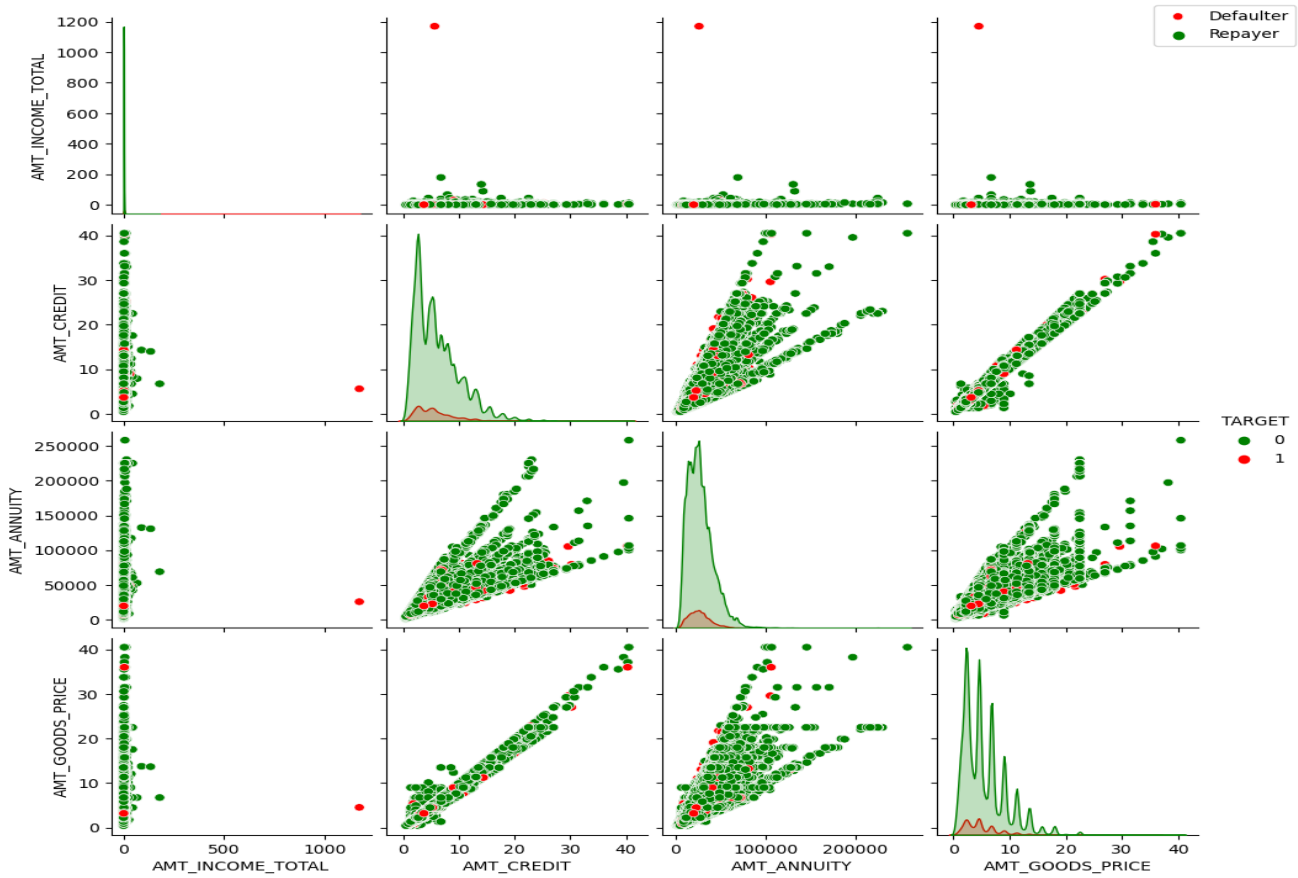
This finding highlights the importance of considering the credit amount as a factor in assessing the likelihood of loan default. Lending institutions should exercise caution when approving loans with larger amounts, as they may carry a higher risk of non-repayment.

By understanding the relationship between credit amount and loan repayment status, lenders can make more informed decisions and implement appropriate risk mitigation strategies to minimize the likelihood of default for higher-value loans.

