In [3]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as st
import os
%matplotlib inline
```

In [4]:

```

sns.set(style="darkgrid")
import warnings
warnings.filterwarnings('ignore')
```

In [5]:

```
current_directory =os.getcwd()
current_directory

```

Out[5]:

```
'/Users/myyntiimac'
```

In [6]:

```
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 [7]:

```
df = pd.read_csv("/Users/myyntiimac/Desktop/application_data.csv")
df.head()
```

Out[7]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TC
0	100002	1	Cash loans	M	N	Y	0	202
1	100003	0	Cash loans	F	N	N	0	270
2	100004	0	Revolving loans	M	Y	Y	0	67
3	100006	0	Cash loans	F	N	Y	0	135
4	100007	0	Cash loans	M	N	Y	0	121

In [6]:

```
df.shape
```

Out[6]:

```
(307511, 122)
```

In [7]:

```
df.columns
```

Out[7]:

```
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',
       ...
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_B
UREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON', 'AM
T_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR'], dtype='object', length=122)
```

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
```

In [9]:

```
#Check the null containing column
df.isnull().any()
```

```
NONLIVINGAREA_MODE              True
APARTMENTS_MEDI                 True
BASEMENTAREA_MEDI               True
YEARS_BEGINEXPLUATATION_MEDI    True
YEARS_BUILD_MEDI                True
COMMONAREA_MEDI                 True
ELEVATORS_MEDI                  True
ENTRANCES_MEDI                  True
FLOORSMAX_MEDI                  True
FLOORSMIN_MEDI                  True
LANDAREA_MEDI                   True
LIVINGAPARTMENTS_MEDI           True
LIVINGAREA_MEDI                 True
NONLIVINGAPARTMENTS_MEDI        True
NONLIVINGAREA_MEDI              True
FONDKAPREMONT_MODE              True
HOUSETYPE_MODE                  True
TOTALAREA_MODE                  True
WALLSMATERIAL_MODE              True
```

Insight:several column containing null values, check the percentage of null

Data cleaning and manipulation

In [10]:

```
# check the percentage of null in whole dataframe
null_percentages = df.isnull().mean() * 100
null_percentages
```

```
FLAG_DOCUMENT_8                0.000000
FLAG_DOCUMENT_9                0.000000
FLAG_DOCUMENT_10               0.000000
FLAG_DOCUMENT_11               0.000000
FLAG_DOCUMENT_12               0.000000
FLAG_DOCUMENT_13               0.000000
FLAG_DOCUMENT_14               0.000000
FLAG_DOCUMENT_15               0.000000
FLAG_DOCUMENT_16               0.000000
FLAG_DOCUMENT_17               0.000000
FLAG_DOCUMENT_18               0.000000
FLAG_DOCUMENT_19               0.000000
FLAG_DOCUMENT_20               0.000000
FLAG_DOCUMENT_21               0.000000
AMT_REQ_CREDIT_BUREAU_HOUR    13.501631
AMT_REQ_CREDIT_BUREAU_DAY     13.501631
AMT_REQ_CREDIT_BUREAU_WEEK    13.501631
AMT_REQ_CREDIT_BUREAU_MON     13.501631
AMT_REQ_CREDIT_BUREAU_QRT     13.501631
AMT_REQ_CREDIT_BUREAU_YEAR    13.501631
```

insight: several attribute containg null value more than 40%

In [11]:

```
#removing a column with a null percentage of more than 40% is a common practice in data analysis for the followin
#if little useful information left in the column, removing it can simplify your analysis and prevent potential bi
#A column with a high null percentage may have limited predictive power or explanatory value.
#If the missing values are likely to introduce noise or reduce the column's usefulness, removing it can lead to m
#Dimensionality reduction: Removing columns with a high null percentage can help reduce the dimensionality of the
```

In [12]:

```
#Let's plot the columns vs missing value % with 40% being the cut-off marks
# before that adds your null_percentages as column to df
null_percentages = (df.isnull().mean() * 100).reset_index()
null_percentages
```

37	REG_CITY_NOT_LIVE_CITY	0.000000
38	REG_CITY_NOT_WORK_CITY	0.000000
39	LIVE_CITY_NOT_WORK_CITY	0.000000
40	ORGANIZATION_TYPE	0.000000
41	EXT_SOURCE_1	56.381073
42	EXT_SOURCE_2	0.214626
43	EXT_SOURCE_3	19.825307
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

In [13]:

```
#creates a DataFrame named null_percentageDF to store the null values percentage of each column in the applicatio
null_percentageDF = pd.DataFrame((df.isnull().sum())*100/df.shape[0]).reset_index()
null_percentageDF
```

38	REG_CITY_NOT_WORK_CITY	0.000000
39	LIVE_CITY_NOT_WORK_CITY	0.000000
40	ORGANIZATION_TYPE	0.000000
41	EXT_SOURCE_1	56.381073
42	EXT_SOURCE_2	0.214626
43	EXT_SOURCE_3	19.825307
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

In [14]:

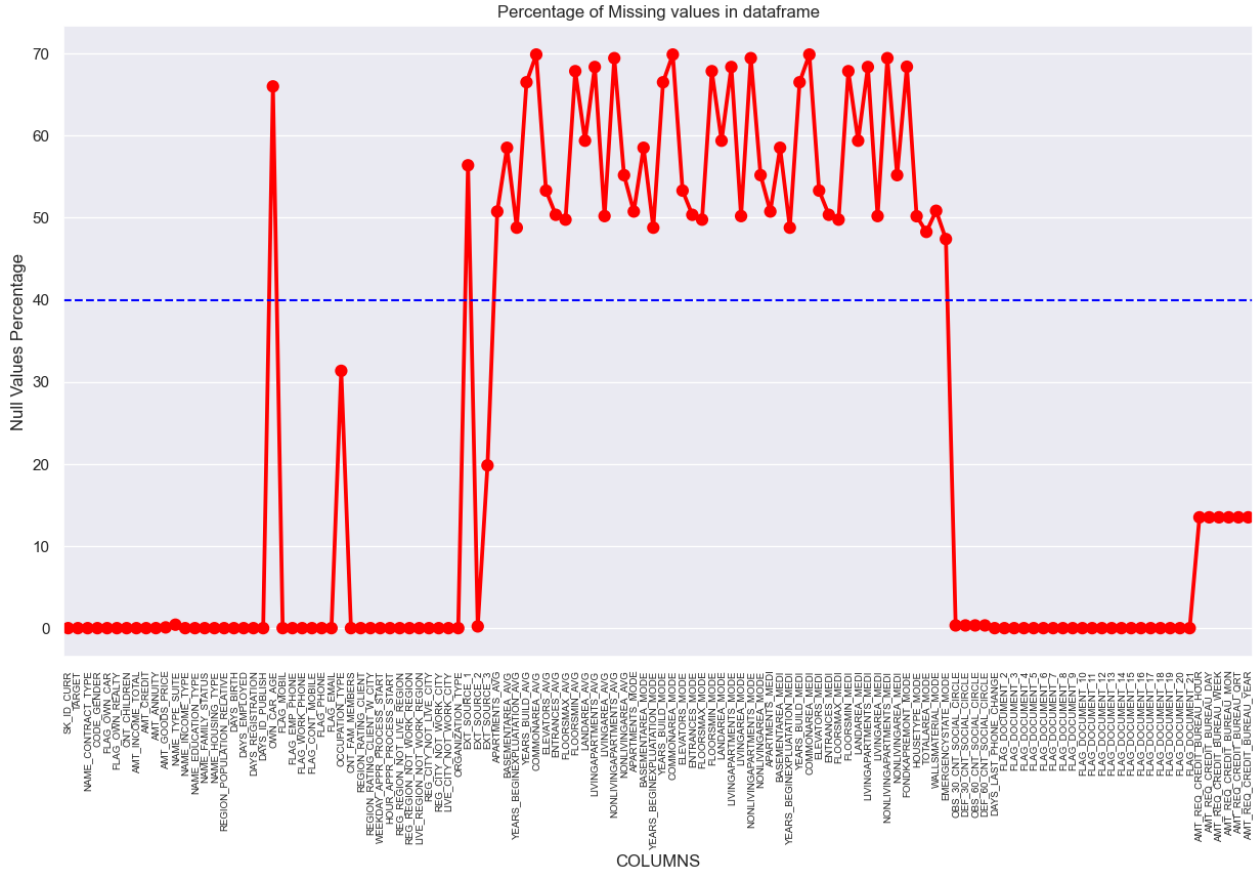
```
# asign the new column name to derived calculation of null in newly generated dataframe
null_percentageDF.columns = ['Column Name', 'Null Values Percentage']
null_percentageDF.head()

```

Out[14]:

	Column Name	Null Values Percentage
0	SK_ID_CURR	0.0
1	TARGET	0.0
2	NAME_CONTRACT_TYPE	0.0
3	CODE_GENDER	0.0
4	FLAG_OWN_CAR	0.0

In [15]:

```
#no visualize the null percentage value against column name
fig = plt.figure(figsize=(15,8))#define figure so that you can manipulate figure
ax = sns.pointplot(x="Column Name", y="Null Values Percentage", data=null_percentageDF, color='red')
plt.xticks(rotation=90, fontsize=7)#for better visibility in x-axies
ax.axhline(40, ls='--', color='blue')#ccuttoff line, parameter ls like -- and color blue
plt.title("Percentage of Missing values in dataframe")
plt.ylabel("Null Values Percentage")
plt.xlabel("COLUMNS")
plt.show()
```



Insight:plot showing there is vast no of column conating more than 40%, which are shown upper of blue line Now filter the attribute which have more than 4o% null value

In [16]:

```
# Filter and assign to varaible nullcol_40_df
nullcol_40_df = null_percentageDF[null_percentageDF["Null Values Percentage"]>=40]
nullcol_40_df
```

Out[16]:

	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
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 [17]:

```
# calculate how many column have more than 40% null value
len(nullcol_40_df)
```

Out[17]:

```
49
```

Insight: we found 49 column have more than 40% null value, and notice that this column related to apartment and apartment sizes

Now Analyzed and delete unnacessary , irrelevant column from df by visualization

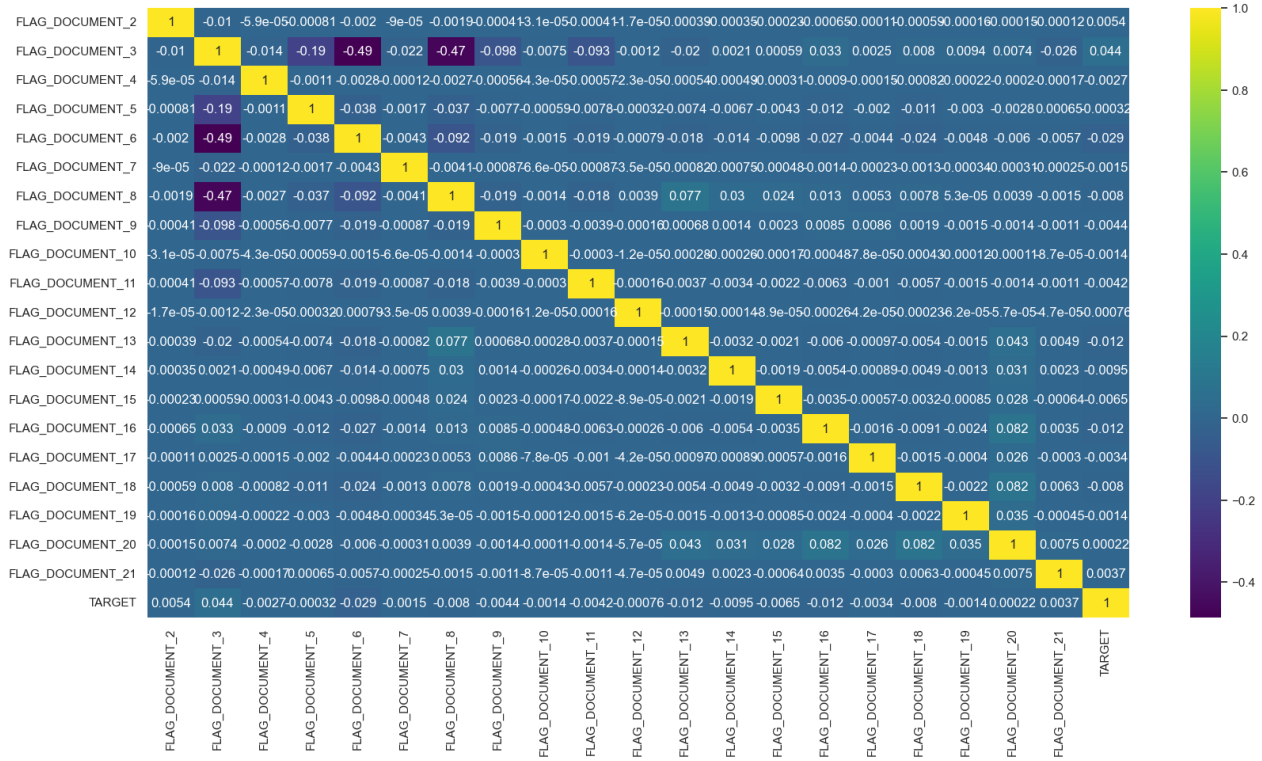
In [8]:

```
# Checking correlation of EXT_SOURCE_X columns vs TARGET column
# define  and new df and assign into  variable called source then make corelation among them by corr() , ehich as
#then create the heatmap with corelation result
Source = df[["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 ="inferno")
```


```
# Insight: From this heatmap analysis we can see there is no corelation with ext source  and target , where we see ext_source 1 has 56%null value and 3 has 19.2%
```

In [19]:

```
# Checking the relevance of Flag_Document and whether it has any relation with loan repayment status
Flagdf=df[['FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3','FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6','FLAG_DOC
           'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9','FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12','FLAG_
           'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15','FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18',
           'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21',"TARGET"]]
Flagdf_corr=Flagdf.corr()
fig = plt.figure(figsize=(20,10))
ax = sns.heatmap(Flagdf_corr,
            xticklabels=Flagdf_corr.columns,
            yticklabels=Flagdf_corr.columns,
            annot = True,
            cmap ="viridis")

```



Insight: from this map we can see no crealtion between target and flag document, buut there is some doc showing differnet color , Lets check this relation flag doc and loan status with count plot. to understand better

In [20]:

```
import itertools

```

In [21]:

```
Flagdf=['FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3','FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6','FLAG_DOCUME
           'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9','FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12','FLAG_
           'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15','FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18',
           'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21']
df_flag = df[['FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3','FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6','FLAG_
           'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9','FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12','FLAG_
           'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15','FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18',
           'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21',"TARGET"]]
length = len(Flagdf)
df_flag["TARGET"] = df_flag["TARGET"].replace({1:"Defaulter",0:"Repayer"})

fig, axes = plt.subplots(5, 4, figsize=(21, 24))

for i, ax in zip(Flagdf, axes.flatten()):#the calculation of row and column indices is handled by axes.flatten()
    sns.countplot(x=df_flag[i], hue=df_flag["TARGET"], palette=["r", "g"], ax=ax)
    ax.tick_params(axis="y", labelsize=8)
    ax.set_xlabel("")

plt.tight_layout()
```

In [22]:

```
Flagdf1 = df_flag.drop(['FLAG_DOCUMENT_3', 'TARGET'], axis=1)
Flagdf1.head()
```

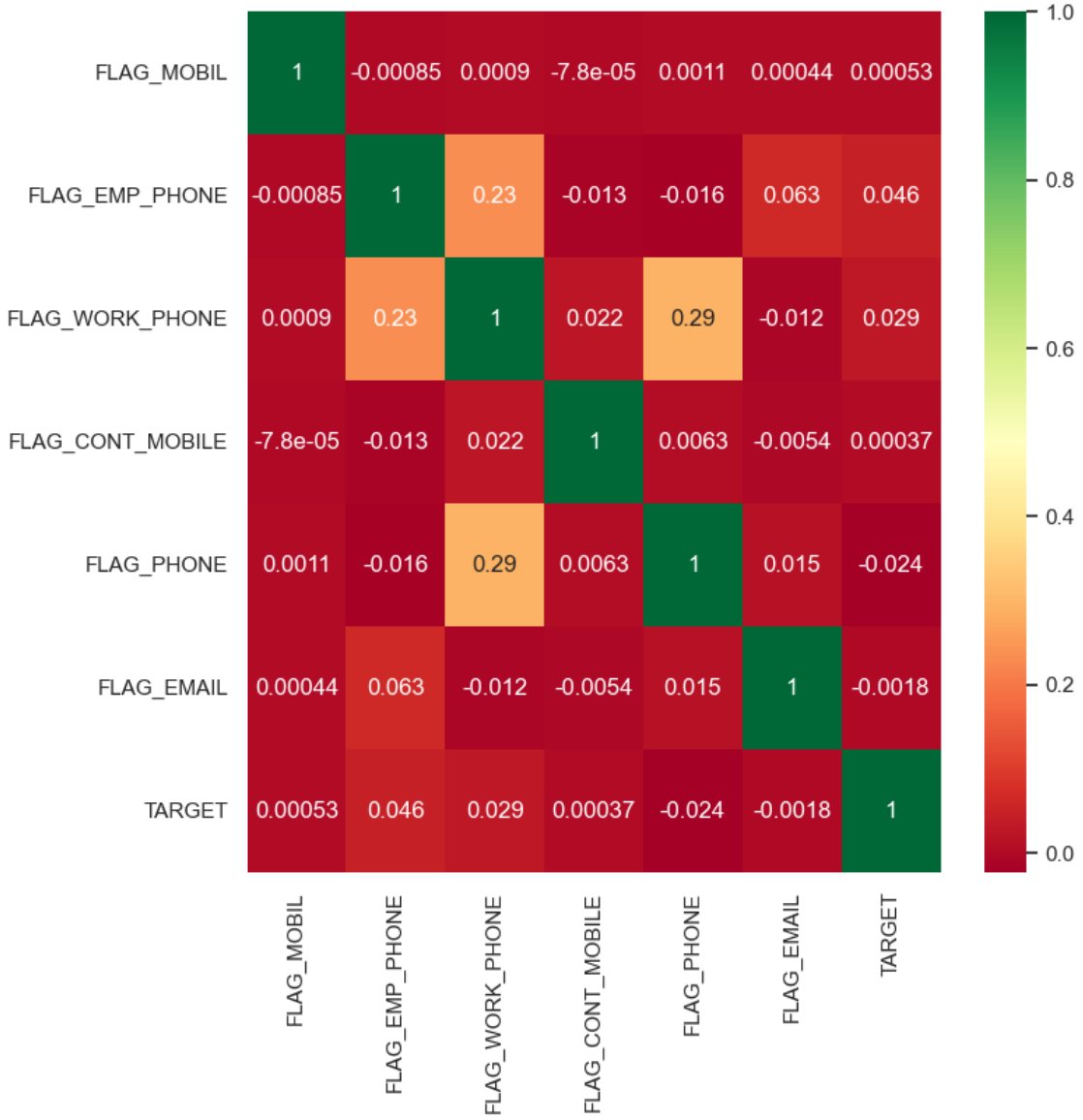
Out[22]:

	FLAG_DOCUMENT_2	FLAG_DOCUMENT_4	FLAG_DOCUMENT_5	FLAG_DOCUMENT_6	FLAG_DOCUMENT_7	FLAG_DOCUMENT_8	FLAG_DO
0	0	0	0	0	0	0	
1	0	0	0	0	0	0	
2	0	0	0	0	0	0	
3	0	0	0	0	0	0	
4	0	0	0	0	0	1	

Insight:The above graph shows that in most of the loan application cases, clients who applied for loans has not submitted FLAG_DOCUMENT_X except FLAG_DOCUMENT_3. Thus, Except for FLAG_DOCUMENT_3, we can delete rest of the columns. Data shows if borrower has submitted FLAG_DOCUMENT_3 then there is a less chance of defaulting the loan.

In [23]:

```
# checking is there is any correlation between mobile phone, work phone etc, email, Family members and Region rat
contact_col = ['FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE',
       'FLAG_PHONE', 'FLAG_EMAIL','TARGET']
Contact_corr = df[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")
```



In [24]:

```
# Lets filter the unwanted attribute and sum it so that we can delete its from main df
# create a list of columns that needs to be dropped including the columns with >40% null values
Unwanted_application = nullcol_40_df["Column Name"].tolist()+ ['EXT_SOURCE_2','EXT_SOURCE_3']
# as EXT_SOURCE_1 column is already included in nullcol_40_application
len(Unwanted_application)
```

Out[24]:

```
51
```

In [25]:

```
if 'FLAG_DOCUMENT_3' in Flagdf:
    Flagdf.remove('FLAG_DOCUMENT_3')

```

In [26]:

```
combined_list = Unwanted_application + Flagdf
len(combined_list)
```

Out[26]:

```
70
```

In [27]:

```
contact_col.remove('TARGET')
Unwanted_application = combined_list + contact_col
len(Unwanted_application)
```

Out[27]:

```
76
```

#Total 76 column found where 49 column have more than 40% null values and others have very low or no relationship with target value, so we can delete those column

In [28]:

```
# Dropping the unnecessary columns from applicationDF
df.drop(labels=Unwanted_application,axis=1,inplace=True)
```

In [29]:

```
df.shape
```

Out[29]:

```
(307511, 46)
```

In [30]:

```
#now we have 46 column , Lets check the total info of df
df.info()
```

```
<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 [31]:

```
# we see in our df  some attribute is showing the object, Lets convert those object into catagory type
#Befire that check how many unique elements ,those atribute contain.

# Calculate cardinality of each column
cardinality = df.nunique()

# Print cardinality for each column
print(cardinality)

# Identify columns with low cardinality and clear categorical semantics
low_cardinality_cols = [col for col in df.columns if cardinality[col] <= 15]
categorical_cols = [col for col in low_cardinality_cols if df[col].dtype == 'object']

# Print identified columns
print("Columns with low cardinality:")
print(low_cardinality_cols)
print("\nCategorical columns:")
print(categorical_cols)
```

```
SK_ID_CURR                    307511
TARGET                             2
NAME_CONTRACT_TYPE                 2
CODE_GENDER                        3
FLAG_OWN_CAR                       2
FLAG_OWN_REALTY                    2
CNT_CHILDREN                      15
AMT_INCOME_TOTAL                2548
AMT_CREDIT                      5603
AMT_ANNUITY                    13672
AMT_GOODS_PRICE                 1002
NAME_TYPE_SUITE                    7
NAME_INCOME_TYPE                   8
NAME_EDUCATION_TYPE                5
NAME_FAMILY_STATUS                 6
NAME_HOUSING_TYPE                  6
REGION_POPULATION_RELATIVE        81
DAYS_BIRTH                     17460
DAYS_EMPLOYED                  12574
DAYS_REGISTRATION              15688
DAYS_ID_PUBLISH                 6168
OCCUPATION_TYPE                   18
CNT_FAM_MEMBERS                   17
REGION_RATING_CLIENT               3
REGION_RATING_CLIENT_W_CITY        3
WEEKDAY_APPR_PROCESS_START         7
HOUR_APPR_PROCESS_START           24
REG_REGION_NOT_LIVE_REGION         2
REG_REGION_NOT_WORK_REGION         2
LIVE_REGION_NOT_WORK_REGION        2
REG_CITY_NOT_LIVE_CITY             2
REG_CITY_NOT_WORK_CITY             2
LIVE_CITY_NOT_WORK_CITY            2
ORGANIZATION_TYPE                 58
OBS_30_CNT_SOCIAL_CIRCLE          33
DEF_30_CNT_SOCIAL_CIRCLE          10
OBS_60_CNT_SOCIAL_CIRCLE          33
DEF_60_CNT_SOCIAL_CIRCLE           9
DAYS_LAST_PHONE_CHANGE          3773
FLAG_DOCUMENT_3                    2
AMT_REQ_CREDIT_BUREAU_HOUR         5
AMT_REQ_CREDIT_BUREAU_DAY          9
AMT_REQ_CREDIT_BUREAU_WEEK         9
AMT_REQ_CREDIT_BUREAU_MON         24
AMT_REQ_CREDIT_BUREAU_QRT         11
AMT_REQ_CREDIT_BUREAU_YEAR        25
dtype: int64
Columns with low cardinality:
['TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'NAME
_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'REG
ION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_START', 'REG_REGION_NOT_LIVE_RE
GION', 'REG_REGION_NOT_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_
NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'DEF_30_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'FLAG
_DOCUMENT_3', 'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
'AMT_REQ_CREDIT_BUREAU_QRT']

Categorical columns:
['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'NAME_TYPE_SUITE', 'NAME_INCOME
_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'WEEKDAY_APPR_PROCESS_START']
```

