In [1]:
```
# Import Python Libraries:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.style as style
import seaborn as sns
import itertools
%matplotlib inline

# setting up plot style
style.use('seaborn-poster')
style.use('fivethirtyeight')
```

In [2]:
```
#2.2 Supress Warnings:
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
#2.3 Adjust Jupyer Views:
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)
```

In [4]:
```
#3. Reading & Understanding the data

#3.1 Importing the input files
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all
input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [5]:
```
applicationDF = pd.read_csv(r"C:\Users\raju sabhavath\Desktop\FSDS &AI RESSUME PROJE
previousDF = pd.read_csv(r"C:\Users\raju sabhavath\Desktop\FSDS &AI RESSUME PROJECTS
applicationDF.head()
```

Out[5]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REAL
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100004	0	Revolving loans	M	Y	
3	100006	0	Cash loans	F	N	
4	100007	0	Cash loans	M	N	

In [6]:
```
previousDF.head()
```

Out[6]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREI
0	2030495	271877	Consumer loans	1730.430	17145.0	1714
1	2802425	108129	Cash loans	25188.615	607500.0	67967
2	2523466	122040	Cash loans	15060.735	112500.0	13644
3	2819243	176158	Cash loans	47041.335	450000.0	47079
4	1784265	202054	Cash loans	31924.395	337500.0	40405

In [7]:
```
#3.2 Inspect Data Frames
# Database dimension
print("Database dimension - applicationDF     :",applicationDF.shape)
print("Database dimension - previousDF        :",previousDF.shape)

#Database size
print("Database size - applicationDF          :",applicationDF.size)
print("Database size - previousDF             :",previousDF.size)
```

```
Database dimension - applicationDF     : (307511, 122)
Database dimension - previousDF        : (1670214, 37)
Database size - applicationDF          : 37516342
Database size - previousDF             : 61797918
```

In [8]:
```
# Database column types
applicationDF.info(verbose=True)
```

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

```
21   OWN_CAR_AGE                   float64
22   FLAG_MOBIL                    int64
23   FLAG_EMP_PHONE                int64
24   FLAG_WORK_PHONE               int64
25   FLAG_CONT_MOBILE              int64
26   FLAG_PHONE                    int64
27   FLAG_EMAIL                    int64
28   OCCUPATION_TYPE               object
29   CNT_FAM_MEMBERS               float64
30   REGION_RATING_CLIENT          int64
31   REGION_RATING_CLIENT_W_CITY   int64
32   WEEKDAY_APPR_PROCESS_START    object
33   HOUR_APPR_PROCESS_START       int64
34   REG_REGION_NOT_LIVE_REGION    int64
35   REG_REGION_NOT_WORK_REGION    int64
36   LIVE_REGION_NOT_WORK_REGION   int64
37   REG_CITY_NOT_LIVE_CITY        int64
38   REG_CITY_NOT_WORK_CITY        int64
39   LIVE_CITY_NOT_WORK_CITY       int64
40   ORGANIZATION_TYPE             object
41   EXT_SOURCE_1                  float64
42   EXT_SOURCE_2                  float64
43   EXT_SOURCE_3                  float64
44   APARTMENTS_AVG                float64
45   BASEMENTAREA_AVG              float64
46   YEARS_BEGINEXPLUATATION_AVG   float64
47   YEARS_BUILD_AVG               float64
48   COMMONAREA_AVG                float64
49   ELEVATORS_AVG                 float64
50   ENTRANCES_AVG                 float64
51   FLOORSMAX_AVG                 float64
52   FLOORSMIN_AVG                 float64
53   LANDAREA_AVG                  float64
54   LIVINGAPARTMENTS_AVG          float64
55   LIVINGAREA_AVG                float64
56   NONLIVINGAPARTMENTS_AVG       float64
57   NONLIVINGAREA_AVG             float64
58   APARTMENTS_MODE               float64
59   BASEMENTAREA_MODE             float64
60   YEARS_BEGINEXPLUATATION_MODE  float64
61   YEARS_BUILD_MODE              float64
62   COMMONAREA_MODE               float64
63   ELEVATORS_MODE                float64
64   ENTRANCES_MODE                float64
65   FLOORSMAX_MODE                float64
66   FLOORSMIN_MODE                float64
67   LANDAREA_MODE                 float64
68   LIVINGAPARTMENTS_MODE         float64
69   LIVINGAREA_MODE               float64
70   NONLIVINGAPARTMENTS_MODE      float64
71   NONLIVINGAREA_MODE            float64
72   APARTMENTS_MEDI               float64
73   BASEMENTAREA_MEDI             float64
74   YEARS_BEGINEXPLUATATION_MEDI  float64
75   YEARS_BUILD_MEDI              float64
76   COMMONAREA_MEDI               float64
77   ELEVATORS_MEDI                float64
78   ENTRANCES_MEDI                float64
79   FLOORSMAX_MEDI                float64
80   FLOORSMIN_MEDI                float64
81   LANDAREA_MEDI                 float64
82   LIVINGAPARTMENTS_MEDI         float64
83   LIVINGAREA_MEDI               float64
84   NONLIVINGAPARTMENTS_MEDI      float64
```

```
 85   NONLIVINGAREA_MEDI            float64
 86   FONDKAPREMONT_MODE            object
 87   HOUSETYPE_MODE                object
 88   TOTALAREA_MODE                float64
 89   WALLSMATERIAL_MODE            object
 90   EMERGENCYSTATE_MODE           object
 91   OBS_30_CNT_SOCIAL_CIRCLE      float64
 92   DEF_30_CNT_SOCIAL_CIRCLE      float64
 93   OBS_60_CNT_SOCIAL_CIRCLE      float64
 94   DEF_60_CNT_SOCIAL_CIRCLE      float64
 95   DAYS_LAST_PHONE_CHANGE        float64
 96   FLAG_DOCUMENT_2               int64
 97   FLAG_DOCUMENT_3               int64
 98   FLAG_DOCUMENT_4               int64
 99   FLAG_DOCUMENT_5               int64
 100  FLAG_DOCUMENT_6               int64
 101  FLAG_DOCUMENT_7               int64
 102  FLAG_DOCUMENT_8               int64
 103  FLAG_DOCUMENT_9               int64
 104  FLAG_DOCUMENT_10              int64
 105  FLAG_DOCUMENT_11              int64
 106  FLAG_DOCUMENT_12              int64
 107  FLAG_DOCUMENT_13              int64
 108  FLAG_DOCUMENT_14              int64
 109  FLAG_DOCUMENT_15              int64
 110  FLAG_DOCUMENT_16              int64
 111  FLAG_DOCUMENT_17              int64
 112  FLAG_DOCUMENT_18              int64
 113  FLAG_DOCUMENT_19              int64
 114  FLAG_DOCUMENT_20              int64
 115  FLAG_DOCUMENT_21              int64
 116  AMT_REQ_CREDIT_BUREAU_HOUR    float64
 117  AMT_REQ_CREDIT_BUREAU_DAY     float64
 118  AMT_REQ_CREDIT_BUREAU_WEEK    float64
 119  AMT_REQ_CREDIT_BUREAU_MON     float64
 120  AMT_REQ_CREDIT_BUREAU_QRT     float64
 121  AMT_REQ_CREDIT_BUREAU_YEAR    float64
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
```

In [9]:

```
previousDF.info(verbose=True)
```

```
<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
```

In [10]:

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

Out[10]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANN
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.0(
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.5
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.7
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.5(
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.0(
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.0(
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.0(
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.5(

In [11]:

```
previousDF.describe()
```

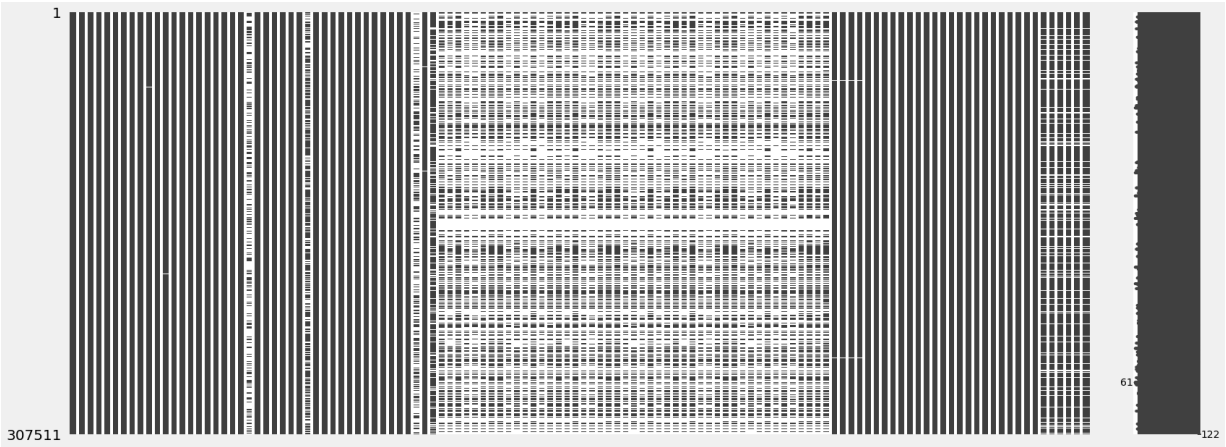
Out[11]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_I
count	1.670214e+06	1.670214e+06	1.297979e+06	1.670214e+06	1.670213e+06	7.74
mean	1.923089e+06	2.783572e+05	1.595512e+04	1.752339e+05	1.961140e+05	6.69
std	5.325980e+05	1.028148e+05	1.478214e+04	2.927798e+05	3.185746e+05	2.09
min	1.000001e+06	1.000010e+05	0.000000e+00	0.000000e+00	0.000000e+00	-9.0(
25%	1.461857e+06	1.893290e+05	6.321780e+03	1.872000e+04	2.416050e+04	0.00
50%	1.923110e+06	2.787145e+05	1.125000e+04	7.104600e+04	8.054100e+04	1.63
75%	2.384280e+06	3.675140e+05	2.065842e+04	1.803600e+05	2.164185e+05	7.74

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_I
max	2.845382e+06	4.562550e+05	4.180581e+05	6.905160e+06	6.905160e+06	3.06

In [16]:
```
 #4.1 Data Cleaning & Manipulation

#4.1 Null Value Calculation
#4.1.1 applicationDF Missing values
import missingno as mn
mn.matrix(applicationDF)

#Insight:
#Based on the above Matrix, it is evidednt that the dataset has many missing values.
each column what is the % of missing values
```

Out[16]:
```
<AxesSubplot:>
```



In [18]:
```
# % null value in each column
round(applicationDF.isnull().sum() / applicationDF.shape[0] * 100.00,2)


#Insight:
#There are many columns in applicationDF dataframe where missing value is more than
Let's plot the columns vs missing value % with 40% being the cut-off marks
```

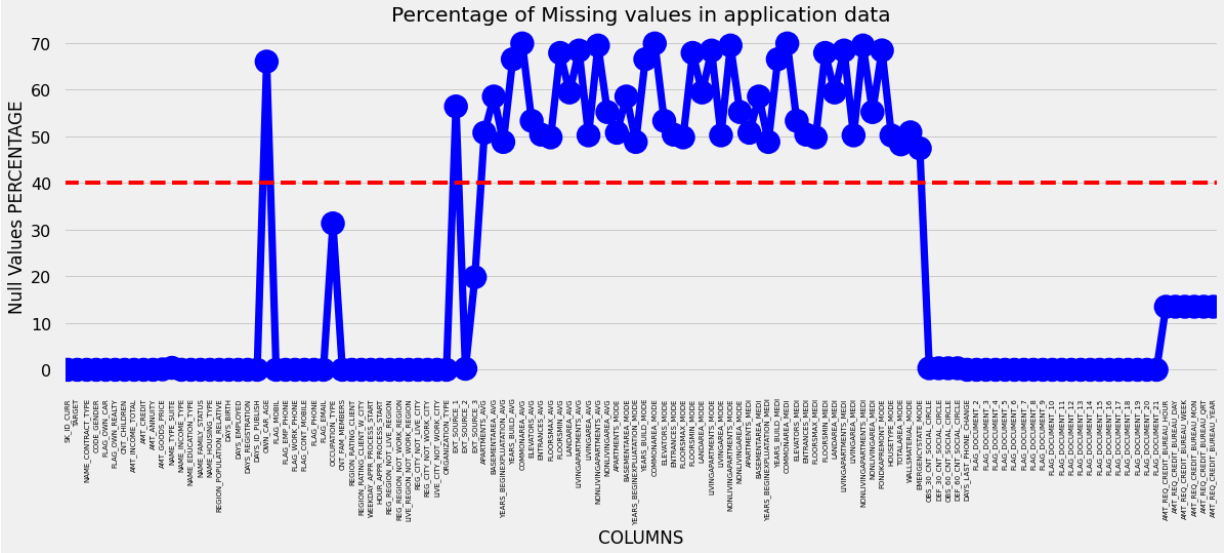
Out[18]:
```
SK_ID_CURR                     0.00
TARGET                         0.00
NAME_CONTRACT_TYPE             0.00
CODE_GENDER                    0.00
FLAG_OWN_CAR                   0.00
FLAG_OWN_REALTY                0.00
CNT_CHILDREN                   0.00
AMT_INCOME_TOTAL               0.00
AMT_CREDIT                     0.00
AMT_ANNUITY                    0.00
AMT_GOODS_PRICE                0.09
NAME_TYPE_SUITE                0.42
NAME_INCOME_TYPE               0.00
NAME_EDUCATION_TYPE            0.00
NAME_FAMILY_STATUS             0.00
NAME_HOUSING_TYPE              0.00
REGION_POPULATION_RELATIVE     0.00
DAYS_BIRTH                     0.00
DAYS_EMPLOYED                  0.00
DAYS_REGISTRATION              0.00
```

DAYS_ID_PUBLISH	0.00
OWN_CAR_AGE	65.99
FLAG_MOBIL	0.00
FLAG_EMP_PHONE	0.00
FLAG_WORK_PHONE	0.00
FLAG_CONT_MOBILE	0.00
FLAG_PHONE	0.00
FLAG_EMAIL	0.00
OCCUPATION_TYPE	31.35
CNT_FAM_MEMBERS	0.00
REGION_RATING_CLIENT	0.00
REGION_RATING_CLIENT_W_CITY	0.00
WEEKDAY_APPR_PROCESS_START	0.00
HOUR_APPR_PROCESS_START	0.00
REG_REGION_NOT_LIVE_REGION	0.00
REG_REGION_NOT_WORK_REGION	0.00
LIVE_REGION_NOT_WORK_REGION	0.00
REG_CITY_NOT_LIVE_CITY	0.00
REG_CITY_NOT_WORK_CITY	0.00
LIVE_CITY_NOT_WORK_CITY	0.00
ORGANIZATION_TYPE	0.00
EXT_SOURCE_1	56.38
EXT_SOURCE_2	0.21
EXT_SOURCE_3	19.83
APARTMENTS_AVG	50.75
BASEMENTAREA_AVG	58.52
YEARS_BEGINEXPLUATATION_AVG	48.78
YEARS_BUILD_AVG	66.50
COMMONAREA_AVG	69.87
ELEVATORS_AVG	53.30
ENTRANCES_AVG	50.35
FLOORSMAX_AVG	49.76
FLOORSMIN_AVG	67.85
LANDAREA_AVG	59.38
LIVINGAPARTMENTS_AVG	68.35
LIVINGAREA_AVG	50.19
NONLIVINGAPARTMENTS_AVG	69.43
NONLIVINGAREA_AVG	55.18
APARTMENTS_MODE	50.75
BASEMENTAREA_MODE	58.52
YEARS_BEGINEXPLUATATION_MODE	48.78
YEARS_BUILD_MODE	66.50
COMMONAREA_MODE	69.87
ELEVATORS_MODE	53.30
ENTRANCES_MODE	50.35
FLOORSMAX_MODE	49.76
FLOORSMIN_MODE	67.85
LANDAREA_MODE	59.38
LIVINGAPARTMENTS_MODE	68.35
LIVINGAREA_MODE	50.19
NONLIVINGAPARTMENTS_MODE	69.43
NONLIVINGAREA_MODE	55.18
APARTMENTS_MEDI	50.75
BASEMENTAREA_MEDI	58.52
YEARS_BEGINEXPLUATATION_MEDI	48.78
YEARS_BUILD_MEDI	66.50
COMMONAREA_MEDI	69.87
ELEVATORS_MEDI	53.30
ENTRANCES_MEDI	50.35
FLOORSMAX_MEDI	49.76
FLOORSMIN_MEDI	67.85
LANDAREA_MEDI	59.38
LIVINGAPARTMENTS_MEDI	68.35
LIVINGAREA_MEDI	50.19

```
NONLIVINGAPARTMENTS_MEDI        69.43
NONLIVINGAREA_MEDI              55.18
FONDKAPREMONT_MODE              68.39
HOUSETYPE_MODE                  50.18
TOTALAREA_MODE                  48.27
WALLSMATERIAL_MODE              50.84
EMERGENCYSTATE_MODE             47.40
OBS_30_CNT_SOCIAL_CIRCLE         0.33
DEF_30_CNT_SOCIAL_CIRCLE         0.33
OBS_60_CNT_SOCIAL_CIRCLE         0.33
DEF_60_CNT_SOCIAL_CIRCLE         0.33
DAYS_LAST_PHONE_CHANGE           0.00
FLAG_DOCUMENT_2                  0.00
FLAG_DOCUMENT_3                  0.00
FLAG_DOCUMENT_4                  0.00
FLAG_DOCUMENT_5                  0.00
FLAG_DOCUMENT_6                  0.00
FLAG_DOCUMENT_7                  0.00
FLAG_DOCUMENT_8                  0.00
FLAG_DOCUMENT_9                  0.00
FLAG_DOCUMENT_10                 0.00
FLAG_DOCUMENT_11                 0.00
FLAG_DOCUMENT_12                 0.00
FLAG_DOCUMENT_13                 0.00
FLAG_DOCUMENT_14                 0.00
FLAG_DOCUMENT_15                 0.00
FLAG_DOCUMENT_16                 0.00
FLAG_DOCUMENT_17                 0.00
FLAG_DOCUMENT_18                 0.00
FLAG_DOCUMENT_19                 0.00
FLAG_DOCUMENT_20                 0.00
FLAG_DOCUMENT_21                 0.00
AMT_REQ_CREDIT_BUREAU_HOUR      13.50
AMT_REQ_CREDIT_BUREAU_DAY       13.50
AMT_REQ_CREDIT_BUREAU_WEEK      13.50
AMT_REQ_CREDIT_BUREAU_MON       13.50
AMT_REQ_CREDIT_BUREAU_QRT       13.50
AMT_REQ_CREDIT_BUREAU_YEAR      13.50
dtype: float64
```

In [19]:

```
null_applicationDF = pd.DataFrame((applicationDF.isnull().sum())*100/applicationDF.s
null_applicationDF.columns = ['Column Name', 'Null Values Percentage']
fig = plt.figure(figsize=(18,6))
ax = sns.pointplot(x="Column Name",y="Null Values Percentage",data=null_applicationD
plt.xticks(rotation =90,fontsize =7)
ax.axhline(40, ls='--',color='red')
plt.title("Percentage of Missing values in application data")
plt.ylabel("Null Values PERCENTAGE")
plt.xlabel("COLUMNS")
plt.show()

#Insight:
From the plot we can see the columns in which percentage of null values more than 40
above the red line and the columns which have less than 40 % null values below the r
Let's check the columns which has more than 40% missing values
```



In [20]:

```
# more than or equal to 40% empty rows columns
nullcol_40_application = null_applicationDF[null_applicationDF["Null Values Percenta
nullcol_40_application
```

Out[20]:

	Column Name	Null Values Percentage
21	OWN_CAR_AGE	65.990810
41	EXT_SOURCE_1	56.381073
44	APARTMENTS_AVG	50.749729
45	BASEMENTAREA_AVG	58.515956
46	YEARS_BEGINEXPLUATATION_AVG	48.781019
47	YEARS_BUILD_AVG	66.497784
48	COMMONAREA_AVG	69.872297
49	ELEVATORS_AVG	53.295980
50	ENTRANCES_AVG	50.348768
51	FLOORSMAX_AVG	49.760822
52	FLOORSMIN_AVG	67.848630
53	LANDAREA_AVG	59.376738
54	LIVINGAPARTMENTS_AVG	68.354953
55	LIVINGAREA_AVG	50.193326
56	NONLIVINGAPARTMENTS_AVG	69.432963
57	NONLIVINGAREA_AVG	55.179164
58	APARTMENTS_MODE	50.749729
59	BASEMENTAREA_MODE	58.515956
60	YEARS_BEGINEXPLUATATION_MODE	48.781019
61	YEARS_BUILD_MODE	66.497784
62	COMMONAREA_MODE	69.872297
63	ELEVATORS_MODE	53.295980
64	ENTRANCES_MODE	50.348768

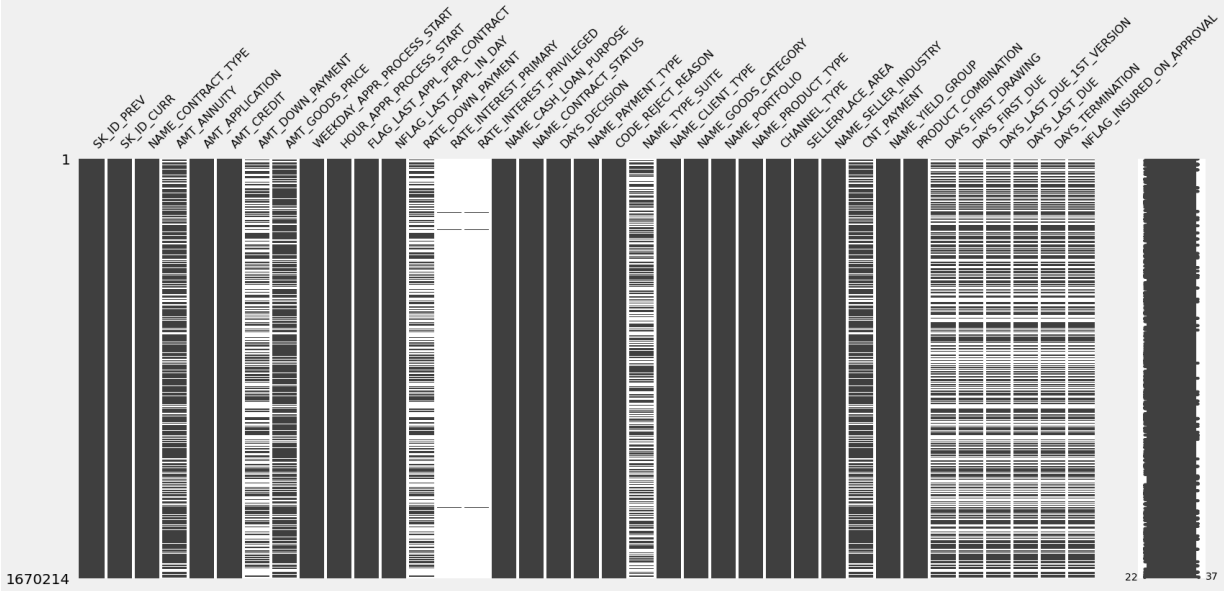
	Column Name	Null Values Percentage
65	FLOORSMAX_MODE	49.760822
66	FLOORSMIN_MODE	67.848630
67	LANDAREA_MODE	59.376738
68	LIVINGAPARTMENTS_MODE	68.354953
69	LIVINGAREA_MODE	50.193326
70	NONLIVINGAPARTMENTS_MODE	69.432963
71	NONLIVINGAREA_MODE	55.179164
72	APARTMENTS_MEDI	50.749729
73	BASEMENTAREA_MEDI	58.515956
74	YEARS_BEGINEXPLUATATION_MEDI	48.781019
75	YEARS_BUILD_MEDI	66.497784
76	COMMONAREA_MEDI	69.872297
77	ELEVATORS_MEDI	53.295980
78	ENTRANCES_MEDI	50.348768
79	FLOORSMAX_MEDI	49.760822
80	FLOORSMIN_MEDI	67.848630
81	LANDAREA_MEDI	59.376738
82	LIVINGAPARTMENTS_MEDI	68.354953
83	LIVINGAREA_MEDI	50.193326
84	NONLIVINGAPARTMENTS_MEDI	69.432963
85	NONLIVINGAREA_MEDI	55.179164
86	FONDKAPREMONT_MODE	68.386172
87	HOUSETYPE_MODE	50.176091
88	TOTALAREA_MODE	48.268517
89	WALLSMATERIAL_MODE	50.840783
90	EMERGENCYSTATE_MODE	47.398304

In [21]:
```
# How many columns have more than or euqal to 40% null values ?
len(nullcol_40_application)

#Insight:
Total of 49 columns are there which have more than 40% null values.Seems like most o
high missing values arerelated to different area sizes on apartment owned/rented by
```

Out[21]:
```
49
```

In [22]:
```
# 4.1.2 previousDF Missing Values
mn.matrix(previousDF)
```

Out[22]:
```
<AxesSubplot:>
```



In [23]:

```
# checking the null value % of each column in previousDF dataframe
round(previousDF.isnull().sum() / previousDF.shape[0] * 100.00,2)

