Project 6
Bank Loan Case Study

Introduction

This case study aims to give you an idea of applying EDA in a real business scenario. In this case study, apart from applying the techniques that you have learnt in the EDA module, you will also develop a basic understanding of risk analytics in banking and financial services and understand how data is used to minimise the risk of losing money while lending to customers.

Business Understanding

The loan providing companies find it hard to give loans to the people due to their insufficient or non-existent credit history. Because of that, some consumers use it as their advantage by becoming a defaulter. Suppose you work for a consumer finance company which specialises in lending various types of loans to urban customers. You have to use EDA to analyse the patterns present in the data. This will ensure that the applicants capable of repaying the loan are not rejected.

When the company receives a loan application, the company has to decide for loan approval based on the applicant's profile. Two types of risks are associated with the bank's decision:

- If the applicant is likely to repay the loan, then not approving the loan results in a loss of business to the company

- If the applicant is not likely to repay the loan, i.e., he/she is likely to default, then approving the loan may lead to a financial loss for the company.

The data given below contains the information about the loan application at the time of applying for the loan. It contains two types of scenarios:

- The client with payment difficulties: he/she had late payment more than X days on at least one of the first Y instalments of the loan in our sample,

- All other cases: All other cases when the payment is paid on time.

When a client applies for a loan, there are four types of decisions that could be taken by the client/company):

Approved: The Company has approved loan Application

Cancelled: The client cancelled the application sometime during approval. Either the client changed her/his mind about the loan or in some cases due to a higher risk of the client he received worse pricing which he did not want.

Refused: The company had rejected the loan (because the client does not meet their requirements etc.).

Unused offer: Loan has been cancelled by the client but on different stages of the process.

In this case study, you will use EDA to understand how consumer attributes and loan attributes influence the tendency of default.

Data Understanding

This dataset has 3 files as explained below:

1. **'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.

2. **'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.

3. **'columns_description.csv'** is data dictionary which describes the meaning of the variables.

Results Expected by Learners

- **Present** the overall approach of the analysis in a presentation. Mention the problem statement and the analysis approach briefly.

- **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: How will you analyse the data in case of data **imbalance**? You can plot more than one type of plot to analyse the different aspects due to data imbalance. For example, you can choose your own scale for the graphs, i.e., one can plot in terms of percentage or absolute value. Do this analysis for the 'Target variable' in the dataset (clients with payment difficulties and all other cases). Use a mix of univariate and bivariate analysis etc.

- 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 **visualisations** and summarise 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.

You need to submit one/two Python notebook which clearly explains the thought process behind your analysis (either in comments of markdown text), code and relevant plots. The presentation file needs to be in PDF format and should contain the points discussed above with the necessary visualisations. Also, all the visualisations and plots must be done in

Python(should be present in the Python notebook), though they may be recreated in Tableau for better aesthetics in the PPT file.

IMPORTING ALL THE NECCESSARY MODULES

```
#importing all the important libraries like numpy. pandas, matlplolib, 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
```

```
# to suppress warnings

import warnings
warnings.filterwarnings("ignore")
```

```
#notebook setting to display all the rowns and columns to have better clearity on the data.

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
pd.set_option('display.expand_frame_repr', False)
```

Dataset 1 - "application_data.csv"

Reading and understanding the data

Importing the dataset

```
# importing application_data.csv

appl_data = pd.read_csv("application_data.csv")
```

Understanding the dataset

```
#checking the rows and columns of the raw dataset

appl_data.shape
```

(307511, 122)

```
appl_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

INSIGHT

- **307511 rows and 122 columns are present.**

- **columns with days with negative and positive values are present.**
- **There are columns with extremely high values, including Amount-related columns (Price). Standardization is necessary.**

Data Cleaning & Manipulation

Null Values

```
#checking how many null values are present in each of the columns

#creating a function to find null values for the dataframe
def null_values(df):
    return round((df.isnull().sum()*100/len(df)).sort_values(ascending = False),2)
```

```
null_values(appl_data)
```

```
COMMONAREA_MEDI               69.87
COMMONAREA_AVG                69.87
COMMONAREA_MODE               69.87
NONLIVINGAPARTMENTS_MODE      69.43
NONLIVINGAPARTMENTS_MEDI      69.43
NONLIVINGAPARTMENTS_AVG       69.43
FONDKAPREMONT_MODE            68.39
LIVINGAPARTMENTS_MEDI         68.35
LIVINGAPARTMENTS_MODE         68.35
LIVINGAPARTMENTS_AVG          68.35
FLOORSMIN_MEDI                67.85
FLOORSMIN_MODE                67.85
FLOORSMIN_AVG                 67.85
YEARS_BUILD_MEDI              66.50
YEARS_BUILD_AVG               66.50
YEARS_BUILD_MODE              66.50
OWN_CAR_AGE                   65.99
LANDAREA_MODE                 59.38
LANDAREA_AVG                  59.38
LANDAREA_MEDI                 59.38
BASEMENTAREA_MEDI             58.52
BASEMENTAREA_AVG              58.52
BASEMENTAREA_MODE             58.52
EXT_SOURCE_1                  56.38
NONLIVINGAREA_MEDI            55.18
NONLIVINGAREA_AVG             55.18
NONLIVINGAREA_MODE            55.18
ELEVATORS_MODE                53.30
ELEVATORS_AVG                 53.30
ELEVATORS_MEDI                53.30
```

Dealing with Null values more than 50 %

```
#creating a variable null_col_50 for storing null columns having missing values more than 50%

null_col_50 = null_values(appl_data)[null_values(appl_data)>50]
```

```
#reviewing null_col_50

print(null_col_50)
print()
print("Num of columns having missing values more than 50% :",len(null_col_50))
```

```
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
```

INSIGHT

- **41 columns have null values more than 50% which are related to different area sizes on apartment owned/rented by the loan applicant**

```
null_col_50.index  # Will drop all these columns
```

```
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'], dtype='object')
```

```
# Now lets drop all the columns having missing values more than 50% that is 41 columns

appl_data.drop(columns = null_col_50.index, inplace = True)
```

```
appl_data.shape  # Now there are 81 columns remaining
```

```
(307511, 81)
```

- **After dropping 41 columns there are 81 columns left.**

Dealing with null values more than 15%

```
# now we will deal with null values more than 15%

null_col_15 = null_values(appl_data)[null_values(appl_data)>15]
```

```
null_col_15
```

```
FLOORSMAX_AVG                  49.76
FLOORSMAX_MODE                 49.76
FLOORSMAX_MEDI                 49.76
YEARS_BEGINEXPLUATATION_AVG    48.78
YEARS_BEGINEXPLUATATION_MODE   48.78
YEARS_BEGINEXPLUATATION_MEDI   48.78
TOTALAREA_MODE                 48.27
EMERGENCYSTATE_MODE            47.40
OCCUPATION_TYPE                31.35
EXT_SOURCE_3                   19.83
dtype: float64
```

- **From the columns dictionary it can be concluded that only 'OCCUPATION_TYPE', 'EXT_SOURCE_3 is relevant to TARGET column. Thus we will drop all other columns except 'OCCUPATION_TYPE','EXT_SOURCE_3**

```
#removing 'OCCUPATION_TYPE', 'EXT_SOURCE_3' from "null_col_15" so that we can drop all other at once.

null_col_15.drop(["OCCUPATION_TYPE","EXT_SOURCE_3"], inplace = True)
```

```
print(null_col_15)
print()
print("No of columns having missing values more than 15% and are not reletable:",len(null_col_15))
```

```
FLOORSMAX_AVG                  49.76
FLOORSMAX_MODE                 49.76
FLOORSMAX_MEDI                 49.76
YEARS_BEGINEXPLUATATION_AVG    48.78
YEARS_BEGINEXPLUATATION_MODE   48.78
YEARS_BEGINEXPLUATATION_MEDI   48.78
TOTALAREA_MODE                 48.27
EMERGENCYSTATE_MODE            47.40
dtype: float64

No of columns having missing values more than 15% and are not reletable: 8
```

```
#thus removing columns having missing values more than 15% and which are not reletable to TARGET column.

appl_data.drop(null_col_15.index,axis=1, inplace = True)
```

```
appl_data.shape  # After dropping null_col_15, we have left with 73 columns
```

```
(307511, 73)
```

- **After dropping 8 columns there are 73 columns left**
- **There are 2 more Columns with values missing more than 15%**

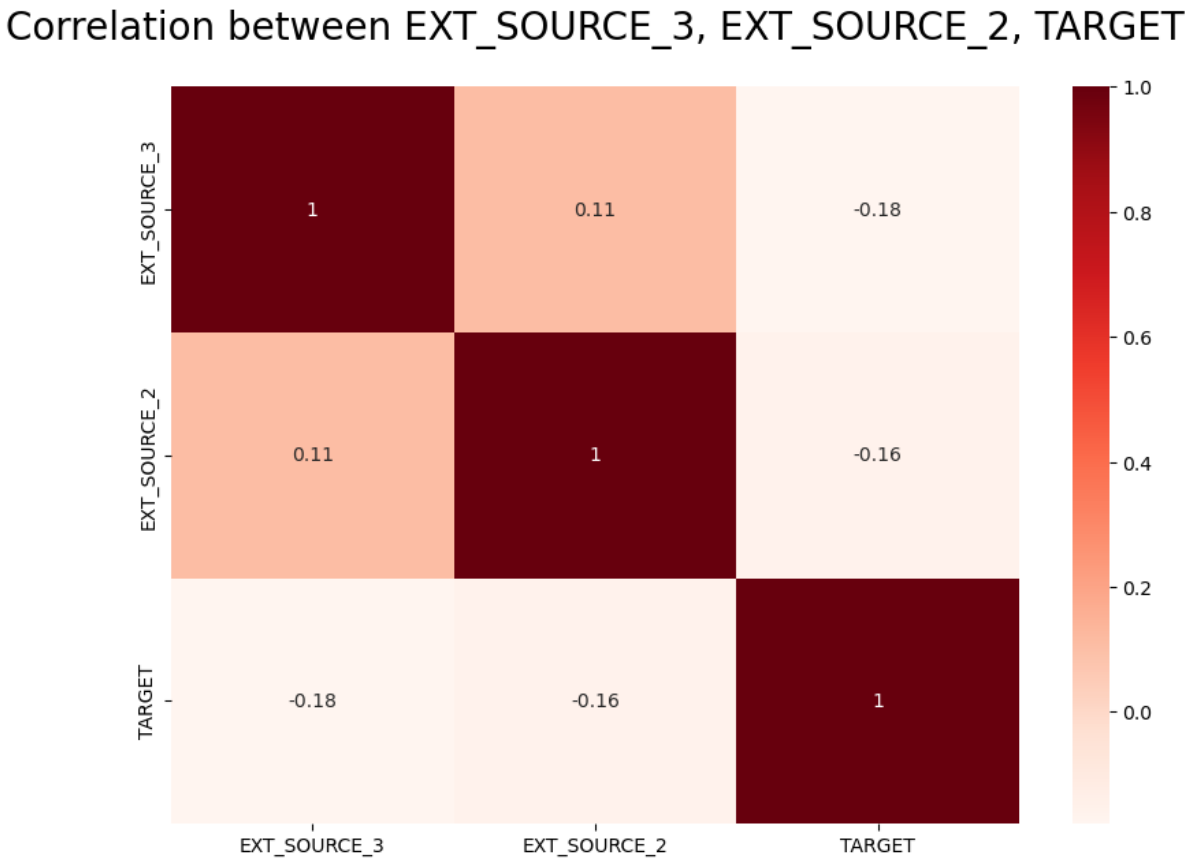
```
null_values(appl_data).head(10)
```

```
OCCUPATION_TYPE              31.35
EXT_SOURCE_3                 19.83
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
OBS_30_CNT_SOCIAL_CIRCLE      0.33
dtype: float64
```

Analyse & Removing Unnecessary Columns

```
irrev = ["EXT_SOURCE_3","EXT_SOURCE_2"]  # putting irrevlent columns in varibale "irrev"
```

```
plt.figure(figsize= [10,7])

sns.heatmap(appl_data[irrev+["TARGET"]].corr(), cmap="Reds",annot=True)

plt.title("Correlation between EXT_SOURCE_3, EXT_SOURCE_2, TARGET", fontdict={"fontsize":20}, pad=25)
plt.show()
```



- **There seems to be no linear correlation.**
- **Also, from columns description we decided to remove these columns.**

Dropping above columns

```
#dropping above columns as decided

appl_data.drop(irrev, axis=1, inplace= True)
```

```
appl_data.shape  # Now we are left with 71 columns
```

```
(307511, 71)
```

```
null_values(appl_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
```

Now we will check columns with FLAGS and their relation with TARGET columns to remove irrelevant columns

For this we will create a data frame containing all FLAG columns and then plot bar graphs for each column with respect to TARGET column where "0" will represent as Repayer and "1" will represent as Defaulter

```
# adding all flags coloumns in variable "flag_columns"

flag_columns = [col for col in appl_data.columns if "FLAG" in col]

flag_columns  # Viewing all FLAG columns
```

```
['FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'FLAG_MOBIL',
 'FLAG_EMP_PHONE',
 'FLAG_WORK_PHONE',
 'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 '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']
```

Creating flag_df dataframe having all FLAG columns and TARGET column

```
# creating flag_df dataframe having all FLAG columns and TARGET column

flag_df = appl_data[flag_columns+["TARGET"]]
```

replacing "0" as Repayer and "1" as defaulter for TARGET column

```
# replacing "0" as repayer and "1" as defaulter for TARGET column

flag_df["TARGET"] = flag_df["TARGET"].replace({1:"Defaulter", 0:"Repayer"})
```

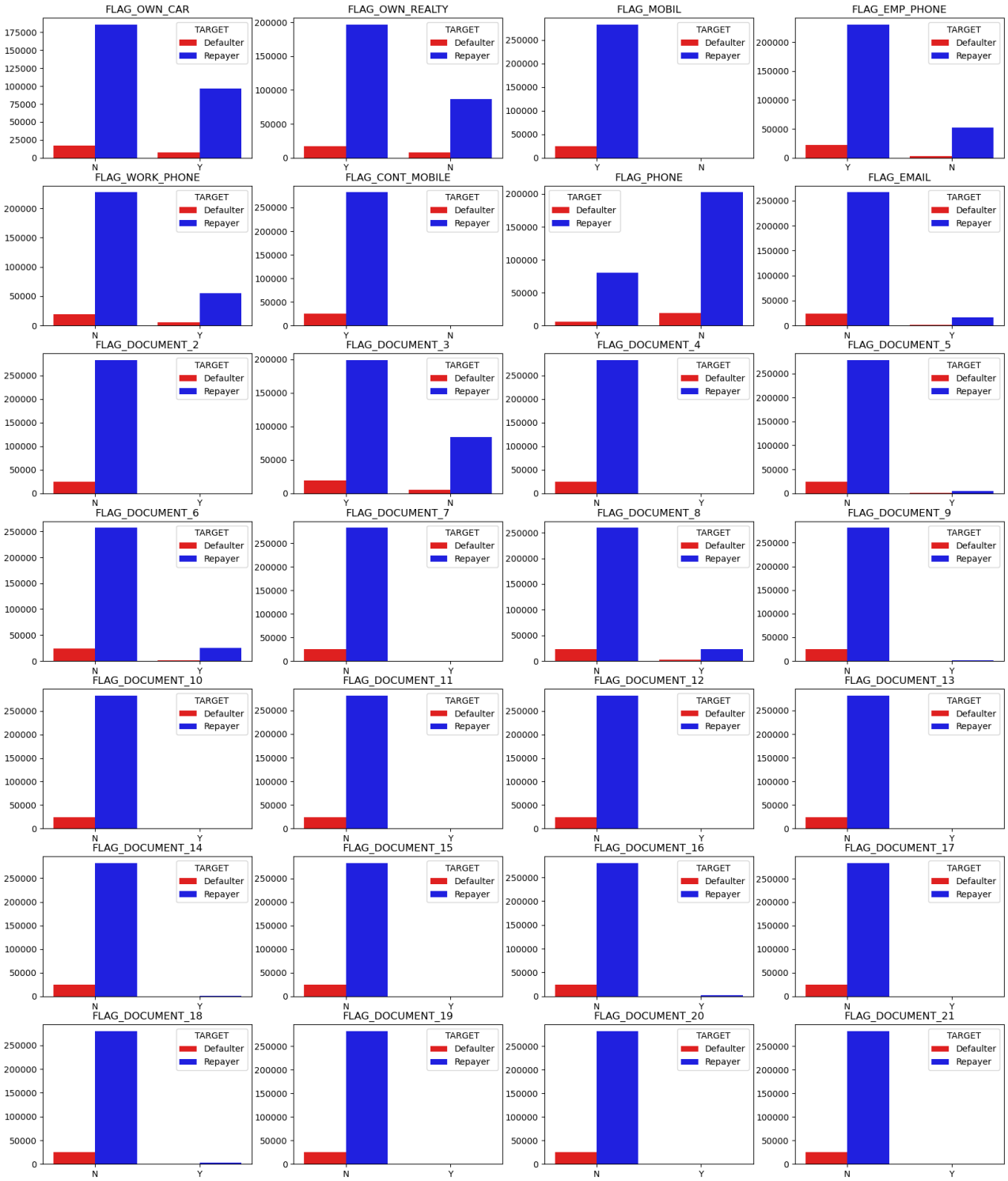
As stated in column description replacing "1" as Y being TRUE and "0" as N being False

```
# as stated in columnn description replacing "1" as Y being TRUE and "0" as N being False

for i in flag_df:
    if i!= "TARGET":
        flag_df[i] = flag_df[i].replace({1:"Y", 0:"N"})
```

```
flag_df.head()
```

	FLAG_OWN_CAR	FLAG_OWN_REALTY	FLAG_MOBIL	FLAG_EMP_PHONE	FLAG_WORK_PHONE	FLAG_CONT_MOBILE	FLAG_PHONE	FLAG_EMAIL	FLAG_
0	N	Y	Y	Y	N	Y	Y	N	
1	N	N	Y	Y	N	Y	Y	N	
2	Y	Y	Y	Y	Y	Y	Y	N	
3	N	Y	Y	Y	N	Y	N	N	
4	N	Y	Y	Y	N	Y	N	N	

```
import itertools # using itertools for efficient looping plotting subplots

# Plotting all the graph to find the relation and evaluting for dropping such columns

plt.figure(figsize = [20,24])

for i,j in itertools.zip_longest(flag_columns,range(len(flag_columns))):
    plt.subplot(7,4,j+1)
    ax = sns.countplot(flag_df[i], hue = flag_df["TARGET"], palette = ["r","b"])
    #plt.yticks(fontsize=8)
    plt.xlabel("")
    plt.ylabel("")
    plt.title(i)
```



```
# removing required columns from "flag_df" such that we can remove the irrelevent columns from "appl_data" dataset.

flag_df.drop(["TARGET","FLAG_OWN_REALTY","FLAG_MOBIL","FLAG_DOCUMENT_3"], axis=1 , inplace = True)
```

```
len(flag_df.columns)
```

```
25
```

```
# dropping the columns of "flag_df" dataframe that is removing more 25 columns from "appl_data" dataframe

appl_data.drop(flag_df.columns, axis=1, inplace= True)
```

```
appl_data.shape    # Now we are left 46 revelent columns
```

```
(307511, 46)
```

- **After removing unnecessary and missing columns we are left with 46 columns.**

Replacing missing values

Now that we have removed all the unnecessary columns, we will proceed with imputing values for missing columns wherever required.