- A pairplot was created to analyze the relationship between the "amount" variable and the loan repayment status. The pairplot allows for the visualization of multiple scatterplots, histograms, and kernel density estimates, providing a comprehensive view of the data.

```
# plotting pairplot between amount variable to draw reference against loan repayment status

amount = app_data[[ 'AMT_INCOME_TOTAL','AMT_CREDIT','AMT_ANNUITY', 'AMT_GOODS_PRICE','TARGET']]
amount = amount[(amount["AMT_GOODS_PRICE"].notnull()) & (amount["AMT_ANNUITY"].notnull())]
ax= sns.pairplot(amount,hue="TARGET",palette=["g","r"])
ax.fig.legend(labels=['Defaulter','Repayer'])
plt.show()
```



Based on the analysis, it can be observed that when the annuity amount exceeds 15,000 and the goods price amount is above 20 lakhs, the likelihood of defaulters decreases. This suggests that higher annuity payments and expensive goods indicate a lower risk of loan default.

Furthermore, the scatterplot between the loan amount (AMT_CREDIT) and goods price (AMT_GOODS_PRICE) reveals a strong positive correlation, as the majority of the data points are concentrated along a diagonal line. This implies that the loan amount and goods price are closely related, with higher loan amounts associated with higher goods prices.

Additionally, it is noteworthy that there are very few defaulters for loan amounts exceeding 20 lakhs. This suggests that larger loan amounts may have stricter approval criteria or are granted to borrowers with more stable financial situations, resulting in a lower default rate.

Overall, these findings provide insights into the relationship between annuity amount, goods price, loan amount, and loan default. They can be valuable in risk assessment and decision-making processes related to loan approvals and mitigating the risk of defaults.

- Next, I'll be merging both the dataframes under SK_ID_CURR

```
# merge both the dataframe on SK_ID_CURR with Inner Joins
loanapp = pd.merge(app_data, prev_app, how='inner', on='SK_ID_CURR')
```

```
loanapp.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CR
0	100002	1	Cash loans	M	N	Y	0	2.025	4
1	100003	0	Cash loans	F	N	N	0	2.700	12
2	100003	0	Cash loans	F	N	N	0	2.700	12
3	100003	0	Cash loans	F	N	N	0	2.700	12
4	100004	0	Revolving loans	M	Y	Y	0	0.675	1

- Checking for the types of columns present in the dataframe and then the number of rows and columns present in the dataframe

```
# checking the column types of the dataframe
loanapp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1413701 entries, 0 to 1413700
Columns: 111 entries, SK_ID_CURR to YEARLY_DECISION
dtypes: category(39), float64(30), int64(38), object(4)
memory usage: 839.9+ MB
```

```
# checking the number of rows and columns
loanapp.shape
```

```
(1413701, 111)
```

- Next, I'll be performing the bisecting operation on loanapp dataframe based on the values of the TARGET column.

```
# bisecting the new dataframe based on the value 0 and 1 od TARGET column for correlation and analysis