#Insight:
There are many columns in previousDF dataframe where missing value is more than 40%.
Let's plot the columns vs missing value % with 40% being the cut-off marks
```

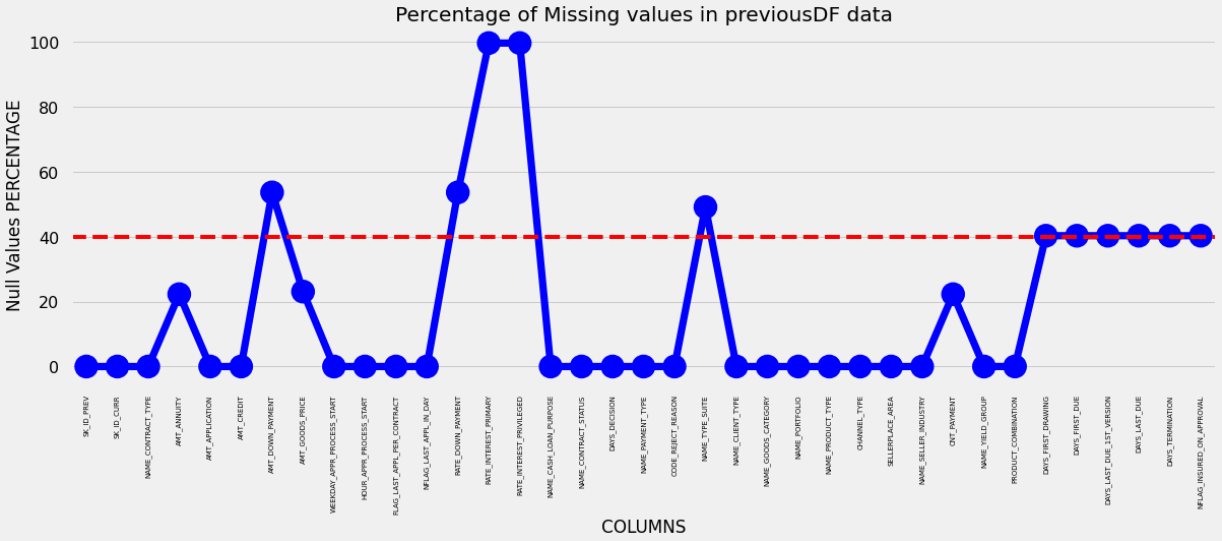
Out[23]:

SK_ID_PREV	0.00
SK_ID_CURR	0.00
NAME_CONTRACT_TYPE	0.00
AMT_ANNUITY	22.29
AMT_APPLICATION	0.00
AMT_CREDIT	0.00
AMT_DOWN_PAYMENT	53.64
AMT_GOODS_PRICE	23.08
WEEKDAY_APPR_PROCESS_START	0.00
HOUR_APPR_PROCESS_START	0.00
FLAG_LAST_APPL_PER_CONTRACT	0.00
NFLAG_LAST_APPL_IN_DAY	0.00
RATE_DOWN_PAYMENT	53.64
RATE_INTEREST_PRIMARY	99.64
RATE_INTEREST_PRIVILEGED	99.64
NAME_CASH_LOAN_PURPOSE	0.00
NAME_CONTRACT_STATUS	0.00
DAYS_DECISION	0.00
NAME_PAYMENT_TYPE	0.00
CODE_REJECT_REASON	0.00
NAME_TYPE_SUITE	49.12
NAME_CLIENT_TYPE	0.00
NAME_GOODS_CATEGORY	0.00
NAME_PORTFOLIO	0.00
NAME_PRODUCT_TYPE	0.00
CHANNEL_TYPE	0.00
SELLERPLACE_AREA	0.00
NAME_SELLER_INDUSTRY	0.00
CNT_PAYMENT	22.29
NAME_YIELD_GROUP	0.00
PRODUCT_COMBINATION	0.02
DAYS_FIRST_DRAWING	40.30
DAYS_FIRST_DUE	40.30
DAYS_LAST_DUE_1ST_VERSION	40.30
DAYS_LAST_DUE	40.30
DAYS_TERMINATION	40.30

```
NFLAG_INSURED_ON_APPROVAL       40.30
dtype: float64
```

In [24]:

```
null_previousDF = pd.DataFrame((previousDF.isnull().sum())*100/previousDF.shape[0]).
null_previousDF.columns = ['Column Name', 'Null Values Percentage']
fig = plt.figure(figsize=(18,6))
ax = sns.pointplot(x="Column Name",y="Null Values Percentage",data=null_previousDF,c
plt.xticks(rotation =90,fontsize =7)
ax.axhline(40, ls='--',color='red')
plt.title("Percentage of Missing values in previousDF data")
plt.ylabel("Null Values PERCENTAGE")
plt.xlabel("COLUMNS")
plt.show()


#Insight:
From the plot we can see the columns in which percentage of null values more than 40
the red line and the columns which have less than 40 % null values below the red lin
Let's check the columns which has more than 40% missing values
```



In [25]:

```
# more than or equal to 40% empty rows columns
nullcol_40_previous = null_previousDF[null_previousDF["Null Values Percentage"]>=40]
nullcol_40_previous
```

Out[25]:

	Column Name	Null Values Percentage
6	AMT_DOWN_PAYMENT	53.636480
12	RATE_DOWN_PAYMENT	53.636480
13	RATE_INTEREST_PRIMARY	99.643698
14	RATE_INTEREST_PRIVILEGED	99.643698
20	NAME_TYPE_SUITE	49.119754
31	DAYS_FIRST_DRAWING	40.298129
32	DAYS_FIRST_DUE	40.298129
33	DAYS_LAST_DUE_1ST_VERSION	40.298129
34	DAYS_LAST_DUE	40.298129
35	DAYS_TERMINATION	40.298129
36	NFLAG_INSURED_ON_APPROVAL	40.298129

In [26]:
```
# How many columns have more than or euqal to 40% null values ?
len(nullcol_40_previous)

Insight:
Total of 11 columns are there which have more than 40% null values. These columns ca
Before deleting these columns, let's review if there are more columns which can be d
not[](http://)
```

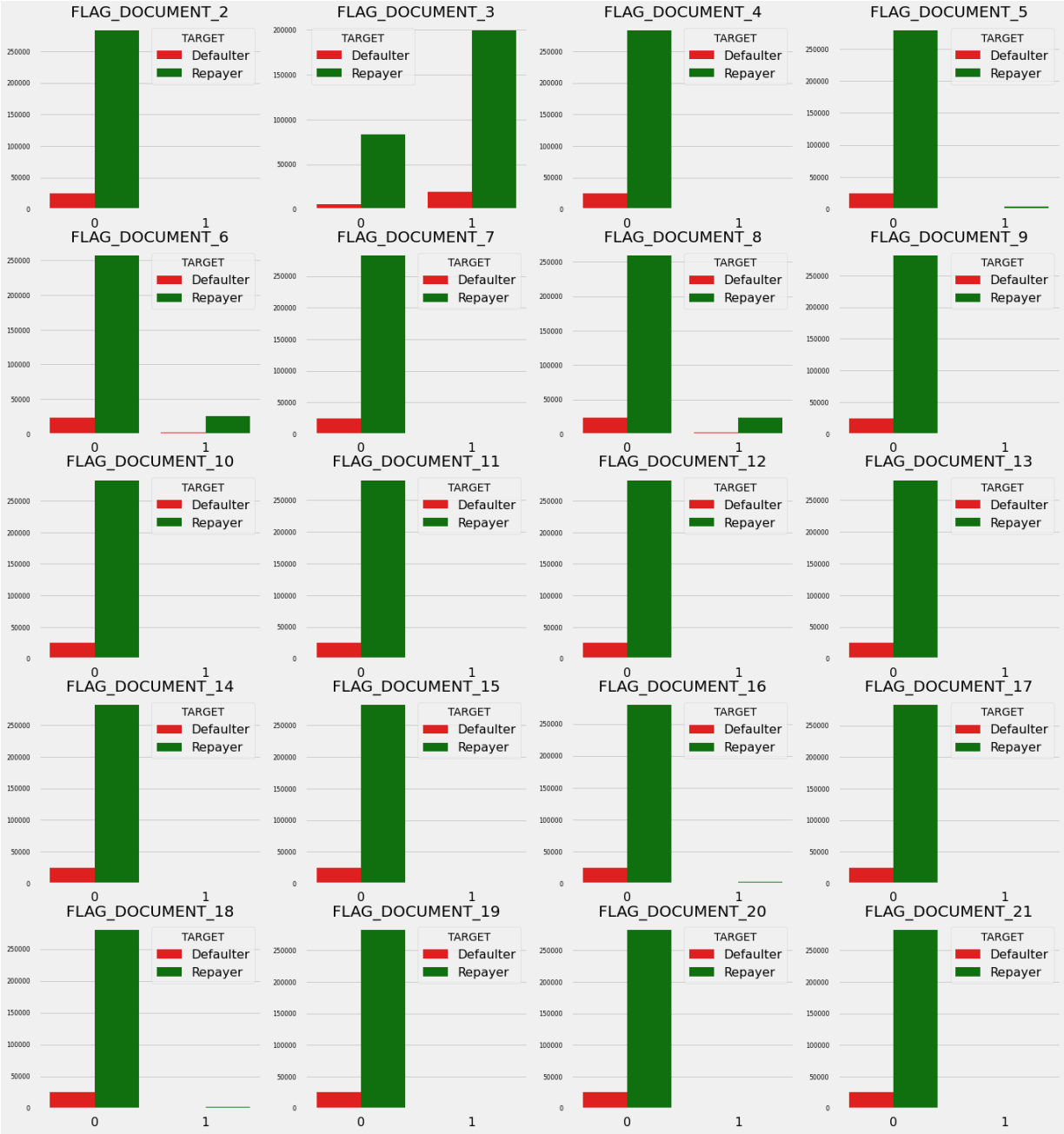
Out[26]:
```
11
```

In [27]:
```
# 4.2 Analyze & Delete Unnecessary Columns in applicationDF
# 4.2.1 EXT_SOURCE_X
# Checking correlation of EXT_SOURCE_X columns vs TARGET column
Source = applicationDF[["EXT_SOURCE_1","EXT_SOURCE_2","EXT_SOURCE_3","TARGET"]]
source_corr = Source.corr()
ax = sns.heatmap(source_corr,
            xticklabels=source_corr.columns,
            yticklabels=source_corr.columns,
            annot = True,
            cmap ="RdYlGn")


#Insight:
Based on the above Heatmap, we can see there is almost no correlation between EXT_SO
target column, thus we can drop these columns. EXT_SOURCE_1 has 56% null values,
where as EXT_SOURCE_3 has close to 20% null values
```

In [28]:
```
# create a list of columns that needs to be dropped including the columns with >40%
Unwanted_application = nullcol_40_application["Column Name"].tolist()+ ['EXT_SOURCE_
# as EXT_SOURCE_1 column is already included in nullcol_40_application
len(Unwanted_application)
```

Out[28]: 51

In [29]:
```
# 4.2.2 Flag Document
# Checking the relevance of Flag_Document and whether it has any relation with loan
col_Doc = [ 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3','FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5
            'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9','FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_1
            'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15','FLAG_DOCUMENT_16', 'FLAG_DOCUMENT
            'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21']
df_flag = applicationDF[col_Doc+["TARGET"]]

length = len(col_Doc)

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

fig = plt.figure(figsize=(21,24))

for i,j in itertools.zip_longest(col_Doc,range(length)):
    plt.subplot(5,4,j+1)
    ax = sns.countplot(df_flag[i],hue=df_flag["TARGET"],palette=["r","g"])
    plt.yticks(fontsize=8)
    plt.xlabel("")
    plt.ylabel("")
    plt.title(i)


#Insight:
The above graph shows that in most of the loan application cases, clients who applie
not submitted FLAG_DOCUMENT_X except FLAG_DOCUMENT_3. Thus, Except for FLAG_DOCUMENT
rest of the columns. Data shows if borrower has submitted FLAG_DOCUMENT_3 then there
of defaulting the loan.
```



In [30]:

```
# Including the flag documents for dropping the Document columns
col_Doc.remove('FLAG_DOCUMENT_3')
Unwanted_application = Unwanted_application + col_Doc
len(Unwanted_application)
```

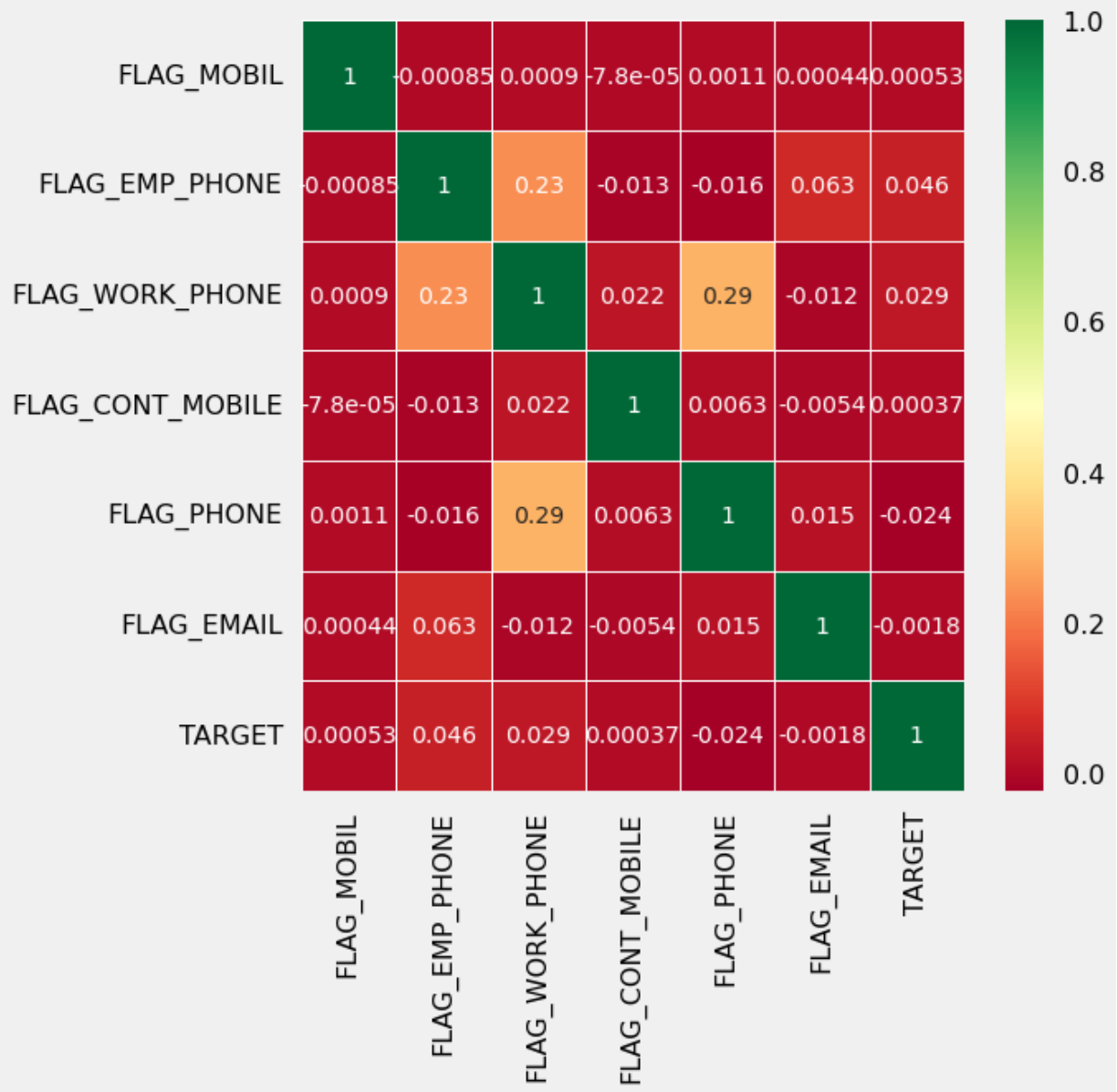
Out[30]:

```
70
```

In [31]:

```
#4.2.3 Contact Parameters
# checking is there is any correlation between mobile phone, work phone etc, email,
contact_col = ['FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE'
       'FLAG_PHONE', 'FLAG_EMAIL','TARGET']
Contact_corr = applicationDF[contact_col].corr()
fig = plt.figure(figsize=(8,8))
ax = sns.heatmap(Contact_corr,
            xticklabels=Contact_corr.columns,
            yticklabels=Contact_corr.columns,
            annot = True,
            cmap ="RdYlGn",
            linewidth=1)

#Insight:
```

There is no correlation between flags of mobile phone, email etc with loan repayment thus these columns can be deleted



In [32]:
```
# including the 6 FLAG columns to be deleted
contact_col.remove('TARGET')
Unwanted_application = Unwanted_application + contact_col
len(Unwanted_application)

#Insight:
Total 76 columns can be deleted from applicationDF
```

Out[32]: 76

In [33]:
```
# Dropping the unnecessary columns from applicationDF
applicationDF.drop(labels=Unwanted_application,axis=1,inplace=True)
```

In [34]:
```
# Inspecting the dataframe after removal of unnecessary columns
applicationDF.shape
```

Out[34]: (307511, 46)

In [35]:

```
# inspecting the column types after removal of unnecessary columns
applicationDF.info()

#Insight:
After deleting unnecessary columns, there are 46 columns remaining in applicationDF
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 46 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_CAR                 307511 non-null  object 
 5   FLAG_OWN_REALTY              307511 non-null  object 
 6   CNT_CHILDREN                 307511 non-null  int64  
 7   AMT_INCOME_TOTAL             307511 non-null  float64
 8   AMT_CREDIT                   307511 non-null  float64
 9   AMT_ANNUITY                  307499 non-null  float64
 10  AMT_GOODS_PRICE              307233 non-null  float64
 11  NAME_TYPE_SUITE              306219 non-null  object 
 12  NAME_INCOME_TYPE             307511 non-null  object 
 13  NAME_EDUCATION_TYPE          307511 non-null  object 
 14  NAME_FAMILY_STATUS           307511 non-null  object 
 15  NAME_HOUSING_TYPE            307511 non-null  object 
 16  REGION_POPULATION_RELATIVE   307511 non-null  float64
 17  DAYS_BIRTH                   307511 non-null  int64  
 18  DAYS_EMPLOYED                307511 non-null  int64  
 19  DAYS_REGISTRATION            307511 non-null  float64
 20  DAYS_ID_PUBLISH              307511 non-null  int64  
 21  OCCUPATION_TYPE              211120 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   265992 non-null  float64
 41  AMT_REQ_CREDIT_BUREAU_DAY    265992 non-null  float64
 42  AMT_REQ_CREDIT_BUREAU_WEEK   265992 non-null  float64
 43  AMT_REQ_CREDIT_BUREAU_MON    265992 non-null  float64
 44  AMT_REQ_CREDIT_BUREAU_QRT    265992 non-null  float64
 45  AMT_REQ_CREDIT_BUREAU_YEAR   265992 non-null  float64
dtypes: float64(18), int64(16), object(12)
memory usage: 107.9+ MB
```

In [36]:

```
# 4.3 Analyze & Delete Unnecessary Columns in previousDF
# Getting the 11 columns which has more than 40% unknown
```

```
Unwanted_previous = nullcol_40_previous["Column Name"].tolist()
Unwanted_previous
```

Out[36]:
```
['AMT_DOWN_PAYMENT',
 'RATE_DOWN_PAYMENT',
 'RATE_INTEREST_PRIMARY',
 'RATE_INTEREST_PRIVILEGED',
 'NAME_TYPE_SUITE',
 'DAYS_FIRST_DRAWING',
 'DAYS_FIRST_DUE',
 'DAYS_LAST_DUE_1ST_VERSION',
 'DAYS_LAST_DUE',
 'DAYS_TERMINATION',
 'NFLAG_INSURED_ON_APPROVAL']
```

In [37]:
```
# Listing down columns which are not needed
Unnecessary_previous = ['WEEKDAY_APPR_PROCESS_START','HOUR_APPR_PROCESS_START',
                        'FLAG_LAST_APPL_PER_CONTRACT','NFLAG_LAST_APPL_IN_DAY']
```

In [38]:
```
Unwanted_previous = Unwanted_previous + Unnecessary_previous
len(Unwanted_previous)

#Insight:
Total 15 columns can be deleted from previousDF
```

Out[38]:
```
15
```

In [39]:
```
# Dropping the unnecessary columns from previous
previousDF.drop(labels=Unwanted_previous,axis=1,inplace=True)
# Inspecting the dataframe after removal of unnecessary columns
previousDF.shape
```

Out[39]:
```
(1670214, 22)
```

In [40]:
```
# inspecting the column types after after removal of unnecessary columns
previousDF.info()

#Insight:
After deleting unnecessary columns, there are 22 columns remaining in applicationDF
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 22 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_GOODS_PRICE         1284699 non-null  float64
 7   NAME_CASH_LOAN_PURPOSE  1670214 non-null  object
 8   NAME_CONTRACT_STATUS    1670214 non-null  object
 9   DAYS_DECISION           1670214 non-null  int64
 10  NAME_PAYMENT_TYPE       1670214 non-null  object
 11  CODE_REJECT_REASON      1670214 non-null  object
 12  NAME_CLIENT_TYPE        1670214 non-null  object
 13  NAME_GOODS_CATEGORY     1670214 non-null  object
```

```
 14  NAME_PORTFOLIO          1670214 non-null  object
 15  NAME_PRODUCT_TYPE       1670214 non-null  object
 16  CHANNEL_TYPE            1670214 non-null  object
 17  SELLERPLACE_AREA        1670214 non-null  int64
 18  NAME_SELLER_INDUSTRY    1670214 non-null  object
 19  CNT_PAYMENT             1297984 non-null  float64
 20  NAME_YIELD_GROUP        1670214 non-null  object
 21  PRODUCT_COMBINATION     1669868 non-null  object
dtypes: float64(5), int64(4), object(13)
memory usage: 280.3+ MB
```

In [43]:

```
4.4 Standardize Values
Strategy for applicationDF:
Convert DAYS_DECISION,DAYS_EMPLOYED, DAYS_REGISTRATION,DAYS_ID_PUBLISH from negative
Convert DAYS_BIRTH from negative to positive values and calculate age and create cat
Categorize the amount variables into bins
Convert region rating column and few other columns to categorical
# Converting Negative days to positive days

date_col = ['DAYS_BIRTH','DAYS_EMPLOYED','DAYS_REGISTRATION','DAYS_ID_PUBLISH']

for col in date_col:
    applicationDF[col] = abs(applicationDF[col])
```

In [44]:

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

# Creating bins for income amount
applicationDF['AMT_INCOME_TOTAL']=applicationDF['AMT_INCOME_TOTAL']/100000

bins = [0,1,2,3,4,5,6,7,8,9,10,11]
slot = ['0-100K','100K-200K', '200k-300k','300k-400k','400k-500k','500k-600k','600k-

applicationDF['AMT_INCOME_RANGE']=pd.cut(applicationDF['AMT_INCOME_TOTAL'],bins,labe
```

In [45]:

```
applicationDF['AMT_INCOME_RANGE'].value_counts(normalize=True)*100

#Insight:
More than 50% loan applicants have income amount in the range of 100K-200K. Almost 9
have income less than 300K
```

Out[45]:

```
100K-200K    50.735000
200k-300k    21.210691
0-100K       20.729695
300k-400k     4.776116
400k-500k     1.744669
500k-600k     0.356354
600k-700k     0.282805
800k-900k     0.096980
700k-800k     0.052721
900k-1M       0.009112
1M Above      0.005858
Name: AMT_INCOME_RANGE, dtype: float64
```

In [46]:

```
# Creating bins for Credit amount
applicationDF['AMT_CREDIT']=applicationDF['AMT_CREDIT']/100000

bins = [0,1,2,3,4,5,6,7,8,9,10,100]
slots = ['0-100K','100K-200K', '200k-300k','300k-400k','400k-500k','500k-600k','600k
       '800k-900k','900k-1M', '1M Above']
```

```
applicationDF['AMT_CREDIT_RANGE']=pd.cut(applicationDF['AMT_CREDIT'],bins=bins,label
```

In [47]:
```
#checking the binning of data and % of data in each category
applicationDF['AMT_CREDIT_RANGE'].value_counts(normalize=True)*100

#Insight:
More Than 16% loan applicants have taken loan which amounts to more than 1M.
```

Out[47]:
```
200k-300k    17.824728
1M Above     16.254703
500k-600k    11.131960
400k-500k    10.418489
100K-200K     9.801275
300k-400k     8.564897
600k-700k     7.820533
800k-900k     7.086576
700k-800k     6.241403
900k-1M       2.902986
0-100K        1.952450
Name: AMT_CREDIT_RANGE, dtype: float64
```

In [48]:
```
# Creating bins for Age
applicationDF['AGE'] = applicationDF['DAYS_BIRTH'] // 365
bins = [0,20,30,40,50,100]
slots = ['0-20','20-30','30-40','40-50','50 above']

applicationDF['AGE_GROUP']=pd.cut(applicationDF['AGE'],bins=bins,labels=slots)
```

In [49]:
```
#checking the binning of data and % of data in each category
applicationDF['AGE_GROUP'].value_counts(normalize=True)*100

#Insight:
31% loan applicants have age above 50 years. More than 55% of loan applicants have a
```

Out[49]:
```
50 above    31.604398
30-40       27.028952
40-50       24.194582
20-30       17.171743
0-20         0.000325
Name: AGE_GROUP, dtype: float64
```

In [50]:
```
# Creating bins for Employement Time
applicationDF['YEARS_EMPLOYED'] = applicationDF['DAYS_EMPLOYED'] // 365
bins = [0,5,10,20,30,40,50,60,150]
slots = ['0-5','5-10','10-20','20-30','30-40','40-50','50-60','60 above']

applicationDF['EMPLOYMENT_YEAR']=pd.cut(applicationDF['YEARS_EMPLOYED'],bins=bins,la
```

In [51]:
```
#checking the binning of data and % of data in each category
applicationDF['EMPLOYMENT_YEAR'].value_counts(normalize=True)*100

Insight:
More than 55% of the loan applicants have work experience within 0-5 years and almos
have less than 10 years of work experience
```