```
null_values(appl_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
OBS_60_CNT_SOCIAL_CIRCLE       0.33
OBS_30_CNT_SOCIAL_CIRCLE       0.33
dtype: float64
```

Insight

- **Now we have only 7 columns which have missing values more than 1%. Thus, we will only impute them for further analysis**
- **Following are the columns: OCCUPATION_TYPE, AMT_REQ_CREDIT_BUREAU_YEAR, AMT_REQ_CREDIT_BUREAU_QRT, AMT_REQ_CREDIT_BUREAU_MON, AMT_REQ_CREDIT_BUREAU_WEEK, AMT_REQ_CREDIT_BUREAU_DAY, AMT_REQ_CREDIT_BUREAU_HOUR**

Imputing values for column "OCCUPATION_TYPE"

```
#Percentage of each category present in "OCCUPATION_TYPE"

appl_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
```

Insight:

- **Since "OCCUPATION_TYPE" column is categorical one and have missing values of 31.35%. To fix this we will impute a category as "Unknown" for the missing values.**

Replacing missing values as “Unknown”

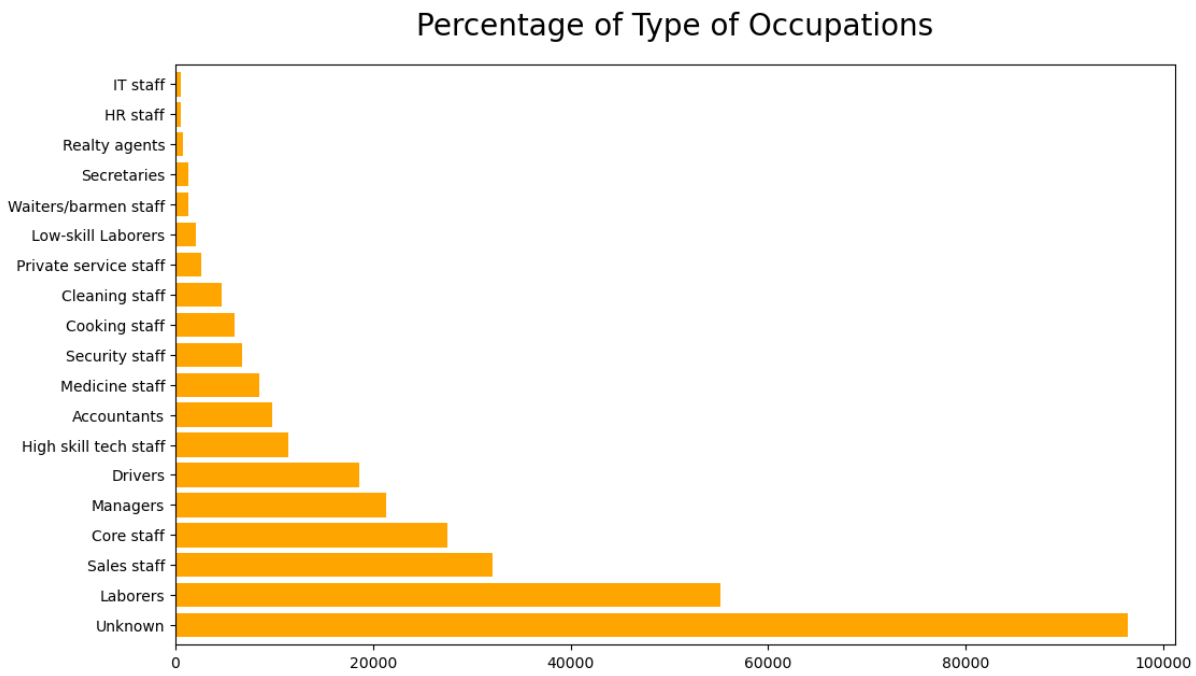
```
# imputing null values with "Unknown"

appl_data["OCCUPATION_TYPE"] = appl_data["OCCUPATION_TYPE"].fillna("Unknown")
```

```
appl_data["OCCUPATION_TYPE"].isnull().sum() # Now we have zero null values
```

```
0
```

Each category of Occupation_Type

```
# Plotting a percentage graph having each category of "OCCUPATION_TYPE"

plt.figure(figsize = [12,7])
(appl_data["OCCUPATION_TYPE"].value_counts()).plot.barh(color= "orange",width = .8)
plt.title("Percentage of Type of Occupations", fontdict={"fontsize":20}, pad =20)
plt.show()
```




```
appl_data[["AMT_REQ_CREDIT_BUREAU_YEAR","AMT_REQ_CREDIT_BUREAU_QRT","AMT_REQ_CREDIT_BUREAU_MON","AMT_REQ_CREDIT_BUREAU_WEEK",
"AMT_REQ_CREDIT_BUREAU_DAY","AMT_REQ_CREDIT_BUREAU_HOUR"]].describe()
```

	AMT_REQ_CREDIT_BUREAU_YEAR	AMT_REQ_CREDIT_BUREAU_QRT	AMT_REQ_CREDIT_BUREAU_MON	AMT_REQ_CREDIT_BUREAU_WEEK	AMT_REQ_C
count	265992.000000	265992.000000	265992.000000	265992.000000	
mean	1.899974	0.265474	0.267395	0.034362	
std	1.869295	0.794056	0.916002	0.204685	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	
50%	1.000000	0.000000	0.000000	0.000000	
75%	3.000000	0.000000	0.000000	0.000000	
max	25.000000	261.000000	27.000000	8.000000	

- **For imputing missing values, we will not use mean as it is in decimal form, hence for imputing purpose we will use median for all these columns.**

```
#creating "amt_credit" variable having these columns "AMT_REQ_CREDIT_BUREAU_YEAR","AMT_REQ_CREDIT_BUREAU_QRT","AMT_REQ_CREDIT_BUR
#"AMT_REQ_CREDIT_BUREAU_DAY","AMT_REQ_CREDIT_BUREAU_HOUR"

amt_credit = ["AMT_REQ_CREDIT_BUREAU_YEAR","AMT_REQ_CREDIT_BUREAU_QRT","AMT_REQ_CREDIT_BUREAU_MON","AMT_REQ_CREDIT_BUREAU_WEEK",
"AMT_REQ_CREDIT_BUREAU_DAY","AMT_REQ_CREDIT_BUREAU_HOUR"]
```

Filling missing values with median values

```
#filling missing values with median values

appl_data.fillna(appl_data[amt_credit].median(),inplace = True)
```

```
null_values(appl_data).head(10)
```

```
NAME_TYPE_SUITE             0.42
DEF_60_CNT_SOCIAL_CIRCLE    0.33
OBS_60_CNT_SOCIAL_CIRCLE    0.33
DEF_30_CNT_SOCIAL_CIRCLE    0.33
OBS_30_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
ORGANIZATION_TYPE           0.00
dtype: float64
```

Standardising values

```
appl_data.describe()
```

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATI
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

Insights:

- **Columns AMT_INCOME_TOTAL, AMT_CREDIT, AMT_GOODS_PRICE have very high values, thus will make these numerical columns in categorical columns for better understanding.**
- **Columns DAYS_BIRTH, DAYS_EMPLOYED, DAYS_REGISTRATION, DAYS_ID_PUBLISH, DAYS_LAST_PHONE_CHANGE have negative values. Thus those values will be corrected**
- **Convert DAYS_BIRTH column to AGE in years , DAYS_EMPLOYED column to YEARS EMPLOYED**

Binning columns: AMT_INCOME_TOTAL, AMT_CREDIT, AMT_GOODS_PRICE

```
# Binning Numerical Columns to create a categorical column

# Creating bins for income amount in term of Lakhs
appl_data['AMT_INCOME_TOTAL']=appl_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']

appl_data['AMT_INCOME_RANGE']=pd.cut(appl_data['AMT_INCOME_TOTAL'],bins,labels=slot)
```

```
round((appl_data["AMT_INCOME_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_RANGE, dtype: float64
```

Creating bins for Credit amount in term of Lakhs

```
appl_data['AMT_CREDIT']=appl_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']

appl_data['AMT_CREDIT_RANGE']=pd.cut(appl_data['AMT_CREDIT'],bins=bins,labels=slots)
```

```
round((appl_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 bins for Price of Goods in term of Lakhs

```
# Creating bins for Price of Goods in term of Lakhs
appl_data['AMT_GOODS_PRICE']=appl_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']

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

```
round((appl_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
```

Correcting the following columns :

DAYS_BIRTH, DAYS_EMPLOYED, DAYS_REGISTRATION, DAYS_ID_PUBLISH, DAYS_LAST_PHONE_CHANGE

```
# creating "days_col" varibale to store all days columns
days_col = ["DAYS_BIRTH", "DAYS_EMPLOYED", "DAYS_REGISTRATION", "DAYS_ID_PUBLISH", "DAYS_LAST_PHONE_CHANGE"]

appl_data[days_col].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

Correcting the days value:

```
#using abs() function to correct the days values

appl_data[days_col]= abs(appl_data[days_col])
```

```
# Data is correct now

appl_data[days_col].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

Converting DAYS_BIRTH, DAYS_EMPLOYED columns in terms of Years and bininng Years

```
appl_data["AGE"] = appl_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"]

appl_data["AGE_GROUP"] = pd.cut(appl_data["AGE"], bins=bins, labels=slots)
```

```
appl_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
```

Creating column "EMPLOYEMENT_YEARS" from "DAYS_EMPLOYED"

```
#creating column "EMPLOYEMENT_YEARS" from "DAYS_EMPLOYED"

appl_data["YEARS_EMPLOYED"] = appl_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"]

appl_data["EMPLOYEMENT_YEARS"] = pd.cut(appl_data["YEARS_EMPLOYED"], bins=bins, labels=slots)
```

```
appl_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
```

Identifying Outliers

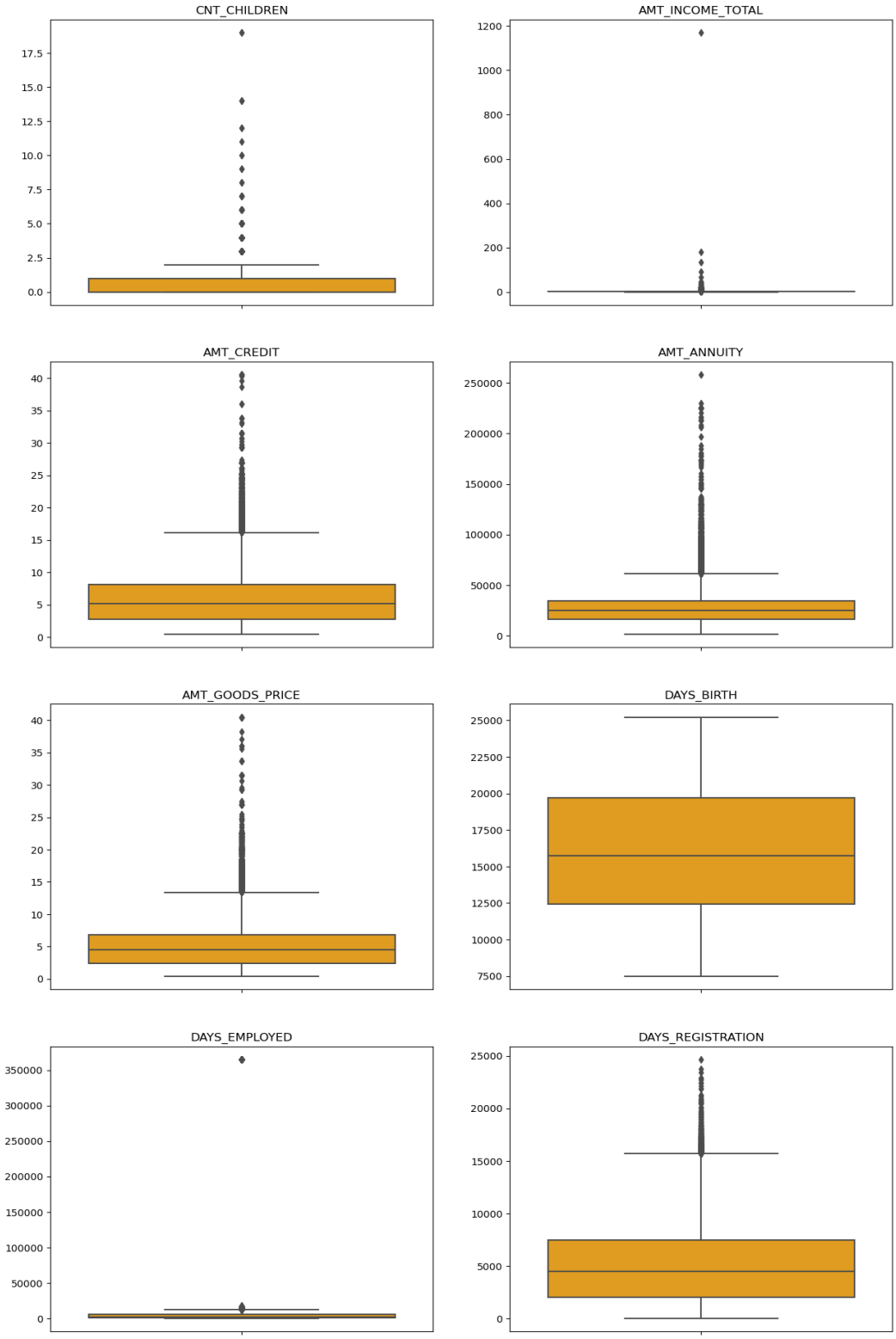
```
appl_data.describe()
```

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATI
count	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000	307499.000000	307233.000000	307511.000(
mean	278180.518577	0.080729	0.417052	1.687979	5.990260	27108.573909	5.383962	0.020
std	102790.175348	0.272419	0.722121	2.371231	4.024908	14493.737315	3.694465	0.013
min	100002.000000	0.000000	0.000000	0.256500	0.450000	1615.500000	0.405000	0.000
25%	189145.500000	0.000000	0.000000	1.125000	2.700000	16524.000000	2.385000	0.010(
50%	278202.000000	0.000000	0.000000	1.471500	5.135310	24903.000000	4.500000	0.018
75%	367142.500000	0.000000	1.000000	2.025000	8.086500	34596.000000	6.795000	0.028
max	456255.000000	1.000000	19.000000	1170.000000	40.500000	258025.500000	40.500000	0.072

- **We could find all the columns those who have high difference between max and 75 percentile and the ones which makes no sense having max value to be so high are captured below:**

```
outlier_col = ["CNT_CHILDREN","AMT_INCOME_TOTAL", "AMT_CREDIT", "AMT_ANNUITY", "AMT_GOODS_PRICE",
               "DAYS_BIRTH", "DAYS_EMPLOYED", "DAYS_REGISTRATION"]
```

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



Insight:

It can be seen that in current application data

- **AMT_ANNUITY, AMT_CREDIT, AMT_GOODS_PRICE,CNT_CHILDREN have some number of outliers.**

- **AMT_INCOME_TOTAL has huge number of outliers which indicate that few of the loan applicants have high income when compared to the others.**
- **DAYS_BIRTH has no outliers which means the data available is reliable.**
- **DAYS_EMPLOYED has outlier values around 350000(days) which is around 958 years which is impossible and hence this has to be incorrect entry.**

```
appl_data.nunique().sort_values()
```

```
LIVE_REGION_NOT_WORK_REGION          2
TARGET                               2
NAME_CONTRACT_TYPE                   2
REG_REGION_NOT_LIVE_REGION           2
FLAG_OWN_REALTY                      2
LIVE_CITY_NOT_WORK_CITY              2
REG_CITY_NOT_WORK_CITY               2
REG_CITY_NOT_LIVE_CITY               2
FLAG_DOCUMENT_3                      2
REG_REGION_NOT_WORK_REGION           2
FLAG_MOBIL                           2
REGION_RATING_CLIENT                 3
CODE_GENDER                          3
REGION_RATING_CLIENT_W_CITY          3
NAME_EDUCATION_TYPE                  5
AMT_REQ_CREDIT_BUREAU_HOUR           5
NAME_FAMILY_STATUS                   6
NAME_HOUSING_TYPE                    6
EMPLOYEMENT_YEARS                    7
WEEKDAY_APPR_PROCESS_START           7
NAME_TYPE_SUITE                      7
NAME_INCOME_TYPE                     8
AMT_REQ_CREDIT_BUREAU_WEEK           9
AMT_REQ_CREDIT_BUREAU_DAY            9
DEF_60_CNT_SOCIAL_CIRCLE             9
AGE_GROUP                            9
DEF_30_CNT_SOCIAL_CIRCLE            10
AMT_REQ_CREDIT_BUREAU_QRT           11
AMT_INCOME_RANGE                    11
AMT_CREDIT_RANGE                    11
AMT_GOODS_PRICE_RANGE               11
CNT_CHILDREN                        15
CNT_FAM_MEMBERS                     17
OCCUPATION_TYPE                     19
AMT_REQ_CREDIT_BUREAU_MON           24
HOUR_APPR_PROCESS_START             24
AMT_REQ_CREDIT_BUREAU_YEAR          25
OBS_60_CNT_SOCIAL_CIRCLE            33
OBS_30_CNT_SOCIAL_CIRCLE            33
ORGANIZATION_TYPE                   58
REGION_POPULATION_RELATIVE          81
AMT_GOODS_PRICE                   1002
AMT_INCOME_TOTAL                  2548
DAYS_LAST_PHONE_CHANGE            3773
AMT_CREDIT                        5603
DAYS_ID_PUBLISH                   6168
DAYS_EMPLOYED                    12574
YEARS_EMPLOYED                   12574
AMT_ANNUITY                      13672
DAYS_REGISTRATION                15688
DAYS_BIRTH                       17460
AGE                              17460
SK_ID_CURR                      307511
dtype: int64
```

Identifying Categorical columns

```
#Checking the number of unique values each column possess to identify categorical columns

appl_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 53 columns):
 #   Column                       Non-Null Count   Dtype   
---  ------                       --------------   -----   
 0   SK_ID_CURR                   307511 non-null  int64   
 1   TARGET                       307511 non-null  int64   
 2   NAME_CONTRACT_TYPE           307511 non-null  object  
 3   CODE_GENDER                  307511 non-null  object  
 4   FLAG_OWN_REALTY              307511 non-null  object  
 5   CNT_CHILDREN                 307511 non-null  int64   
 6   AMT_INCOME_TOTAL             307511 non-null  float64 
 7   AMT_CREDIT                   307511 non-null  float64 
 8   AMT_ANNUITY                  307499 non-null  float64 
 9   AMT_GOODS_PRICE              307233 non-null  float64 
 10  NAME_TYPE_SUITE              306219 non-null  object  
 11  NAME_INCOME_TYPE             307511 non-null  object  
 12  NAME_EDUCATION_TYPE          307511 non-null  object  
 13  NAME_FAMILY_STATUS           307511 non-null  object  
 14  NAME_HOUSING_TYPE            307511 non-null  object  
 15  REGION_POPULATION_RELATIVE   307511 non-null  float64 
 16  DAYS_BIRTH                   307511 non-null  int64   
 17  DAYS_EMPLOYED                307511 non-null  int64   
 18  DAYS_REGISTRATION            307511 non-null  float64 
 19  DAYS_ID_PUBLISH              307511 non-null  int64   
 20  FLAG_MOBIL                   307511 non-null  int64   
 21  OCCUPATION_TYPE              307511 non-null  object  
 22  CNT_FAM_MEMBERS              307509 non-null  float64 
 23  REGION_RATING_CLIENT         307511 non-null  int64   
 24  REGION_RATING_CLIENT_W_CITY  307511 non-null  int64   
 25  WEEKDAY_APPR_PROCESS_START   307511 non-null  object  
 26  HOUR_APPR_PROCESS_START      307511 non-null  int64   
 27  REG_REGION_NOT_LIVE_REGION   307511 non-null  int64   
 28  REG_REGION_NOT_WORK_REGION   307511 non-null  int64   
 29  LIVE_REGION_NOT_WORK_REGION  307511 non-null  int64   
 30  REG_CITY_NOT_LIVE_CITY       307511 non-null  int64   
 31  REG_CITY_NOT_WORK_CITY       307511 non-null  int64   
 32  LIVE_CITY_NOT_WORK_CITY      307511 non-null  int64   
 33  ORGANIZATION_TYPE            307511 non-null  object  
 34  OBS_30_CNT_SOCIAL_CIRCLE     306490 non-null  float64 
 35  DEF_30_CNT_SOCIAL_CIRCLE     306490 non-null  float64 
 36  OBS_60_CNT_SOCIAL_CIRCLE     306490 non-null  float64 
 37  DEF_60_CNT_SOCIAL_CIRCLE     306490 non-null  float64 
 38  DAYS_LAST_PHONE_CHANGE       307510 non-null  float64 
 39  FLAG_DOCUMENT_3              307511 non-null  int64   
 40  AMT_REQ_CREDIT_BUREAU_HOUR   307511 non-null  float64 
 41  AMT_REQ_CREDIT_BUREAU_DAY    307511 non-null  float64 
 42  AMT_REQ_CREDIT_BUREAU_WEEK   307511 non-null  float64 
 43  AMT_REQ_CREDIT_BUREAU_MON    307511 non-null  float64 
 44  AMT_REQ_CREDIT_BUREAU_QRT    307511 non-null  float64 
 45  AMT_REQ_CREDIT_BUREAU_YEAR   307511 non-null  float64 
 46  AMT_INCOME_RANGE             307279 non-null  category
 47  AMT_CREDIT_RANGE             307511 non-null  category
 48  AMT_GOODS_PRICE_RANGE        307233 non-null  category
 49  AGE                          307511 non-null  float64 
 50  AGE_GROUP                    307511 non-null  category
 51  YEARS_EMPLOYED               307511 non-null  float64 
 52  EMPLOYEMENT_YEARS            252135 non-null  category
dtypes: category(5), float64(20), int64(17), object(11)
```

Converting Desired columns from Object to categorical column

```
appl_data.columns
```

```
Index(['SK_ID_CURR', 'TARGET', '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', 'FLAG
_MOBIL', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_STA
RT', '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', 'DE
F_30_CNT_SOCIAL_CIRCLE', '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', 'AMT_INCOME_RANGE', 'AMT_CREDIT_
RANGE', 'AMT_GOODS_PRICE_RANGE', 'AGE', 'AGE_GROUP', 'YEARS_EMPLOYED', 'EMPLOYEMENT_YEARS'],
      dtype='object')
```

Selecting columns to convert into categorical columns

```
#from the list, we have taken out the desired columns for conversion

categorical_columns = ['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 categorical_columns:
    appl_data[col] = pd.Categorical(appl_data[col])
```

```
len(categorical_columns) # Converting total of 21 columns to categorical one
```

```
21
```

Insight

- **After imputing we have 53 columns and we will move ahead with Data Analysis on these columns**

Dataset 2 - "previous_application.csv"

Importing the dataset

```
# importing previous_application.csv

prev_appl = pd.read_csv('C:\\Users\\Rishabh Negi\\Desktop\\Projects\\Project 6\\previous_application.csv')
```

```
prev_appl.head()
```

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

```
#Checking rows and columns of the raw data
prev_appl.shape
```

```
(1670214, 37)
```

- **There are 1670214 rows and 37 columns in the dataset.**

```
#Checking information of all the columns like data types
prev_appl.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
```

- **There are 37 columns having various data types like object, int, float and 1670214 rows.**