Then, the code identifies columns with low cardinality (less than or equal to 15) and checks if they have an object data type to determine the categorical columns

In [32]:

```
#Conversion of Object and Numerical columns to Categorical Columns
categorical_columns = ['NAME_CONTRACT_TYPE','CODE_GENDER','NAME_TYPE_SUITE','NAME_INCOME_TYPE','NAME_EDUCATION_TY
                       'NAME_FAMILY_STATUS','NAME_HOUSING_TYPE','OCCUPATION_TYPE','WEEKDAY_APPR_PROCESS_START',
                       'ORGANIZATION_TYPE','FLAG_OWN_CAR','FLAG_OWN_REALTY','LIVE_CITY_NOT_WORK_CITY',
                       'REG_CITY_NOT_LIVE_CITY','REG_CITY_NOT_WORK_CITY','REG_REGION_NOT_WORK_REGION',
                       'LIVE_REGION_NOT_WORK_REGION','REGION_RATING_CLIENT','WEEKDAY_APPR_PROCESS_START',
                       'REGION_RATING_CLIENT_W_CITY'
                      ]
for col in categorical_columns:
    df[col] =pd.Categorical(df[col])


```

In [33]:

```
df.info()
```

```
<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  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 
dtypes: category(19), float64(18), int64(9)
memory usage: 68.9 MB
```

#Check the null value and imputation To impute null values in categorical variables which has lower null percentage, mode() is used to impute the most frequent items. To impute null values in categorical variables which has higher null percentage, a new category is created. To impute null values in numerical variables which has lower null percentage, median() is used as There are no outliers in the columns Mean returned decimal values and median returned whole numbers and the columns were number of requests

In [34]:

```
null_percentageDF = pd.DataFrame((df.isnull().sum())*100/df.shape[0]).reset_index()
null_percentageDF
```

Out[34]:

	index	0
0	SK_ID_CURR	0.000000
1	TARGET	0.000000
2	NAME_CONTRACT_TYPE	0.000000
3	CODE_GENDER	0.000000
4	FLAG_OWN_CAR	0.000000
5	FLAG_OWN_REALTY	0.000000
6	CNT_CHILDREN	0.000000
7	AMT_INCOME_TOTAL	0.000000
8	AMT_CREDIT	0.000000
9	AMT_ANNUITY	0.003902
10	AMT_GOODS_PRICE	0.090403
11	NAME_TYPE_SUITE	0.420148
12	NAME_INCOME_TYPE	0.000000
13	NAME_EDUCATION_TYPE	0.000000
14	NAME_FAMILY_STATUS	0.000000
15	NAME_HOUSING_TYPE	0.000000
16	REGION_POPULATION_RELATIVE	0.000000
17	DAYS_BIRTH	0.000000
18	DAYS_EMPLOYED	0.000000
19	DAYS_REGISTRATION	0.000000
20	DAYS_ID_PUBLISH	0.000000
21	OCCUPATION_TYPE	31.345545
22	CNT_FAM_MEMBERS	0.000650
23	REGION_RATING_CLIENT	0.000000
24	REGION_RATING_CLIENT_W_CITY	0.000000
25	WEEKDAY_APPR_PROCESS_START	0.000000
26	HOUR_APPR_PROCESS_START	0.000000
27	REG_REGION_NOT_LIVE_REGION	0.000000
28	REG_REGION_NOT_WORK_REGION	0.000000
29	LIVE_REGION_NOT_WORK_REGION	0.000000
30	REG_CITY_NOT_LIVE_CITY	0.000000
31	REG_CITY_NOT_WORK_CITY	0.000000
32	LIVE_CITY_NOT_WORK_CITY	0.000000
33	ORGANIZATION_TYPE	0.000000
34	OBS_30_CNT_SOCIAL_CIRCLE	0.332021
35	DEF_30_CNT_SOCIAL_CIRCLE	0.332021
36	OBS_60_CNT_SOCIAL_CIRCLE	0.332021
37	DEF_60_CNT_SOCIAL_CIRCLE	0.332021
38	DAYS_LAST_PHONE_CHANGE	0.000325
39	FLAG_DOCUMENT_3	0.000000
40	AMT_REQ_CREDIT_BUREAU_HOUR	13.501631
41	AMT_REQ_CREDIT_BUREAU_DAY	13.501631
42	AMT_REQ_CREDIT_BUREAU_WEEK	13.501631
43	AMT_REQ_CREDIT_BUREAU_MON	13.501631
44	AMT_REQ_CREDIT_BUREAU_QRT	13.501631
45	AMT_REQ_CREDIT_BUREAU_YEAR	13.501631

In [131]:

```
# First impute catagoriacl attribute which have lower percentage of null with mode
df['NAME_TYPE_SUITE'].fillna((df['NAME_TYPE_SUITE'].mode()[0]),inplace = True)
```

In [36]:

```
impute impute catagoriacl attribute which have higher percentage of null, 31.34% with unknown suitabl elemnts of this

CUPATION_TYPE'] = df['OCCUPATION_TYPE'].cat.add_categories(['Unknown'])# First add a elemnt to column name Unknown
CUPATION_TYPE'].fillna('Unknown', inplace=True)#Then fill the the null values with unknown
```

In [37]:

```
#Impute numerical variables with the median as there are no outliers that can be seen from results of describe()
#and mean() returns decimal values and , values that are more than 2 or 3 standard deviations away from the mean c
#Lets check mean and std.dev
df[['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()
```

Out[37]:

IT_BUREAU_HOUR	AMT_REQ_CREDIT_BUREAU_DAY	AMT_REQ_CREDIT_BUREAU_WEEK	AMT_REQ_CREDIT_BUREAU_MON	AMT_REQ_CREDIT_E
265992.000000	265992.000000	265992.000000	265992.000000	2
0.006402	0.007000	0.034362	0.267395	
0.083849	0.110757	0.204685	0.916002	
0.000000	0.000000	0.000000	0.000000	
0.000000	0.000000	0.000000	0.000000	
0.000000	0.000000	0.000000	0.000000	
0.000000	0.000000	0.000000	0.000000	
4.000000	9.000000	8.000000	27.000000	

Insight:its show all the attribute std.dev is 0, so it suggest there is no outliar , so we can change the null value with median

In [38]:

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

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

In [39]:

```
# Lets check the null percentage again
null_percentageDF = pd.DataFrame((df.isnull().sum())*100/df.shape[0]).reset_index()
null_percentageDF
```

Out[39]:

	index	0
0	SK_ID_CURR	0.000000
1	TARGET	0.000000
2	NAME_CONTRACT_TYPE	0.000000
3	CODE_GENDER	0.000000
4	FLAG_OWN_CAR	0.000000
5	FLAG_OWN_REALTY	0.000000
6	CNT_CHILDREN	0.000000
7	AMT_INCOME_TOTAL	0.000000
8	AMT_CREDIT	0.000000
9	AMT_ANNUITY	0.003902
10	AMT_GOODS_PRICE	0.090403
11	NAME_TYPE_SUITE	0.000000
12	NAME_INCOME_TYPE	0.000000
13	NAME_EDUCATION_TYPE	0.000000
14	NAME_FAMILY_STATUS	0.000000
15	NAME_HOUSING_TYPE	0.000000
16	REGION_POPULATION_RELATIVE	0.000000
17	DAYS_BIRTH	0.000000
18	DAYS_EMPLOYED	0.000000
19	DAYS_REGISTRATION	0.000000
20	DAYS_ID_PUBLISH	0.000000
21	OCCUPATION_TYPE	0.000000
22	CNT_FAM_MEMBERS	0.000650
23	REGION_RATING_CLIENT	0.000000
24	REGION_RATING_CLIENT_W_CITY	0.000000
25	WEEKDAY_APPR_PROCESS_START	0.000000
26	HOUR_APPR_PROCESS_START	0.000000
27	REG_REGION_NOT_LIVE_REGION	0.000000
28	REG_REGION_NOT_WORK_REGION	0.000000
29	LIVE_REGION_NOT_WORK_REGION	0.000000
30	REG_CITY_NOT_LIVE_CITY	0.000000
31	REG_CITY_NOT_WORK_CITY	0.000000
32	LIVE_CITY_NOT_WORK_CITY	0.000000
33	ORGANIZATION_TYPE	0.000000
34	OBS_30_CNT_SOCIAL_CIRCLE	0.332021
35	DEF_30_CNT_SOCIAL_CIRCLE	0.332021
36	OBS_60_CNT_SOCIAL_CIRCLE	0.332021
37	DEF_60_CNT_SOCIAL_CIRCLE	0.332021
38	DAYS_LAST_PHONE_CHANGE	0.000325
39	FLAG_DOCUMENT_3	0.000000
40	AMT_REQ_CREDIT_BUREAU_HOUR	0.000000
41	AMT_REQ_CREDIT_BUREAU_DAY	0.000000
42	AMT_REQ_CREDIT_BUREAU_WEEK	0.000000
43	AMT_REQ_CREDIT_BUREAU_MON	0.000000
44	AMT_REQ_CREDIT_BUREAU_QRT	0.000000
45	AMT_REQ_CREDIT_BUREAU_YEAR	0.000000

```
#Now its looking very ggod, the least percentage of null we can ignore
```

```
#Lets do the 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
```

In [40]:

```
#First check the data balance
# Calculate the imbalance ratio
# and  see in visualization
imbalance = df["TARGET"].value_counts().reset_index()
imbalance
```

Out[40]:

	index	TARGET
0	0	282686
1	1	24825

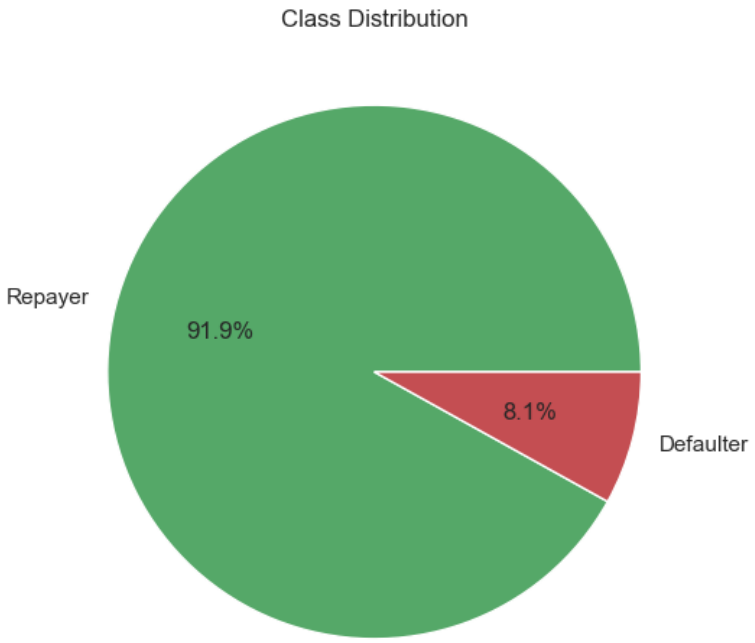
In [41]:

```
# See this result by visualization
# Calculate the imbalance ratio
imbalance_ratio = imbalance.iloc[1, 1] / imbalance.iloc[0, 1]
print("Imbalance Ratio:", imbalance_ratio)


```

```
Imbalance Ratio: 0.08781828601345662
```

In [42]:

```
# Plot the class distribution using a pie chart
plt.figure(figsize=(6, 6))
x= ['Repayer','Defaulter']
plt.pie(imbalance["TARGET"], labels=x, autopct='%1.1f%%', colors=['g', 'r'])
plt.title("Class Distribution")
plt.show()
```



In [43]:

```
#Ploting function
#Following are the common functions customized to perform uniform anaysis that is called for all plots:
#To understand function Create and check without function
```

Define a univariate catagorical function The function takes the input parameter feature, which represents the categorical column in the DF DataFrame. temp calculates the count of each unique category in the feature column. df1 creates a DataFrame that stores the categories and their corresponding counts in two columns: feature and 'Number of contracts'.

#We want to calculate the percentage of defaulters within each unique value of the Feature column. #DF[[feature, 'TARGET']] selects the columns Feature and TARGET: #.groupby([feature], as_index=False) groups the dataframe by the Feature column: #.mean() calculates the mean value (percentage of 1s) for each group: Lets do it

In [44]:

```
cat_perc = df[["NAME_CONTRACT_TYPE", 'TARGET']].groupby(["NAME_CONTRACT_TYPE"], as_index=False).mean()
cat_perc
```

Out[44]:

	NAME_CONTRACT_TYPE	TARGET
0	Cash loans	0.083459
1	Revolving loans	0.054783

In [45]:

```
#Convert Target attribute into percentage
cat_perc["TARGET"] = cat_perc["TARGET"]*100
cat_perc
```

Out[45]:

	NAME_CONTRACT_TYPE	TARGET
0	Cash loans	8.345913
1	Revolving loans	5.478329

In [46]:

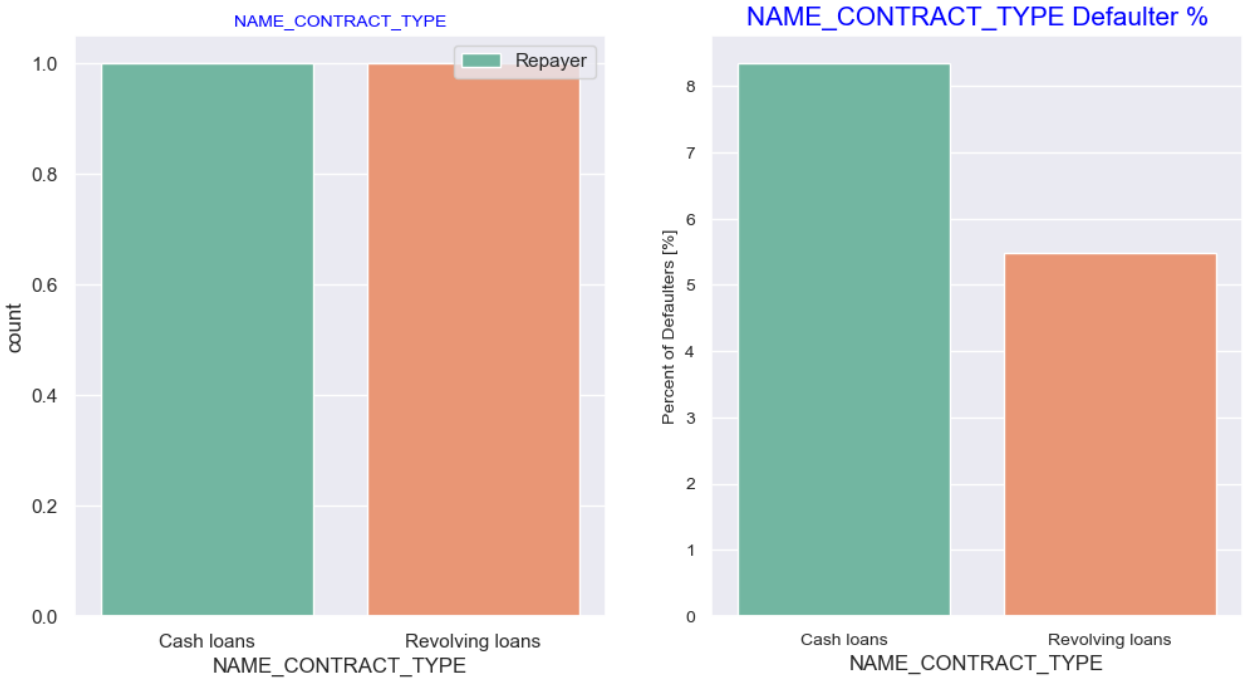
```
#sorted the resulted value in descending
cat_perc.sort_values(by='TARGET', ascending=False, inplace=True)
cat_perc
```

Out[46]:

	NAME_CONTRACT_TYPE	TARGET
0	Cash loans	8.345913
1	Revolving loans	5.478329

In [47]:

```
# Then define the subplot with conditional if, else statement
# How it plots if horizontal layout is True, if not
horizontal_layout = 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))

# Now make the count plot for categorical column
s = sns.countplot(ax=ax1, x="NAME_CONTRACT_TYPE", data=cat_perc, palette='Set2')

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

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




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


plt.ylabel('Percent of Defaulters [%]', fontsize=10)
plt.tick_params(axis='both', which='major', labelsize=10)
ax2.set_title("NAME_CONTRACT_TYPE Defaulter %", fontdict={'fontsize': 15, 'fontweight': 5, 'color': 'Blue'})

plt.show()

```



In [48]:

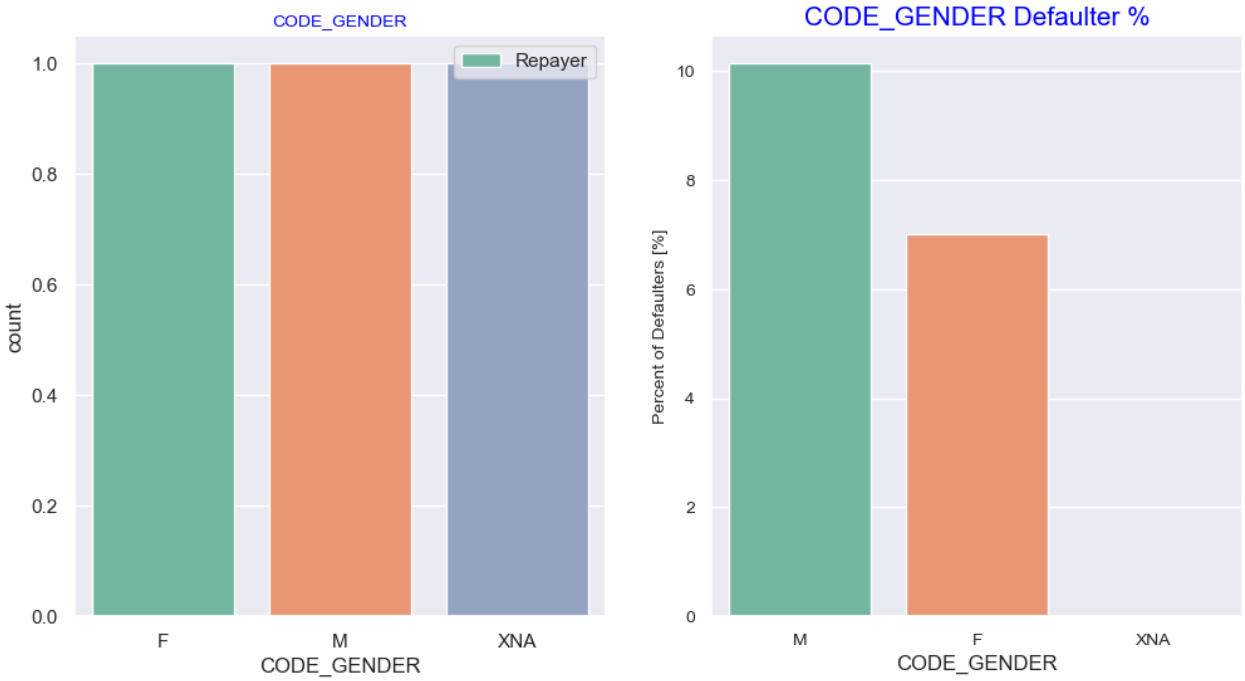
```
cat_perc = df[["CODE_GENDER", 'TARGET']].groupby(["CODE_GENDER"], as_index=False).mean()
cat_perc["TARGET"] = cat_perc["TARGET"]*100
cat_perc.sort_values(by='TARGET', ascending=False, inplace=True)

cat_perc

```

Out[48]:

	CODE_GENDER	TARGET
1	M	10.141920
0	F	6.999328
2	XNA	0.000000

In [49]:

```
# Then define the subplot with conditional if, else statement
# How it plots if horizontal layout is True, if not
horizontal_layout = 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))

# Now make the count plot for categorical column
s = sns.countplot(ax=ax1, x="CODE_GENDER", data=cat_perc, palette='Set2')

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

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




# Subplot 2: Percentage of defaulters within the categorical column
s = sns.barplot(ax=ax2, x="CODE_GENDER", y='TARGET', order=cat_perc["CODE_GENDER"], data=cat_perc, palette='Set2'


plt.ylabel('Percent of Defaulters [%]', fontsize=10)
plt.tick_params(axis='both', which='major', labelsize=10)
ax2.set_title("CODE_GENDER Defaulter %", fontdict={'fontsize': 15, 'fontweight': 5, 'color': 'Blue'})

plt.show()
```

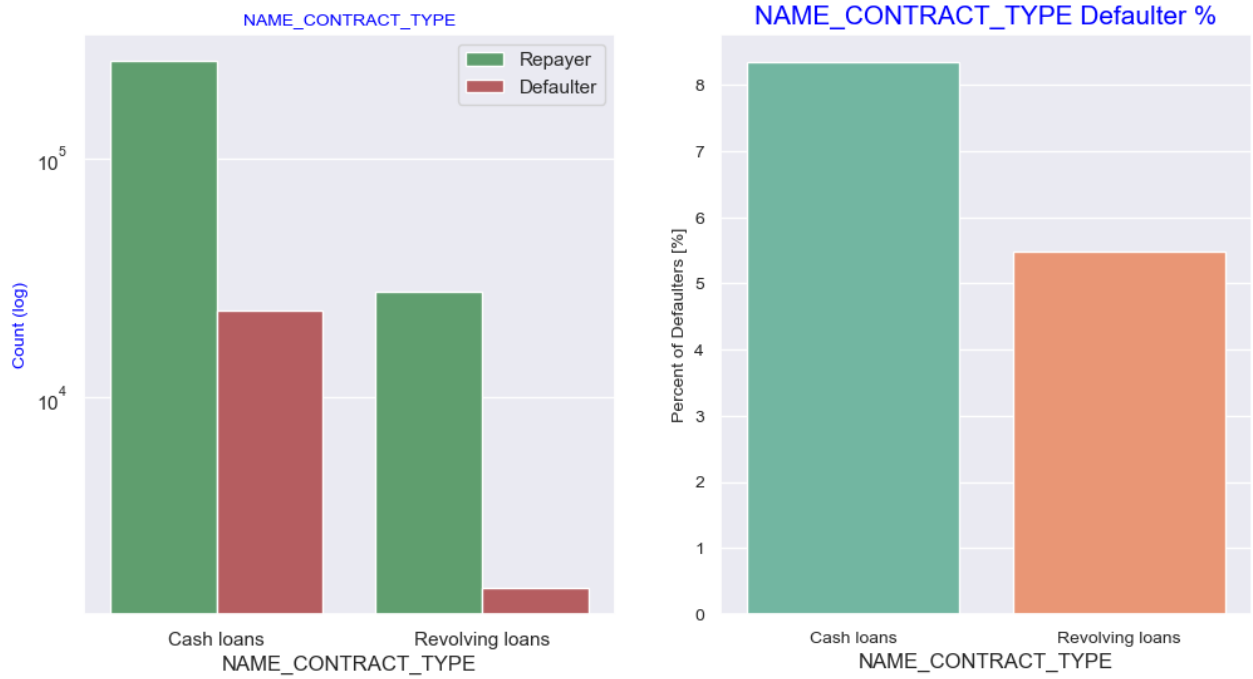


In [50]:

```
#Lets  do univariate analysis  by defining function
def univariate_categorical(feature,ylog=False,label_rotation=False,horizontal_layout=True):
    temp = df[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 = df[[feature, 'TARGET']].groupby([feature],as_index=False).mean()
    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=df,
                    hue ="TARGET",
                    order=cat_perc[feature],
                    palette=['g','r'])

    # Define common styling
    ax1.set_title(feature, fontdict={'fontsize' : 10, 'fontweight' : 3, 'color' : 'Blue'})
    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, 'color' : 'Blue'})


    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' : 5, 'color' : 'Blue'})

    plt.show();
```