Out[51]:
```
0-5         55.582363
5-10        24.966441
10-20       14.564315
```

```
20-30        3.750117
30-40        1.058720
40-50        0.078044
50-60        0.000000
60 above     0.000000
Name: EMPLOYMENT_YEAR, dtype: float64
```

In [52]:

```
#Checking the number of unique values each column possess to identify categorical co
applicationDF.nunique().sort_values()
```

Out[52]:

```
LIVE_CITY_NOT_WORK_CITY             2
TARGET                              2
NAME_CONTRACT_TYPE                  2
REG_REGION_NOT_LIVE_REGION          2
FLAG_OWN_CAR                        2
FLAG_OWN_REALTY                     2
REG_REGION_NOT_WORK_REGION          2
LIVE_REGION_NOT_WORK_REGION         2
FLAG_DOCUMENT_3                     2
REG_CITY_NOT_LIVE_CITY              2
REG_CITY_NOT_WORK_CITY              2
REGION_RATING_CLIENT                3
CODE_GENDER                         3
REGION_RATING_CLIENT_W_CITY         3
AMT_REQ_CREDIT_BUREAU_HOUR          5
NAME_EDUCATION_TYPE                 5
AGE_GROUP                           5
NAME_FAMILY_STATUS                  6
NAME_HOUSING_TYPE                   6
EMPLOYMENT_YEAR                     6
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
DEF_30_CNT_SOCIAL_CIRCLE           10
AMT_CREDIT_RANGE                   11
AMT_INCOME_RANGE                   11
AMT_REQ_CREDIT_BUREAU_QRT          11
CNT_CHILDREN                       15
CNT_FAM_MEMBERS                    17
OCCUPATION_TYPE                    18
HOUR_APPR_PROCESS_START            24
AMT_REQ_CREDIT_BUREAU_MON          24
AMT_REQ_CREDIT_BUREAU_YEAR         25
OBS_60_CNT_SOCIAL_CIRCLE           33
OBS_30_CNT_SOCIAL_CIRCLE           33
AGE                                50
YEARS_EMPLOYED                     51
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
AMT_ANNUITY                     13672
DAYS_REGISTRATION               15688
DAYS_BIRTH                      17460
SK_ID_CURR                     307511
dtype: int64
```

In [53]:

```
# 4.5 Data Type Conversion
# inspecting the column types if they are in correct data type using the above resul
applicationDF.info()

Insight:
Numeric columns are already in int64 and float64 format. Hence proceeding with other
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 52 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_CAR                 307511 non-null  object
 5   FLAG_OWN_REALTY              307511 non-null  object
 6   CNT_CHILDREN                 307511 non-null  int64
 7   AMT_INCOME_TOTAL             307511 non-null  float64
 8   AMT_CREDIT                   307511 non-null  float64
 9   AMT_ANNUITY                  307499 non-null  float64
 10  AMT_GOODS_PRICE              307233 non-null  float64
 11  NAME_TYPE_SUITE              306219 non-null  object
 12  NAME_INCOME_TYPE             307511 non-null  object
 13  NAME_EDUCATION_TYPE          307511 non-null  object
 14  NAME_FAMILY_STATUS           307511 non-null  object
 15  NAME_HOUSING_TYPE            307511 non-null  object
 16  REGION_POPULATION_RELATIVE   307511 non-null  float64
 17  DAYS_BIRTH                   307511 non-null  int64
 18  DAYS_EMPLOYED                307511 non-null  int64
 19  DAYS_REGISTRATION            307511 non-null  float64
 20  DAYS_ID_PUBLISH              307511 non-null  int64
 21  OCCUPATION_TYPE              211120 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   265992 non-null  float64
 41  AMT_REQ_CREDIT_BUREAU_DAY    265992 non-null  float64
 42  AMT_REQ_CREDIT_BUREAU_WEEK   265992 non-null  float64
 43  AMT_REQ_CREDIT_BUREAU_MON    265992 non-null  float64
 44  AMT_REQ_CREDIT_BUREAU_QRT    265992 non-null  float64
 45  AMT_REQ_CREDIT_BUREAU_YEAR   265992 non-null  float64
 46  AMT_INCOME_RANGE             307279 non-null  category
 47  AMT_CREDIT_RANGE             307511 non-null  category
 48  AGE                          307511 non-null  int64
 49  AGE_GROUP                    307511 non-null  category
 50  YEARS_EMPLOYED               307511 non-null  int64
```

```
 51  EMPLOYMENT_YEAR              224233 non-null  category
dtypes: category(4), float64(18), int64(18), object(12)
memory usage: 113.8+ MB
```

In [54]:

```
#Conversion of Object and Numerical columns to Categorical Columns
categorical_columns = ['NAME_CONTRACT_TYPE','CODE_GENDER','NAME_TYPE_SUITE','NAME_IN
                       'NAME_FAMILY_STATUS','NAME_HOUSING_TYPE','OCCUPATION_TYPE','W
                       'ORGANIZATION_TYPE','FLAG_OWN_CAR','FLAG_OWN_REALTY','LIVE_CI
                       'REG_CITY_NOT_LIVE_CITY','REG_CITY_NOT_WORK_CITY','REG_REGION
                       'LIVE_REGION_NOT_WORK_REGION','REGION_RATING_CLIENT','WEEKDAY
                       'REGION_RATING_CLIENT_W_CITY'
                      ]
for col in categorical_columns:
    applicationDF[col] =pd.Categorical(applicationDF[col])
```

In [55]:

```
# inspecting the column types if the above conversion is reflected
applicationDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 52 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  category
 3   CODE_GENDER                  307511 non-null  category
 4   FLAG_OWN_CAR                 307511 non-null  category
 5   FLAG_OWN_REALTY              307511 non-null  category
 6   CNT_CHILDREN                 307511 non-null  int64
 7   AMT_INCOME_TOTAL             307511 non-null  float64
 8   AMT_CREDIT                   307511 non-null  float64
 9   AMT_ANNUITY                  307499 non-null  float64
 10  AMT_GOODS_PRICE              307233 non-null  float64
 11  NAME_TYPE_SUITE              306219 non-null  category
 12  NAME_INCOME_TYPE             307511 non-null  category
 13  NAME_EDUCATION_TYPE          307511 non-null  category
 14  NAME_FAMILY_STATUS           307511 non-null  category
 15  NAME_HOUSING_TYPE            307511 non-null  category
 16  REGION_POPULATION_RELATIVE   307511 non-null  float64
 17  DAYS_BIRTH                   307511 non-null  int64
 18  DAYS_EMPLOYED                307511 non-null  int64
 19  DAYS_REGISTRATION            307511 non-null  float64
 20  DAYS_ID_PUBLISH              307511 non-null  int64
 21  OCCUPATION_TYPE              211120 non-null  category
 22  CNT_FAM_MEMBERS              307509 non-null  float64
 23  REGION_RATING_CLIENT         307511 non-null  category
 24  REGION_RATING_CLIENT_W_CITY  307511 non-null  category
 25  WEEKDAY_APPR_PROCESS_START   307511 non-null  category
 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  category
 29  LIVE_REGION_NOT_WORK_REGION  307511 non-null  category
 30  REG_CITY_NOT_LIVE_CITY       307511 non-null  category
 31  REG_CITY_NOT_WORK_CITY       307511 non-null  category
 32  LIVE_CITY_NOT_WORK_CITY      307511 non-null  category
 33  ORGANIZATION_TYPE            307511 non-null  category
 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   265992 non-null  float64
 41  AMT_REQ_CREDIT_BUREAU_DAY    265992 non-null  float64
 42  AMT_REQ_CREDIT_BUREAU_WEEK   265992 non-null  float64
 43  AMT_REQ_CREDIT_BUREAU_MON    265992 non-null  float64
 44  AMT_REQ_CREDIT_BUREAU_QRT    265992 non-null  float64
 45  AMT_REQ_CREDIT_BUREAU_YEAR   265992 non-null  float64
 46  AMT_INCOME_RANGE             307279 non-null  category
 47  AMT_CREDIT_RANGE             307511 non-null  category
 48  AGE                          307511 non-null  int64
 49  AGE_GROUP                    307511 non-null  category
 50  YEARS_EMPLOYED               307511 non-null  int64
 51  EMPLOYMENT_YEAR              224233 non-null  category
dtypes: category(23), float64(18), int64(11)
memory usage: 74.8 MB
```

In [56]:

```
4.4.2 Standardize Values for previousDF
Strategy for previousDF:
Convert DAYS_DECISION from negative to positive values and create categorical bins c
Convert loan purpose and few other columns to categorical.

#Checking the number of unique values each column possess to identify categorical co
previousDF.nunique().sort_values()
```

Out[56]:

```
NAME_PRODUCT_TYPE               3
NAME_PAYMENT_TYPE               4
NAME_CONTRACT_TYPE              4
NAME_CLIENT_TYPE                4
NAME_CONTRACT_STATUS            4
NAME_PORTFOLIO                  5
NAME_YIELD_GROUP                5
CHANNEL_TYPE                    8
CODE_REJECT_REASON              9
NAME_SELLER_INDUSTRY           11
PRODUCT_COMBINATION            17
NAME_CASH_LOAN_PURPOSE         25
NAME_GOODS_CATEGORY            28
CNT_PAYMENT                    49
SELLERPLACE_AREA             2097
DAYS_DECISION                2922
AMT_CREDIT                  86803
AMT_GOODS_PRICE             93885
AMT_APPLICATION             93885
SK_ID_CURR                 338857
AMT_ANNUITY                357959
SK_ID_PREV                1670214
dtype: int64
```

In [57]:

```
# inspecting the column types if the above conversion is reflected
previousDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 22 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_GOODS_PRICE        1284699 non-null  float64
```

```
 7   NAME_CASH_LOAN_PURPOSE  1670214 non-null  object
 8   NAME_CONTRACT_STATUS    1670214 non-null  object
 9   DAYS_DECISION           1670214 non-null  int64
 10  NAME_PAYMENT_TYPE       1670214 non-null  object
 11  CODE_REJECT_REASON      1670214 non-null  object
 12  NAME_CLIENT_TYPE        1670214 non-null  object
 13  NAME_GOODS_CATEGORY     1670214 non-null  object
 14  NAME_PORTFOLIO          1670214 non-null  object
 15  NAME_PRODUCT_TYPE       1670214 non-null  object
 16  CHANNEL_TYPE            1670214 non-null  object
 17  SELLERPLACE_AREA        1670214 non-null  int64
 18  NAME_SELLER_INDUSTRY    1670214 non-null  object
 19  CNT_PAYMENT             1297984 non-null  float64
 20  NAME_YIELD_GROUP        1670214 non-null  object
 21  PRODUCT_COMBINATION     1669868 non-null  object
dtypes: float64(5), int64(4), object(13)
memory usage: 280.3+ MB
```

In [58]:

```
#Converting negative days to positive days
previousDF['DAYS_DECISION'] = abs(previousDF['DAYS_DECISION'])
```

In [59]:

```
#age group calculation e.g. 388 will be grouped as 300-400
previousDF['DAYS_DECISION_GROUP'] = (previousDF['DAYS_DECISION']-(previousDF['DAYS_D
```

In [60]:

```
previousDF['DAYS_DECISION_GROUP'].value_counts(normalize=True)*100

#Insight:
Almost 37% loan applicatants have applied for a new loan within 0-400 days of previo
```

Out[60]:

```
0-400        37.490525
400-800      22.944724
800-1200     12.444753
1200-1600     7.904556
2400-2800     6.297456
1600-2000     5.795784
2000-2400     5.684960
2800-3200     1.437241
Name: DAYS_DECISION_GROUP, dtype: float64
```

In [61]:

```
#Converting Categorical columns from Object to categorical
Catgorical_col_p = ['NAME_CASH_LOAN_PURPOSE','NAME_CONTRACT_STATUS','NAME_PAYMENT_TY
                    'CODE_REJECT_REASON','NAME_CLIENT_TYPE','NAME_GOODS_CATEGORY','N
                   'NAME_PRODUCT_TYPE','CHANNEL_TYPE','NAME_SELLER_INDUSTRY','NAME_Y
                    'NAME_CONTRACT_TYPE','DAYS_DECISION_GROUP']

for col in Catgorical_col_p:
    previousDF[col] =pd.Categorical(previousDF[col])
```

In [62]:

```
# inspecting the column types after conversion
previousDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 23 columns):
 #   Column                  Non-Null Count    Dtype
---  ------                  --------------    -----
 0   SK_ID_PREV              1670214 non-null  int64
 1   SK_ID_CURR              1670214 non-null  int64
 2   NAME_CONTRACT_TYPE      1670214 non-null  category
```

```
 3   AMT_ANNUITY              1297979 non-null  float64
 4   AMT_APPLICATION          1670214 non-null  float64
 5   AMT_CREDIT               1670213 non-null  float64
 6   AMT_GOODS_PRICE          1284699 non-null  float64
 7   NAME_CASH_LOAN_PURPOSE   1670214 non-null  category
 8   NAME_CONTRACT_STATUS     1670214 non-null  category
 9   DAYS_DECISION            1670214 non-null  int64
 10  NAME_PAYMENT_TYPE        1670214 non-null  category
 11  CODE_REJECT_REASON       1670214 non-null  category
 12  NAME_CLIENT_TYPE         1670214 non-null  category
 13  NAME_GOODS_CATEGORY      1670214 non-null  category
 14  NAME_PORTFOLIO           1670214 non-null  category
 15  NAME_PRODUCT_TYPE        1670214 non-null  category
 16  CHANNEL_TYPE             1670214 non-null  category
 17  SELLERPLACE_AREA         1670214 non-null  int64
 18  NAME_SELLER_INDUSTRY     1670214 non-null  category
 19  CNT_PAYMENT              1297984 non-null  float64
 20  NAME_YIELD_GROUP         1670214 non-null  category
 21  PRODUCT_COMBINATION      1669868 non-null  category
 22  DAYS_DECISION_GROUP      1670214 non-null  category
dtypes: category(14), float64(5), int64(4)
memory usage: 137.0 MB
```

In [63]:

```
4.6 Null Value Data Imputation
4.6.1 Imputing Null Values in applicationDF
Strategy for applicationDF:
To impute null values in categorical variables which has lower null percentage, mode
To impute null values in categorical variables which has higher null percentage, a n
To impute null values in numerical variables which has lower null percentage, median
There are no outliers in the columns
Mean returned decimal values and median returned whole numbers and the columns were

# checking the null value % of each column in applicationDF dataframe
round(applicationDF.isnull().sum() / applicationDF.shape[0] * 100.00,2)

#Impute categorical variable 'NAME_TYPE_SUITE' which has lower null percentage(0.42%
```

Out[63]:

```
SK_ID_CURR                    0.00
TARGET                        0.00
NAME_CONTRACT_TYPE            0.00
CODE_GENDER                   0.00
FLAG_OWN_CAR                  0.00
FLAG_OWN_REALTY               0.00
CNT_CHILDREN                  0.00
AMT_INCOME_TOTAL              0.00
AMT_CREDIT                    0.00
AMT_ANNUITY                   0.00
AMT_GOODS_PRICE               0.09
NAME_TYPE_SUITE               0.42
NAME_INCOME_TYPE              0.00
NAME_EDUCATION_TYPE           0.00
NAME_FAMILY_STATUS            0.00
NAME_HOUSING_TYPE             0.00
REGION_POPULATION_RELATIVE    0.00
DAYS_BIRTH                    0.00
DAYS_EMPLOYED                 0.00
DAYS_REGISTRATION             0.00
DAYS_ID_PUBLISH               0.00
OCCUPATION_TYPE              31.35
CNT_FAM_MEMBERS               0.00
REGION_RATING_CLIENT          0.00
REGION_RATING_CLIENT_W_CITY   0.00
WEEKDAY_APPR_PROCESS_START    0.00
```

```
HOUR_APPR_PROCESS_START         0.00
REG_REGION_NOT_LIVE_REGION      0.00
REG_REGION_NOT_WORK_REGION      0.00
LIVE_REGION_NOT_WORK_REGION     0.00
REG_CITY_NOT_LIVE_CITY          0.00
REG_CITY_NOT_WORK_CITY          0.00
LIVE_CITY_NOT_WORK_CITY         0.00
ORGANIZATION_TYPE               0.00
OBS_30_CNT_SOCIAL_CIRCLE        0.33
DEF_30_CNT_SOCIAL_CIRCLE        0.33
OBS_60_CNT_SOCIAL_CIRCLE        0.33
DEF_60_CNT_SOCIAL_CIRCLE        0.33
DAYS_LAST_PHONE_CHANGE          0.00
FLAG_DOCUMENT_3                 0.00
AMT_REQ_CREDIT_BUREAU_HOUR     13.50
AMT_REQ_CREDIT_BUREAU_DAY      13.50
AMT_REQ_CREDIT_BUREAU_WEEK     13.50
AMT_REQ_CREDIT_BUREAU_MON      13.50
AMT_REQ_CREDIT_BUREAU_QRT      13.50
AMT_REQ_CREDIT_BUREAU_YEAR     13.50
AMT_INCOME_RANGE                0.08
AMT_CREDIT_RANGE                0.00
AGE                             0.00
AGE_GROUP                       0.00
YEARS_EMPLOYED                  0.00
EMPLOYMENT_YEAR                27.08
dtype: float64
```

In [64]:
```
applicationDF['NAME_TYPE_SUITE'].describe()
```

Out[64]:
```
count            306219
unique                7
top       Unaccompanied
freq             248526
Name: NAME_TYPE_SUITE, dtype: object
```

In [66]:
```
applicationDF['NAME_TYPE_SUITE'].fillna((applicationDF['NAME_TYPE_SUITE'].mode()[0])


#Impute categorical variable 'OCCUPATION_TYPE' which has higher null percentage(31.3
category as assigning to any existing category might influence the analysis:
```

In [67]:
```
applicationDF['OCCUPATION_TYPE'] = applicationDF['OCCUPATION_TYPE'].cat.add_categori
applicationDF['OCCUPATION_TYPE'].fillna('Unknown', inplace =True)

Impute numerical variables with the median as there are no outliers that can be seen
of describe() and mean() returns decimal values and these columns represent number o
which cannot be decimal:
```

In [68]:
```
applicationDF[['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']].describe()

#Impute with median as mean has decimals and this is number of requests
```

Out[68]:

	AMT_REQ_CREDIT_BUREAU_HOUR	AMT_REQ_CREDIT_BUREAU_DAY	AMT_REQ_CREDIT_BUREAU_
count	265992.000000	265992.000000	265992.0
mean	0.006402	0.007000	0.0

	AMT_REQ_CREDIT_BUREAU_HOUR	AMT_REQ_CREDIT_BUREAU_DAY	AMT_REQ_CREDIT_BUREAU_
std	0.083849	0.110757	0.2
min	0.000000	0.000000	0.0
25%	0.000000	0.000000	0.0
50%	0.000000	0.000000	0.0
75%	0.000000	0.000000	0.0
max	4.000000	9.000000	8.0

In [69]:

```
amount = ['AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY','AMT_REQ_CREDIT_
          'AMT_REQ_CREDIT_BUREAU_QRT','AMT_REQ_CREDIT_BUREAU_YEAR']

for col in amount:
    applicationDF[col].fillna(applicationDF[col].median(),inplace = True)
```

In [70]:

```
# checking the null value % of each column in previousDF dataframe
round(applicationDF.isnull().sum() / previousDF.shape[0] * 100.00,2)


Insight:
We still have few null values in the columns: AMT_GOODS_PRICE, OBS_30_CNT_SOCIAL_CIR
    DEF_30_CNT_SOCIAL_CIRCLE, OBS_60_CNT_SOCIAL_CIRCLE, DEF_60_CNT_SOCIAL_CIRCLE. We
    as this percentage is very less.
```

Out[70]:

```
SK_ID_CURR                    0.00
TARGET                        0.00
NAME_CONTRACT_TYPE            0.00
CODE_GENDER                   0.00
FLAG_OWN_CAR                  0.00
FLAG_OWN_REALTY               0.00
CNT_CHILDREN                  0.00
AMT_INCOME_TOTAL              0.00
AMT_CREDIT                    0.00
AMT_ANNUITY                   0.00
AMT_GOODS_PRICE               0.02
NAME_TYPE_SUITE               0.00
NAME_INCOME_TYPE              0.00
NAME_EDUCATION_TYPE           0.00
NAME_FAMILY_STATUS            0.00
NAME_HOUSING_TYPE             0.00
REGION_POPULATION_RELATIVE    0.00
DAYS_BIRTH                    0.00
DAYS_EMPLOYED                 0.00
DAYS_REGISTRATION             0.00
DAYS_ID_PUBLISH               0.00
OCCUPATION_TYPE               0.00
CNT_FAM_MEMBERS               0.00
REGION_RATING_CLIENT          0.00
REGION_RATING_CLIENT_W_CITY   0.00
WEEKDAY_APPR_PROCESS_START    0.00
HOUR_APPR_PROCESS_START       0.00
REG_REGION_NOT_LIVE_REGION    0.00
REG_REGION_NOT_WORK_REGION    0.00
LIVE_REGION_NOT_WORK_REGION   0.00
REG_CITY_NOT_LIVE_CITY        0.00
REG_CITY_NOT_WORK_CITY        0.00
```

```
LIVE_CITY_NOT_WORK_CITY        0.00
ORGANIZATION_TYPE              0.00
OBS_30_CNT_SOCIAL_CIRCLE       0.06
DEF_30_CNT_SOCIAL_CIRCLE       0.06
OBS_60_CNT_SOCIAL_CIRCLE       0.06
DEF_60_CNT_SOCIAL_CIRCLE       0.06
DAYS_LAST_PHONE_CHANGE         0.00
FLAG_DOCUMENT_3                0.00
AMT_REQ_CREDIT_BUREAU_HOUR     0.00
AMT_REQ_CREDIT_BUREAU_DAY      0.00
AMT_REQ_CREDIT_BUREAU_WEEK     0.00
AMT_REQ_CREDIT_BUREAU_MON      0.00
AMT_REQ_CREDIT_BUREAU_QRT      0.00
AMT_REQ_CREDIT_BUREAU_YEAR     0.00
AMT_INCOME_RANGE               0.01
AMT_CREDIT_RANGE               0.00
AGE                            0.00
AGE_GROUP                      0.00
YEARS_EMPLOYED                 0.00
EMPLOYMENT_YEAR                4.99
dtype: float64
```

In [71]:

```
4.6.2 Imputing Null Values in previousDF
Strategy for applicationDF:
To impute null values in numerical column, we analysed the loan status and assigned
To impute null values in continuous variables, we plotted the distribution of the co
median if the distribution is skewed
mode if the distribution pattern is preserved.

# checking the null value % of each column in previousDF dataframe
round(previousDF.isnull().sum() / previousDF.shape[0] * 100.00,2)

#Impute AMT_ANNUITY with median as the distribution is greatly skewed:
```

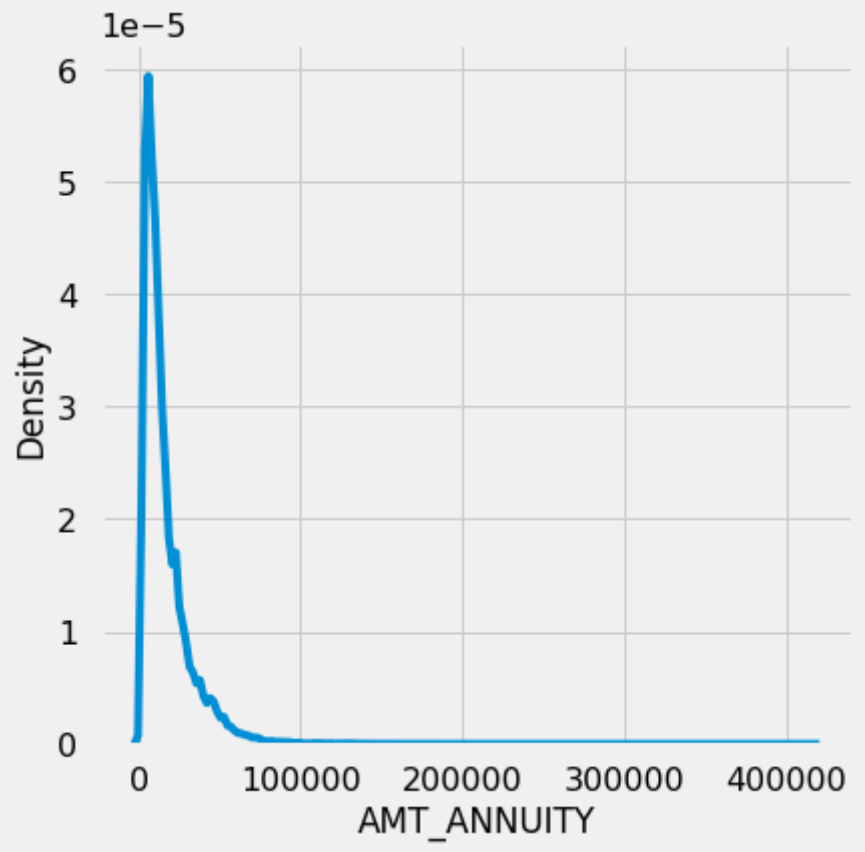
Out[71]:

```
SK_ID_PREV                 0.00
SK_ID_CURR                 0.00
NAME_CONTRACT_TYPE         0.00
AMT_ANNUITY               22.29
AMT_APPLICATION            0.00
AMT_CREDIT                 0.00
AMT_GOODS_PRICE           23.08
NAME_CASH_LOAN_PURPOSE     0.00
NAME_CONTRACT_STATUS       0.00
DAYS_DECISION              0.00
NAME_PAYMENT_TYPE          0.00
CODE_REJECT_REASON         0.00
NAME_CLIENT_TYPE           0.00
NAME_GOODS_CATEGORY        0.00
NAME_PORTFOLIO             0.00
NAME_PRODUCT_TYPE          0.00
CHANNEL_TYPE               0.00
SELLERPLACE_AREA           0.00
NAME_SELLER_INDUSTRY       0.00
CNT_PAYMENT               22.29
NAME_YIELD_GROUP           0.00
PRODUCT_COMBINATION        0.02
DAYS_DECISION_GROUP        0.00
dtype: float64
```