```
# Checking the numeric variables of the dataframes
prev_appl.describe()
```

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	HOUR_APPR_PROCESS_STA
count	1.670214e+06	1.670214e+06	1.297979e+06	1.670214e+06	1.670213e+06	7.743700e+05	1.284699e+06	1.670214e+
mean	1.923089e+06	2.783572e+05	1.595512e+04	1.752339e+05	1.961140e+05	6.697402e+03	2.278473e+05	1.248418e+
std	5.325980e+05	1.028148e+05	1.478214e+04	2.927798e+05	3.185746e+05	2.092150e+04	3.153966e+05	3.334028e+
min	1.000001e+06	1.000010e+05	0.000000e+00	0.000000e+00	0.000000e+00	-9.000000e-01	0.000000e+00	0.000000e+
25%	1.461857e+06	1.893290e+05	6.321780e+03	1.872000e+04	2.416050e+04	0.000000e+00	5.084100e+04	1.000000e+
50%	1.923110e+06	2.787145e+05	1.125000e+04	7.104600e+04	8.054100e+04	1.638000e+03	1.123200e+05	1.200000e+
75%	2.384280e+06	3.675140e+05	2.065842e+04	1.803600e+05	2.164185e+05	7.740000e+03	2.340000e+05	1.500000e+
max	2.845382e+06	4.562550e+05	4.180581e+05	6.905160e+06	6.905160e+06	3.060045e+06	6.905160e+06	2.300000e+

Insight

- **There are 37 columns and 1679214 rows.**
- **There are columns having negative and positive values which includes days. The data is needed to be fixed.**

Checking null values present in the dataset:

```
#checking how many null values are present in each of the columns in percentage
null_values(prev_appl)
```

```
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
AMT_CREDIT                      0.00
NAME_YIELD_GROUP                0.00
NAME_PORTFOLIO                  0.00
NAME_SELLER_INDUSTRY            0.00
SELLERPLACE_AREA                0.00
CHANNEL_TYPE                    0.00
NAME_PRODUCT_TYPE               0.00
SK_ID_PREV                      0.00
NAME_GOODS_CATEGORY             0.00
NAME_CLIENT_TYPE                0.00
CODE_REJECT_REASON              0.00
SK_ID_CURR                      0.00
DAYS_DECISION                   0.00
NAME_CONTRACT_STATUS            0.00
NAME_CASH_LOAN_PURPOSE          0.00
NFLAG_LAST_APPL_IN_DAY          0.00
FLAG_LAST_APPL_PER_CONTRACT     0.00
HOUR_APPR_PROCESS_START         0.00
WEEKDAY_APPR_PROCESS_START      0.00
AMT_APPLICATION                 0.00
NAME_CONTRACT_TYPE              0.00
NAME_PAYMENT_TYPE               0.00
dtype: float64
```

Creating a variable p_null_col_50 for storing null columns having missing values more than 50%

```
#creating a variable p_null_col_50 for storing null columns having missing values more than 50%

p_null_col_50 = null_values(prev_appl)[null_values(prev_appl)>50]
```

```
p_null_col_50 # There only 4 columns with missing valus more than 50%
```

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

- **There are 4 columns having missing values more than 50%.**

Dropping Null Columns

```
#dropping null columns having missing values more than 50%

prev_appl.drop(columns = p_null_col_50.index, inplace = True)
```

Creating a variable p_null_col_15 for storing null columns having missing values more than 15%

```
#creating a variable p_null_col_15 for storing null columns having missing values more than 15%

p_null_col_15 = null_values(prev_appl)[null_values(prev_appl)>15]
```

```
p_null_col_15
```

```
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
```

- **There are 4 columns having missing values more than 15% but less than 50%.**

```
prev_appl[p_null_col_15.index]
```

	NAME_TYPE_SUITE	DAYS_FIRST_DRAWING	DAYS_TERMINATION	DAYS_LAST_DUE	DAYS_LAST_DUE_1ST_VERSION	DAYS_FIRST_DUE	NFLAG_INSU
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	

1670214 rows × 10 columns

```
prev_appl.columns
```

```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY', 'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_GOODS_PRICE', 'WE
EKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START', 'FLAG_LAST_APPL_PER_CONTRACT', 'NFLAG_LAST_APPL_IN_DAY', 'NAME_CASH_LOAN_
PURPOSE', 'NAME_CONTRACT_STATUS', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE', 'CODE_REJECT_REASON', 'NAME_TYPE_SUITE', 'NAME_CLIENT_T
YPE', 'NAME_GOODS_CATEGORY', 'NAME_PORTFOLIO', 'NAME_PRODUCT_TYPE', 'CHANNEL_TYPE', 'SELLERPLACE_AREA', 'NAME_SELLER_INDUSTRY',
'CNT_PAYMENT', 'NAME_YIELD_GROUP', 'PRODUCT_COMBINATION', 'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION',
'DAYS_LAST_DUE', 'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROVAL'], dtype='object')
```

Listing Unnecessary columns

```
# Listing down columns which are not needed
Unnecessary_prev = ['WEEKDAY_APPR_PROCESS_START','HOUR_APPR_PROCESS_START','FLAG_LAST_APPL_PER_CONTRACT','NFLAG_LAST_APPL_IN_DAY'

prev_appl.drop(Unnecessary_prev,axis =1, inplace = True)

prev_appl.shape
```

```
(1670214, 29)
```

Imputing values “Unknown” as this is a categorical column

```
# Imputing values "Unknown" as this a categorical column
prev_appl["NAME_TYPE_SUITE"] = prev_appl["NAME_TYPE_SUITE"].fillna("Unknown")

null_values(prev_appl)
```

```
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
AMT_CREDIT                    0.00
NAME_PRODUCT_TYPE             0.00
NAME_YIELD_GROUP              0.00
NAME_SELLER_INDUSTRY          0.00
SELLERPLACE_AREA              0.00
CHANNEL_TYPE                  0.00
SK_ID_PREV                    0.00
NAME_PORTFOLIO                0.00
SK_ID_CURR                    0.00
NAME_CLIENT_TYPE              0.00
NAME_TYPE_SUITE               0.00
CODE_REJECT_REASON            0.00
NAME_PAYMENT_TYPE             0.00
DAYS_DECISION                 0.00
NAME_CONTRACT_STATUS          0.00
NAME_CASH_LOAN_PURPOSE        0.00
AMT_APPLICATION               0.00
NAME_CONTRACT_TYPE            0.00
NAME_GOODS_CATEGORY           0.00
dtype: float64
```

- **There are missing values in columns 'DAYS_FIRST_DUE', 'DAYS_TERMINATION', 'DAYS_FIRST_DRAWING','DAYS_LAST_DUE_1ST_VERSION', 'DAYS_LAST_DUE' and these columns count days thus will keeping null values as they are.**

```
#Analyzing numerical columns using describe

prev_appl[p_null_col_15.index].describe()
```

	DAYS_FIRST_DRAWING	DAYS_TERMINATION	DAYS_LAST_DUE	DAYS_LAST_DUE_1ST_VERSION	DAYS_FIRST_DUE	NFLAG_INSURED_ON_APPROVAL	A
count	997149.000000	997149.000000	997149.000000	997149.000000	997149.000000	997149.000000	
mean	342209.855039	81992.343838	76582.403064	33767.774054	13826.269337	0.332570	
std	88916.115834	153303.516729	149647.415123	106857.034789	72444.869708	0.471134	
min	-2922.000000	-2874.000000	-2889.000000	-2801.000000	-2892.000000	0.000000	
25%	365243.000000	-1270.000000	-1314.000000	-1242.000000	-1628.000000	0.000000	
50%	365243.000000	-499.000000	-537.000000	-361.000000	-831.000000	0.000000	
75%	365243.000000	-44.000000	-74.000000	129.000000	-411.000000	1.000000	
max	365243.000000	365243.000000	365243.000000	365243.000000	365243.000000	1.000000	

Creating a variable to convert negative days into positive days

```
# To convert negative days to postive days creating a varaible "p_days_col"

p_days_col = ['DAYS_DECISION','DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION', 'DAYS_LAST_DUE', 'DAYS_TERMINA

prev_appl[p_days_col].describe() # Analysis before conversion
```

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

Converting negative days to positive days

```
# Converting Negative days to positive days

prev_appl[p_days_col] = abs(prev_appl[p_days_col])

prev_appl[p_null_col_15.index].describe() # analysing after conversion
```

	DAYS_FIRST_DRAWING	DAYS_TERMINATION	DAYS_LAST_DUE	DAYS_LAST_DUE_1ST_VERSION	DAYS_FIRST_DUE	NFLAG_INSURED_ON_APPROVAL	A
count	997149.000000	997149.000000	997149.000000	997149.000000	997149.000000	997149.000000	
mean	342340.056543	83505.775017	78152.730207	35163.363265	15949.224065	0.332570	
std	88413.495220	152484.418802	148833.342466	106405.950190	72007.270877	0.471134	
min	2.000000	2.000000	2.000000	0.000000	2.000000	0.000000	
25%	365243.000000	447.000000	455.000000	257.000000	475.000000	0.000000	
50%	365243.000000	1171.000000	1155.000000	741.000000	921.000000	0.000000	
75%	365243.000000	2501.000000	2418.000000	1735.000000	1825.000000	1.000000	
max	365243.000000	365243.000000	365243.000000	365243.000000	365243.000000	1.000000	

```
#days group calculation e.g. 369 will be grouped as with in 2 years

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_appl['YEARLY_DECISION'] = pd.cut(prev_appl['DAYS_DECISION'],bins,labels=slots)
```

```
prev_appl['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
```

Insight:

- **Almost 35% loan applicants have applied for a new loan within 1 year of previous loan decision**

Now dealing with continuous variables "AMT_ANNUITY", "AMT_GOODS_PRICE"

```
prev_appl.nunique()
```

```
SK_ID_PREV                   1670214
SK_ID_CURR                    338857
NAME_CONTRACT_TYPE                4
AMT_ANNUITY                  357959
AMT_APPLICATION               93885
AMT_CREDIT                    86803
AMT_GOODS_PRICE               93885
NAME_CASH_LOAN_PURPOSE           25
NAME_CONTRACT_STATUS              4
DAYS_DECISION                  2922
NAME_PAYMENT_TYPE                 4
CODE_REJECT_REASON                9
NAME_TYPE_SUITE                   8
NAME_CLIENT_TYPE                  4
NAME_GOODS_CATEGORY              28
NAME_PORTFOLIO                    5
NAME_PRODUCT_TYPE                 3
CHANNEL_TYPE                      8
SELLERPLACE_AREA               2097
NAME_SELLER_INDUSTRY             11
CNT_PAYMENT                      49
NAME_YIELD_GROUP                  5
PRODUCT_COMBINATION              17
DAYS_FIRST_DRAWING             2838
DAYS_FIRST_DUE                 2892
DAYS_LAST_DUE_1ST_VERSION      2803
DAYS_LAST_DUE                  2873
DAYS_TERMINATION               2830
NFLAG_INSURED_ON_APPROVAL         2
YEARLY_DECISION                   8
dtype: int64
```

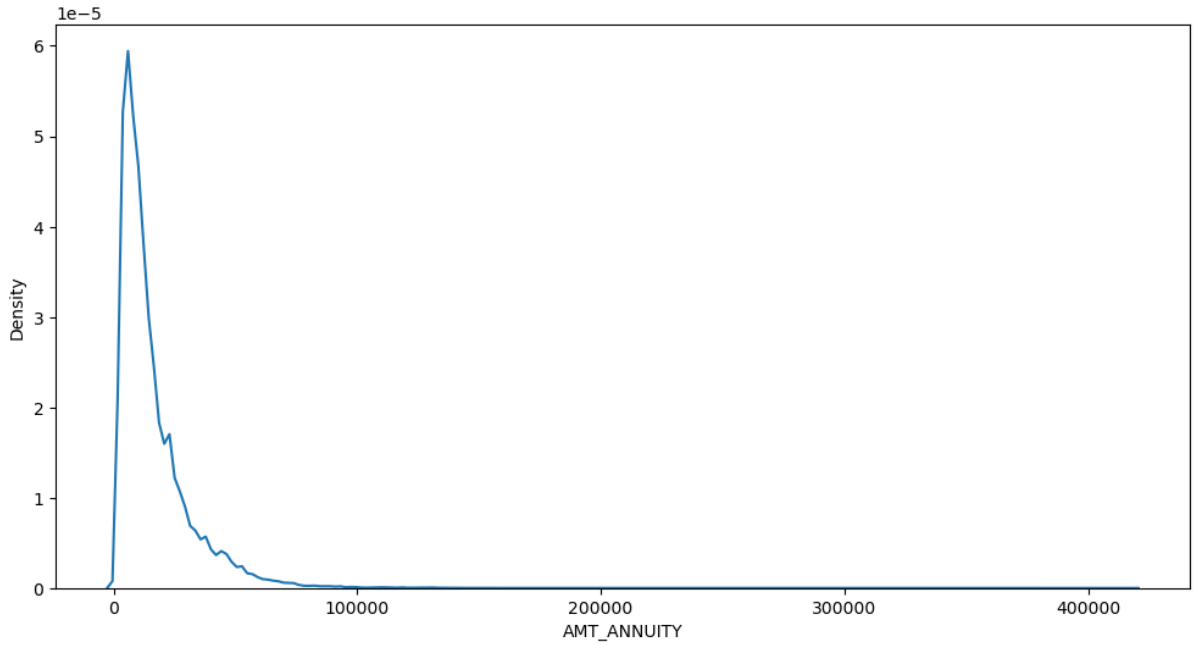
```
prev_appl.nunique()
```

```
SK_ID_PREV                   1670214
SK_ID_CURR                    338857
NAME_CONTRACT_TYPE                4
AMT_ANNUITY                  357959
AMT_APPLICATION               93885
AMT_CREDIT                    86803
AMT_GOODS_PRICE               93885
NAME_CASH_LOAN_PURPOSE           25
NAME_CONTRACT_STATUS              4
DAYS_DECISION                  2922
NAME_PAYMENT_TYPE                 4
CODE_REJECT_REASON                9
NAME_TYPE_SUITE                   8
NAME_CLIENT_TYPE                  4
NAME_GOODS_CATEGORY              28
NAME_PORTFOLIO                    5
NAME_PRODUCT_TYPE                 3
CHANNEL_TYPE                      8
SELLERPLACE_AREA               2097
NAME_SELLER_INDUSTRY             11
CNT_PAYMENT                      49
NAME_YIELD_GROUP                  5
PRODUCT_COMBINATION              17
DAYS_FIRST_DRAWING             2838
DAYS_FIRST_DUE                 2892
DAYS_LAST_DUE_1ST_VERSION      2803
DAYS_LAST_DUE                  2873
DAYS_TERMINATION               2830
NFLAG_INSURED_ON_APPROVAL         2
YEARLY_DECISION                   8
dtype: int64
```

To impute null values in continuous variables, we plotted the distribution of the columns and used following methods

- **median if the distribution is skewed**
- **mode if the distribution pattern is preserved.**

Distribution of "AMT_ANNUITY"



Insight:

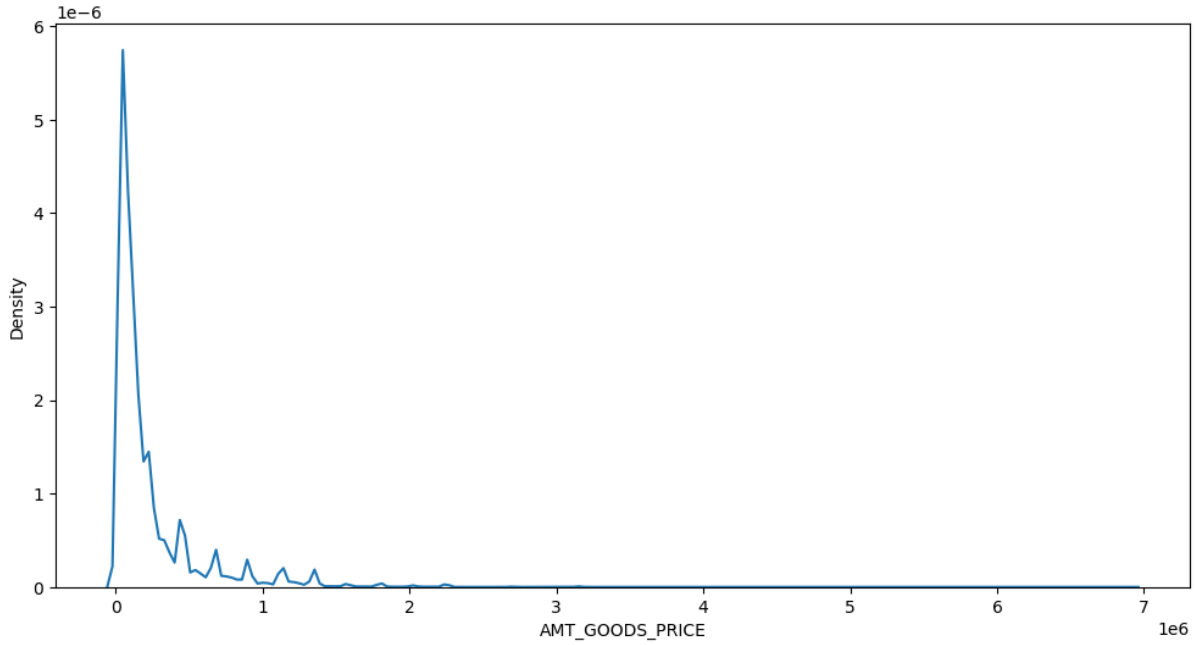
- **There is a single peak at the left side of the distribution and it indicates the presence of outliers and hence imputing with mean would not be the right approach and hence imputing with median.**

Imputing missing values with median

```
#imputing missing values with median

prev_appl['AMT_ANNUITY'].fillna(prev_appl['AMT_ANNUITY'].median(),inplace = True)
```

Distribution of "AMT_GOODS_PRICE"

```
# Plotting kde plot for "AMT_GOODS_PRICE" to understand the distribution

plt.figure(figsize=(12,6))
sns.kdeplot(prev_appl['AMT_GOODS_PRICE'])
plt.show()
```

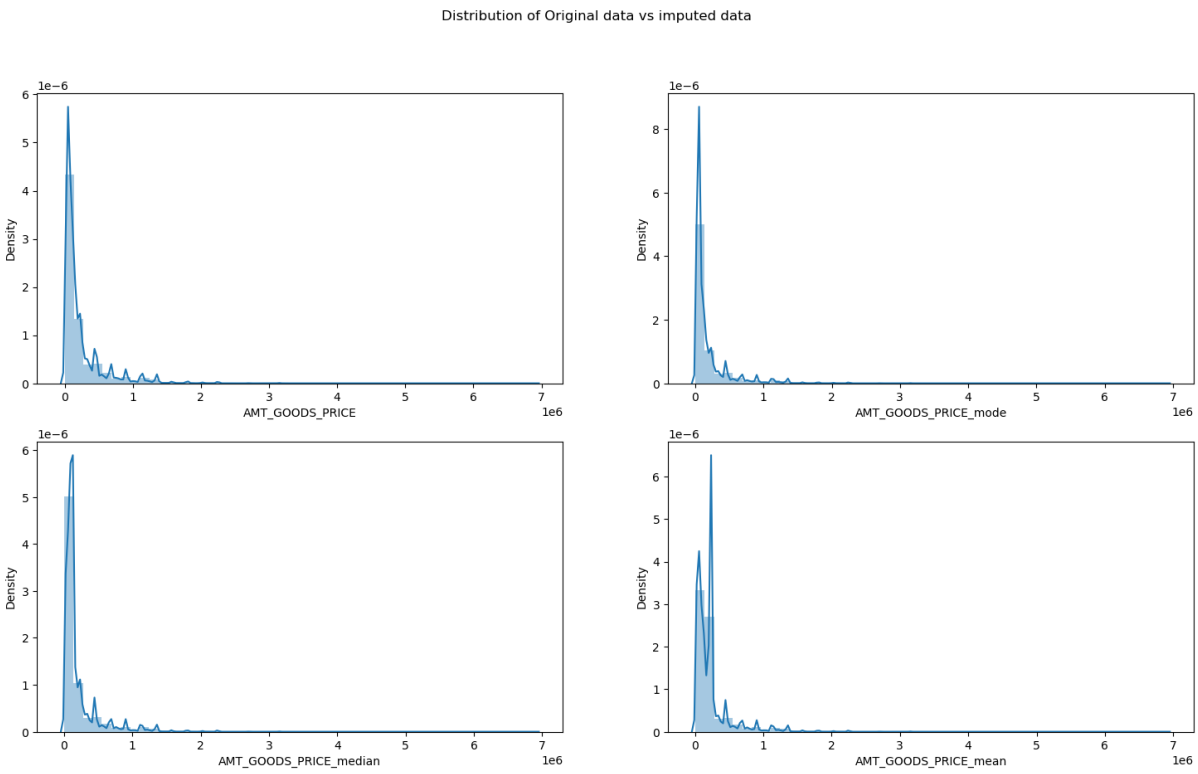


Insight:

- **There are several peaks along the distribution. Let's impute using the mode, mean and median and check if the distribution is still about the same.**

Creating new dataframe for "AMT_GOODS_PRICE" with columns imputed with mode, median and mean