In [51]:

```
# by calling defined function , now we can see importance of each catagorical attribute
# Checking the contract type based on loan repayment status
univariate_categorical('NAME_CONTRACT_TYPE',True)
```



Insight:among the contract type, cashloans contract more greater than Revolving loans contract And the people who take cash loan they are more defaulter

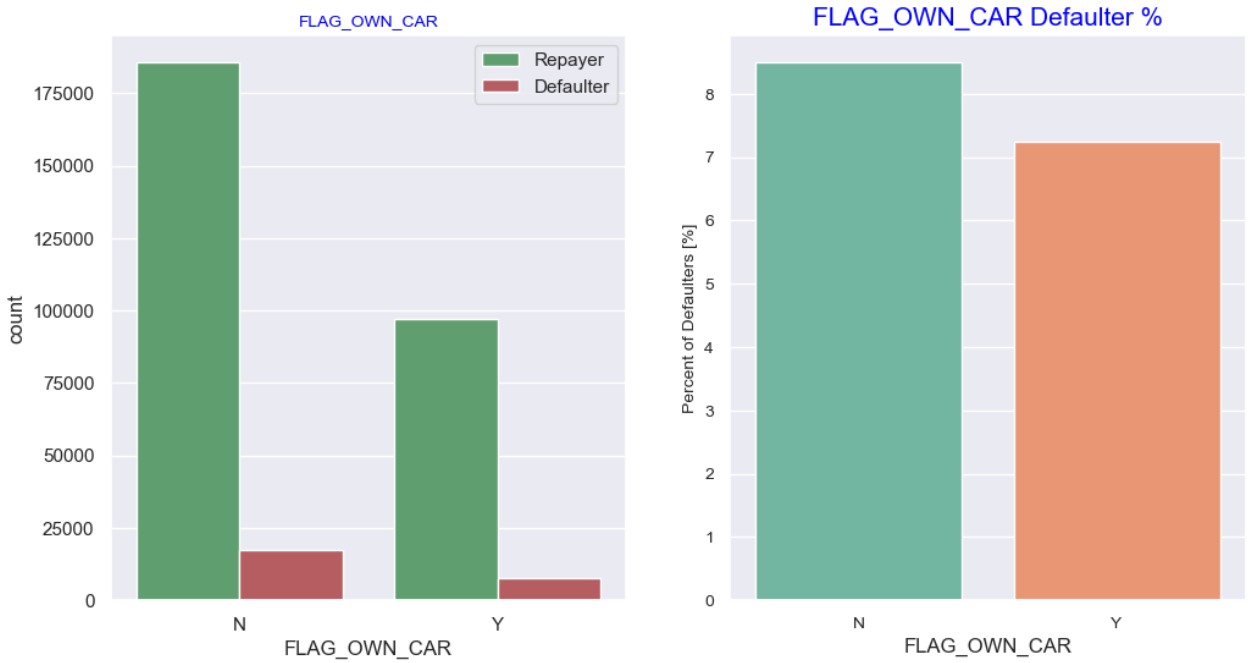
In [52]:

```
# Now we can see every catagorical attribute by calling function

# Checking the contract type based on loan repayment status
univariate_categorical('CODE_GENDER',True)
```

Insight: Female no greater than male and XNA, MAle are more defaulter than feamle

In [53]:

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



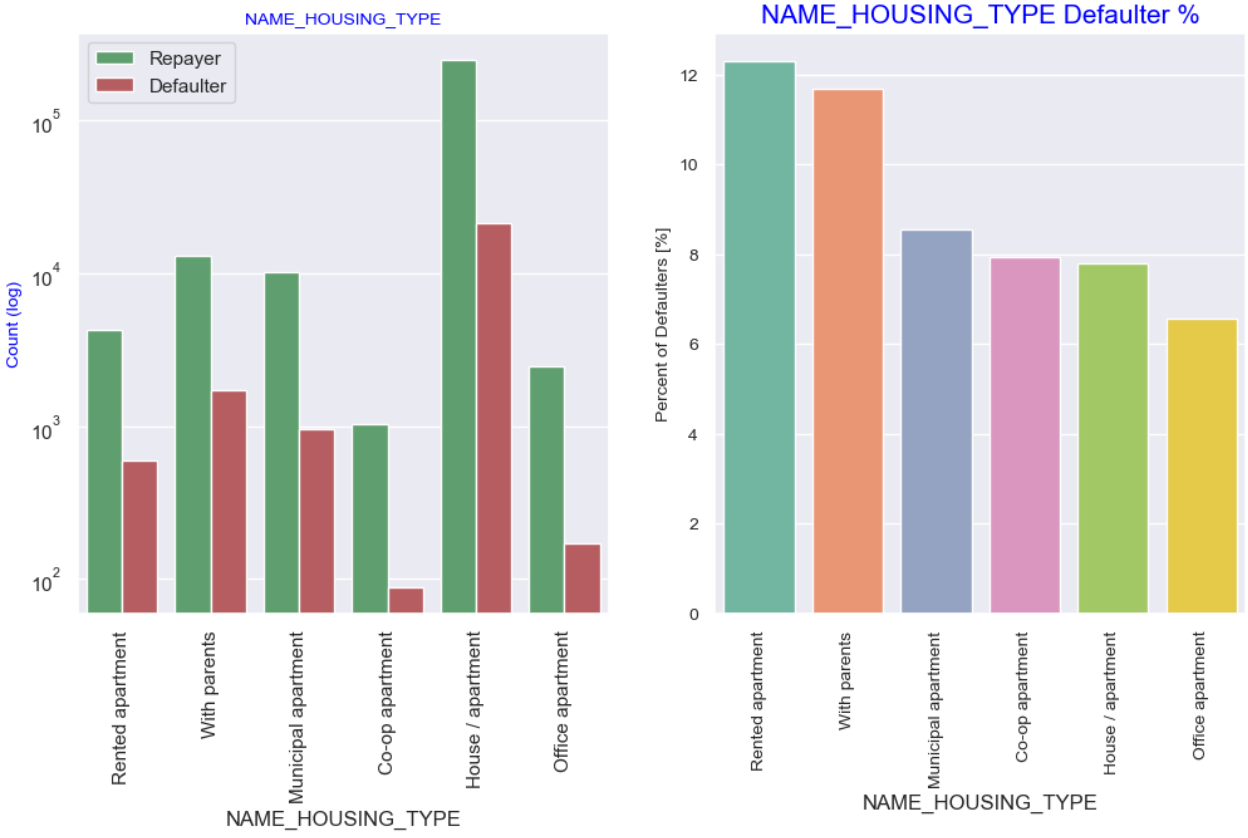
Inferences: Clients who own a car are half in number of the clients who dont own a car. But based on the percentage of deault, there is no correlation between owning a car and loan repayment as in both cases the default percentage is almost same.

In [ ]:

```

```

In [54]:

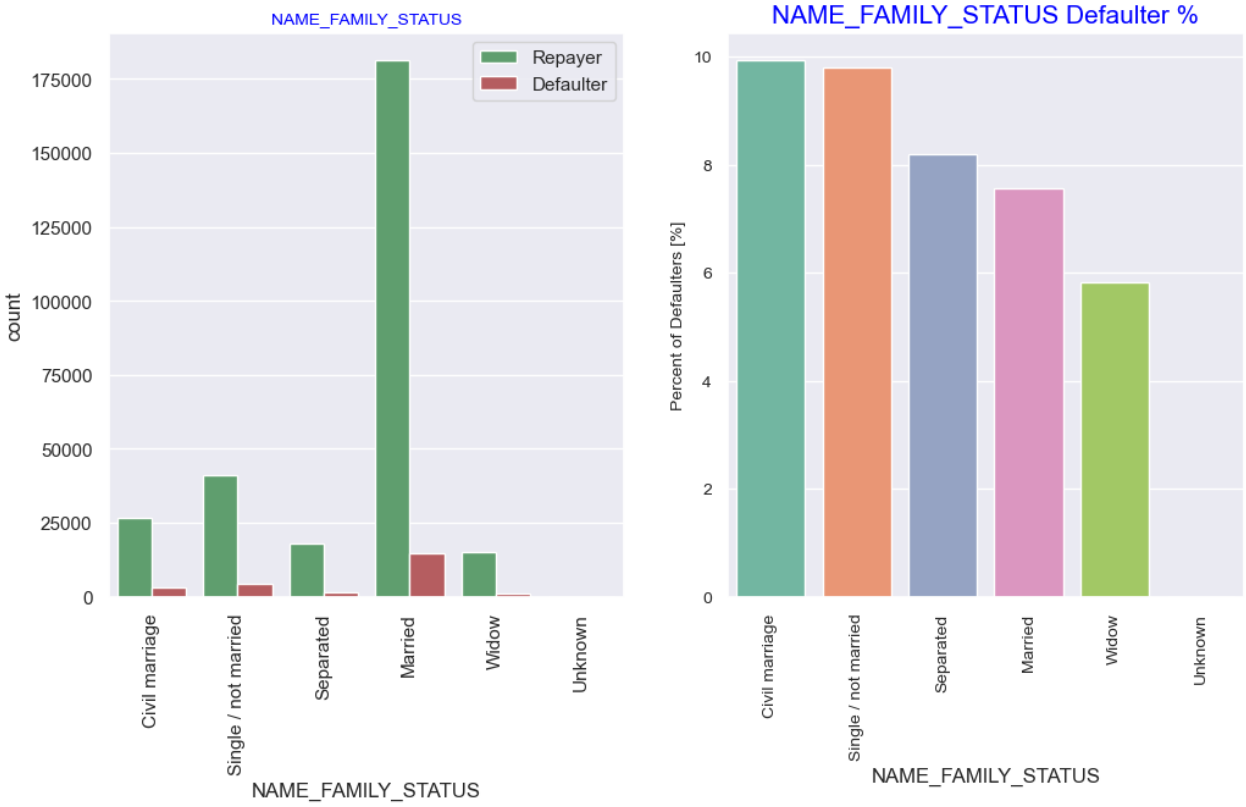
```
# Analyzing Housing Type based on loan repayment status
univariate_categorical("NAME_HOUSING_TYPE",True,True,True)
```



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 [55]:

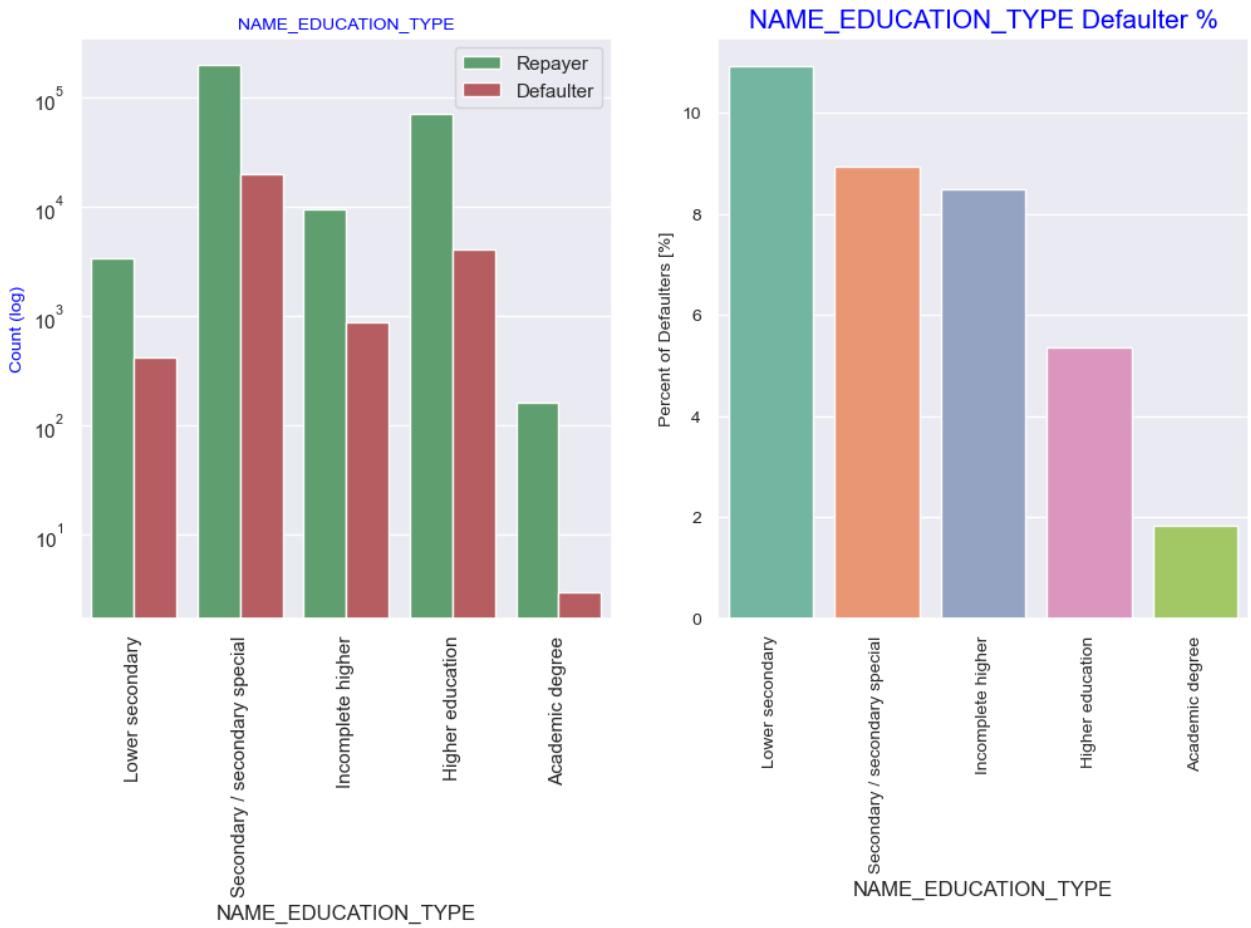
```
# Analyzing Family status based on loan repayment status
univariate_categorical("NAME_FAMILY_STATUS",False,True,True)
```



Insight:Married people higher than others family status, but civil marriage and single people are higher defaulter than others, where widow are lowest

In [56]:

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



Insight: The people who completed secondery take the loan than the other education, And people who have lower education high defaulter than others, where people with academic degree least dea loan defaulter

In [57]:

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

Insight: Count of Working people are more than other income type people, And the Maternity leave person almost 40% ratio of not returning loan, followed by Unemployed (37%). The rest of types of incomes are under the average of 10% for not returning loans.

False: This is the value for the ylog parameter. By setting it to False, it means that the count plot will not be displayed using a logarithmic scale on the y-axis. False: This is the value for the label_rotation parameter. By setting it to False, it means that the x-axis labels will not be rotated. True: This is the value for the horizontal_layout parameter. By setting it to True, it means that the count plot and percentage plot will be displayed side by side in a horizontal layout.

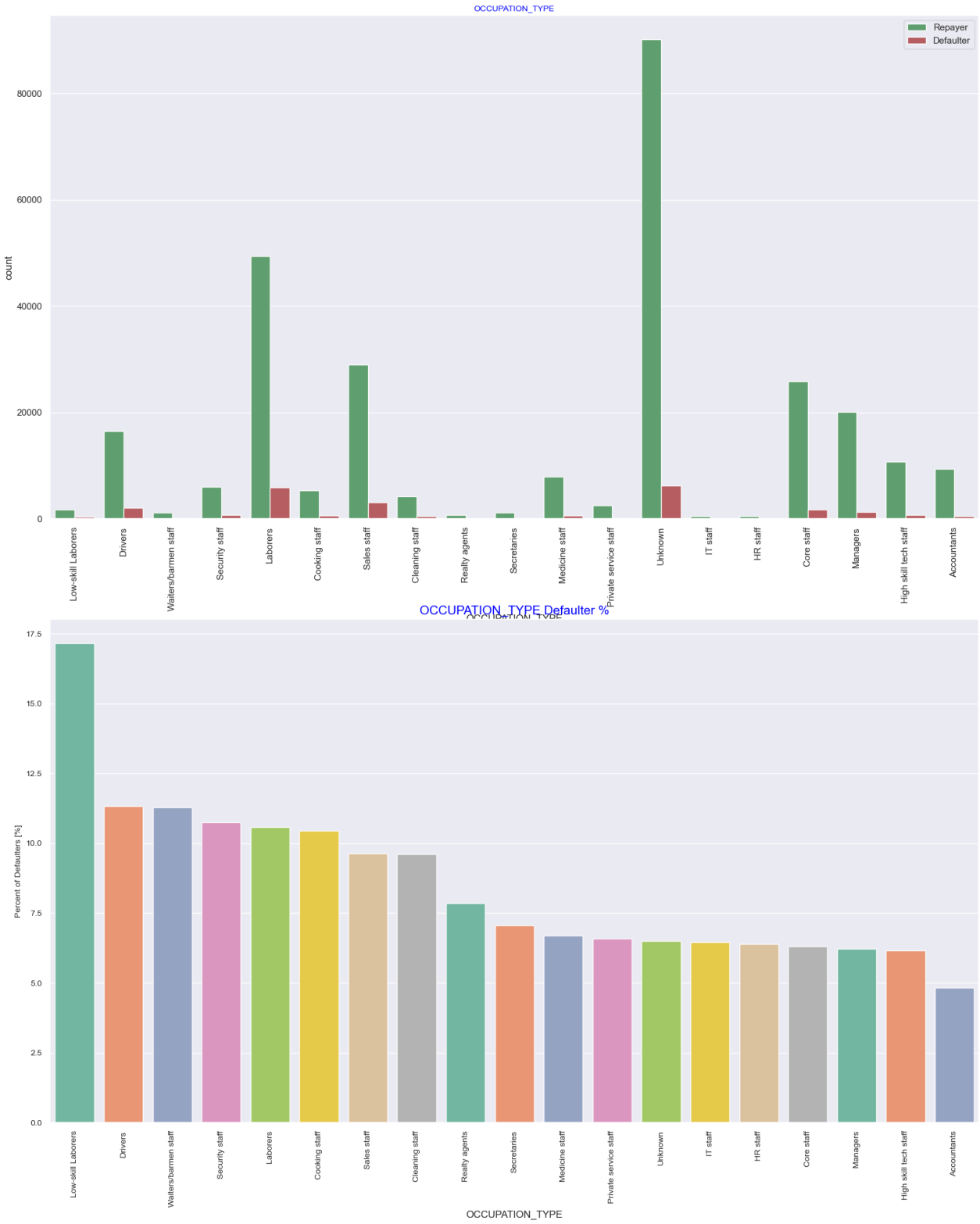
In [58]:

```
#check region rating client
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 [59]:

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



Insight:The unknown profeshion are most , followed by Lboures people , who took the loan more than the other profeshion, where low_skills labour around (around17%)are defaulter than the others profeshional people

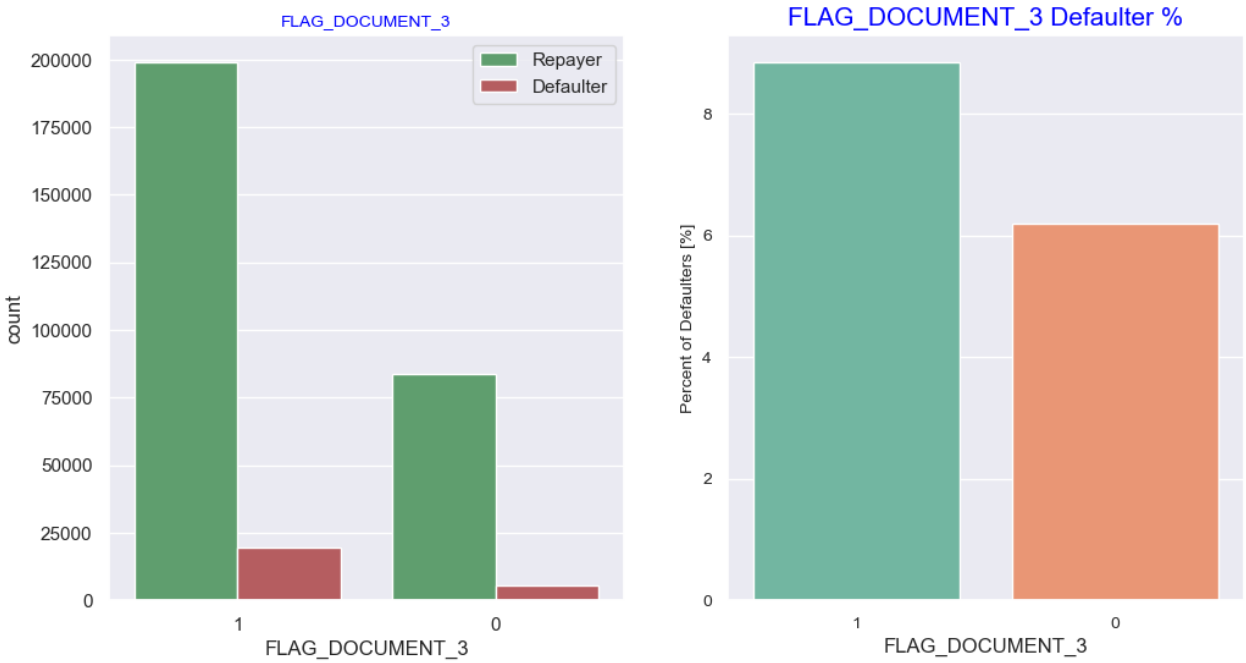
In [60]:

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