In [72]:

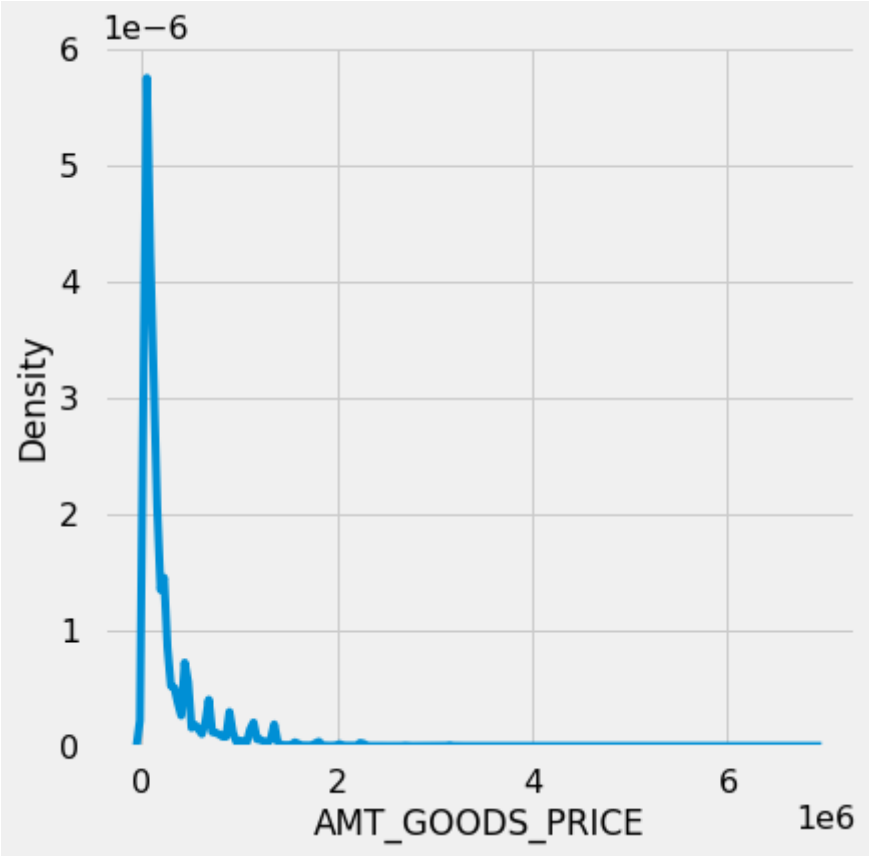
```
plt.figure(figsize=(6,6))
sns.kdeplot(previousDF['AMT_ANNUITY'])
plt.show()
```

```
#Insight:
There is a single peak at the left side of the distribution and it indicates the pre
and hence imputing with mean would not be the right approach and hence imputing with
```



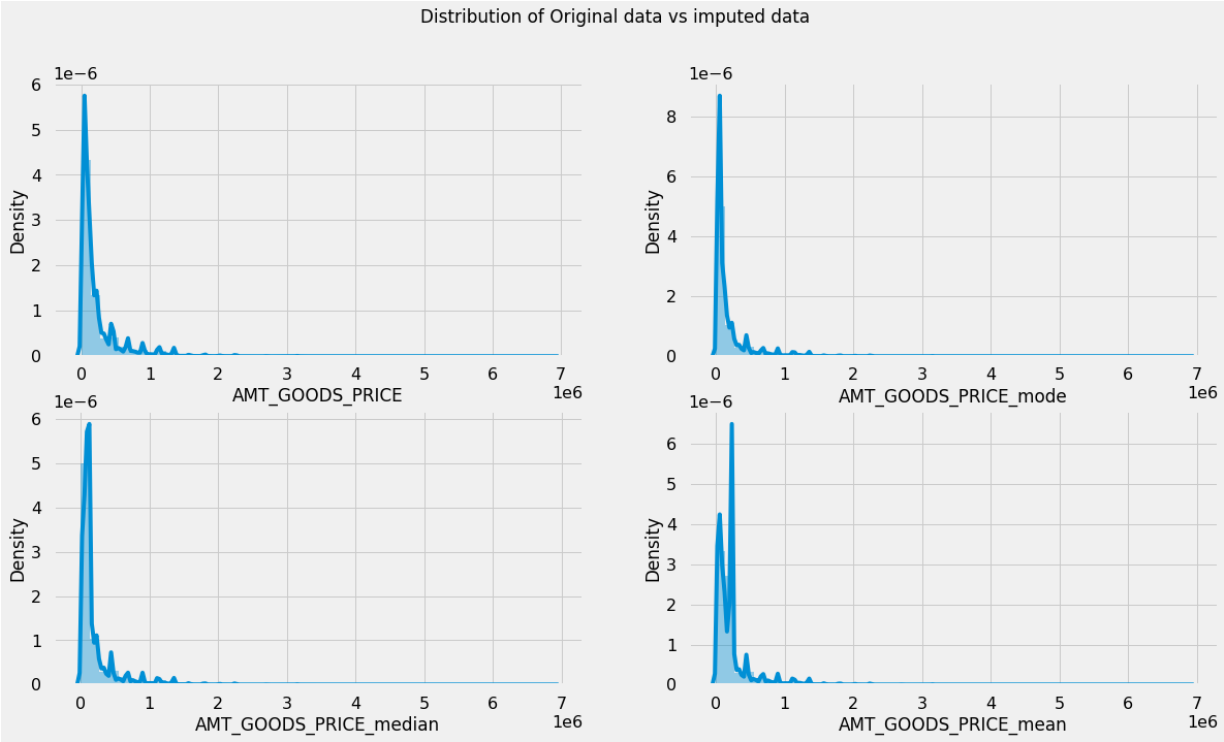
In [73]:
```
previousDF['AMT_ANNUITY'].fillna(previousDF['AMT_ANNUITY'].median(),inplace = True)

#Impute AMT_GOODS_PRICE with mode as the distribution is closely similar:
```

In [74]:
```
plt.figure(figsize=(6,6))
sns.kdeplot(previousDF['AMT_GOODS_PRICE'][pd.notnull(previousDF['AMT_GOODS_PRICE'])]
plt.show()


#There are several peaks along the distribution. Let's impute using the mode, mean a
see if the distribution is still about the same.
```



In [78]:

```
statsDF = pd.DataFrame() # new dataframe with columns imputed with mode, median and
statsDF['AMT_GOODS_PRICE_mode'] = previousDF['AMT_GOODS_PRICE'].fillna(previousDF['A
statsDF['AMT_GOODS_PRICE_median'] = previousDF['AMT_GOODS_PRICE'].fillna(previousDF[
statsDF['AMT_GOODS_PRICE_mean'] = previousDF['AMT_GOODS_PRICE'].fillna(previousDF['A

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(previousDF['AMT_GOODS_PRICE'][pd.notnull(previousDF['AMT_GOODS_PRICE'])
for i in enumerate(cols):
    plt.subplot(2,2,i[0]+2)
    sns.distplot(statsDF[i[1]])


#Insight:
#The original distribution is closer with the distribution of data imputed with mode
```



In [80]:
```
previousDF['AMT_GOODS_PRICE'].fillna(previousDF['AMT_GOODS_PRICE'].mode()[0], inplac

#Impute CNT_PAYMENT with 0 as the NAME_CONTRACT_STATUS for these indicate that most
#were not started:
```

In [81]:
```
previousDF.loc[previousDF['CNT_PAYMENT'].isnull(),'NAME_CONTRACT_STATUS'].value_coun
```

Out[81]:
```
Canceled        305805
Refused          40897
Unused offer     25524
Approved             4
Name: NAME_CONTRACT_STATUS, dtype: int64
```

In [82]:
```
previousDF['CNT_PAYMENT'].fillna(0,inplace = True)
```

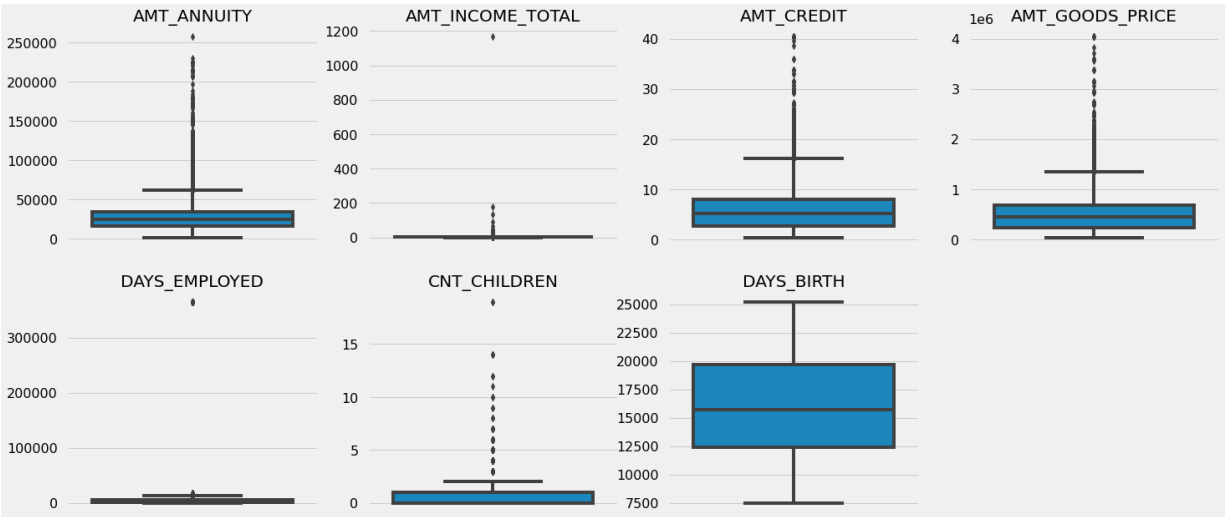
In [83]:
```
# checking the null value % of each column in previousDF dataframe
round(previousDF.isnull().sum() / previousDF.shape[0] * 100.00,2)

#Insight:
We still have few null values in the PRODUCT_COMBINATION column.
We can ignore as this percentage is very less.
```

Out[83]:
```
SK_ID_PREV                0.00
SK_ID_CURR                0.00
NAME_CONTRACT_TYPE        0.00
AMT_ANNUITY               0.00
AMT_APPLICATION           0.00
AMT_CREDIT                0.00
AMT_GOODS_PRICE           0.00
NAME_CASH_LOAN_PURPOSE    0.00
NAME_CONTRACT_STATUS      0.00
DAYS_DECISION             0.00
NAME_PAYMENT_TYPE         0.00
CODE_REJECT_REASON        0.00
NAME_CLIENT_TYPE          0.00
NAME_GOODS_CATEGORY       0.00
```

```
NAME_PORTFOLIO            0.00
NAME_PRODUCT_TYPE         0.00
CHANNEL_TYPE              0.00
SELLERPLACE_AREA          0.00
NAME_SELLER_INDUSTRY      0.00
CNT_PAYMENT               0.00
NAME_YIELD_GROUP          0.00
PRODUCT_COMBINATION       0.02
DAYS_DECISION_GROUP       0.00
dtype: float64
```

In [84]:

```
# 4.7 Identifying the outliers
# Finding outlier information in applicationDF

plt.figure(figsize=(22,10))

app_outlier_col_1 = ['AMT_ANNUITY','AMT_INCOME_TOTAL','AMT_CREDIT','AMT_GOODS_PRICE'
app_outlier_col_2 = ['CNT_CHILDREN','DAYS_BIRTH']
for i in enumerate(app_outlier_col_1):
    plt.subplot(2,4,i[0]+1)
    sns.boxplot(y=applicationDF[i[1]])
    plt.title(i[1])
    plt.ylabel("")

for i in enumerate(app_outlier_col_2):
    plt.subplot(2,4,i[0]+6)
    sns.boxplot(y=applicationDF[i[1]])
    plt.title(i[1])
    plt.ylabel("")



#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 app
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
and hence this has to be incorrect entry.
```



In [85]:

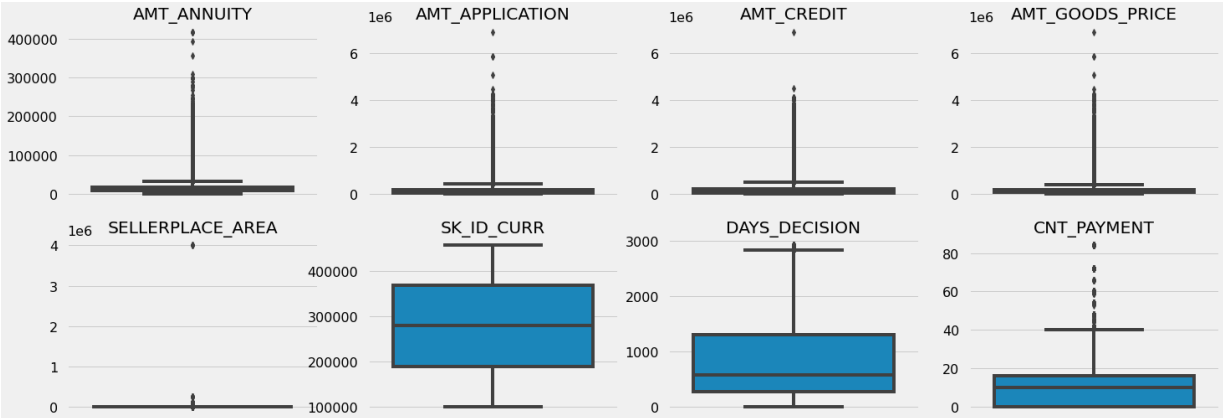
```
applicationDF[['AMT_ANNUITY', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_GOODS_PRICE', '


We can see the stats for these columns below as WELL
```

Out[85]:

	AMT_ANNUITY	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_GOODS_PRICE	DAYS_BIRTH	CNT
count	307499.000000	307511.000000	307511.000000	3.072330e+05	307511.000000	30
mean	27108.573909	1.687979	5.990260	5.383962e+05	16036.995067	
std	14493.737315	2.371231	4.024908	3.694465e+05	4363.988632	
min	1615.500000	0.256500	0.450000	4.050000e+04	7489.000000	
25%	16524.000000	1.125000	2.700000	2.385000e+05	12413.000000	
50%	24903.000000	1.471500	5.135310	4.500000e+05	15750.000000	
75%	34596.000000	2.025000	8.086500	6.795000e+05	19682.000000	
max	258025.500000	1170.000000	40.500000	4.050000e+06	25229.000000	

In [86]:

```
#Finding outlier information in previousDF

plt.figure(figsize=(22,8))

prev_outlier_col_1 = ['AMT_ANNUITY','AMT_APPLICATION','AMT_CREDIT','AMT_GOODS_PRICE'
prev_outlier_col_2 = ['SK_ID_CURR','DAYS_DECISION','CNT_PAYMENT']
for i in enumerate(prev_outlier_col_1):
    plt.subplot(2,4,i[0]+1)
    sns.boxplot(y=previousDF[i[1]])
    plt.title(i[1])
    plt.ylabel("")

for i in enumerate(prev_outlier_col_2):
    plt.subplot(2,4,i[0]+6)
    sns.boxplot(y=previousDF[i[1]])
    plt.title(i[1])
    plt.ylabel("")


Insight: It can be seen that in previous application data
AMT_ANNUITY, AMT_APPLICATION, AMT_CREDIT, AMT_GOODS_PRICE, SELLERPLACE_AREA have hug
CNT_PAYMENT has few outlier values.
SK_ID_CURR is an ID column and hence no outliers.
DAYS_DECISION has little number of outliers indicating that these previous applicati
were taken long back.
```



In [87]:

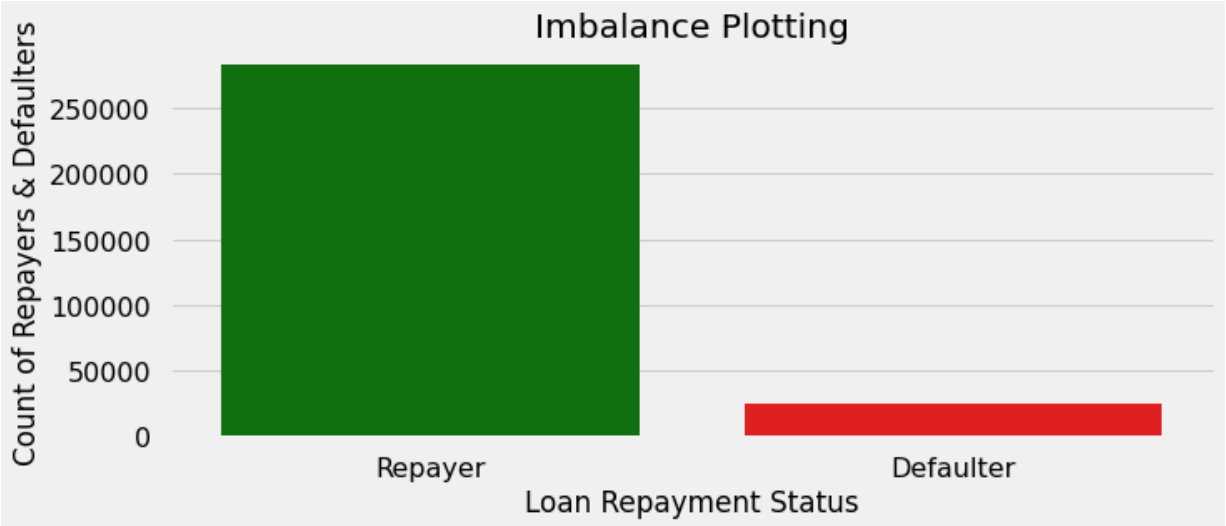
```
# We can see the stats for these columns below as well.

previousDF[['AMT_ANNUITY', 'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_GOODS_PRICE', 'SELL
```

Out[87]:

	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE	SELLERPLACE_AREA	C
count	1.670214e+06	1.670214e+06	1.670213e+06	1.670214e+06	1.670214e+06	
mean	1.490651e+04	1.752339e+05	1.961140e+05	1.856429e+05	3.139511e+02	
std	1.317751e+04	2.927798e+05	3.185746e+05	2.871413e+05	7.127443e+03	
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	-1.000000e+00	
25%	7.547096e+03	1.872000e+04	2.416050e+04	4.500000e+04	-1.000000e+00	
50%	1.125000e+04	7.104600e+04	8.054100e+04	7.105050e+04	3.000000e+00	
75%	1.682403e+04	1.803600e+05	2.164185e+05	1.804050e+05	8.200000e+01	
max	4.180581e+05	6.905160e+06	6.905160e+06	6.905160e+06	4.000000e+06	

In [88]:

```
5. Data Analysis
Strategy:
The data analysis flow has been planned in following way :

Imbalance in Data
Categorical Data Analysis
Categorical segmented Univariate Analysis
Categorical Bi/Multivariate analysis
Numeric Data Analysis
Bi-furcation of databased based on TARGET data
Correlation Matrix
Numerical segmented Univariate Analysis
Numerical Bi/Multivariate analysis

# 5.1 Imbalance Analysis
Imbalance = applicationDF["TARGET"].value_counts().reset_index()

plt.figure(figsize=(10,4))
x= ['Repayer','Defaulter']
sns.barplot(x,"TARGET",data = Imbalance,palette= ['g','r'])
plt.xlabel("Loan Repayment Status")
plt.ylabel("Count of Repayers & Defaulters")
plt.title("Imbalance Plotting")
plt.show()
```



In [89]:

```
count_0 = Imbalance.iloc[0]["TARGET"]
count_1 = Imbalance.iloc[1]["TARGET"]
count_0_perc = round(count_0/(count_0+count_1)*100,2)
count_1_perc = round(count_1/(count_0+count_1)*100,2)

print('Ratios of imbalance in percentage with respect to Repayer and Defaulter datas
print('Ratios of imbalance in relative with respect to Repayer and Defaulter datas i
```

```
Ratios of imbalance in percentage with respect to Repayer and Defaulter datas are: 9
1.93 and 8.07
Ratios of imbalance in relative with respect to Repayer and Defaulter datas is 11.39
: 1 (approx)
```

In [91]:

```
# 5.2 Plotting Functions
# Following are the common functions customized to perform uniform anaysis that is c

# function for plotting repetitive countplots in univariate categorical analysis on
# This function will create two subplots:
# 1. Count plot of categorical column w.r.t TARGET;
# 2. Percentage of defaulters within column

def univariate_categorical(feature,ylog=False,label_rotation=False,horizontal_layout
    temp = applicationDF[feature].value_counts()
    df1 = pd.DataFrame({feature: temp.index,'Number of contracts': temp.values})

    # Calculate the percentage of target=1 per category value
    cat_perc = applicationDF[[feature, 'TARGET']].groupby([feature],as_index=False).
    cat_perc["TARGET"] = cat_perc["TARGET"]*100
    cat_perc.sort_values(by='TARGET', ascending=False, inplace=True)

    if(horizontal_layout):
        fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12,6))
    else:
        fig, (ax1, ax2) = plt.subplots(nrows=2, figsize=(20,24))

    # 1. Subplot 1: Count plot of categorical column
    # sns.set_palette("Set2")
    s = sns.countplot(ax=ax1,
                      x = feature,
                      data=applicationDF,
                      hue ="TARGET",
                      order=cat_perc[feature],
                      palette=['g','r'])

    # Define common styling
    ax1.set_title(feature, fontdict={'fontsize' : 10, 'fontweight' : 3, 'color' : 'B
    ax1.legend(['Repayer','Defaulter'])

    # If the plot is not readable, use the log scale.
    if ylog:
        ax1.set_yscale('log')
        ax1.set_ylabel("Count (log)",fontdict={'fontsize' : 10, 'fontweight' : 3, 'c


    if(label_rotation):
        s.set_xticklabels(s.get_xticklabels(),rotation=90)

    # 2. Subplot 2: Percentage of defaulters within the categorical column
    s = sns.barplot(ax=ax2,
                      x = feature,
                      y='TARGET',
                      order=cat_perc[feature],
```

```
                    data=cat_perc,
                    palette='Set2')

    if(label_rotation):
        s.set_xticklabels(s.get_xticklabels(),rotation=90)
    plt.ylabel('Percent of Defaulters [%]', fontsize=10)
    plt.tick_params(axis='both', which='major', labelsize=10)
    ax2.set_title(feature + " Defaulter %", fontdict={'fontsize' : 15, 'fontweight'

    plt.show();
```

In [92]:

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

def bivariate_bar(x,y,df,hue,figsize):

    plt.figure(figsize=figsize)
    sns.barplot(x=x,
                y=y,
                data=df,
                hue=hue,
                palette =['g','r'])

    # Defining aesthetics of Labels and Title of the plot using style dictionaries
    plt.xlabel(x,fontdict={'fontsize' : 10, 'fontweight' : 3, 'color' : 'Blue'})
    plt.ylabel(y,fontdict={'fontsize' : 10, 'fontweight' : 3, 'color' : 'Blue'})
    plt.title(col, fontdict={'fontsize' : 15, 'fontweight' : 5, 'color' : 'Blue'})
    plt.xticks(rotation=90, ha='right')
    plt.legend(labels = ['Repayer','Defaulter'])
    plt.show()
```

In [93]:

```
# function for plotting repetitive rel plots in bivaritae numerical analysis on appl

def bivariate_rel(x,y,data, hue, kind, palette, legend,figsize):

    plt.figure(figsize=figsize)
    sns.relplot(x=x,
                y=y,
                data=applicationDF,
                hue="TARGET",
                kind=kind,
                palette = ['g','r'],
                legend = False)
    plt.legend(['Repayer','Defaulter'])
    plt.xticks(rotation=90, ha='right')
    plt.show()
```

In [94]:

```
#function for plotting repetitive countplots in univariate categorical analysis on t