```
# Creating new dataframe for "AMT_GOODS_PRICE" with columns imputed with mode, median and mean

statsDF = pd.DataFrame()
statsDF['AMT_GOODS_PRICE_mode'] = prev_appl['AMT_GOODS_PRICE'].fillna(prev_appl['AMT_GOODS_PRICE'].mode()[0])
statsDF['AMT_GOODS_PRICE_median'] = prev_appl['AMT_GOODS_PRICE'].fillna(prev_appl['AMT_GOODS_PRICE'].median())
statsDF['AMT_GOODS_PRICE_mean'] = prev_appl['AMT_GOODS_PRICE'].fillna(prev_appl['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_appl['AMT_GOODS_PRICE'][pd.notnull(prev_appl['AMT_GOODS_PRICE'])]);
for i in enumerate(cols):
    plt.subplot(2,2,i[0]+2)
    sns.distplot(statsDF[i[1]])
```



- **The original distribution is closer with the distribution of data imputed with mode in this case, thus will impute mode for missing values**

Imputing CNT_PAYMENT with 0 as the NAME_CONTRACT_STATUS for these indicate that most of these loans were not started:

```
#taking out values count for NAME_CONTRACT_STATUS categories where CNT_PAYMENT have null values.

prev_appl.loc[prev_appl['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 as 0

prev_appl['CNT_PAYMENT'].fillna(0,inplace = True)
```

CONVERTING REQUIRED CATEGORICAL COLUMNS FROM OBJECT TO CATEGORICAL

```
prev_appl.columns
```

```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY', 'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_GOODS_PRICE', 'NA
ME_CASH_LOAN_PURPOSE', 'NAME_CONTRACT_STATUS', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE', 'CODE_REJECT_REASON', 'NAME_TYPE_SUITE',
'NAME_CLIENT_TYPE', 'NAME_GOODS_CATEGORY', 'NAME_PORTFOLIO', 'NAME_PRODUCT_TYPE', 'CHANNEL_TYPE', 'SELLERPLACE_AREA', 'NAME_SEL
LER_INDUSTRY', 'CNT_PAYMENT', 'NAME_YIELD_GROUP', 'PRODUCT_COMBINATION', 'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE
_1ST_VERSION', 'DAYS_LAST_DUE', 'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROVAL', 'YEARLY_DECISION'], dtype='object')
```

```
#Converting required categoical columns from Object to categorical

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

for col in p_catgorical_col:
    prev_appl[col] =pd.Categorical(prev_appl[col])
```

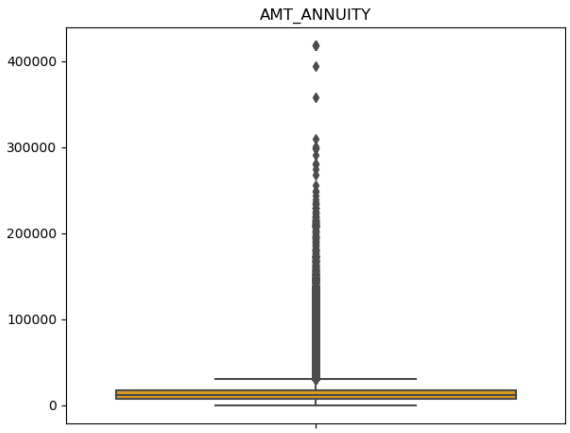
Finding outliers

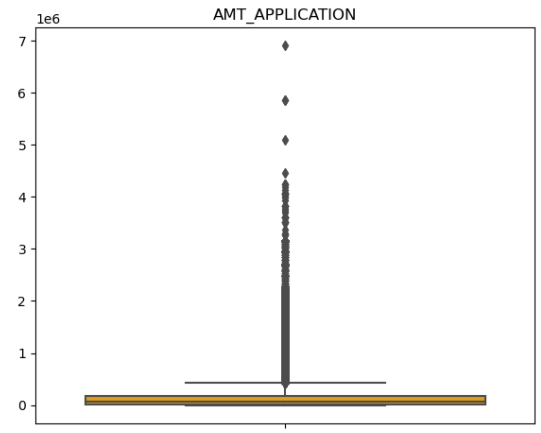
```
prev_appl.describe()
```

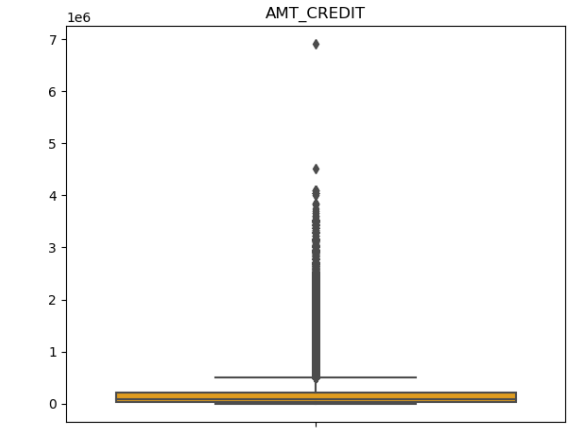
	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE	DAYS_DECISION	SELLERPLACE_AREA	CNT_PAYM
count	1.670214e+06	1.670214e+06	1.670214e+06	1.670214e+06	1.670213e+06	1.670214e+06	1.670214e+06	1.670214e+06	1.670214e
mean	1.923089e+06	2.783572e+05	1.490651e+04	1.752339e+05	1.961140e+05	1.856429e+05	8.806797e+02	3.139511e+02	1.247621e
std	5.325980e+05	1.028148e+05	1.317751e+04	2.927798e+05	3.185746e+05	2.871413e+05	7.790997e+02	7.127443e+03	1.447588e
min	1.000001e+06	1.000010e+05	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	-1.000000e+00	0.000000e
25%	1.461857e+06	1.893290e+05	7.547096e+03	1.872000e+04	2.416050e+04	4.500000e+04	2.800000e+02	-1.000000e+00	0.000000e
50%	1.923110e+06	2.787145e+05	1.125000e+04	7.104600e+04	8.054100e+04	7.105050e+04	5.810000e+02	3.000000e+00	1.000000e
75%	2.384280e+06	3.675140e+05	1.682403e+04	1.803600e+05	2.164185e+05	1.804050e+05	1.300000e+03	8.200000e+01	1.600000e
max	2.845382e+06	4.562550e+05	4.180581e+05	6.905160e+06	6.905160e+06	6.905160e+06	2.922000e+03	4.000000e+06	8.400000e

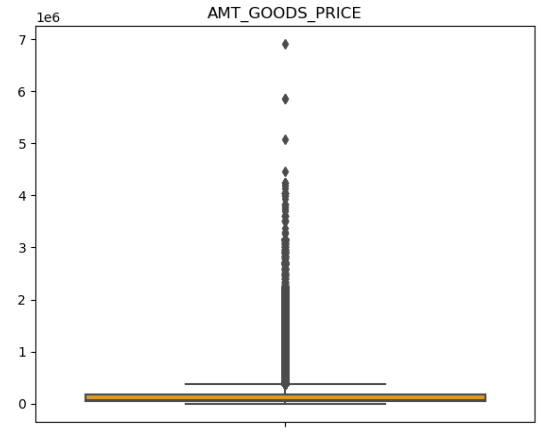
- **We could find all the columns those who have high difference between max and 75 percentile and the ones which makes no sense having max value to be so high are captured below**

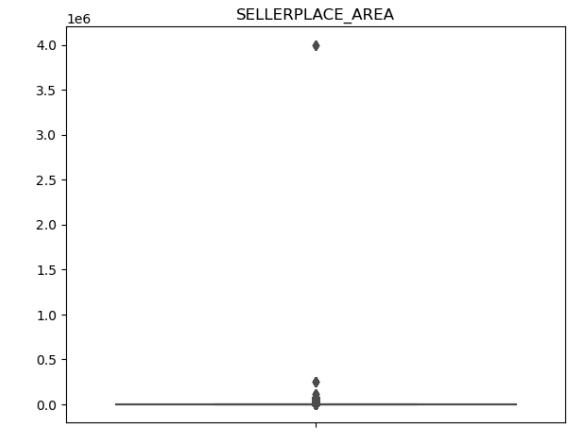
```
p_outlier_col = ['AMT_ANNUITY','AMT_APPLICATION','AMT_CREDIT','AMT_GOODS_PRICE',
                 'SELLERPLACE_AREA','DAYS_DECISION','CNT_PAYMENT']

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

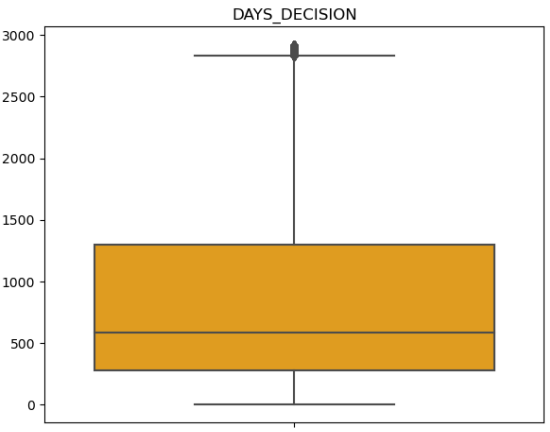

AMT_ANNUITY
400000
300000
200000
100000
0

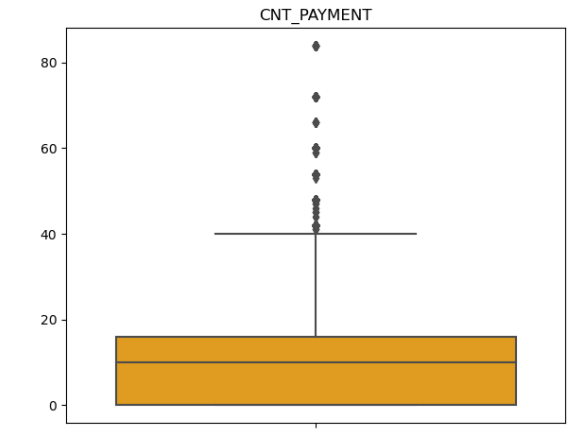

AMT_APPLICATION
1e6
7
6
5
4
3
2
1
0


AMT_CREDIT
1e6
7
6
5
4
3
2
1
0


AMT_GOODS_PRICE
1e6
7
6
5
4
3
2
1
0


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


DAYS_DECISION
3000
2500
2000
1500
1000
500
0


CNT_PAYMENT
80
60
40
20
0

Insight:

It can be seen that in previous application data

- **AMT_ANNUITY, AMT_APPLICATION, AMT_CREDIT, AMT_GOODS_PRICE, SELLERPLACE_AREA have huge number of outliers.**
- **CNT_PAYMENT has few outlier values.**
- **DAYS_DECISION has little number of outliers indicating that these previous applications decisions were taken long back.**

Analysis

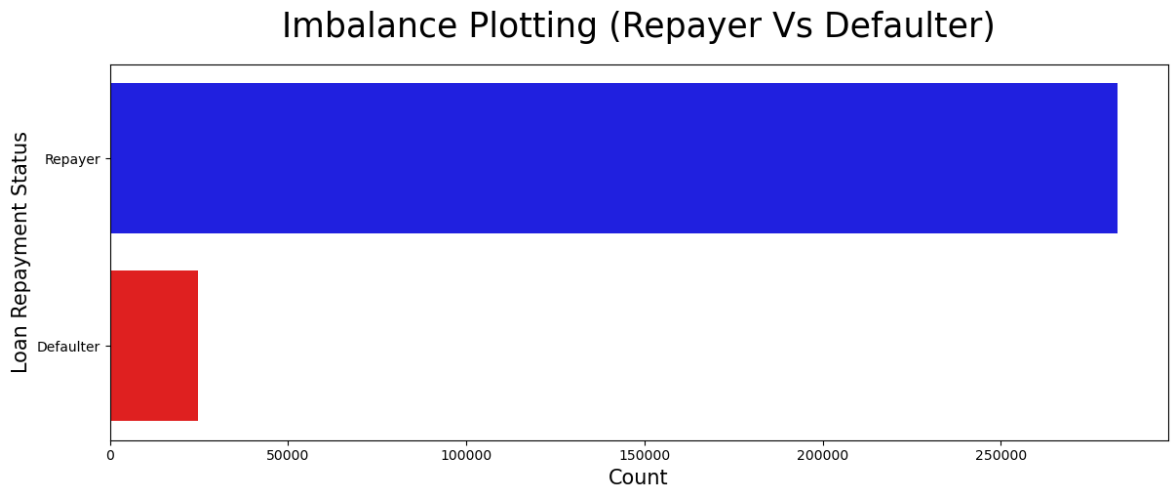
The data will be analysed on the basis of the following:

1. Imbalance in Data
2. Categorical Data Analysis
    I. Categorical segmented Univariate Analysis
    II. Categorical Bi/Multivariate analysis

3. Numeric Data Analysis
4. Bi-furcation of databased based on TARGET data
5. Correlation Matrix
6. Numerical segmented Univariate Analysis
7. Numerical Bi/Multivariate analysis

1. Imbalance in Data

```
plt.figure(figsize= [14,5])
sns.barplot(y=["Repayer","Defaulter"], x = appl_data["TARGET"].value_counts(), palette = ["blue","r"],orient="h")
plt.ylabel("Loan Repayment Status",fontdict = {"fontsize":15})
plt.xlabel("Count",fontdict = {"fontsize":15})
plt.title("Imbalance Plotting (Repayer Vs Defaulter)", fontdict = {"fontsize":25}, pad = 20)
plt.show()
```



Ratio of imbalance percentage with respect to Defaulter and Repayer

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

Insights:

- **Repayer Percentage is 91.93%**
- **Defaulter Percentage is 8.07%**
- **Imbalance Ratio with respect to Repayer and Defaulter is given: 11.39/1 (approx.)**

Important Function for Univariate analysis

Creating a function for plotting Variables to do univariate analysis. This function will create two plots

1. Count plot of given column w.r.t TARGET column
2. Percentage of defaulters within that column

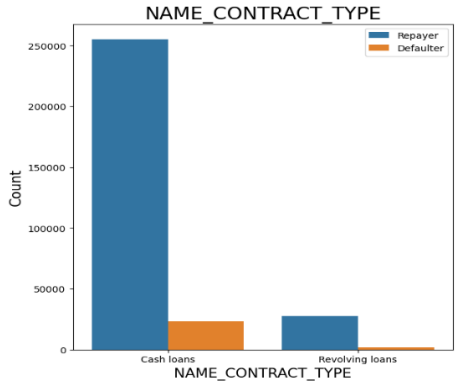
The function is taking 6 arguments

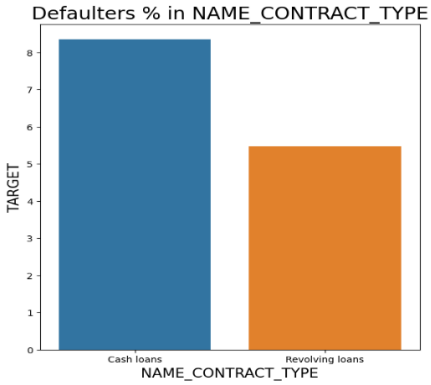
1. dataset : to put the dataset we want to use
2. col : column name for which we need to the analysis
3. target_col : column name for with which we will be comparing
4. ylog : to have y-axis in log10 terms, in case the plot is not readable
5. x_label_angle : to maintain the orientation of x-axis labels
6. h_layout : to give horizontal layout of the subplots

2. Categorical Variables Analysis

I. Segmented Univariate Analysis

Checking the contract type based on loan repayment status



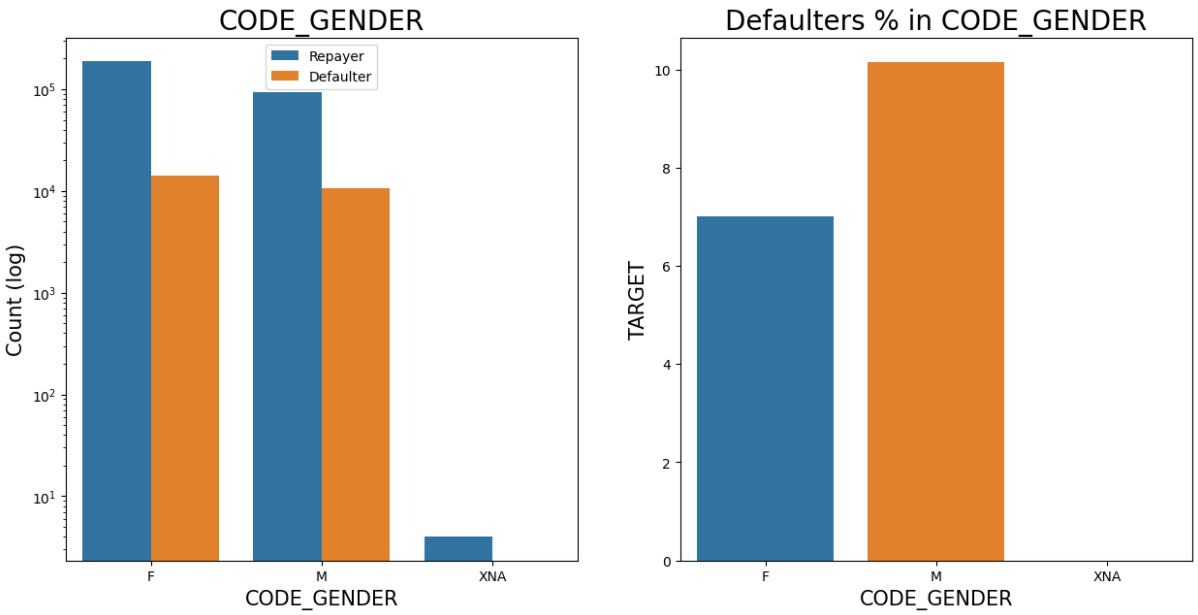


Insights: Contract type

- **Revolving loans are just a small fraction (10%) from the total number of loans**
- **Around 8-9% Cash loan applicants and 5-6% Revolving loan applicant are in defaulters**

Gender type based on loan repayment status

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

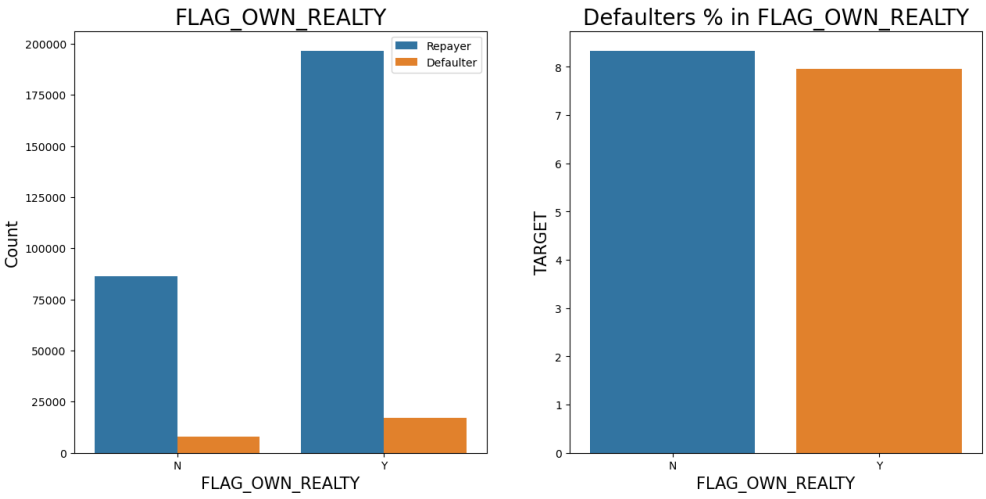


Insights: Gender Type

- **The number of female clients is almost double the number of male clients.**
- **Based on the percentage of defaulted credits, males have a higher chance of not returning their loans about 10%, comparing with women about 7%.**

Ownership of real estate based on loan repayment status

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

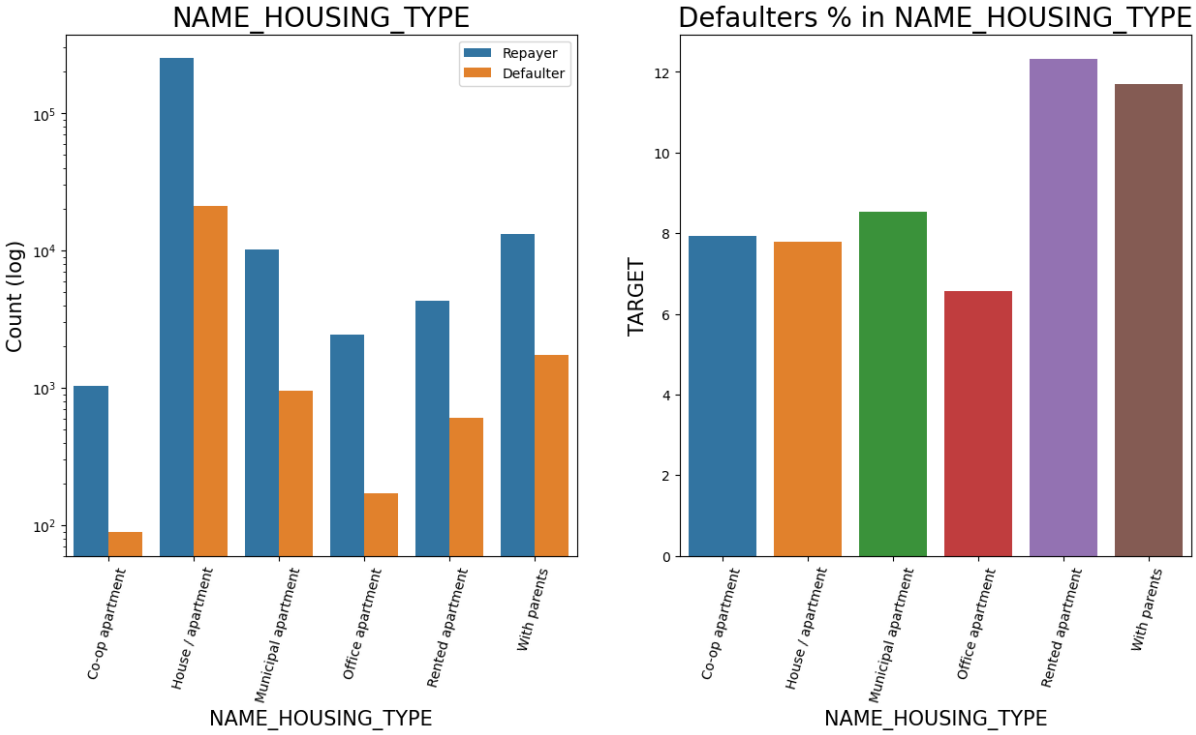


Insights:

- **The clients who own real estate are more than double of the ones that don't own.**
- **The defaulting rate of both categories are around the same (~8%). Thus, we can infer that there is no correlation between owning a reality and defaulting the loan.**

Housing Type based on loan repayment status

```
#4 Analyzing Housing Type based on loan repayment status
univariate(appl_data,"NAME_HOUSING_TYPE","TARGET",True,True,True)
```

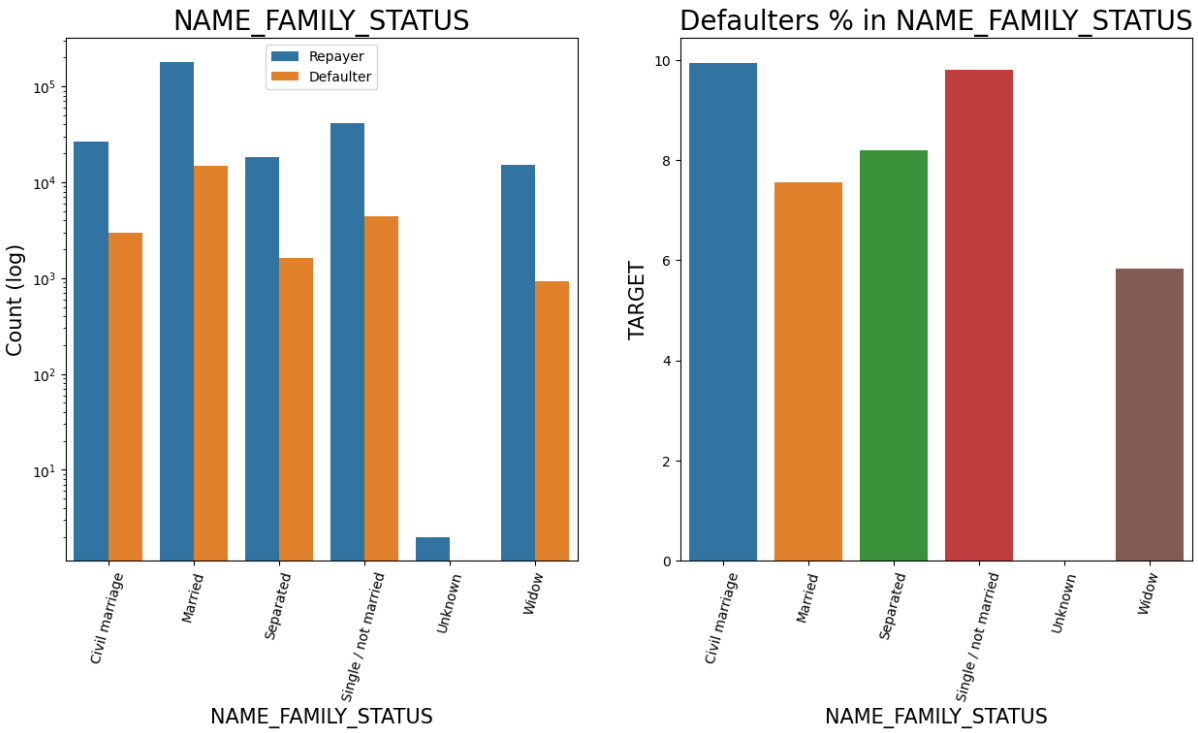


Insights: Applicant House type

- **Majority of people live in House/apartment**
- **People living in office apartments have lowest default rate**
- **People living with parents (~11.5%) and living in rented apartments(>12%) have higher probability of defaulting.**

Family status based on loan repayment status

```
#5 Analyzing Family status based on loan repayment status
univariate(appl_data,"NAME_FAMILY_STATUS","TARGET",True,True,True)
```

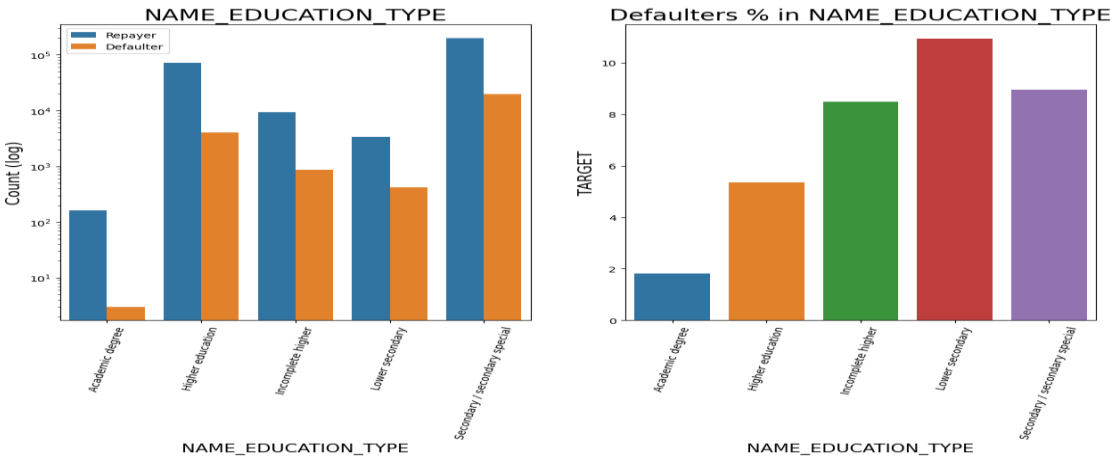


Insights:

- **Most of the people who have taken loan are married, followed by Single/not married and civil marriage**
- **In Percentage of defaulters, Civil marriage has the highest percent around (10%) and widow has the lowest around 6% (exception being Unknown).**

Education Type based on loan repayment status

```
#6 Analyzing Education Type based on loan repayment status
univariate(appl_data,"NAME_EDUCATION_TYPE","TARGET",True,True,True)
```

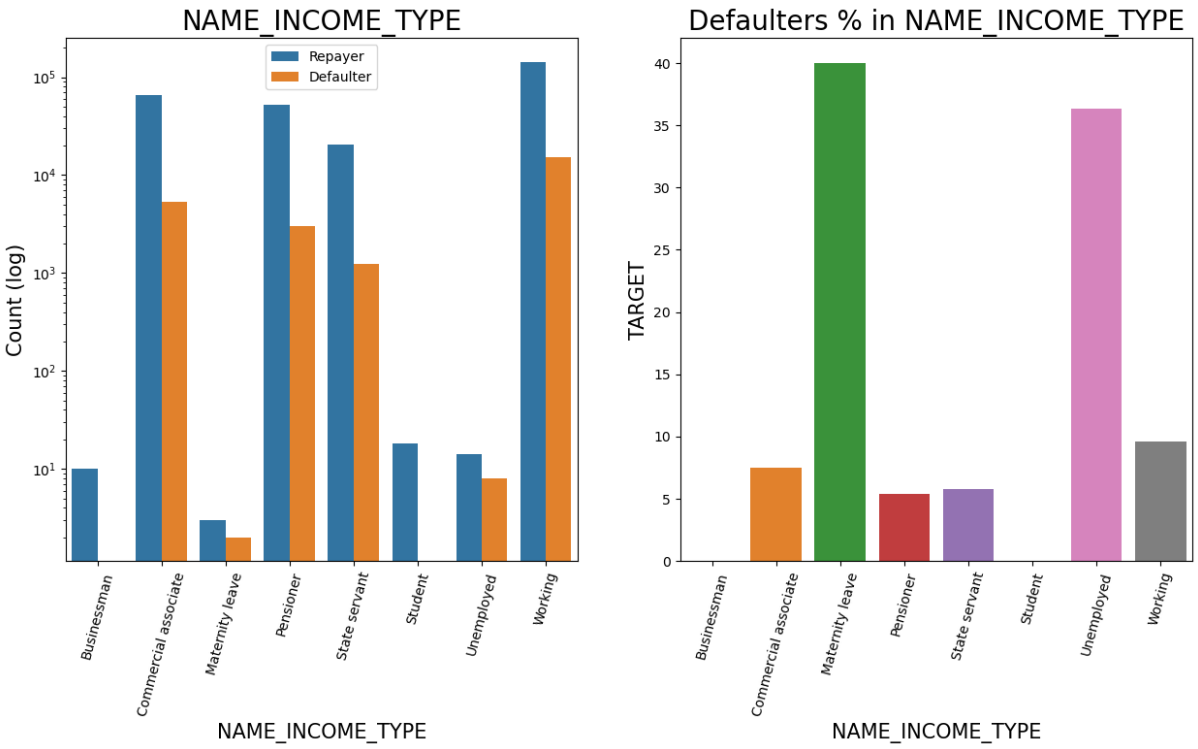


Insights: Education Type

- **Majority of clients have Secondary/secondary special education, followed by clients with Higher education.**
- **Very few clients have an academic degree**
- **Lower secondary category has highest rate of defaulting around 11%.**
- **People with Academic degree are least likely to default.**

Income Type based on loan repayment status

```
#7 Analyzing Income Type based on loan repayment status
univariate(appl_data,"NAME_INCOME_TYPE","TARGET",True,True,True)
```

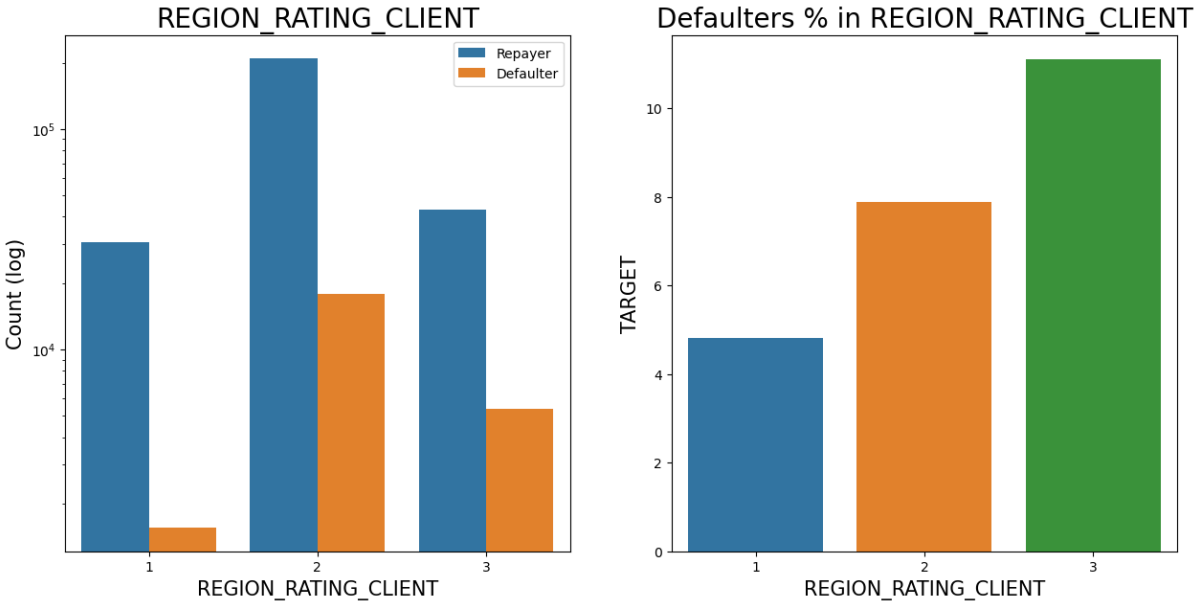


Insights:

- **Most of applicants for loans income type is Working, followed by Commercial associate, Pensioner and State servant.**
- **The applicants who are on Maternity leave have defaulting percentage of 40% which is the highest, followed by Unemployed (37%). The rest under average around 10% defaulters.**
- **Student and Businessmen though less in numbers, do not have default record. Safest two categories for providing loan.**

Region rating where applicant lives based on loan repayment status

```
#8 Analyzing Region rating where applicant lives based on loan repayment status
univariate(appl_data,"REGION_RATING_CLIENT","TARGET",True,False,True)
```

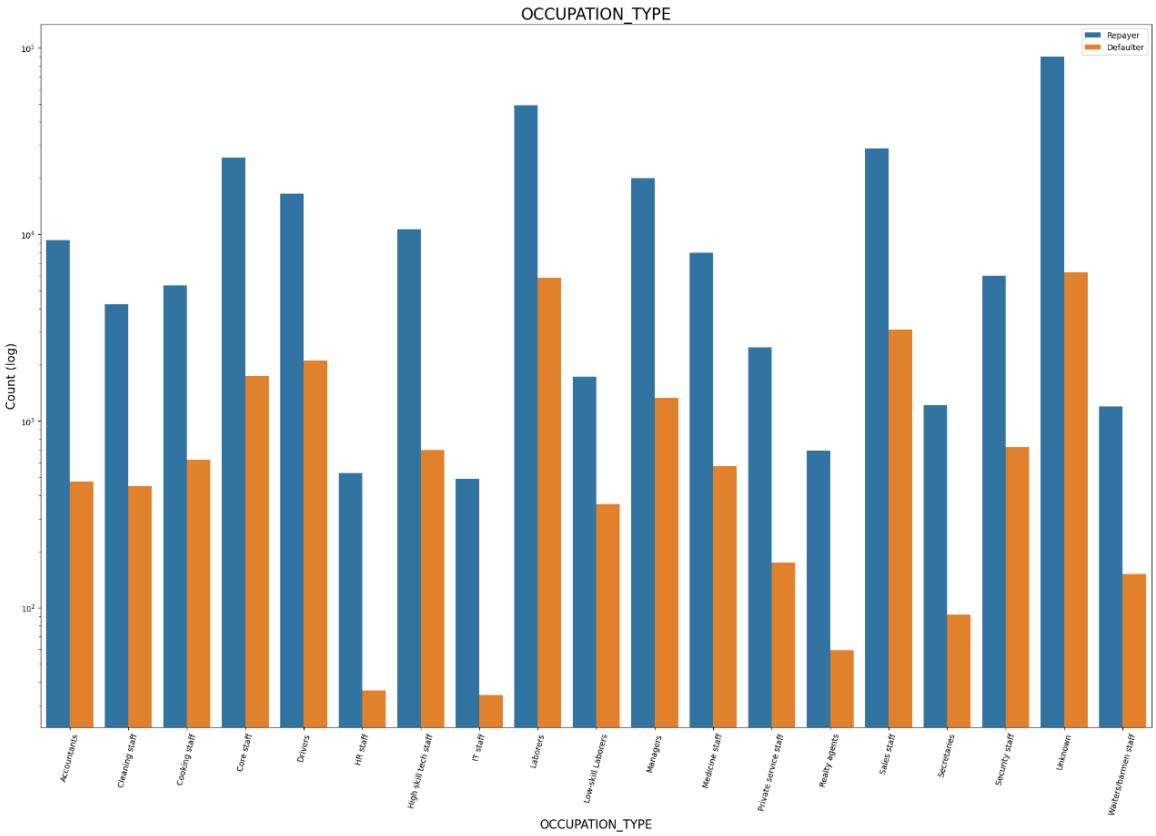


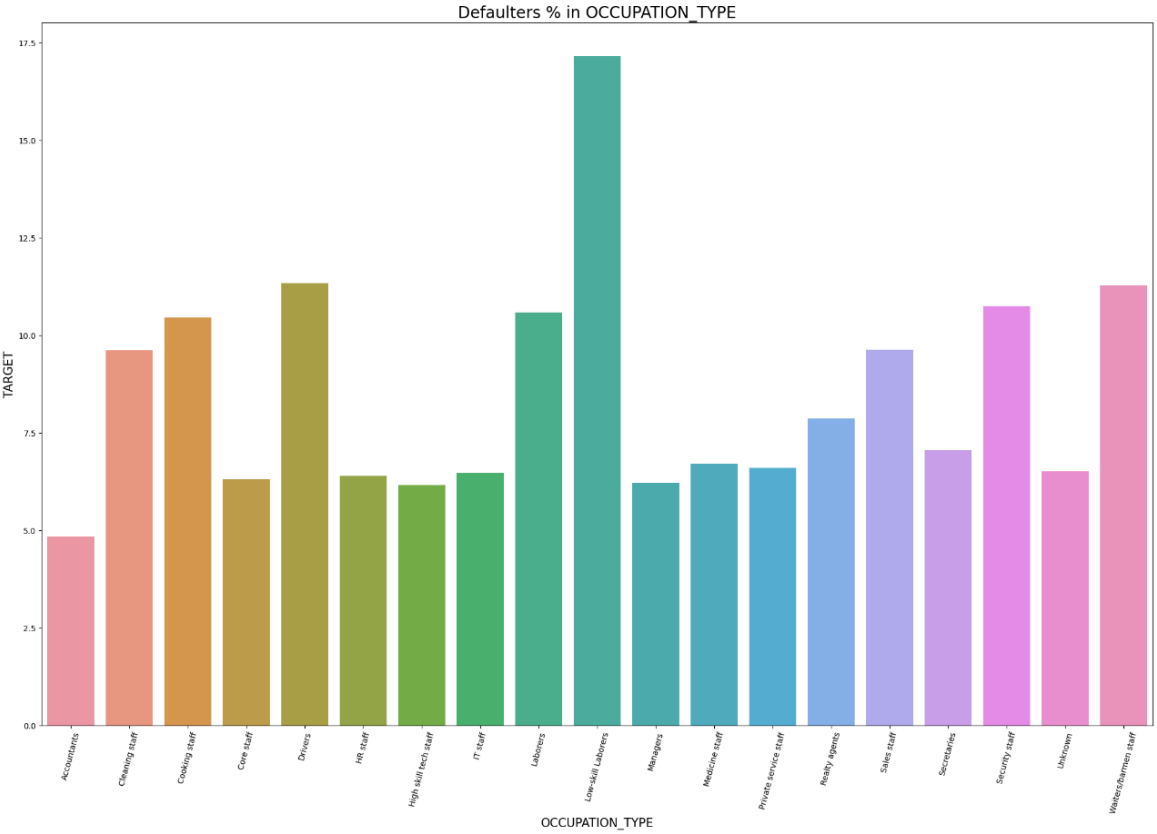
Insights: Client Region Rating

- **Most of the applicants are living in Region with Rating 2 place.**
- **Region Rating 3 has the highest default rate (11%)**
- **Applicant living in Region_Rating 1 has the lowest probability of defaulting, thus safer for approving loans.**

Occupation Type where applicant lives based on loan repayment status

```
#9 Analyzing Occupation Type where applicant lives based on loan repayment status
univariate(appl_data,"OCCUPATION_TYPE","TARGET",True,True,False)
```