```
Insight:Most of the people application for loan are from Business Entity Type 3
For a very high number of applications, Organization type information is unavailable(XNA)
It can be seen that following category of organization type has lesser defaulters thus safer for providing loans:
Trade Type 4 and 5.
Industry type 8, transport type3, industry type 13 have relative high defaulting rate, and thus should be avoided to be approved for loan or provide loan with higher interest rate to mitigate the risk of defaulting.
```

In [61]:

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

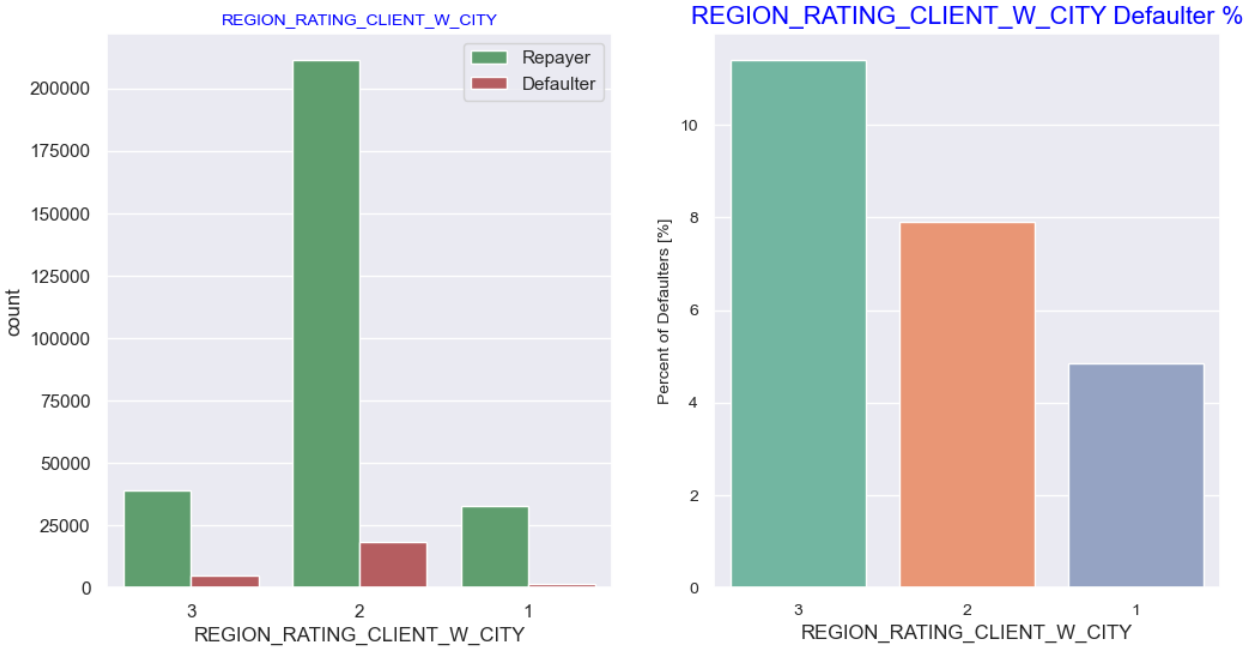


Insight:The people who submit fla_document _3 higher than the people who dont submit, There is no significant correlation between repayers and defaulters in terms of submitting document 3 as we see even if applicants have submitted the document, they have defaulted a slightly more (~9%) than who have not submitted the document (6%)

#catagorical value need to check
24 REGION_RATING_CLIENT_W_CITY
25 WEEKDAY_APPR_PROCESS_START REG_REGION_NOT_LIVE_REGION
28 REG_REGION_NOT_WORK_REGION
29 LIVE_REGION_NOT_WORK_REGION
30 REG_CITY_NOT_LIVE_CITY
31 REG_CITY_NOT_WORK_CITY
32 LIVE_CITY_NOT_WORK_CITY

In [62]:

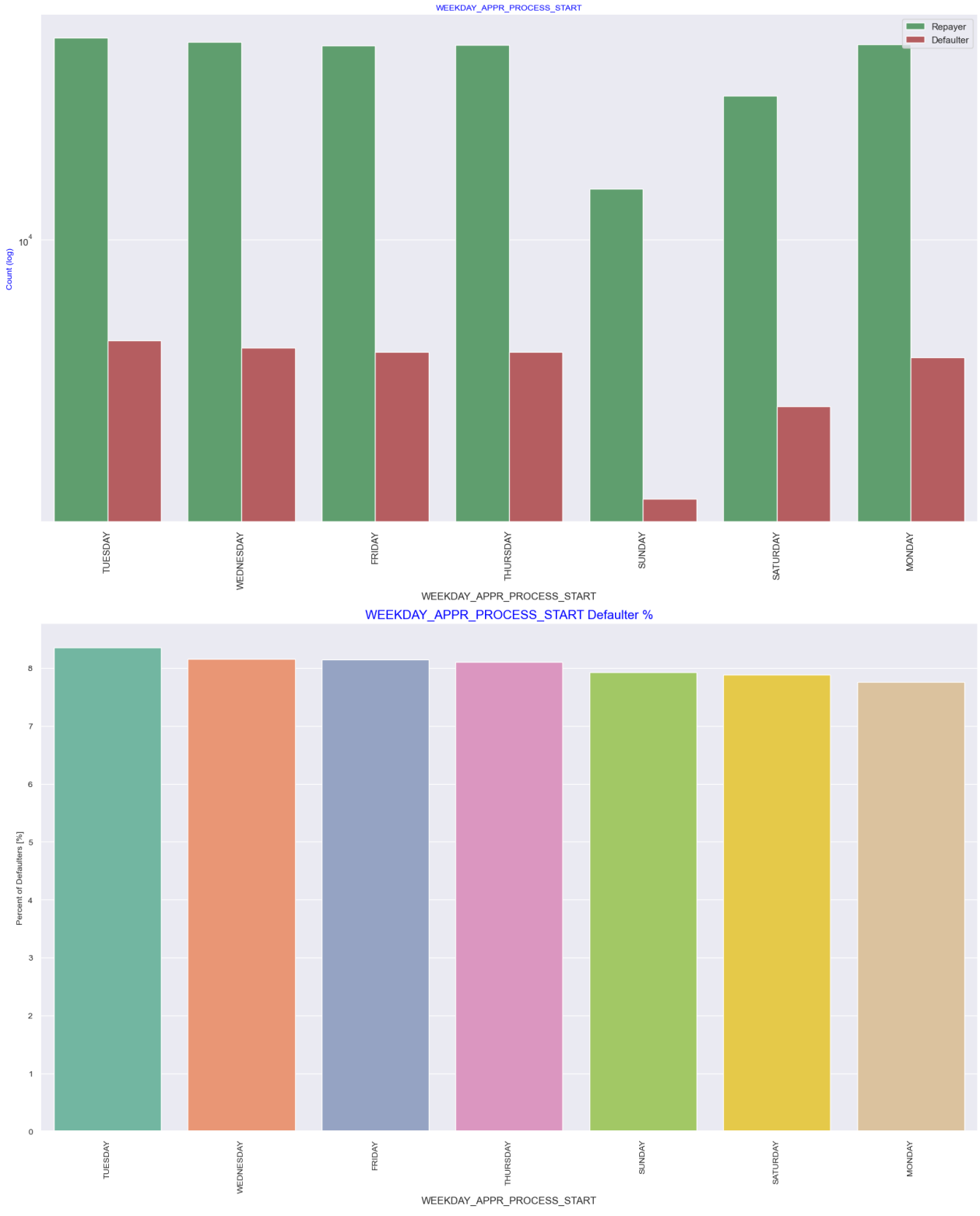
```
univariate_categorical("REGION_RATING_CLIENT_W_CITY",False,False,True)
```



Insight: the people who region rating 2are more applicant for loan, the defaulter is high in region rating 3, and least defaulter in region rating 1

In [63]:

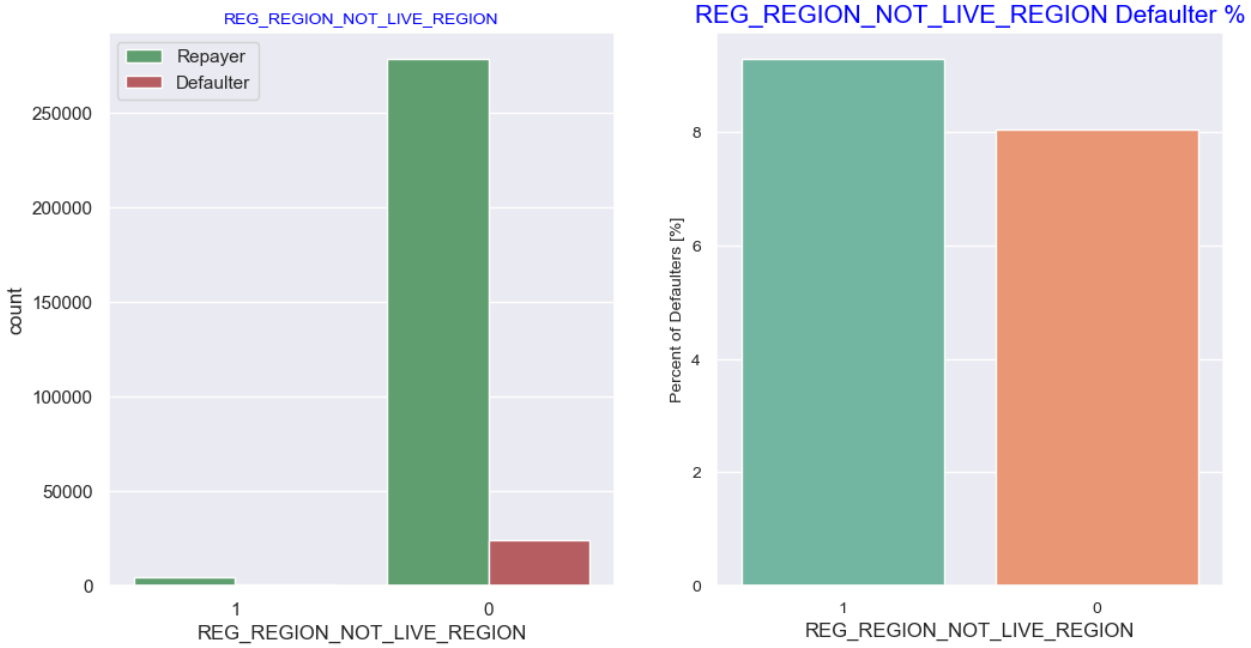
```
univariate_categorical("WEEKDAY_APPR_PROCESS_START",True,True,False)
```



Insight:there is no relatioship with this attributea nd risk of defaulter percentage

In [65]:

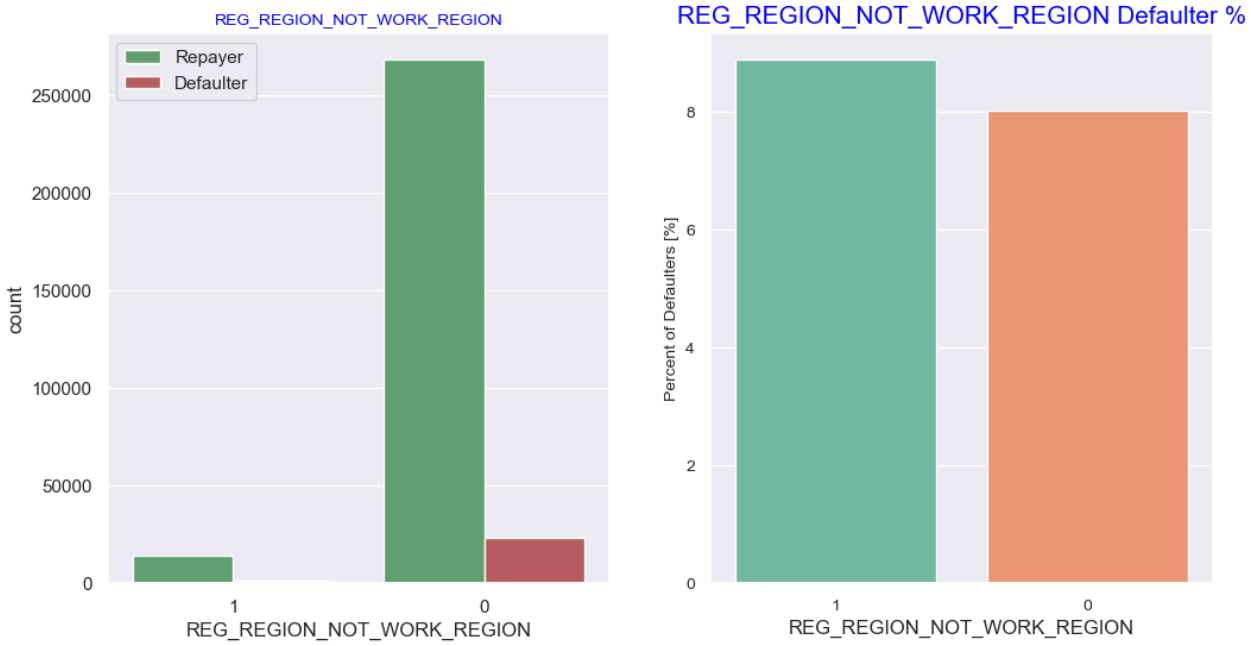
```
univariate_categorical("REG_REGION_NOT_LIVE_REGION",False,False,True)
```



insight:client adress no matched higher than the not matched. and this adress match or not match has no effects on defaulting the loan

In [66]:

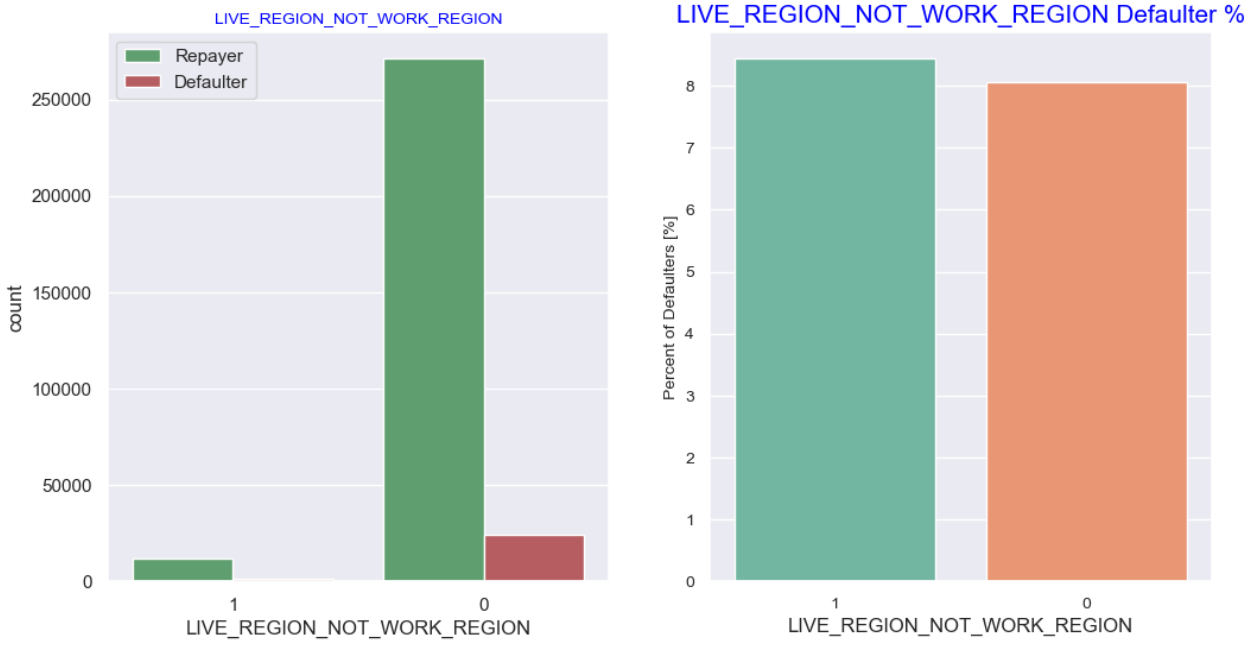
```
univariate_categorical("REG_REGION_NOT_WORK_REGION",False,False,True)
```



Insight:client's permanent address does not match the region where they work.0-match ,not match-1, match is higher than the match, but it has no effect in percent of defaulter

In [67]:

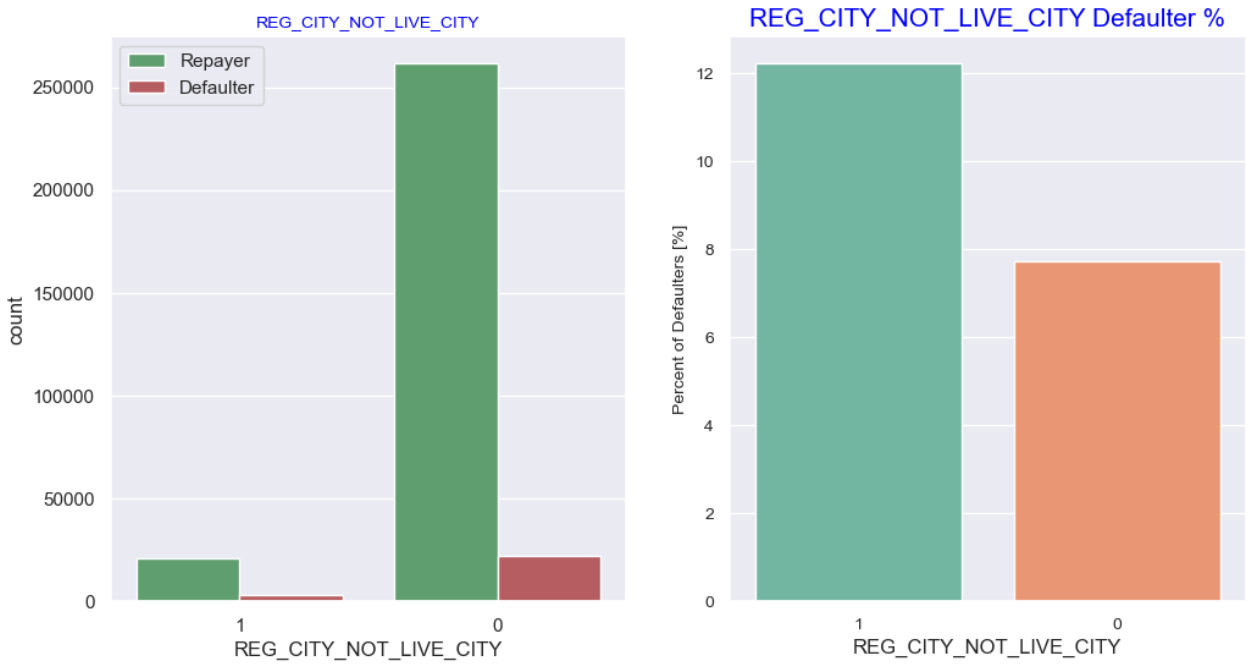
```
univariate_categorical("LIVE_REGION_NOT_WORK_REGION",False,False,True)
```



Insight:, Adress match , 0-match ,not match-1, match is higher than the match, but it has no effect in percent of defaulter

In [68]:

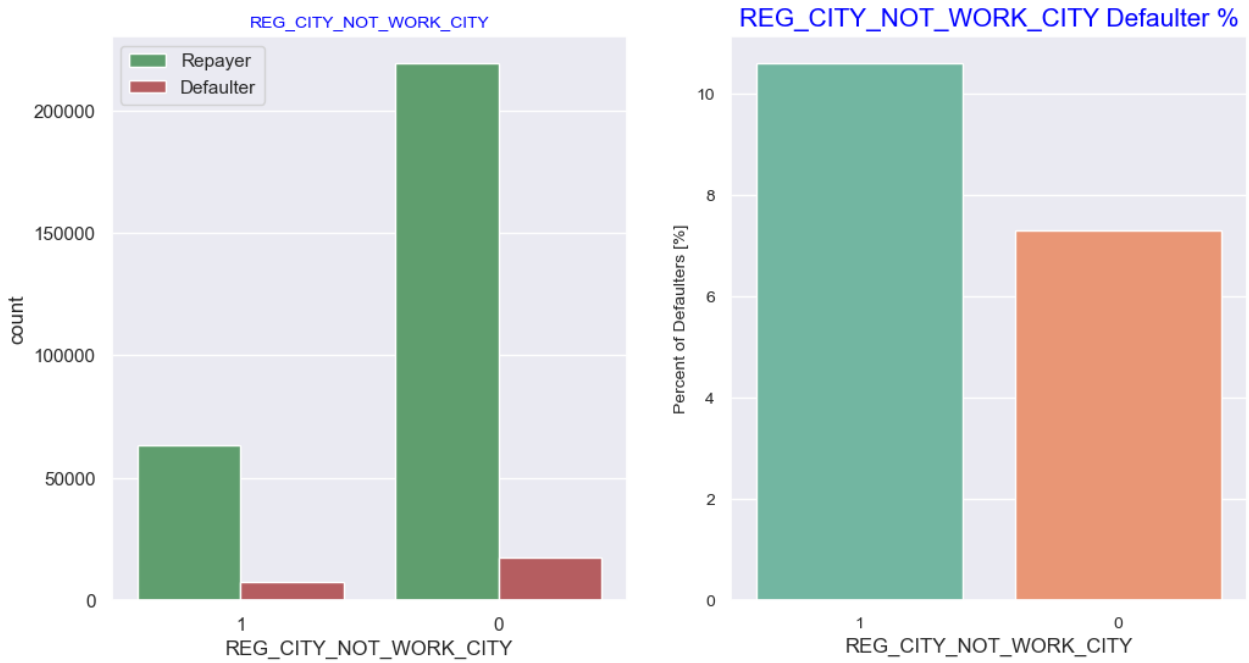
```
univariate_categorical("REG_CITY_NOT_LIVE_CITY",False,False,True)
```



Insight:, Adress match , 0-match ,not match-1, match is higher than the match, but the applicant whose address not match has effect in percent of defaulter(more than 12%)than around 7%(match address aplicant)

In [69]:

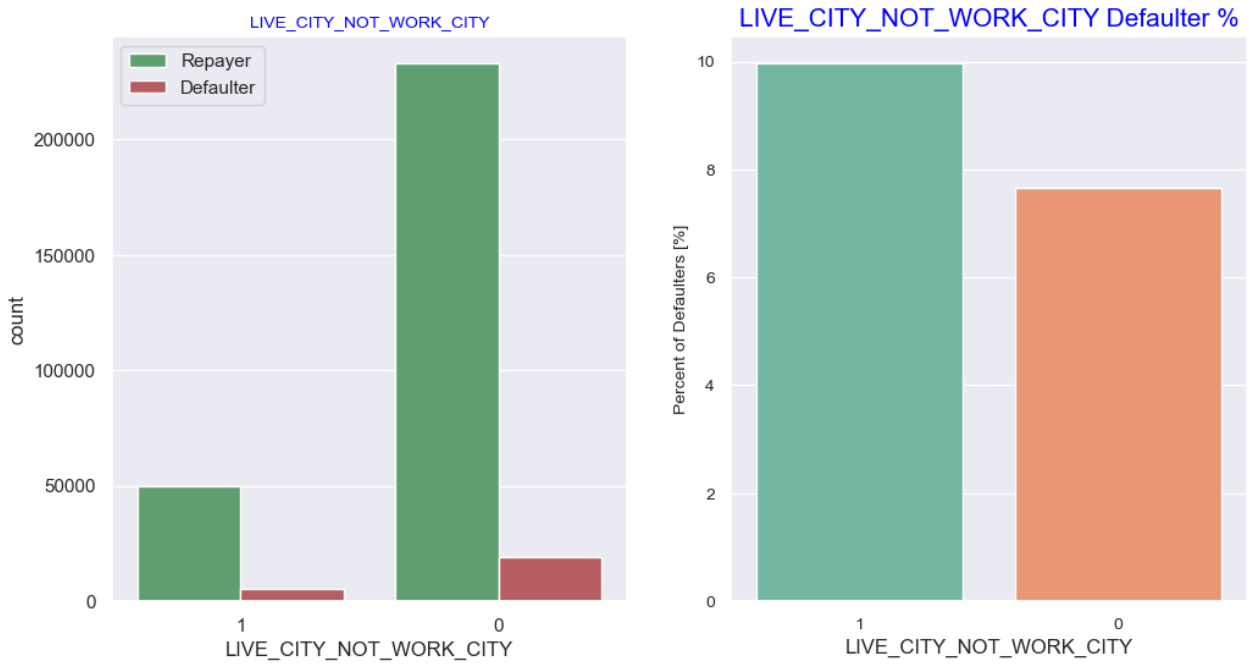
```
univariate_categorical("REG_CITY_NOT_WORK_CITY",False,False,True)
```



Insight:, Adress match , 0-match ,not match-1, match is higher than the match, but the applicant whose address not match has effect in percent of defaulter(more than 10%)than around 7%(match address aplicant)

In [70]:

```
univariate_categorical("LIVE_CITY_NOT_WORK_CITY",False,False,True)
```



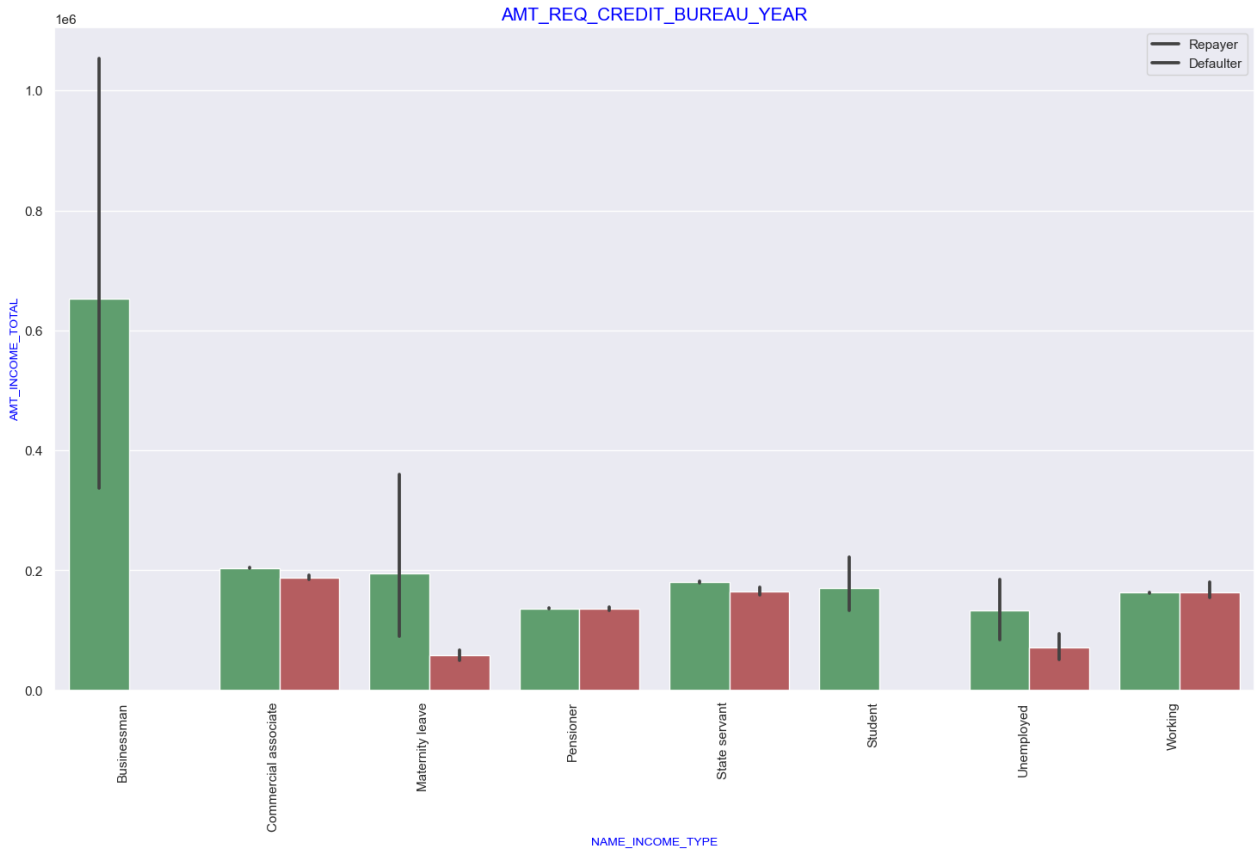
Insight:, Adress match , 0-match ,not match-1, match is higher than the match, but the applicant whose address not match has effect in percent of defaulter(around 10%)than around 7%(match address aplicant)

In [73]:

```
#Bivariate anlysis
#Define Bi-variate function
# 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 [74]:

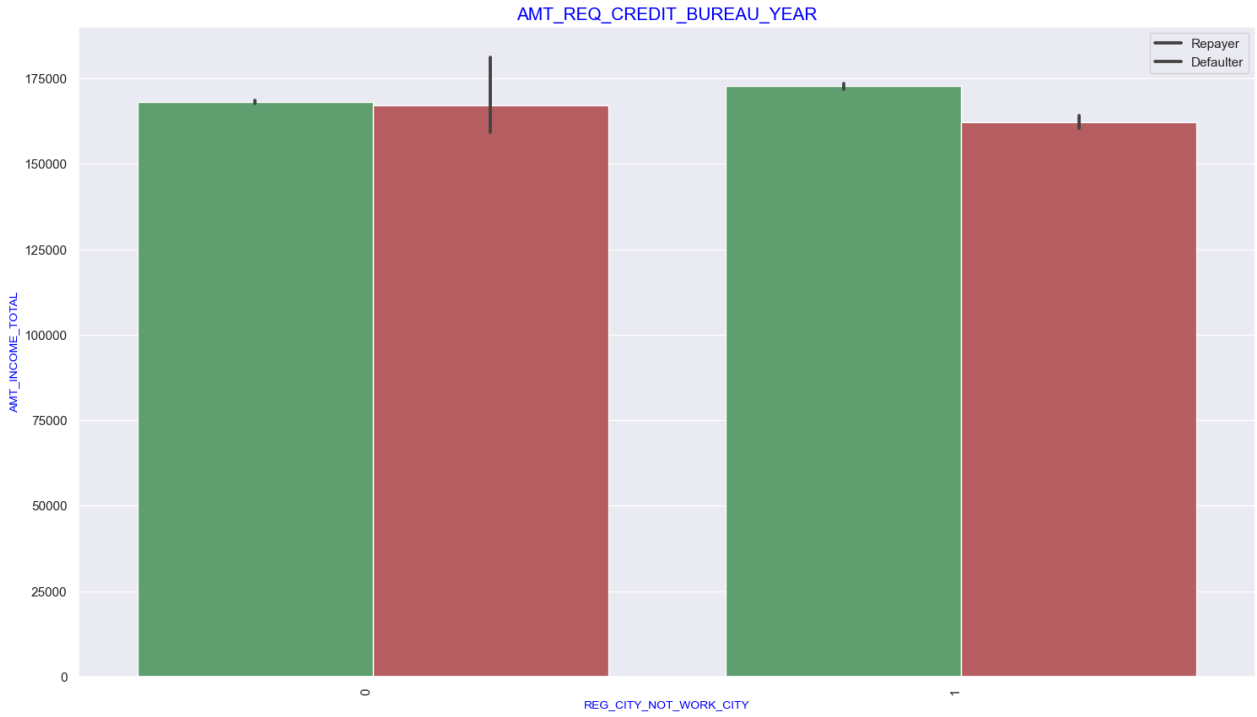
```
# Income type vs Income Amount Range
bivariate_bar("NAME_INCOME_TYPE","AMT_INCOME_TOTAL",df,"TARGET",(18,10))
```



Inferences: It can be seen that business man's income is the highest and the estimated range with default 95% confidence level seem to indicate that the income of a business man could be in the range of slightly close to 4 lakhs and slightly above 10 lakhs

In [75]:

```
# "REG_CITY_NOT_WORK_CITY "VS"LIVE_CITY_NOT_WORK_CITY
#y axis must be  anumerical value
bivariate_bar("REG_CITY_NOT_WORK_CITY","AMT_INCOME_TOTAL",df,"TARGET",(18,10))

```



Numeric Variables Analysis Bifurcating DF dataframe based on Target value 0 and 1 for correlation and other analysis Bifurcating data is a valuable technique in data analysis that supports exploratory analysis, comparative analysis, targeted analysis, and decision-making. It helps uncover insights, understand variations, and make data-driven recommendations by dividing data into meaningful segments or categories.

In [ ]:

```
df.columns
```

In [76]:

```
cols_for_correlation = ['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_RE
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR']

Repayer_df = df.loc[df['TARGET']==0, cols_for_correlation] # Repayers
Defaulter_df =df.loc[df['TARGET']==1, cols_for_correlation] # Defaulter
```

The code explanation corr_repayer = Repayer_df.corr(): This calculates the correlation matrix for the "Repayers" data. The corr() function computes the pairwise correlation of columns in the DataFrame.

corr_repayer = corr_repayer.where(np.triu(np.ones(corr_repayer.shape),k=1).astype(np.bool)): This line uses NumPy functions to create a mask to select only the upper triangular part of the correlation matrix. The np.triu() function creates an upper triangular matrix of ones, and the astype(np.bool) converts it to a boolean matrix. The where() function applies this mask to the correlation matrix, setting all values outside the upper triangle to NaN (Not a Number).

corr_df_repayer = corr_repayer.unstack().reset_index(): This line unstacks the correlation matrix, converting it into a DataFrame with three columns: "VAR1", "VAR2", and "Correlation". The variables that were correlated are represented by "VAR1" and "VAR2".

corr_df_repayer.columns =['VAR1','VAR2','Correlation']: This line assigns column names to the DataFrame created in the previous step.

corr_df_repayer.dropna(subset=["Correlation"], inplace=True): This drops rows from the DataFrame where the "Correlation" column contains NaN values. It removes any correlations that were outside the upper triangular part of the correlation matrix.

corr_df_repayer["Correlation"] = corr_df_repayer["Correlation"].abs(): This line takes the absolute values of the "Correlation" column, ensuring that all correlation values are positive.

corr_df_repayer.sort_values(by='Correlation', ascending=False, inplace=True): This sorts the DataFrame based on the "Correlation" column in descending order. The most highly correlated variables will appear at the top.

corr_df_repayer.head(10): This selects the top 10 rows of the DataFrame, showing the variables with the highest absolute correlation values.

In [77]:

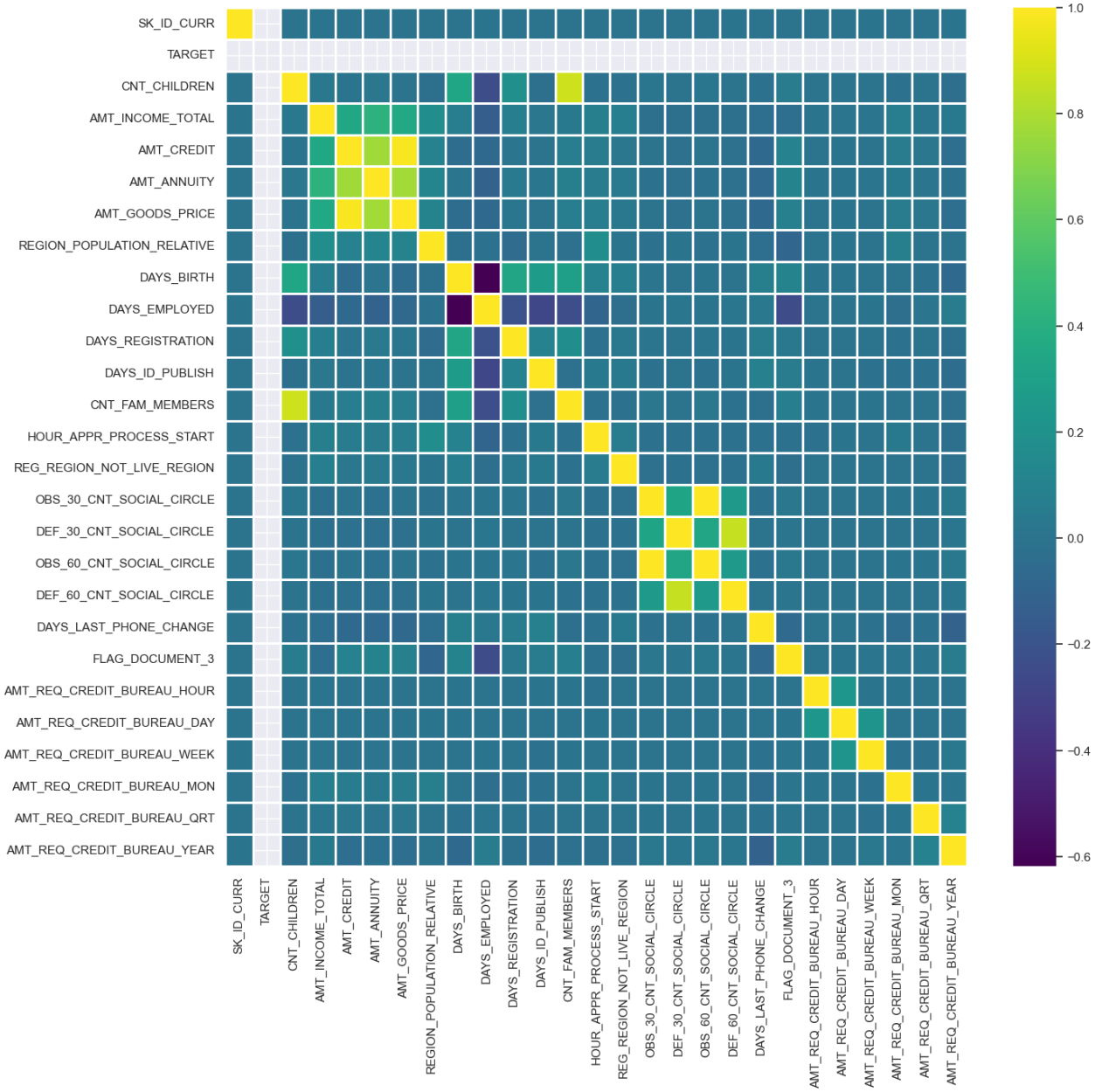
```
# Getting the top 10 correlation for the Repayers data
corr_repayer = Repayer_df.corr()
corr_repayer = corr_repayer.where(np.triu(np.ones(corr_repayer.shape),k=1).astype(np.bool))
corr_df_repayer = corr_repayer.unstack().reset_index()
corr_df_repayer.columns =['VAR1','VAR2','Correlation']
corr_df_repayer.dropna(subset = ["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[77]:

	VAR1	VAR2	Correlation
474	OBS_60_CNT_SOCIAL_CIRCLE	OBS_30_CNT_SOCIAL_CIRCLE	0.998508
166	AMT_GOODS_PRICE	AMT_CREDIT	0.987250
326	CNT_FAM_MEMBERS	CNT_CHILDREN	0.878571
502	DEF_60_CNT_SOCIAL_CIRCLE	DEF_30_CNT_SOCIAL_CIRCLE	0.859332
167	AMT_GOODS_PRICE	AMT_ANNUITY	0.776686
139	AMT_ANNUITY	AMT_CREDIT	0.771309
251	DAYS_EMPLOYED	DAYS_BIRTH	0.618048
138	AMT_ANNUITY	AMT_INCOME_TOTAL	0.418953
165	AMT_GOODS_PRICE	AMT_INCOME_TOTAL	0.349462
111	AMT_CREDIT	AMT_INCOME_TOTAL	0.342799

In [78]:

```
#see the corealtion with visualization by Heatmap
fig = plt.figure(figsize=(15,14))
ax = sns.heatmap(Repayer_df.corr(), cmap="viridis",annot=False,linewidth =1)
```



Insight: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 [79]:

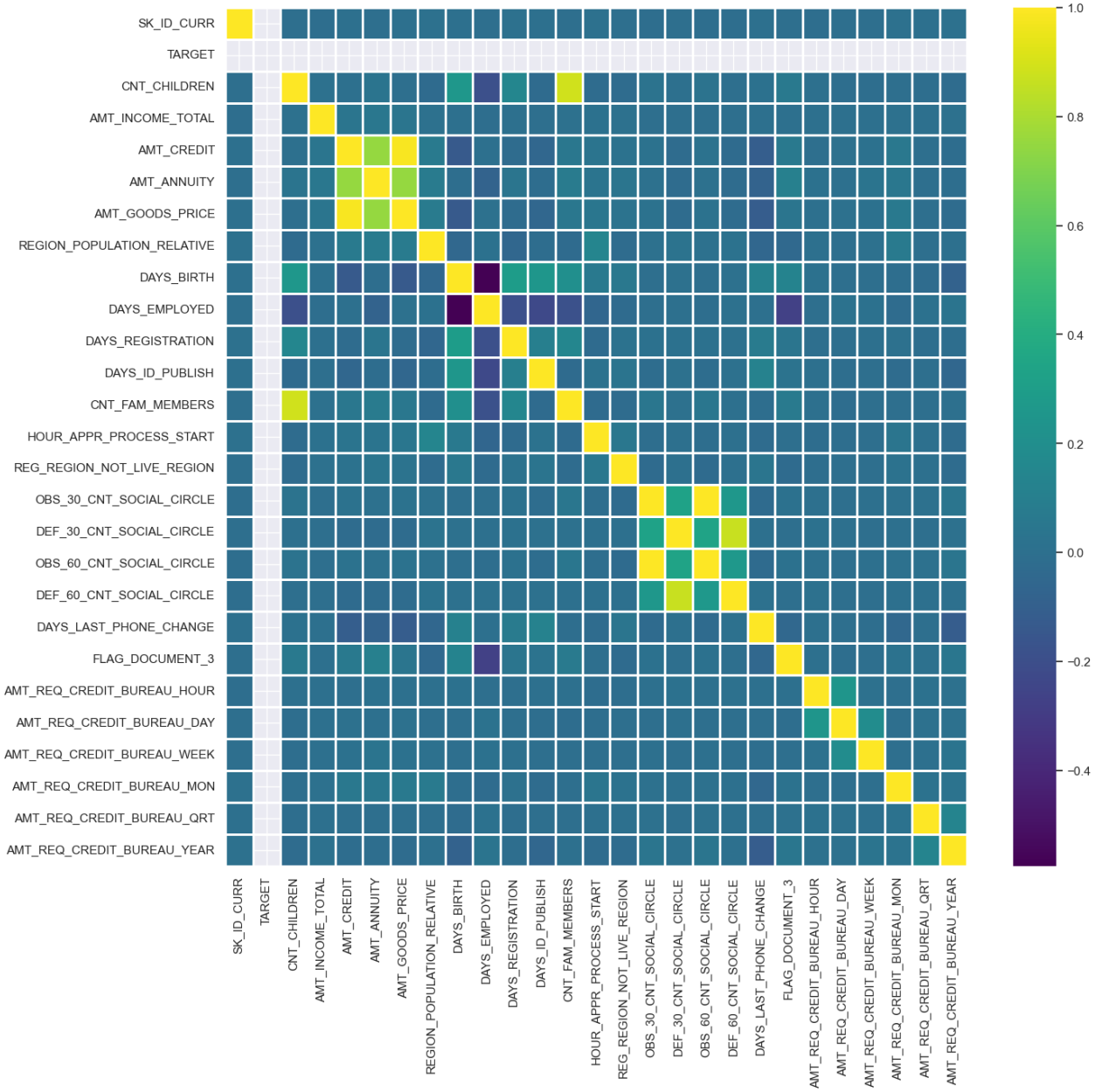
```
# Getting the top 10 correlation for the Defaulter data
corr_Defaulter = Defaulter_df.corr()
corr_Defaulter = corr_Defaulter.where(np.triu(np.ones(corr_Defaulter.shape),k=1).astype(np.bool))
corr_df_Defaulter = corr_Defaulter.unstack().reset_index()
corr_df_Defaulter.columns =['VAR1','VAR2','Correlation']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

Out[79]:

	VAR1	VAR2	Correlation
474	OBS_60_CNT_SOCIAL_CIRCLE	OBS_30_CNT_SOCIAL_CIRCLE	0.998269
166	AMT_GOODS_PRICE	AMT_CREDIT	0.983103
326	CNT_FAM_MEMBERS	CNT_CHILDREN	0.885484
502	DEF_60_CNT_SOCIAL_CIRCLE	DEF_30_CNT_SOCIAL_CIRCLE	0.868994
167	AMT_GOODS_PRICE	AMT_ANNUITY	0.752699
139	AMT_ANNUITY	AMT_CREDIT	0.752195
251	DAYS_EMPLOYED	DAYS_BIRTH	0.575097
475	OBS_60_CNT_SOCIAL_CIRCLE	DEF_30_CNT_SOCIAL_CIRCLE	0.337181
447	DEF_30_CNT_SOCIAL_CIRCLE	OBS_30_CNT_SOCIAL_CIRCLE	0.333825
278	DAYS_REGISTRATION	DAYS_BIRTH	0.289114

In [80]:

```
#see the corealtion with visualization by Heatmap
fig = plt.figure(figsize=(15,14))
ax = sns.heatmap(Defaulter_df.corr(), cmap="viridis",annot=False,linewidth =1)
```



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

Numerical varaible analysis:numeric univariate analysis provides essential insights into the distribution, patterns, and characteristics of individual numeric variables. It serves as a foundation for further data analysis, modeling, and decision-making.

Code explanation: amount = applicationDF[['AMT_INCOME_TOTAL','AMT_CREDIT','AMT_ANNUITY', 'AMT_GOODS_PRICE']]: This line selects four columns ('AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE') from the applicationDF DataFrame and assigns them to the amount variable.

fig = plt.figure(figsize=(16,12)): This line creates a new figure with a specified size of 16 inches in width and 12 inches in height. The fig variable is assigned the created figure object.

for i in enumerate(amount):: This line starts a for loop that iterates over the columns in the amount variable. The enumerate function is used to get both the index and value of each column.

plt.subplot(2,2,i[0]+1): This line creates a subplot within the figure. The subplot has a 2x2 grid, and the index of the current column (i[0]) is used to determine the position of the subplot. The i[0]+1 is added to convert the index to a subplot number starting from 1.

sns.distplot(Defaulter_df[i[1]], hist=False, color='r',label ="Defaulter"): This line plots a kernel density estimate plot (distribution plot) for the values of the current column in the Defaulter_df DataFrame. The hist=False argument specifies not to show the histogram, color='r' sets the color of the plot to red, and label="Defaulter" provides a label for the plot legend.

sns.distplot(Repayer_df[i[1]], hist=False, color='g', label ="Repayer"): This line plots a kernel density estimate plot for the values of the current column in the Repayer_df DataFrame. Similar to the previous line, it specifies the color as green and provides a label for the plot legend.

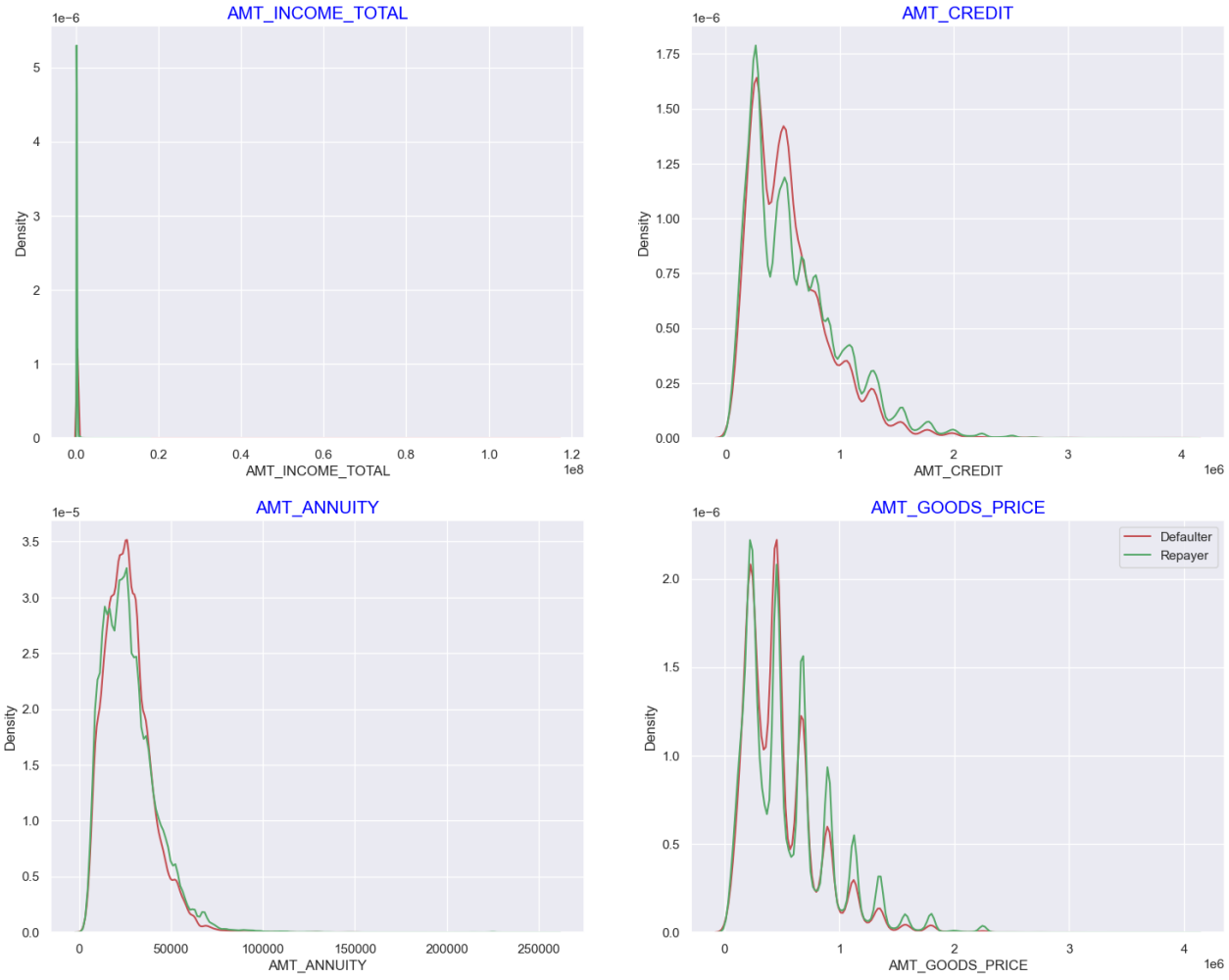
plt.title(i[1], fontdict={'fontsize' : 15, 'fontweight' : 5, 'color' : 'Blue'}): This line sets the title of the current subplot to the name of the current column (i[1]). The title is styled with a font size of 15, font weight of 5 (bold), and color blue.

plt.legend(): This line adds a legend to the plot, displaying the labels specified in the previous two lines ("Defaulter" and "Repayer").

plt.show(): This line displays the figure with all the subplots and plots.