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


    if ylog:
        plt.yscale('log')
        plt.ylabel("Count (log)",fontdict={'fontsize' : 10, 'fontweight' : 3, 'color
```

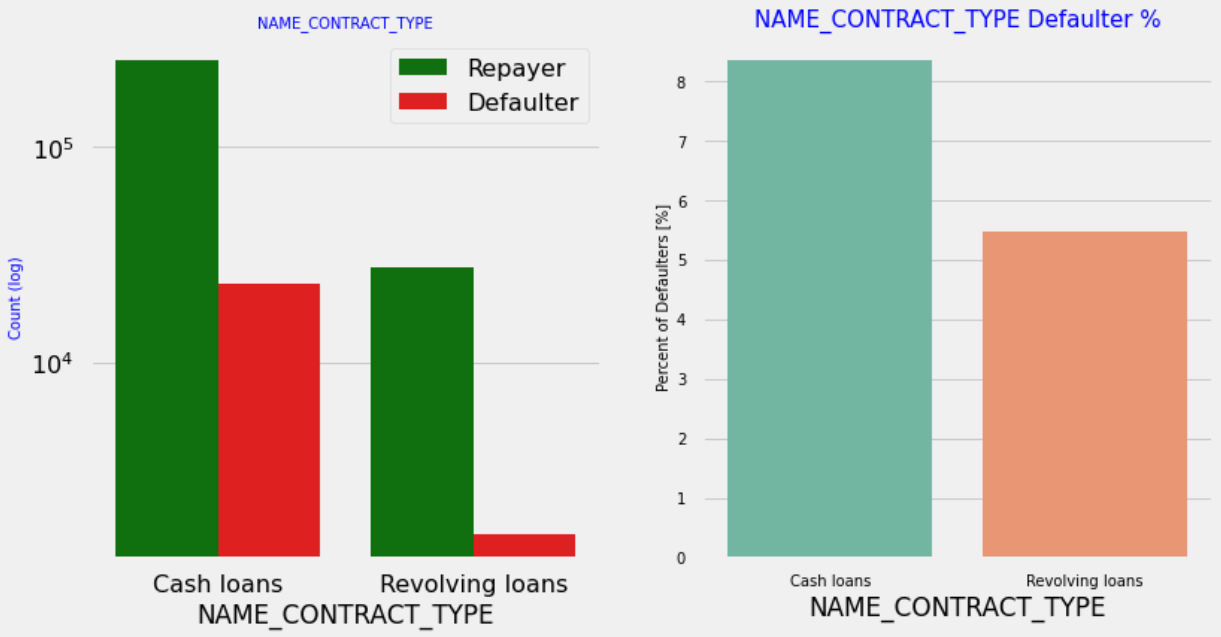
```
    else:
        plt.ylabel("Count",fontdict={'fontsize' : 10, 'fontweight' : 3, 'color' : 'B

    plt.title(col , fontdict={'fontsize' : 15, 'fontweight' : 5, 'color' : 'Blue'})
    plt.legend(loc = "upper right")
    plt.xticks(rotation=90, ha='right')

    plt.show()
```

In [95]:

```
# Function to plot point plots on merged dataframe

def merged_pointplot(x,y):
    plt.figure(figsize=(8,4))
    sns.pointplot(x=x,
                  y=y,
                  hue="TARGET",
                  data=loan_process_df,
                  palette =['g','r'])
   # plt.legend(['Repayer','Defaulter'])
```

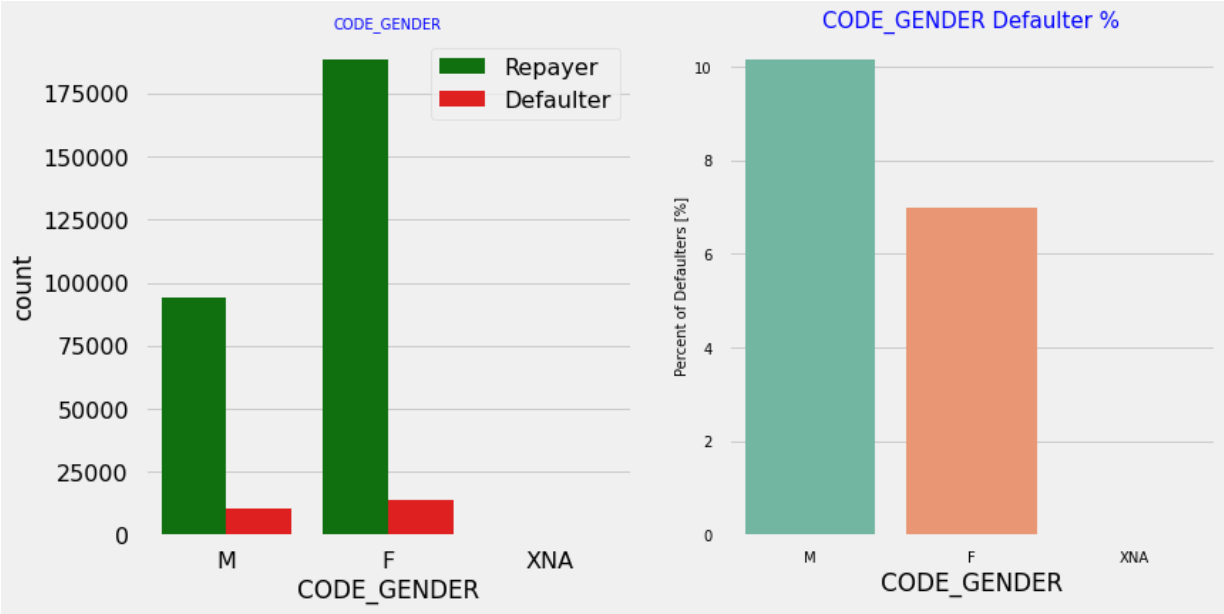
In [96]:

```
# 5.3 Categorical Variables Analysis
# 5.3.1 Segmented Univariate Analysis
# Checking the contract type based on loan repayment status
univariate_categorical('NAME_CONTRACT_TYPE',True)


#Inferences:
Contract type: Revolving loans are just a small fraction (10%) from the total number
in the same time, a larger amount of Revolving loans, comparing with their frequency
```



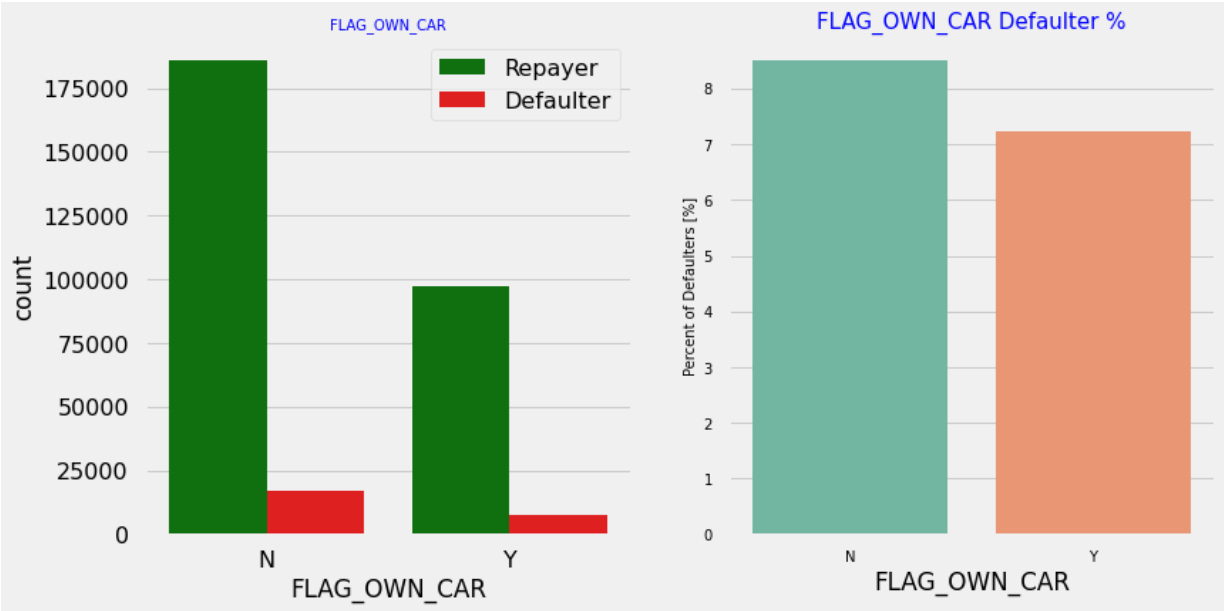
In [97]:

```
# Checking the type of Gender on loan repayment status
univariate_categorical('CODE_GENDER')


#Inferences:
The number of female clients is almost double the number of male clients. Based on t
defaulted credits, males have a higher chance of not returning their loans (~10%), c
women (~7%)
```



In [98]:

```
# Checking if owning a car is related to loan repayment status
univariate_categorical('FLAG_OWN_CAR')


#Inferences:
Clients who own a car are half in number of the clients who dont own a car. But base
of deault, there is no correlation between owning a car and loan repayment as in bot
percentage is almost same.
```

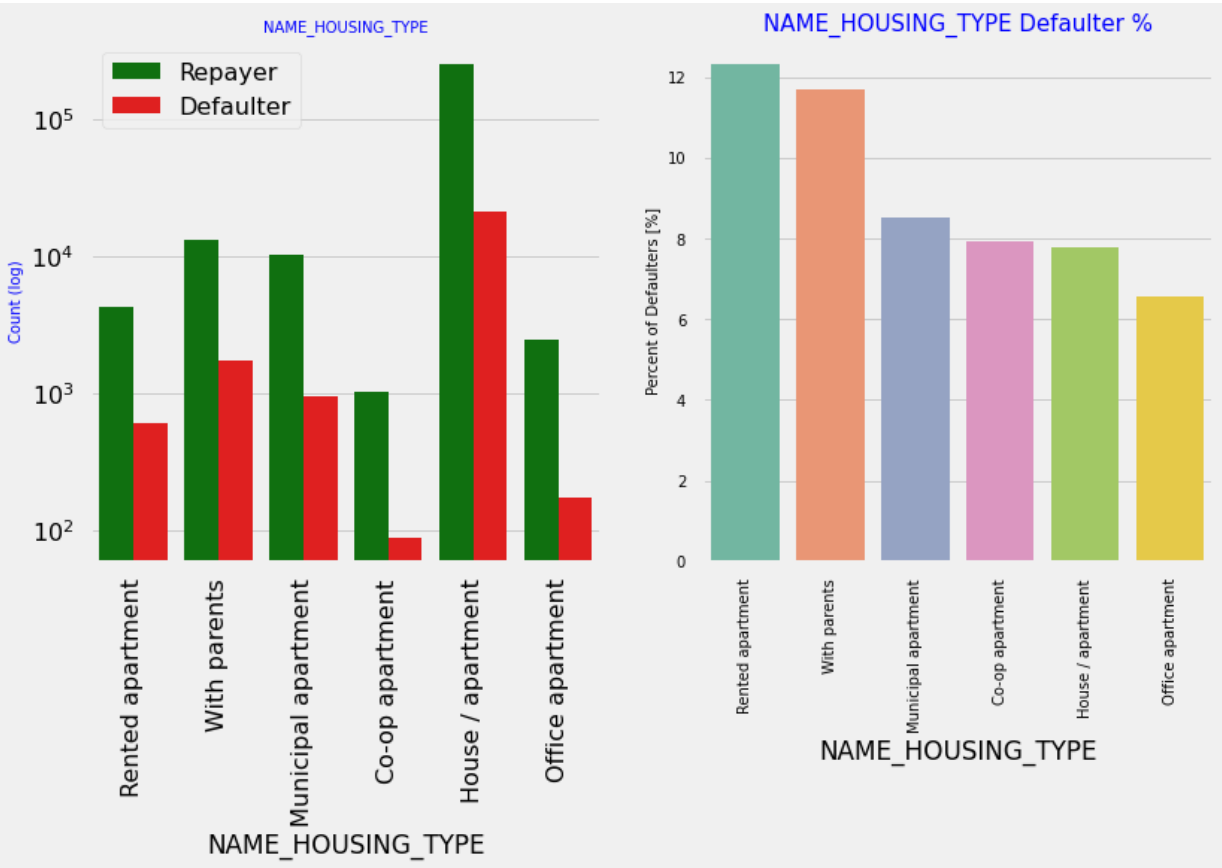


In [99]:

```
# Checking if owning a realty is related to loan repayment status
univariate_categorical('FLAG_OWN_REALTY')

#Inferences:
The clients who own real estate are more than double of the ones that don't own.
But the defaulting rate of both categories are around the same (~8%).
Thus there is no correlation between owning a reality and defaulting the loan.
```

In [100…

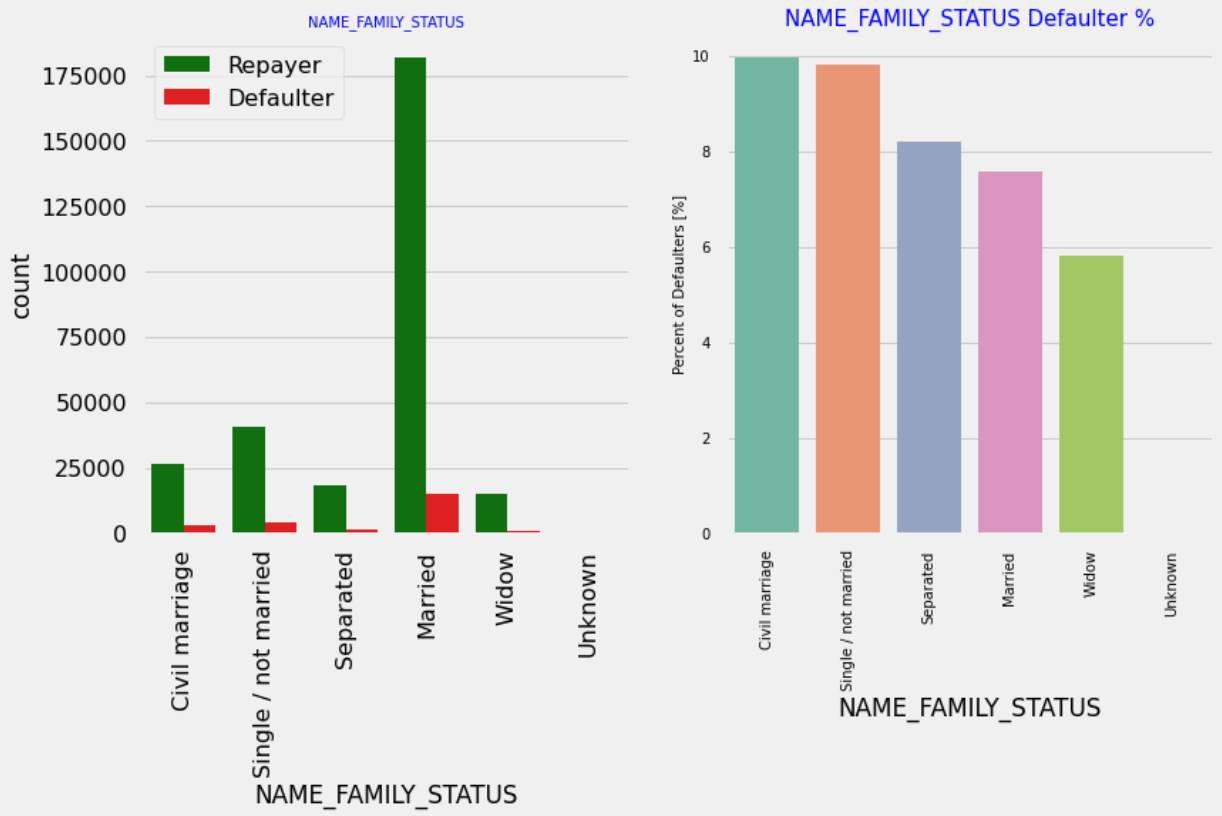
```
# Analyzing Housing Type based on loan repayment status
univariate_categorical("NAME_HOUSING_TYPE",True,True,True)


#Inferences:
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
```



In [101…

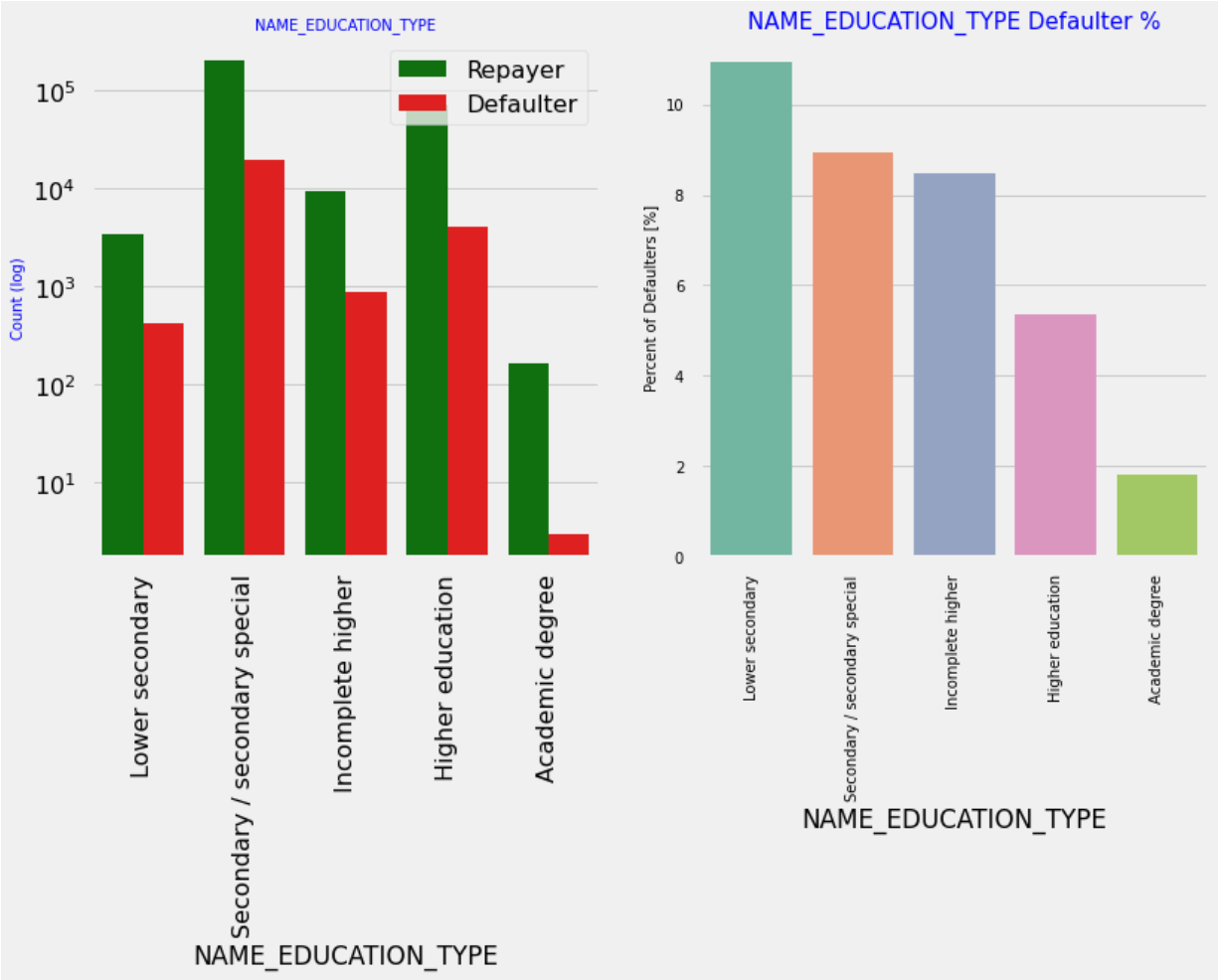
```
# Analyzing Family status based on loan repayment status
univariate_categorical("NAME_FAMILY_STATUS",False,True,True)


#Inferences:
```

Most of the people who have taken loan are married, followed by Single/not married a
In terms of percentage of not repayment of loan, Civil marriage has the highest perc
repayment (10%), with Widow the lowest (exception being Unknown).



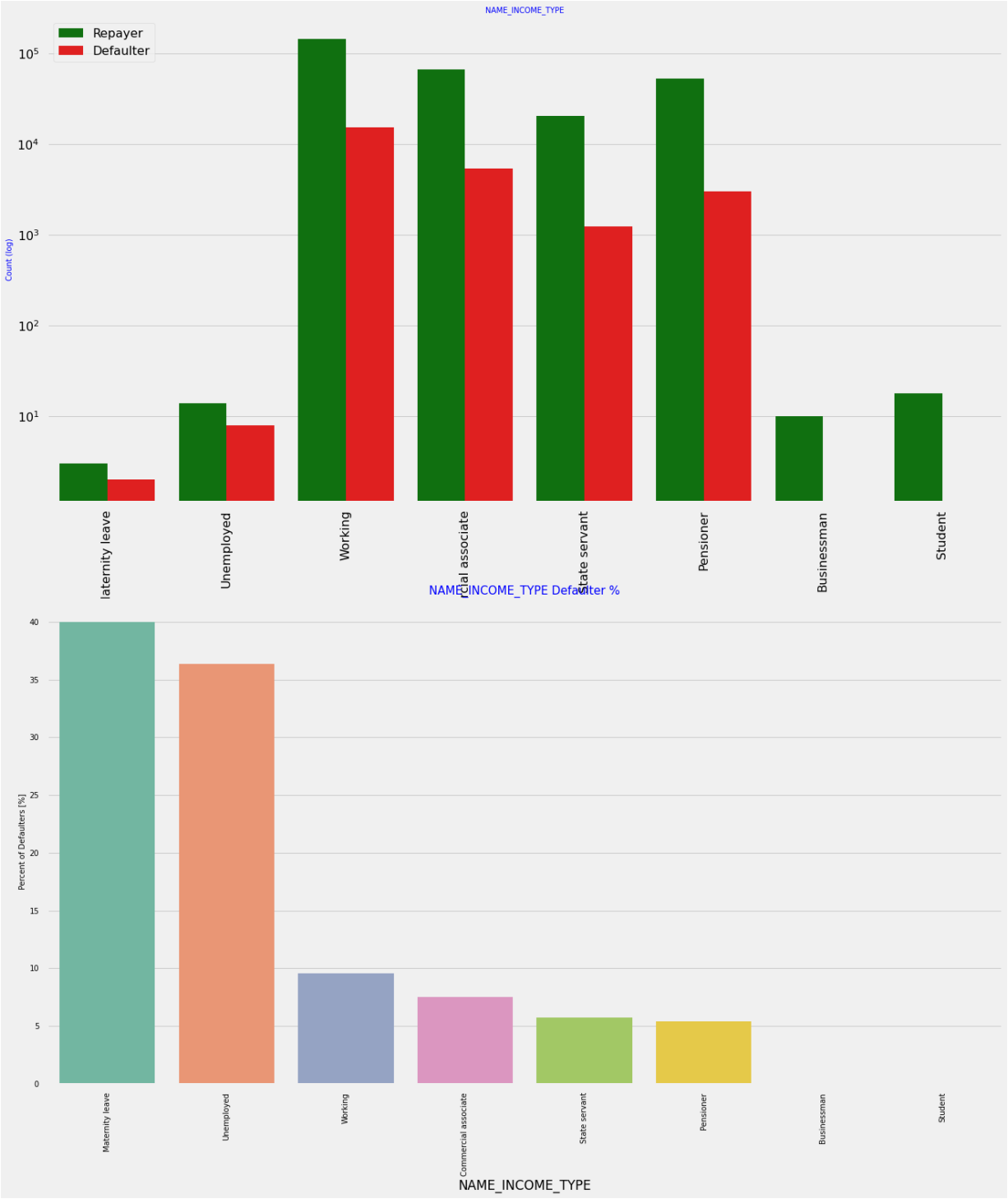
In [102…

```
# Analyzing Education Type based on loan repayment status
univariate_categorical("NAME_EDUCATION_TYPE",True,True,True)


#Inferences:
Majority of the clients have Secondary / secondary special education, followed by cl
education. Only a very small number having an academic degree
The Lower secondary category, although rare, have the largest rate of not returning
The people with Academic degree have less than 2% defaulting rate.
```



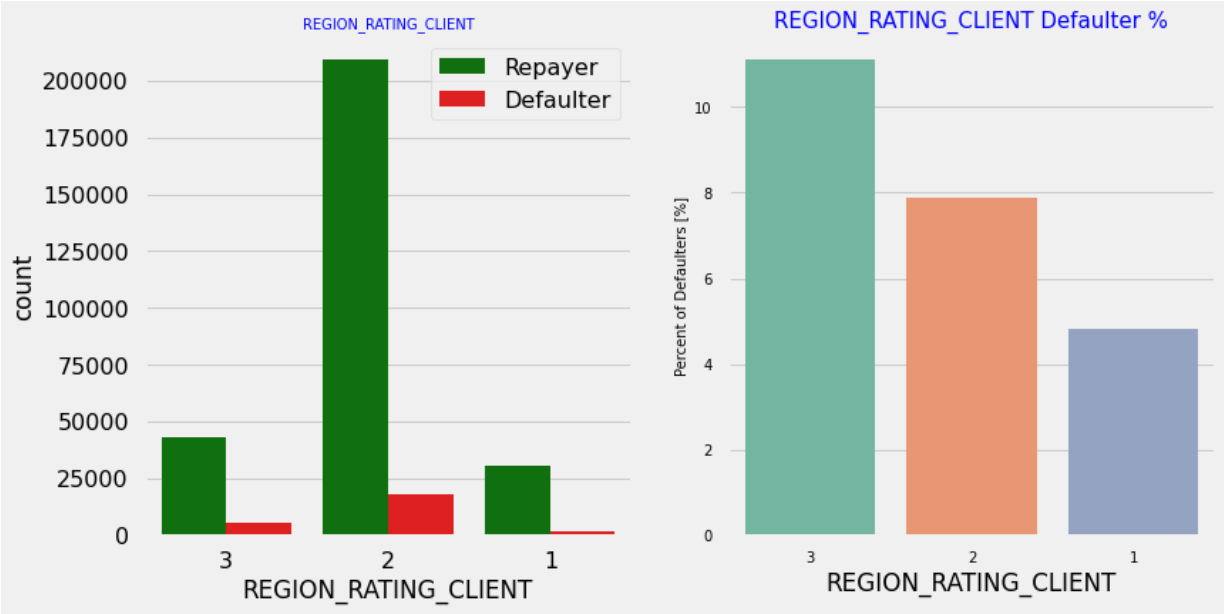
In [103…

```
# Analyzing Income Type based on loan repayment status
univariate_categorical("NAME_INCOME_TYPE",True,True,False)