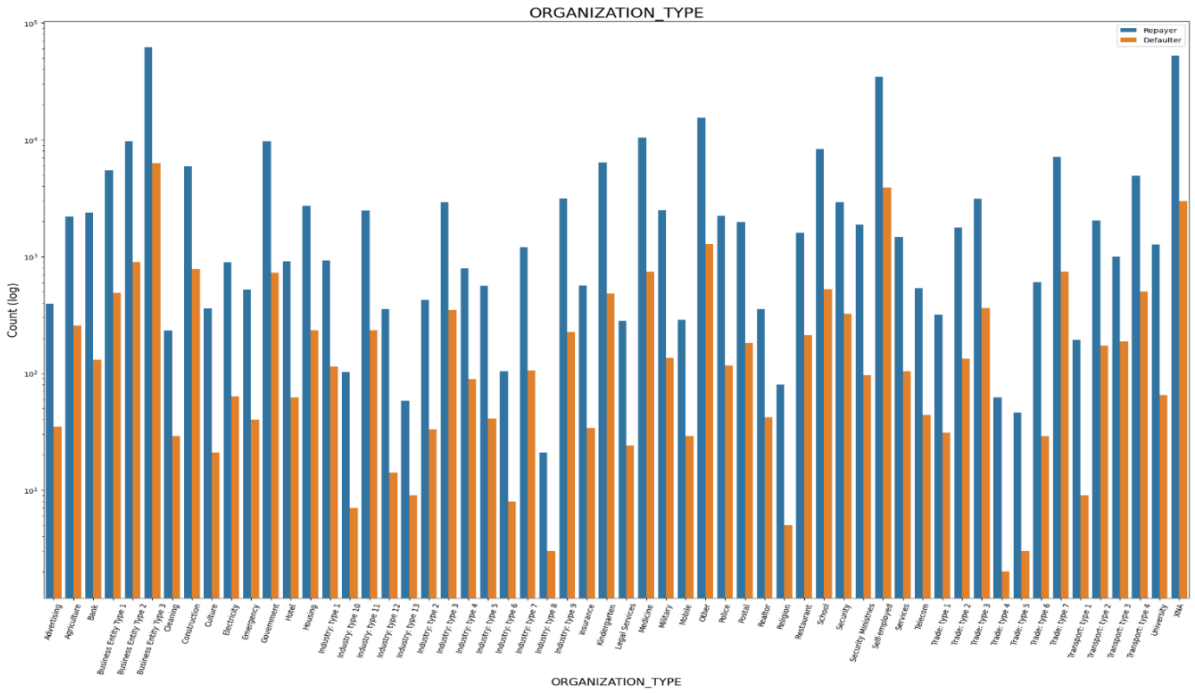


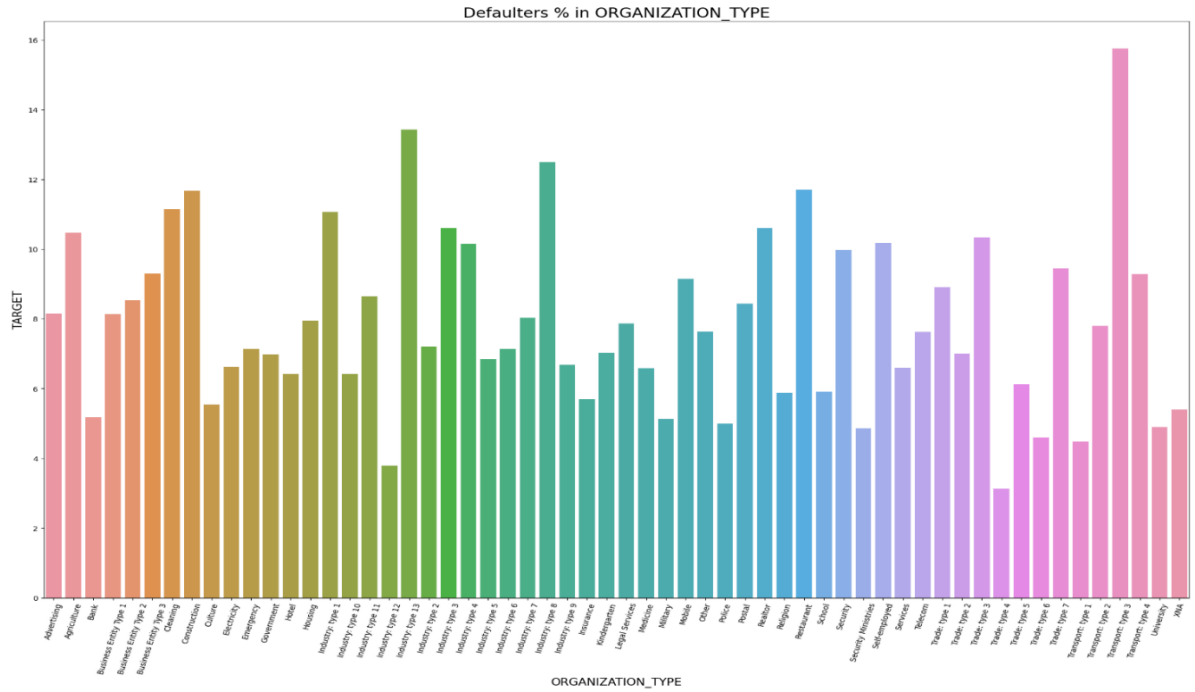
Insights:

- **Most of the loans are taken by Laborers, followed by Sales staff.**
- **IT staff are less likely to apply for Loan.**
- **Category with highest percent of defaulters is Low-skill Laborers (above 17%), followed by Drivers and Waiters/barmen staff, Security staff, Laborers and Cooking staff.**

Loan repayment status based on Organization type

```
#10 Checking Loan repayment status based on Organization type

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



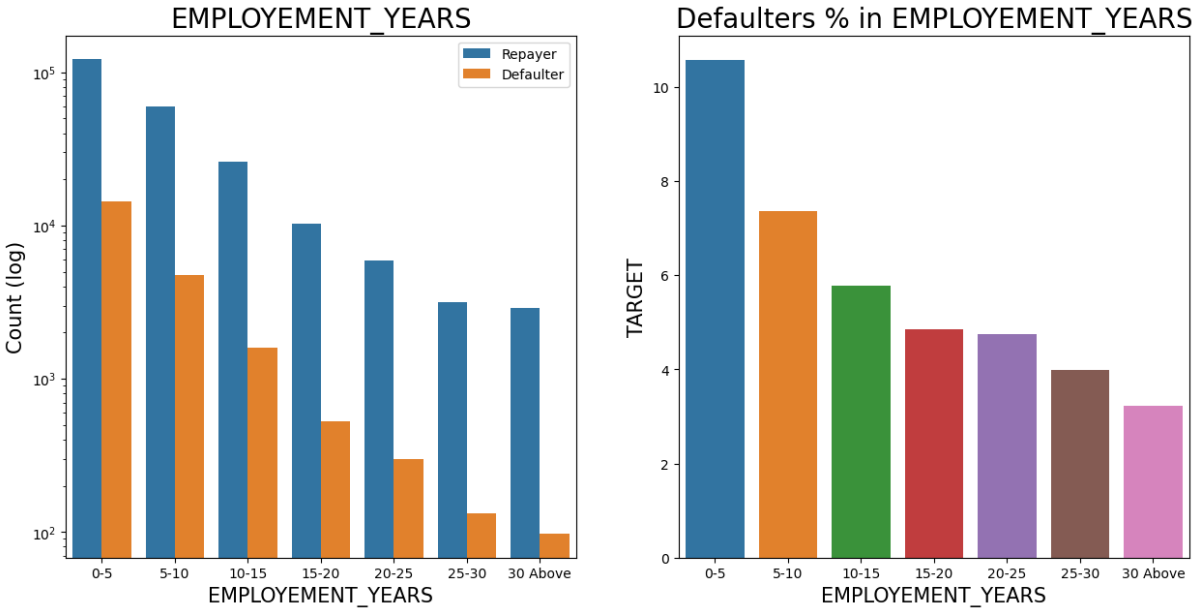


Insights: Organization Type

- **Organizations with highest percent of defaulters are Transport: type 3 (16%), Industry: type 13 (13.5%), Industry: type 8 (12.5%) and Restaurant (less than 12%).**
- **Self-employed people have relative high defaulting rate, to be safer side loan disbursement should be avoided or provide loan with higher interest rate to mitigate the risk of defaulting.**
- **Most of the people application for loan are from Business Entity Type 3**
- **For a very high number of applications, Organization type information is unavailable(XNA)**
- **It can be seen that following category of organization type has lesser defaulters thus safer for providing loans: Trade Type 4 and 5, Industry type 8**

Employment_Year based on loan repayment status

```
#11 Analyzing Employment_Year based on loan repayment status
univariate(appl_data,"EMPLOYEMENT_YEARS","TARGET",True,False,True)
```

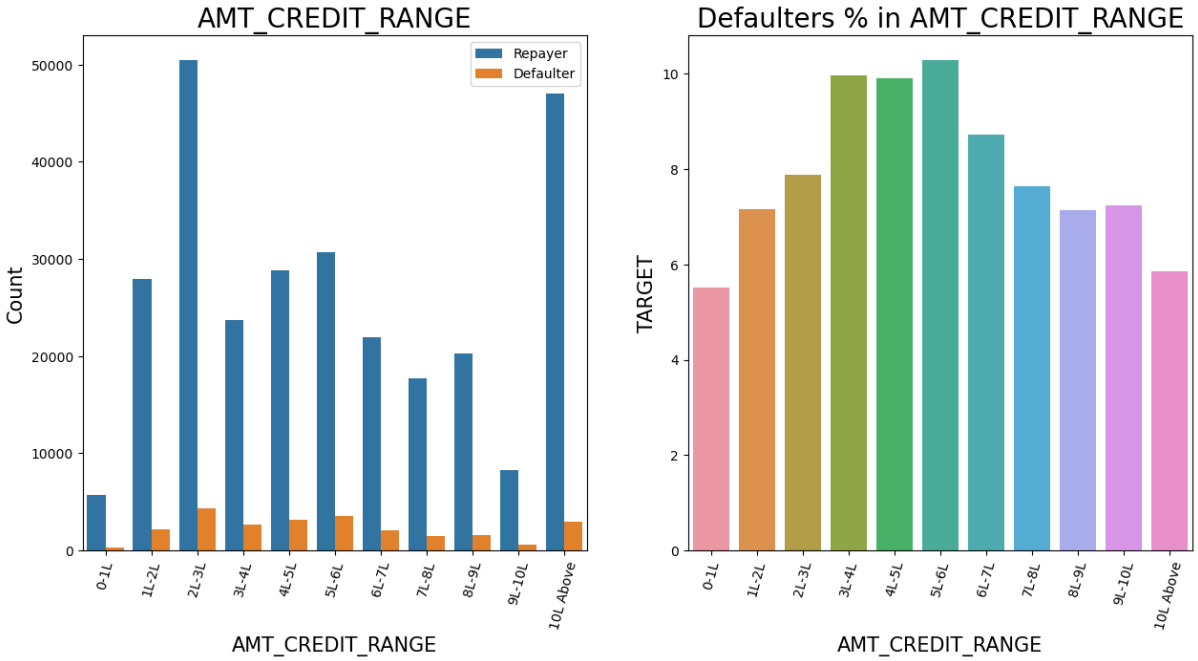


Insights: Employment in Years

- **Majority of the applicants having working experience between 0-5 years are defaulters. The defaulting rating of this group is also the highest which is around 10%**
- **With increase of employment year, defaulting rate is gradually decreasing.**
- **with people having 40+ year experience has less than 1% default rate.**

Amount_Credit based on loan repayment status

```
#12 Analyzing Amount_Credit based on loan repayment status

univariate(appl_data,"AMT_CREDIT_RANGE","TARGET",False,True,True)
```

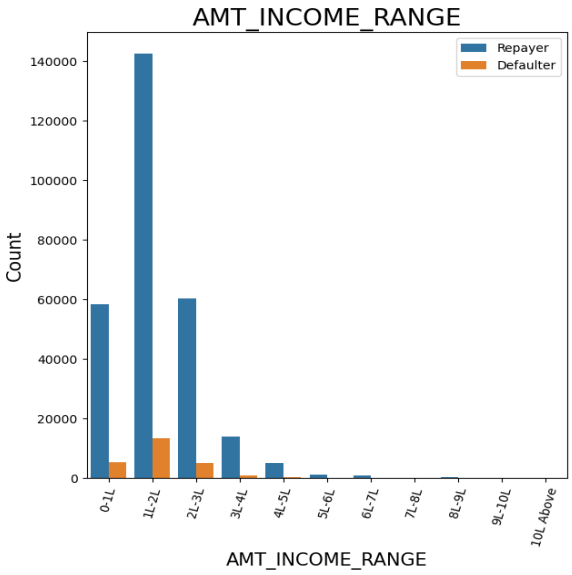


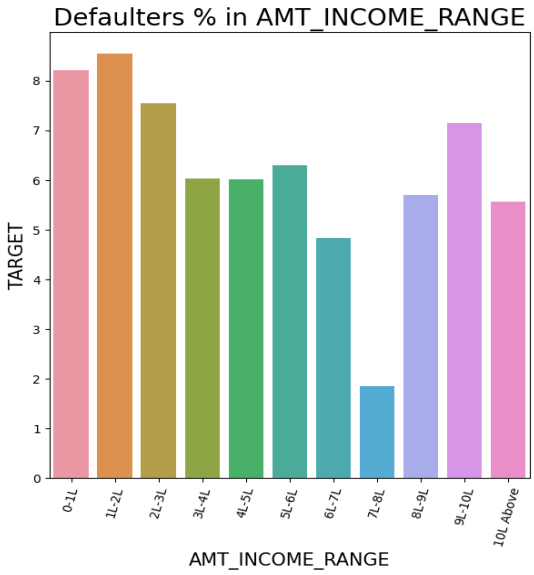
Insights: Loan Amount

- **there are high number of applicants have loan in range of 2-3 Lakhs followed by 10 Lakh above range.**
- **People who get loan for 3-6 Lakhs have the greatest number of defaulters than other loan range.**

Amount_Income Range based on loan repayment status

```
#13  Analyzing Amount_Income Range based on loan repayment status

univariate(appl_data,"AMT_INCOME_RANGE","TARGET",False,True,True)
```



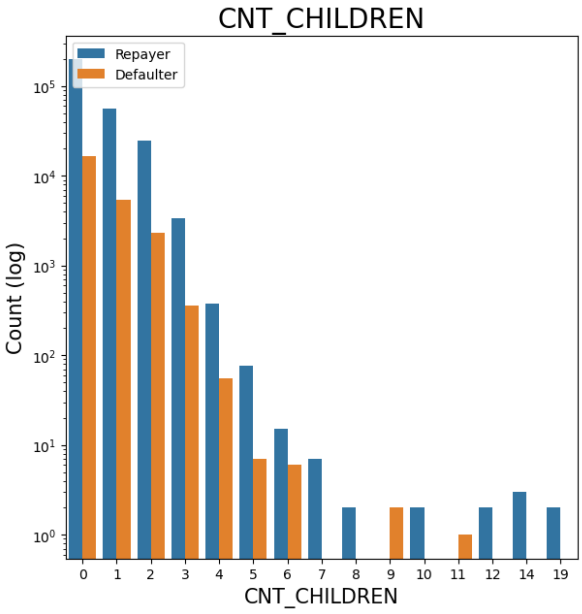


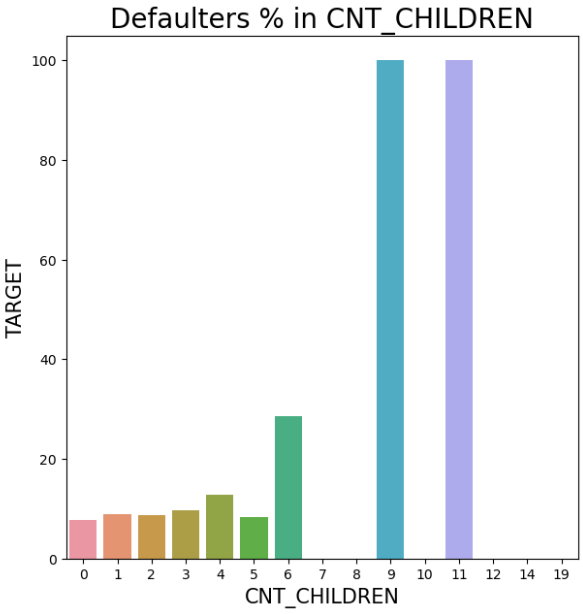
Insights: Applicant Income

- **Majority of the applications have Income total less than 3 Lakhs.**
- **Application with Income less than 3 Lakhs has high probability of defaulting**
- **Applicant with Income 7-8 Lakhs are less likely to default.**

Number of children based on loan repayment status

```
#14  Analyzing Number of children based on loan repayment status

univariate(appl_data,"CNT_CHILDREN","TARGET",True,False,True)
```





Insights: Client Children's Count

- **Most of the applicants do not have children.**
- **Very few clients have more than 3 children.**

- **Client who has more than 4 children has a very high default rate with child count 9 and 11 showing 100% default rate.**

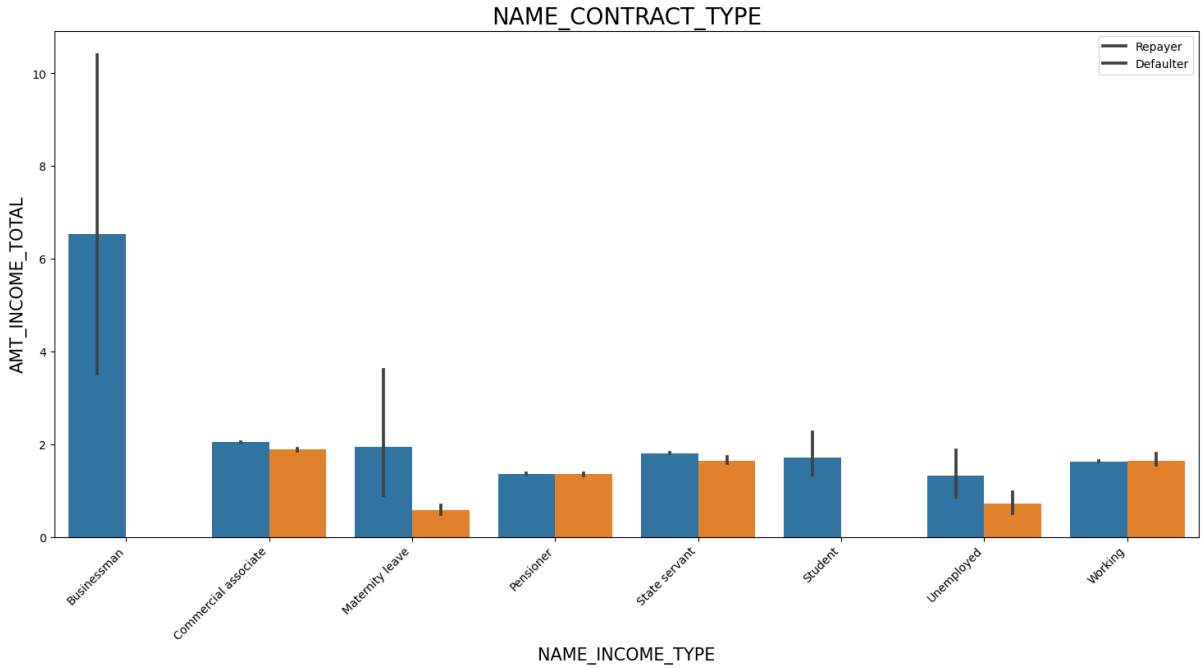
II. Categorical Bivariate or Multivariate Analysis

```
appl_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

Income type vs Income Amount Range

```
# Income type vs Income Amount Range on a Seaborn Barplot
bivariate_c("NAME_INCOME_TYPE","AMT_INCOME_TOTAL",appl_data,"TARGET",(18,8),['Repayer','Defaulter'])
```



Insights:

- **It can be seen that Businessman income is the highest and the estimated range with default 95% confidence level seem to indicate that the income of a Businessman could be in the range of slightly close to 4 lakhs and slightly above 10 lakhs.**

3. Numeric Variables Analysis

I. Bisecting the app_data dataframe based on Target value 0 and 1 for correlation and other analysis

```
#Listing all the columnns of dataframe "appl_data"
appl_data.columns
```

```
Index(['SK_ID_CURR', 'TARGET', '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', 'FLAG
_MOBIL', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_STA
RT', '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', 'DE
F_30_CNT_SOCIAL_CIRCLE', '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', 'AMT_INCOME_RANGE', 'AMT_CREDIT_
RANGE', 'AMT_GOODS_PRICE_RANGE', 'AGE', 'AGE_GROUP', 'YEARS_EMPLOYED', 'EMPLOYEMENT_YEARS'],
      dtype='object')
```

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

cols_for_correlation = ['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
Repayer_df = appl_data.loc[appl_data['TARGET']==0, cols_for_correlation]

# Defaulters dataframe
Defaulter_df = appl_data.loc[appl_data['TARGET']==1, cols_for_correlation]
```

```
len(cols_for_correlation)
```

```
41
```

4. Correlation between numeric variable

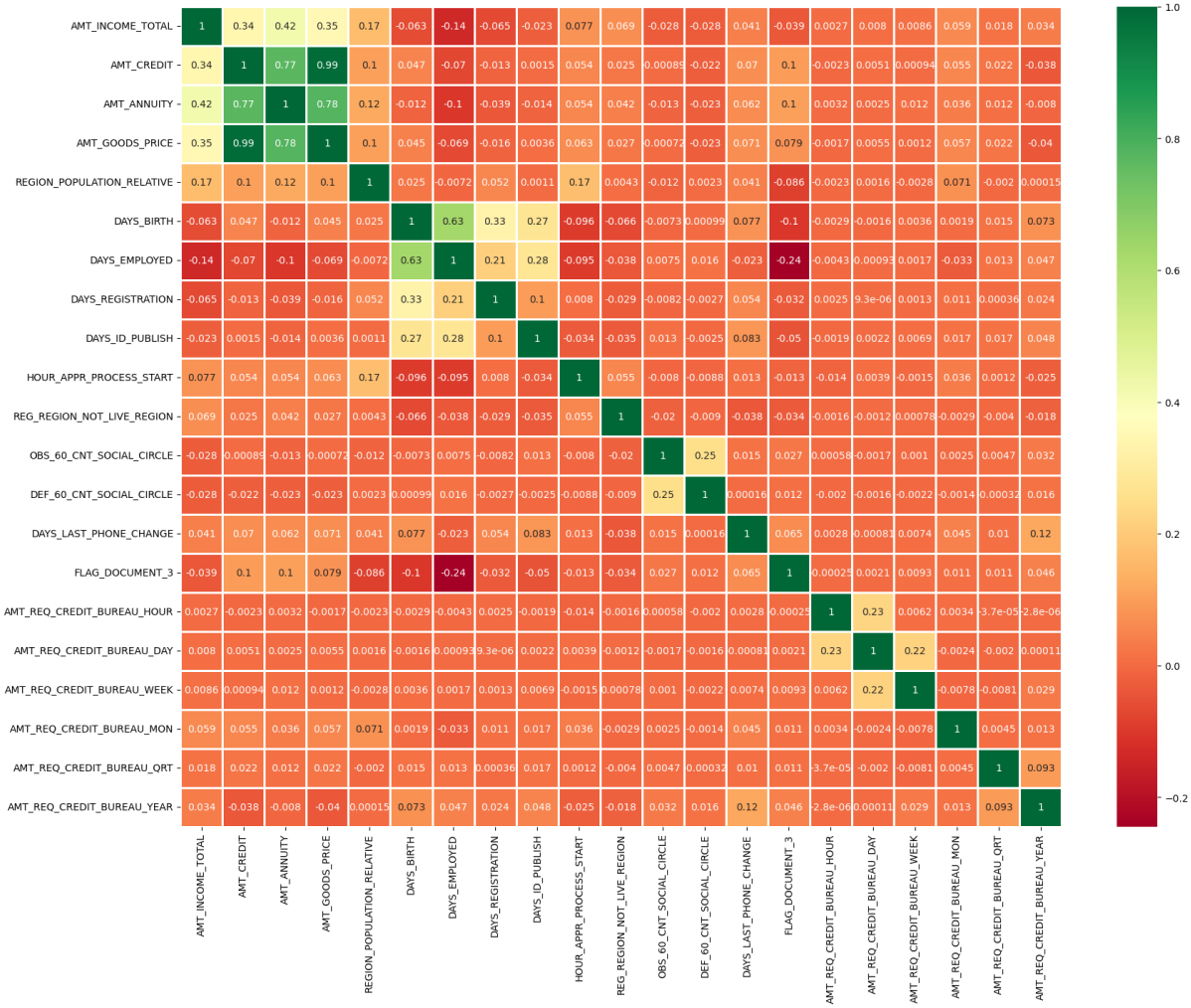
Correlation for the Repayer dataframe

```
# Getting  top 10 correlation for the Repayers dataframe

corr_repayer = Repayer_df.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

Linear correlation among Repayer

```
#plotting heatmap to see linear correlation amoung Repayers

fig = plt.figure(figsize=(20,15))
ax = sns.heatmap(Repayer_df.corr(), cmap="RdYlGn",annot=True,linewidth =1)
```



Insights: Correlating factors amongst Repayer

1. Credit amount is highly correlated with:

- **Goods Price Amount**
- **Loan Annuity**
- **Total Income**

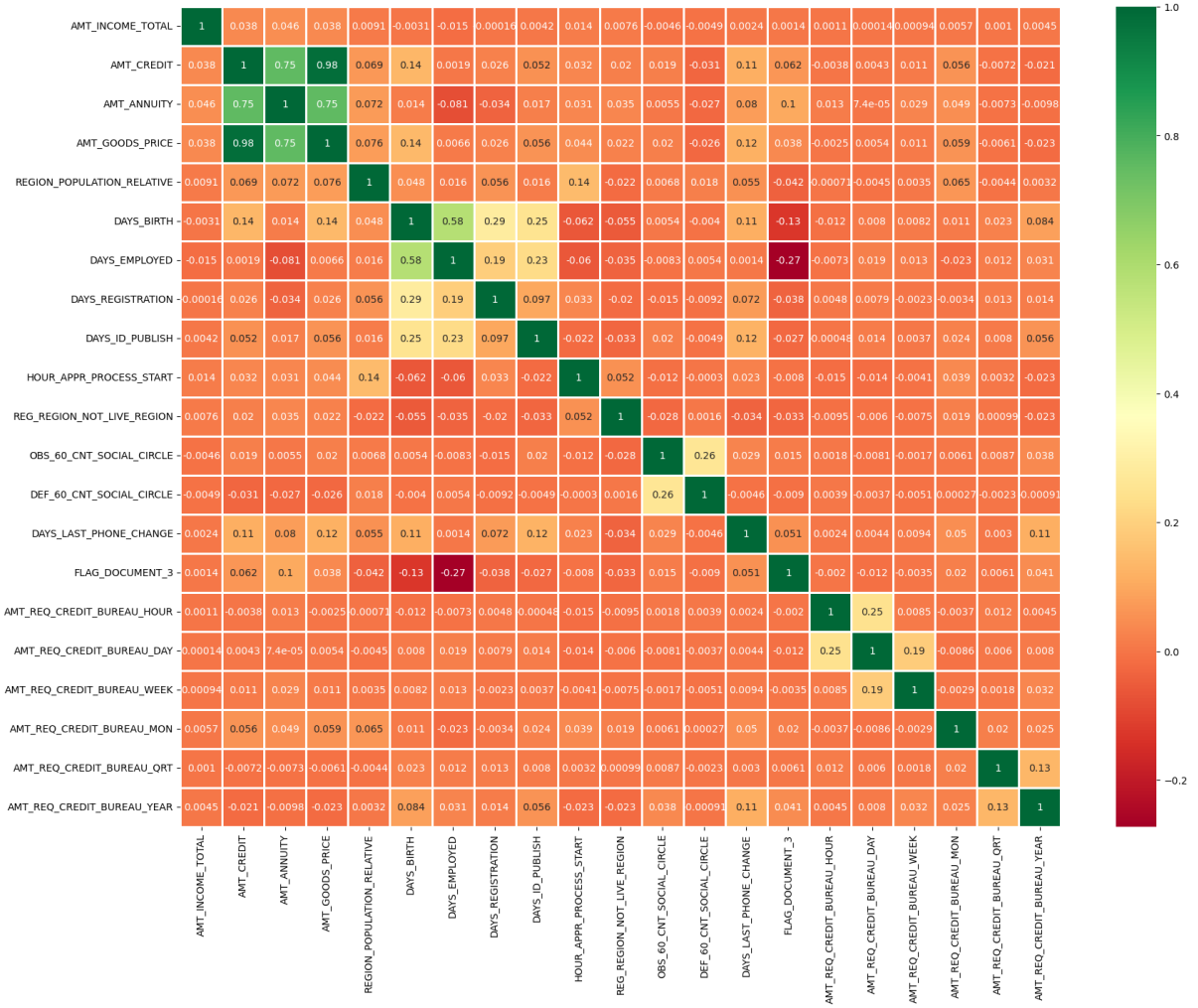
2. We can also see that Repayer have high correlation in number of days employed.

Correlation for the Defaulter dataframe

```
# Getting the top 10 correlation for the Defaulter data
corr_Defaulter = Defaulter_df.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	0.264159
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

```
fig = plt.figure(figsize=(20,15))
ax = sns.heatmap(Defaulter_df.corr(), cmap="RdYlGn",annot=True,linewidth =1)
```

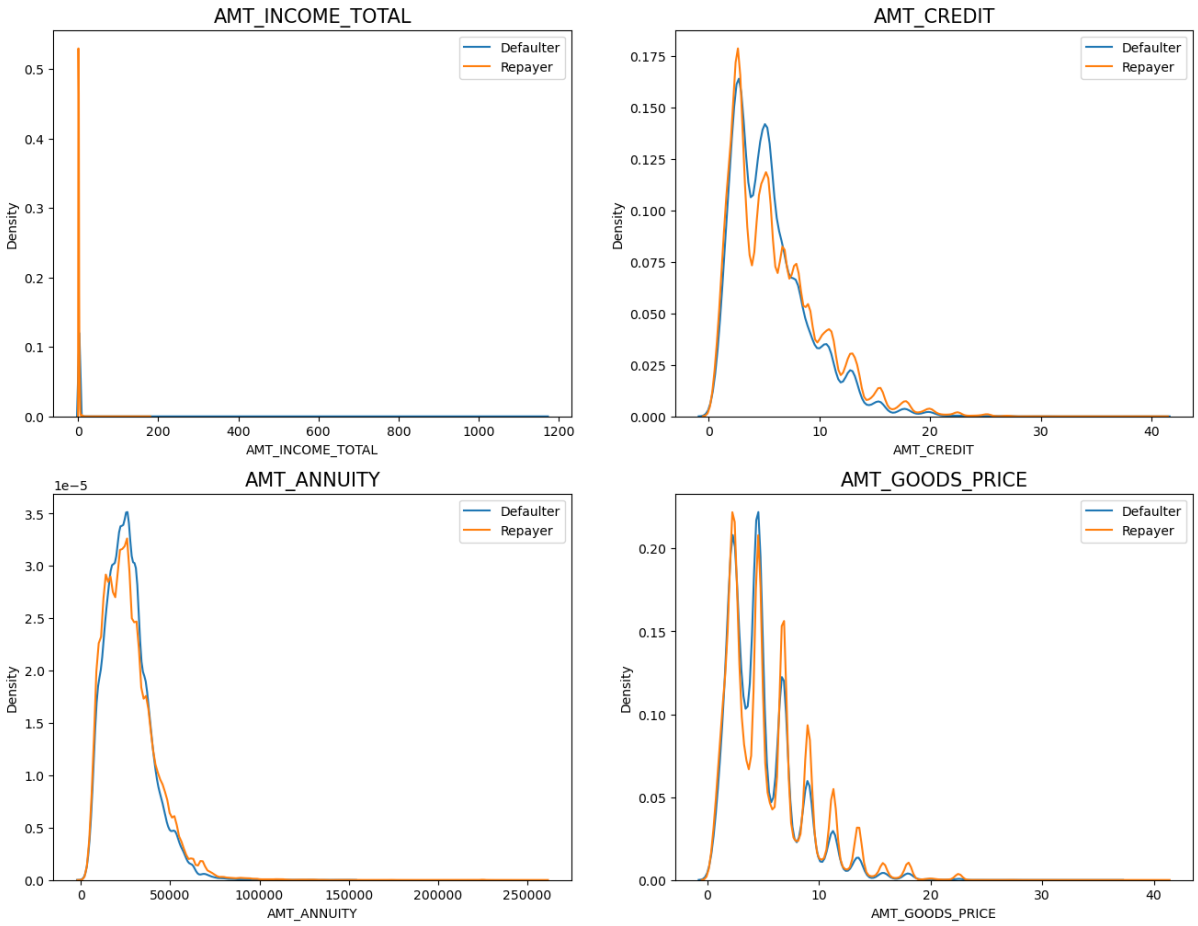


Insights: Correlating factors amongst Repayer

- **Credit amount is highly correlated with good price amount which is same as Repayer.**
- **Loan annuity correlation with credit amount has slightly reduced in defaulters(0.75) when compared to Repayer(0.77)**
- **We can also see that Repayer have high correlation in number of days employed(0.62) when compared to defaulters(0.58).**
- **There is a severe drop in the correlation between total income of the client and the credit amount(0.038) amongst defaulters whereas it is 0.342 among Repayer.**
- **Days_birth and number of children correlation has reduced to 0.259 in defaulters when compared to 0.337 in Repayer.**
- **There is a slight increase in defaulted to observed count in social circle among defaulters(0.264) when compared to Repayer(0.254).**

5. Numerical Univariate Analysis

```
# Plotting the numerical columns related to amount as distribution plot to see density
amount = appl_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(Defaulter_df[i[1]], hist=False,label ="Defaulter")
    sns.distplot(Repayer_df[i[1]], hist=False, label ="Repayer")
    plt.title(i[1], fontdict={'fontsize' : 15, 'fontweight' : 5})
    plt.legend()


plt.show()
```

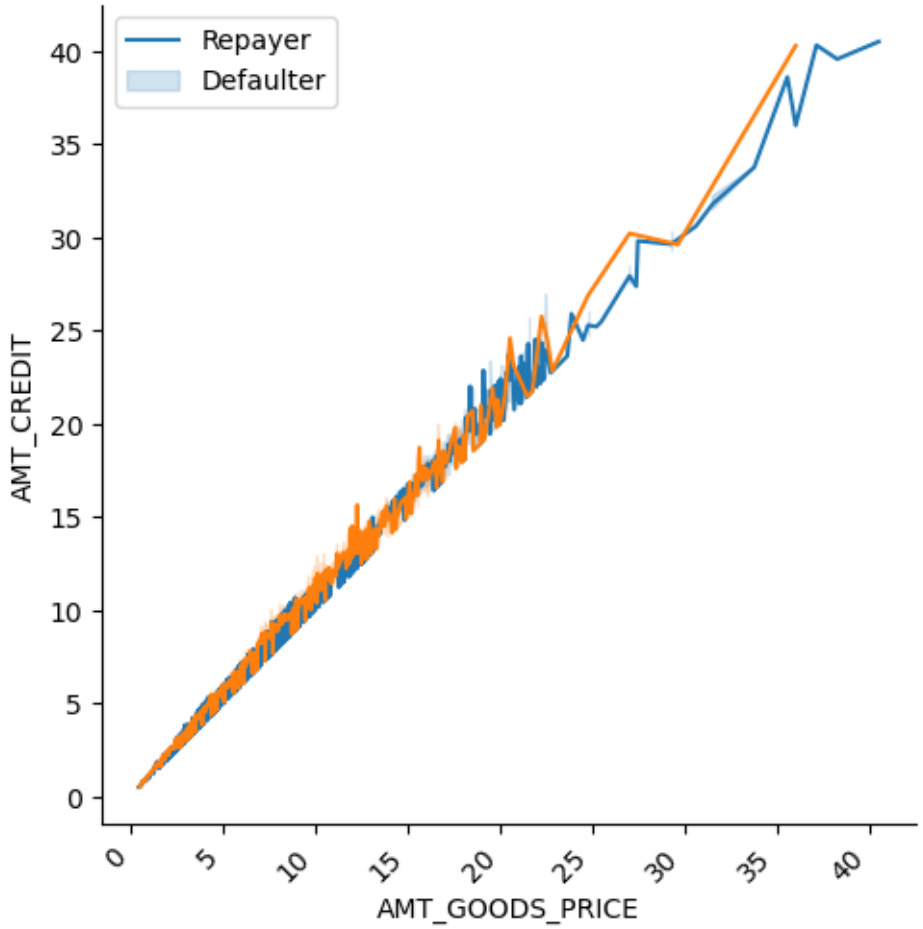


Insights:

- **Most no of loans are given for goods price below 10 lakhs.**
- **Most people pay annuity below 50K for the credit loan.**
- **Credit amount of the loan is mostly less than 10 lakhs.**
- **The repayers and defaulters' distribution overlap in all the plots and hence we cannot use any of these variables in isolation to make a decision.**

6. Numerical Bivariate Analysis

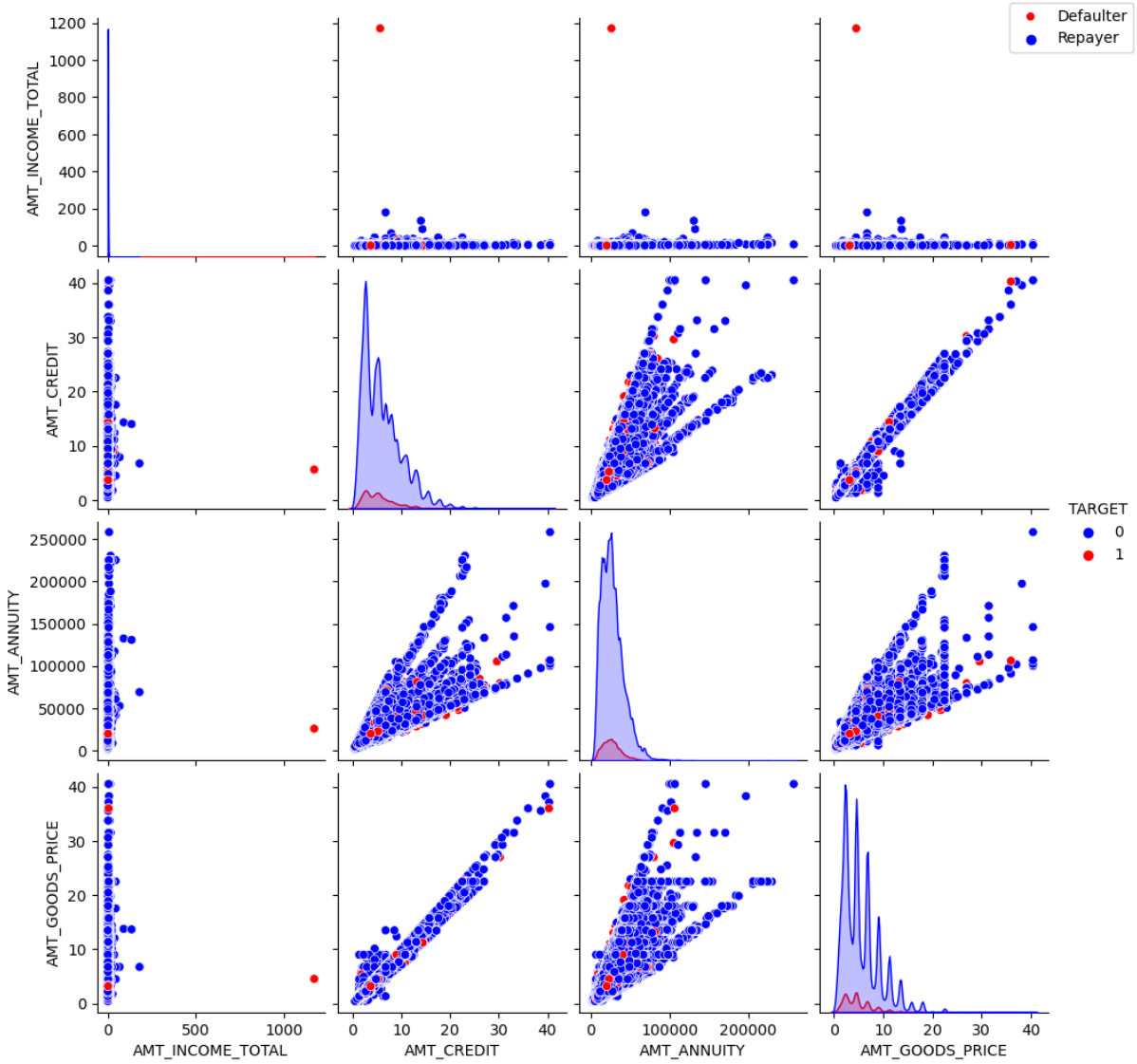
```
# Checking the relationship between Goods price and credit and comparing with loan repayment staus
bivariate_n('AMT_GOODS_PRICE','AMT_CREDIT',appl_data,"TARGET", "line",['Repayer','Defaulter'])
```



Insights:

- **Most no of loans are given for goods price below 10 lakhs.**
- **Most people pay annuity below 50K for the credit loan.**
- **Credit amount of the loan is mostly less than 10 lakhs.**
- **The repayers and defaulters' distribution overlap in all the plots and hence we cannot use any of these variables in isolation to make a decision**

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

amount = appl_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=["b","r"])
ax.fig.legend(labels=['Defaulter','Repayer'])
plt.show()
```



Insights:

- **When Annuity Amount > 15K and Good Price Amount > 20 Lakhs, there is a lesser chance of defaulters**
- **Loan Amount(AMT_CREDIT) and Goods price(AMT_GOODS_PRICE) are highly correlated as based on the scatterplot where most of the data are consolidated in form of a line**
- **There are very less defaulters for AMT_CREDIT >20 Lakhs**

7. Merged Dataframe Analysis

```
# merge both the dataframe on SK_ID_CURR with Inner Joins
loan_df = pd.merge(appl_data, prev_appl, how='inner', on='SK_ID_CURR')
loan_df.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANN
0	100002	1	Cash loans	M	Y	0	2.025	4.065975	
1	100003	0	Cash loans	F	N	0	2.700	12.935025	
2	100003	0	Cash loans	F	N	0	2.700	12.935025	
3	100003	0	Cash loans	F	N	0	2.700	12.935025	
4	100004	0	Revolving loans	M	Y	0	0.675	1.350000	