In [81]:

```
this code , we are going to  visualize in grapph of the attribute which have higher corelation and risk of defaulting
entify distribution, comparing distribution,and detect outliar
unt = df[[ 'AMT_INCOME_TOTAL','AMT_CREDIT','AMT_ANNUITY', 'AMT_GOODS_PRICE']]

plt.figure(figsize=(18,14))

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'})

.legend()

.show()
```



Insight: 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 cannot use any of these variables in isolation to make a decision

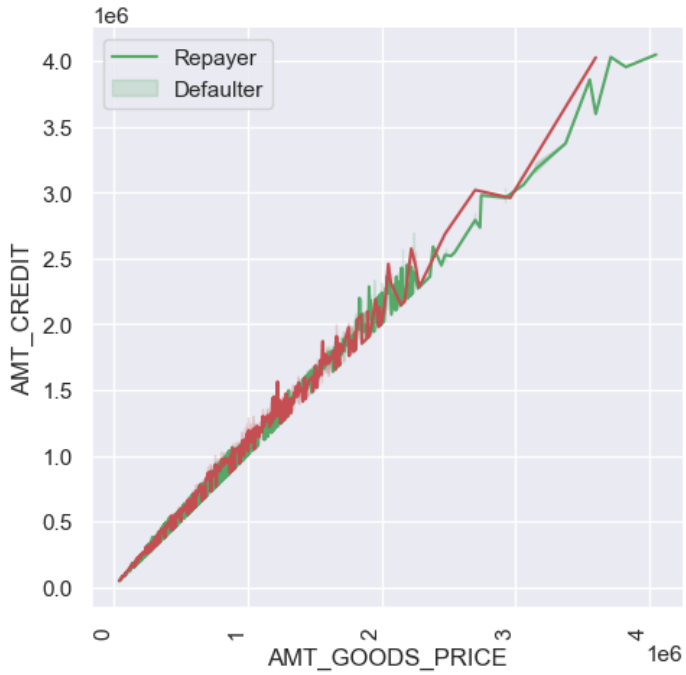
In [83]:

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

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

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

In [84]:

```
# Checking the relationship between Goods price and credit and comparing with loan repayment staus
bivariate_rel('AMT_GOODS_PRICE','AMT_CREDIT',df,"TARGET", "line", ['r','g'], False,(15,6))
```

```
<Figure size 1500x600 with 0 Axes>
```

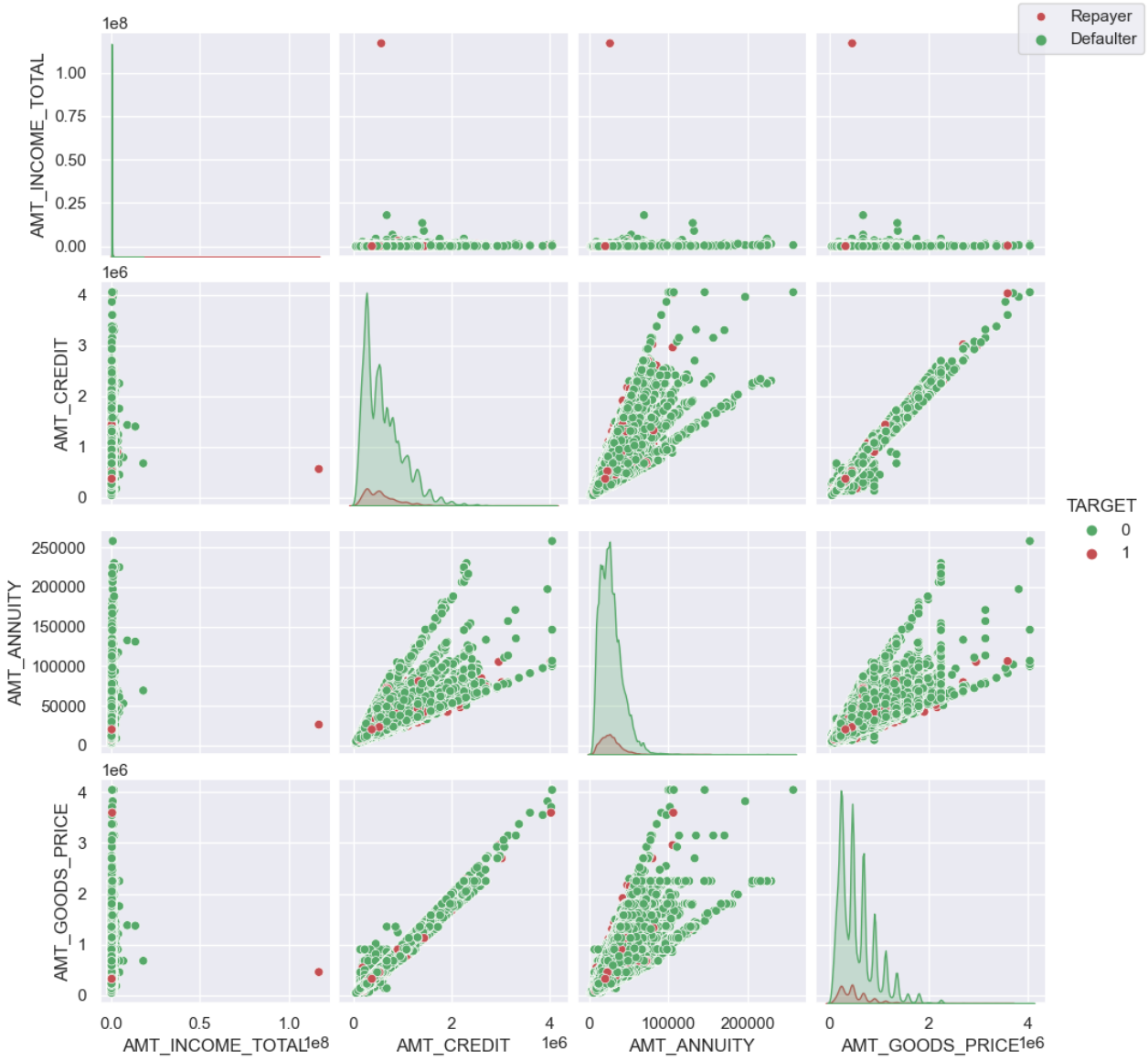


Insight:the people who have more than 3 million and good price more than 30 lakh , shows increase in defaulters

Pairplots and scatterplots can help us determine the importance or relevance of variables in relation to the target variable. If a variable exhibits a strong and distinct relationship with the target variable, it suggests that the variable is potentially important in predicting or explaining the target variable's values. On the other hand, if there is no clear relationship or the relationship is weak, it indicates that the variable may have less impact on the target variable.

In [10]:

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



Inferences: When amt_annuity >10000 amt_goods_price> 0.4M, there is a greater chance of defaulters

In [86]:

```
df1=pd.read_csv("/Users/myyntiimac/Desktop/previous_application.csv")
df1.head()
```

Out[86]:

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

In [87]:

```
1 df1.shape
```

Out[87]:

```
(1670214, 37)
```

In [88]:

```
1 df1.size
```

Out[88]:

```
61797918
```

In [89]:

```
1 df1.columns
```

Out[89]:

```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY', 'AMT_APPLICATION', 'AMT_CREDIT',
'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRICE', 'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START', 'FLAG_LA
ST_APPL_PER_CONTRACT', 'NFLAG_LAST_APPL_IN_DAY', 'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY', 'RATE_INT
EREST_PRIVILEGED', 'NAME_CASH_LOAN_PURPOSE', 'NAME_CONTRACT_STATUS', 'DAYS_DECISION', 'NAME_PAYMENT_TYP
E', 'CODE_REJECT_REASON', 'NAME_TYPE_SUITE', 'NAME_CLIENT_TYPE', 'NAME_GOODS_CATEGORY', 'NAME_PORTFOLI
O', 'NAME_PRODUCT_TYPE', 'CHANNEL_TYPE', 'SELLERPLACE_AREA', 'NAME_SELLER_INDUSTRY', 'CNT_PAYMENT', 'NAM
E_YIELD_GROUP', 'PRODUCT_COMBINATION', 'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSIO
N', 'DAYS_LAST_DUE', 'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROVAL'], dtype='object')
```

In [91]:

```
1 df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   SK_ID_PREV                   1670214 non-null  int64
 1   SK_ID_CURR                   1670214 non-null  int64
 2   NAME_CONTRACT_TYPE           1670214 non-null  object
 3   AMT_ANNUITY                  1297979 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_DOWN_PAYMENT             774370 non-null   float64
 7   AMT_GOODS_PRICE              1284699 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object
 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64
 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object
 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64
 12  RATE_DOWN_PAYMENT            774370 non-null   float64
 13  RATE_INTEREST_PRIMARY        5951 non-null     float64
 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64
 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object
 16  NAME_CONTRACT_STATUS         1670214 non-null  object
 17  DAYS_DECISION                1670214 non-null  int64
 18  NAME_PAYMENT_TYPE            1670214 non-null  object
 19  CODE_REJECT_REASON           1670214 non-null  object
 20  NAME_TYPE_SUITE              849809 non-null   object
 21  NAME_CLIENT_TYPE             1670214 non-null  object
 22  NAME_GOODS_CATEGORY          1670214 non-null  object
 23  NAME_PORTFOLIO               1670214 non-null  object
 24  NAME_PRODUCT_TYPE            1670214 non-null  object
 25  CHANNEL_TYPE                 1670214 non-null  object
 26  SELLERPLACE_AREA             1670214 non-null  int64
 27  NAME_SELLER_INDUSTRY         1670214 non-null  object
 28  CNT_PAYMENT                  1297984 non-null  float64
 29  NAME_YIELD_GROUP             1670214 non-null  object
 30  PRODUCT_COMBINATION          1669868 non-null  object
 31  DAYS_FIRST_DRAWING           997149 non-null   float64
 32  DAYS_FIRST_DUE               997149 non-null   float64
 33  DAYS_LAST_DUE_1ST_VERSION    997149 non-null   float64
 34  DAYS_LAST_DUE                997149 non-null   float64
 35  DAYS_TERMINATION             997149 non-null   float64
 36  NFLAG_INSURED_ON_APPROVAL    997149 non-null   float64
dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
```

In [92]:

```
df1.isnull().any()
```

Out[92]:

```
SK_ID_PREV                   False
SK_ID_CURR                   False
NAME_CONTRACT_TYPE           False
AMT_ANNUITY                   True
AMT_APPLICATION              False
AMT_CREDIT                    True
AMT_DOWN_PAYMENT              True
AMT_GOODS_PRICE               True
WEEKDAY_APPR_PROCESS_START   False
HOUR_APPR_PROCESS_START      False
FLAG_LAST_APPL_PER_CONTRACT  False
NFLAG_LAST_APPL_IN_DAY       False
RATE_DOWN_PAYMENT             True
RATE_INTEREST_PRIMARY         True
RATE_INTEREST_PRIVILEGED      True
NAME_CASH_LOAN_PURPOSE       False
NAME_CONTRACT_STATUS         False
DAYS_DECISION                False
NAME_PAYMENT_TYPE            False
CODE_REJECT_REASON           False
NAME_TYPE_SUITE               True
NAME_CLIENT_TYPE             False
NAME_GOODS_CATEGORY          False
NAME_PORTFOLIO               False
NAME_PRODUCT_TYPE            False
CHANNEL_TYPE                 False
SELLERPLACE_AREA             False
NAME_SELLER_INDUSTRY         False
CNT_PAYMENT                   True
NAME_YIELD_GROUP             False
PRODUCT_COMBINATION           True
DAYS_FIRST_DRAWING            True
DAYS_FIRST_DUE                True
DAYS_LAST_DUE_1ST_VERSION     True
DAYS_LAST_DUE                 True
DAYS_TERMINATION              True
NFLAG_INSURED_ON_APPROVAL     True
dtype: bool
```

In [93]:

```
df1.describe()
```

Out[93]:

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

In [94]:

```
#null value calculation
!pip install missingno
```

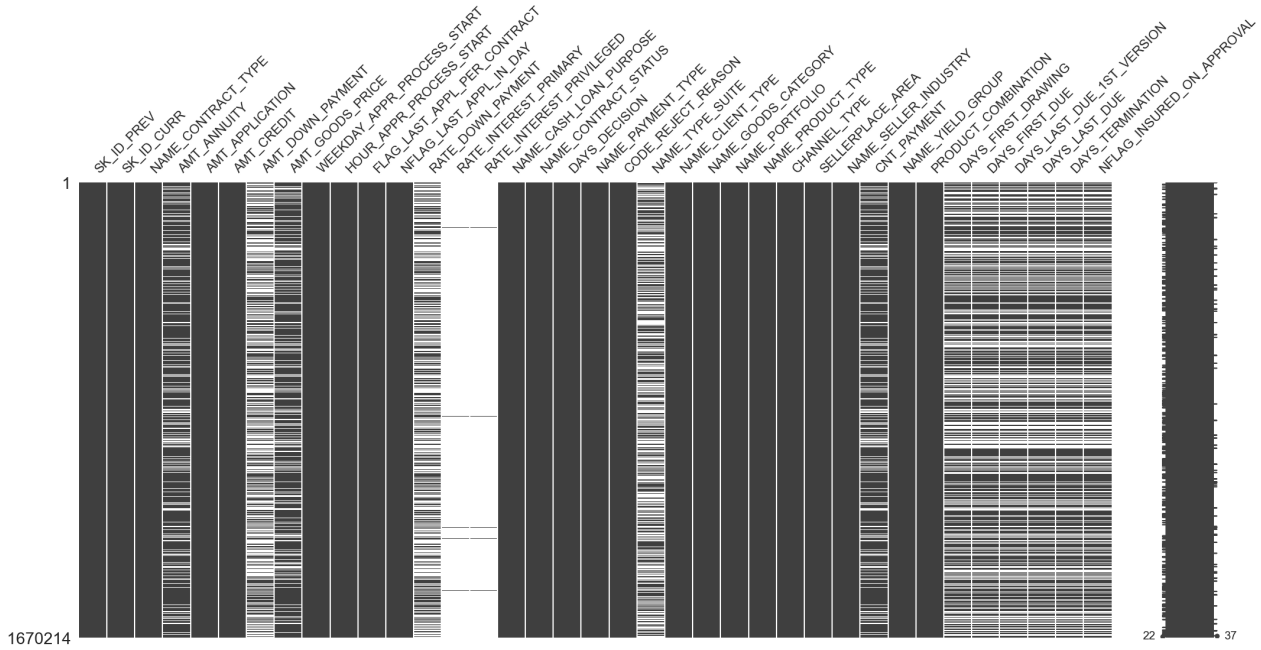
```
Requirement already satisfied: missingno in ./anaconda3/lib/python3.10/site-packages (0.5.2)
Requirement already satisfied: seaborn in ./anaconda3/lib/python3.10/site-packages (from missingno) (0.1
2.2)
Requirement already satisfied: matplotlib in ./anaconda3/lib/python3.10/site-packages (from missingno)
(3.7.0)
Requirement already satisfied: scipy in ./anaconda3/lib/python3.10/site-packages (from missingno) (1.10.
0)
Requirement already satisfied: numpy in ./anaconda3/lib/python3.10/site-packages (from missingno) (1.23.
5)
Requirement already satisfied: fonttools>=4.22.0 in ./anaconda3/lib/python3.10/site-packages (from matpl
otlib->missingno) (4.25.0)
Requirement already satisfied: python-dateutil>=2.7 in ./anaconda3/lib/python3.10/site-packages (from ma
tplotlib->missingno) (2.8.2)
Requirement already satisfied: packaging>=20.0 in ./anaconda3/lib/python3.10/site-packages (from matplot
lib->missingno) (22.0)
Requirement already satisfied: kiwisolver>=1.0.1 in ./anaconda3/lib/python3.10/site-packages (from matpl
otlib->missingno) (1.4.4)
Requirement already satisfied: pyparsing>=2.3.1 in ./anaconda3/lib/python3.10/site-packages (from matplo
tlib->missingno) (3.0.9)
Requirement already satisfied: contourpy>=1.0.1 in ./anaconda3/lib/python3.10/site-packages (from matplo
tlib->missingno) (1.0.5)
Requirement already satisfied: cycler>=0.10 in ./anaconda3/lib/python3.10/site-packages (from matplotlib
->missingno) (0.11.0)
Requirement already satisfied: pillow>=6.2.0 in ./anaconda3/lib/python3.10/site-packages (from matplotli
b->missingno) (9.4.0)
Requirement already satisfied: pandas>=0.25 in ./anaconda3/lib/python3.10/site-packages (from seaborn->m
issingno) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in ./anaconda3/lib/python3.10/site-packages (from pandas>=0.
25->seaborn->missingno) (2022.7)
Requirement already satisfied: six>=1.5 in ./anaconda3/lib/python3.10/site-packages (from python-dateuti
l>=2.7->matplotlib->missingno) (1.16.0)
```

In [95]:

```
import missingno as mn
mn.matrix(df1)
```

Out[95]:

```
<Axes: >
```



Insight:In df1 , there is no of columns contain the missing value , lets try to fid missing value percentage

In [96]:

```
# % null value in each column
round(df1.isnull().sum() / df1.shape[0] * 100.00,2)
```

Out[96]:

```
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
```

insight:there is no of column have more than 40%null values, Lets see this in plot , column name vers percentage

In [97]:

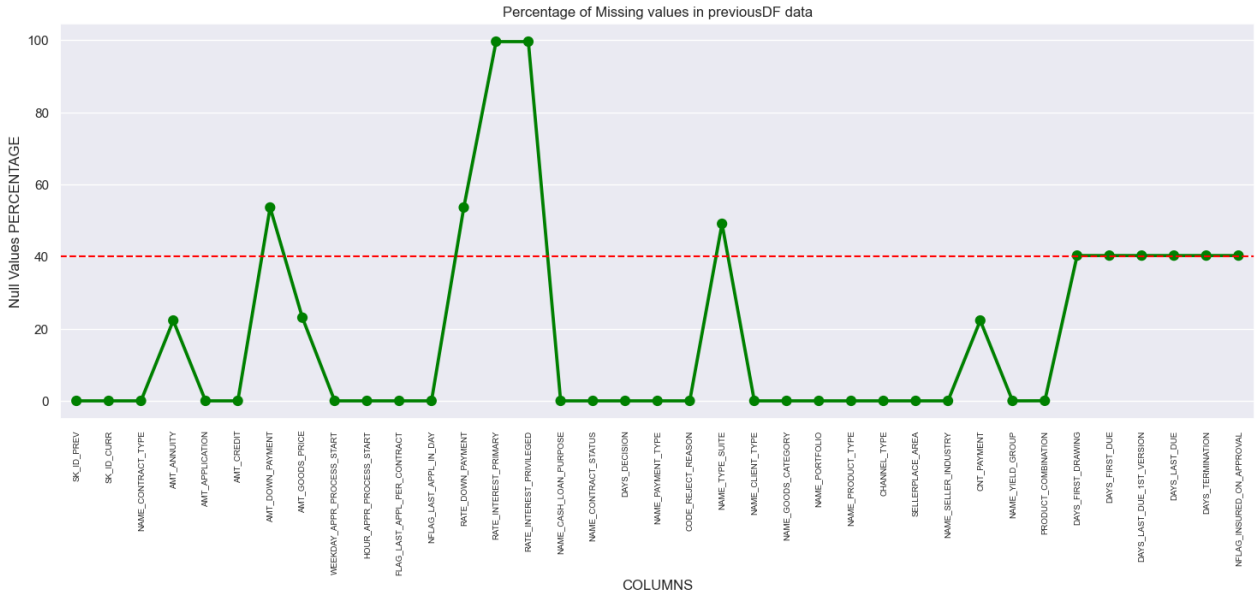
```
# for this  plot we need to make  a new dataframe and assign the column name
null_df1 = pd.DataFrame((df1.isnull().sum())*100/df1.shape[0]).reset_index()
null_df1.columns = ['Column Name', 'Null Values Percentage']
null_df1.head()
```

Out[97]:

	Column Name	Null Values Percentage
0	SK_ID_PREV	0.000000
1	SK_ID_CURR	0.000000
2	NAME_CONTRACT_TYPE	0.000000
3	AMT_ANNUITY	22.286665
4	AMT_APPLICATION	0.000000

In [98]:

```
# then make the plot where x= name, y=percentage
fig = plt.figure(figsize=(18,6))
ax = sns.pointplot(x="Column Name",y="Null Values Percentage",data=null_df1,color ='green')
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()
```



from that point plot, its clearly seen there is 11 variable which have more than 40% null value, Lets treat this

In [99]:

```
# more than or equal to 40% empty rows columns
nullcol_40_df1 = null_df1[null_df1["Null Values Percentage"]>=40]
nullcol_40_df1
```

Out[99]:

	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 [100]:

```
nullcol_not40_df2 = null_df1[null_df1["Null Values Percentage"]<40]
nullcol_not40_df2
```

Out[100]:

	Column Name	Null Values Percentage
0	SK_ID_PREV	0.000000
1	SK_ID_CURR	0.000000
2	NAME_CONTRACT_TYPE	0.000000
3	AMT_ANNUITY	22.286665
4	AMT_APPLICATION	0.000000
5	AMT_CREDIT	0.000060
7	AMT_GOODS_PRICE	23.081773
8	WEEKDAY_APPR_PROCESS_START	0.000000
9	HOUR_APPR_PROCESS_START	0.000000
10	FLAG_LAST_APPL_PER_CONTRACT	0.000000
11	NFLAG_LAST_APPL_IN_DAY	0.000000
15	NAME_CASH_LOAN_PURPOSE	0.000000
16	NAME_CONTRACT_STATUS	0.000000
17	DAYS_DECISION	0.000000
18	NAME_PAYMENT_TYPE	0.000000
19	CODE_REJECT_REASON	0.000000
21	NAME_CLIENT_TYPE	0.000000
22	NAME_GOODS_CATEGORY	0.000000
23	NAME_PORTFOLIO	0.000000
24	NAME_PRODUCT_TYPE	0.000000
25	CHANNEL_TYPE	0.000000
26	SELLERPLACE_AREA	0.000000
27	NAME_SELLER_INDUSTRY	0.000000
28	CNT_PAYMENT	22.286366
29	NAME_YIELD_GROUP	0.000000
30	PRODUCT_COMBINATION	0.020716

In [101]:

```
nullcol_not40_df2["Column Name"].to_list()