#Inferences:
Most of applicants for loans have income type as Working, followed by Commercial ass
Pensioner and State servant.
The applicants with the type of income Maternity leave have almost 40% ratio of not
followed by Unemployed (37%).
The rest of types of incomes are under the average of 10% for not returning loans.
Student and Businessmen, though less in numbers do not have any default record.
Thus these two category are safest for providing loan.
```



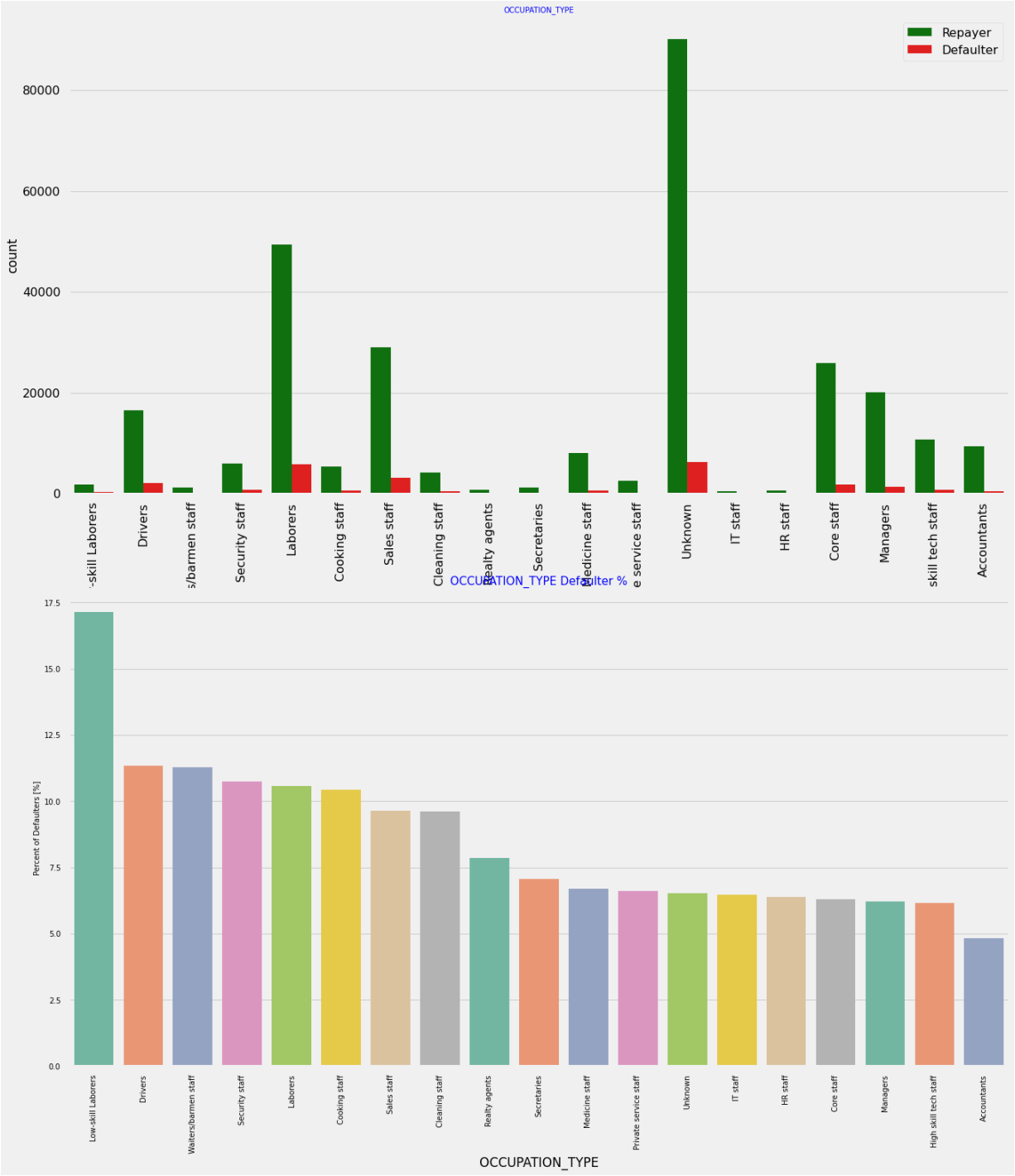
In [104…

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


#Inferences:
Most of the applicants are living in Region_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
```



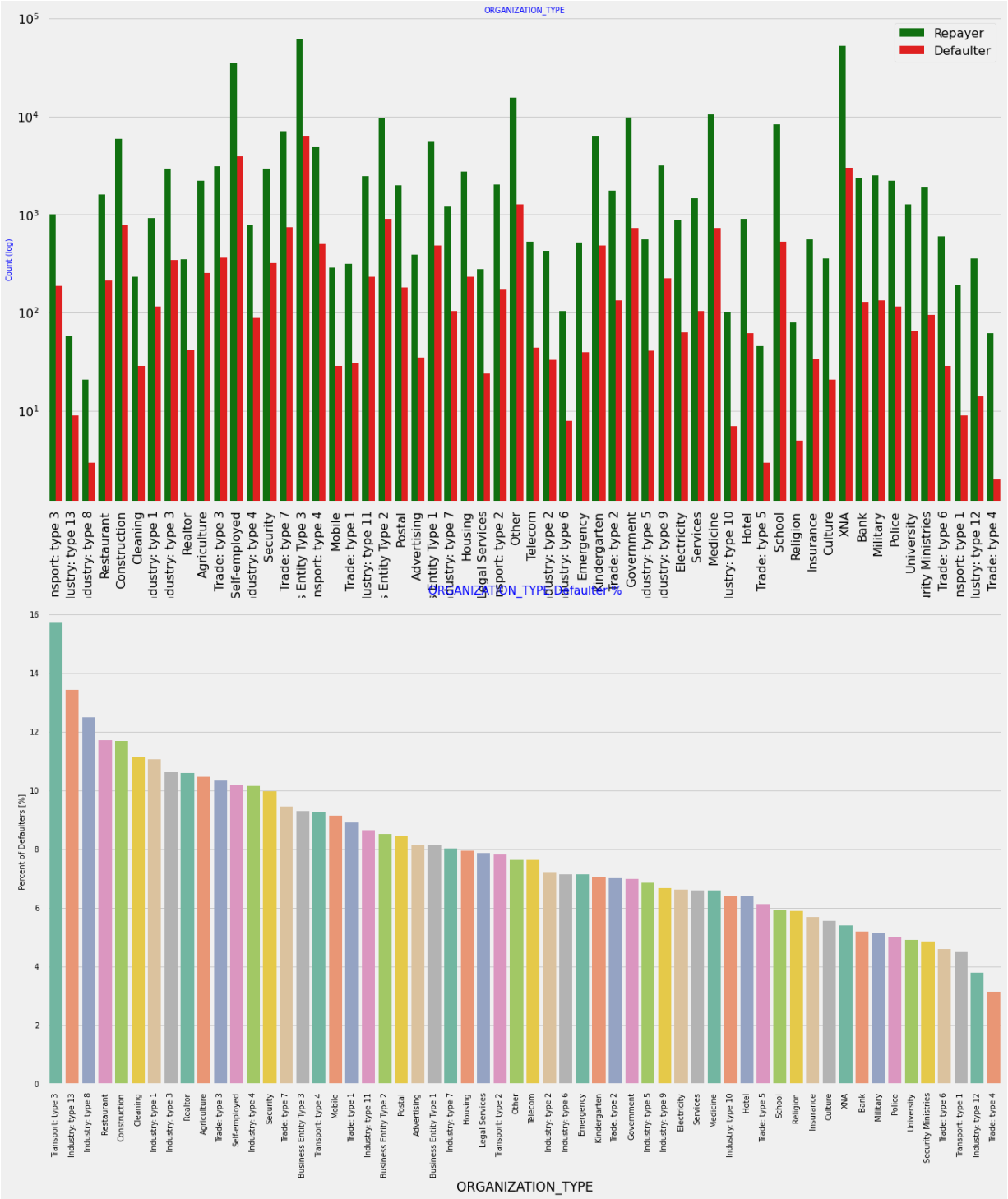
In [105…

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


#Inferences:
Most of the loans are taken by Laborers, followed by Sales staff.
IT staff take the lowest amount of loans.
The category with highest percent of not repaid loans are Low-skill Laborers (above
followed by Drivers and Waiters/barmen staff, Security staff,Laborers and Cooking st
```



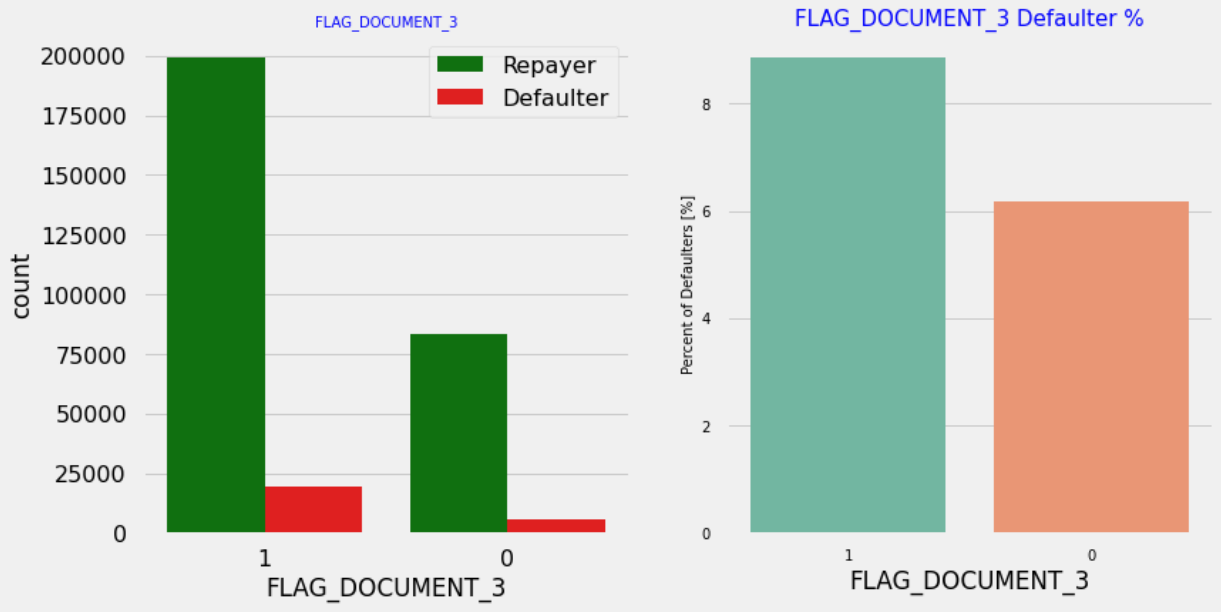
In [106…

```
# Checking Loan repayment status based on Organization type
univariate_categorical("ORGANIZATION_TYPE",True,True,False)


Inferences:
Organizations with highest percent of loans not repaid are Transport: type 3 (16%),
type 13 (13.5%), Industry: type 8 (12.5%) and Restaurant (less than 12%). Self emplo
relative high defaulting rate, and thus should be avoided to be approved for loan or
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
It can be seen that following category of organization type has lesser defaulters th
providing loans:
Trade Type 4 and 5
Industry type 8
```



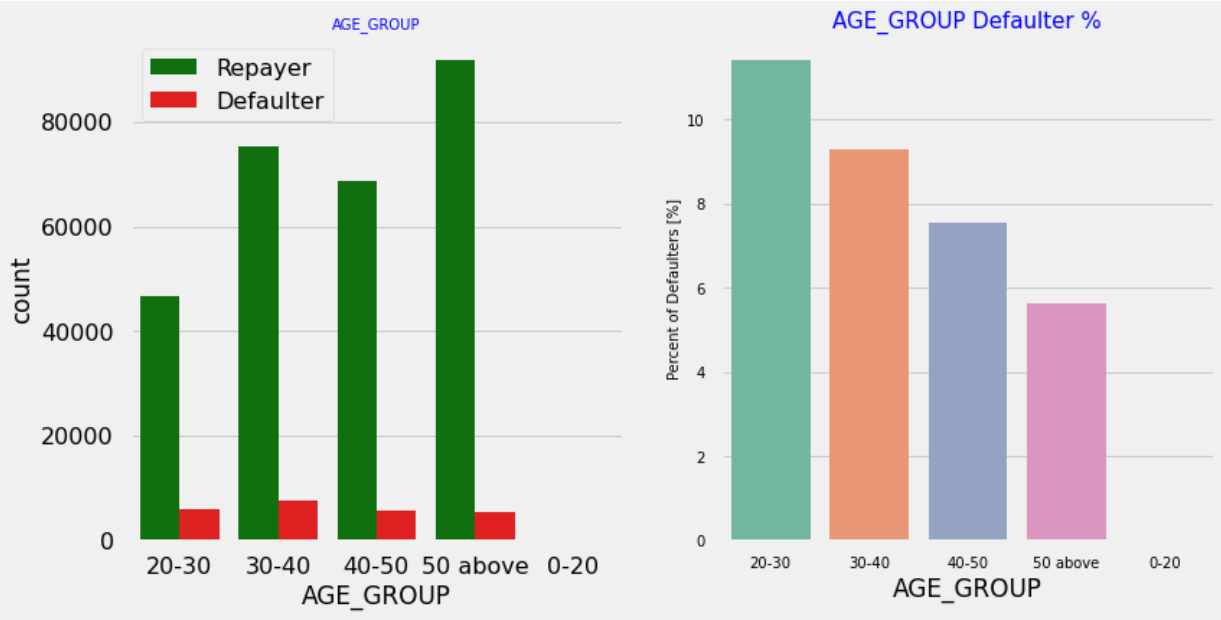
In [107…

```
# Analyzing Flag_Doc_3 submission status based on loan repayment status
univariate_categorical("FLAG_DOCUMENT_3",False,False,True)


#Inferences:
There is no significant correlation between repayers and defaulters in terms of subm
3 as we see even if applicants have submitted the document, they have defaulted a sl
than who have not submitted the document (6%)
```



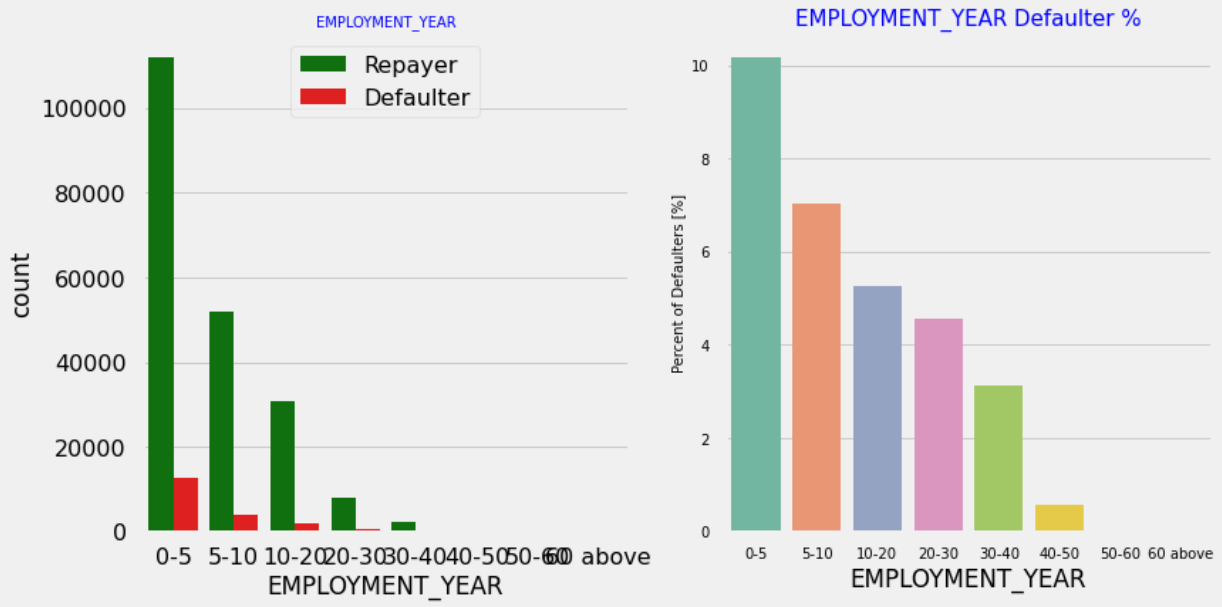
In [108…

```
# Analyzing Age Group based on loan repayment status
univariate_categorical("AGE_GROUP",False,False,True)

#Inferences:
People in the age group range 20-40 have higher probability of defaulting
People above age of 50 have low probability of defailting
```



In [109…

```
# Analyzing Employment_Year based on loan repayment status
univariate_categorical("EMPLOYMENT_YEAR",False,False,True)

#Inferences:
Majority of the applicants have been employeed in between 0-5 years. The defaulting
group is also the highest which is 10%
With increase of employment year, defaulting rate is gradually decreasing with peopl
experience having less than 1% default rate
```

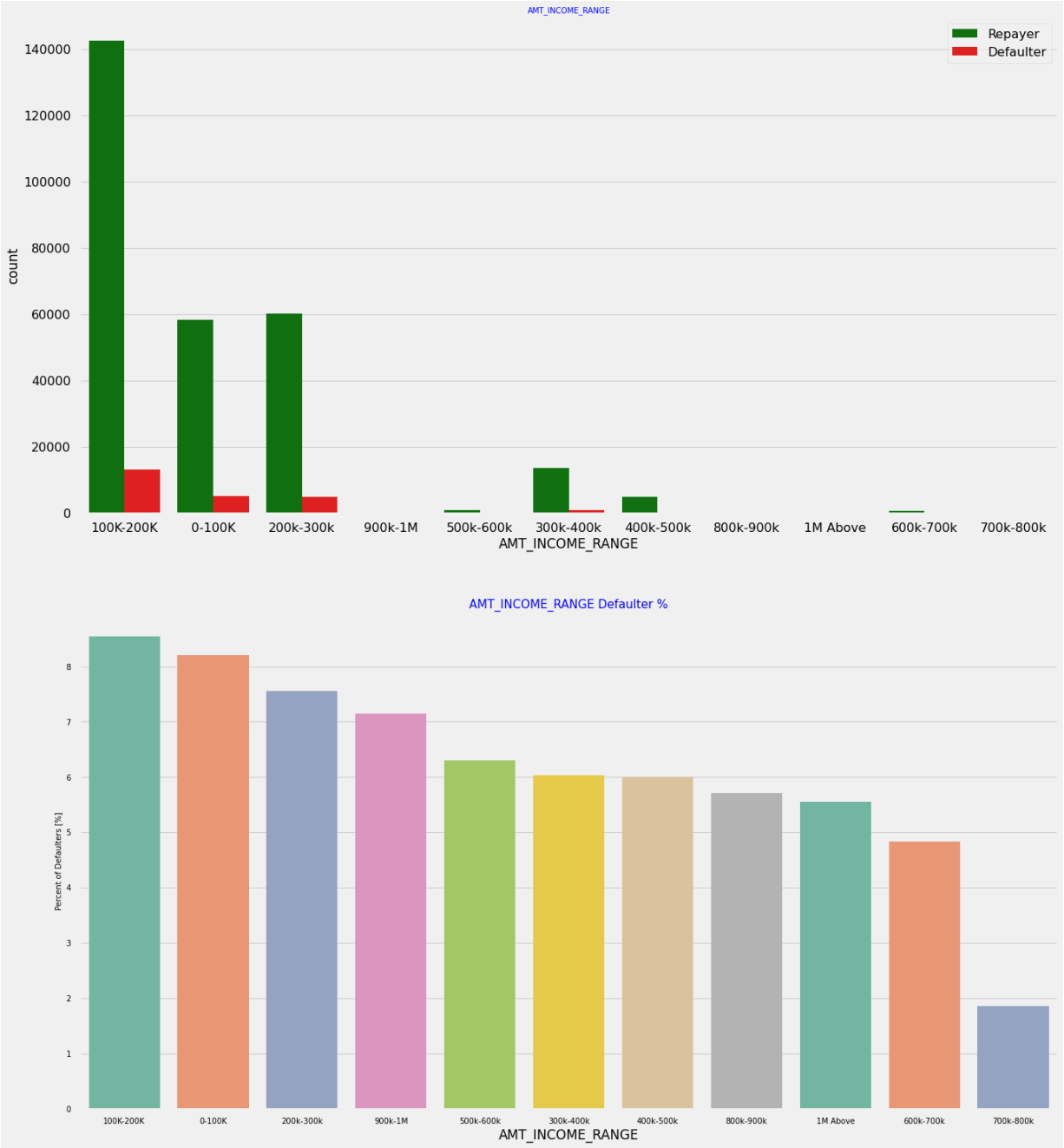


In [110…

```
# Analyzing Amount_Credit based on loan repayment status
univariate_categorical("AMT_CREDIT_RANGE",False,False,False)


#Inferences:
More than 80% of the loan provided are for amount less than 900,000
People who get loan for 300-600k tend to default more than others.
```

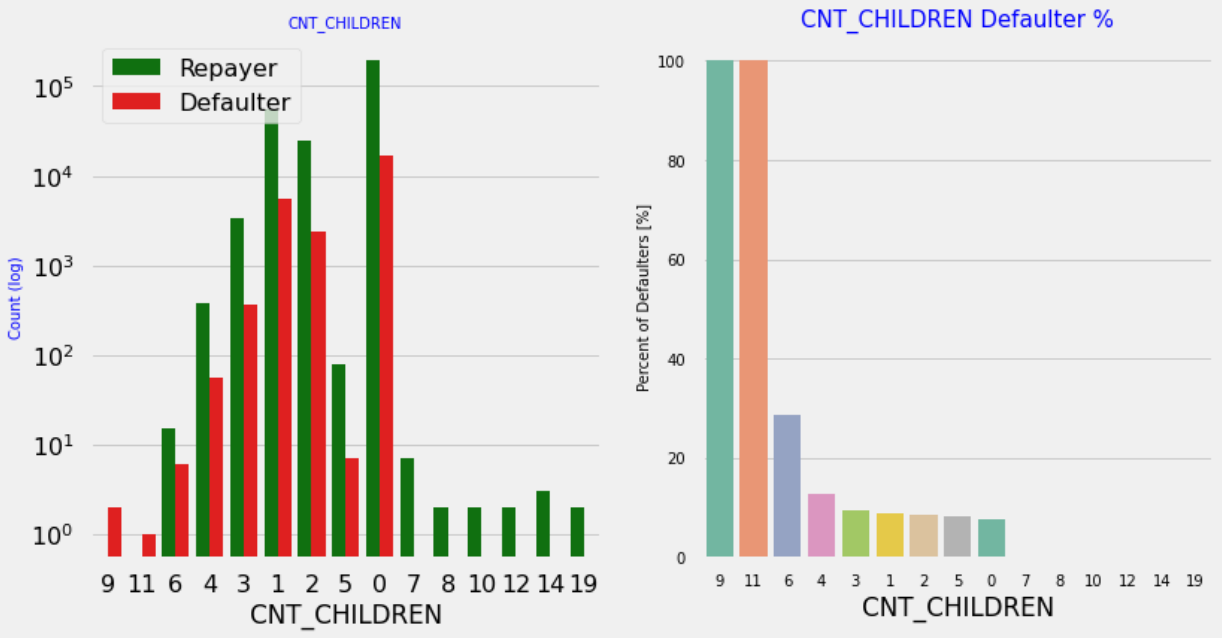
In [111…

```
# Analyzing Amount_Income Range based on loan repayment status
univariate_categorical("AMT_INCOME_RANGE",False,False,False)