L0 = loanapp[loanapp['TARGET']==0] # Repayers
L1 = loanapp[loanapp['TARGET']==1] # Defaulters
```

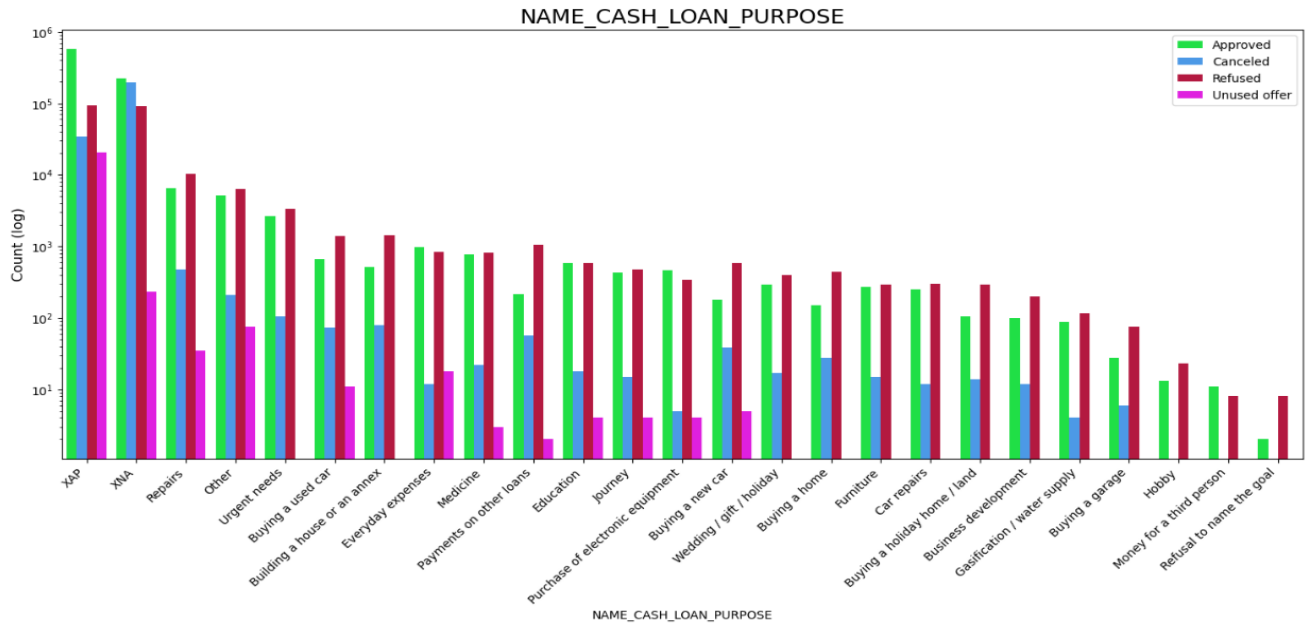
The above code is creating two new DataFrames, L0 and L1, by filtering the loanapp dataFrame based on the values in the TARGET column.

L0 represents the subset of data where the TARGET column is equal to 0, indicating repayers. This DataFrame contains all the rows from loanapp where the target variable is 0.

L1 represents the subset of data where the TARGET column is equal to 1, indicating defaulters. This DataFrame contains all the rows from loanapp where the target variable is 1.

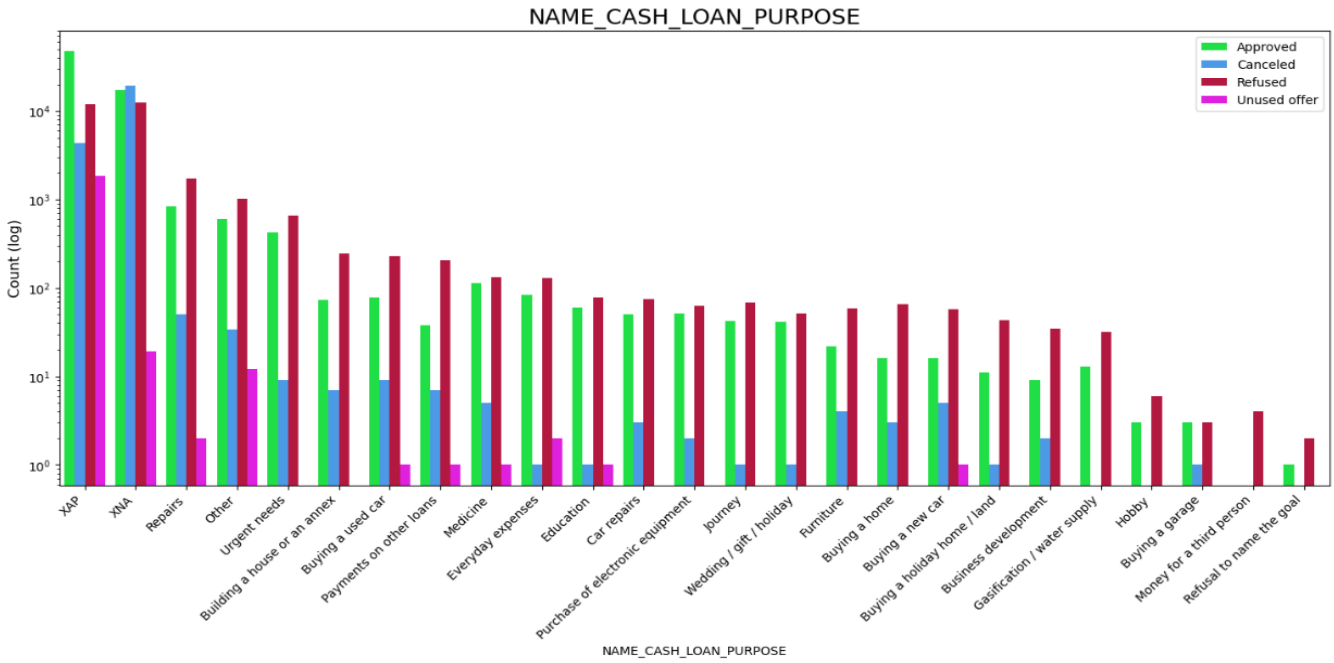
- Now that I have two separate DataFrames, it'll allow for separate analysis and comparison between repayers and defaulters, helping to understand any differences or patterns between the two groups.
- Therefore, I'll be creating a plot to check the contract status under the influence of the purpose of the loan for the repayers

```
# univariate plotting of contract status along with purpose of the loan.
univariate_c_merged("NAME_CASH_LOAN_PURPOSE",L0,"NAME_CONTRACT_STATUS",["#00ff33","#3399ff","#cc0033","#ff00ff"],True,(18,7))
```



- Similarly following the above process, I'll be plotting to check the relation between contract status and purpose of the loan for defaulters.

```
# univariate plotting of contract status along with purpose of the loan for defaulters
univariate_c_merged("NAME_CASH_LOAN_PURPOSE",L1,"NAME_CONTRACT_STATUS",["#00ff33","#3399ff","#cc0033","#ff00ff"],True,(18,7))
```



The loan data analysis reveals that the loan purpose variable has a significant number of unknown values, represented by "XAP" and "XNA". This indicates a lack of specific information about the purpose of the loan for these cases.

Furthermore, among the known loan purposes, loans taken for the purpose of Repairs have the highest default rate. This suggests that borrowers who take loans for repair purposes are more likely to default on their payments.

Additionally, a large number of loan applications for Repair or Other purposes have been rejected by the bank or refused by the clients themselves. This implies that the bank considers loans for repair purposes to be high-risk. It is possible that these loan applications were either declined due to the perceived risk or offered with high-interest rates, which made them unfeasible for the clients. As a result, many clients refused to accept the loan offer.

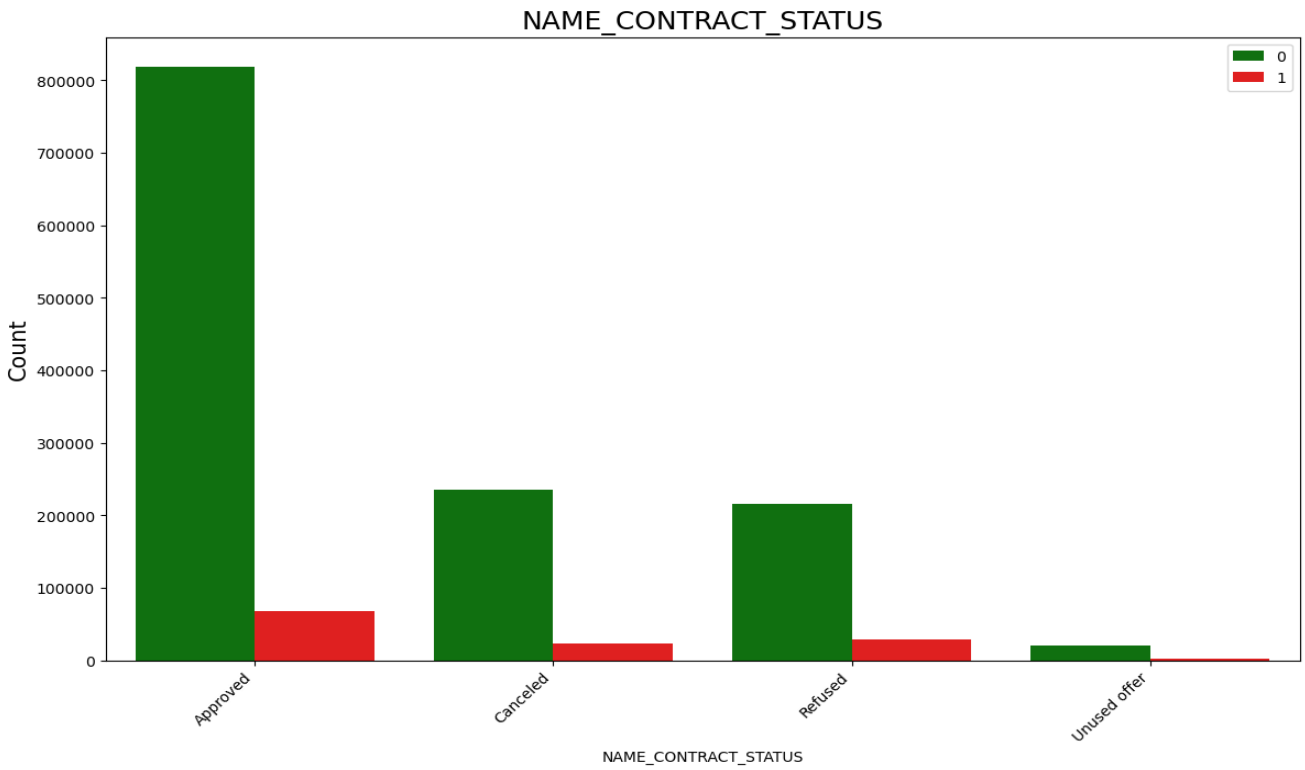
Overall, these findings highlight the importance of considering the loan purpose and the associated default rates, as well as the bank's caution and the clients' hesitance towards repair-related loans.

- Next, I'll be performing a univariate analysis on the "NAME_CONTRACT_STATUS" variable in the loanapp dataframe, specifically comparing it with the "TARGET" variable.

```
# checking contract status based on loan repayment status to identify if there's any loss
univariate_c_merged("NAME_CONTRACT_STATUS",loanapp,"TARGET",['g','r'],False,(14,8))
r = loanapp.groupby("NAME_CONTRACT_STATUS")["TARGET"]
df1 = pd.concat([r.value_counts(),round(r.value_counts(normalize=True).mul(100),2)],axis=1, keys=('Counts','Percentage'))
df1['Percentage'] = df1['Percentage'].astype(str) + "%"
df1
```

The resulting plot shows the distribution of contract statuses for repayers (green) and defaulters (red), with bars representing the counts of each category.

After grouping the loanapp dataframe by "NAME_CONTRACT_STATUS" and "TARGET", the code creates a new dataframe, df1, which includes the counts and percentages of each contract status category for both repayers and defaulters.



NAME_CONTRACT_STATUS	TARGET	Counts	Percentage
Approved	0	818856	92.41%
	1	67243	7.59%
Canceled	0	235641	90.83%
	1	23800	9.17%
Refused	0	215952	88.0%
	1	29438	12.0%
Unused offer	0	20892	91.75%
	1	1879	8.25%

Based on the analysis, it can be observed that 90% of clients who had their loan previously cancelled have actually repaid the loan in the current case. This suggests that revising the interest rates for these clients could create more business opportunities, as they have

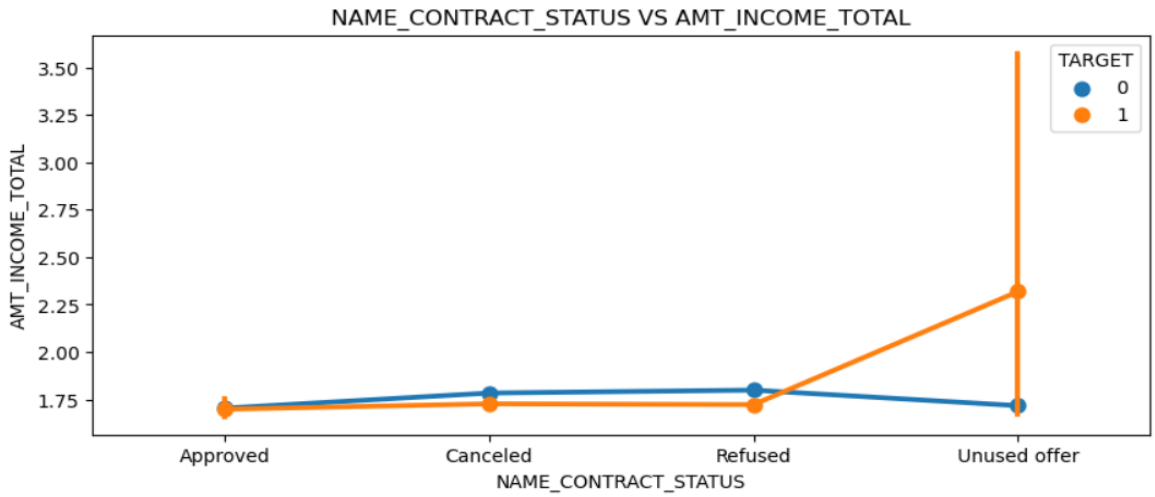
demonstrated their ability to repay.