```

Out[101]:

```
['SK_ID_PREV',
 'SK_ID_CURR',
 'NAME_CONTRACT_TYPE',
 'AMT_ANNUITY',
 'AMT_APPLICATION',
 'AMT_CREDIT',
 'AMT_GOODS_PRICE',
 'WEEKDAY_APPR_PROCESS_START',
 'HOUR_APPR_PROCESS_START',
 'FLAG_LAST_APPL_PER_CONTRACT',
 'NFLAG_LAST_APPL_IN_DAY',
 'NAME_CASH_LOAN_PURPOSE',
 'NAME_CONTRACT_STATUS',
 'DAYS_DECISION',
 'NAME_PAYMENT_TYPE',
 'CODE_REJECT_REASON',
 'NAME_CLIENT_TYPE',
 'NAME_GOODS_CATEGORY',
 'NAME_PORTFOLIO',
 'NAME_PRODUCT_TYPE',
 'CHANNEL_TYPE',
 'SELLERPLACE_AREA',
 'NAME_SELLER_INDUSTRY',
 'CNT_PAYMENT',
 'NAME_YIELD_GROUP',
 'PRODUCT_COMBINATION']
```

In [102]:

```
df1.head()
```

Out[102]:

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

Insight:11 columns have more than 40% null value, this column can be deleted, Lets try to find there is any other column we can also deleted

In [103]:

```
df2=df1[['SK_ID_PREV',
 'SK_ID_CURR',
 'AMT_ANNUITY',
 'AMT_APPLICATION',
 'AMT_CREDIT',
 'AMT_GOODS_PRICE',
 'HOUR_APPR_PROCESS_START',
 'DAYS_DECISION',
 'NAME_CLIENT_TYPE',
 'NAME_GOODS_CATEGORY',
 'NAME_PORTFOLIO',
 'SELLERPLACE_AREA',
 'CNT_PAYMENT']]
# Perform correlation analysis
correlation_matrix = df2.corr()

print("Correlation matrix:")
print(correlation_matrix)
```

```
Correlation matrix:
                         SK_ID_PREV  SK_ID_CURR  AMT_ANNUITY  AMT_APPLICATION  AMT_CREDIT  AMT_GOODS_PRI
CE  HOUR_APPR_PROCESS_START  DAYS_DECISION  SELLERPLACE_AREA  CNT_PAYMENT
SK_ID_PREV                    1.000000   -0.000321     0.011459         0.003302    0.003659         0.0152
93                -0.002652       0.019100         -0.001079     0.015589
SK_ID_CURR                   -0.000321    1.000000     0.000577         0.000280    0.000195         0.0003
69                 0.002842      -0.000637          0.001265     0.000031
AMT_ANNUITY                   0.011459    0.000577     1.000000         0.808872    0.816429         0.8208
95                -0.036201       0.279051         -0.015027     0.394535
AMT_APPLICATION               0.003302    0.000280     0.808872         1.000000    0.975824         0.9998
84                -0.014415       0.133660         -0.007649     0.680630
AMT_CREDIT                    0.003659    0.000195     0.816429         0.975824    1.000000         0.9930
87                -0.021039       0.133763         -0.009567     0.674278
AMT_GOODS_PRICE               0.015293    0.000369     0.820895         0.999884    0.993087         1.0000
00                -0.045267       0.290422         -0.015842     0.672129
HOUR_APPR_PROCESS_START   -0.002652    0.002842    -0.036201        -0.014415   -0.021039        -0.0452
67                 1.000000      -0.039962          0.015671    -0.055511
DAYS_DECISION                 0.019100   -0.000637     0.279051         0.133660    0.133763         0.2904
22                -0.039962       1.000000         -0.018382     0.246453
SELLERPLACE_AREA             -0.001079    0.001265    -0.015027        -0.007649   -0.009567        -0.0158
42                 0.015671      -0.018382          1.000000    -0.010646
CNT_PAYMENT                   0.015589    0.000031     0.394535         0.680630    0.674278         0.6721
29                -0.055511       0.246453         -0.010646     1.000000
```

In [104]:

```
# Step 1: Look for Strong Correlations
strong_correlations = correlation_matrix.abs() > 0.6
strong_pairs = [(correlation_matrix.index[i], correlation_matrix.columns[j])
                for i, j in zip(*np.where(strong_correlations))]
print("Pairs of attributes with strong correlations:")
for pair in strong_pairs:
    print(pair)
```

```
Pairs of attributes with strong correlations:
('SK_ID_PREV', 'SK_ID_PREV')
('SK_ID_CURR', 'SK_ID_CURR')
('AMT_ANNUITY', 'AMT_ANNUITY')
('AMT_ANNUITY', 'AMT_APPLICATION')
('AMT_ANNUITY', 'AMT_CREDIT')
('AMT_ANNUITY', 'AMT_GOODS_PRICE')
('AMT_APPLICATION', 'AMT_ANNUITY')
('AMT_APPLICATION', 'AMT_APPLICATION')
('AMT_APPLICATION', 'AMT_CREDIT')
('AMT_APPLICATION', 'AMT_GOODS_PRICE')
('AMT_APPLICATION', 'CNT_PAYMENT')
('AMT_CREDIT', 'AMT_ANNUITY')
('AMT_CREDIT', 'AMT_APPLICATION')
('AMT_CREDIT', 'AMT_CREDIT')
('AMT_CREDIT', 'AMT_GOODS_PRICE')
('AMT_CREDIT', 'CNT_PAYMENT')
('AMT_GOODS_PRICE', 'AMT_ANNUITY')
('AMT_GOODS_PRICE', 'AMT_APPLICATION')
('AMT_GOODS_PRICE', 'AMT_CREDIT')
('AMT_GOODS_PRICE', 'AMT_GOODS_PRICE')
('AMT_GOODS_PRICE', 'CNT_PAYMENT')
('HOUR_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START')
('DAYS_DECISION', 'DAYS_DECISION')
('SELLERPLACE_AREA', 'SELLERPLACE_AREA')
('CNT_PAYMENT', 'AMT_APPLICATION')
('CNT_PAYMENT', 'AMT_CREDIT')
('CNT_PAYMENT', 'AMT_GOODS_PRICE')
('CNT_PAYMENT', 'CNT_PAYMENT')
```

Insight:we found in null value less than 40 % attribute, where attribute contain numerical value have high corelation each other . and found some attribute have only one unique value such as FLAG_LAST_APPL_PER_CONTRACT, NFLAG_LAST_APPL_IN_DAY, and other 2 attribute name WEEKDAY_APPR_PROCESS_START, HOUR_APPR_PROCESS_START which have no effect on loan defaulter, so we can remove this 4 column with 11 column which have high null value more than 40%

In [105]:

```
Unnecessary_att = ['WEEKDAY_APPR_PROCESS_START','HOUR_APPR_PROCESS_START',
                       'FLAG_LAST_APPL_PER_CONTRACT','NFLAG_LAST_APPL_IN_DAY']
```

In [106]:

```
Unwanted_previous = nullcol_40_df1["Column Name"].tolist()
Unwanted_previous
```

Out[106]:

```
['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 [107]:

```
combinedunwanted=Unnecessary_att+Unwanted_previous
combinedunwanted
```

Out[107]:

```
['WEEKDAY_APPR_PROCESS_START',
 'HOUR_APPR_PROCESS_START',
 'FLAG_LAST_APPL_PER_CONTRACT',
 'NFLAG_LAST_APPL_IN_DAY',
 '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 [108]:

```
df1.drop(columns=combinedunwanted, inplace=True)
df1.head()
```

Out[108]:

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

In [109]:

```
df1.shape
```

Out[109]:

```
(1670214, 22)
```

In [110]:

```
df1.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 [ ]:

```
#standersization the value in df1
#converting the column contain  negative value to positive
#convert column data type object to catagory

```

In [111]:

```
#we find the day of decision and sellerprice _area contain negative values
# so convert those column to positive
#Converting negative days to positive days
columns_to_convert = ['DAYS_DECISION', 'SELLERPLACE_AREA']
df1[columns_to_convert] = df1[columns_to_convert].abs()


```

In [112]:

```
df1.head()
```

Out[112]:

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

In [113]:

```

# Define the bin edges for the age groups
bin_edges = range(0, 4000, 400)

# Define the labels for the age groups
bin_labels = [f'{start}-{start+400}' for start in bin_edges[:-1]]

# Use pd.cut() to bin the 'DAYS_DECISION' column into age groups
df1['DAYS_DECISION_GROUP'] = pd.cut(df1['DAYS_DECISION'], bins=bin_edges, labels=bin_labels, right=False)

```

In [114]:

```
df1['DAYS_DECISION_GROUP'].value_counts(normalize=True)*100
```

Out[114]:

```
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
3200-3600     0.000000
Name: DAYS_DECISION_GROUP, dtype: float64
```

Insight:37% of loan aplicant applied for new loan in 0 to 400 dayas

In [ ]:

```
df1.info()
```

In [115]:

```
# Lets convert all catgory and object attribute to catagory
Catgorical_col_p = ['NAME_CASH_LOAN_PURPOSE','NAME_CONTRACT_STATUS','NAME_PAYMENT_TYPE',
                    'CODE_REJECT_REASON','NAME_CLIENT_TYPE','NAME_GOODS_CATEGORY','NAME_PORTFOLIO',
                   'NAME_PRODUCT_TYPE','CHANNEL_TYPE','NAME_SELLER_INDUSTRY','NAME_YIELD_GROUP','PRODUCT_COMBINAT
                    'NAME_CONTRACT_TYPE','DAYS_DECISION_GROUP']
for col in Catgorical_col_p:
    df1[col] =pd.Categorical(df1[col])
```

In [116]:

```
df1.info()

```

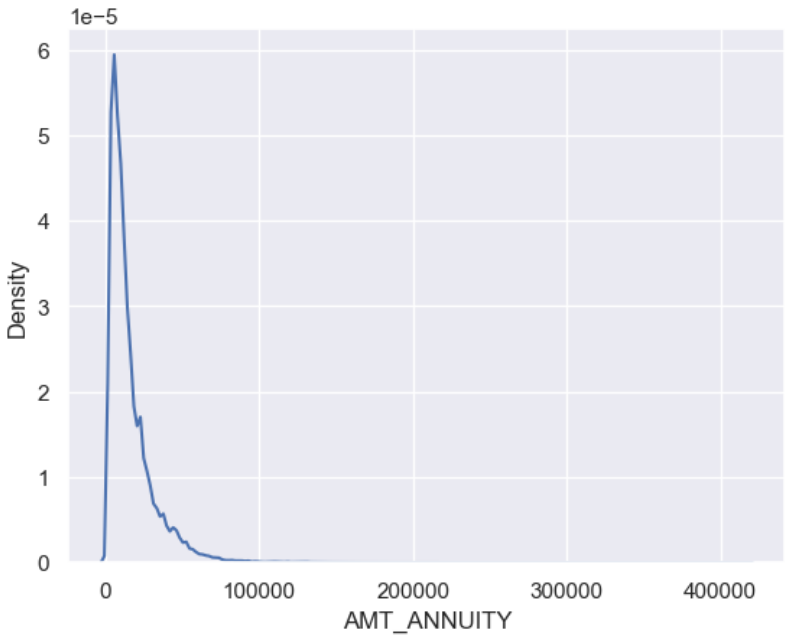
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 23 columns):
 #   Column                  Non-Null Count    Dtype   
---  ------                  --------------    -----   
 0   SK_ID_PREV              1670214 non-null  int64   
 1   SK_ID_CURR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

```
#now fill the missing value with imputation.
#AMT_ANNUITY''AMT_GOODS_PRICE' CNT_PAYMENT , This attribute has high null value, lets check the uni distribution
# and check which strategy is best for imputation
```

In [117]:

```

sns.kdeplot(df1['AMT_ANNUITY'])
plt.show()
```



we see AMT_annuity is positively skewed means there is outliar, so mean is not apprpriate (its can be affects by outliar). median would be good

In [154]:

```
df1['AMT_ANNUITY'].fillna(df1['AMT_ANNUITY'].median(),inplace = True)
```

In [155]:

```
df1['AMT_ANNUITY'].isnull().any()
```

Out[155]:

```
False
```

In [178]:

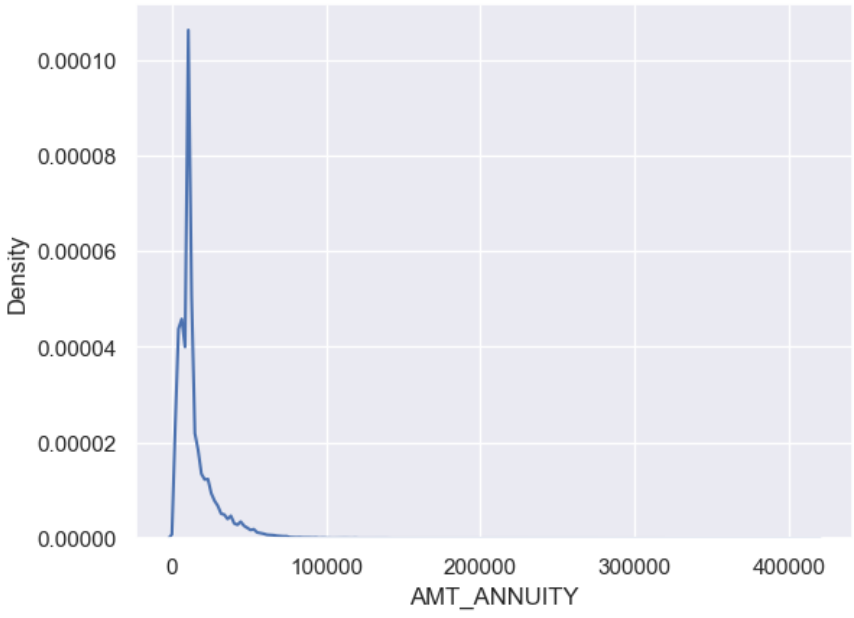
```
df1.isnull().sum()
```

Out[178]:

```
SK_ID_PREV                  0
SK_ID_CURR                  0
NAME_CONTRACT_TYPE          0
AMT_ANNUITY                 0
AMT_APPLICATION             0
AMT_CREDIT                  1
AMT_GOODS_PRICE             0
NAME_CASH_LOAN_PURPOSE      0
NAME_CONTRACT_STATUS        0
DAYS_DECISION               0
NAME_PAYMENT_TYPE           0
CODE_REJECT_REASON          0
NAME_CLIENT_TYPE            0
NAME_GOODS_CATEGORY         0
NAME_PORTFOLIO              0
NAME_PRODUCT_TYPE           0
CHANNEL_TYPE                0
SELLERPLACE_AREA            0
NAME_SELLER_INDUSTRY        0
CNT_PAYMENT                 0
NAME_YIELD_GROUP            0
PRODUCT_COMBINATION       346
DAYS_DECISION_GROUP         0
dtype: int64
```

In [149]:

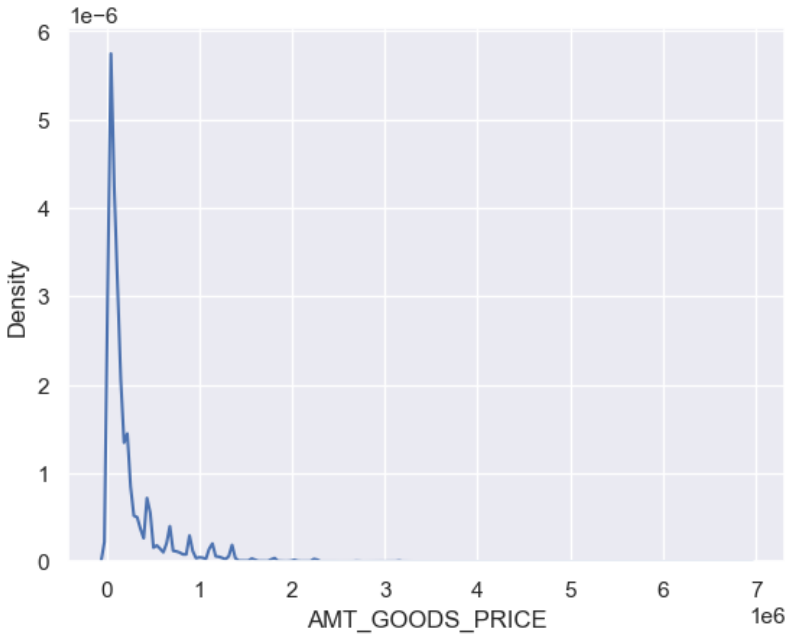
```
#Lets check again, is make diffence
sns.kdeplot(df1['AMT_ANNUITY'])
plt.show()
```



Insight: we can see there is slight diffence before imputing and after imputing

In [119]:

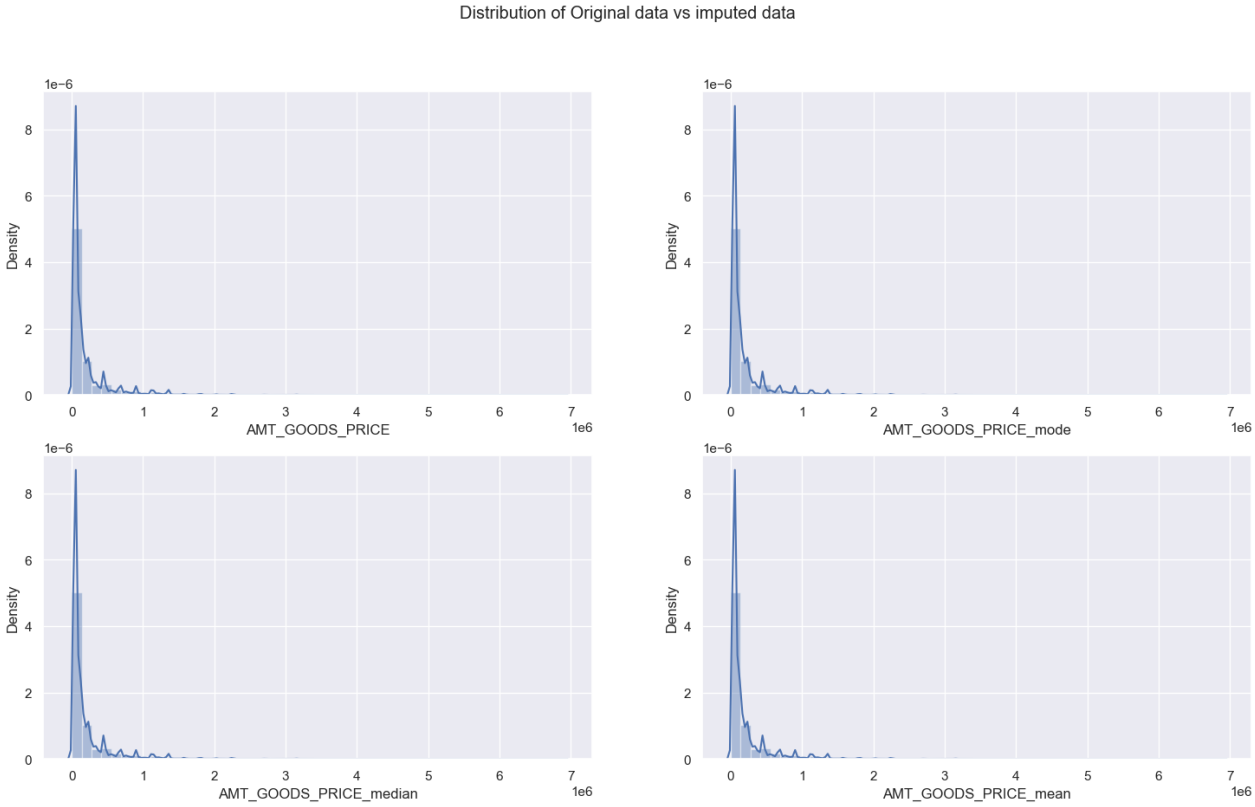
```
# we will create distributionplot using kdeplot and non_null value
sns.kdeplot(df1['AMT_GOODS_PRICE'][pd.notnull(df1['AMT_GOODS_PRICE'])])
plt.show()
```



Insght: in this plot, we saw there is several peaks , we dont know which strategy will be best, for this we need create datafame containing 3 imputation strategy and make the plot using differnt strategy column and compare with orginal plot

In [159]:

```
df3 = pd.DataFrame() # new dataframe with columns imputed with mode, median and mean
df3['AMT_GOODS_PRICE_mode'] = df1['AMT_GOODS_PRICE'].fillna(df1['AMT_GOODS_PRICE'].mode()[0])
df3['AMT_GOODS_PRICE_median'] = df1['AMT_GOODS_PRICE'].fillna(df1['AMT_GOODS_PRICE'].median())
df3['AMT_GOODS_PRICE_mean'] = df1['AMT_GOODS_PRICE'].fillna(df1['AMT_GOODS_PRICE'].mean())

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

plt.figure(figsize=(18,10))
plt.suptitle('Distribution of Original data vs imputed data')
plt.subplot(221)
sns.distplot(df1['AMT_GOODS_PRICE'][pd.notnull(df1['AMT_GOODS_PRICE'])]);
for i in enumerate(cols):
    plt.subplot(2,2,i[0]+2)
    sns.distplot(df3[i[1]])
```



Insight:we can see that the AMT_good price without imputation is almost similer to mode and median strategy, we chose median

In [160]:

```
df1['AMT_GOODS_PRICE'].fillna(df1['AMT_GOODS_PRICE'].mode()[0], inplace=True)
```

In [124]:

```
#As contract and CNT payment is related (if contact not happen , no payment happened). For this reason , mean ,me
#lets see how  many contract have null value for CNT_payment
df1.loc[df1['CNT_PAYMENT'].isnull(),'NAME_CONTRACT_STATUS'].value_counts()
```

Out[124]:

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

insight: counts the occurrences of each unique 'NAME_CONTRACT_STATUS' value for the rows where 'CNT_PAYMENT' is null , as most of the contact status refused and canceled , so there is no pay ment, so we can chage thses nAN value with 0

In [125]:

```
df1['CNT_PAYMENT'].fillna(0,inplace = True)
```

In [161]:

```
# checking the null value % of each column in previousDF dataframe
round(df1.isnull().sum() / df1.shape[0] * 100.00,2)
```

Out[161]:

```
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
```

Insight: no null value in our data, Lets check how outliar reside in each column by box plot and descrptive statistics

In [163]:

```
df1.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             1670214 non-null  float64
 4   AMT_APPLICATION         1670214 non-null  float64
 5   AMT_CREDIT              1670213 non-null  float64
 6   AMT_GOODS_PRICE         1670214 non-null  float64
 7   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             1670214 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 [151]:

```
df1.shape
```