#Inferences:
90% of the applications have Income total less than 300,000
Application with Income less than 300,000 has high probability of defaulting
Applicant with Income more than 700,000 are less likely to default
```



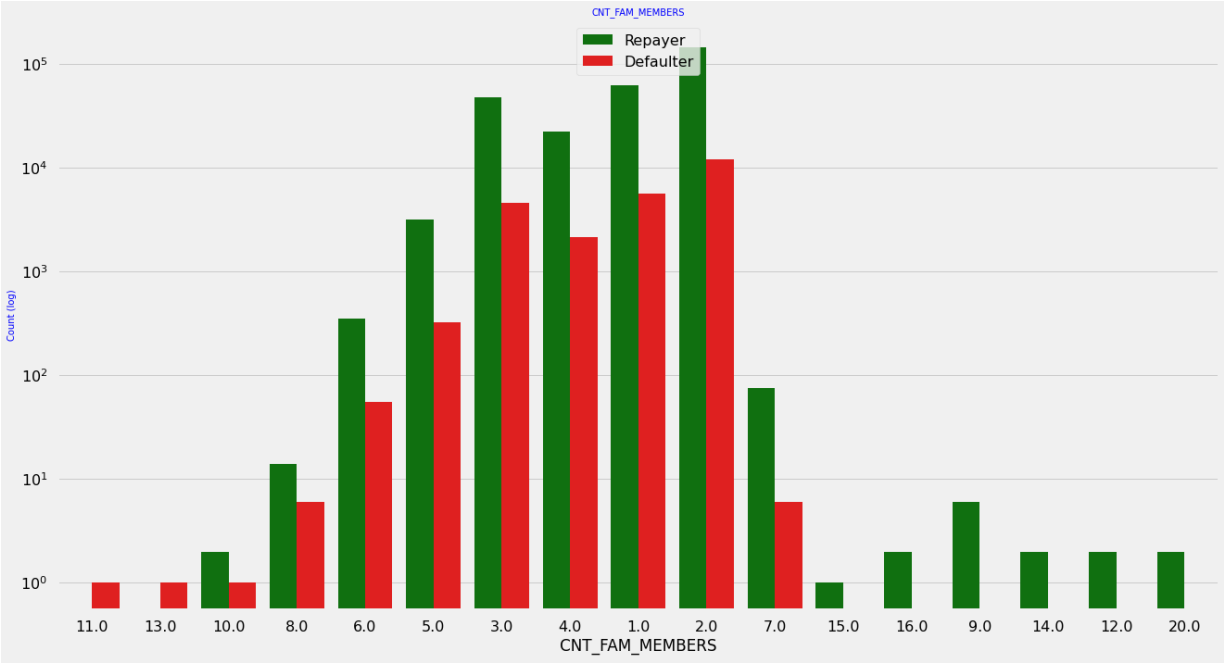
In [112…

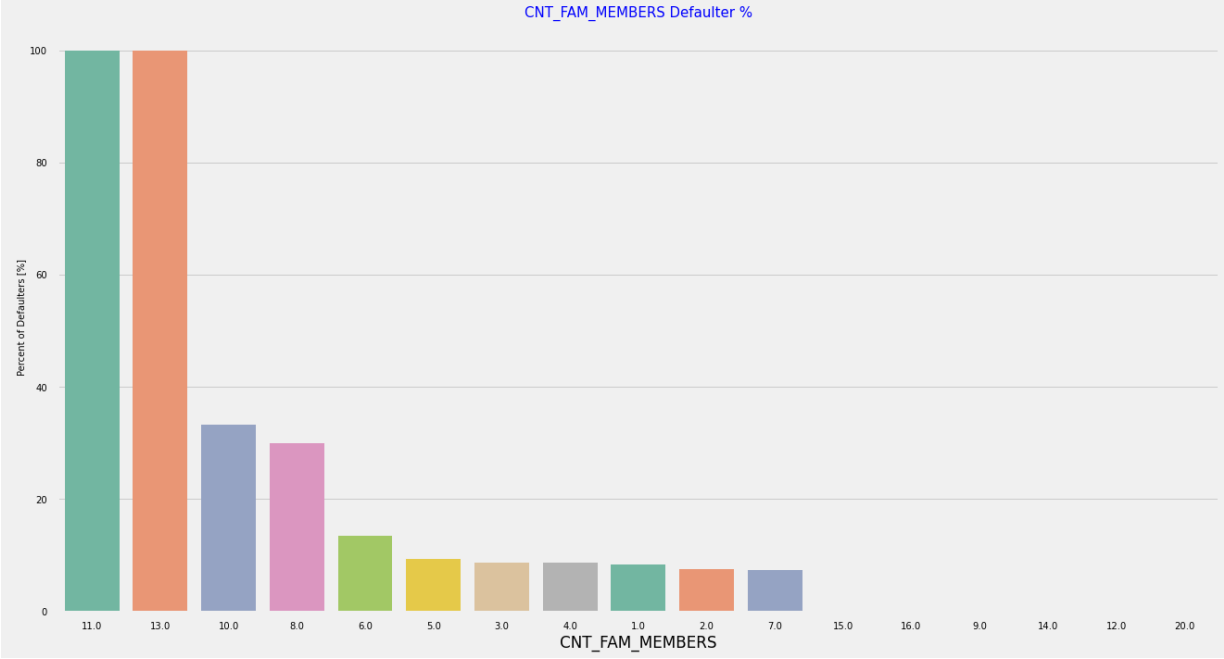
```
# Analyzing Number of children based on loan repayment status
univariate_categorical("CNT_CHILDREN",True)


#Inferences:
Most of the applicants do not have children
Very few clients have more than 3 children.
Client who have more than 4 children has a very high default rate with child count 9
showing 100% default rate
```



In [113…

```
# Analyzing Number of family members based on loan repayment status
univariate_categorical("CNT_FAM_MEMBERS",True, False, False


#Inferences:
Family member follows the same trend as children where having more family members in
of defaulting
```





In [114...

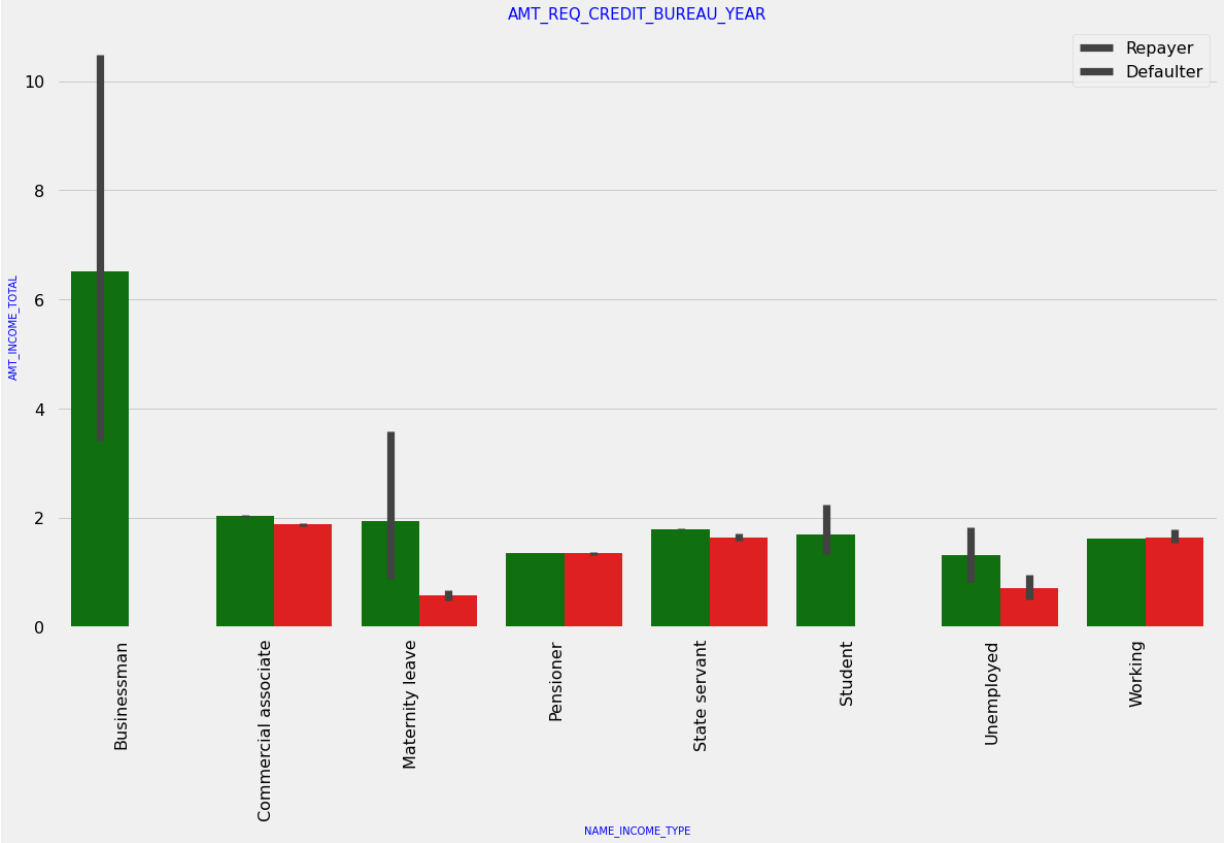
```
# 5.3.2 Categorical Bi/Multivariate Analysis
applicationDF.groupby('NAME_INCOME_TYPE')['AMT_INCOME_TOTAL'].describe()
```

Out[114...

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

In [115…

```
# Income type vs Income Amount Range
bivariate_bar("NAME_INCOME_TYPE","AMT_INCOME_TOTAL",applicationDF,"TARGET",(18,10))


Inferences:
It can be seen that business man's income is the highest and the estimated range wit
confidence level seem to indicate that the income of a business man could be in the
close to 4 lakhs and slightly above 10 lakhs
```



In [116…

```
# 5.4 Numeric Variables Analysis
# 5.4.1 Bifurcating the applicationDF dataframe based on Target value 0 and 1 for co
applicationDF.columns
```

Out[116…

```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR',
'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY',
'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NA
ME_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_PROC
ESS_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_NO
T_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'ORGANIZATION_TYPE', 'OBS_30_CNT_SOCIAL_CIR
CLE', 'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIR
CLE', 'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_3', 'AMT_REQ_CREDIT_BUREAU_HOUR', 'AM
T_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUR
EAU_YEAR', 'AMT_INCOME_RANGE', 'AMT_CREDIT_RANGE', 'AGE', 'AGE_GROUP', 'YEARS_EMPLOY
ED', 'EMPLOYMENT_YEAR'],
      dtype='object')
```

In [117…

```
# Bifurcating the applicationDF dataframe based on Target value 0 and 1 for correlat
cols_for_correlation = ['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_O
                        'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUI
                        'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE'
```

```
                           'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE', 'DAYS_BIR
                           'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'OCCUPATION_TYPE', '
                           'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_START',
                           'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
                           'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY', 'LIVE_CI
                           'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'DAY
                           'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', '
                           'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT', 'A


Repayer_df = applicationDF.loc[applicationDF['TARGET']==0, cols_for_correlation] # R
Defaulter_df = applicationDF.loc[applicationDF['TARGET']==1, cols_for_correlation] #
```

In [118…

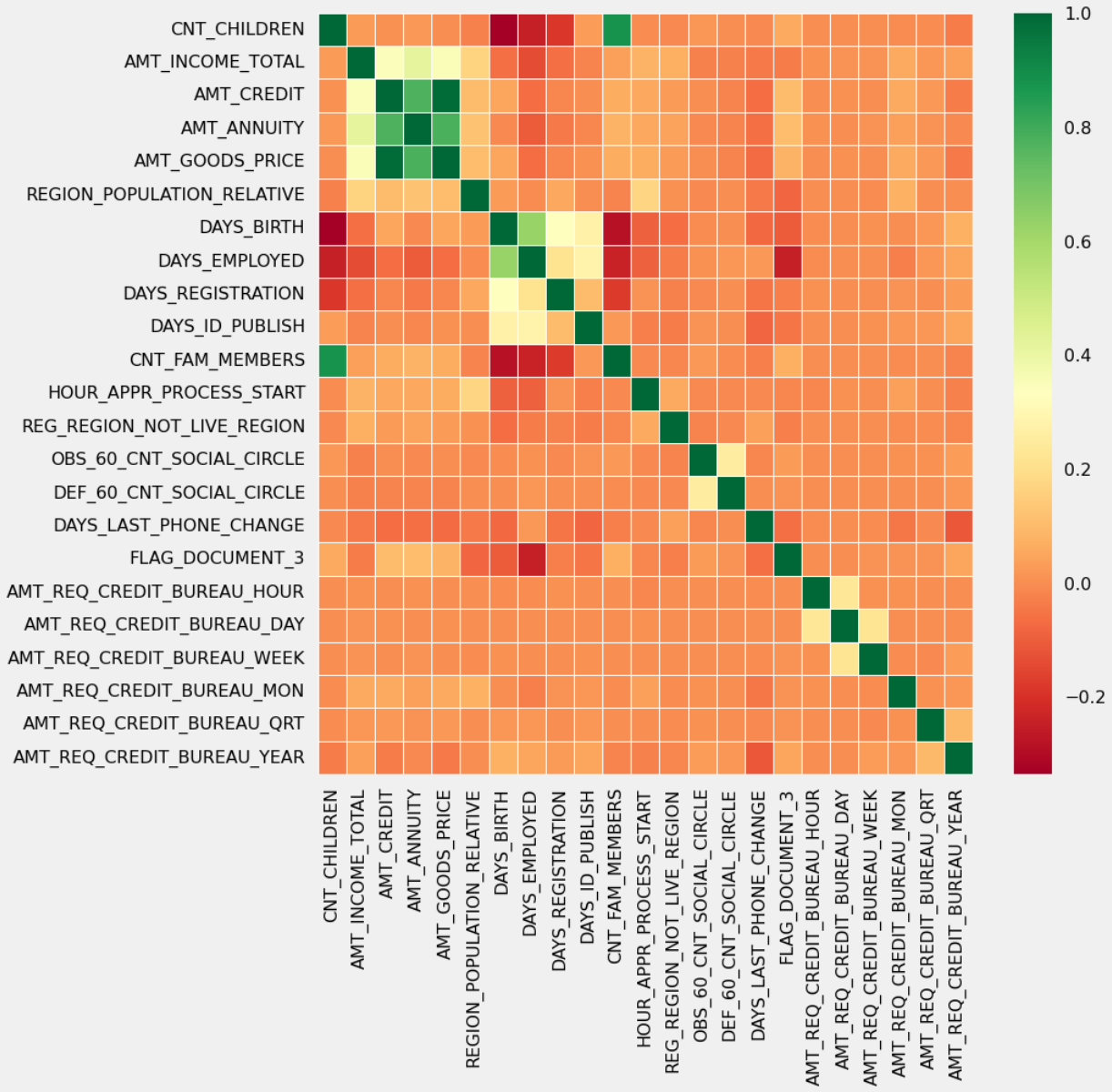
```
# 5.4.2 Correlation between numeric variable
# Getting the top 10 correlation for the Repayers data
corr_repayer = Repayer_df.corr()
corr_repayer = corr_repayer.where(np.triu(np.ones(corr_repayer.shape),k=1).astype(np
corr_df_repayer = corr_repayer.unstack().reset_index()
corr_df_repayer.columns =['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)
```

Out[118…

	VAR1	VAR2	Correlation
94	AMT_GOODS_PRICE	AMT_CREDIT	0.987250
230	CNT_FAM_MEMBERS	CNT_CHILDREN	0.878571
95	AMT_GOODS_PRICE	AMT_ANNUITY	0.776686
71	AMT_ANNUITY	AMT_CREDIT	0.771309
167	DAYS_EMPLOYED	DAYS_BIRTH	0.626114
70	AMT_ANNUITY	AMT_INCOME_TOTAL	0.418953
93	AMT_GOODS_PRICE	AMT_INCOME_TOTAL	0.349462
47	AMT_CREDIT	AMT_INCOME_TOTAL	0.342799
138	DAYS_BIRTH	CNT_CHILDREN	0.336966
190	DAYS_REGISTRATION	DAYS_BIRTH	0.333151

In [119…

```
fig = plt.figure(figsize=(12,12))
ax = sns.heatmap(Repayer_df.corr(), cmap="RdYlGn",annot=False,linewidth =1)


#Inferences:
Correlating factors amongst repayers:
Credit amount is highly correlated with
amount of goods price
loan annuity
total income
We can also see that repayers have high correlation in number of days employed.
```



In [120…

```
# 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).ast
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)
```

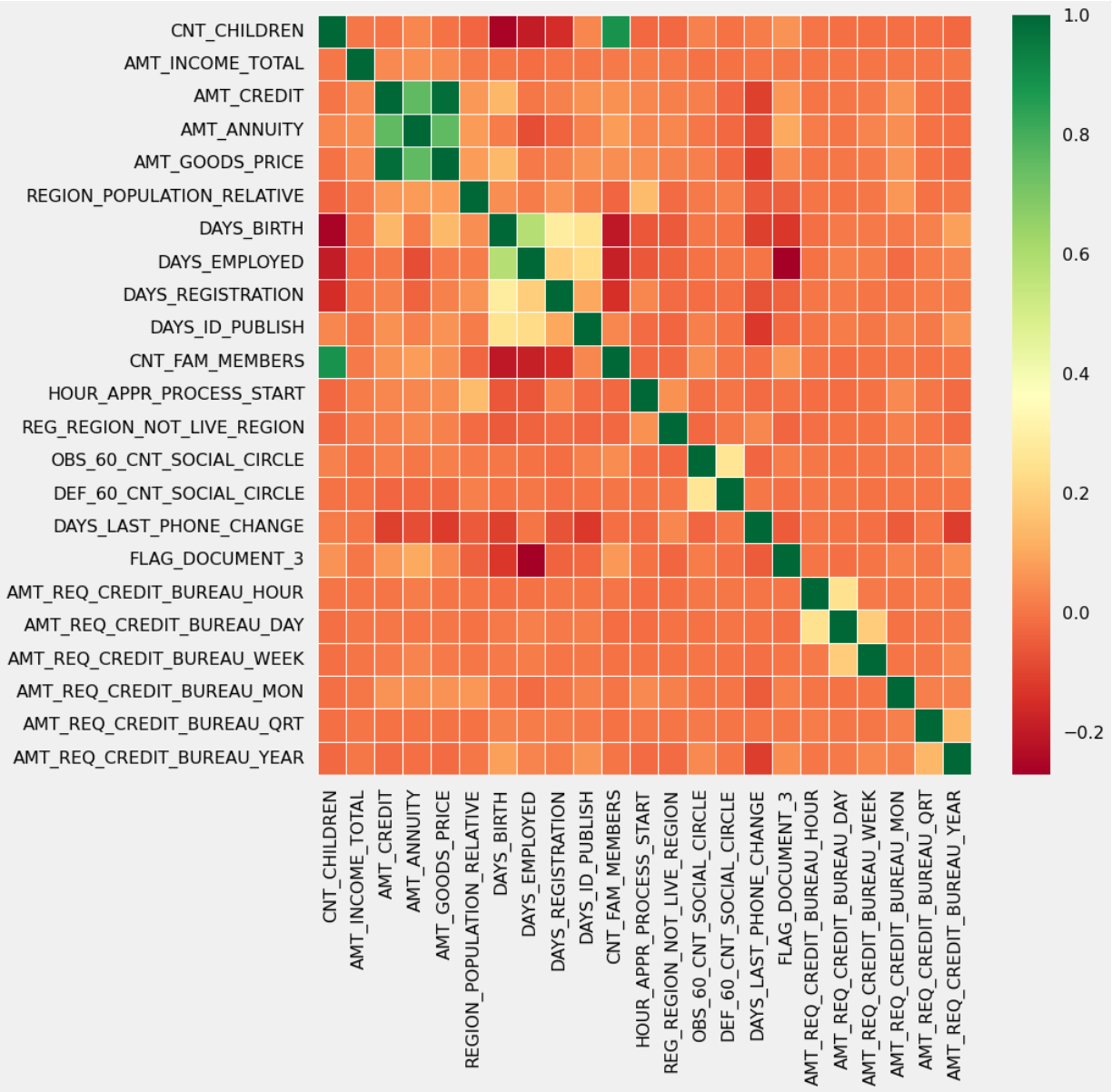
Out[120…

	VAR1	VAR2	Correlation
94	AMT_GOODS_PRICE	AMT_CREDIT	0.983103
230	CNT_FAM_MEMBERS	CNT_CHILDREN	0.885484
95	AMT_GOODS_PRICE	AMT_ANNUITY	0.752699
71	AMT_ANNUITY	AMT_CREDIT	0.752195
167	DAYS_EMPLOYED	DAYS_BIRTH	0.582185
190	DAYS_REGISTRATION	DAYS_BIRTH	0.289114
375	FLAG_DOCUMENT_3	DAYS_EMPLOYED	0.272169

	VAR1	VAR2	Correlation
335	DEF_60_CNT_SOCIAL_CIRCLE	OBS_60_CNT_SOCIAL_CIRCLE	0.264159
138	DAYS_BIRTH	CNT_CHILDREN	0.259109
213	DAYS_ID_PUBLISH	DAYS_BIRTH	0.252863

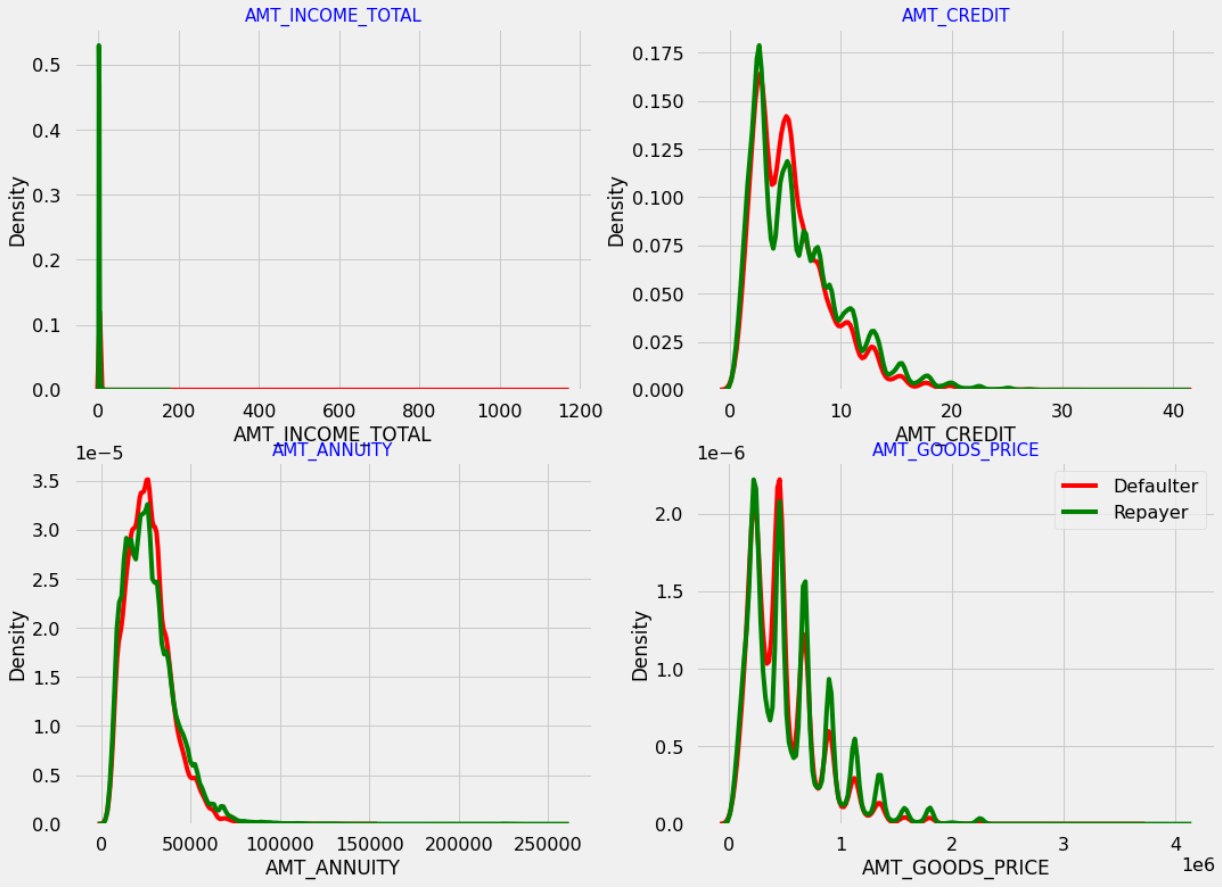
In [121…

```
fig = plt.figure(figsize=(12,12))
ax = sns.heatmap(Defaulter_df.corr(), cmap="RdYlGn",annot=False,linewidth =1)


#Inferences:
Credit amount is highly correlated with amount of goods price which is same as repay
But the loan annuity correlation with credit amount has slightly reduced in defaulte
when compared to repayers(0.77)
We can also see that repayers have high correlation in number of days employed(0.62)
to defaulters(0.58).
There is a severe drop in the correlation between total income of the client and the
amount(0.038) amongst defaulters whereas it is 0.342 among repayers.
Days_birth and number of children correlation has reduced to 0.259 in defaulters whe
compared to 0.337 in repayers.
There is a slight increase in defaulted to observed count in social circle among def
when compared to repayers(0.254)
```



In [122…

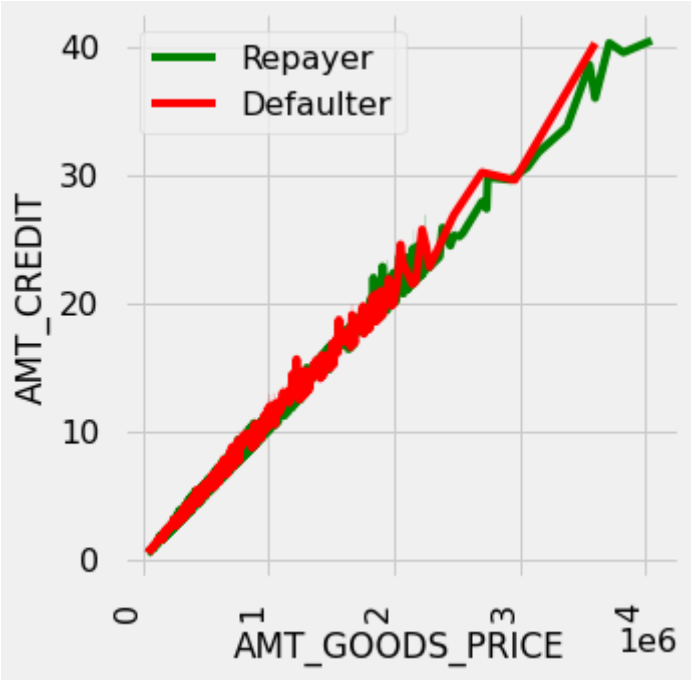
```
# 5.4.3 Numerical Univariate Analysis
# Plotting the numerical columns related to amount as distribution plot to see densi
amount = applicationDF[[ 'AMT_INCOME_TOTAL','AMT_CREDIT','AMT_ANNUITY', 'AMT_GOODS_P

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, color='r',label ="Defaulter")
    sns.distplot(Repayer_df[i[1]], hist=False, color='g', label ="Repayer")
    plt.title(i[1], fontdict={'fontsize' : 15, 'fontweight' : 5, 'color' : 'Blue'})

plt.legend()

plt.show()