Furthermore, 88% of clients who were previously refused a loan have successfully paid back the loan in the current case. This highlights the importance of recording the refusal reason for further analysis. Understanding the reasons behind the previous loan refusals can help identify patterns or factors that may indicate potential repaying customers in the future.

Therefore, it is recommended to record the refusal reason for analysis purposes, as it can provide valuable insights and help identify potential opportunities to engage with clients who were previously refused a loan.

- Next I'll be using the pointplot operation to find out the relation between income total and contract status

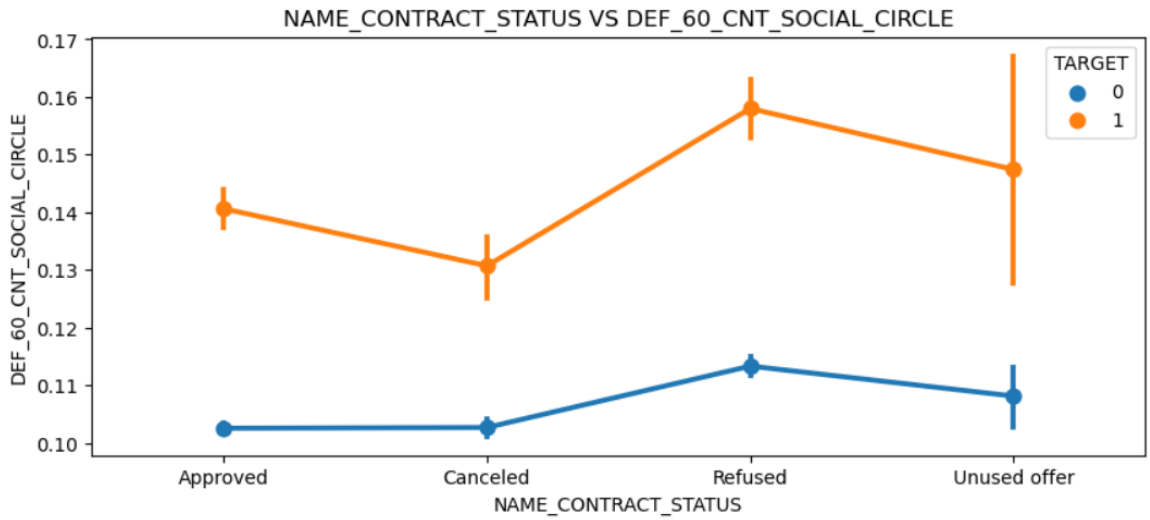
```
# checking the relationship between income total and contract status
pointplot(loanapp,"TARGET","NAME_CONTRACT_STATUS",'AMT_INCOME_TOTAL')
```



The point plot reveals an interesting pattern where individuals who have not utilized an offer earlier have a higher default rate, despite having a higher average income compared to others. This suggests that factors other than income may play a significant role in loan defaulting behaviour, and the previous utilization of offers might be one such influential factor. Further analysis is needed to understand the underlying reasons behind this relationship and to develop appropriate strategies for mitigating default risk among individuals who have not previously utilized offers.

- Similarly, I'll try to find out the relation between people who have defaulted in last 60 days being in client's social circle and contract status

```
# checking the relationship between people who defaulted in last 60 days being in client's social circle and contact status
pointplot(loanapp,"TARGET","NAME_CONTRACT_STATUS",'DEF_60_CNT_SOCIAL_CIRCLE')
```



The analysis reveals that clients who have an average DEF_60_CNT_SOCIAL_CIRCLE score of 0.13 or higher tend to have a higher default rate. This suggests that examining a client's social circle, which includes their social connections and interactions, could be beneficial in assessing the risk of loan default. By considering the social circle as a potential factor in the loan disbursement process, lenders can gain additional insights into a client's financial stability and make more informed decisions regarding loan approvals and interest rates.

5. Tech Stack used

- Jupyter Notebook
- MS Word 2019
- Google Drive

6. Insights

By attempting this project, I understood the real-life analysis process that goes into making some crucial decisions by the banks. Before going through this project, I had some ametuer experience in python for data analysis but after doing this project, I became more confident on applying my python skills for the data analysis process.

The project helped me understand various statistical words and theories that I was unaware of. For eg. Univairate, Bivariate and Multivariate analysis. I do had some idea about the terms but never knew that they play such a vital role for lending loans by the banks.

I have to admit that I became a bit confused while attempting to remove the unwanted columns because the amount of columns present in both the datasets were huge in number. And because of this, I had a constant fear of deleting the important ones. Thankfully, I took some time and understood the importance of each and every column and later this process helped me removing the unwanted column from the data.

It would be an unfair practice if I don't mention the sites that helped me throughout this process. I have listed down the links from where I grasped the knowledge and then used them in attempting the project:

a. https://www.freecodecamp.org/learn/data-analysis-with-python/data-analysis-with-python-course/introduction-to-data-analysis
b. https://www.w3schools.com/python/default.asp
c. https://www.geeksforgeeks.org/python-programming-examples/

7. Conclusion

The analysis of the datasets revealed key attributes that enable the bank to identify whether clients are likely to repay their loans. Factors such as employment history, income level, loan amount, credit history, age, education level, family status, housing type, and organization type all play a role in determining loan repayment likelihood. By considering these factors and categorizing clients based on their attributes, the bank can make more accurate assessments of loan repayment risk. This analysis empowers the bank to optimize loan approvals, offer competitive interest rates, and minimize default risks, leading to improved loan portfolio performance. The following is the pointwise analysis details that has been concluded with certain suggestions:

- Based on the analysis, several factors have emerged as decisive in determining whether an applicant will be a repayer or not. These factors include the applicant's education level, income type, region rating, organization type, age, employment history, income level, loan purpose, and number of children.
- Clients with an academic degree exhibit fewer defaults, while students and businessmen show no defaults. A region rating of 1 indicates a safer profile. Certain organization types, such as trade types 4 and 5 and industry type 8, have default rates below 3%. Clients above the age of 50 are less likely to default, and those with 40+ years of experience have a default rate of less than 1%. Higher income levels, particularly exceeding 700,000, are associated with a lower likelihood of default. Loans taken for hobbies or buying a garage tend to have higher repayment rates. Clients with zero to two children are more likely to repay their loans.
- Men tend to have a relatively higher default rate, and individuals who are single or in a civil marriage also exhibit higher default rates. Those with lower secondary and secondary education levels are more likely to default, as well as individuals who are on maternity leave or unemployed. Living in regions with a rating of 3 is associated with the highest default rates.
- Certain occupation types, such as low-skill laborers, drivers, waiters/barmen staff, security staff, laborers, and cooking staff, have significantly higher default rates and should be avoided for loan approval. Specific organization types, such as transport type 3, industry type 13, industry type 8, and restaurants, also show high

default rates. Self-employed individuals have a relatively high risk of defaulting.

- Younger individuals in the age group of 20-40 have a higher probability of defaulting, and those with less than five years of employment also exhibit a higher default rate. Clients with a high number of children, particularly nine or more, have a 100% default rate, indicating that their applications should be rejected.
- Finally, there is an increase in defaulters when the credit amount exceeds 3 lakhs.

To mitigate the risk of defaults and potential business losses, loans can be offered with higher interest rates based on several factors.

- Firstly, applicants residing in rented apartments or living with parents represent a significant portion of loan applications, making it essential to impose higher interest rates to mitigate losses if they default.
- Secondly, individuals seeking loans in the range of 3-6 lakhs exhibit higher default rates, warranting higher interest rates specifically for this credit range.
- Moreover, considering that approximately 90% of applicants have a total income below 3 lakhs and pose a higher default probability, offering loans to this income category with higher interest rates can help mitigate default risks.
- Clients with 4 to 8 children also demonstrate a considerably higher default rate, making it crucial to impose higher interest rates on their loans to mitigate associated risks.
- Additionally, loans taken for repair purposes show the highest default rates, and a significant number of applications with repair or other purposes have been rejected or refused by clients in the past. This highlights the need to implement stricter criteria or higher interest rates for repair loans to mitigate default risks and avoid potential business losses.

By considering these factors and tailoring interest rates accordingly, lenders can better manage default risks and enhance their loan portfolio performance.

8. Drive Link

https://drive.google.com/drive/folders/1AEFxXsvsETSJQzNKmp5Stjwms8dqSpGd

Folder name: Bank Loan Case Study

Name of the report file: project_6

Name of the python file: bank_loan_case.ipynb