Out[151]:

```
(1670214, 23)
```

In [128]:

```
df1.describe()
```

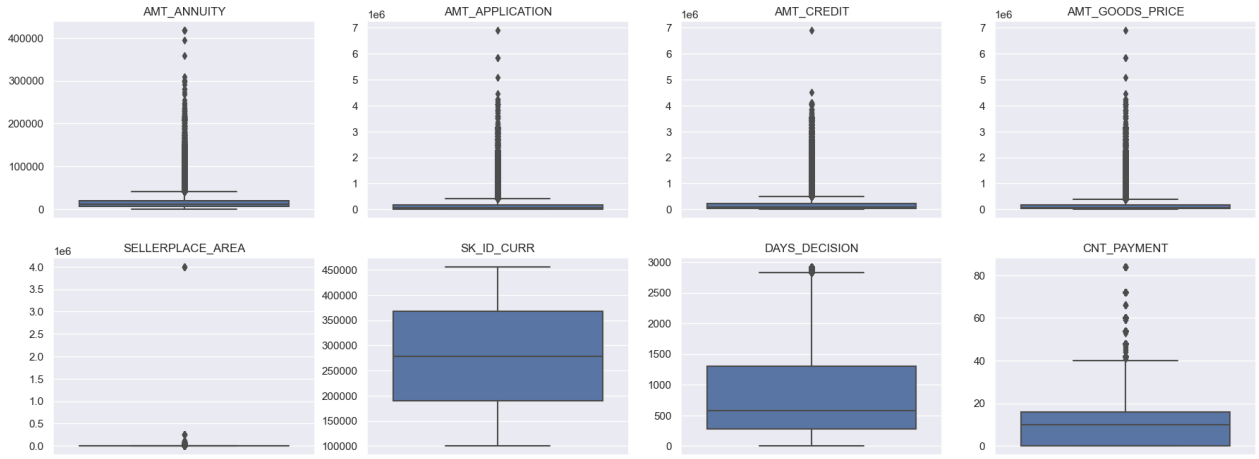
Out[128]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE	DAYS_DECISION	SELLERPLACE_A
count	1.670214e+06	1.670214e+06	1.297979e+06	1.670214e+06	1.670213e+06	1.670214e+06	1.670214e+06	1.670214e
mean	1.923089e+06	2.783572e+05	1.595512e+04	1.752339e+05	1.961140e+05	1.856429e+05	8.806797e+02	3.148644e
std	5.325980e+05	1.028148e+05	1.478214e+04	2.927798e+05	3.185746e+05	2.871413e+05	7.790997e+02	7.127403e
min	1.000001e+06	1.000010e+05	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	0.000000e
25%	1.461857e+06	1.893290e+05	6.321780e+03	1.872000e+04	2.416050e+04	4.500000e+04	2.800000e+02	1.000000e
50%	1.923110e+06	2.787145e+05	1.125000e+04	7.104600e+04	8.054100e+04	7.105050e+04	5.810000e+02	3.000000e
75%	2.384280e+06	3.675140e+05	2.065842e+04	1.803600e+05	2.164185e+05	1.804050e+05	1.300000e+03	8.200000e
max	2.845382e+06	4.562550e+05	4.180581e+05	6.905160e+06	6.905160e+06	6.905160e+06	2.922000e+03	4.000000e

Insight:we see AMT:_credit, Days _decision, sellerprice area have highher STD.DEV than other feature , it seem s that have outliar # Lets check with boxplot and confirm it

In [130]:

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

df1_col_1 = ['AMT_ANNUITY','AMT_APPLICATION','AMT_CREDIT','AMT_GOODS_PRICE','SELLERPLACE_AREA']
df1_col_2 = ['SK_ID_CURR','DAYS_DECISION','CNT_PAYMENT']
for i in enumerate(df1_col_1):
    plt.subplot(2,4,i[0]+1)
    sns.boxplot(y=df1[i[1]])
    plt.title(i[1])
    plt.ylabel("")

for i in enumerate(df1_col_2):
    plt.subplot(2,4,i[0]+6)
    sns.boxplot(y=df1[i[1]])
    plt.title(i[1])
    plt.ylabel("")
```



Insight: we saw 'AMT_ANNUITY','AMT_APPLICATION','AMT_CREDIT','AMT_GOODS_PRICE','SELLERPLACE_AREA', have several outliirs than others sellerplace area few, sk_id have no outliar And CNT payment has some few outliars

In [134]:

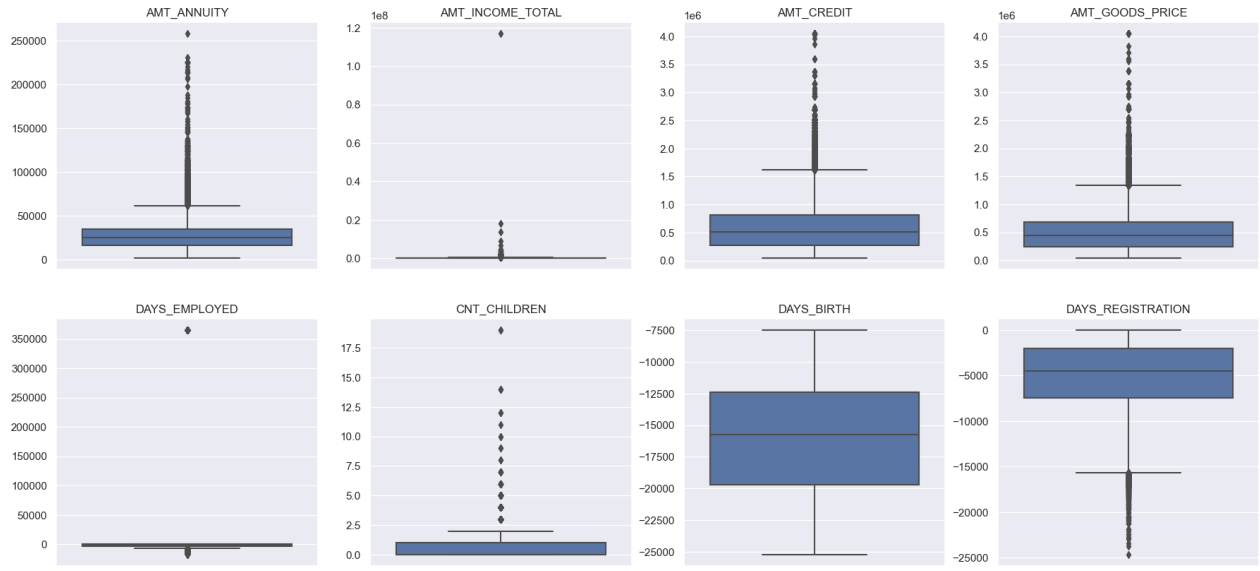
```
#Lets check the outliar in application dataset attribute df
# first check descriptive statistics
df.describe()
```

Out[134]:

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

In [141]:

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

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



Insight:attribute 'AMT_ANNUITY',AMT_CREDIT','AMT_GOODS_PRICE','DAYS_EMPLOYED' and CNT_children have some number of outliar Days of birth have no outliar, Days, employed huge outliar thatis over 35000, its around 958 yers, its impossible Days of registration have few outliars

In [164]:

```
df.shape
```

Out[164]:

```
(307511, 46)
```

MERGED DATAFRAME df and df1, Then analysis

In [142]:

```
#To merged two dataframe  we need a common attribute column , i here common attribute is SK_ID_CURR
# we create  a new dataframe for combined with innerjoin on SK_ID_CURR
df_combined=pd.merge(df,df1, how='inner', on='SK_ID_CURR')
df_combined.head()
```

Out[142]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_
0	100002	1	Cash loans	M	N	Y	0	2(
1	100003	0	Cash loans	F	N	N	0	27
2	100003	0	Cash loans	F	N	N	0	27
3	100003	0	Cash loans	F	N	N	0	27
4	100004	0	Revolving loans	M	Y	Y	0	(

In [144]:

```
df_combined.shape
```

Out[144]:

```
(1413701, 68)
```

In [179]:

```
df_combined.isnull().sum().sum()
```

Out[179]:

```
321417
```

Insight:307511 rows and 68 columns

In [152]:

```
1 df_combined.isnull().any()
```

Out[152]:

```
SK_ID_CURR                    False
TARGET                        False
NAME_CONTRACT_TYPE_x          False
CODE_GENDER                   False
FLAG_OWN_CAR                  False
FLAG_OWN_REALTY               False
CNT_CHILDREN                  False
AMT_INCOME_TOTAL              False
AMT_CREDIT_x                  False
AMT_ANNUITY_x                  True
AMT_GOODS_PRICE_x              True
NAME_TYPE_SUITE               False
NAME_INCOME_TYPE              False
NAME_EDUCATION_TYPE           False
NAME_FAMILY_STATUS            False
NAME_HOUSING_TYPE             False
REGION_POPULATION_RELATIVE    False
DAYS_BIRTH                    False
DAYS_EMPLOYED                 False
DAYS_REGISTRATION             False
DAYS_ID_PUBLISH               False
OCCUPATION_TYPE               False
CNT_FAM_MEMBERS               False
REGION_RATING_CLIENT          False
REGION_RATING_CLIENT_W_CITY   False
WEEKDAY_APPR_PROCESS_START    False
HOUR_APPR_PROCESS_START       False
REG_REGION_NOT_LIVE_REGION    False
REG_REGION_NOT_WORK_REGION    False
LIVE_REGION_NOT_WORK_REGION   False
REG_CITY_NOT_LIVE_CITY        False
REG_CITY_NOT_WORK_CITY        False
LIVE_CITY_NOT_WORK_CITY       False
ORGANIZATION_TYPE             False
OBS_30_CNT_SOCIAL_CIRCLE       True
DEF_30_CNT_SOCIAL_CIRCLE       True
OBS_60_CNT_SOCIAL_CIRCLE       True
DEF_60_CNT_SOCIAL_CIRCLE       True
DAYS_LAST_PHONE_CHANGE        False
FLAG_DOCUMENT_3               False
AMT_REQ_CREDIT_BUREAU_HOUR    False
AMT_REQ_CREDIT_BUREAU_DAY     False
AMT_REQ_CREDIT_BUREAU_WEEK    False
AMT_REQ_CREDIT_BUREAU_MON     False
AMT_REQ_CREDIT_BUREAU_QRT     False
AMT_REQ_CREDIT_BUREAU_YEAR    False
SK_ID_PREV                    False
NAME_CONTRACT_TYPE_y          False
AMT_ANNUITY_y                  True
AMT_APPLICATION               False
AMT_CREDIT_y                   True
AMT_GOODS_PRICE_y             False
NAME_CASH_LOAN_PURPOSE        False
NAME_CONTRACT_STATUS          False
DAYS_DECISION                 False
NAME_PAYMENT_TYPE             False
CODE_REJECT_REASON            False
NAME_CLIENT_TYPE              False
NAME_GOODS_CATEGORY           False
NAME_PORTFOLIO                False
NAME_PRODUCT_TYPE             False
CHANNEL_TYPE                  False
SELLERPLACE_AREA              False
NAME_SELLER_INDUSTRY          False
CNT_PAYMENT                   False
NAME_YIELD_GROUP              False
PRODUCT_COMBINATION            True
DAYS_DECISION_GROUP           False
dtype: bool
```

In [153]:

```
round(df_combined.isnull().sum() / df_combined.shape[0] * 100.00,2)
```

Out[153]:

```
SK_ID_CURR                     0.00
TARGET                         0.00
NAME_CONTRACT_TYPE_x           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_x                   0.00
AMT_ANNUITY_x                  0.01
AMT_GOODS_PRICE_x              0.09
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       0.22
DEF_30_CNT_SOCIAL_CIRCLE       0.22
OBS_60_CNT_SOCIAL_CIRCLE       0.22
DEF_60_CNT_SOCIAL_CIRCLE       0.22
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
SK_ID_PREV                     0.00
NAME_CONTRACT_TYPE_y           0.00
AMT_ANNUITY_y                 21.73
AMT_APPLICATION                0.00
AMT_CREDIT_y                   0.00
AMT_GOODS_PRICE_y              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 [166]:

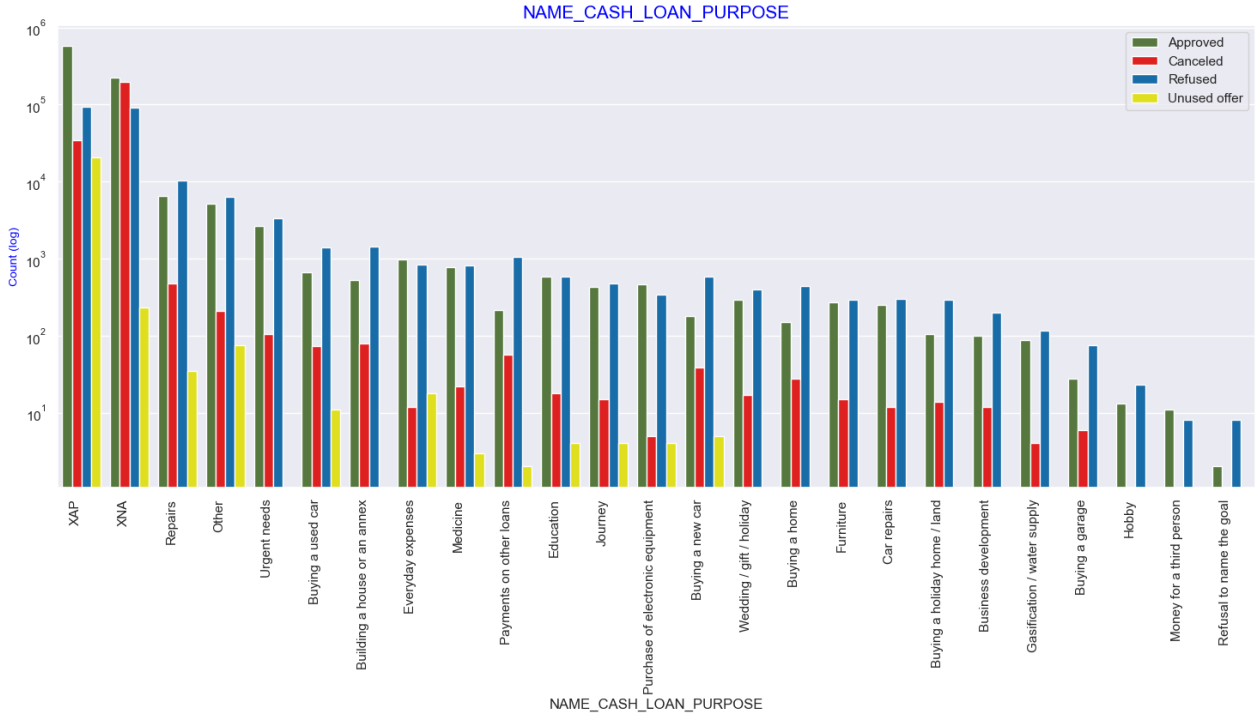
```
# Bifurcating the applicationDF dataframe based on Target value 0 and 1 for correlation and other analysis

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

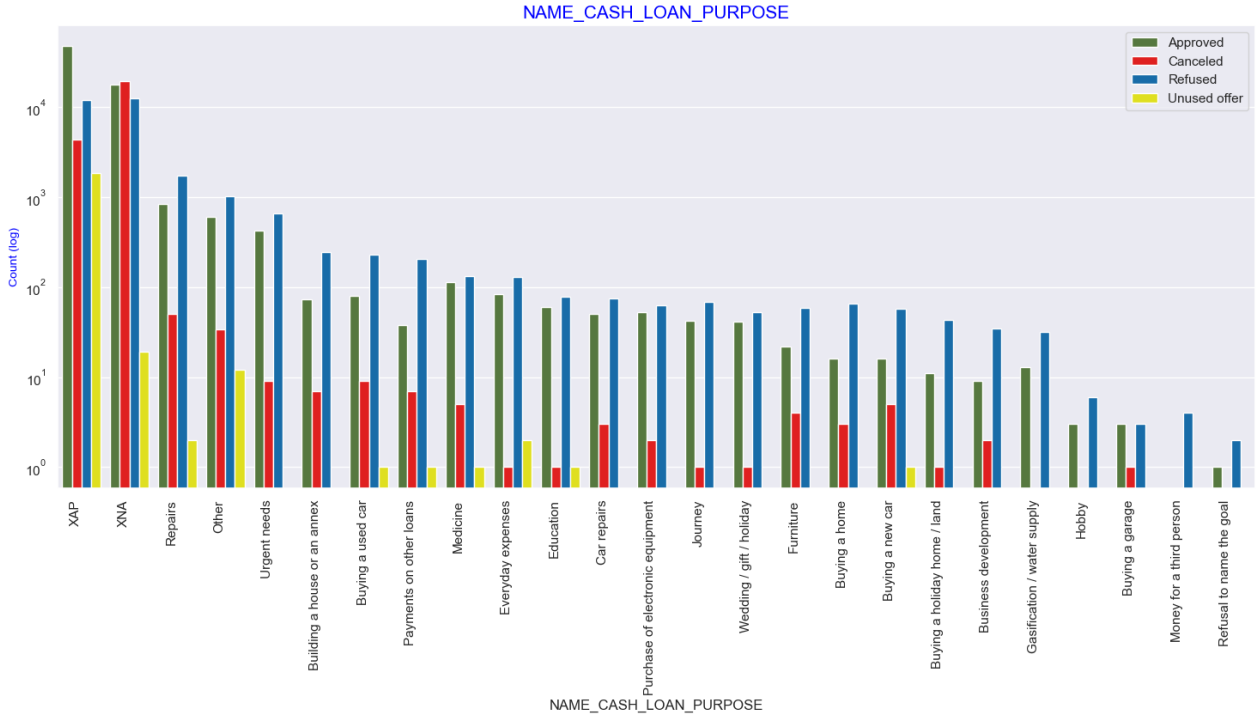
In [167]:

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

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' : 'Blue'})
    else:
        plt.ylabel("Count",fontdict={'fontsize' : 10, 'fontweight' : 3, 'color' : 'Blue'})

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

    plt.show()
```

In [168]:

```
univariate_merged("NAME_CASH_LOAN_PURPOSE",L0,"NAME_CONTRACT_STATUS",["#548235","#FF0000","#0070C0","#FFFF00"],Tr

univariate_merged("NAME_CASH_LOAN_PURPOSE",L1,"NAME_CONTRACT_STATUS",["#548235","#FF0000","#0070C0","#FFFF00"],Tr
```

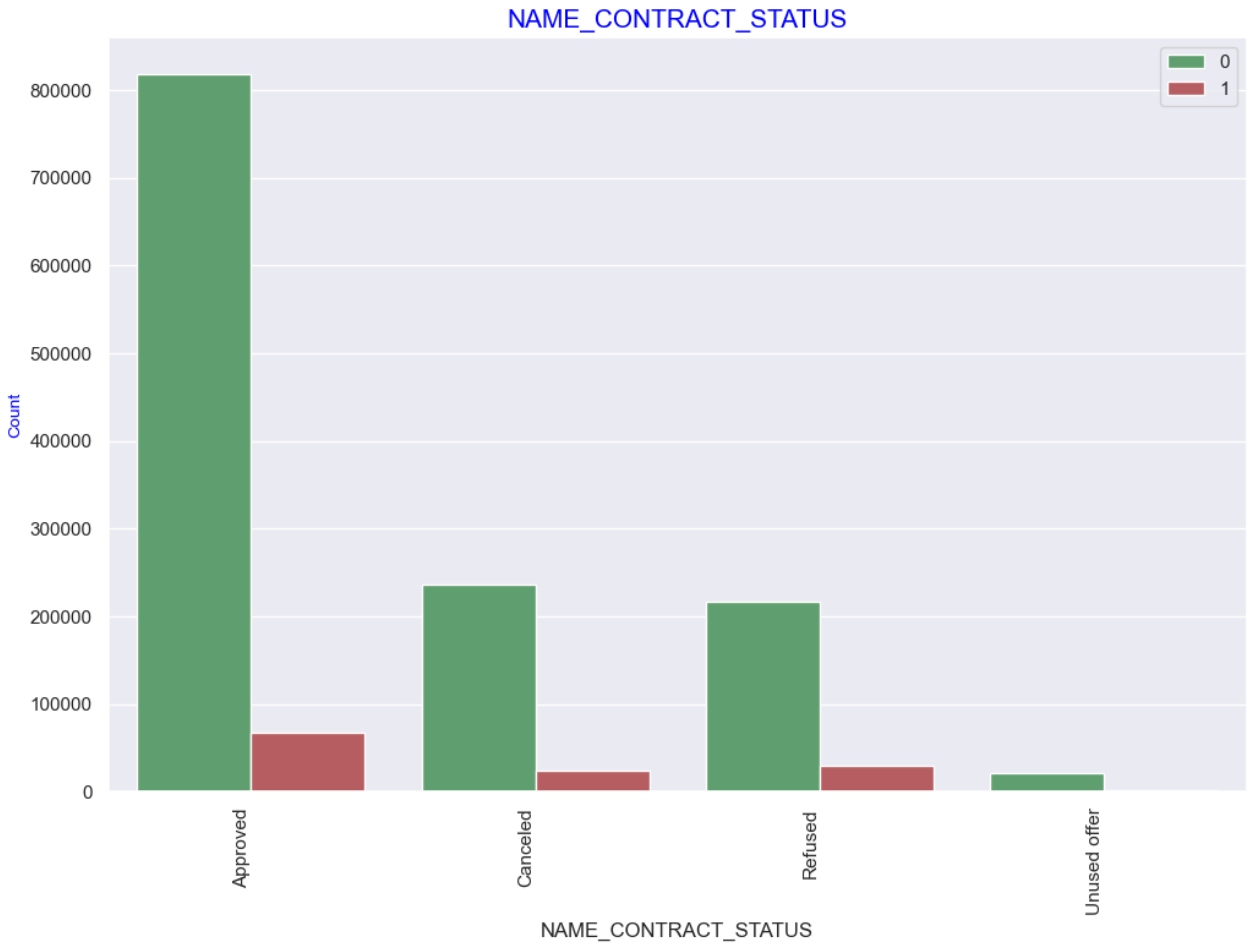




Insight: Loan purpose has high number of unknown values (XAP, XNA) Loan taken for the purpose of Repairs seems to have highest default rate A very high number application have been rejected by bank or refused by client which has purpose as repair or other. This shows that purpose repair is taken as high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

In [169]:

```
# Checking the Contract Status based on loan repayment status and whether there is any business loss or financial
univariate_merged("NAME_CONTRACT_STATUS",df_combined,"TARGET",['g','r'],False,(12,8))
```



In [170]:

```
# to  displays the counts and percentages of each value in the "NAME_CONTRACT_STATUS" column based on the loan re
g = df_combined.groupby("NAME_CONTRACT_STATUS")["TARGET"]
df5 = pd.concat([g.value_counts(),round(g.value_counts(normalize=True).mul(100),2)],axis=1, keys=('Counts','Perce
df5['Percentage'] = df5['Percentage'].astype(str) +"%" # adding percentage symbol in the results for understandir
print (df5)
```

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

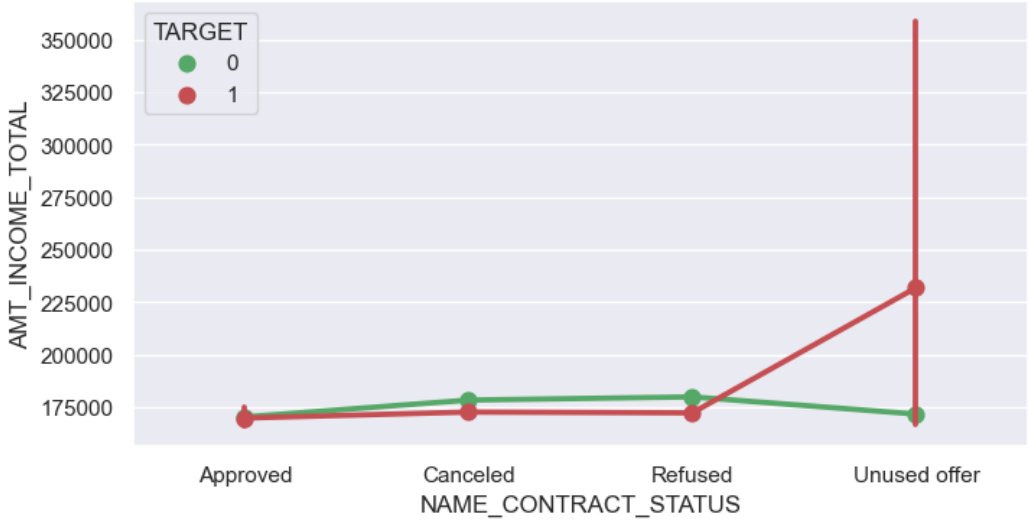
```
Insight:
90% of the previously cancelled client have actually repayed the loan. Revisiting the interest rates would
increase business opoortunity for these clients
88% of the clients who have been previously refused a loan has payed back the loan in current case.
Refual reason should be recorded for further analysis as these clients would turn into potential repaying
customer.
```

In [175]:

```
# 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=df_combined,
                  palette =['g','r'])
   # plt.legend(['Repayer','Defaulter'])
```

In [176]:

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



Insight:the people who have got unused offer are highest defaulter even the have the high income

In [177]:

```
# plotting the relationship between people who defaulted in last 60 days being in client's social circle and cont
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 default more and hence client's social circle has to be analysed before providing the loan.