```
#Checking the details of the merged dataframe
loan_df.shape
```

```
(1413701, 82)
```

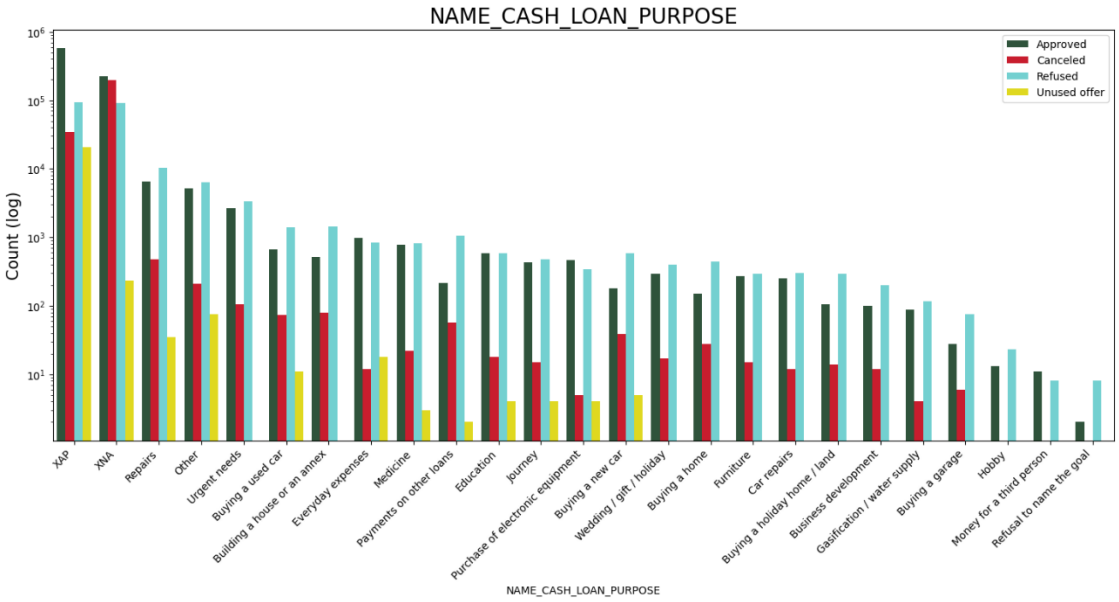
```
# checking the columns and column types of the dataframe
loan_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1413701 entries, 0 to 1413700
Data columns (total 82 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   SK_ID_CURR                   1413701 non-null  int64
 1   TARGET                       1413701 non-null  int64
 2   NAME_CONTRACT_TYPE_x         1413701 non-null  category
 3   CODE_GENDER                  1413701 non-null  category
 4   FLAG_OWN_REALTY              1413701 non-null  category
 5   CNT_CHILDREN                 1413701 non-null  category
 6   AMT_INCOME_TOTAL             1413701 non-null  float64
 7   AMT_CREDIT_x                 1413701 non-null  float64
 8   AMT_ANNUITY_x                1413608 non-null  float64
 9   AMT_GOODS_PRICE_x            1412493 non-null  float64
 10  NAME_TYPE_SUITE_x            1410175 non-null  category
 11  NAME_INCOME_TYPE             1413701 non-null  category
 12  NAME_EDUCATION_TYPE          1413701 non-null  category
 13  NAME_FAMILY_STATUS           1413701 non-null  category
 14  NAME_HOUSING_TYPE            1413701 non-null  category
 15  REGION_POPULATION_RELATIVE   1413701 non-null  float64
 16  DAYS_BIRTH                   1413701 non-null  int64
 17  DAYS_EMPLOYED                1413701 non-null  int64
 18  DAYS_REGISTRATION            1413701 non-null  float64
 19  DAYS_ID_PUBLISH              1413701 non-null  int64
 20  FLAG_MOBIL                   1413701 non-null  int64
 21  OCCUPATION_TYPE              1413701 non-null  category
 22  CNT_FAM_MEMBERS              1413701 non-null  category
 23  REGION_RATING_CLIENT         1413701 non-null  category
 24  REGION_RATING_CLIENT_W_CITY  1413701 non-null  category
 25  WEEKDAY_APPR_PROCESS_START   1413701 non-null  category
 26  HOUR_APPR_PROCESS_START      1413701 non-null  int64
 27  REG_REGION_NOT_LIVE_REGION   1413701 non-null  int64
 28  REG_REGION_NOT_WORK_REGION   1413701 non-null  category
 29  LIVE_REGION_NOT_WORK_REGION  1413701 non-null  category
 30  REG_CITY_NOT_LIVE_CITY       1413701 non-null  category
 31  REG_CITY_NOT_WORK_CITY       1413701 non-null  category
 32  LIVE_CITY_NOT_WORK_CITY      1413701 non-null  category
 33  ORGANIZATION_TYPE            1413701 non-null  category
 34  OBS_30_CNT_SOCIAL_CIRCLE     1410555 non-null  float64
 35  DEF_30_CNT_SOCIAL_CIRCLE     1410555 non-null  float64
 36  OBS_60_CNT_SOCIAL_CIRCLE     1410555 non-null  float64
 37  DEF_60_CNT_SOCIAL_CIRCLE     1410555 non-null  float64
 38  DAYS_LAST_PHONE_CHANGE       1413701 non-null  float64
 39  FLAG_DOCUMENT_3              1413701 non-null  int64
 40  AMT_REQ_CREDIT_BUREAU_HOUR   1413701 non-null  float64
 41  AMT_REQ_CREDIT_BUREAU_DAY    1413701 non-null  float64
 42  AMT_REQ_CREDIT_BUREAU_WEEK   1413701 non-null  float64
 43  AMT_REQ_CREDIT_BUREAU_MON    1413701 non-null  float64
 44  AMT_REQ_CREDIT_BUREAU_QRT    1413701 non-null  float64
 45  AMT_REQ_CREDIT_BUREAU_YEAR   1413701 non-null  float64
 46  AMT_INCOME_RANGE             1413024 non-null  category
 47  AMT_CREDIT_RANGE             1413701 non-null  category
 48  AMT_GOODS_PRICE_RANGE        1412493 non-null  category
 49  AGE                          1413701 non-null  float64
 50  AGE_GROUP                    1413701 non-null  category
 51  YEARS_EMPLOYED               1413701 non-null  float64
 52  EMPLOYEMENT_YEARS            1140109 non-null  category
 53  SK_ID_PREV                   1413701 non-null  int64
 54  NAME_CONTRACT_TYPE_y         1413701 non-null  category
 55  AMT_ANNUITY_y                1413701 non-null  float64
 56  AMT_APPLICATION              1413701 non-null  float64
 57  AMT_CREDIT_y                 1413700 non-null  float64
 58  AMT_GOODS_PRICE_y            1413701 non-null  float64
 59  NAME_CASH_LOAN_PURPOSE       1413701 non-null  category
 60  NAME_CONTRACT_STATUS         1413701 non-null  category
 61  DAYS_DECISION                1413701 non-null  int64
 62  NAME_PAYMENT_TYPE            1413701 non-null  category
 63  CODE_REJECT_REASON           1413701 non-null  category
 64  NAME_TYPE_SUITE_y            1413701 non-null  object
 65  NAME_CLIENT_TYPE             1413701 non-null  category
 66  NAME_GOODS_CATEGORY          1413701 non-null  category
 67  NAME_PORTFOLIO               1413701 non-null  category
 68  NAME_PRODUCT_TYPE            1413701 non-null  category
 69  CHANNEL_TYPE                 1413701 non-null  category
 70  SELLERPLACE_AREA             1413701 non-null  int64
 71  NAME_SELLER_INDUSTRY         1413701 non-null  category
 72  CNT_PAYMENT                  1413701 non-null  float64
 73  NAME_YIELD_GROUP             1413701 non-null  category
 74  PRODUCT_COMBINATION          1413388 non-null  category
 75  DAYS_FIRST_DRAWING           852595 non-null   float64
 76  DAYS_FIRST_DUE               852595 non-null   float64
 77  DAYS_LAST_DUE_1ST_VERSION    852595 non-null   float64
 78  DAYS_LAST_DUE                852595 non-null   float64
 79  DAYS_TERMINATION             852595 non-null   float64
 80  NFLAG_INSURED_ON_APPROVAL    852595 non-null   float64
 81  YEARLY_DECISION              1413701 non-null  category
dtypes: category(39), float64(30), int64(12), object(1)
```

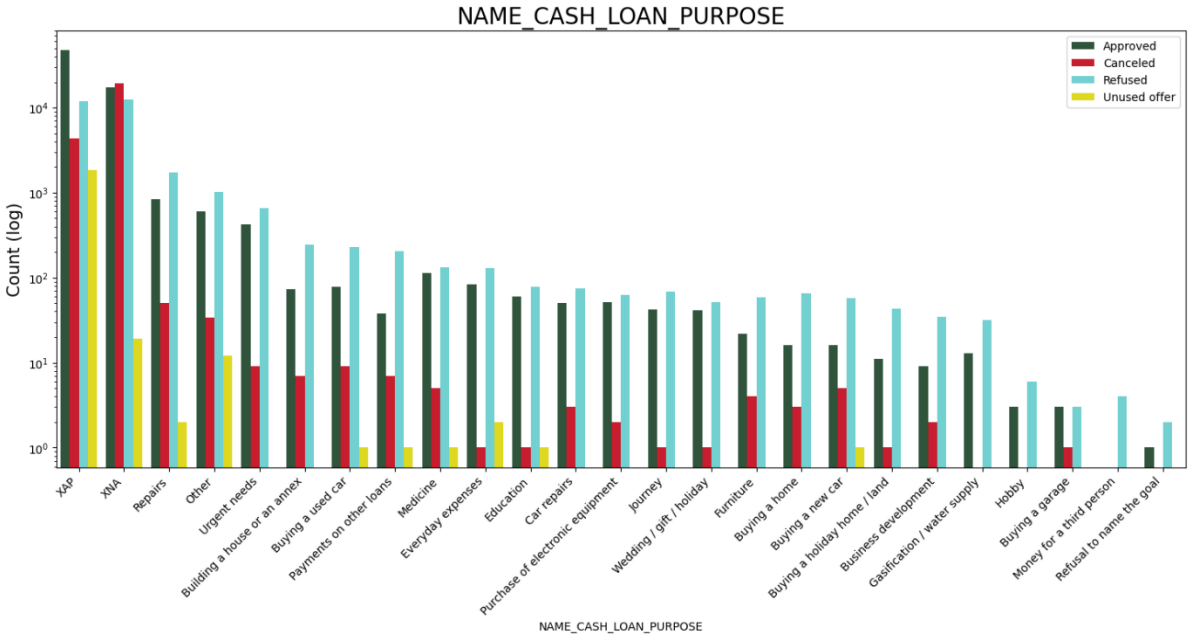
Bisecting the "loan_df" dataframe based on Target value 0 and 1 for analysis

```
# Bisecting the "loan_df" dataframe based on Target value 0 and 1 for correlation and other analysis

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

```
univariate_c_merged("NAME_CASH_LOAN_PURPOSE",L0,"NAME_CONTRACT_STATUS",["#295939","#e40017","#64dfdf","#fff600"],True,(18,7))
```




```
univariate_c_merged("NAME_CASH_LOAN_PURPOSE",L1,"NAME_CONTRACT_STATUS",["#295939","#e40017","#64dfdf","#fff600"],True,(18,7))
```



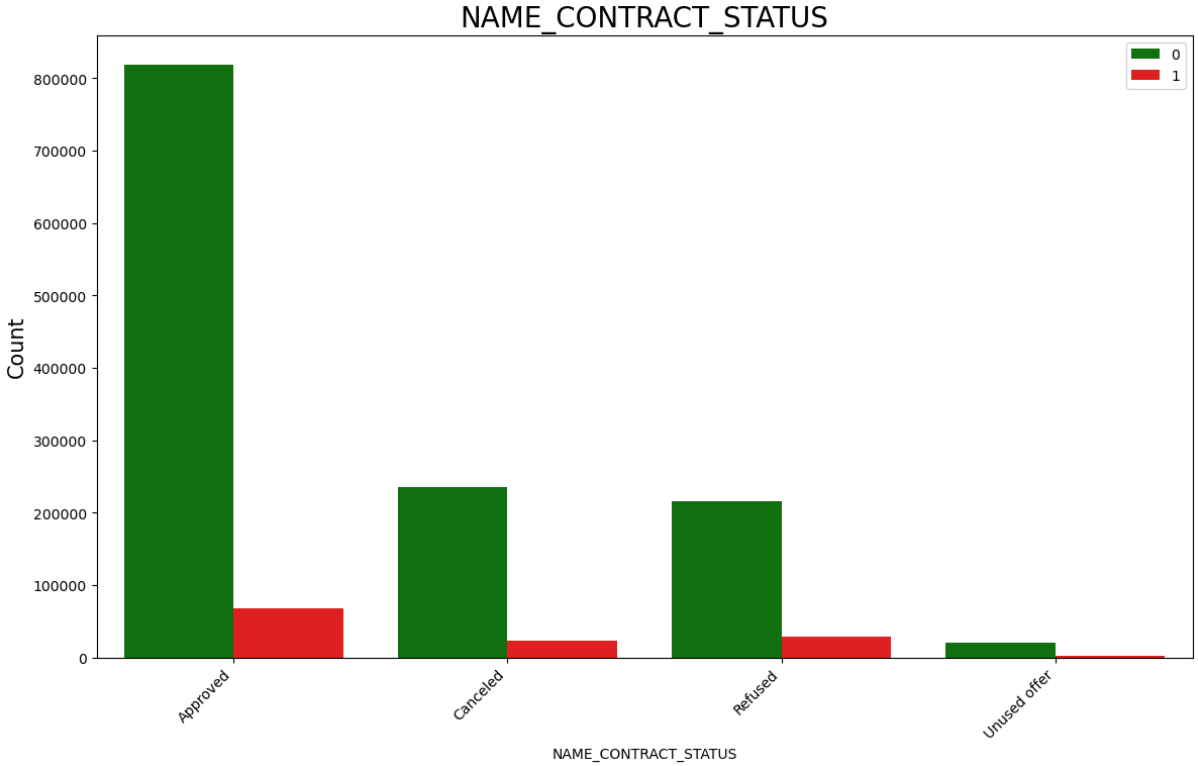
Insights:

- **Loan purpose has high number of unknown values (XAP, XNA)**
- **Loan taken for the purpose of Repairs looks to have highest default rate**
- **Huge number application has been rejected by bank or refused by client which are applied for Repair or Other. from this we can infer that repair is considered high risk by bank. Also,**

either they are rejected or bank offers loan on high interest rate which is not feasible by the clients and they refuse the loan.

Checking Contract Status based on loan repayment status whether there is any business loss or financial loss

```
# Checking Contract Status based on loan repayment status whether there is any business loss or financial loss

univariate_c_merged("NAME_CONTRACT_STATUS",loan_df,"TARGET",['g','r'],False,(14,8))
r = loan_df.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) +"%" # adding percentage symbol in the results for understanding
df1
```



		Counts	Percentage
NAME_CONTRACT_STATUS	TARGET		
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%

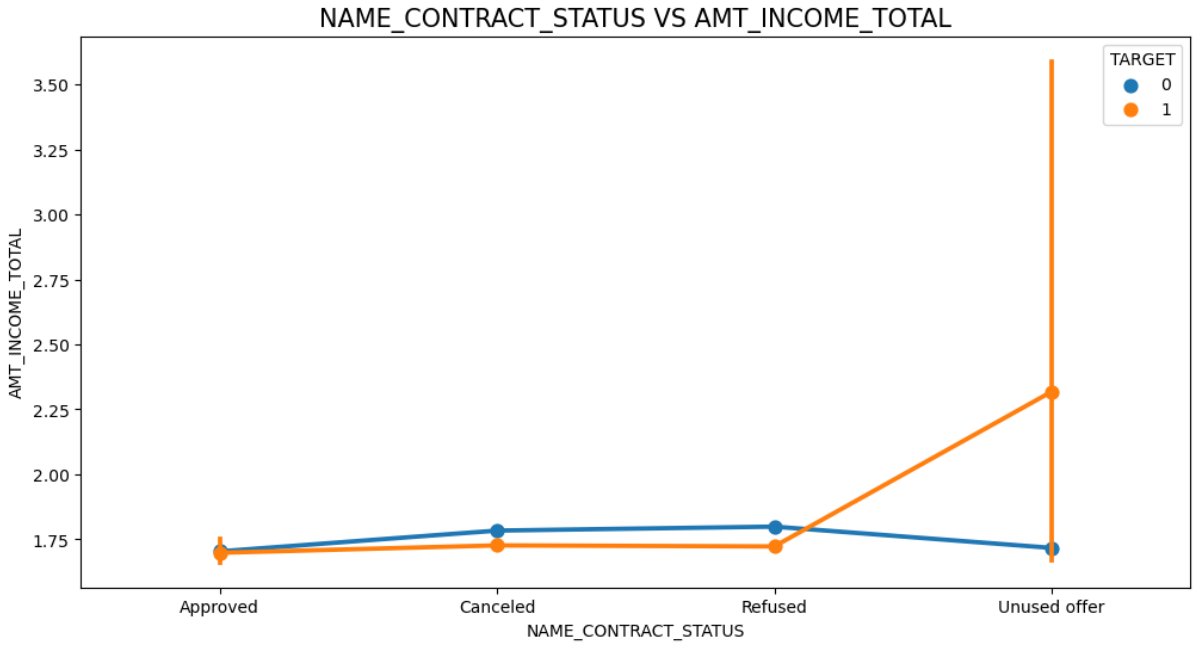
Insights:

- **90% of the previously cancelled client have actually repaid the loan. Revising the interest rates would increase business opportunity for these clients**

- **88% of the clients who have been previously refused a loan has paid back the loan in current case.**
- **Refusal reason should be recorded for further analysis as these clients could turn into potential repaying customer.**

Relationship between income total and contact status

```
# plotting the relationship between income total and contact status

pointplot(loan_df,"TARGET","NAME_CONTRACT_STATUS",'AMT_INCOME_TOTAL')
```

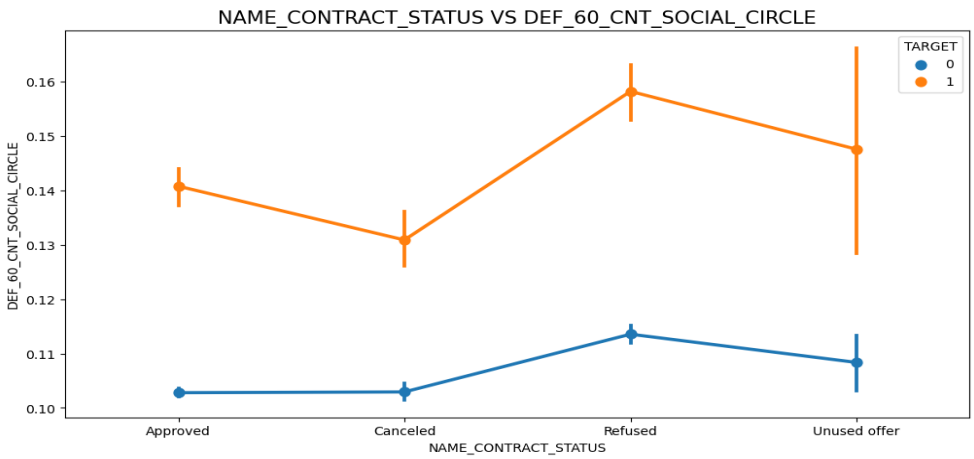


Insights:

- **90% of the previously cancelled client have actually repaid the loan. Revising the interest rates would increase business opportunity for these clients**
- **88% of the clients who have been previously refused a loan has paid back the loan in current case.**
- **Refusal reason should be recorded for further analysis as these clients could turn into potential repaying customer.**

Relationship between people who defaulted in last 60 days being in client's social circle and contact status

```
# plotting the relationship between people who defaulted in last 60 days being in client's social circle and contact status

pointplot(loan_df,"TARGET","NAME_CONTRACT_STATUS",'DEF_60_CNT_SOCIAL_CIRCLE')
```



Insights:

- **Clients who have average of 0.13 or higher their DEF_60_CNT_SOCIAL_CIRCLE score tend to default more and thus analysing client's social circle could help in disbursement of the loan.**

Conclusions

After analysing the datasets, there are few attributes of a client with which the bank would be able to identify if they will repay the loan or not. The analysis is consisted as below with the contributing factors and categorization:

A. Decisive Factor whether an applicant will be Repayer:

1. **NAME_EDUCATION_TYPE**: Academic degree has less defaults.
2. **NAME_INCOME_TYPE:** Student and Businessmen have no defaults.
3. **REGION_RATING_CLIENT:** RATING 1 is safer.
4. **ORGANIZATION_TYPE:** Clients with Trade Type 4 and 5 and Industry type 8 have defaulted less than 3%
5. **DAYS_BIRTH:** People above age of 50 have low probability of defaulting
6. **DAYS_EMPLOYED:** Clients with 40+ year experience having less than 1% default rate
7. **AMT_INCOME_TOTAL:** Applicant with Income more than 700,000 are less likely to default
8. **NAME_CASH_LOAN_PURPOSE:** Loans bought for Hobby, buying garage are being repaid mostly.
9. **CNT_CHILDREN:** People with zero to two children tend to repay the loans.

B. Decisive Factor whether an applicant will be Defaulter:

1. **CODE_GENDER:** Men are at relatively higher default rate
2. **NAME_FAMILY_STATUS :** People who have civil marriage or who are single default a lot.
3. **NAME_EDUCATION_TYPE:** People with Lower Secondary & Secondary education
4. **NAME_INCOME_TYPE:** Clients who are either at Maternity leave OR Unemployed default a lot.
5. **REGION_RATING_CLIENT:** People who live in Rating 3 has highest defaults.

6. **OCCUPATION_TYPE:** Avoid Low-skill Laborers, Drivers and Waiters/barmen staff, Security staff, Laborers and Cooking staff as their default rate is huge.
7. **ORGANIZATION_TYPE:** Organizations with highest percent of loans not repaid are Transport: type 3 (16%), Industry: type 13 (13.5%), Industry: type 8 (12.5%) and Restaurant (less than 12%). Self-employed people have relative high defaulting rate, and thus should be avoided to be approved for loan or provide loan with higher interest rate to mitigate the risk of defaulting.
8. **DAYS_BIRTH:** Avoid young people who are in age group of 20-40 as they have higher probability of defaulting
9. **DAYS_EMPLOYED:** People who have less than 5 years of employment have high default rate.
10. **CNT_CHILDREN & CNT_FAM_MEMBERS:** Client who have children equal to or more than 9 default 100% and hence their applications are to be rejected.
11. **AMT_GOODS_PRICE:** When the credit amount goes beyond 3lakhs, there is an increase in defaulters.

C. Factors that Loan can be given on Condition of High Interest rate to mitigate any default risk leading to business loss:

1. **NAME_HOUSING_TYPE:** High number of loan applications are from the category of people who live in Rented apartments & living with parents and hence offering the loan would mitigate the loss if any of those default.
2. **AMT_CREDIT:** People who get loan for 3-6 Lakhs tend to default more than others and hence having higher interest specifically for this credit range would be ideal.
3. **AMT_INCOME:** Since 90% of the applications have Income total less than 3Lakhs and they have high probability of defaulting, they could be offered loan with higher interest compared to other income category.
4. **CNT_CHILDREN & CNT_FAM_MEMBERS:** Clients who have 4 to 8 children has a very high default rate and hence higher interest should be imposed on their loans.
5. **NAME_CASH_LOAN_PURPOSE:** Loan taken for the purpose of Repairs seems to have highest default rate. A very high number applications have been rejected by bank or refused by client in previous applications as well which has purpose as repair or other. This shows that purpose repair is taken as high risk by bank and either they are rejected, or bank offers very high loan interest rate which is not feasible by the clients, thus they refuse the loan. The same approach could be followed in future as well.