#Inferences:
Most no of loans are given for goods price below 10 lakhs
Most people pay annuity below 50000 for the credit loan
Credit amount of the loan is mostly less then 10 lakhs
The repayers and defaulters distribution overlap in all the plots and hence we canno
variables in isolation to make a decision
```



In [123…

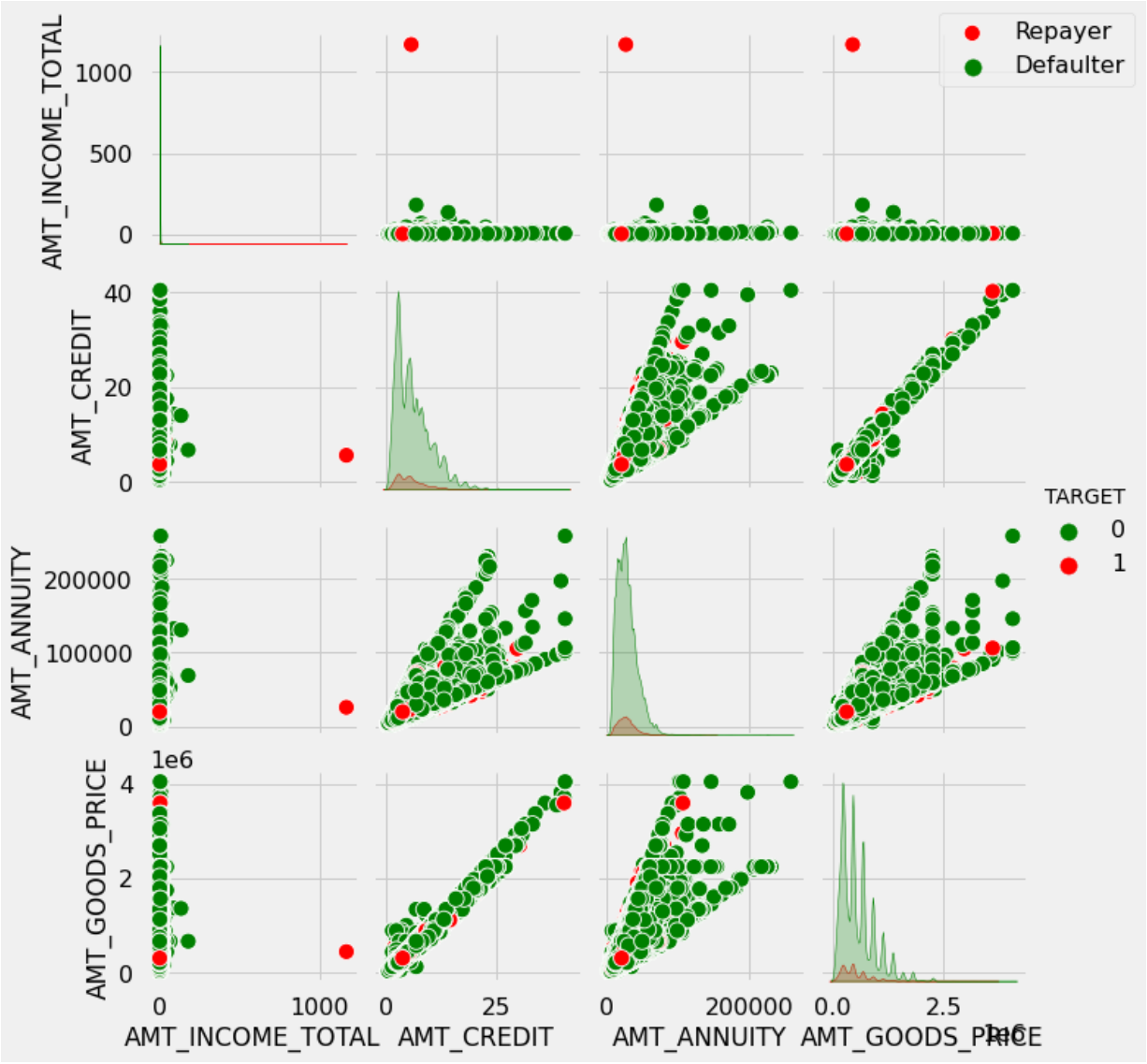
```
# 5.4.4 Numerical Bivariate Analysis
# Checking the relationship between Goods price and credit and comparing with loan r
bivariate_rel('AMT_GOODS_PRICE','AMT_CREDIT',applicationDF,"TARGET", "line", ['g','r

#Inferences:
When the credit amount goes beyond 3M, there is an increase in defaulters.
```

```
<Figure size 1080x432 with 0 Axes>
```



In [124…

```
# Plotting pairplot between amount variable to draw reference against loan repayment
amount = applicationDF[[ 'AMT_INCOME_TOTAL','AMT_CREDIT',
                         'AMT_ANNUITY', 'AMT_GOODS_PRICE','TARGET']]
amount = amount[(amount["AMT_GOODS_PRICE"].notnull()) & (amount["AMT_ANNUITY"].notnu
ax= sns.pairplot(amount,hue="TARGET",palette=["g","r"])
ax.fig.legend(labels=['Repayer','Defaulter'])
plt.show()


#Inferences:
When amt_annuity >15000 amt_goods_price> 3M, there is a lesser chance of defaulters
AMT_CREDIT and AMT_GOODS_PRICE are highly correlated as based on the scatterplot whe
data are consolidated in form of a line
There are very less defaulters for AMT_CREDIT >3M
Inferences related to distribution plot has been already mentioned in previous distp
inferences section
```



In [125…

```
# 6. Merged Dataframes Analysis
#merge both the dataframe on SK_ID_CURR with Inner Joins
loan_process_df = pd.merge(applicationDF, previousDF, how='inner', on='SK_ID_CURR')
loan_process_df.head()
```

Out[125…

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_RE…
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100003	0	Cash loans	F	N	
3	100003	0	Cash loans	F	N	
4	100004	0	Revolving loans	M	Y	

In [126…

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

Out[126…

```
(1413701, 74)
```

In [127…

```
# Checking the element count of the dataframe
loan_process_df.size
```

Out[127…

```
104613874
```

In [128…

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1413701 entries, 0 to 1413700
Data columns (total 74 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_CAR                 1413701 non-null  category
 5   FLAG_OWN_REALTY              1413701 non-null  category
 6   CNT_CHILDREN                 1413701 non-null  int64
 7   AMT_INCOME_TOTAL             1413701 non-null  float64
 8   AMT_CREDIT_x                 1413701 non-null  float64
 9   AMT_ANNUITY_x                1413608 non-null  float64
 10  AMT_GOODS_PRICE_x            1412493 non-null  float64
 11  NAME_TYPE_SUITE              1413701 non-null  category
 12  NAME_INCOME_TYPE             1413701 non-null  category
 13  NAME_EDUCATION_TYPE          1413701 non-null  category
 14  NAME_FAMILY_STATUS           1413701 non-null  category
 15  NAME_HOUSING_TYPE            1413701 non-null  category
 16  REGION_POPULATION_RELATIVE   1413701 non-null  float64
 17  DAYS_BIRTH                   1413701 non-null  int64
 18  DAYS_EMPLOYED                1413701 non-null  int64
 19  DAYS_REGISTRATION            1413701 non-null  float64
 20  DAYS_ID_PUBLISH              1413701 non-null  int64
 21  OCCUPATION_TYPE              1413701 non-null  category
 22  CNT_FAM_MEMBERS              1413701 non-null  float64
 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  AGE                          1413701 non-null  int64
 49  AGE_GROUP                    1413701 non-null  category
 50  YEARS_EMPLOYED               1413701 non-null  int64
 51  EMPLOYMENT_YEAR              1032756 non-null  category
 52  SK_ID_PREV                   1413701 non-null  int64
 53  NAME_CONTRACT_TYPE_y         1413701 non-null  category
 54  AMT_ANNUITY_y                1413701 non-null  float64
 55  AMT_APPLICATION              1413701 non-null  float64
 56  AMT_CREDIT_y                 1413700 non-null  float64
 57  AMT_GOODS_PRICE_y            1413701 non-null  float64
 58  NAME_CASH_LOAN_PURPOSE       1413701 non-null  category
 59  NAME_CONTRACT_STATUS         1413701 non-null  category
 60  DAYS_DECISION                1413701 non-null  int64
 61  NAME_PAYMENT_TYPE            1413701 non-null  category
 62  CODE_REJECT_REASON           1413701 non-null  category
 63  NAME_CLIENT_TYPE             1413701 non-null  category
 64  NAME_GOODS_CATEGORY          1413701 non-null  category
 65  NAME_PORTFOLIO               1413701 non-null  category
 66  NAME_PRODUCT_TYPE            1413701 non-null  category
 67  CHANNEL_TYPE                 1413701 non-null  category
 68  SELLERPLACE_AREA             1413701 non-null  int64
 69  NAME_SELLER_INDUSTRY         1413701 non-null  category
 70  CNT_PAYMENT                  1413701 non-null  float64
 71  NAME_YIELD_GROUP             1413701 non-null  category
 72  PRODUCT_COMBINATION          1413388 non-null  category
 73  DAYS_DECISION_GROUP          1413701 non-null  category
dtypes: category(37), float64(23), int64(14)
memory usage: 459.8 MB
```

In [129…

```
# Checking merged dataframe numerical columns statistics
loan_process_df.describe()
```

Out[129…

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANN
count	1.413701e+06	1.413701e+06	1.413701e+06	1.413701e+06	1.413701e+06	1.4136
mean	2.784813e+05	8.655296e-02	4.048933e-01	1.733160e+00	5.875537e+00	2.7017
std	1.028118e+05	2.811789e-01	7.173454e-01	1.985734e+00	3.849173e+00	1.3951
min	1.000020e+05	0.000000e+00	0.000000e+00	2.565000e-01	4.500000e-01	1.6155
25%	1.893640e+05	0.000000e+00	0.000000e+00	1.125000e+00	2.700000e+00	1.6821
50%	2.789920e+05	0.000000e+00	0.000000e+00	1.575000e+00	5.084955e+00	2.4925
75%	3.675560e+05	0.000000e+00	1.000000e+00	2.070000e+00	8.079840e+00	3.4542
max	4.562550e+05	1.000000e+00	1.900000e+01	1.170000e+03	4.050000e+01	2.2500

In [130…

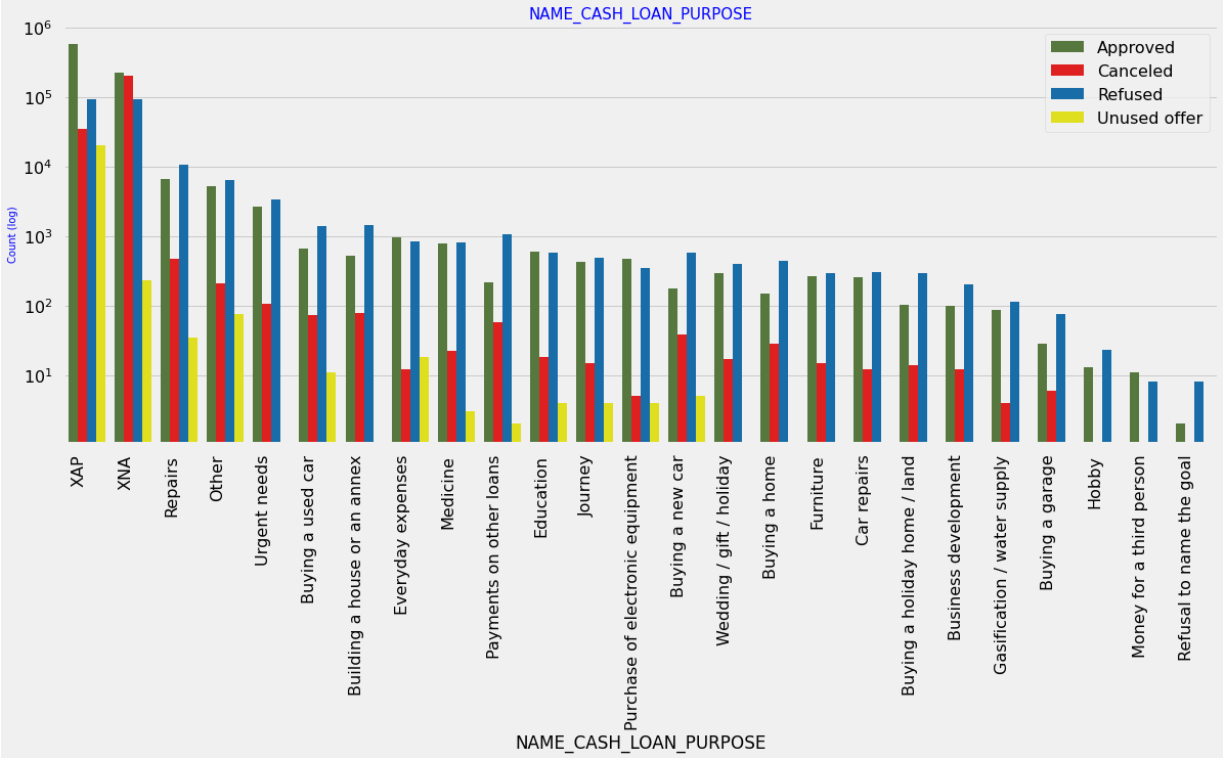
```
# Bifurcating the applicationDF dataframe based on Target value 0 and 1 for correlat

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

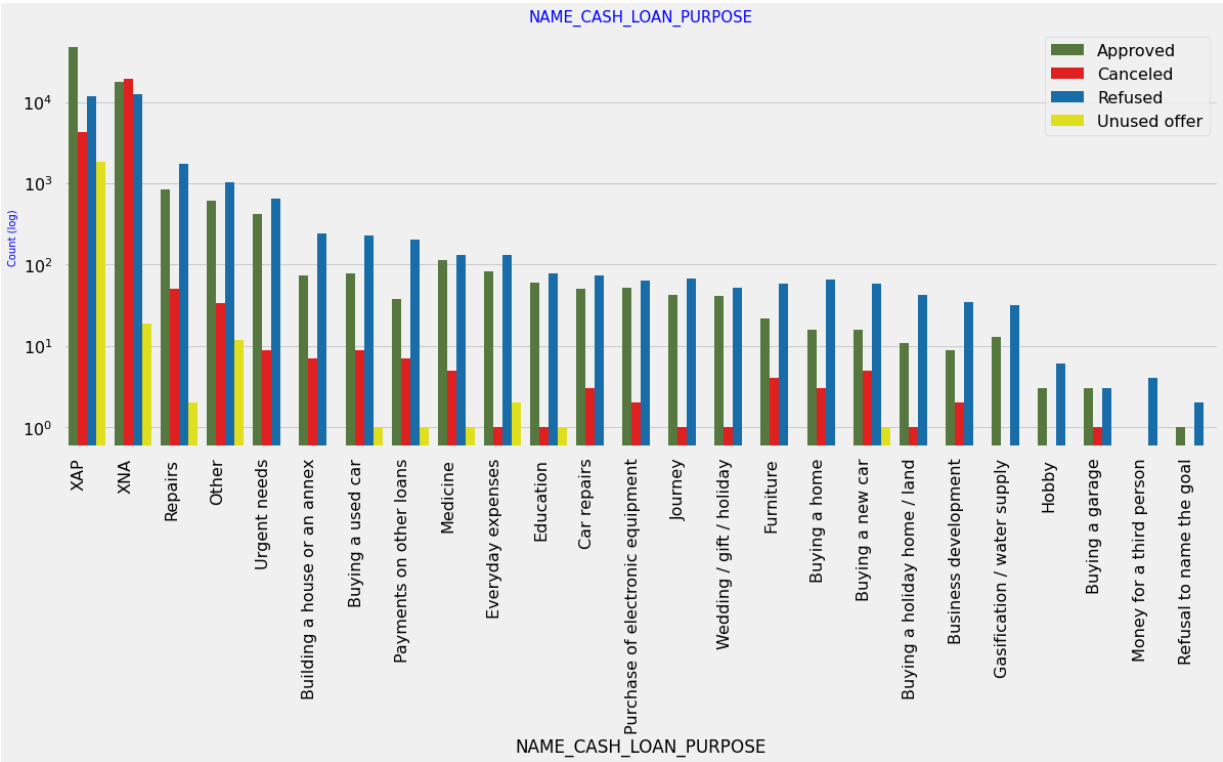
In [131…

```
#Plotting Contract Status vs purpose of the loan:

univariate_merged("NAME_CASH_LOAN_PURPOSE",L0,"NAME_CONTRACT_STATUS",["#548235","#FF
```

```
univariate_merged("NAME_CASH_LOAN_PURPOSE",L1,"NAME_CONTRACT_STATUS",["#548235","#FF
```

```
#Inferences:
Loan purpose has high number of unknown values (XAP, XNA)
Loan taken for the purpose of Repairs seems to have highest default rate
A very high number application have been rejected by bank or refused by client which
repair or other. This shows that purpose repair is taken as high risk by bank and ei
rejected or bank offers very high loan interest rate which is not feasible by the cl
thus they refuse the loan.
```

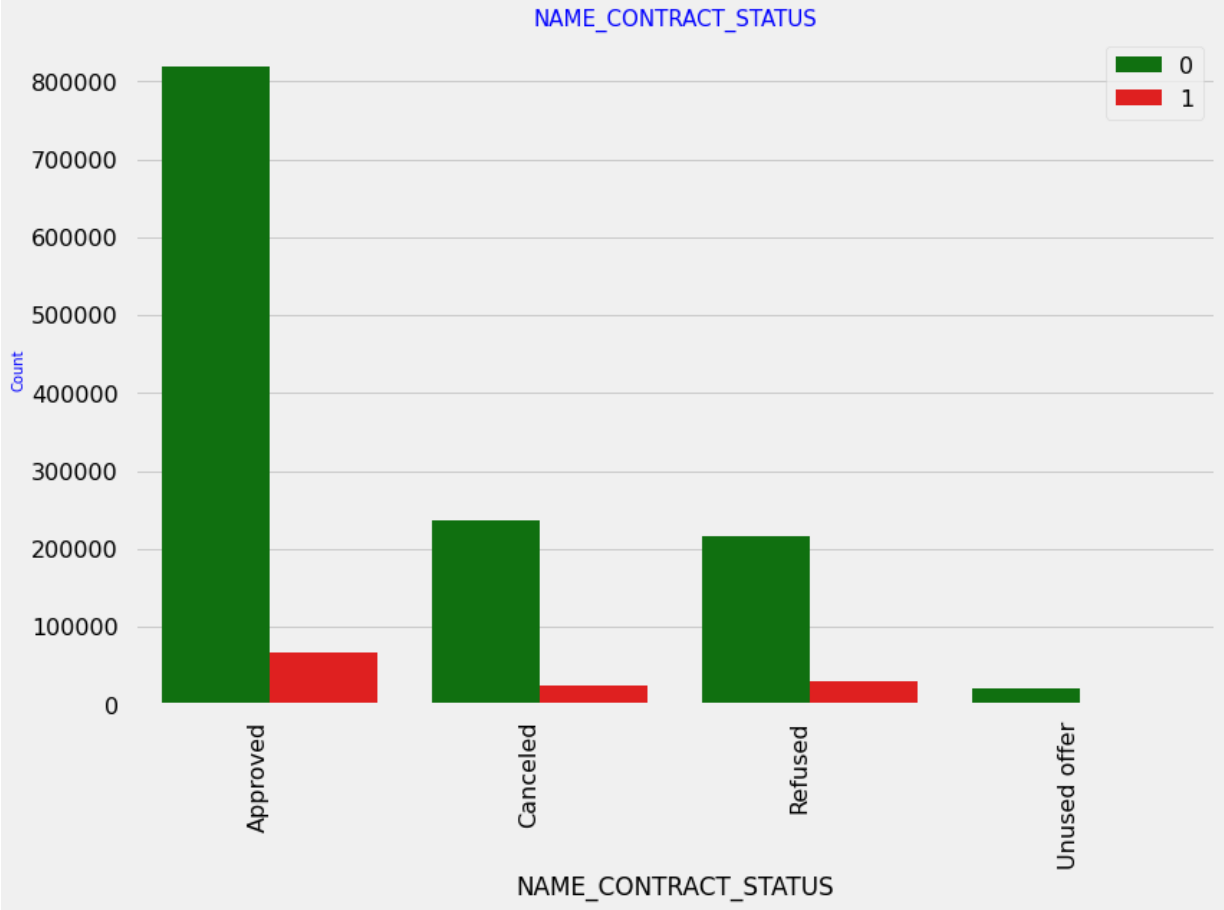




In [132…

```
# Checking the Contract Status based on loan repayment status and whether there is a
loss or financial loss
univariate_merged("NAME_CONTRACT_STATUS",loan_process_df,"TARGET",['g','r'],False,(1
```

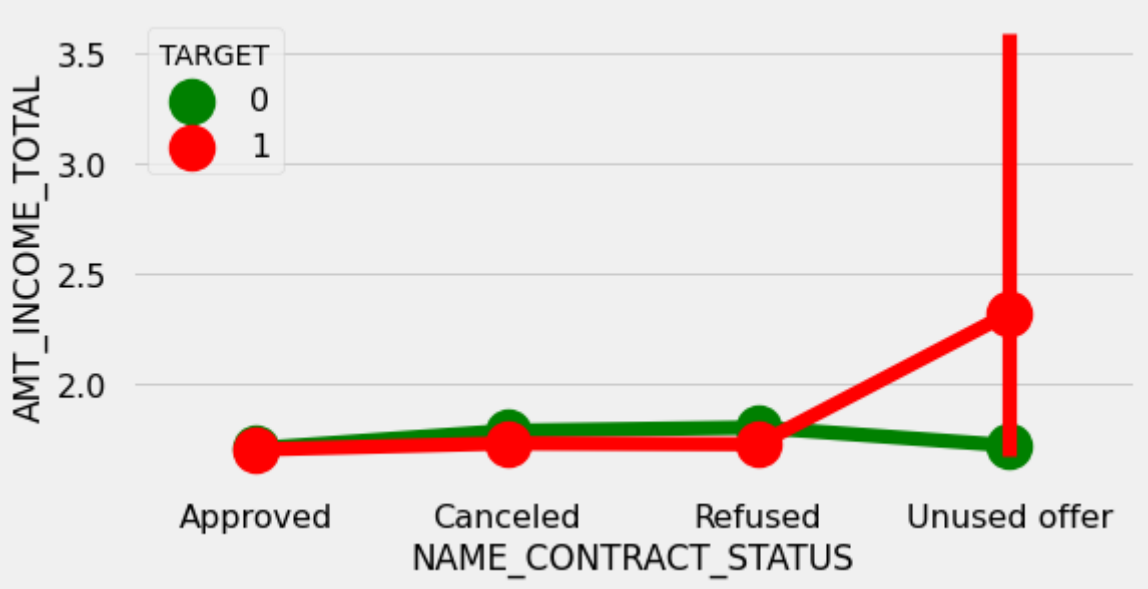
```
g = loan_process_df.groupby("NAME_CONTRACT_STATUS")["TARGET"]
df1 = pd.concat([g.value_counts(),round(g.value_counts(normalize=True).mul(100),2)],
                keys=('Counts','Percentage'))
df1['Percentage'] = df1['Percentage'].astype(str) +"%" # adding percentage symbol
in the results for understanding
print (df1)



#Inferences:
90% of the previously cancelled client have actually repayed the loan. Revisiting th
would increase business opoortunity for these clients
88% of the clients who have been previously refused a loan has payed back the loan i
Refual reason should be recorded for further analysis as these clients would turn in
repaying customer.
```



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

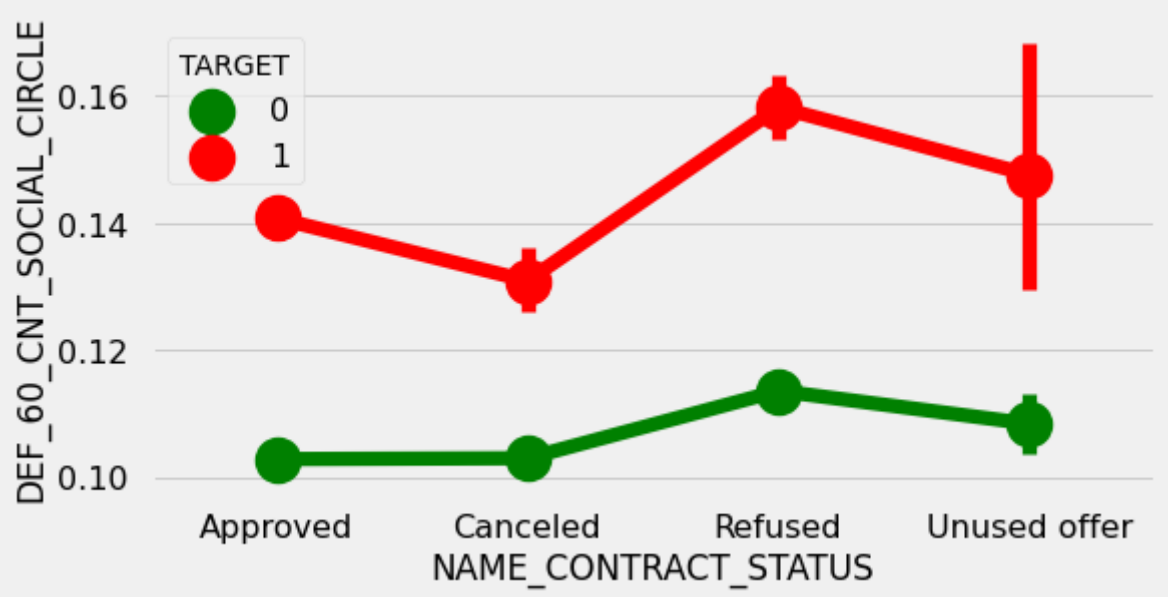
In [133…

```
# plotting the relationship between income total and contact status
merged_pointplot("NAME_CONTRACT_STATUS",'AMT_INCOME_TOTAL')



#Inferences:
The point plot show that the people who have not used offer earlier have defaulted e
average income is higher than others
```



In [134…

```
# plotting the relationship between people who defaulted in last 60 days being in cl
merged_pointplot("NAME_CONTRACT_STATUS",'DEF_60_CNT_SOCIAL_CIRCLE')

#Inferences:
Clients who have average of 0.13 or higher DEF_60_CNT_SOCIAL_CIRCLE score tend to de
hence client's social circle has to be analysed before providing the loan.
```



In [ ]: