Data Analysis on the Bank Loan Application data

Importing the libraries

In [1]:
```
import warnings
warnings.filterwarnings('ignore')  ## importing the required libraries for filt
ering out the warning messages
import numpy as np
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
import os
import itertools
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.tree import export_graphviz
import graphviz
from sklearn import ensemble
from sklearn import datasets
from sklearn.utils import shuffle
from sklearn.metrics import mean_squared_error
```

Verifying CWD path

In [2]:
```
os.getcwd()
```

Out[2]: '/content'

Importing the Dataset

In [3]:
```
bank_df = pd.read_csv("application_data.csv")
bank_df.head(10)
```

Out[3]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY
0	100002	1	Cash loans	M	N	Y
1	100003	0	Cash loans	F	N	N
2	100004	0	Revolving loans	M	Y	Y
3	100006	0	Cash loans	F	N	Y
4	100007	0	Cash loans	M	N	Y
5	100008	0	Cash loans	M	N	Y
6	100009	0	Cash loans	F	Y	Y
7	100010	0	Cash loans	M	Y	Y
8	100011	0	Cash loans	F	N	Y
9	100012	0	Revolving loans	M	N	Y

10 rows × 122 columns

Examining the Data

Following data import, we'll use the .info(), and.describe() methods to learn more about the dataset.

In [4]:
```
## Finding the total number of records in the dataset
bank_df.shape
```

Out[4]: (15523, 122)

Observation:

The following dataset consists of **122** columns and about **307511** rows of data.

In [5]:
```
## Performing descriptive analysis on the entire dataset

bank_df.describe()
```

Out[5]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AM
count	15523.000000	15523.000000	15523.000000	1.552300e+04	1.552300e+04	15523.000000	
mean	109049.560523	0.078400	0.421310	1.753807e+05	6.021176e+05	27141.630709	
std	5222.055991	0.268808	0.725172	9.426409e+05	4.041854e+05	14515.037629	
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	2596.500000	
25%	104539.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16474.500000	
50%	109054.000000	0.000000	0.000000	1.440000e+05	5.172660e+05	25042.500000	
75%	113563.500000	0.000000	1.000000	2.025000e+05	8.104748e+05	34731.000000	
max	118116.000000	1.000000	8.000000	1.170000e+08	4.050000e+06	225000.000000	

8 rows × 106 columns

Observation:

As may be seen from the results described above,

- Columns AMT INCOME TOTAL, AMT CREDIT, and AMT GOODS PRICE contain extremely large values; therefore, for easier comprehension, these numerical columns will be converted to categorical columns.
- Columns that count days, such as Days Born, Days Worked, Days Registered, Days Id Published, and Days Last Phone Change, have negative values. which will update those values.
- Convert DAYS EMPLOYED to YEARS EMPLOYED and DAYS BIRTH to AGE in years

In [6]:
```
bank_df.info("all")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15523 entries, 0 to 15522
Data columns (total 122 columns):
 #    Column                          Dtype
---   ------                          -----
 0    SK_ID_CURR                      int64
 1    TARGET                          int64
 2    NAME_CONTRACT_TYPE              object
 3    CODE_GENDER                     object
 4    FLAG_OWN_CAR                    object
 5    FLAG_OWN_REALTY                 object
 6    CNT_CHILDREN                    int64
 7    AMT_INCOME_TOTAL                float64
 8    AMT_CREDIT                      float64
 9    AMT_ANNUITY                     float64
 10   AMT_GOODS_PRICE                 float64
 11   NAME_TYPE_SUITE                 object
 12   NAME_INCOME_TYPE                object
 13   NAME_EDUCATION_TYPE             object
 14   NAME_FAMILY_STATUS              object
 15   NAME_HOUSING_TYPE               object
 16   REGION_POPULATION_RELATIVE      float64
 17   DAYS_BIRTH                      int64
 18   DAYS_EMPLOYED                   int64
 19   DAYS_REGISTRATION               float64
 20   DAYS_ID_PUBLISH                 int64
 21   OWN_CAR_AGE                     float64
 22   FLAG_MOBIL                      int64
 23   FLAG_EMP_PHONE                  int64
 24   FLAG_WORK_PHONE                 int64
 25   FLAG_CONT_MOBILE                int64
 26   FLAG_PHONE                      int64
 27   FLAG_EMAIL                      float64
 28   OCCUPATION_TYPE                 object
 29   CNT_FAM_MEMBERS                 float64
 30   REGION_RATING_CLIENT            float64
 31   REGION_RATING_CLIENT_W_CITY     float64
 32   WEEKDAY_APPR_PROCESS_START      object
 33   HOUR_APPR_PROCESS_START         float64
 34   REG_REGION_NOT_LIVE_REGION      float64
 35   REG_REGION_NOT_WORK_REGION      float64
 36   LIVE_REGION_NOT_WORK_REGION     float64
 37   REG_CITY_NOT_LIVE_CITY          float64
 38   REG_CITY_NOT_WORK_CITY          float64
 39   LIVE_CITY_NOT_WORK_CITY         float64
 40   ORGANIZATION_TYPE               object
 41   EXT_SOURCE_1                    float64
 42   EXT_SOURCE_2                    float64
 43   EXT_SOURCE_3                    float64
 44   APARTMENTS_AVG                  float64
 45   BASEMENTAREA_AVG                float64
 46   YEARS_BEGINEXPLUATATION_AVG     float64
 47   YEARS_BUILD_AVG                 float64
 48   COMMONAREA_AVG                  float64
 49   ELEVATORS_AVG                   float64
 50   ENTRANCES_AVG                   float64
 51   FLOORSMAX_AVG                   float64
 52   FLOORSMIN_AVG                   float64
 53   LANDAREA_AVG                    float64
 54   LIVINGAPARTMENTS_AVG            float64
 55   LIVINGAREA_AVG                  float64
```

56	NONLIVINGAPARTMENTS_AVG	float64
57	NONLIVINGAREA_AVG	float64
58	APARTMENTS_MODE	float64
59	BASEMENTAREA_MODE	float64
60	YEARS_BEGINEXPLUATATION_MODE	float64
61	YEARS_BUILD_MODE	float64
62	COMMONAREA_MODE	float64
63	ELEVATORS_MODE	float64
64	ENTRANCES_MODE	float64
65	FLOORSMAX_MODE	float64
66	FLOORSMIN_MODE	float64
67	LANDAREA_MODE	float64
68	LIVINGAPARTMENTS_MODE	float64
69	LIVINGAREA_MODE	float64
70	NONLIVINGAPARTMENTS_MODE	float64
71	NONLIVINGAREA_MODE	float64
72	APARTMENTS_MEDI	float64
73	BASEMENTAREA_MEDI	float64
74	YEARS_BEGINEXPLUATATION_MEDI	float64
75	YEARS_BUILD_MEDI	float64
76	COMMONAREA_MEDI	float64
77	ELEVATORS_MEDI	float64
78	ENTRANCES_MEDI	float64
79	FLOORSMAX_MEDI	float64
80	FLOORSMIN_MEDI	float64
81	LANDAREA_MEDI	float64
82	LIVINGAPARTMENTS_MEDI	float64
83	LIVINGAREA_MEDI	float64
84	NONLIVINGAPARTMENTS_MEDI	float64
85	NONLIVINGAREA_MEDI	float64
86	FONDKAPREMONT_MODE	object
87	HOUSETYPE_MODE	object
88	TOTALAREA_MODE	float64
89	WALLSMATERIAL_MODE	object
90	EMERGENCYSTATE_MODE	object
91	OBS_30_CNT_SOCIAL_CIRCLE	float64
92	DEF_30_CNT_SOCIAL_CIRCLE	float64
93	OBS_60_CNT_SOCIAL_CIRCLE	float64
94	DEF_60_CNT_SOCIAL_CIRCLE	float64
95	DAYS_LAST_PHONE_CHANGE	float64
96	FLAG_DOCUMENT_2	float64
97	FLAG_DOCUMENT_3	float64
98	FLAG_DOCUMENT_4	float64
99	FLAG_DOCUMENT_5	float64
100	FLAG_DOCUMENT_6	float64
101	FLAG_DOCUMENT_7	float64
102	FLAG_DOCUMENT_8	float64
103	FLAG_DOCUMENT_9	float64
104	FLAG_DOCUMENT_10	float64
105	FLAG_DOCUMENT_11	float64
106	FLAG_DOCUMENT_12	float64
107	FLAG_DOCUMENT_13	float64
108	FLAG_DOCUMENT_14	float64
109	FLAG_DOCUMENT_15	float64
110	FLAG_DOCUMENT_16	float64
111	FLAG_DOCUMENT_17	float64
112	FLAG_DOCUMENT_18	float64
113	FLAG_DOCUMENT_19	float64
114	FLAG_DOCUMENT_20	float64
115	FLAG_DOCUMENT_21	float64
116	AMT_REQ_CREDIT_BUREAU_HOUR	float64
117	AMT_REQ_CREDIT_BUREAU_DAY	float64

```
 118  AMT_REQ_CREDIT_BUREAU_WEEK    float64
 119  AMT_REQ_CREDIT_BUREAU_MON     float64
 120  AMT_REQ_CREDIT_BUREAU_QRT     float64
 121  AMT_REQ_CREDIT_BUREAU_YEAR    float64
dtypes: float64(95), int64(11), object(16)
memory usage: 14.4+ MB
```

Data Cleaning and Manipulation

Finding the null values in the given data sheet

In [7]:
```
# Creating a function to find the null values in the given sheet

def null(df):
    return round((df.isnull().sum()*100/len(df)).sort_values(ascending = Fals
e),3)
```

In [8]:
```
null(bank_df)
```

Out[8]:
```
COMMONAREA_MODE             69.974
COMMONAREA_AVG              69.974
COMMONAREA_MEDI             69.974
NONLIVINGAPARTMENTS_MODE    69.394
NONLIVINGAPARTMENTS_AVG     69.394
                              ...
NAME_HOUSING_TYPE            0.000
NAME_FAMILY_STATUS           0.000
NAME_EDUCATION_TYPE          0.000
NAME_INCOME_TYPE             0.000
SK_ID_CURR                   0.000
Length: 122, dtype: float64
```

Finding cells missing more than 50% of the data

In [9]:
```
# Creating a variable to store the columns missing more than 50% of the data

value_missing_50 = null(bank_df)[null(bank_df)>50]
print(value_missing_50)
print("No. of columns missing more than 50% data =", len(value_missing_50))
```

```
COMMONAREA_MODE             69.974
COMMONAREA_AVG              69.974
COMMONAREA_MEDI             69.974
NONLIVINGAPARTMENTS_MODE    69.394
NONLIVINGAPARTMENTS_AVG     69.394
NONLIVINGAPARTMENTS_MEDI    69.394
FONDKAPREMONT_MODE          68.543
LIVINGAPARTMENTS_AVG        68.427
LIVINGAPARTMENTS_MODE       68.427
LIVINGAPARTMENTS_MEDI       68.427
FLOORSMIN_MEDI              67.770
FLOORSMIN_MODE              67.770
FLOORSMIN_AVG               67.770
YEARS_BUILD_MEDI            66.443
YEARS_BUILD_AVG             66.443
YEARS_BUILD_MODE            66.443
OWN_CAR_AGE                 66.115
LANDAREA_MEDI               59.067
LANDAREA_MODE               59.067
LANDAREA_AVG                59.067
BASEMENTAREA_MODE           58.198
BASEMENTAREA_AVG            58.198
BASEMENTAREA_MEDI           58.198
EXT_SOURCE_1                56.297
NONLIVINGAREA_MODE          54.815
NONLIVINGAREA_AVG           54.815
NONLIVINGAREA_MEDI          54.815
ELEVATORS_MODE              52.992
ELEVATORS_MEDI              52.992
ELEVATORS_AVG               52.992
WALLSMATERIAL_MODE          50.918
APARTMENTS_AVG              50.693
APARTMENTS_MEDI             50.693
APARTMENTS_MODE             50.693
LIVINGAREA_AVG              50.325
LIVINGAREA_MODE             50.325
LIVINGAREA_MEDI             50.325
ENTRANCES_MODE              50.203
ENTRANCES_AVG               50.203
ENTRANCES_MEDI              50.203
HOUSETYPE_MODE              50.048
dtype: float64
No. of columns missing more than 50% data = 41
```

Observation:

Totally around **41** columns arte missing more than 50 percent data.

In [10]:
```
# Printing the column names that are missing more tha 50% data
value_missing_50.index
```

Out[10]:
```
Index(['COMMONAREA_MODE', 'COMMONAREA_AVG', 'COMMONAREA_MEDI',
       'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAPARTMENTS_AVG',
       'NONLIVINGAPARTMENTS_MEDI', 'FONDKAPREMONT_MODE',
       'LIVINGAPARTMENTS_AVG', 'LIVINGAPARTMENTS_MODE',
       'LIVINGAPARTMENTS_MEDI', 'FLOORSMIN_MEDI', 'FLOORSMIN_MODE',
       'FLOORSMIN_AVG', 'YEARS_BUILD_MEDI', 'YEARS_BUILD_AVG',
       'YEARS_BUILD_MODE', 'OWN_CAR_AGE', 'LANDAREA_MEDI', 'LANDAREA_MODE',
       'LANDAREA_AVG', 'BASEMENTAREA_MODE', 'BASEMENTAREA_AVG',
       'BASEMENTAREA_MEDI', 'EXT_SOURCE_1', 'NONLIVINGAREA_MODE',
       'NONLIVINGAREA_AVG', 'NONLIVINGAREA_MEDI', 'ELEVATORS_MODE',
       'ELEVATORS_MEDI', 'ELEVATORS_AVG', 'WALLSMATERIAL_MODE',
       'APARTMENTS_AVG', 'APARTMENTS_MEDI', 'APARTMENTS_MODE',
       'LIVINGAREA_AVG', 'LIVINGAREA_MODE', 'LIVINGAREA_MEDI',
       'ENTRANCES_MODE', 'ENTRANCES_AVG', 'ENTRANCES_MEDI', 'HOUSETYPE_MODE'],
      dtype='object')
```

In [11]:
```
# Dropping the above columns from the data set, as these following items will n
ot yeild any help while analysis
bank_df.drop(columns = value_missing_50.index, inplace = True)
```

In [12]:
```
bank_df.shape
```

Out[12]:
```
(15523, 81)
```

Observation:

Dropping the above **41** columns from the data set, thus we are left with **81** columns for analysis.

Dealing with columns missing around 20% data

In [13]:
```
#Checking the value again

bank_df.head(10)
```

Out[13]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY
0	100002	1	Cash loans	M	N	Y
1	100003	0	Cash loans	F	N	N
2	100004	0	Revolving loans	M	Y	Y
3	100006	0	Cash loans	F	N	Y
4	100007	0	Cash loans	M	N	Y
5	100008	0	Cash loans	M	N	Y
6	100009	0	Cash loans	F	Y	Y
7	100010	0	Cash loans	M	Y	Y
8	100011	0	Cash loans	F	N	Y
9	100012	0	Revolving loans	M	N	Y

10 rows × 81 columns

Dealing with other null values in the data set

In [14]:
```
# Trying to deal with null values which are missing data greater than 20%

value_greater_20 = null(bank_df)[null(bank_df)>20]
value_greater_20
```

Out[14]:
```
FLOORSMAX_AVG                49.527
FLOORSMAX_MODE               49.527
FLOORSMAX_MEDI               49.527
YEARS_BEGINEXPLUATATION_AVG  48.876
YEARS_BEGINEXPLUATATION_MODE 48.876
YEARS_BEGINEXPLUATATION_MEDI 48.876
TOTALAREA_MODE               48.251
EMERGENCYSTATE_MODE          47.368
OCCUPATION_TYPE              31.038
dtype: float64
```

In [15]:
```
#removing occupation data from the data frame as that column seems important
value_greater_20.drop("OCCUPATION_TYPE", inplace = True)
```

In [16]:
```
print(value_greater_20)
print("No.of columns missing more than 20% data and relatable to target field =
",len(value_greater_20))
```

```
FLOORSMAX_AVG                  49.527
FLOORSMAX_MODE                 49.527
FLOORSMAX_MEDI                 49.527
YEARS_BEGINEXPLUATATION_AVG    48.876
YEARS_BEGINEXPLUATATION_MODE   48.876
YEARS_BEGINEXPLUATATION_MEDI   48.876
TOTALAREA_MODE                 48.251
EMERGENCYSTATE_MODE            47.368
dtype: float64
No.of columns missing more than 20% data and relatable to target field =  8
```

In [17]:
```
value_greater_20.index
```

Out[17]:
```
Index(['FLOORSMAX_AVG', 'FLOORSMAX_MODE', 'FLOORSMAX_MEDI',
       'YEARS_BEGINEXPLUATATION_AVG', 'YEARS_BEGINEXPLUATATION_MODE',
       'YEARS_BEGINEXPLUATATION_MEDI', 'TOTALAREA_MODE',
       'EMERGENCYSTATE_MODE'],
      dtype='object')
```

In [18]:
```
bank_df.drop(columns = value_greater_20.index, inplace = True)
```

In [19]:
```
bank_df.head(20)
```

Out[19]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY
0	100002	1	Cash loans	M	N	Y
1	100003	0	Cash loans	F	N	N
2	100004	0	Revolving loans	M	Y	Y
3	100006	0	Cash loans	F	N	Y
4	100007	0	Cash loans	M	N	Y
5	100008	0	Cash loans	M	N	Y
6	100009	0	Cash loans	F	Y	Y
7	100010	0	Cash loans	M	Y	Y
8	100011	0	Cash loans	F	N	Y
9	100012	0	Revolving loans	M	N	Y
10	100014	0	Cash loans	F	N	Y
11	100015	0	Cash loans	F	N	Y
12	100016	0	Cash loans	F	N	Y
13	100017	0	Cash loans	M	Y	N
14	100018	0	Cash loans	F	N	Y
15	100019	0	Cash loans	M	Y	Y
16	100020	0	Cash loans	M	N	N
17	100021	0	Revolving loans	F	N	Y
18	100022	0	Revolving loans	F	N	Y
19	100023	0	Cash loans	F	N	Y

20 rows × 73 columns

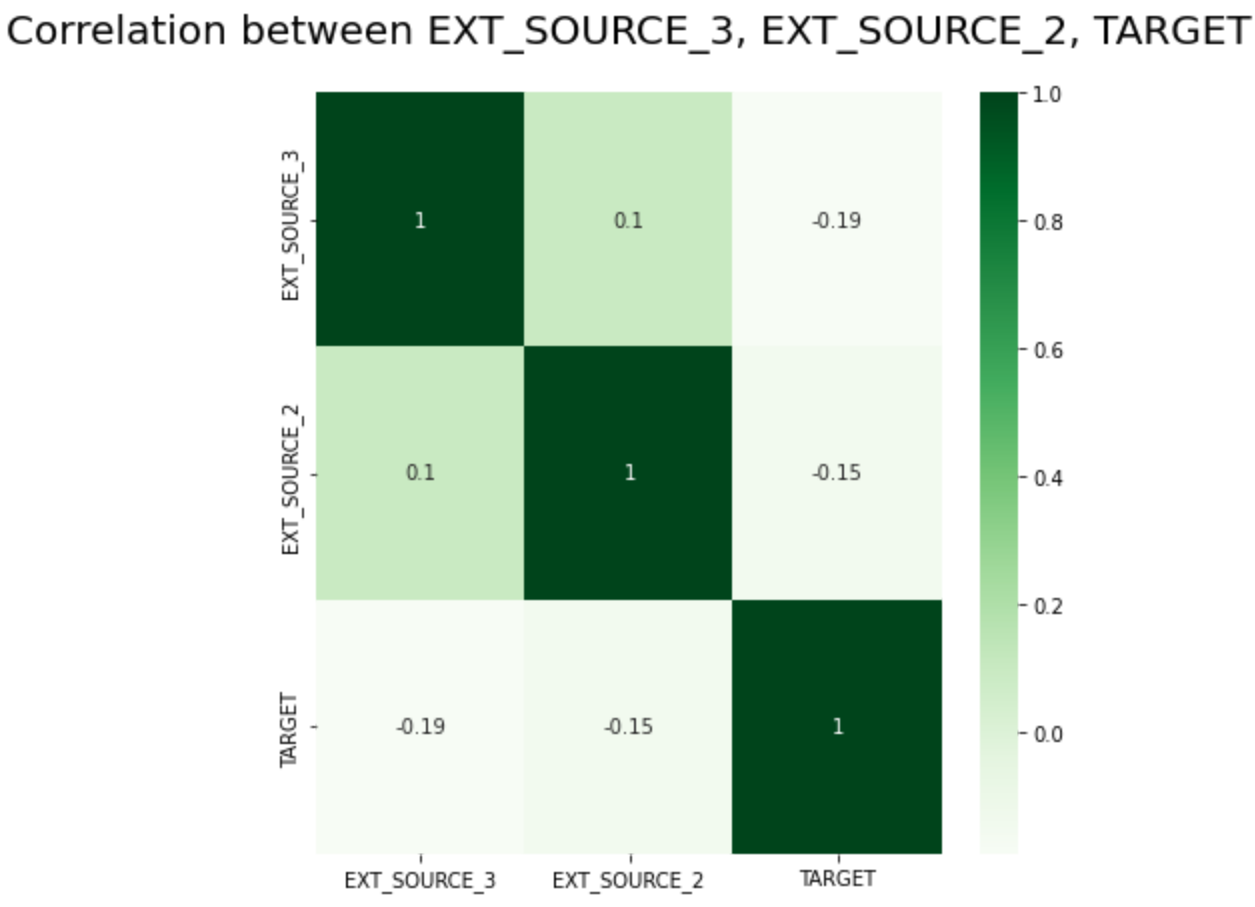
In [20]:
```
# After dropping the curther columns that are missing around 20 percent data

bank_df.shape
```

Out[20]:
```
(15523, 73)
```

In [21]:
```
null(bank_df).head(20)
```

Out[21]:
```
OCCUPATION_TYPE               31.038
EXT_SOURCE_3                  19.900
AMT_REQ_CREDIT_BUREAU_YEAR    13.432
AMT_REQ_CREDIT_BUREAU_QRT     13.432
AMT_REQ_CREDIT_BUREAU_MON     13.432
AMT_REQ_CREDIT_BUREAU_WEEK    13.432
AMT_REQ_CREDIT_BUREAU_DAY     13.432
AMT_REQ_CREDIT_BUREAU_HOUR    13.432
NAME_TYPE_SUITE                0.406
OBS_30_CNT_SOCIAL_CIRCLE       0.399
DEF_30_CNT_SOCIAL_CIRCLE       0.399
OBS_60_CNT_SOCIAL_CIRCLE       0.399
DEF_60_CNT_SOCIAL_CIRCLE       0.399
EXT_SOURCE_2                   0.264
AMT_GOODS_PRICE                0.071
FLAG_DOCUMENT_7                0.006
DAYS_LAST_PHONE_CHANGE         0.006
FLAG_DOCUMENT_2                0.006
FLAG_DOCUMENT_3                0.006
FLAG_DOCUMENT_4                0.006
dtype: float64
```

In [22]:
```
# Removing the unneccasary colums from the data set

irrelevant = ["EXT_SOURCE_3","EXT_SOURCE_2"]

plt.figure(figsize=[7,7])
sns.heatmap(bank_df[irrelevant+["TARGET"]].corr(),cmap="Greens", annot=True)
plt.title("Correlation between EXT_SOURCE_3, EXT_SOURCE_2, TARGET", fontdict=
{"fontsize":20}, pad=25)
plt.show()
```



Observation:

From the above heat map it is clear that there is no linear correlations between the above colums heading. So, dropping the above colums from the data set.

In [23]:
```
# The columns "EXT_SOURCE_3" and "EXT_SOURCE_2" are having negative correlation
with target, so dropping the following values from the data set

bank_df.drop(irrelevant, axis=1, inplace = True)
```

In [24]:
```
bank_df.shape
```

Out[24]:
```
(15523, 71)
```

In [25]:
```
null(bank_df).head(20)
```

Out[25]:
```
OCCUPATION_TYPE               31.038
AMT_REQ_CREDIT_BUREAU_YEAR    13.432
AMT_REQ_CREDIT_BUREAU_QRT     13.432
AMT_REQ_CREDIT_BUREAU_MON     13.432
AMT_REQ_CREDIT_BUREAU_WEEK    13.432
AMT_REQ_CREDIT_BUREAU_DAY     13.432
AMT_REQ_CREDIT_BUREAU_HOUR    13.432
NAME_TYPE_SUITE                0.406
OBS_30_CNT_SOCIAL_CIRCLE       0.399
DEF_30_CNT_SOCIAL_CIRCLE       0.399
OBS_60_CNT_SOCIAL_CIRCLE       0.399
DEF_60_CNT_SOCIAL_CIRCLE       0.399
AMT_GOODS_PRICE                0.071
FLAG_DOCUMENT_8                0.006
DAYS_LAST_PHONE_CHANGE         0.006
FLAG_DOCUMENT_2                0.006
FLAG_DOCUMENT_3                0.006
ORGANIZATION_TYPE              0.006
FLAG_DOCUMENT_4                0.006
FLAG_DOCUMENT_5                0.006
dtype: float64
```

In [26]:
```
# Finding the lenght of the null columns in the data set

len(null(bank_df))
```

Out[26]:
```
71
```

In [27]:
```
print(null(bank_df))
```

```
OCCUPATION_TYPE               31.038
AMT_REQ_CREDIT_BUREAU_YEAR    13.432
AMT_REQ_CREDIT_BUREAU_QRT     13.432
AMT_REQ_CREDIT_BUREAU_MON     13.432
AMT_REQ_CREDIT_BUREAU_WEEK    13.432
                                ...
FLAG_EMP_PHONE                 0.000
FLAG_WORK_PHONE                0.000
FLAG_CONT_MOBILE               0.000
FLAG_PHONE                     0.000
SK_ID_CURR                     0.000
Length: 71, dtype: float64
```

Starting to analyse the columns containing FLAGS w.r.t TARGET column

In [28]:
```
# The following data set have a lot of "Flag" feilds, so we will try to find nu
mber of Flag column and try to understand if its relation with the target feild

flag_col = [col for col in bank_df.columns if "FLAG" in col]

flag_col
```

Out[28]:
```
['FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'FLAG_MOBIL',
 'FLAG_EMP_PHONE',
 'FLAG_WORK_PHONE',
 'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 'FLAG_DOCUMENT_2',
 'FLAG_DOCUMENT_3',
 'FLAG_DOCUMENT_4',
 'FLAG_DOCUMENT_5',
 'FLAG_DOCUMENT_6',
 'FLAG_DOCUMENT_7',
 'FLAG_DOCUMENT_8',
 'FLAG_DOCUMENT_9',
 'FLAG_DOCUMENT_10',
 'FLAG_DOCUMENT_11',
 'FLAG_DOCUMENT_12',
 'FLAG_DOCUMENT_13',
 'FLAG_DOCUMENT_14',
 'FLAG_DOCUMENT_15',
 'FLAG_DOCUMENT_16',
 'FLAG_DOCUMENT_17',
 'FLAG_DOCUMENT_18',
 'FLAG_DOCUMENT_19',
 'FLAG_DOCUMENT_20',
 'FLAG_DOCUMENT_21']
```

In [29]:
```
# Creating a dataframe for the flag content
df_flag = bank_df[flag_col+["TARGET"]]
```

In [30]:
```
# Replacing the values "0" and "1" with appropiate values as per the dictionary defnition, where the value "1" represents people with previous payment issue and "0" represents proper record of payment or no previous loan records

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

In [31]:
```
# For the rest of the flag columns, "0" represents No and "1" represents Yes, so replaciung them with "Y" & "N"

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

In [32]:
```
df_flag.head(20)
```

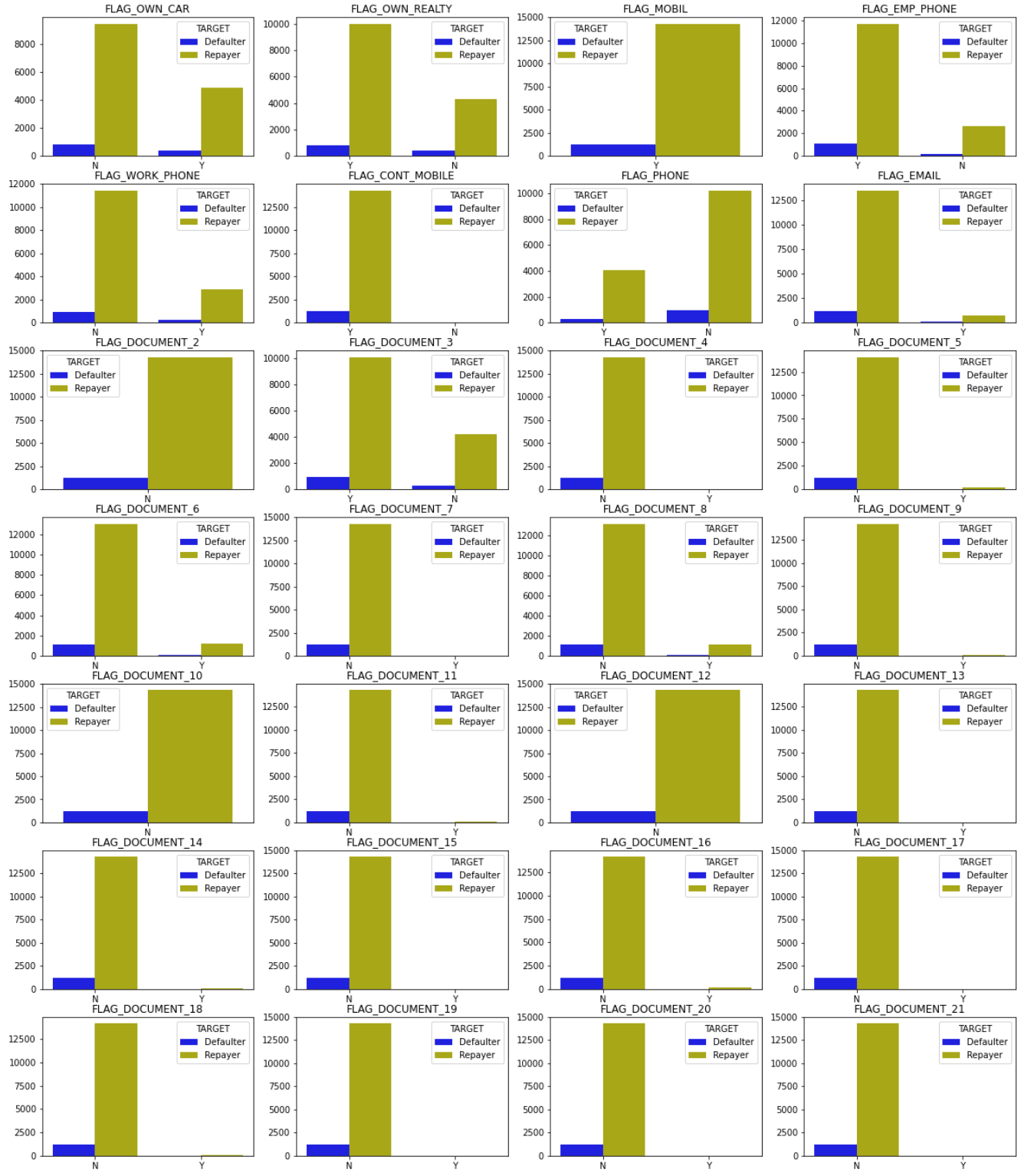
Out[32]:

	FLAG_OWN_CAR	FLAG_OWN_REALTY	FLAG_MOBIL	FLAG_EMP_PHONE	FLAG_WORK_PHONE	FLAG_C
0	N	Y	Y	Y	N	
1	N	N	Y	Y	N	
2	Y	Y	Y	Y	Y	
3	N	Y	Y	Y	N	
4	N	Y	Y	Y	N	
5	N	Y	Y	Y	Y	
6	Y	Y	Y	Y	N	
7	Y	Y	Y	Y	Y	
8	N	Y	Y	N	N	
9	N	Y	Y	Y	N	
10	N	Y	Y	Y	N	
11	N	Y	Y	N	N	
12	N	Y	Y	Y	Y	
13	Y	N	Y	Y	N	
14	N	Y	Y	Y	N	
15	Y	Y	Y	Y	N	
16	N	N	Y	Y	N	
17	N	Y	Y	Y	N	
18	N	Y	Y	Y	N	
19	N	Y	Y	Y	Y	

20 rows × 29 columns

Plotting graph with the FLAG columns and the TARGET, to check if there is a correlation between the elements

In [33]:
```
plt.figure(figsize = [20,24])

for i,j in itertools.zip_longest(flag_col, range(len(flag_col))):
    plt.subplot(7,4,j+1)
    ax = sns.countplot(df_flag[i], hue = df_flag["TARGET"], palette =
["b","y"])
    plt.xlabel(" ")
    plt.ylabel(" ")
    plt.title(i)
```



Oservation from above plots:

1. From the above plots it can be seen that the following plots FLAG_MOBIL, FLAG_OWN_REALTY, FLAG_CONT_MOBILE, FLAG_EMP_PHONE, FLAG_DOCUMENT_3 shows that there are more repayers than defaulters. So, we can keep the flag values such as this and eliminate the rest from the data set fo our analysis.

In [34]:
```
#Deciding to retain only the following columns TARGET, FLAG_OWN_REALTY, FLAG_MO
BIL, FLAG_DOCUMENT_3. Because from the above plots it can seen that the above c
olumns facilitate the maximun number of repayers

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

In [35]:
```
len (df_flag.columns)
```

Out[35]:
```
25
```

In [36]:
```
df_flag.columns
```

Out[36]:
```
Index(['FLAG_OWN_CAR', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE',
       'FLAG_PHONE', 'FLAG_EMAIL', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_4',
       'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7',
       'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9', 'FLAG_DOCUMENT_10',
       'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12', 'FLAG_DOCUMENT_13',
       'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_16',
       'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19',
       'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21'],
      dtype='object')
```

In [37]:
```
#Dropping the above columns from our data set as the following columns are not
relevant to the analysis

bank_df.drop(df_flag.columns, axis=1, inplace=True)
```

In [38]:
```
#Find out the number of relevant columns left out in the data set for analysis

bank_df.shape

# We are left with 46 columns for the analysis of the data
```

Out[38]:
```
(15523, 46)
```

Observation:

Finally we are left with 46 columns for our analysis, after cleaning the dataset with unneccesary columns

Imputing the missing values

In this step we will input the missing values with the appropriate values so that we can proceed with the further analysis.

In [39]:
```
null(bank_df).head(20)
```

Out[39]:
```
OCCUPATION_TYPE               31.038
AMT_REQ_CREDIT_BUREAU_YEAR    13.432
AMT_REQ_CREDIT_BUREAU_QRT     13.432
AMT_REQ_CREDIT_BUREAU_MON     13.432
AMT_REQ_CREDIT_BUREAU_WEEK    13.432
AMT_REQ_CREDIT_BUREAU_DAY     13.432
AMT_REQ_CREDIT_BUREAU_HOUR    13.432
NAME_TYPE_SUITE                0.406
DEF_60_CNT_SOCIAL_CIRCLE       0.399
OBS_30_CNT_SOCIAL_CIRCLE       0.399
DEF_30_CNT_SOCIAL_CIRCLE       0.399
OBS_60_CNT_SOCIAL_CIRCLE       0.399
AMT_GOODS_PRICE                0.071
WEEKDAY_APPR_PROCESS_START     0.006
DAYS_LAST_PHONE_CHANGE         0.006
ORGANIZATION_TYPE              0.006
LIVE_CITY_NOT_WORK_CITY        0.006
REG_CITY_NOT_WORK_CITY         0.006
REG_CITY_NOT_LIVE_CITY         0.006
LIVE_REGION_NOT_WORK_REGION    0.006
dtype: float64
```

Observation:

The below 7 columns are missing more than 1 % of data entries. So, imputing values for it to carry on the further data analysis. Please see the below for column details.

1. AMT_REQ_CREDIT_BUREAU_YEAR
2. OCCUPATION_TYPE
3. AMT_REQ_CREDIT_BUREAU_QRT
4. AMT_REQ_CREDIT_BUREAU_MON
5. AMT_REQ_CREDIT_BUREAU_WEEK
6. AMT_REQ_CREDIT_BUREAU_DAY
7. AMT_REQ_CREDIT_BUREAU_HOUR

In [40]:
```
# Imputing the values for the above columns which are missing data more than 1%

bank_df["OCCUPATION_TYPE"].value_counts(normalize=True)*100
```

Out[40]:
```
Laborers                 25.894442
Sales staff              15.544138
Core staff               13.078001
Managers                  9.911256
Drivers                   8.594115
High skill tech staff     5.455395
Accountants               4.605325
Medicine staff            4.194302
Security staff            3.073330
Cooking staff             2.765063
Cleaning staff            2.167212
Private service staff     1.354507
Low-skill Laborers        1.186362
Secretaries               0.644559
Waiters/barmen staff      0.569827
Realty agents             0.392340
HR staff                  0.289584
IT staff                  0.280243
Name: OCCUPATION_TYPE, dtype: float64
```

Observation:

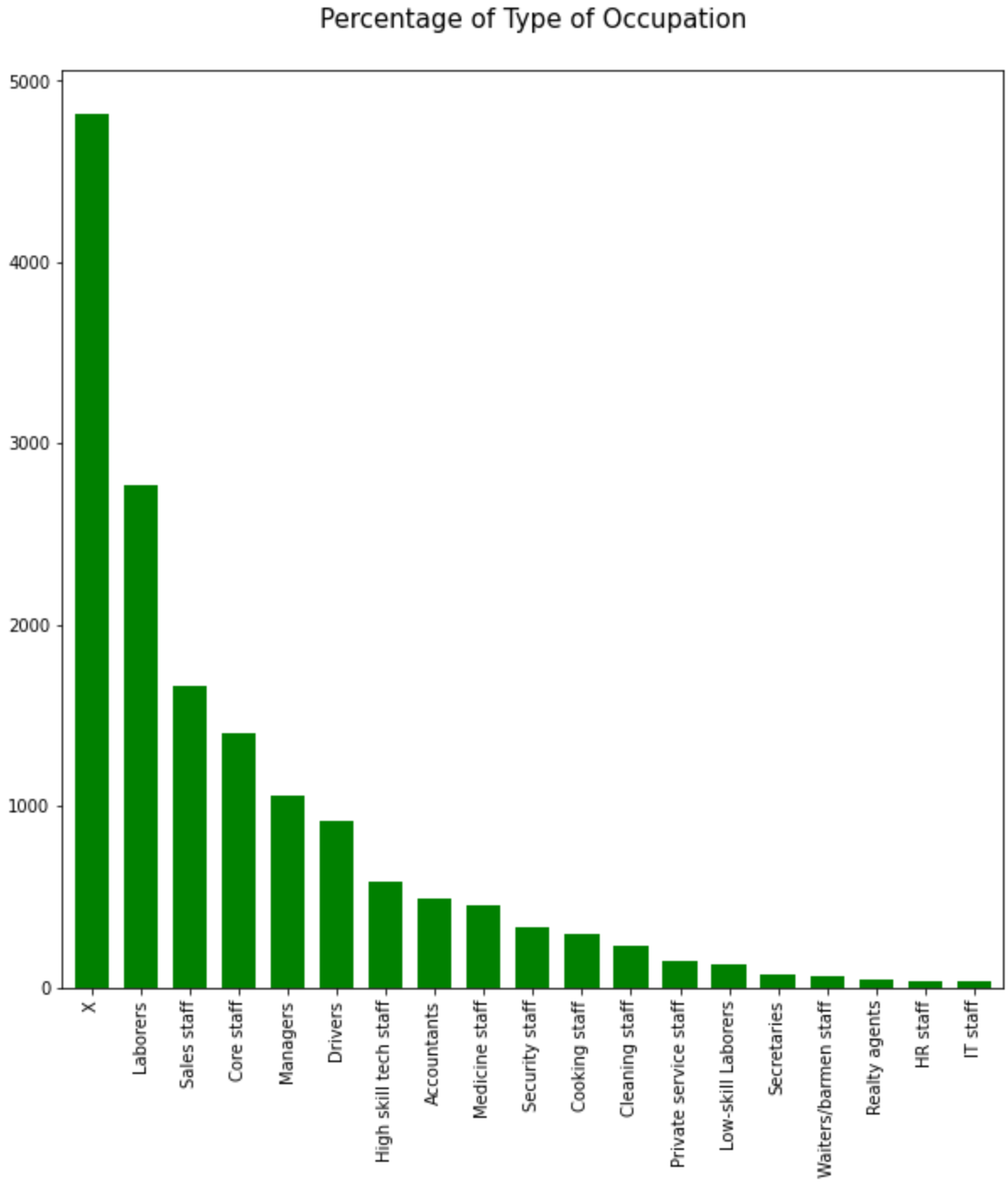
It can be seen from the above analysis that around 32 % of the data is missing value s, which further needs fixing of data. So imputing value "Unknown" for it.

In [41]:
```
# Imputing the missing values with "X"

bank_df["OCCUPATION_TYPE"] = bank_df["OCCUPATION_TYPE"].fillna("X")
```

In [42]:
```
#Checking if  missing values are replaced by the above variable

bank_df["OCCUPATION_TYPE"].isnull().sum()
```

Out[42]:
```
0
```

In [43]:
```
plt.figure(figsize =[10,10])
(bank_df["OCCUPATION_TYPE"].value_counts()).plot.bar(color = "green", width = 0.7)
plt.title("Percentage of Type of Occupation", fontdict={"fontsize":15}, pad=25)
plt.show()
```



Observation:

Close to 90000 applicants occupation is unknown or missing and second highest number of loan applicants are belong to laborers category.

In [44]:
```
# Finding other null value columns and pick the columns necessary for analysis
and impute the median values for further analysis

null(bank_df).head(20)
```

Out[44]:
```
AMT_REQ_CREDIT_BUREAU_YEAR     13.432
AMT_REQ_CREDIT_BUREAU_QRT      13.432
AMT_REQ_CREDIT_BUREAU_MON      13.432
AMT_REQ_CREDIT_BUREAU_WEEK     13.432
AMT_REQ_CREDIT_BUREAU_DAY      13.432
AMT_REQ_CREDIT_BUREAU_HOUR     13.432
NAME_TYPE_SUITE                 0.406
OBS_30_CNT_SOCIAL_CIRCLE        0.399
DEF_30_CNT_SOCIAL_CIRCLE        0.399
DEF_60_CNT_SOCIAL_CIRCLE        0.399
OBS_60_CNT_SOCIAL_CIRCLE        0.399
AMT_GOODS_PRICE                 0.071
REGION_RATING_CLIENT_W_CITY     0.006
DAYS_LAST_PHONE_CHANGE          0.006
FLAG_DOCUMENT_3                 0.006
ORGANIZATION_TYPE               0.006
LIVE_CITY_NOT_WORK_CITY         0.006
REG_CITY_NOT_WORK_CITY          0.006
REG_CITY_NOT_LIVE_CITY          0.006
LIVE_REGION_NOT_WORK_REGION     0.006
dtype: float64
```

In [45]:
```
# Creating a variable for the amount required columns, so we can impute appropi
ate values

amt_req_credit =["AMT_REQ_CREDIT_BUREAU_YEAR", "AMT_REQ_CREDIT_BUREAU_QRT", "AM
T_REQ_CREDIT_BUREAU_MON", "AMT_REQ_CREDIT_BUREAU_WEEK", "AMT_REQ_CREDIT_BUREAU_
DAY", "AMT_REQ_CREDIT_BUREAU_HOUR"]
bank_df[amt_req_credit].describe()
```

Out[45]:

	AMT_REQ_CREDIT_BUREAU_YEAR	AMT_REQ_CREDIT_BUREAU_QRT	AMT_REQ_CREDIT_BUREAU_MOI
count	13438.000000	13438.000000	13438.00000
mean	1.877735	0.264623	0.27392
std	1.848470	0.611529	0.94991
min	0.000000	0.000000	0.00000
25%	0.000000	0.000000	0.00000
50%	1.000000	0.000000	0.00000
75%	3.000000	0.000000	0.00000
max	16.000000	8.000000	24.00000

In [46]:
```
# Filling the missing values in the above mentioned columns with the median val
ues of those columns

bank_df.fillna(bank_df[amt_req_credit].median(), inplace = True)
```

In [47]:
```
# Checking if there are any null values in the "AMT_REQ_CREDIT" related columns

bank_df[amt_req_credit].isnull().sum()

#All the missing values of the above columns are filled with the median value
```

Out[47]:
```
AMT_REQ_CREDIT_BUREAU_YEAR    0
AMT_REQ_CREDIT_BUREAU_QRT     0
AMT_REQ_CREDIT_BUREAU_MON     0
AMT_REQ_CREDIT_BUREAU_WEEK    0
AMT_REQ_CREDIT_BUREAU_DAY     0
AMT_REQ_CREDIT_BUREAU_HOUR    0
dtype: int64
```

In [48]:
```
null(bank_df).head(20)
```

Out[48]:
```
NAME_TYPE_SUITE                0.406
OBS_60_CNT_SOCIAL_CIRCLE       0.399
DEF_60_CNT_SOCIAL_CIRCLE       0.399
DEF_30_CNT_SOCIAL_CIRCLE       0.399
OBS_30_CNT_SOCIAL_CIRCLE       0.399
AMT_GOODS_PRICE                0.071
ORGANIZATION_TYPE              0.006
HOUR_APPR_PROCESS_START        0.006
CNT_FAM_MEMBERS                0.006
REG_REGION_NOT_LIVE_REGION     0.006
REG_REGION_NOT_WORK_REGION     0.006
LIVE_REGION_NOT_WORK_REGION    0.006
REG_CITY_NOT_LIVE_CITY         0.006
REG_CITY_NOT_WORK_CITY         0.006
LIVE_CITY_NOT_WORK_CITY        0.006
REGION_RATING_CLIENT           0.006
WEEKDAY_APPR_PROCESS_START     0.006
DAYS_LAST_PHONE_CHANGE         0.006
FLAG_DOCUMENT_3                0.006
REGION_RATING_CLIENT_W_CITY    0.006
dtype: float64
```

Standardising the Values

In [49]:
```
bank_df.describe()
```

Out[49]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AM
count	15523.000000	15523.000000	15523.000000	1.552300e+04	1.552300e+04	15523.000000	
mean	109049.560523	0.078400	0.421310	1.753807e+05	6.021176e+05	27141.630709	
std	5222.055991	0.268808	0.725172	9.426409e+05	4.041854e+05	14515.037629	
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	2596.500000	
25%	104539.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16474.500000	
50%	109054.000000	0.000000	0.000000	1.440000e+05	5.172660e+05	25042.500000	
75%	113563.500000	0.000000	1.000000	2.025000e+05	8.104748e+05	34731.000000	
max	118116.000000	1.000000	8.000000	1.170000e+08	4.050000e+06	225000.000000	

8 rows × 35 columns

Taking care of the oher high value columns from above and segregating it under the right type of data

In [50]:
```
# Working on the columns AMT_INCOME_TOTAL, AMT_CREDIT, AMT_GOODS_PRICE, as  the
se are high value numeric columns, we are going to place the values of the abov
e columns in categorical bin columns.

# Creating appropiate bins for income columns in terms of Lakhs

bank_df['AMT_INCOME_TOTAL']=bank_df['AMT_INCOME_TOTAL']/100000

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

bank_df['INCOME_RANGE']=pd.cut(bank_df['AMT_INCOME_TOTAL'],bins,labels=slot)
```

In [51]:
```
round((bank_df["INCOME_RANGE"].value_counts(normalize = True)*100).sort_values
(ascending = False),2)
```

Out[51]:
```
1-2 L         49.81
0-1 L         21.49
2-3 L         21.25
3-4 L          4.96
4-5 L          1.66
5-6 L          0.34
6-7 L          0.30
7-8 L          0.08
8-9 L          0.08
10 L Above     0.02
9-10 L         0.01
Name: INCOME_RANGE, dtype: float64
```

In [52]:
```
# Checking the information of the "AMT_INCOME_TOTAL"

bank_df['AMT_INCOME_TOTAL'].describe()
```

Out[52]:
```
count    15523.000000
mean         1.753807
std          9.426409
min          0.256500
25%          1.125000
50%          1.440000
75%          2.025000
max       1170.000000
Name: AMT_INCOME_TOTAL, dtype: float64
```

In [53]:
```
# Creating bins for the credit amount column in terms of Lakhs

bank_df['AMT_CREDIT']= bank_df['AMT_CREDIT']/100000

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

bank_df['CREDIT_RANGE']=pd.cut(bank_df['AMT_CREDIT'],bins,labels=slots)
```

In [54]:
```
round((bank_df["CREDIT_RANGE"].value_counts(normalize = True)*100).sort_values(ascending = False),2)
```

Out[54]:
```
2-3 L         17.39
10 L Above    16.49
4-5 L         10.81
5-6 L         10.49
1-2 L          9.86
3-4 L          8.42
6-7 L          7.78
8-9 L          7.38
7-8L           6.33
9-10 L         2.98
0-1 L          2.07
Name: CREDIT_RANGE, dtype: float64
```

In [55]:
```
# Creating the bins for the price of goods

bank_df['AMT_GOODS_PRICE']= bank_df['AMT_GOODS_PRICE']/100000

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

bank_df['GOODS_PRICE_RANGE']=pd.cut(bank_df['AMT_GOODS_PRICE'],bins,labels=slots)
```

In [56]:
```
round((bank_df["GOODS_PRICE_RANGE"].value_counts(normalize = True)*100).sort_values(ascending = False),2)
```

Out[56]:
```
2L-3L        20.09
4L-5L        18.13
6L-7L        13.20
10L Above    11.20
1L-2L        10.73
8L-9L         7.21
3L-4L         6.97
5L-6L         4.13
0-1L          2.97
7L-8L         2.69
9L-10L        2.68
Name: GOODS_PRICE_RANGE, dtype: float64
```

Dealing with columns :

DAYS_BIRTH, DAYS_EMPLOYED, DAYS_REGISTRATION, DAYS_ID_PUBLISH, DAYS_LAST_PHONE_CHANGE

In [57]:
```
# Working on the columns DAYS_BIRTH, DAYS_EMPLOYED, DAYS_REGISTRATION, DAYS_ID_PUBLISH, DAYS_LAST_PHONE_CHANGE, as  these have negative values, we are going to correct the values of the above columns.

# Creating varibale "column_days" to store all days related values

column_days = [ "DAYS_ID_PUBLISH", "DAYS_REGISTRATION", "DAYS_BIRTH", "DAYS_EMPLOYED", "DAYS_LAST_PHONE_CHANGE"]

bank_df[column_days].describe()
```

Out[57]:

	DAYS_ID_PUBLISH	DAYS_REGISTRATION	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_LAST_PHONE_CHAI
count	15523.000000	15523.000000	15523.000000	15523.000000	15522.000
mean	-2976.703666	-4968.237647	-15996.181473	62832.012047	-952.939
std	1510.537080	3538.714148	4344.671468	140463.808149	823.518
min	-6228.000000	-20981.000000	-25160.000000	-15632.000000	-3983.000
25%	-4287.000000	-7446.500000	-19592.000000	-2798.000000	-1551.000
50%	-3219.000000	-4463.000000	-15747.000000	-1231.000000	-742.000
75%	-1701.500000	-1970.000000	-12333.500000	-290.000000	-260.000
max	0.000000	0.000000	-7689.000000	365243.000000	0.000

Days columns are having negative entries

In [58]:
```
# Correcting the negative day entries using the absolute function, as the negative entries seems to be wrong

bank_df[column_days]= abs(bank_df[column_days])
```

In [59]:
```
bank_df[column_days].describe()
```

Out[59]:

	DAYS_ID_PUBLISH	DAYS_REGISTRATION	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_LAST_PHONE_CHAN
count	15523.000000	15523.000000	15523.000000	15523.000000	15522.000
mean	2976.703666	4968.237647	15996.181473	66766.547768	952.939
std	1510.537080	3538.714148	4344.671468	138636.713702	823.518
min	0.000000	0.000000	7689.000000	11.000000	0.000
25%	1701.500000	1970.000000	12333.500000	936.000000	260.000
50%	3219.000000	4463.000000	15747.000000	2218.000000	742.000
75%	4287.000000	7446.500000	19592.000000	5629.000000	1551.000
max	6228.000000	20981.000000	25160.000000	365243.000000	3983.000

In [60]:
```
# Convert DAYS_BIRTH columns and binning in terms of years for better understan
ding

bank_df["AGE"] = bank_df["DAYS_BIRTH"]/365
bins = [0,20,25,30,35,40,45,50,55,60,100]
slots = ["0-20","20-25","25-30","30-35","35-40","40-45","45-50","50-55","55-6
0","60 Above"]

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

In [61]:
```
bank_df["AGE_GROUP"].value_counts(normalize= True)*100
```

Out[61]:
```
30-35       13.682922
35-40       13.534755
40-45       13.360819
50-55       11.737422
45-50       11.608581
60 Above    11.260710
25-30       10.461895
55-60       10.371707
20-25        3.981189
0-20         0.000000
Name: AGE_GROUP, dtype: float64
```

In [62]:
```
# Convert DAYS_EMPLOYED to Years

bank_df["YEARS_EMPLOYED"] = bank_df["DAYS_EMPLOYED"]/365
bins = [0,5,10,15,20,25,30,50]
slots = ["0-5","5-10","10-15","15-20","20-25","25-30","30 Above"]

bank_df["EMPLOYEMENT_YEARS"] = pd.cut(bank_df["YEARS_EMPLOYED"], bins, labels=s
lots)
```

In [63]:
```
bank_df["EMPLOYEMENT_YEARS"].value_counts(normalize= True)*100
```

Out[63]:
```
0-5          53.723862
5-10         25.859503
10-15        10.901402
15-20         4.456105
20-25         2.553058
25-30         1.323518
30 Above      1.182551
Name: EMPLOYEMENT_YEARS, dtype: float64
```

Identifying Outliers

In [64]:
```
bank_df.describe()
```

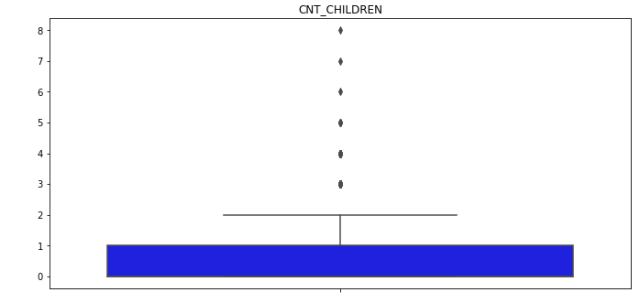
Out[64]:

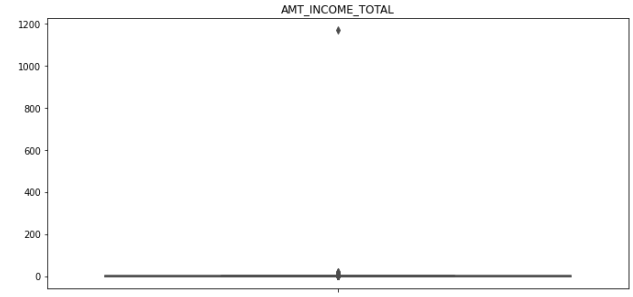
	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AM
count	15523.000000	15523.000000	15523.000000	15523.000000	15523.000000	15523.000000	
mean	109049.560523	0.078400	0.421310	1.753807	6.021176	27141.630709	
std	5222.055991	0.268808	0.725172	9.426409	4.041854	14515.037629	
min	100002.000000	0.000000	0.000000	0.256500	0.450000	2596.500000	
25%	104539.500000	0.000000	0.000000	1.125000	2.700000	16474.500000	
50%	109054.000000	0.000000	0.000000	1.440000	5.172660	25042.500000	
75%	113563.500000	0.000000	1.000000	2.025000	8.104748	34731.000000	
max	118116.000000	1.000000	8.000000	1170.000000	40.500000	225000.000000	

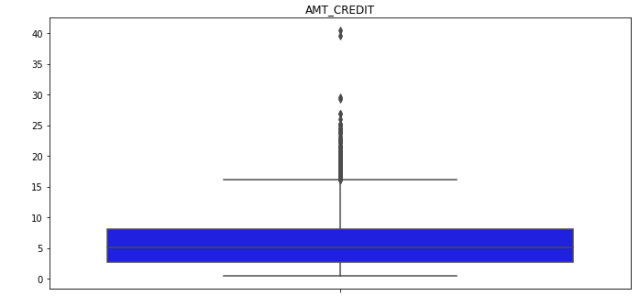
8 rows × 37 columns

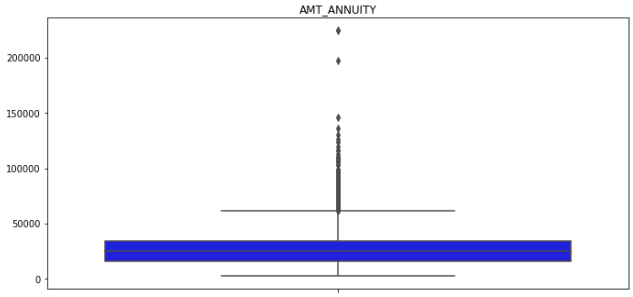
In [65]:
```
# Select the columns that have highesh difference between maximum value and it
75th percentile value to find the outliers for them

outliers= ["CNT_CHILDREN","AMT_INCOME_TOTAL", "AMT_CREDIT", "AMT_ANNUITY", "AMT
_GOODS_PRICE",
                "DAYS_BIRTH", "DAYS_EMPLOYED", "DAYS_REGISTRATION"]
```

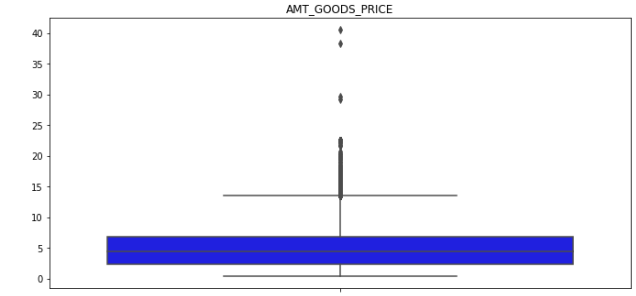
In [66]:
```
plt.figure(figsize=[25,25])
for i,j in itertools.zip_longest(outliers, range(len(outliers))):
    plt.subplot(4,2,j+1)
    sns.boxplot(y = bank_df[i], orient = "h", color = "blue")
    #plt.yticks(fontsize=8)
    plt.xlabel(" ")
    plt.ylabel(" ")
    plt.title(i)
```

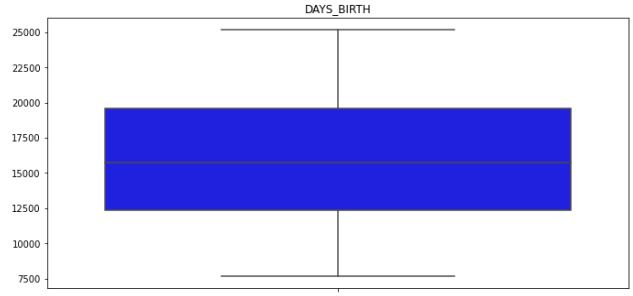


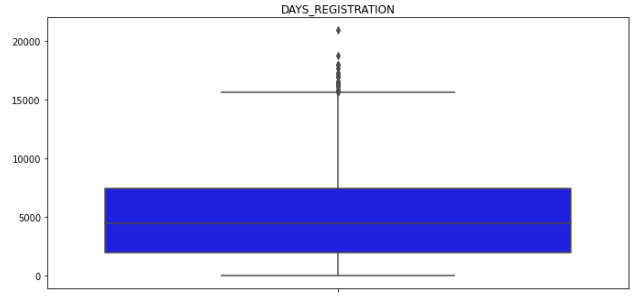












Observations from Above:

Columns that have few outliers : AMT_ANNUITY, AMT_CREDIT, AMT_GOODS_PRICE,CNT_CHILDREN

Columns with the highest number of Outliers : AMT_INCOME_TOTAL (the graph indicates that a few select people have higher income compared to the rest of the members)

The column, DAYS_EMPLOYED have outlier values at 350000 days, which is a humongous amount of days, which indicates that there is mistake while collecting data or some bogus information was provided.

The column, DAYS_BIRTH doesn't have any outlier indicating that it is most relaible inform among the following values.

In [67]:
```
bank_df.nunique().sort_values()
```

Out[67]:
```
FLAG_MOBIL                       1
REG_CITY_NOT_LIVE_CITY           2
TARGET                           2
NAME_CONTRACT_TYPE               2
CODE_GENDER                      2
FLAG_OWN_REALTY                  2
FLAG_DOCUMENT_3                  2
LIVE_CITY_NOT_WORK_CITY          2
REG_CITY_NOT_WORK_CITY           2
REG_REGION_NOT_LIVE_REGION       2
LIVE_REGION_NOT_WORK_REGION      2
REG_REGION_NOT_WORK_REGION       2
REGION_RATING_CLIENT             3
REGION_RATING_CLIENT_W_CITY      3
AMT_REQ_CREDIT_BUREAU_HOUR       3
NAME_FAMILY_STATUS               5
NAME_EDUCATION_TYPE              5
AMT_REQ_CREDIT_BUREAU_DAY        6
NAME_INCOME_TYPE                 6
DEF_60_CNT_SOCIAL_CIRCLE         6
NAME_HOUSING_TYPE                6
AMT_REQ_CREDIT_BUREAU_WEEK       6
WEEKDAY_APPR_PROCESS_START       7
EMPLOYEMENT_YEARS                7
DEF_30_CNT_SOCIAL_CIRCLE         7
NAME_TYPE_SUITE                  7
AMT_REQ_CREDIT_BUREAU_QRT        9
AGE_GROUP                        9
CNT_CHILDREN                     9
CNT_FAM_MEMBERS                 10
INCOME_RANGE                    11
CREDIT_RANGE                    11
GOODS_PRICE_RANGE               11
AMT_REQ_CREDIT_BUREAU_YEAR      14
AMT_REQ_CREDIT_BUREAU_MON       18
OCCUPATION_TYPE                 19
HOUR_APPR_PROCESS_START         23
OBS_30_CNT_SOCIAL_CIRCLE        23
OBS_60_CNT_SOCIAL_CIRCLE        24
ORGANIZATION_TYPE               58
REGION_POPULATION_RELATIVE      80
AMT_GOODS_PRICE                391
AMT_INCOME_TOTAL               406
AMT_CREDIT                    2248
DAYS_LAST_PHONE_CHANGE        2950
DAYS_ID_PUBLISH               5046
YEARS_EMPLOYED                5263
DAYS_EMPLOYED                 5263
AMT_ANNUITY                   5347
DAYS_REGISTRATION             8627
DAYS_BIRTH                    9927
AGE                           9927
SK_ID_CURR                   15523
dtype: int64
```

In [68]:
```
# Trying to find out the unique values in each columns

bank_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15523 entries, 0 to 15522
Data columns (total 53 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   SK_ID_CURR                    15523 non-null  int64
 1   TARGET                        15523 non-null  int64
 2   NAME_CONTRACT_TYPE            15523 non-null  object
 3   CODE_GENDER                   15523 non-null  object
 4   FLAG_OWN_REALTY               15523 non-null  object
 5   CNT_CHILDREN                  15523 non-null  int64
 6   AMT_INCOME_TOTAL              15523 non-null  float64
 7   AMT_CREDIT                    15523 non-null  float64
 8   AMT_ANNUITY                   15523 non-null  float64
 9   AMT_GOODS_PRICE               15512 non-null  float64
 10  NAME_TYPE_SUITE               15460 non-null  object
 11  NAME_INCOME_TYPE              15523 non-null  object
 12  NAME_EDUCATION_TYPE           15523 non-null  object
 13  NAME_FAMILY_STATUS            15523 non-null  object
 14  NAME_HOUSING_TYPE             15523 non-null  object
 15  REGION_POPULATION_RELATIVE    15523 non-null  float64
 16  DAYS_BIRTH                    15523 non-null  int64
 17  DAYS_EMPLOYED                 15523 non-null  int64
 18  DAYS_REGISTRATION             15523 non-null  float64
 19  DAYS_ID_PUBLISH               15523 non-null  int64
 20  FLAG_MOBIL                    15523 non-null  int64
 21  OCCUPATION_TYPE               15523 non-null  object
 22  CNT_FAM_MEMBERS               15522 non-null  float64
 23  REGION_RATING_CLIENT          15522 non-null  float64
 24  REGION_RATING_CLIENT_W_CITY   15522 non-null  float64
 25  WEEKDAY_APPR_PROCESS_START    15522 non-null  object
 26  HOUR_APPR_PROCESS_START       15522 non-null  float64
 27  REG_REGION_NOT_LIVE_REGION    15522 non-null  float64
 28  REG_REGION_NOT_WORK_REGION    15522 non-null  float64
 29  LIVE_REGION_NOT_WORK_REGION   15522 non-null  float64
 30  REG_CITY_NOT_LIVE_CITY        15522 non-null  float64
 31  REG_CITY_NOT_WORK_CITY        15522 non-null  float64
 32  LIVE_CITY_NOT_WORK_CITY       15522 non-null  float64
 33  ORGANIZATION_TYPE             15522 non-null  object
 34  OBS_30_CNT_SOCIAL_CIRCLE      15461 non-null  float64
 35  DEF_30_CNT_SOCIAL_CIRCLE      15461 non-null  float64
 36  OBS_60_CNT_SOCIAL_CIRCLE      15461 non-null  float64
 37  DEF_60_CNT_SOCIAL_CIRCLE      15461 non-null  float64
 38  DAYS_LAST_PHONE_CHANGE        15522 non-null  float64
 39  FLAG_DOCUMENT_3               15522 non-null  float64
 40  AMT_REQ_CREDIT_BUREAU_HOUR    15523 non-null  float64
 41  AMT_REQ_CREDIT_BUREAU_DAY     15523 non-null  float64
 42  AMT_REQ_CREDIT_BUREAU_WEEK    15523 non-null  float64
 43  AMT_REQ_CREDIT_BUREAU_MON     15523 non-null  float64
 44  AMT_REQ_CREDIT_BUREAU_QRT     15523 non-null  float64
 45  AMT_REQ_CREDIT_BUREAU_YEAR    15523 non-null  float64
 46  INCOME_RANGE                  15513 non-null  category
 47  CREDIT_RANGE                  15523 non-null  category
 48  GOODS_PRICE_RANGE             15512 non-null  category
 49  AGE                           15523 non-null  float64
 50  AGE_GROUP                     15523 non-null  category
 51  YEARS_EMPLOYED                15523 non-null  float64
 52  EMPLOYEMENT_YEARS             12769 non-null  category
dtypes: category(5), float64(30), int64(7), object(11)
memory usage: 5.8+ MB
```

Converting selective columns from Object to categorical column

In [69]:
```
bank_df.columns
```

Out[69]:
```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT',
       'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE',
       'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE',
       'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED',
       'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'FLAG_MOBIL', 'OCCUPATION_TYPE',
       'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT',
       'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_START',
       'HOUR_APPR_PROCESS_START', 'REG_REGION_NOT_LIVE_REGION',
       'REG_REGION_NOT_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION',
       'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY',
       'LIVE_CITY_NOT_WORK_CITY', 'ORGANIZATION_TYPE',
       'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE',
       'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE',
       'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_3',
       'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY',
       'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON',
       'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR',
       'INCOME_RANGE', 'CREDIT_RANGE', 'GOODS_PRICE_RANGE', 'AGE', 'AGE_GROUP',
       'YEARS_EMPLOYED', 'EMPLOYEMENT_YEARS'],
      dtype='object')
```

In [70]:
```
# Converting datatypes of below columns Categorical data

category_col = ['FLAG_OWN_REALTY','LIVE_CITY_NOT_WORK_CITY', 'REG_CITY_NOT_LIVE
_CITY','REG_CITY_NOT_WORK_CITY', 'NAME_CONTRACT_TYPE','CODE_GENDER','NAME_TYPE_
SUITE','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS','NAME_HOU
SING_TYPE','OCCUPATION_TYPE','WEEKDAY_APPR_PROCESS_START', 'ORGANIZATION_TYP
E','REG_REGION_NOT_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION', 'CNT_CHILDRE
N','CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT','WEEKDAY_APPR_PROCESS_START', 'REG
ION_RATING_CLIENT_W_CITY',]

for column in category_col:
    bank_df[column] = pd.Categorical(bank_df[column])
```

In [71]:
```
len (category_col)
```

Out[71]:
```
21
```

In [72]:
```
bank_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15523 entries, 0 to 15522
Data columns (total 53 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   SK_ID_CURR                    15523 non-null  int64
 1   TARGET                        15523 non-null  int64
 2   NAME_CONTRACT_TYPE            15523 non-null  category
 3   CODE_GENDER                   15523 non-null  category
 4   FLAG_OWN_REALTY               15523 non-null  category
 5   CNT_CHILDREN                  15523 non-null  category
 6   AMT_INCOME_TOTAL              15523 non-null  float64
 7   AMT_CREDIT                    15523 non-null  float64
 8   AMT_ANNUITY                   15523 non-null  float64
 9   AMT_GOODS_PRICE               15512 non-null  float64
 10  NAME_TYPE_SUITE               15460 non-null  category
 11  NAME_INCOME_TYPE              15523 non-null  category
 12  NAME_EDUCATION_TYPE           15523 non-null  category
 13  NAME_FAMILY_STATUS            15523 non-null  category
 14  NAME_HOUSING_TYPE             15523 non-null  category
 15  REGION_POPULATION_RELATIVE    15523 non-null  float64
 16  DAYS_BIRTH                    15523 non-null  int64
 17  DAYS_EMPLOYED                 15523 non-null  int64
 18  DAYS_REGISTRATION             15523 non-null  float64
 19  DAYS_ID_PUBLISH               15523 non-null  int64
 20  FLAG_MOBIL                    15523 non-null  int64
 21  OCCUPATION_TYPE               15523 non-null  category
 22  CNT_FAM_MEMBERS               15522 non-null  category
 23  REGION_RATING_CLIENT          15522 non-null  category
 24  REGION_RATING_CLIENT_W_CITY   15522 non-null  category
 25  WEEKDAY_APPR_PROCESS_START    15522 non-null  category
 26  HOUR_APPR_PROCESS_START       15522 non-null  float64
 27  REG_REGION_NOT_LIVE_REGION    15522 non-null  float64
 28  REG_REGION_NOT_WORK_REGION    15522 non-null  category
 29  LIVE_REGION_NOT_WORK_REGION   15522 non-null  category
 30  REG_CITY_NOT_LIVE_CITY        15522 non-null  category
 31  REG_CITY_NOT_WORK_CITY        15522 non-null  category
 32  LIVE_CITY_NOT_WORK_CITY       15522 non-null  category
 33  ORGANIZATION_TYPE             15522 non-null  category
 34  OBS_30_CNT_SOCIAL_CIRCLE      15461 non-null  float64
 35  DEF_30_CNT_SOCIAL_CIRCLE      15461 non-null  float64
 36  OBS_60_CNT_SOCIAL_CIRCLE      15461 non-null  float64
 37  DEF_60_CNT_SOCIAL_CIRCLE      15461 non-null  float64
 38  DAYS_LAST_PHONE_CHANGE        15522 non-null  float64
 39  FLAG_DOCUMENT_3               15522 non-null  float64
 40  AMT_REQ_CREDIT_BUREAU_HOUR    15523 non-null  float64
 41  AMT_REQ_CREDIT_BUREAU_DAY     15523 non-null  float64
 42  AMT_REQ_CREDIT_BUREAU_WEEK    15523 non-null  float64
 43  AMT_REQ_CREDIT_BUREAU_MON     15523 non-null  float64
 44  AMT_REQ_CREDIT_BUREAU_QRT     15523 non-null  float64
 45  AMT_REQ_CREDIT_BUREAU_YEAR    15523 non-null  float64
 46  INCOME_RANGE                  15513 non-null  category
 47  CREDIT_RANGE                  15523 non-null  category
 48  GOODS_PRICE_RANGE             15512 non-null  category
 49  AGE                           15523 non-null  float64
 50  AGE_GROUP                     15523 non-null  category
 51  YEARS_EMPLOYED                15523 non-null  float64
 52  EMPLOYEMENT_YEARS             12769 non-null  category
dtypes: category(25), float64(22), int64(6)
memory usage: 3.7 MB
```

Observation from above:

After removing all the unnecessary and imputing the missing values, there are a total of 53 columns with which we can proceed for Data Analysis.

Dataset 2 - "Previous _Application.csv"

Variable name: pre_df

In [74]:
```
# Importing the previous_application.csv

pre_df = pd.read_csv("previous_application.csv")
```

In [75]:
```
pre_df.head(20)
```

Out[75]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0
5	1383531	199383	Cash loans	23703.930	315000.0	340573.5
6	2315218	175704	Cash loans	NaN	0.0	0.0
7	1656711	296299	Cash loans	NaN	0.0	0.0
8	2367563	342292	Cash loans	NaN	0.0	0.0
9	2579447	334349	Cash loans	NaN	0.0	0.0
10	1715995	447712	Cash loans	11368.620	270000.0	335754.0
11	2257824	161140	Cash loans	13832.775	211500.0	246397.5
12	2330894	258628	Cash loans	12165.210	148500.0	174361.5
13	1397919	321676	Consumer loans	7654.860	53779.5	57564.0
14	2273188	270658	Consumer loans	9644.220	26550.0	27252.0
15	1232483	151612	Consumer loans	21307.455	126490.5	119853.0
16	2163253	154602	Consumer loans	4187.340	26955.0	27297.0
17	1285768	142748	Revolving loans	9000.000	180000.0	180000.0
18	2393109	396305	Cash loans	10181.700	180000.0	180000.0
19	1173070	199178	Cash loans	4666.500	45000.0	49455.0

20 rows × 37 columns

In [76]:
```
# Checking the data sets rows amd columns

pre_df.shape
```

Out[76]:
```
(175483, 37)
```

In [77]:
```
# Understanding the type of data in all the columns

pre_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175483 entries, 0 to 175482
Data columns (total 37 columns):
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   SK_ID_PREV                   175483 non-null  int64  
 1   SK_ID_CURR                   175483 non-null  int64  
 2   NAME_CONTRACT_TYPE           175483 non-null  object 
 3   AMT_ANNUITY                  138000 non-null  float64
 4   AMT_APPLICATION              175483 non-null  float64
 5   AMT_CREDIT                   175483 non-null  float64
 6   AMT_DOWN_PAYMENT             85709 non-null   float64
 7   AMT_GOODS_PRICE              137026 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   175483 non-null  object 
 9   HOUR_APPR_PROCESS_START      175483 non-null  int64  
 10  FLAG_LAST_APPL_PER_CONTRACT  175483 non-null  object 
 11  NFLAG_LAST_APPL_IN_DAY       175483 non-null  int64  
 12  RATE_DOWN_PAYMENT            85709 non-null   float64
 13  RATE_INTEREST_PRIMARY        627 non-null     float64
 14  RATE_INTEREST_PRIVILEGED     627 non-null     float64
 15  NAME_CASH_LOAN_PURPOSE       175483 non-null  object 
 16  NAME_CONTRACT_STATUS         175483 non-null  object 
 17  DAYS_DECISION                175483 non-null  int64  
 18  NAME_PAYMENT_TYPE            175483 non-null  object 
 19  CODE_REJECT_REASON           175482 non-null  object 
 20  NAME_TYPE_SUITE              90183 non-null   object 
 21  NAME_CLIENT_TYPE             175482 non-null  object 
 22  NAME_GOODS_CATEGORY          175482 non-null  object 
 23  NAME_PORTFOLIO               175482 non-null  object 
 24  NAME_PRODUCT_TYPE            175482 non-null  object 
 25  CHANNEL_TYPE                 175482 non-null  object 
 26  SELLERPLACE_AREA             175482 non-null  float64
 27  NAME_SELLER_INDUSTRY         175482 non-null  object 
 28  CNT_PAYMENT                  137999 non-null  float64
 29  NAME_YIELD_GROUP             175482 non-null  object 
 30  PRODUCT_COMBINATION          175446 non-null  object 
 31  DAYS_FIRST_DRAWING           107767 non-null  float64
 32  DAYS_FIRST_DUE               107767 non-null  float64
 33  DAYS_LAST_DUE_1ST_VERSION    107767 non-null  float64
 34  DAYS_LAST_DUE                107767 non-null  float64
 35  DAYS_TERMINATION             107767 non-null  float64
 36  NFLAG_INSURED_ON_APPROVAL    107767 non-null  float64
dtypes: float64(16), int64(5), object(16)
memory usage: 49.5+ MB
```

In [78]:
```
# Checking the datframes for more specific information

pre_df.describe()
```

Out[78]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT
count	1.754830e+05	175483.000000	138000.000000	1.754830e+05	1.754830e+05	8.570900e+04
mean	1.919434e+06	278661.624830	15514.995593	1.695229e+05	1.892379e+05	6.650545e+03
std	5.344200e+05	102834.352184	14508.783968	2.835496e+05	3.096062e+05	1.831351e+04
min	1.000001e+06	100006.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00
25%	1.455628e+06	189721.500000	6164.403750	2.111850e+04	2.546100e+04	0.000000e+00
50%	1.918764e+06	278907.000000	10941.030000	7.101000e+04	7.902450e+04	1.665000e+03
75%	2.382861e+06	368138.000000	19795.770000	1.800000e+05	2.015820e+05	7.740000e+03
max	2.845377e+06	456254.000000	417927.645000	3.826372e+06	4.104351e+06	1.201500e+06

8 rows × 21 columns

Observation from above:

Some columns above have negative values which should be corrected, the columns AMT_DOWN_PAYMENT, DAYS_FIRST_DRAWING, etc. which needs fixing.

In [79]:
```
# Checking for the null values using the function that we hav defined in the st
art of the analysis.

null(pre_df)
```

Out[79]:
```
RATE_INTEREST_PRIVILEGED       99.643
RATE_INTEREST_PRIMARY          99.643
RATE_DOWN_PAYMENT              51.158
AMT_DOWN_PAYMENT               51.158
NAME_TYPE_SUITE                48.609
DAYS_FIRST_DUE                 38.588
DAYS_FIRST_DRAWING             38.588
NFLAG_INSURED_ON_APPROVAL      38.588
DAYS_LAST_DUE_1ST_VERSION      38.588
DAYS_LAST_DUE                  38.588
DAYS_TERMINATION               38.588
AMT_GOODS_PRICE                21.915
CNT_PAYMENT                    21.360
AMT_ANNUITY                    21.360
PRODUCT_COMBINATION             0.021
NAME_YIELD_GROUP                0.001
NAME_PRODUCT_TYPE               0.001
NAME_SELLER_INDUSTRY            0.001
SELLERPLACE_AREA                0.001
CHANNEL_TYPE                    0.001
CODE_REJECT_REASON              0.001
NAME_CLIENT_TYPE                0.001
NAME_GOODS_CATEGORY             0.001
NAME_PORTFOLIO                  0.001
SK_ID_PREV                      0.000
SK_ID_CURR                      0.000
DAYS_DECISION                   0.000
NAME_CONTRACT_STATUS            0.000
NAME_CASH_LOAN_PURPOSE          0.000
NFLAG_LAST_APPL_IN_DAY          0.000
FLAG_LAST_APPL_PER_CONTRACT     0.000
HOUR_APPR_PROCESS_START         0.000
WEEKDAY_APPR_PROCESS_START      0.000
AMT_CREDIT                      0.000
AMT_APPLICATION                 0.000
NAME_CONTRACT_TYPE              0.000
NAME_PAYMENT_TYPE               0.000
dtype: float64
```

Finding cells missing more than 50% of the data

In [80]:
```
#Creating a variable for storing all column missing more than 50 % data.

pre_null_grt50 = null(pre_df)[null(pre_df)>50]
pre_null_grt50
```

Out[80]:
```
RATE_INTEREST_PRIVILEGED    99.643
RATE_INTEREST_PRIMARY       99.643
RATE_DOWN_PAYMENT           51.158
AMT_DOWN_PAYMENT            51.158
dtype: float64
```

In [81]:
```
#drop the columns that have data missing more than 50 %

pre_df.drop(columns = pre_null_grt50.index, inplace = True)
```

In [82]:
```
# Confirming if the above columns are dropped from the dataset
pre_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175483 entries, 0 to 175482
Data columns (total 33 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   SK_ID_PREV                   175483 non-null  int64
 1   SK_ID_CURR                   175483 non-null  int64
 2   NAME_CONTRACT_TYPE           175483 non-null  object
 3   AMT_ANNUITY                  138000 non-null  float64
 4   AMT_APPLICATION              175483 non-null  float64
 5   AMT_CREDIT                   175483 non-null  float64
 6   AMT_GOODS_PRICE              137026 non-null  float64
 7   WEEKDAY_APPR_PROCESS_START   175483 non-null  object
 8   HOUR_APPR_PROCESS_START      175483 non-null  int64
 9   FLAG_LAST_APPL_PER_CONTRACT  175483 non-null  object
 10  NFLAG_LAST_APPL_IN_DAY       175483 non-null  int64
 11  NAME_CASH_LOAN_PURPOSE       175483 non-null  object
 12  NAME_CONTRACT_STATUS         175483 non-null  object
 13  DAYS_DECISION                175483 non-null  int64
 14  NAME_PAYMENT_TYPE            175483 non-null  object
 15  CODE_REJECT_REASON           175482 non-null  object
 16  NAME_TYPE_SUITE              90183 non-null   object
 17  NAME_CLIENT_TYPE             175482 non-null  object
 18  NAME_GOODS_CATEGORY          175482 non-null  object
 19  NAME_PORTFOLIO               175482 non-null  object
 20  NAME_PRODUCT_TYPE            175482 non-null  object
 21  CHANNEL_TYPE                 175482 non-null  object
 22  SELLERPLACE_AREA             175482 non-null  float64
 23  NAME_SELLER_INDUSTRY         175482 non-null  object
 24  CNT_PAYMENT                  137999 non-null  float64
 25  NAME_YIELD_GROUP             175482 non-null  object
 26  PRODUCT_COMBINATION          175446 non-null  object
 27  DAYS_FIRST_DRAWING           107767 non-null  float64
 28  DAYS_FIRST_DUE               107767 non-null  float64
 29  DAYS_LAST_DUE_1ST_VERSION    107767 non-null  float64
 30  DAYS_LAST_DUE                107767 non-null  float64
 31  DAYS_TERMINATION             107767 non-null  float64
 32  NFLAG_INSURED_ON_APPROVAL    107767 non-null  float64
dtypes: float64(12), int64(5), object(16)
memory usage: 44.2+ MB
```

In [83]:
```
#Creating a variable for storing all column missing more than 20 % data.

pre_null_grt20 = null(pre_df)[null(pre_df)>20]
pre_null_grt20
```

Out[83]:
```
NAME_TYPE_SUITE              48.609
DAYS_FIRST_DRAWING           38.588
DAYS_TERMINATION             38.588
DAYS_LAST_DUE                38.588
DAYS_LAST_DUE_1ST_VERSION    38.588
DAYS_FIRST_DUE               38.588
NFLAG_INSURED_ON_APPROVAL    38.588
AMT_GOODS_PRICE              21.915
CNT_PAYMENT                  21.360
AMT_ANNUITY                  21.360
dtype: float64
```

In [84]:
```
pre_df[pre_null_grt20.index]
```

Out[84]:

	NAME_TYPE_SUITE	DAYS_FIRST_DRAWING	DAYS_TERMINATION	DAYS_LAST_DUE	DAYS_LAST_DU
0	NaN	365243.0	-37.0	-42.0	
1	Unaccompanied	365243.0	365243.0	365243.0	
2	Spouse, partner	365243.0	365243.0	365243.0	
3	NaN	365243.0	-177.0	-182.0	
4	NaN	NaN	NaN	NaN	
...	...	...	...	...	
175478	Children	NaN	NaN	NaN	
175479	Unaccompanied	NaN	NaN	NaN	
175480	NaN	NaN	NaN	NaN	
175481	Unaccompanied	365243.0	-499.0	-502.0	
175482	NaN	NaN	NaN	NaN	

175483 rows × 10 columns

In [85]:
```
pre_df.columns
```

Out[85]:
```
Index(['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_TYPE_SUITE',
       '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', 'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE',
       'DAYS_LAST_DUE_1ST_VERSION', 'DAYS_LAST_DUE', 'DAYS_TERMINATION',
       'NFLAG_INSURED_ON_APPROVAL'],
      dtype='object')
```

In [86]:
```
# Storing the non-necessary columns from the data set

unnecessary_col = ['WEEKDAY_APPR_PROCESS_START','HOUR_APPR_PROCESS_START','NFLA
G_LAST_APPL_IN_DAY', 'FLAG_LAST_APPL_PER_CONTRACT']

pre_df.drop(unnecessary_col, axis=1, inplace = True)

pre_df.shape
```

Out[86]:
```
(175483, 29)
```

In [87]:
```
# Imputing the missing value with "Unknown"

pre_df["NAME_TYPE_SUITE"] = pre_df["NAME_TYPE_SUITE"].fillna("Unknown")

null(pre_df)
```

Out[87]:
```
NFLAG_INSURED_ON_APPROVAL    38.588
DAYS_TERMINATION             38.588
DAYS_LAST_DUE                38.588
DAYS_LAST_DUE_1ST_VERSION    38.588
DAYS_FIRST_DUE               38.588
DAYS_FIRST_DRAWING           38.588
AMT_GOODS_PRICE              21.915
CNT_PAYMENT                  21.360
AMT_ANNUITY                  21.360
PRODUCT_COMBINATION           0.021
NAME_PRODUCT_TYPE             0.001
NAME_YIELD_GROUP              0.001
NAME_SELLER_INDUSTRY          0.001
SELLERPLACE_AREA              0.001
CHANNEL_TYPE                  0.001
NAME_GOODS_CATEGORY           0.001
NAME_PORTFOLIO                0.001
NAME_CLIENT_TYPE              0.001
CODE_REJECT_REASON            0.001
SK_ID_CURR                    0.000
NAME_TYPE_SUITE               0.000
NAME_PAYMENT_TYPE             0.000
DAYS_DECISION                 0.000
NAME_CONTRACT_STATUS          0.000
NAME_CASH_LOAN_PURPOSE        0.000
AMT_CREDIT                    0.000
AMT_APPLICATION               0.000
NAME_CONTRACT_TYPE            0.000
SK_ID_PREV                    0.000
dtype: float64
```

Observation from above:

The following columns have missing values which should be takem care before starting the analysis 'DAYS_TERMINATION', 'DAYS_LAST_DUE', 'DAYS_FIRST_DRAWING','DAYS_LAST_DUE_1ST_VERSION', 'DAYS_FIRST_DUE'.

In [88]:
```
# Analysing the data using describe

pre_df[pre_null_grt20.index].describe()
```

Out[88]:

	DAYS_FIRST_DRAWING	DAYS_TERMINATION	DAYS_LAST_DUE	DAYS_LAST_DUE_1ST_VERSION	DAYS
count	107767.000000	107767.000000	107767.000000	107767.000000	1
mean	343506.426754	81355.756883	76384.265044	32480.185604	
std	86536.071083	152893.584830	149514.644165	105061.623879	
min	-2920.000000	-2847.000000	-2888.000000	-2800.000000	
25%	365243.000000	-1292.500000	-1335.000000	-1266.000000	
50%	365243.000000	-502.000000	-538.000000	-369.000000	
75%	365243.000000	-47.000000	-73.000000	114.000000	
max	365243.000000	365243.000000	365243.000000	365243.000000	3

In [89]:
```
# Creating a variable to convert the negative date entries to positive using va
riable "pre_days"

pre_day = ['DAYS_DECISION','DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_D
UE', 'DAYS_LAST_DUE_1ST_VERSION', 'DAYS_TERMINATION']

pre_df[pre_day].describe()
```

Out[89]:

	DAYS_DECISION	DAYS_FIRST_DRAWING	DAYS_FIRST_DUE	DAYS_LAST_DUE	DAYS_LAST_DUE_1ST_
count	175483.000000	107767.000000	107767.000000	107767.000000	10776
mean	-899.189072	343506.426754	14031.917795	76384.265044	3248
std	786.748470	86536.071083	72927.719854	149514.644165	10506
min	-2922.000000	-2920.000000	-2891.000000	-2888.000000	-280
25%	-1341.000000	365243.000000	-1645.000000	-1335.000000	-126
50%	-598.000000	365243.000000	-828.000000	-538.000000	-36
75%	-288.000000	365243.000000	-406.000000	-73.000000	11
max	-2.000000	365243.000000	365243.000000	365243.000000	3652

In [90]:
```
# Converting the negative values under day to positive using the absolute funct
ion

pre_df[pre_day] = abs(pre_df[pre_day])

pre_df[pre_day].describe()
```

Out[90]:

	DAYS_DECISION	DAYS_FIRST_DRAWING	DAYS_FIRST_DUE	DAYS_LAST_DUE	DAYS_LAST_DUE_1ST_
count	175483.000000	107767.000000	107767.000000	107767.000000	1077
mean	899.189072	343625.035623	16159.000056	77966.294088	338
std	786.748470	86063.877187	72486.089434	148695.795120	1046
min	2.000000	2.000000	2.000000	2.000000	
25%	288.000000	365243.000000	471.000000	455.000000	2
50%	598.000000	365243.000000	921.000000	1166.000000	7
75%	1341.000000	365243.000000	1844.500000	2422.500000	17
max	2922.000000	365243.000000	365243.000000	365243.000000	3652

In [91]:
```
# Grouping the days

bins = [0,1*365,2*365,3*365,4*365,5*365,6*365,7*365,10*365]
slots = ["1","2","3","4","5","6","7","7 above"]
pre_df['YEARLY_DECISION'] = pd.cut(pre_df['DAYS_DECISION'],bins,labels=slots)
```

In [92]:
```
pre_df['YEARLY_DECISION'].value_counts(normalize=True)*100
```

Out[92]:
```
1          33.402666
2          23.202248
3          12.822895
4           7.875407
5           6.282660
7           6.078651
7 above     5.329861
6           5.005613
Name: YEARLY_DECISION, dtype: float64
```

Observation:

From above, it is clearly seen that 34% of the applicants have applied for a new loa
n within a years time.

In [93]:
```
pre_df.nunique()
```

Out[93]:
```
SK_ID_PREV                   175483
SK_ID_CURR                   124504
NAME_CONTRACT_TYPE                4
AMT_ANNUITY                   82858
AMT_APPLICATION               30632
AMT_CREDIT                    38777
AMT_GOODS_PRICE               30632
NAME_CASH_LOAN_PURPOSE           25
NAME_CONTRACT_STATUS              4
DAYS_DECISION                  2921
NAME_PAYMENT_TYPE                 5
CODE_REJECT_REASON                9
NAME_TYPE_SUITE                   8
NAME_CLIENT_TYPE                  4
NAME_GOODS_CATEGORY              27
NAME_PORTFOLIO                    5
NAME_PRODUCT_TYPE                 3
CHANNEL_TYPE                      8
SELLERPLACE_AREA               1633
NAME_SELLER_INDUSTRY             11
CNT_PAYMENT                      37
NAME_YIELD_GROUP                  5
PRODUCT_COMBINATION              17
DAYS_FIRST_DRAWING             2040
DAYS_FIRST_DUE                 2891
DAYS_LAST_DUE_1ST_VERSION      2796
DAYS_LAST_DUE                  2822
DAYS_TERMINATION               2767
NFLAG_INSURED_ON_APPROVAL         2
YEARLY_DECISION                   8
dtype: int64
```

In [94]:
```
# Finding if there is any null value in the column

null (pre_df)
```

Out[94]:
```
DAYS_FIRST_DRAWING            38.588
NFLAG_INSURED_ON_APPROVAL     38.588
DAYS_TERMINATION              38.588
DAYS_LAST_DUE                 38.588
DAYS_LAST_DUE_1ST_VERSION     38.588
DAYS_FIRST_DUE                38.588
AMT_GOODS_PRICE               21.915
CNT_PAYMENT                   21.360
AMT_ANNUITY                   21.360
PRODUCT_COMBINATION            0.021
NAME_GOODS_CATEGORY            0.001
CHANNEL_TYPE                   0.001
NAME_PRODUCT_TYPE              0.001
NAME_PORTFOLIO                 0.001
NAME_CLIENT_TYPE               0.001
NAME_YIELD_GROUP               0.001
CODE_REJECT_REASON             0.001
NAME_SELLER_INDUSTRY           0.001
SELLERPLACE_AREA               0.001
SK_ID_PREV                     0.000
SK_ID_CURR                     0.000
NAME_TYPE_SUITE                0.000
NAME_PAYMENT_TYPE              0.000
DAYS_DECISION                  0.000
NAME_CONTRACT_STATUS           0.000
NAME_CASH_LOAN_PURPOSE         0.000
AMT_CREDIT                     0.000
AMT_APPLICATION                0.000
NAME_CONTRACT_TYPE             0.000
YEARLY_DECISION                0.000
dtype: float64
```

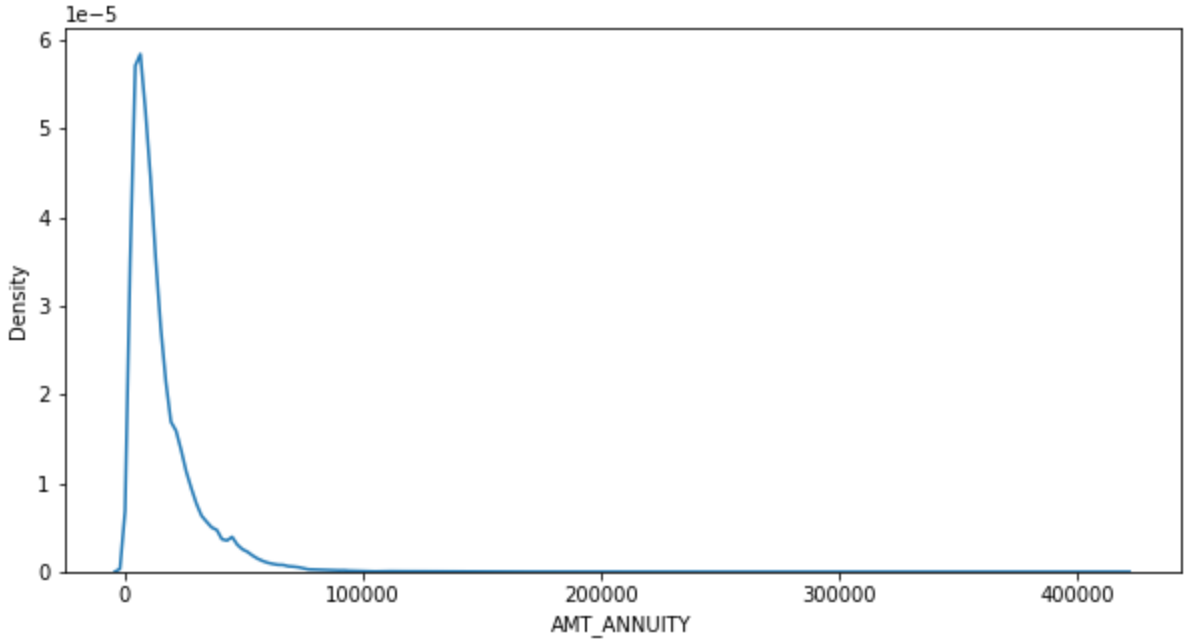
Starting to work on continuos variable and plotting to see the curve and impute the values accordingly

Imputing value for the above are :

1. If the distribution is positively skewed it is advised to impute Median values
2. Mode to be imputed, if the districution pattern is preserved
3. In case of negatively skewed distribution, it is advised to imput the mean values instead of missing values

Continuos variables are "AMT_ANNUITY", "AMT_GOODS_PRICE"

In [95]:
```
# Plotting distribution plot for the column "AMT_ANNUITY"

plt.figure(figsize=(10,5))
sns.kdeplot(pre_df['AMT_ANNUITY'])
plt.show()
```



Observation:

As the above plot is positively skewed, imputing median is advised as the median value gives the a more convincing substitution value

In [96]:
```
pre_df['AMT_ANNUITY'].head(20)
```

Out[96]:
```
0      1730.430
1     25188.615
2     15060.735
3     47041.335
4     31924.395
5     23703.930
6           NaN
7           NaN
8           NaN
9           NaN
10    11368.620
11    13832.775
12    12165.210
13     7654.860
14     9644.220
15    21307.455
16     4187.340
17     9000.000
18    10181.700
19     4666.500
Name: AMT_ANNUITY, dtype: float64
```

In [97]:
```
pre_df['AMT_ANNUITY'].describe()
```

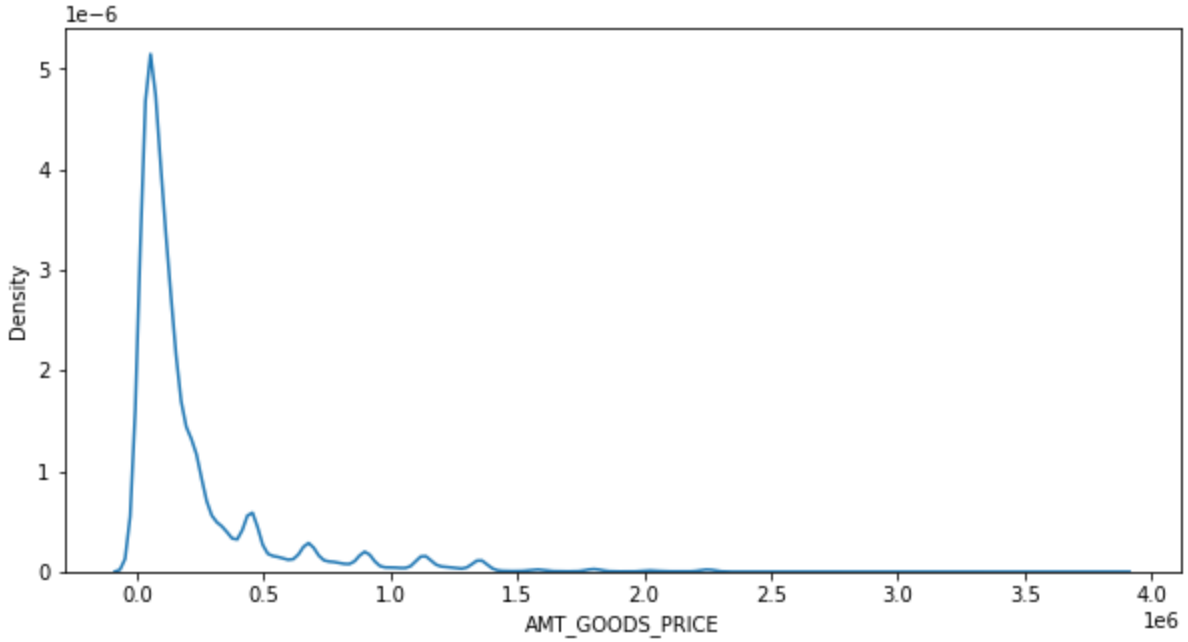
Out[97]:
```
count    138000.000000
mean      15514.995593
std       14508.783968
min           0.000000
25%        6164.403750
50%       10941.030000
75%       19795.770000
max      417927.645000
Name: AMT_ANNUITY, dtype: float64
```

In [98]:
```
pre_df['AMT_ANNUITY'].fillna(pre_df['AMT_ANNUITY'].median(), inplace = True)
```

In [99]:
```
pre_df["AMT_ANNUITY"].head(20)
```

Out[99]:
```
0      1730.430
1     25188.615
2     15060.735
3     47041.335
4     31924.395
5     23703.930
6     10941.030
7     10941.030
8     10941.030
9     10941.030
10    11368.620
11    13832.775
12    12165.210
13     7654.860
14     9644.220
15    21307.455
16     4187.340
17     9000.000
18    10181.700
19     4666.500
Name: AMT_ANNUITY, dtype: float64
```

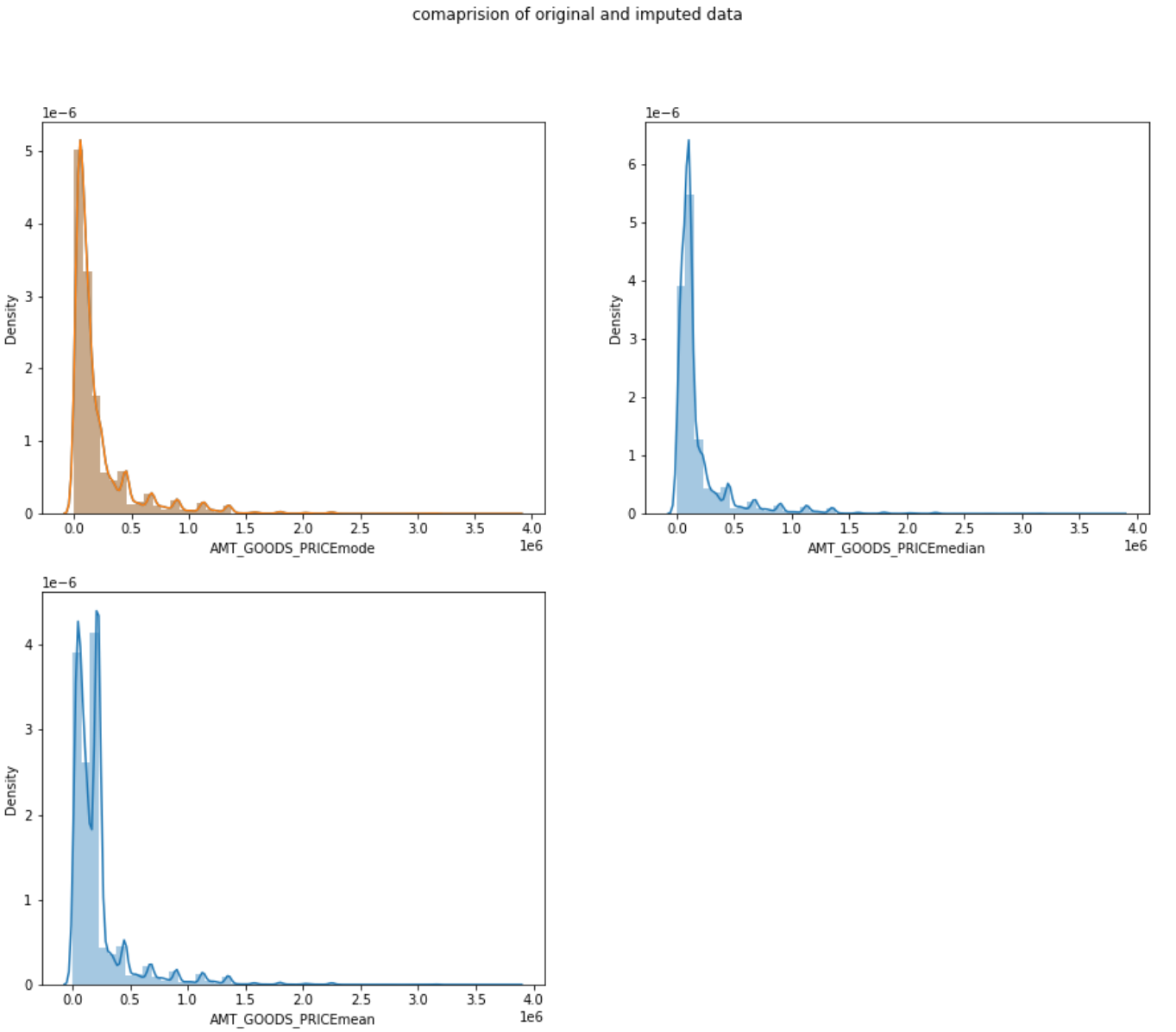
In [100]:
```
# Plotting distribution plot for the column "AMT_GOODS_PRICE"

plt.figure(figsize=(10,5))
sns.kdeplot(pre_df['AMT_GOODS_PRICE'])
plt.show()
```



Observation:

The above plot shows that the following variable have a number of peaks in its distribution. So, not sure what values to impute for better understanding we are imputing the mean, median and mode values and ploting the distribution again with those values and deciding on the imputing based on the plots

In [101]:
```
# plotting to find the best substitute value for the above distribution

statframe = pd.DataFrame()
statframe['AMT_GOODS_PRICEmode'] = pre_df['AMT_GOODS_PRICE'].fillna(pre_df['AMT
_GOODS_PRICE'].mode())
statframe['AMT_GOODS_PRICEmedian'] = pre_df['AMT_GOODS_PRICE'].fillna(pre_df['A
MT_GOODS_PRICE'].median())
statframe['AMT_GOODS_PRICEmean'] = pre_df['AMT_GOODS_PRICE'].fillna(pre_df['AMT
_GOODS_PRICE'].mean())

column = ['AMT_GOODS_PRICEmode', 'AMT_GOODS_PRICEmedian','AMT_GOODS_PRICEmean']

plt.figure(figsize=(15,12))
plt.suptitle('comaprision of original and imputed data')
plt.subplot(221)
sns.distplot(pre_df['AMT_GOODS_PRICE'][pd.notnull(pre_df['AMT_GOODS_PRICE'])]);
for i in enumerate(column):
    plt.subplot(2,2,i[0]+1)
    sns.distplot(statframe[i[1]])
```



Observation:

This plot, positively skewed, imputing mode is advised as the graph of the mode is very close the actual distribution and make a more convincing substitution value

In [102]:
```
# Imputing the null content with mode values

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

Finding the null values in the column "CNT_PAYMENT" and imputing thr mull values with '0'

In [103]:
```
pre_df.loc[pre_df['CNT_PAYMENT'].isnull(),'NAME_CONTRACT_STATUS'].value_counts()
```

Out[103]:
```
Canceled        30444
Refused          4260
Unused offer     2779
Approved            1
Name: NAME_CONTRACT_STATUS, dtype: int64
```

Obsevation:

NAME_CONTRACT_STATUS include loans that are not started under its column information, for a proper analysis the following values must be removed from the column data so using the imputed column values to find the actual numbers

In [104]:
```
pre_df.columns
```

Out[104]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_GOODS_PRICE',
       'NAME_CASH_LOAN_PURPOSE', 'NAME_CONTRACT_STATUS', 'DAYS_DECISION',
       'NAME_PAYMENT_TYPE', 'CODE_REJECT_REASON', 'NAME_TYPE_SUITE',
       'NAME_CLIENT_TYPE', 'NAME_GOODS_CATEGORY', 'NAME_PORTFOLIO',
       'NAME_PRODUCT_TYPE', 'CHANNEL_TYPE', 'SELLERPLACE_AREA',
       'NAME_SELLER_INDUSTRY', 'CNT_PAYMENT', 'NAME_YIELD_GROUP',
       'PRODUCT_COMBINATION', 'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE',
       'DAYS_LAST_DUE_1ST_VERSION', 'DAYS_LAST_DUE', 'DAYS_TERMINATION',
       'NFLAG_INSURED_ON_APPROVAL', 'YEARLY_DECISION'],
      dtype='object')
```

In [105]:
```
# Converting required columns data type to the necessary type (i.e., Object to Categorical)

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

# Initiating a loop statement to convert the data type
for col in pre_appl_catgorical:
    pre_df[col] =pd.Categorical(pre_df[col])
```

Finding outliers in the second dataset

In [106]:
```
pre_df.describe()
```

Out[106]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE
count	1.754830e+05	175483.000000	175483.000000	1.754830e+05	1.754830e+05	1.370260e+05
mean	1.919434e+06	278661.624830	14538.000942	1.695229e+05	1.892379e+05	2.171305e+05
std	5.344200e+05	102834.352184	13002.121826	2.835496e+05	3.096062e+05	3.043896e+05
min	1.000001e+06	100006.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00
25%	1.455628e+06	189721.500000	7252.965000	2.111850e+04	2.546100e+04	4.941000e+04
50%	1.918764e+06	278907.000000	10941.030000	7.101000e+04	7.902450e+04	1.062495e+05
75%	2.382861e+06	368138.000000	16327.980000	1.800000e+05	2.015820e+05	2.250000e+05
max	2.845377e+06	456254.000000	417927.645000	3.826372e+06	4.104351e+06	3.826372e+06

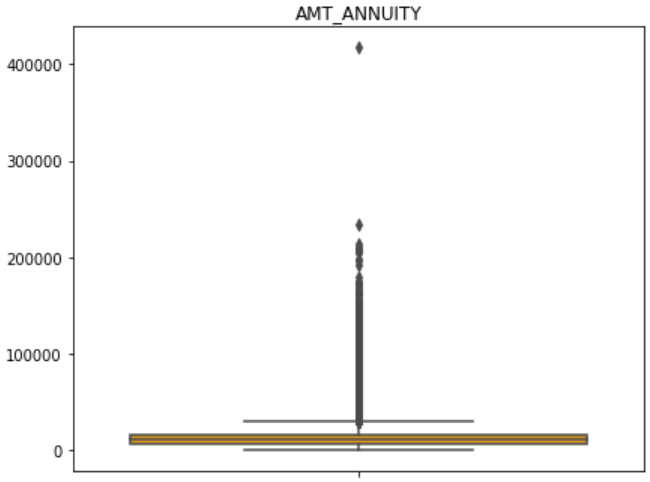
Observation:

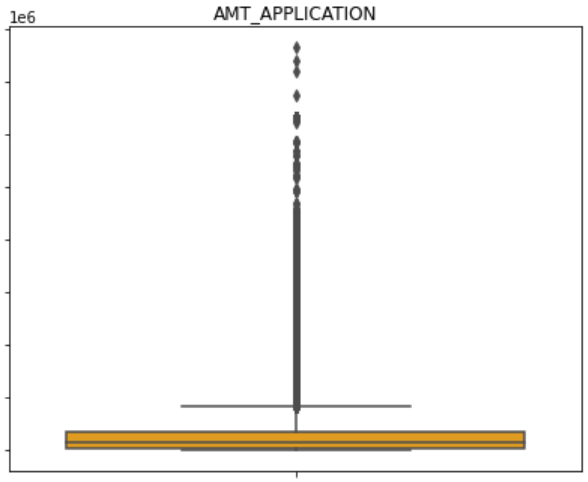
Capturing the the name of columns that are showing high difference and values, which doesn't show much relationship with the other values are captured

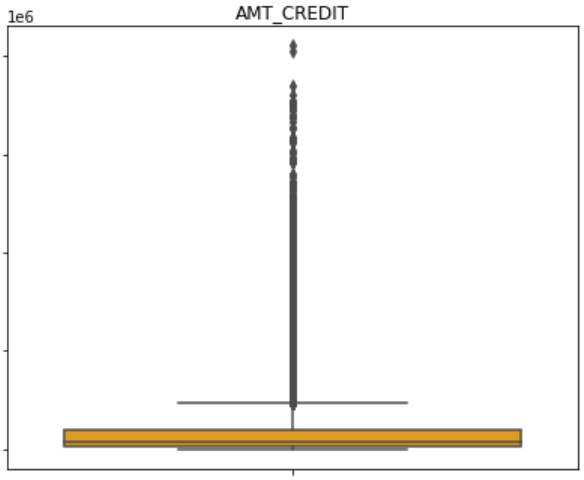
In [107]:
```
pre_appl_outliers = ['AMT_ANNUITY','AMT_APPLICATION','AMT_CREDIT','AMT_GOODS_PRICE',
                     'SELLERPLACE_AREA','DAYS_DECISION','CNT_PAYMENT']
```

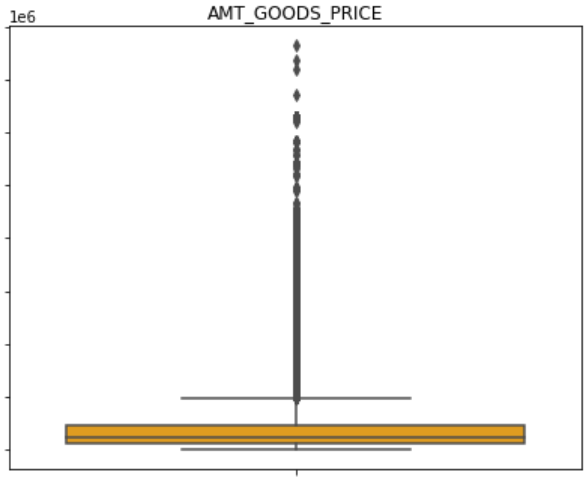
In [108]:

```
plt.figure(figsize=[15,25])
for i,j in itertools.zip_longest (pre_appl_outliers, range(len(pre_appl_outlier
s))):
    plt.subplot(4,2,j+1)
    sns.boxplot(y = pre_df[i], orient = "h", color = "orange")
    plt.xlabel("")
    plt.ylabel("")
    plt.title(i)
```

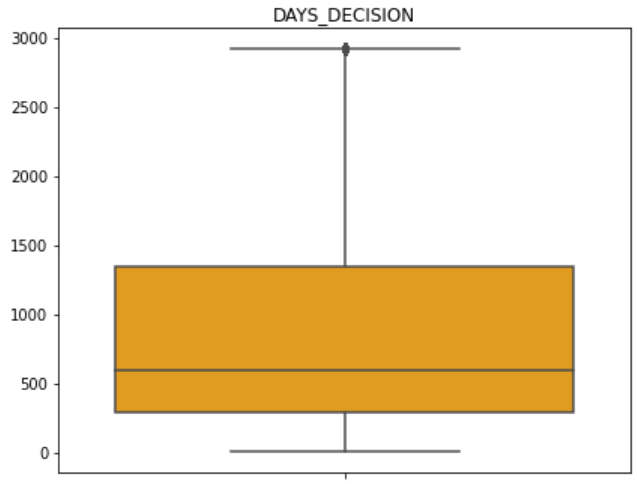

AMT_ANNUITY
400000
300000
200000
100000
0

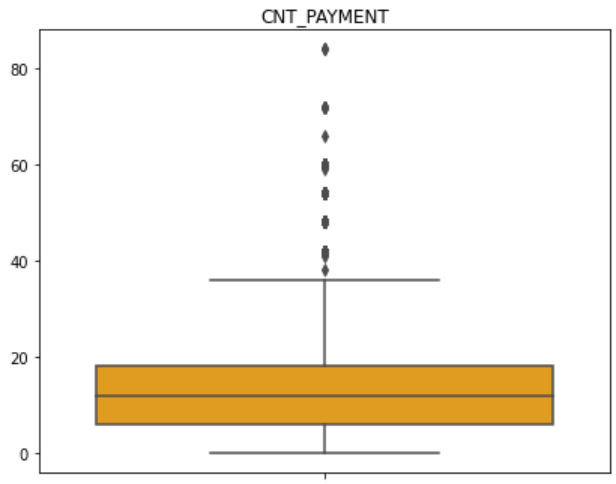

AMT_APPLICATION
1e6
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0


AMT_CREDIT
1e6
4
3
2
1
0


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


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


DAYS_DECISION
3000
2500
2000
1500
1000
500
0


CNT_PAYMENT
80
60
40
20
0

Observations:

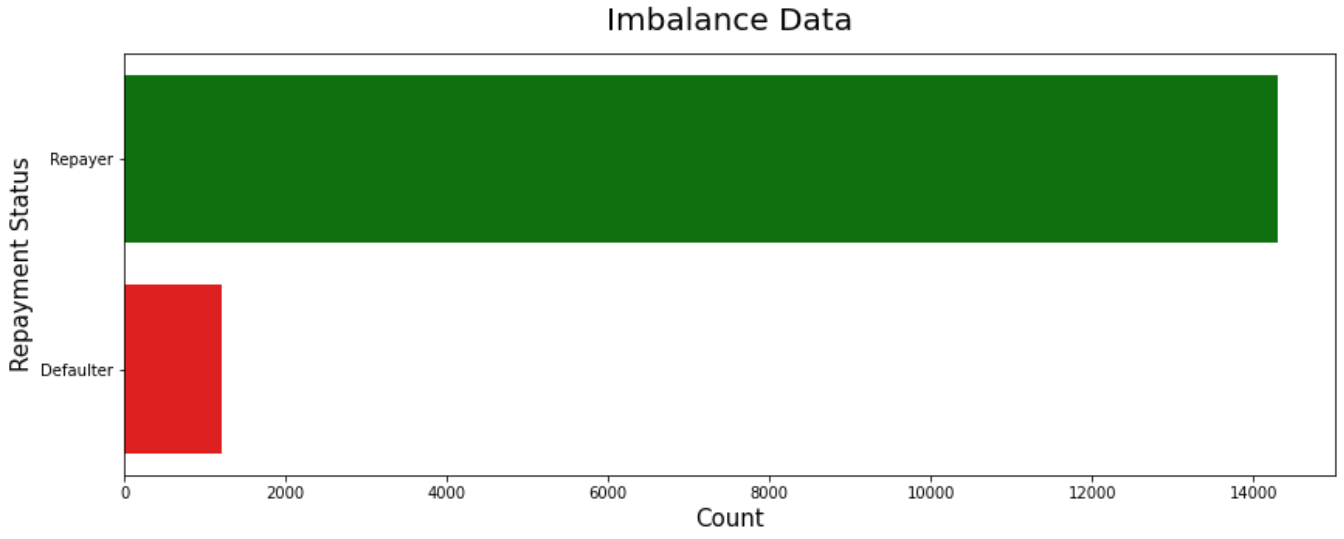
1. The columns "SELLERPLACE_AREA", "AMT_ANNUITY", "AMT_CREDIT", "AMT_GOODS_PRICE", "AMT_APPLICATION" exibits the highest number of outliers
2. The column CNT_Payment has the least number of outliers in the values recoeded
3.The column DAYS_DECISION has few outlier values, but the entries such as 2900 or 1900 days are really to long duration for an bank to make decision which doesn't make much sense. Must be that the values are recorded wrongly

Data Analysis Part

Important Function for Univariate analysis

Creating a function for plotting Variables to do univariate analysis. This function will create two plots
1. Plot of the given column in relation to the TARGET column
2. Finding the percentage of defaulters within each column of the given datasets

In [109]:
```
plt.figure(figsize= [14,5])
sns.barplot(y=["Repayer","Defaulter"], x = bank_df["TARGET"].value_counts(), palette = ["green","r"],orient="h")
plt.ylabel("Repayment Status",fontdict = {"fontsize":15})
plt.xlabel("Count",fontdict = {"fontsize":15})
plt.title("Imbalance Data", fontdict = {"fontsize":20}, pad = 15)
plt.show()
```



In [110]:

```
# defining function decide on the type of data

def data_frame(dataset,column):
    if dataset[column].dtype == np.int64 or dataset[column].dtype == np.float6
4:
        return "numerical"
    if dataset[column].dtype == "category":
        return "categorical"

# Creating a function for univariate analysis (Analysis over single variable)

def uni_var(dataset,column,target_column,y_log_axis=False,x_axis=False,horzonta
l_layout=True):
    if data_frame(dataset,column) == "numerical":
        sns.distplot(dataset[column],hist=False)


    elif data_frame(dataset,column) == "categorical":
        count = dataset[column].value_counts()
        df = pd.DataFrame({column: count.index,'count': count.values})


        target_p = dataset[[column, target_column]].groupby([column],as_index=F
alse).mean()
        target_p[target_column] = target_p[target_column]*100
        target_p.sort_values(by=target_column,inplace = True)

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

        if(horzontal_layout):
            fig, (axis1, axis2) = plt.subplots(ncols=2, figsize=(15,7))
        else:
            fig, (axis1, axis2) = plt.subplots(nrows=2, figsize=(25,35))

# 1. Subplot 1:  Ploting column data to give a picture of the repayer and defau
lter

        s = sns.countplot(ax=axis1, x=column, data=dataset, hue=target_column)
        axis1.set_title(column +" Col Data", fontsize = 15)
        axis1.legend(['Repayer','Defaulter'])
        axis1.set_xlabel(column,fontdict={'fontsize' : 12, 'fontweight' : 4})

        if(x_axis):
            s.set_xticklabels(s.get_xticklabels(),rotation=100)

# 2. Subplot 2: plotting column wise percentage dat to find the defaulters in e
ach data columns

        s = sns.barplot(ax=axis2, x = column, y=target_column, data=target_p)
        axis2.set_title("Percentage of Defaulters in "+col, fontsize = 15)
        axis2.set_xlabel(column,fontdict={'fontsize' : 12, 'fontweight' : 4})
        axis2.set_ylabel(target_column,fontdict={'fontsize' : 12, 'fontweight'
: 4})

        if(x_axis):
            s.set_xticklabels(s.get_xticklabels(),rotation=100)


        plt.show()
```

In [111]:
```
# defining bivariate function for repetitive plotting of numeric data

def bi_var_numerical(x,y,df,hue,kind,labels):
    plt.figure(figsize=[20,20])
    sns.relplot(x=x, y=y, data=df, hue=hue,kind=kind,legend = False)
    plt.legend(labels=labels)
    plt.xticks(rotation=100, ha='left')
    plt.show()
```

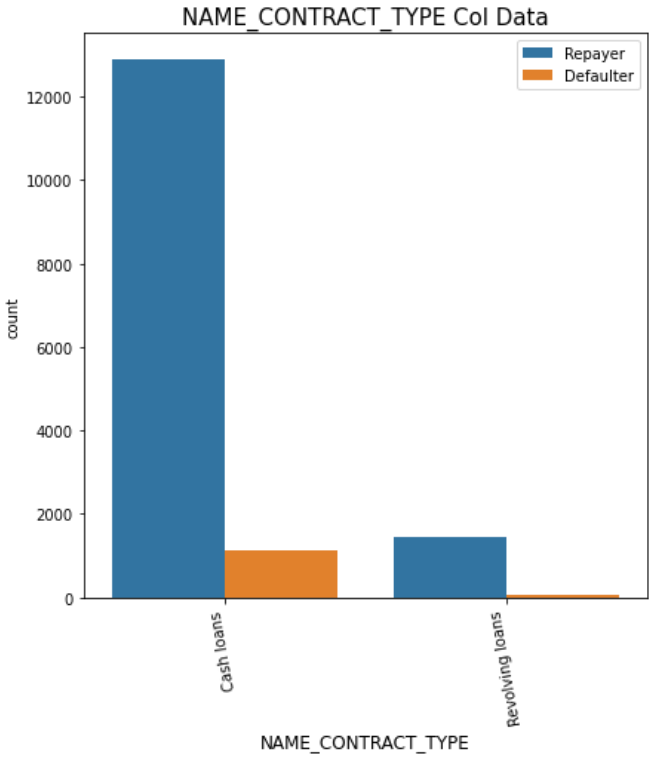
In [112]:
```
#defining function to plot repetitive countplots for categorical bivariate data

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

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

    # Determining aesthetics
    plt.xlabel(x,fontsize = 10)
    plt.ylabel(y,fontsize = 10)
    plt.title(col,fontsize = 25)
    plt.xticks(rotation=100, ha='left')
    plt.legend(labels = labels )
    plt.show()
```

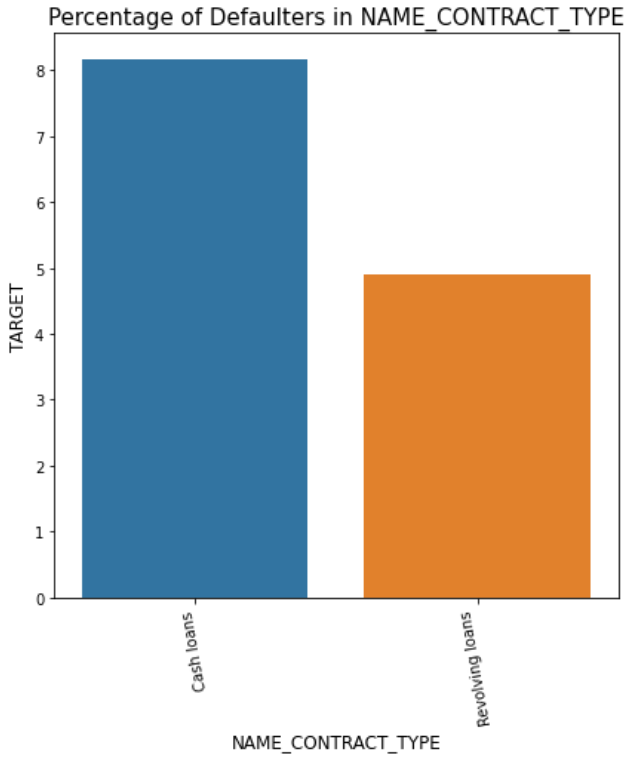
In [113]:
```
# defining function to plot repetitive countplots for categorical univariate da
ta

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


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

    plt.title(col , fontsize=20)
    plt.legend(loc = "upper right")
    plt.xticks(rotation=45, ha='right')

    plt.show()
```

In [114]:
```
# Saving  numercial and categorical data

cat_col = list(bank_df.select_dtypes(["category"]).columns)   # Categorical col
umns list
num_col = list(bank_df.select_dtypes(["int","float"]).columns)  #N Numerical Co
lumn list
```

Categorical Variables Analysis

In [115]:
```
#1 Impact of contract type with the bank on loan replayment
uni_var(bank_df,"NAME_CONTRACT_TYPE","TARGET",False,True,True)
```



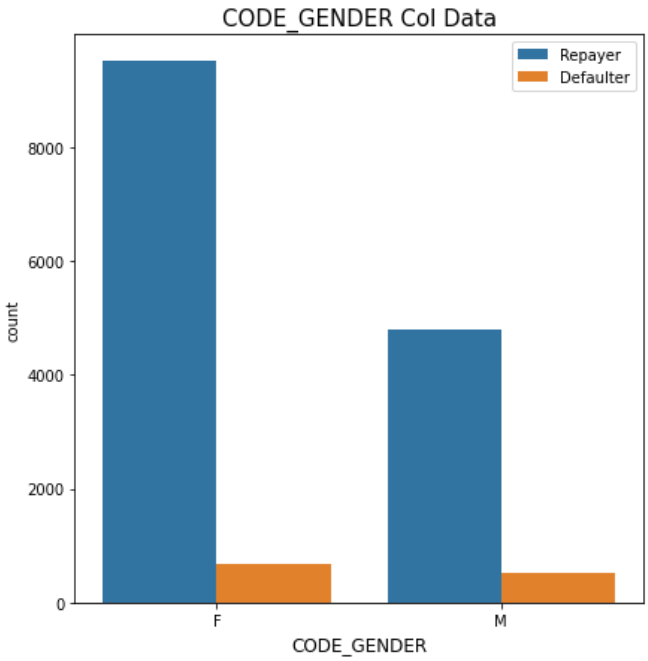


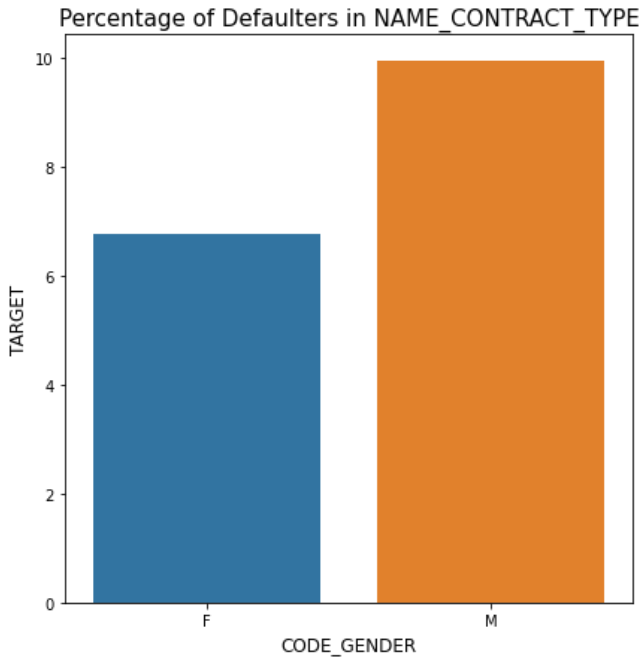
Observation on type of loan:

1. Around 90% of the loans are cash loans, which indicate that the loan applicant are in the need of liquid cash.
2. Only 10% of the loans are revolving loans.
3. Solid 8 % of the cash loan recipients turn out to be defaulters. Where as in the c ase of revolving loans, 5.5 % of the loan ecipients turn out to be defaulters.

It is clearly seen that most of the cash loan and revolving loan recipients repay the ir loans. But in terms of sheer numbers the defaulters are more in the case of cash l oans.

In [116]:
```
#2 Impact of Gender with loan replayment
uni_var(bank_df,"CODE_GENDER","TARGET",True,False,True)
```



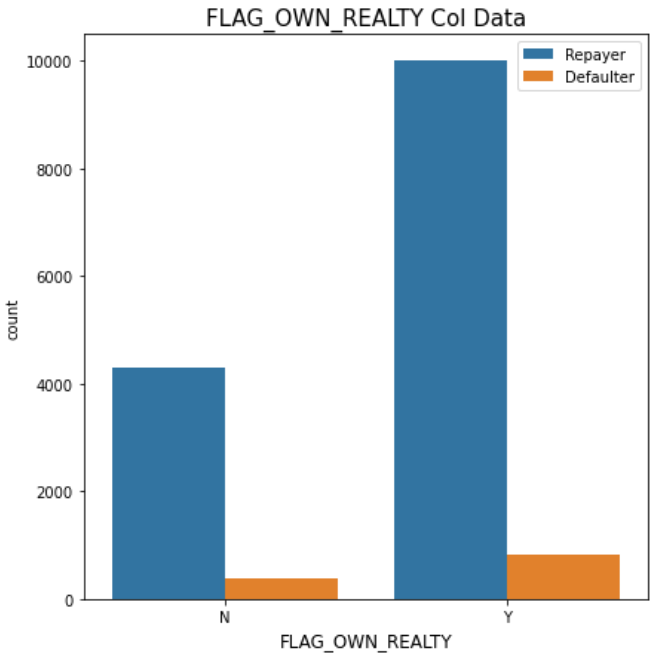


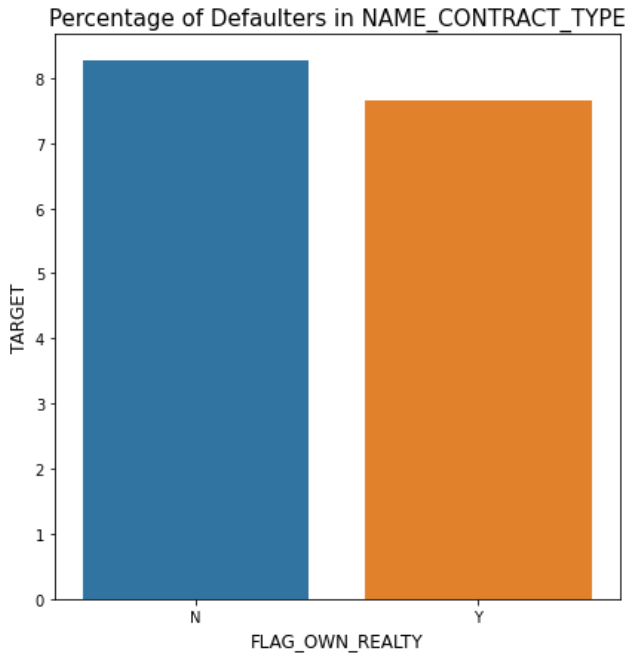
Observation on type of Gender:

1. From the above plot it is clear that most of the loan applicants are female.
2 But, when it comes to the defaulter percentage, it can seen that male clients tend to not return their loans. 10% of the male clients turn out to be defaulters but in case of the female clients, the defaulter percentage is as low as 7%.

This clearly indicates that most the female clients are more trust worthy clients to the bank and this inturn would impact the decisions matrix. Its more likely that female clients would have their loan approvals faster and the overall process would be seemingly smooth.

In [117]:
```
#3 Impact of property holdings on loan replayment
uni_var(bank_df,"FLAG_OWN_REALTY","TARGET",False,False,True)
```

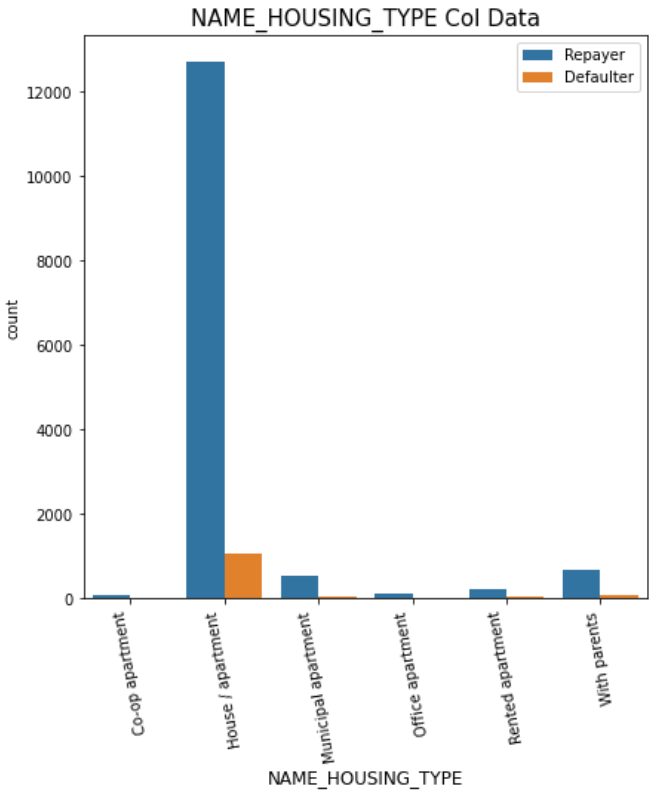


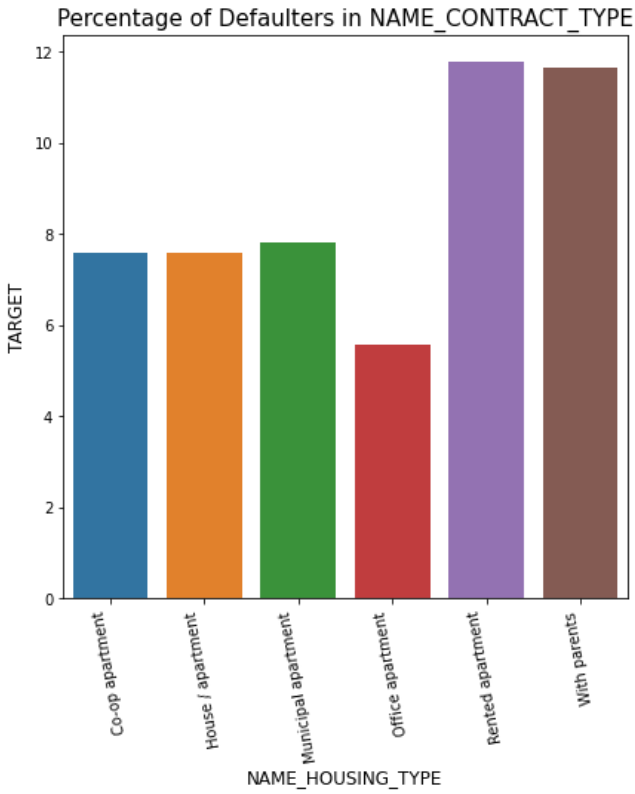


Observation based on applicants real estate status:

1. From the above plot it is clear that most of the loan applicants have own property of their own. Which is more than double the amount of applicatns without any propert y.
2. However, in terms of the defaulter percent, approximately 8% of the applicants in both parties default. But the applicants without property slightly default morethan t he other one.

In [118]:
```
 #4 Impact of housing on loan replayment
uni_var(bank_df,"NAME_HOUSING_TYPE","TARGET",True,True,True)
```

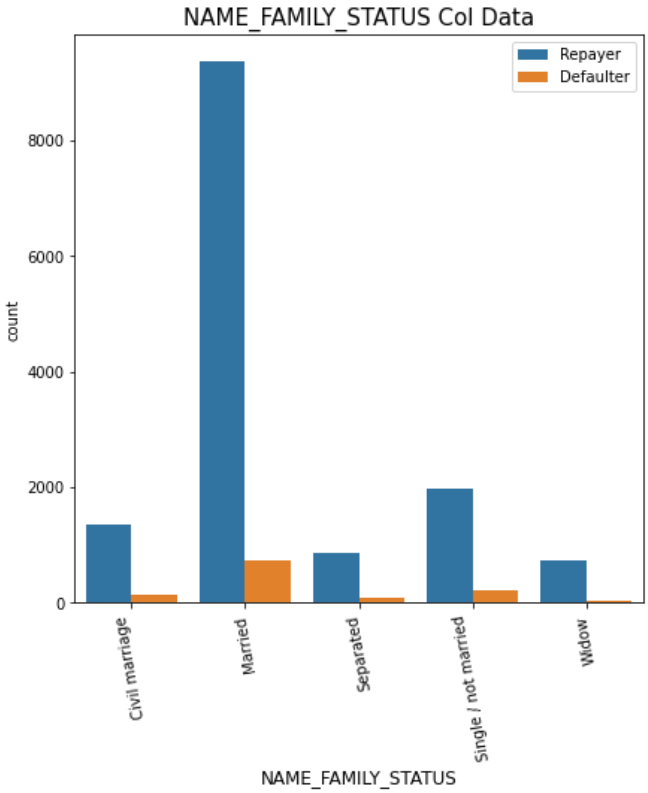


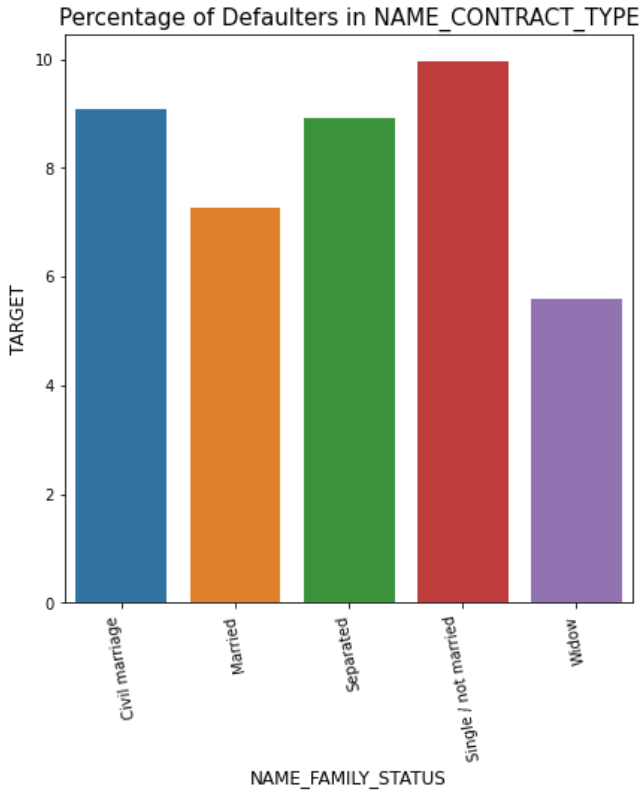


Obersation based on applicants housing

1. Most of the loan applicants live in independent house or apartment.
2. People living in co-op apartment don't tend to apply for loans in general.
3. Coming to the defaulter percentage, people living in office appartments tend to default the least. only 6.5 % of the loan applicants default in their case.
4. Majority of the defaultsers fall under the rented appartment category (12% applicants) and applicants living with their parents (around 11%).
5. But in terms of sheer numbers, majority of defaulters fall under the house and apartment category, which can be clearly seen with the help of the plot above.

In [119]:
```
#5 Impact of marital status on loan replayment
uni_var(bank_df,"NAME_FAMILY_STATUS","TARGET",True,True,True)
```

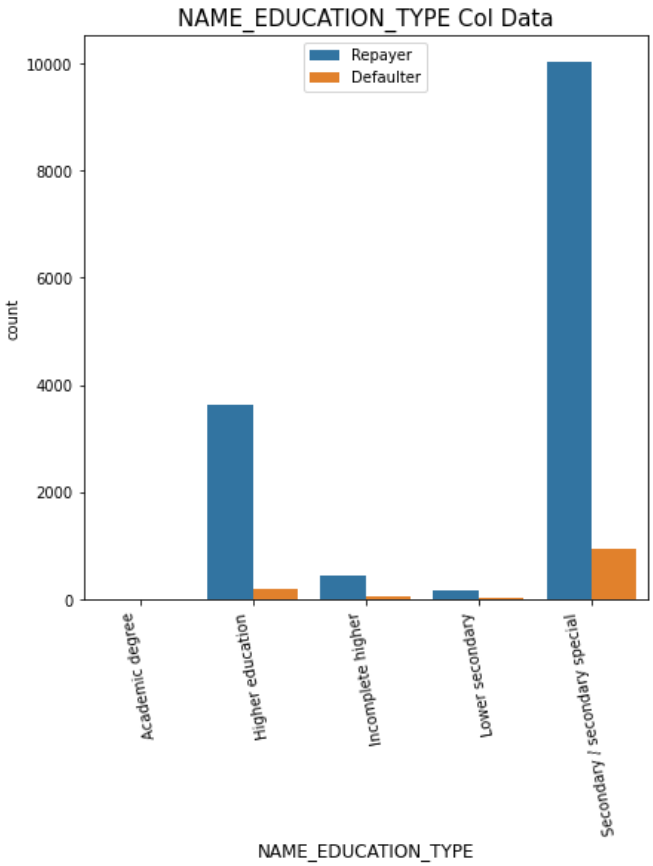


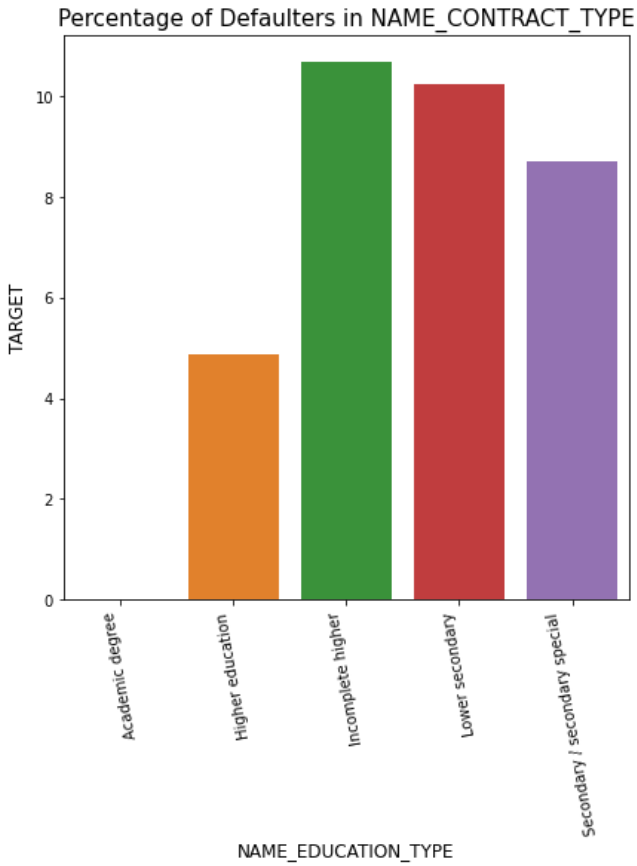


Observation based on marital status:

1. Majority of the loan applicants are married couples, follwed by single and civil marriage couples.
2. Most of the civil marriage couple along with single applicants tend to default more, which is around 10%.
3. However, widow applicants are least defaulters, that is with excluding the unknown category, as the data collected is very less so it can be excluded.

In [120]:
```
#6 Impact of Educational qualification on loan replayment
uni_var(bank_df,"NAME_EDUCATION_TYPE","TARGET",True,True,True)
```

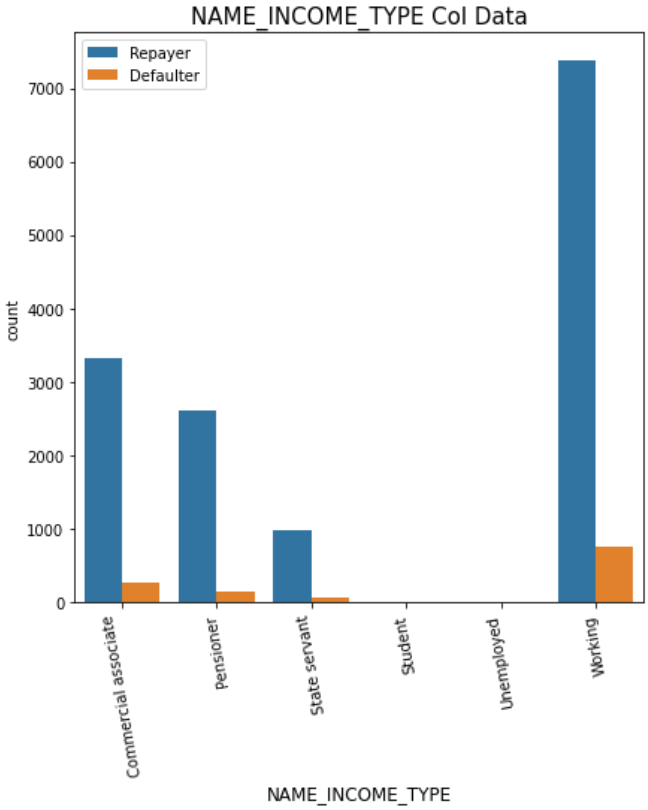


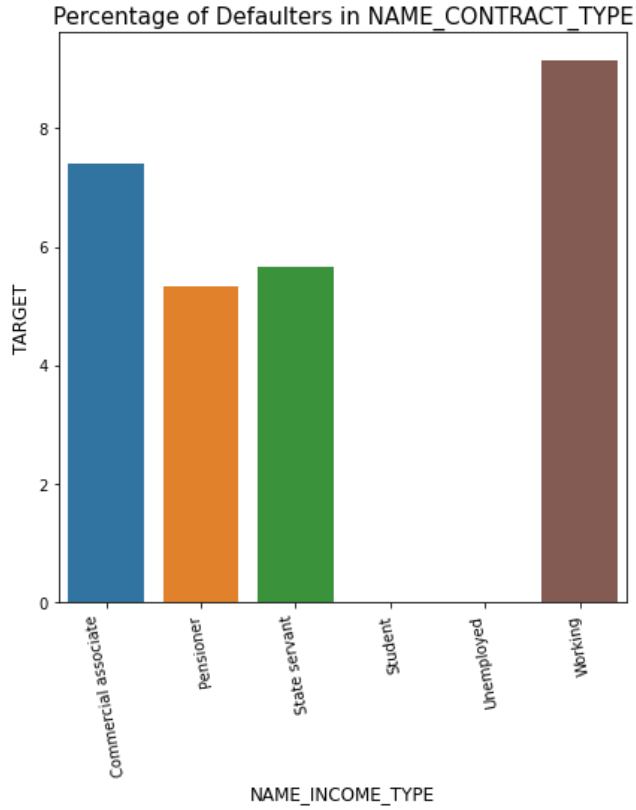


Observations based on Educational qualification:

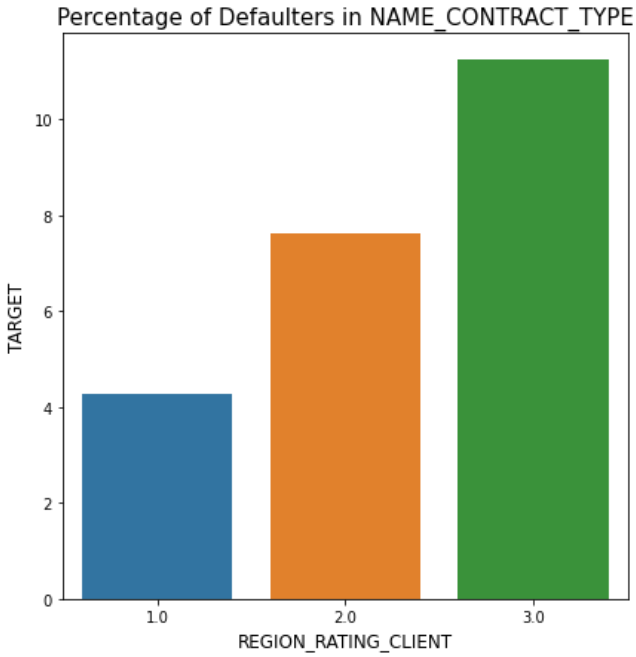
1. People with secondary educational qualification are higher education are the mojar applicants for loan.
2. But when it comes to defaulter percentage, applicants with lower secondary degree are more likely to default which is around 12%.
3. Majority of the higher education applicants tend to return the loan.

In [121]:
```
#7 Impact of income type on loan replayment
uni_var(bank_df,"NAME_INCOME_TYPE","TARGET",True,True,True)
```





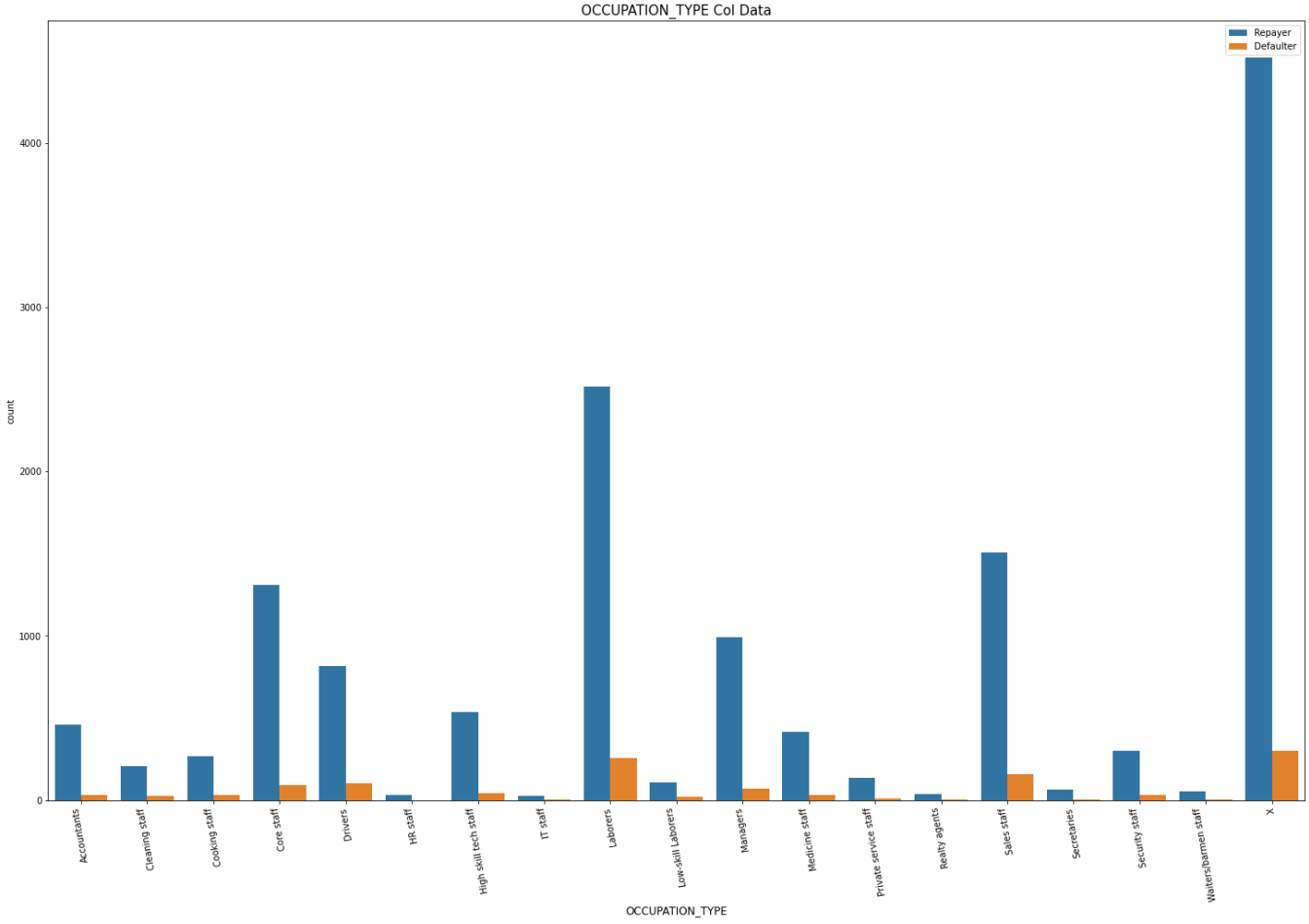
In [122]:
```
#8 Relationship of geographical loaction with loan replayment
uni_var(bank_df,"REGION_RATING_CLIENT","TARGET",True,False,True)
```

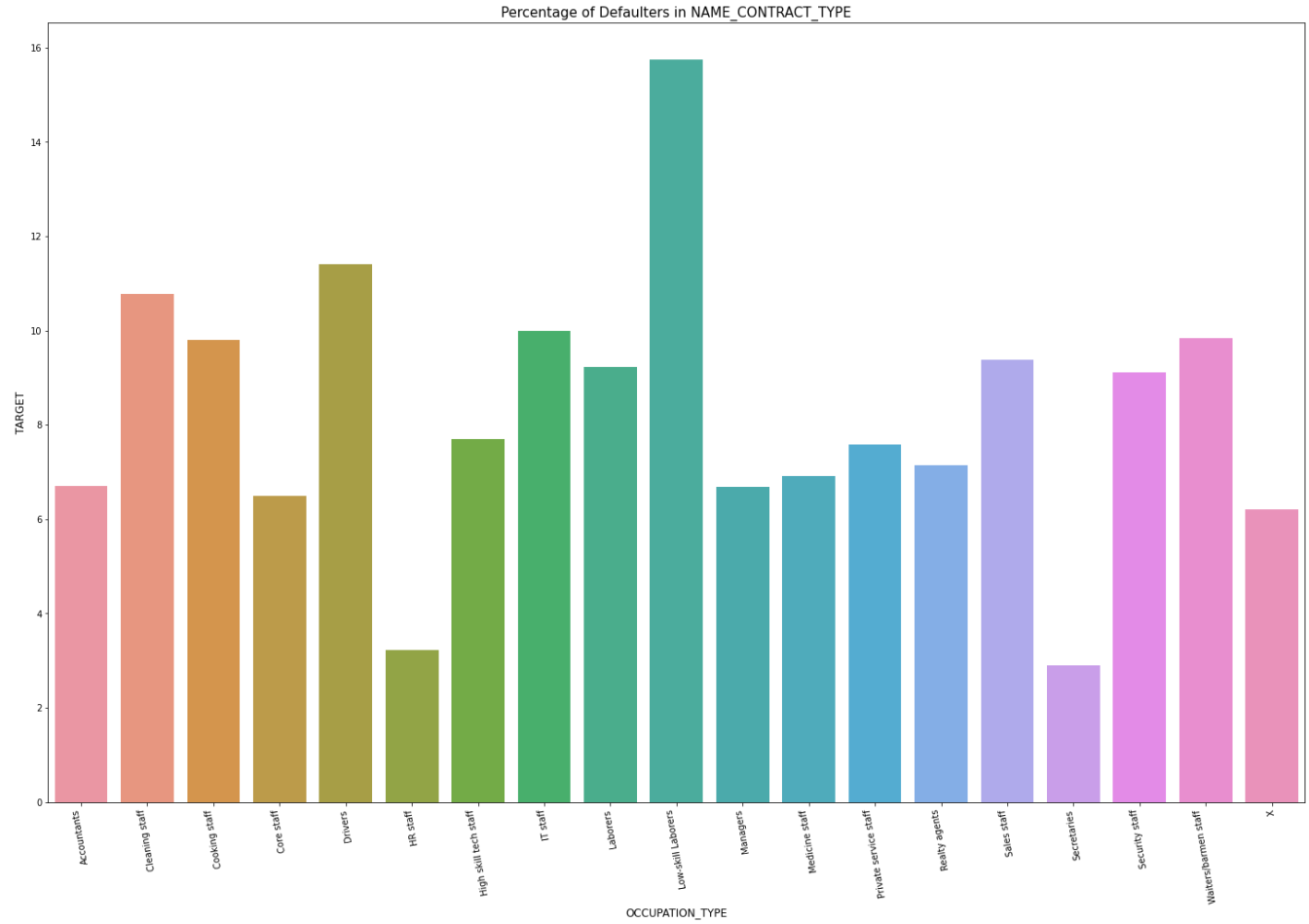


Observations based on client location:

1. Most of the loan applicants are staying in locations with Rating 2.
2. the highest number of defaulter are from the regions with Rating 3 (Defaulting per cent = ~12%).
3. Applicant living in locations with Rating 1 have low probability of defaulting. So, these applicants loan approval is much safer.

In [123]:
```
#9 Impact of employment role on loan replayment
uni_var(bank_df,"OCCUPATION_TYPE","TARGET",True,True,False)
```

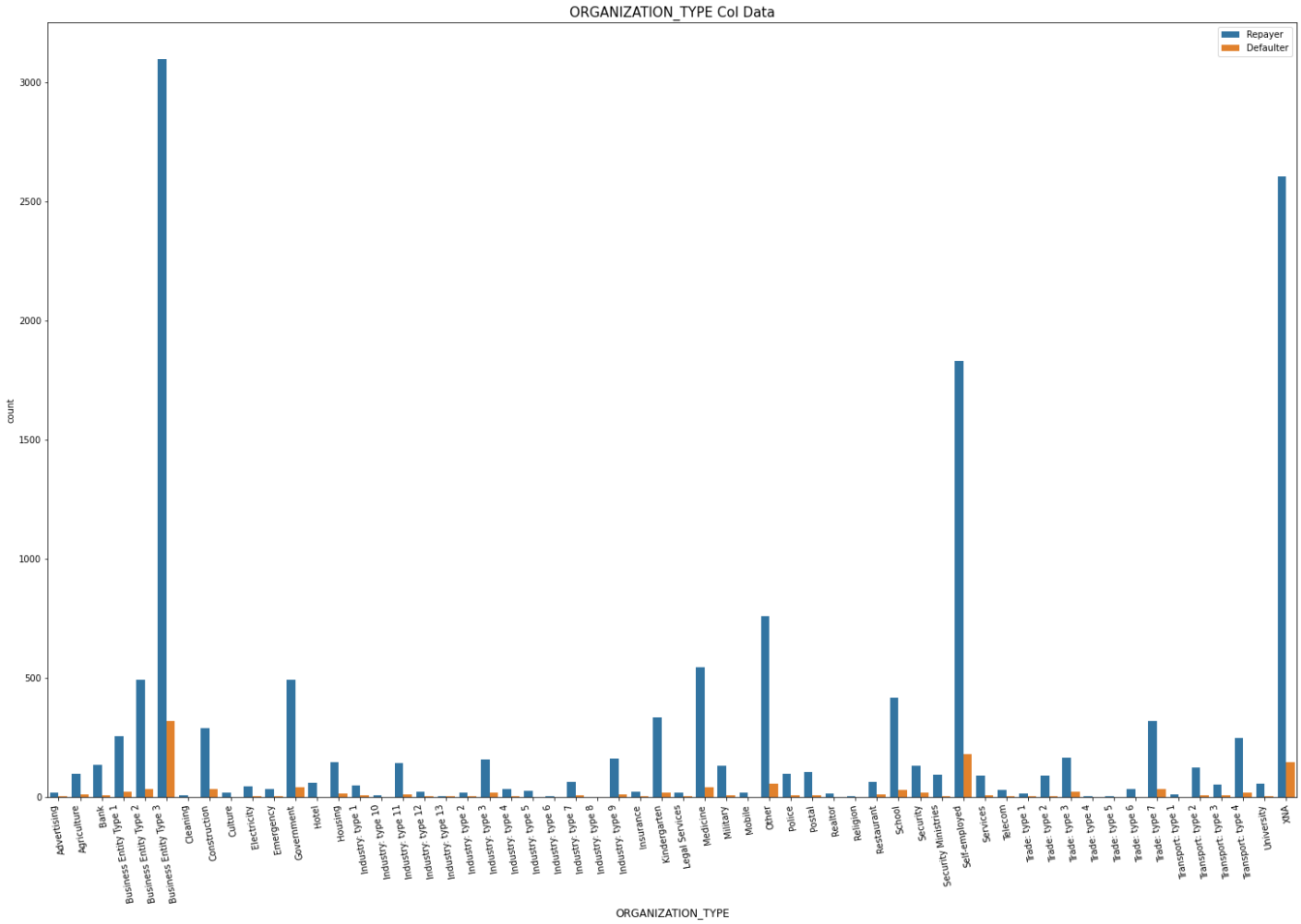


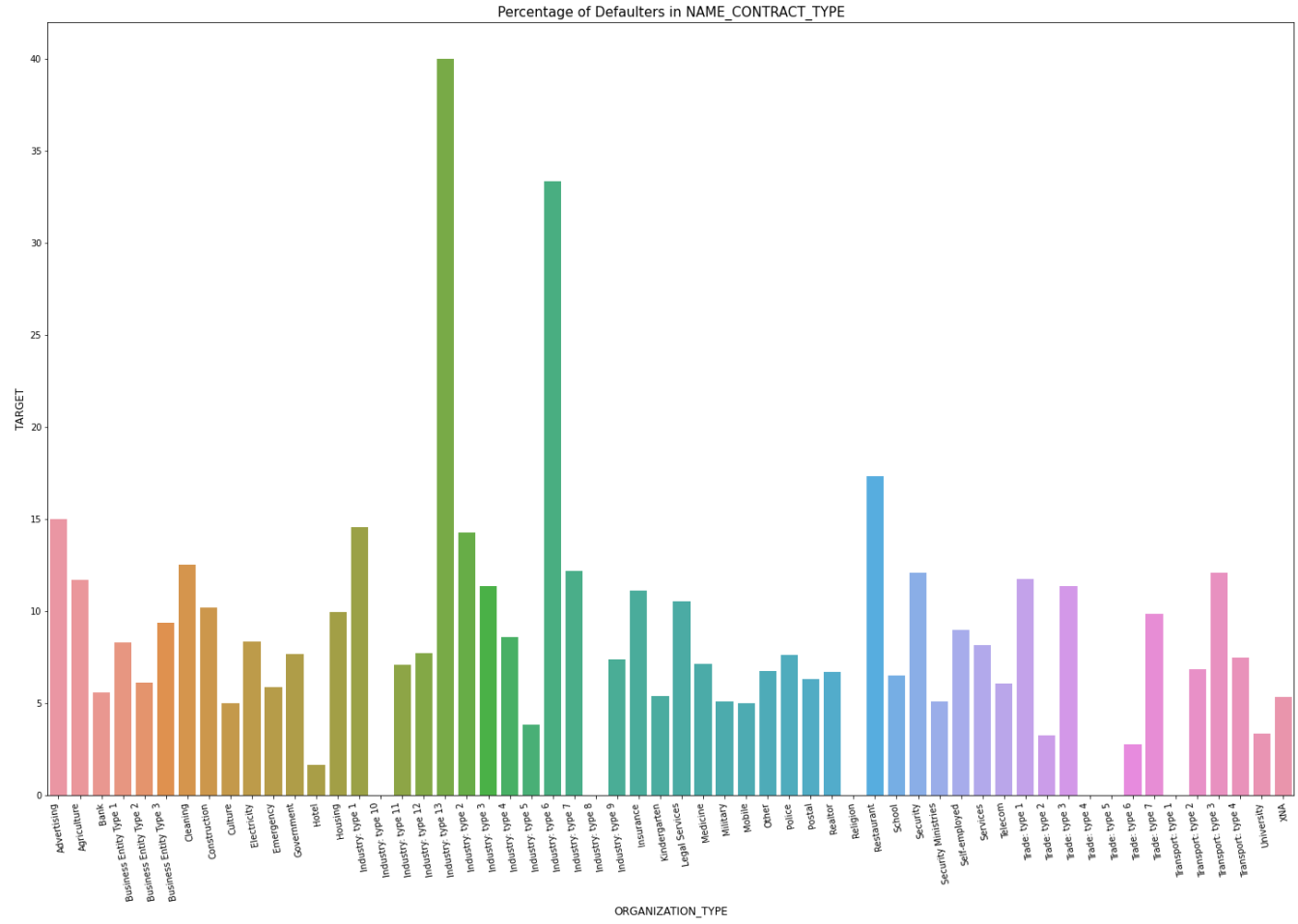


Observations based on Occupation type:

1. Most of the loans are taken by Laborers, followed by Sales staff.
2. IT staff, HR staff and reality agents are less likely to apply for Loan.
3. Category with highest percent of defautess are Low-skill Laborers (above 17%), followed by Drivers and Waiters/barmen staff, Security staff, Laborers and Cooking staff
4. Accountant are least defaulters, only 5% of the applicants tend to default from their loans. Which makes them a good candidate for loan applications.

In [124]:
```
#10 Impact of industry category on loan replayment

uni_var(bank_df,"ORGANIZATION_TYPE","TARGET",True,True,False)
```

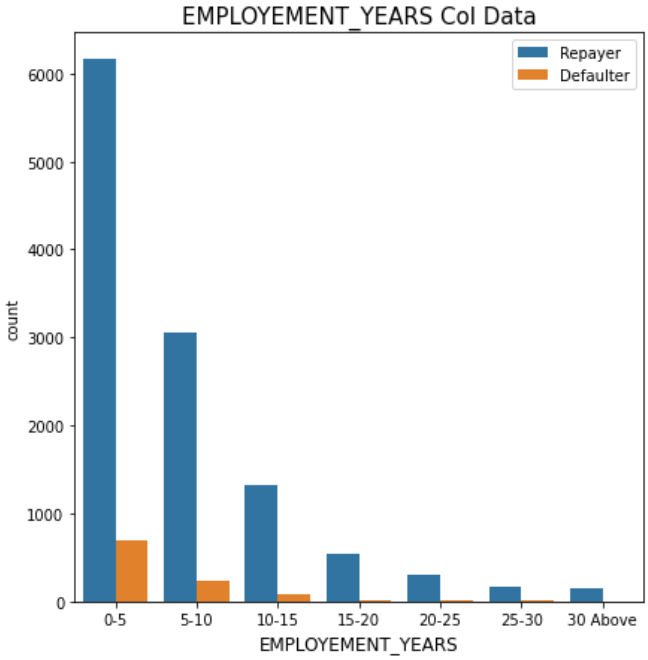


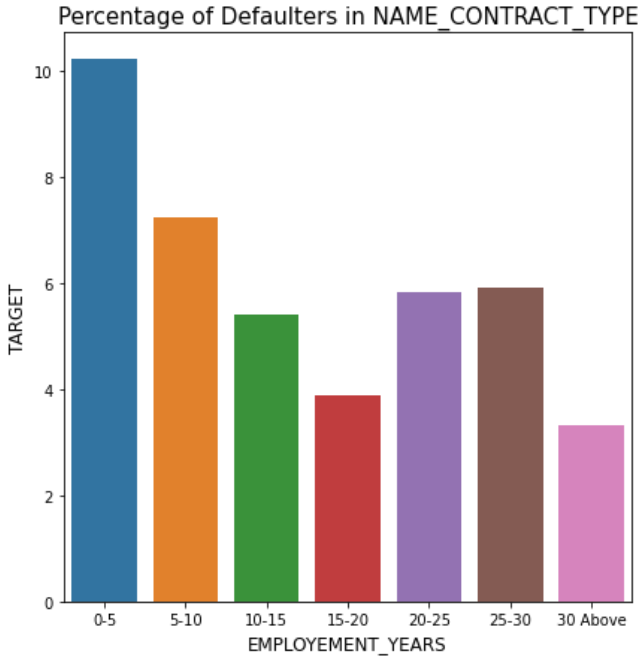


Observations based on organization type:

1. The highest number of defaulter are from the following organisation type:
    1. 16% from Transport: type 3
    2. 13.5% from Industry: type 13 (13.5%)
    3. 12.5% from Industry: type 8 (12.5%)
    4. ~12% from Restaurant and construction feilds
2. Majority of the loan application are from Business Entity Type 3, Unknown and Self-employed people.
3. For a very high number of applications the Organization information is Unknown or missing (XNA). However, the defaulter percent is also low for it.
4. For the following organisation, it is observed that the defaulter percent is less and thus make them a potentially good candidate for loan approval.
    1. Trade Type 4
    2. Trade Type 5
    3. Industry type 8

In [125]:
```
#11 Impact of employment years on loan replayment
uni_var(bank_df,"EMPLOYEMENT_YEARS","TARGET",True,False,True)
```

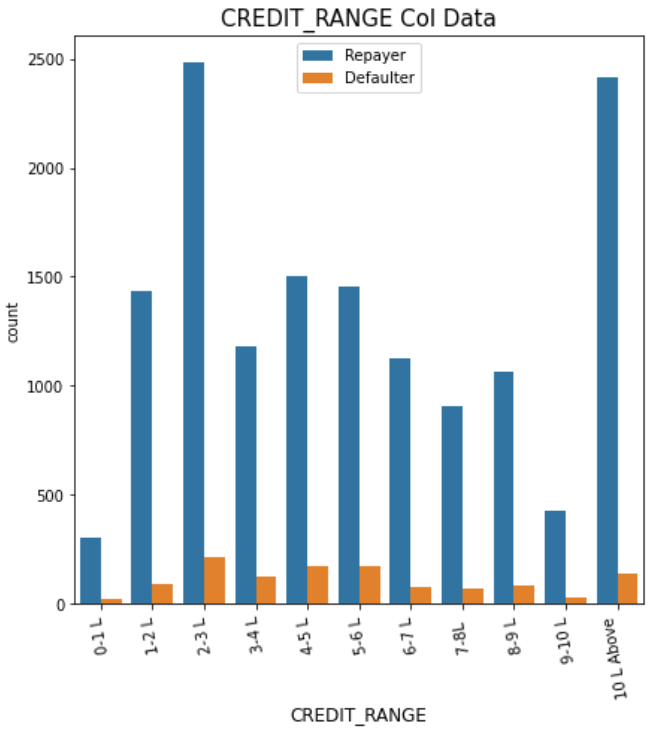


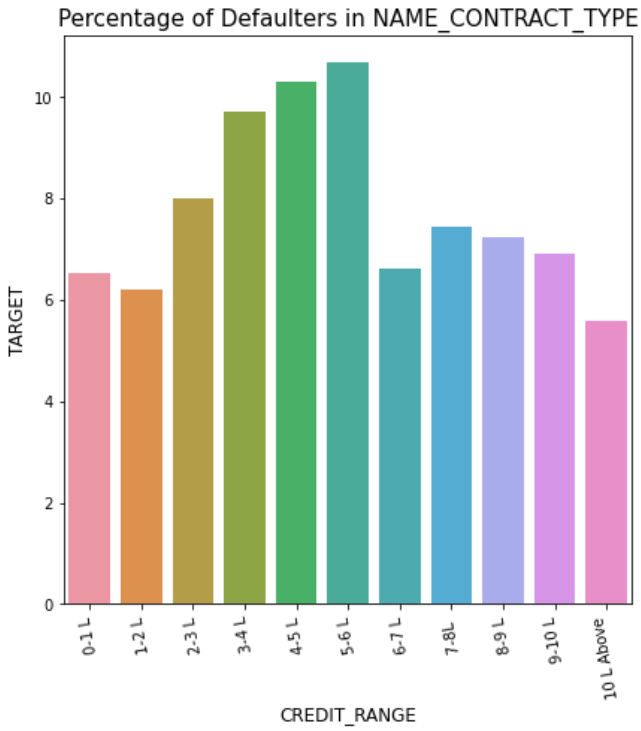


Obsevation based on the employment years:

1. Applicants having working experience between 0-5 years are the major defaulters (Around 11%). Followed by applicants with a work expierence of 5-10 years which is around 7.5 %.
2. people with 25-30 and 30 above years of expierence tend to default less. Thus a steady decreasein defaulter percent is noticed with an increase in work expierence.

In [126]:
```
#12 Impact of loan amount on loan replayment

uni_var(bank_df,"CREDIT_RANGE","TARGET",False,True,True)
```

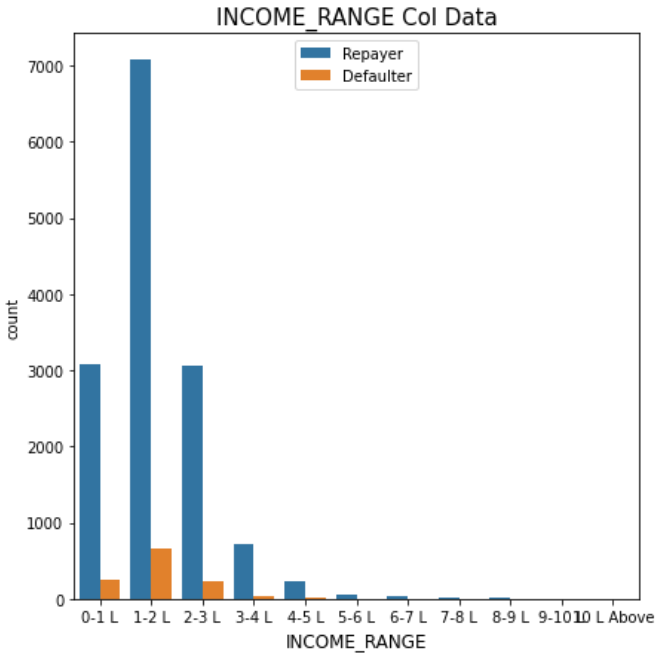


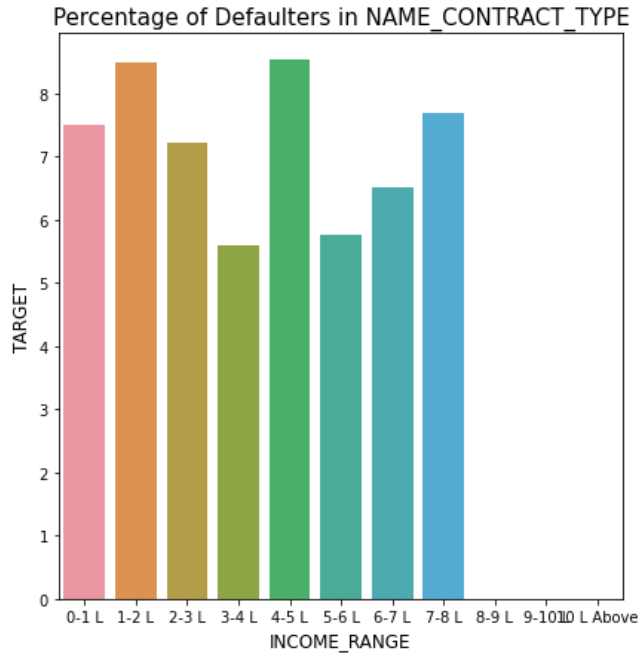


Observation with respect to loan amount:

1. Majority of the loan applicants have their salary range between 2-3 Lakhs (50000+) and then people with saraly above 10 Lakh range (45000+)
2. People who get loan for 3-6 Lakhs have most number of defaulters than other loan range.

In [127]:
```
#13  Impact of income range with loan replayment

uni_var(bank_df,"INCOME_RANGE","TARGET",False,False,True)
```

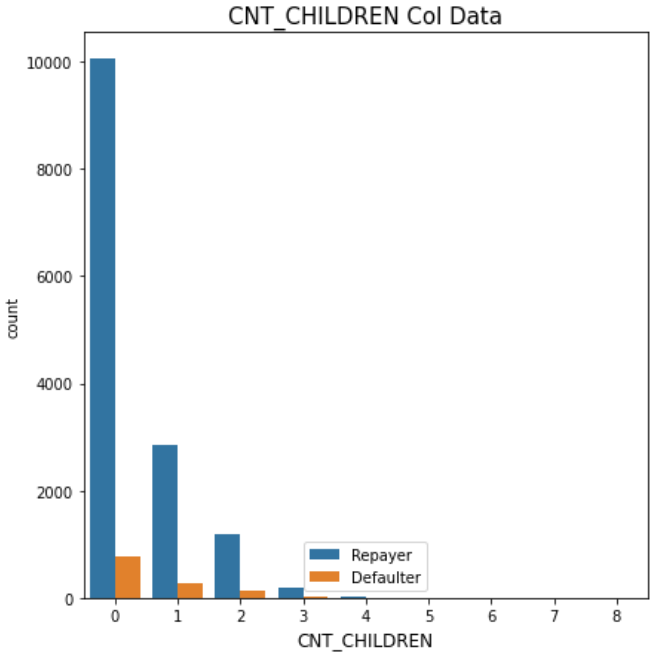


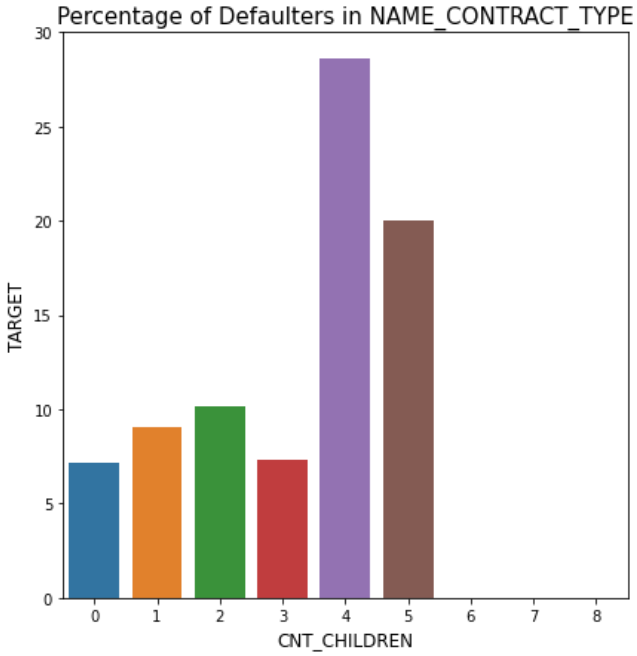


Observation based on Applicant Income

1. Applicants with income range between 1-2 lakh are the major applicants for loan.
2. applicants with income range between 0-3 lakh, tend to default often.
3. Applicant with Income 7-8 Lakhs are least defaulters (~2%).

In [128]:
```
#14  Impact of number of childrens on loan replayment

uni_var(bank_df,"CNT_CHILDREN","TARGET",True,False,True)
```

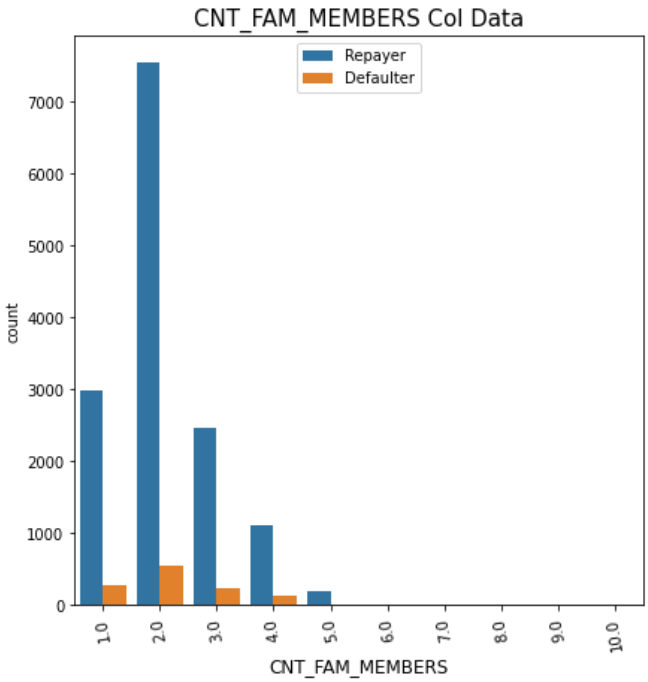


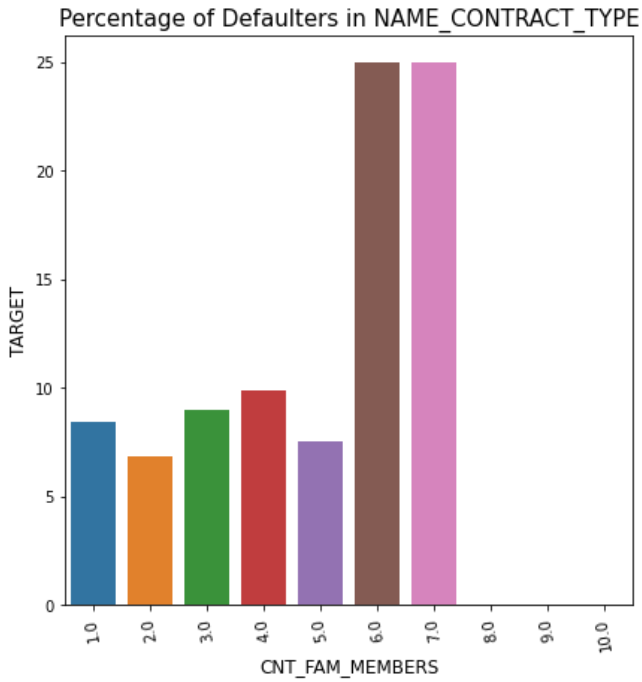


Observations based on children count:

1. Majority of the loan applicants do not have childrens or have a 1-2 childrens.
2. Clients with more than 6 children have very high default rate and clients with 9-1
1 children showed 100% default rate. So it is not advised to approve loan for this ca
tegory of applicants

In [129]:
```
#15 family size relatinship with loan replayment
uni_var(bank_df,"CNT_FAM_MEMBERS","TARGET",True,True,True)
```





Observation based on family members

1. Majority of the loan applicants are either single or a family of 2.
2. However, it can be noticed that applicatant with a family size of 8-10 have very high defaulting rate. But applicants with more than 11-13 family members have close to 100 % defaulting rate. So these set of candidates are not a good prospect for the bank loans.

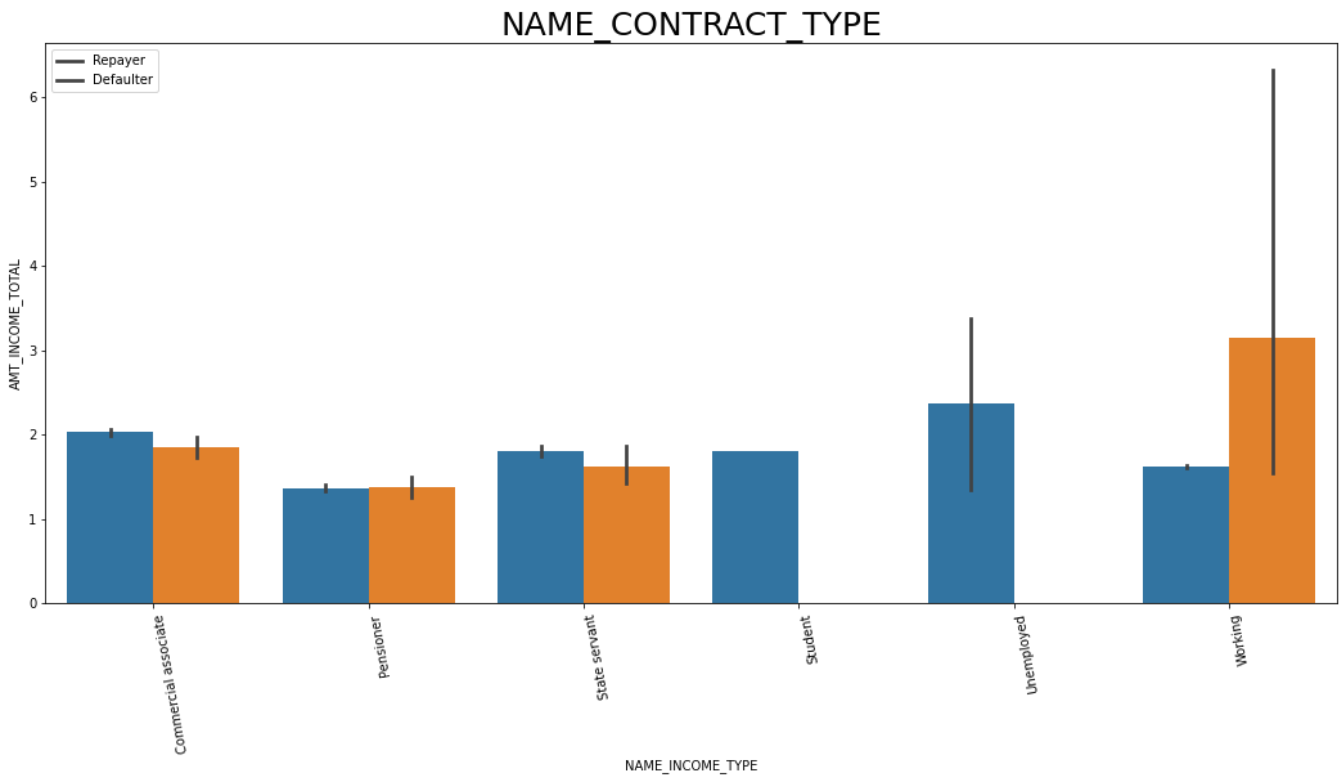
Categorical Bivariate vs Multivariate

In [130]:
```
bank_df.groupby('NAME_INCOME_TYPE')['AMT_INCOME_TOTAL'].describe()
```

Out[130]:

NAME_INCOME_TYPE	count	mean	std	min	25%	50%	75%	max
Commercial associate	3593.0	2.017590	1.154213	0.3600	1.35000	1.8000	2.25000	13.500
Pensioner	2752.0	1.367871	0.828602	0.2700	0.90000	1.1250	1.71000	19.350
State servant	1042.0	1.794275	0.914540	0.2700	1.12500	1.5750	2.25000	9.000
Student	1.0	1.800000	NaN	1.8000	1.80000	1.8000	1.80000	1.800
Unemployed	2.0	2.362500	1.431891	1.3500	1.85625	2.3625	2.86875	3.375
Working	8133.0	1.762523	12.984494	0.2565	1.12500	1.3500	2.02500	1170.000

In [131]:
```
# Analysing relationship between income type and income total
bi_var_categorical("NAME_INCOME_TYPE","AMT_INCOME_TOTAL",bank_df,"TARGET",(18,8),['Repayer','Defaulter'])
```



Obseravtion:

Businessman category seems to have the highest income when compared with the rest of the profession, also along with students they also have the least defaulting percent. Which ideally make them a very good candidate for bank oans and approval of loans.

Numeric Variables Analysis

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

In [132]:
```
#Listing columnns of dataframe "bnk_appl"
bank_df.columns
```

Out[132]:
```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT',
       'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE',
       'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE',
       'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED',
       'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'FLAG_MOBIL', 'OCCUPATION_TYPE',
       'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT',
       'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_START',
       'HOUR_APPR_PROCESS_START', 'REG_REGION_NOT_LIVE_REGION',
       'REG_REGION_NOT_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION',
       'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY',
       'LIVE_CITY_NOT_WORK_CITY', 'ORGANIZATION_TYPE',
       'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE',
       'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE',
       'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_3',
       'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY',
       'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON',
       'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR',
       'INCOME_RANGE', 'CREDIT_RANGE', 'GOODS_PRICE_RANGE', 'AGE', 'AGE_GROUP',
       'YEARS_EMPLOYED', 'EMPLOYEMENT_YEARS'],
      dtype='object')
```

In [133]:
```
# dividing the bnk_appl dataframe based on Target data, inorder to perform corr
elation analysis

cols_for_correlation = ['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_REALTY',
                        'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_
ANNUITY', 'AMT_GOODS_PRICE',
                        'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_
TYPE', 'DAYS_ID_PUBLISH', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS', 'REGION_RATING_
CLIENT',
                        'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_ST
ART', 'HOUR_APPR_PROCESS_START',
                        'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGI
ON', 'LIVE_REGION_NOT_WORK_REGION',
                        'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY', 'LI
VE_CITY_NOT_WORK_CITY', 'ORGANIZATION_TYPE',
                        'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE',
'NAME_FAMILY_STATUS',
                        'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE', 'DAY
S_BIRTH', 'DAYS_EMPLOYED',
                        'DAYS_REGISTRATION', 'DAYS_LAST_PHONE_CHANGE', 'FLAG_DO
CUMENT_3',
                        'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DA
Y', 'AMT_REQ_CREDIT_BUREAU_WEEK',
                        'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QR
T', 'AMT_REQ_CREDIT_BUREAU_YEAR']


# Repayers dataframe
Repayer_df = bank_df.loc[bank_df['TARGET']==0, cols_for_correlation]

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

In [134]:
```
len(cols_for_correlation)
```

Out[134]: 41

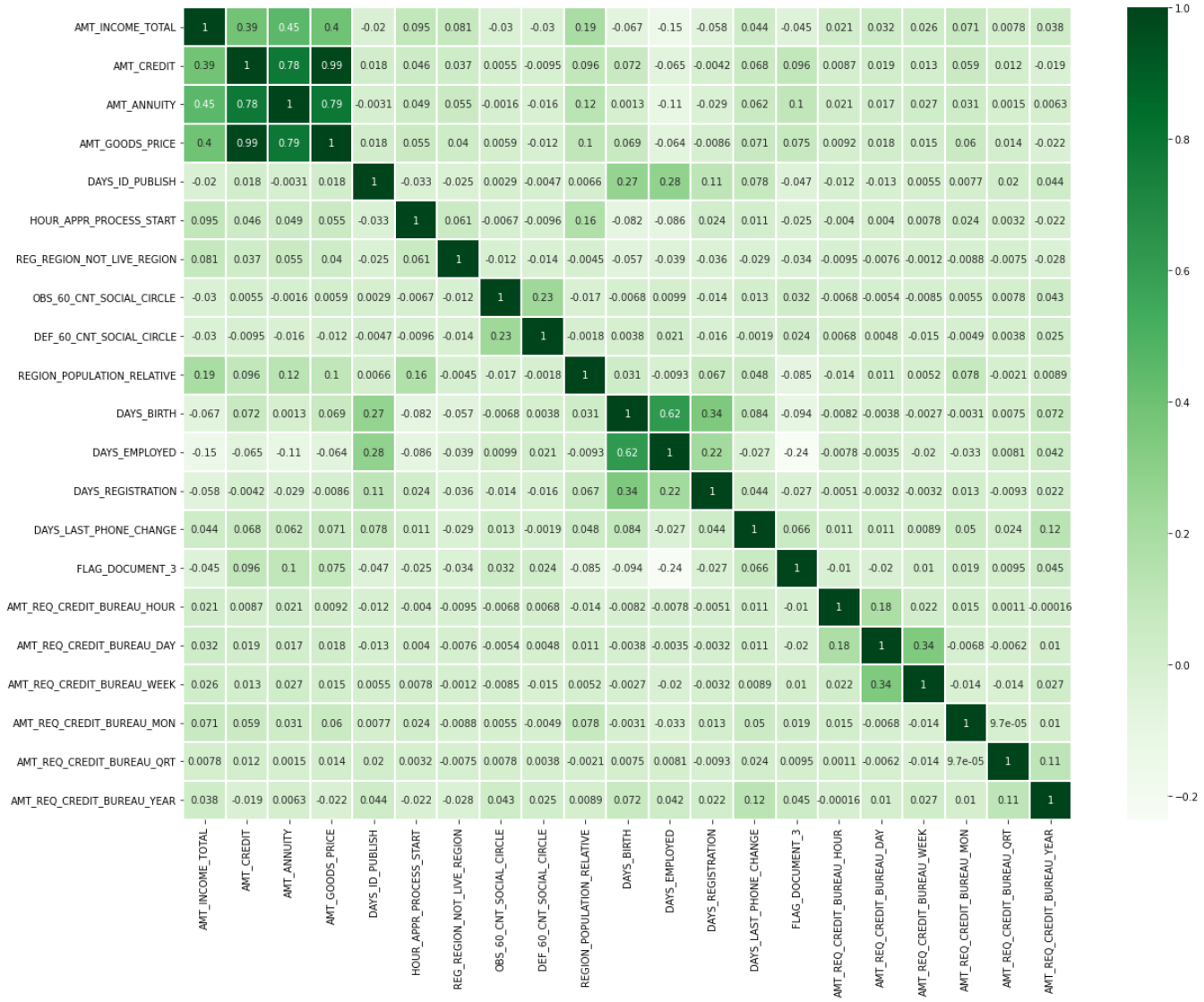
Correlation between numeric variable

In [135]:
```
# Top 10 correlation of Repayers data

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

Out[135]:

	VAR1	VAR2	Correlation
64	AMT_GOODS_PRICE	AMT_CREDIT	0.987045
65	AMT_GOODS_PRICE	AMT_ANNUITY	0.786573
43	AMT_ANNUITY	AMT_CREDIT	0.782472
241	DAYS_EMPLOYED	DAYS_BIRTH	0.620993
42	AMT_ANNUITY	AMT_INCOME_TOTAL	0.454994
63	AMT_GOODS_PRICE	AMT_INCOME_TOTAL	0.395727
21	AMT_CREDIT	AMT_INCOME_TOTAL	0.389681
262	DAYS_REGISTRATION	DAYS_BIRTH	0.344100
373	AMT_REQ_CREDIT_BUREAU_WEEK	AMT_REQ_CREDIT_BUREAU_DAY	0.341603
235	DAYS_EMPLOYED	DAYS_ID_PUBLISH	0.276893

In [136]:
```
fig = plt.figure(figsize=(20,15))
ax = sns.heatmap(Repayer_df.corr(), cmap="Greens",annot=True,linewidth =1)
```



Observation: Correlating factors amongst repayers

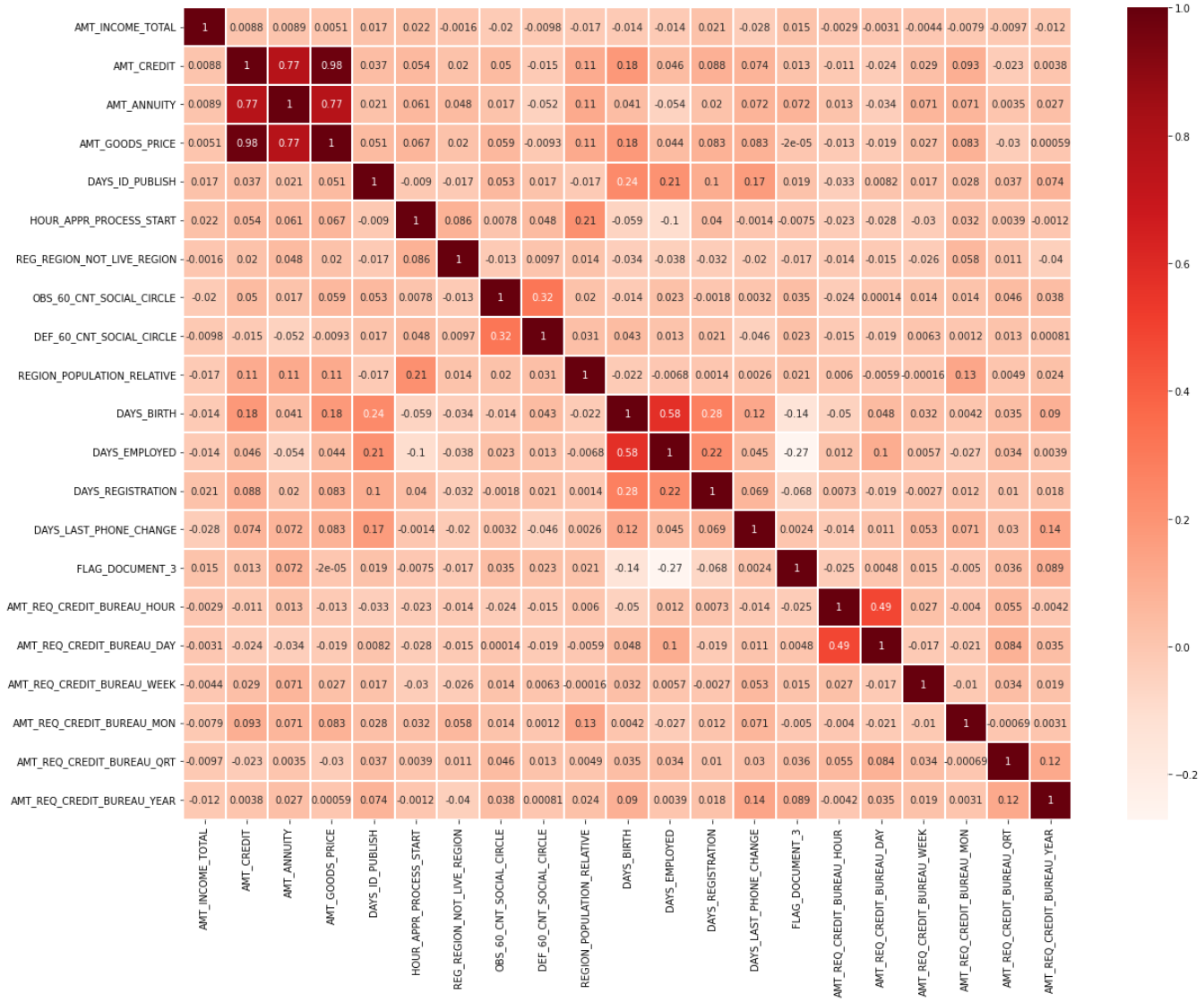
1. Credit amount is highly correlated with "Goods Price Amount", "Loan Annuity", "Total Income"
2. We can also see that repayers have high correlation in number of days employed.

In [137]:
```
# Top 10 correlation of Defaulter data
corr_Defaulter = Defaulter_df.corr()
Defaulter_corr = corr_Defaulter.where(np.triu(np.ones(corr_Defaulter.shape),k=
1).astype(np.bool))
Defaulter_corr = corr_Defaulter.unstack().reset_index()
Defaulter_corr.columns =['VAR1','VAR2','Correlation']
Defaulter_corr.dropna(subset = ["Correlation"], inplace = True)
Defaulter_corr["Correlation"]=Defaulter_corr["Correlation"].abs()
Defaulter_corr.sort_values(by='Correlation', ascending=False, inplace=True)
Defaulter_corr.head(10)
```

Out[137]:

	VAR1	VAR2	Correlation
0	AMT_INCOME_TOTAL	AMT_INCOME_TOTAL	1.0
242	DAYS_EMPLOYED	DAYS_EMPLOYED	1.0
66	AMT_GOODS_PRICE	AMT_GOODS_PRICE	1.0
88	DAYS_ID_PUBLISH	DAYS_ID_PUBLISH	1.0
110	HOUR_APPR_PROCESS_START	HOUR_APPR_PROCESS_START	1.0
132	REG_REGION_NOT_LIVE_REGION	REG_REGION_NOT_LIVE_REGION	1.0
154	OBS_60_CNT_SOCIAL_CIRCLE	OBS_60_CNT_SOCIAL_CIRCLE	1.0
176	DEF_60_CNT_SOCIAL_CIRCLE	DEF_60_CNT_SOCIAL_CIRCLE	1.0
198	REGION_POPULATION_RELATIVE	REGION_POPULATION_RELATIVE	1.0
264	DAYS_REGISTRATION	DAYS_REGISTRATION	1.0

In [138]:
```
fig = plt.figure(figsize=(20,15))
ax = sns.heatmap(Defaulter_df.corr(), cmap="Reds",annot=True,linewidth =1)
```

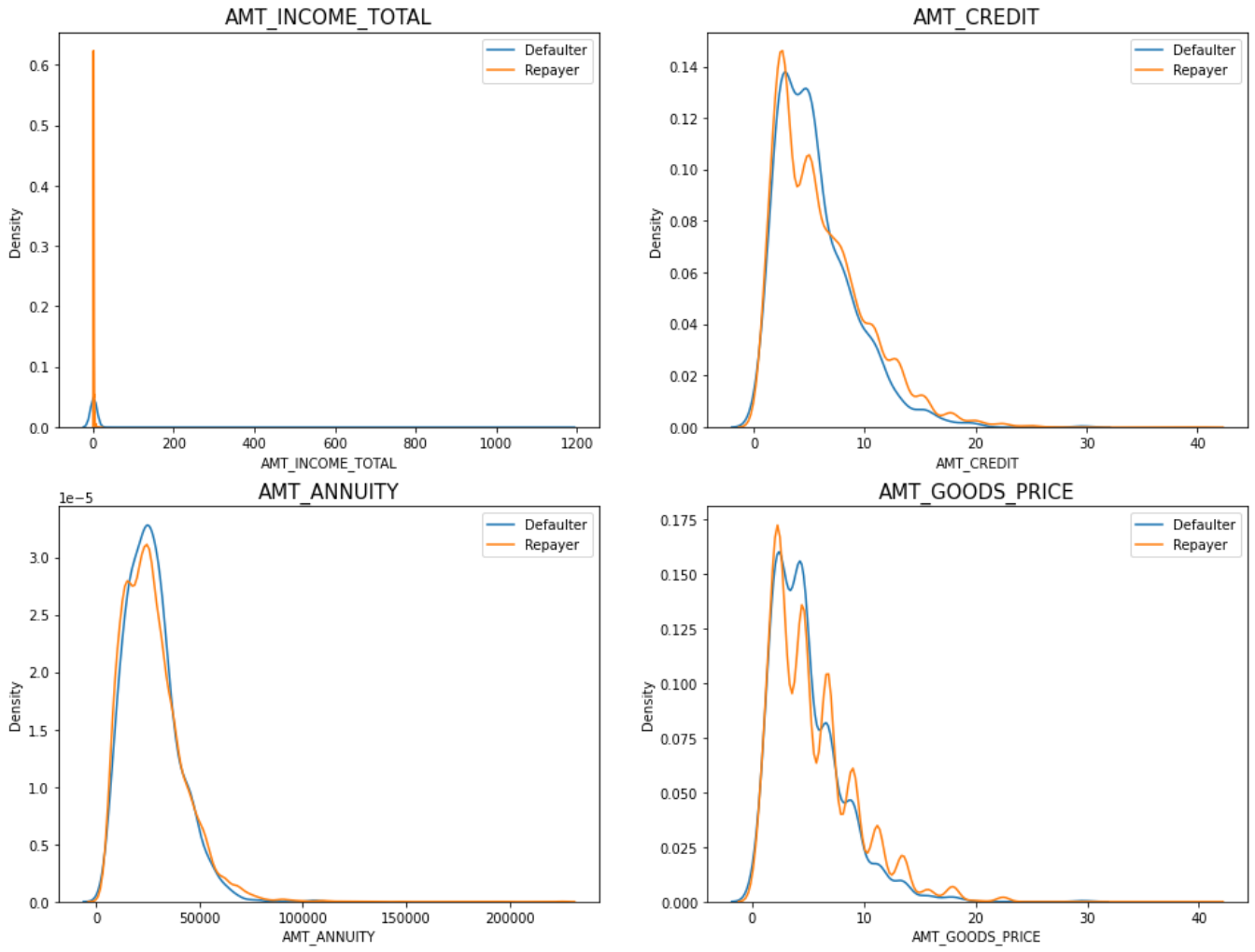


Observation: Correlating factors amongst defaulter

1. There is a high correlation between the goods price and Credit amount, which intur n infers that most of the applicants take loan topurchase consumer goods.
2. The correlation between credit amount and loan annuity is less among the defaulter (0.75) when compared to that of repayers (0.77).
3. Similarly even for the employed days, the repayers exibit more correlation than de faulters.
4. The correlation between credit amount and total income sees a major difference, as the repayers are seen to have fairly average correlation around (0.34), whereas for d efaulter (0.03), we can notice a sharp drop.
5. The correlation between days birth column and number of childrens column has dropp ed to 0.259 in defaulters when compared to that of the repayers 0.33.
6. There is a slight increase in defaulted to observed count in social circle among d efaulters(0.264) when compared to repayers(0.254)

Numerical Univariate Analysis

In [139]:
```
amount = bank_df[[ 'AMT_INCOME_TOTAL','AMT_CREDIT','AMT_ANNUITY', 'AMT_GOODS_PR
ICE']]

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

for i in enumerate(amount):
    plt.subplot(2,2,i[0]+1)
    sns.distplot(Defaulter_df[i[1]], hist=False,label ="Defaulter")
    sns.distplot(Repayer_df[i[1]], hist=False, label ="Repayer")
    plt.title(i[1], fontdict={'fontsize' : 15, 'fontweight' : 10})
    plt.legend()


plt.show()
```



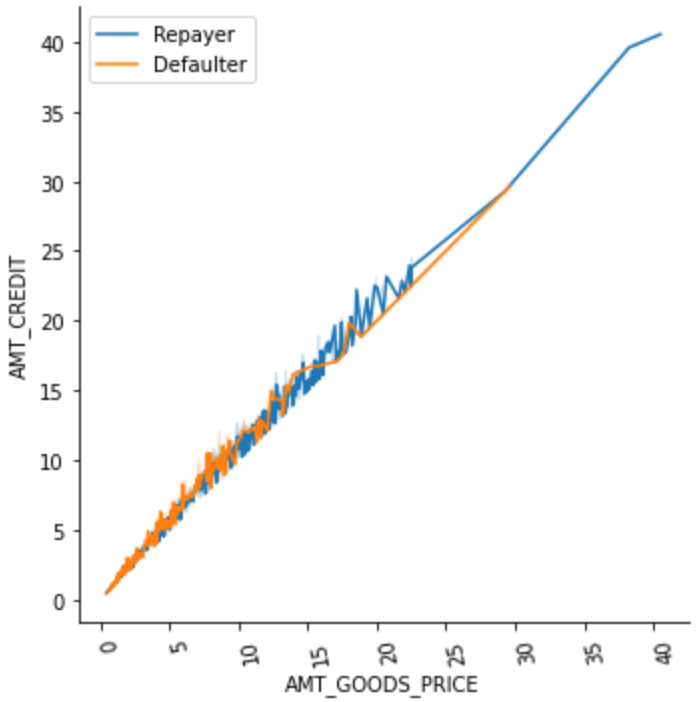
Observation:

1. Most number of loans range between 2 - 10 lakhs for the goods price
2. Most people pay annuity below 50K for the credit loan
3. Highest amount of Credit amount loans are for the amount 2-5 lakhs and majorly ranges from 2-10 lakhs bracket.

Numerical Bivariate data

In [140]:
```
# Checking the relationship between Goods price and credit and comparing with loan repayment staus
bi_var_numerical('AMT_GOODS_PRICE','AMT_CREDIT',bank_df,"TARGET", "line",['Repayer','Defaulter'])
```

```
<Figure size 1440x1440 with 0 Axes>
```



Observation:

The amount of repayers and defaulter are continuosly seens to raise with the increase in the loan amount. However, it can seen that the defaulter percentage seems to steadily raise but post 30 lakhs, there is spike in the defaulter percentage and the repayers percentage sees a small downfall.

Merged Dataframes Analysis

In [141]:
```
# Merging dataframe with the help of SK_ID_CURR with Inner Joins
loan_df = pd.merge(bank_df, pre_df, on='SK_ID_CURR')
loan_df.head()
```

Out[141]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_REALTY	CNT_CHILDREN
0	100006	0	Cash loans	F	Y	0
1	100007	0	Cash loans	M	Y	0
2	100007	0	Cash loans	M	Y	0
3	100009	0	Cash loans	F	Y	1
4	100012	0	Revolving loans	M	Y	0

5 rows × 82 columns

In [142]:
```
# Merged Dataframes information
loan_df.shape
```

Out[142]: (7416, 82)

In [143]:
```
# checking dataframes information
loan_df.info()
```

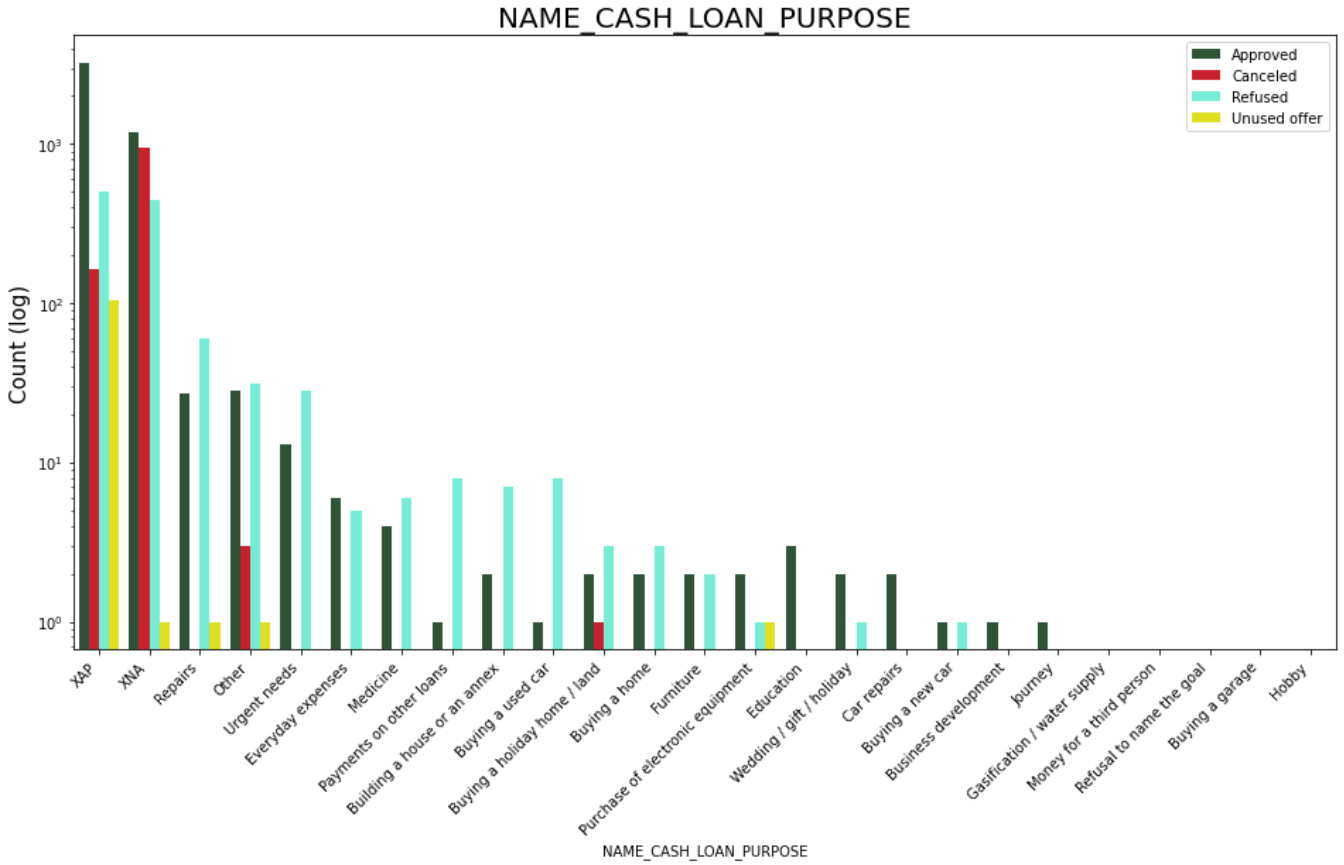
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7416 entries, 0 to 7415
Data columns (total 82 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   SK_ID_CURR                   7416 non-null   int64
 1   TARGET                       7416 non-null   int64
 2   NAME_CONTRACT_TYPE_x         7416 non-null   category
 3   CODE_GENDER                  7416 non-null   category
 4   FLAG_OWN_REALTY              7416 non-null   category
 5   CNT_CHILDREN                 7416 non-null   category
 6   AMT_INCOME_TOTAL             7416 non-null   float64
 7   AMT_CREDIT_x                 7416 non-null   float64
 8   AMT_ANNUITY_x                7416 non-null   float64
 9   AMT_GOODS_PRICE_x            7406 non-null   float64
 10  NAME_TYPE_SUITE_x            7399 non-null   category
 11  NAME_INCOME_TYPE             7416 non-null   category
 12  NAME_EDUCATION_TYPE          7416 non-null   category
 13  NAME_FAMILY_STATUS           7416 non-null   category
 14  NAME_HOUSING_TYPE            7416 non-null   category
 15  REGION_POPULATION_RELATIVE   7416 non-null   float64
 16  DAYS_BIRTH                   7416 non-null   int64
 17  DAYS_EMPLOYED                7416 non-null   int64
 18  DAYS_REGISTRATION            7416 non-null   float64
 19  DAYS_ID_PUBLISH              7416 non-null   int64
 20  FLAG_MOBIL                   7416 non-null   int64
 21  OCCUPATION_TYPE              7416 non-null   category
 22  CNT_FAM_MEMBERS              7416 non-null   category
 23  REGION_RATING_CLIENT         7416 non-null   category
 24  REGION_RATING_CLIENT_W_CITY  7416 non-null   category
 25  WEEKDAY_APPR_PROCESS_START   7416 non-null   category
 26  HOUR_APPR_PROCESS_START      7416 non-null   float64
 27  REG_REGION_NOT_LIVE_REGION   7416 non-null   float64
 28  REG_REGION_NOT_WORK_REGION   7416 non-null   category
 29  LIVE_REGION_NOT_WORK_REGION  7416 non-null   category
 30  REG_CITY_NOT_LIVE_CITY       7416 non-null   category
 31  REG_CITY_NOT_WORK_CITY       7416 non-null   category
 32  LIVE_CITY_NOT_WORK_CITY      7416 non-null   category
 33  ORGANIZATION_TYPE            7416 non-null   category
 34  OBS_30_CNT_SOCIAL_CIRCLE     7406 non-null   float64
 35  DEF_30_CNT_SOCIAL_CIRCLE     7406 non-null   float64
 36  OBS_60_CNT_SOCIAL_CIRCLE     7406 non-null   float64
 37  DEF_60_CNT_SOCIAL_CIRCLE     7406 non-null   float64
 38  DAYS_LAST_PHONE_CHANGE       7416 non-null   float64
 39  FLAG_DOCUMENT_3              7416 non-null   float64
 40  AMT_REQ_CREDIT_BUREAU_HOUR   7416 non-null   float64
 41  AMT_REQ_CREDIT_BUREAU_DAY    7416 non-null   float64
 42  AMT_REQ_CREDIT_BUREAU_WEEK   7416 non-null   float64
 43  AMT_REQ_CREDIT_BUREAU_MON    7416 non-null   float64
 44  AMT_REQ_CREDIT_BUREAU_QRT    7416 non-null   float64
 45  AMT_REQ_CREDIT_BUREAU_YEAR   7416 non-null   float64
 46  INCOME_RANGE                 7412 non-null   category
 47  CREDIT_RANGE                 7416 non-null   category
 48  GOODS_PRICE_RANGE            7406 non-null   category
 49  AGE                          7416 non-null   float64
 50  AGE_GROUP                    7416 non-null   category
 51  YEARS_EMPLOYED               7416 non-null   float64
 52  EMPLOYEMENT_YEARS            6047 non-null   category
 53  SK_ID_PREV                   7416 non-null   int64
 54  NAME_CONTRACT_TYPE_y         7416 non-null   category
 55  AMT_ANNUITY_y                7416 non-null   float64
```

```
 56  AMT_APPLICATION              7416 non-null   float64
 57  AMT_CREDIT_y                 7416 non-null   float64
 58  AMT_GOODS_PRICE_y            5907 non-null   float64
 59  NAME_CASH_LOAN_PURPOSE       7416 non-null   category
 60  NAME_CONTRACT_STATUS         7416 non-null   category
 61  DAYS_DECISION                7416 non-null   int64
 62  NAME_PAYMENT_TYPE            7416 non-null   category
 63  CODE_REJECT_REASON           7416 non-null   category
 64  NAME_TYPE_SUITE_y            7416 non-null   object
 65  NAME_CLIENT_TYPE             7416 non-null   category
 66  NAME_GOODS_CATEGORY          7416 non-null   category
 67  NAME_PORTFOLIO               7416 non-null   category
 68  NAME_PRODUCT_TYPE            7416 non-null   category
 69  CHANNEL_TYPE                 7416 non-null   category
 70  SELLERPLACE_AREA             7416 non-null   float64
 71  NAME_SELLER_INDUSTRY         7416 non-null   category
 72  CNT_PAYMENT                  5939 non-null   float64
 73  NAME_YIELD_GROUP             7416 non-null   category
 74  PRODUCT_COMBINATION          7414 non-null   category
 75  DAYS_FIRST_DRAWING           4641 non-null   float64
 76  DAYS_FIRST_DUE               4641 non-null   float64
 77  DAYS_LAST_DUE_1ST_VERSION    4641 non-null   float64
 78  DAYS_LAST_DUE                4641 non-null   float64
 79  DAYS_TERMINATION             4641 non-null   float64
 80  NFLAG_INSURED_ON_APPROVAL    4641 non-null   float64
 81  YEARLY_DECISION              7416 non-null   category
dtypes: category(39), float64(34), int64(8), object(1)
memory usage: 2.8+ MB
```

In [144]:
```
# Dividing loan dataset between repayers and defaulters

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

Plotting Contract Status vs purpose of the loan

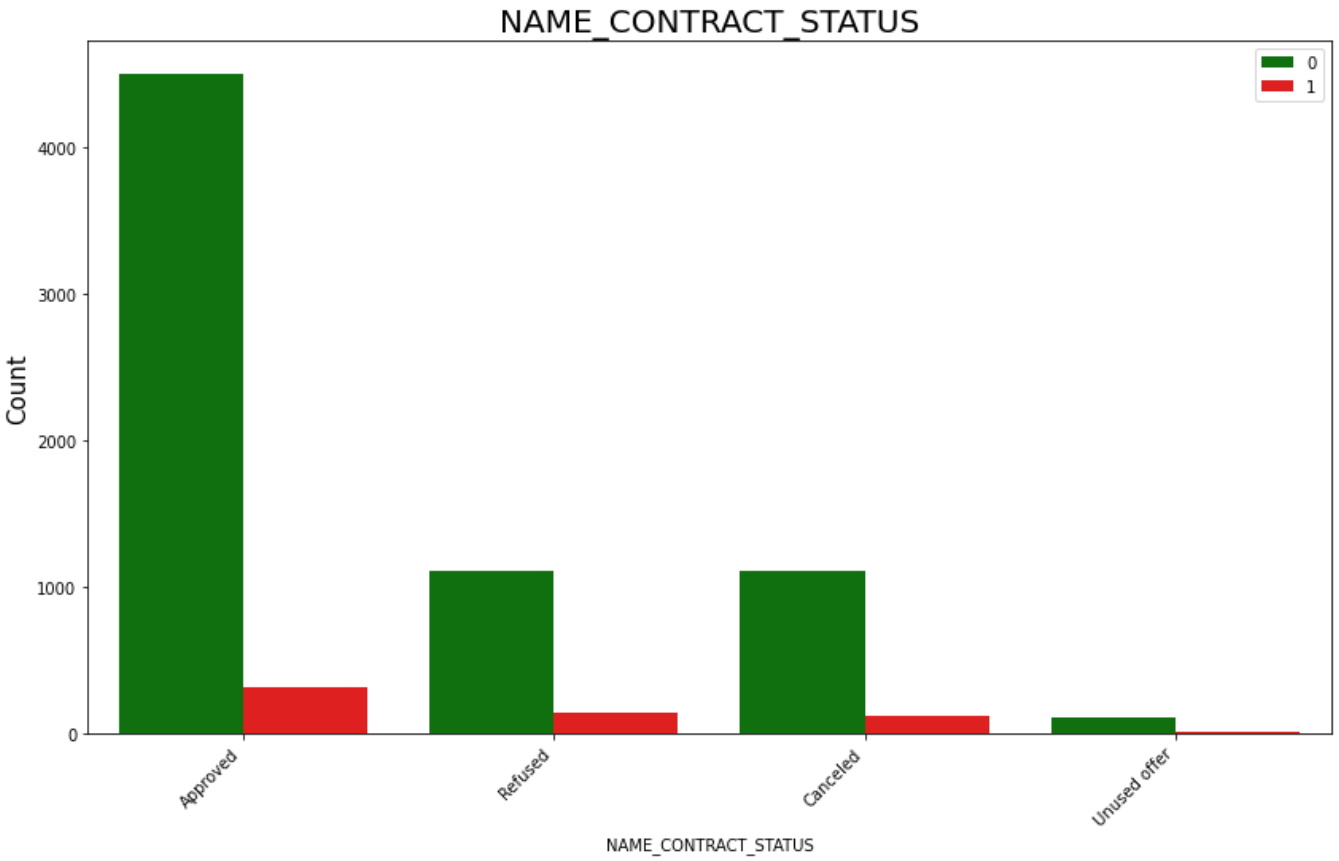
In [145]:
```
uni_var_c_merged("NAME_CASH_LOAN_PURPOSE",L0,"NAME_CONTRACT_STATUS",["#295934","#e40517","#64ffdf","#ffff00"],True,(16,8))
```



Observations:

1. Majority of the loan applicants have not mentioned the reason for loan (XAP, XNA), which is a disadvantage to the bank as this informtion would have been valuable to the bank as well as the concerned industry.
2. Loan provided for Repairs are seen to default all the time. Thus majority of application with Repairs as a pupose are rejected by the bank.

In [146]:
```
# Analysing the relationship of contract status with the loan repayment

uni_var_c_merged("NAME_CONTRACT_STATUS",loan_df,"TARGET",['g','r'],False,(14,
8))
r = loan_df.groupby("NAME_CONTRACT_STATUS")["TARGET"]
df = pd.concat([r.value_counts(),round(r.value_counts(normalize=False).mul(10
0),2)],axis=1, keys=('Counts','Percentage'))
df['Percentage'] = df['Percentage'].astype(str) +"%"
df
```



Out[146]:

		Counts	Percentage
NAME_CONTRACT_STATUS	**TARGET**		
Approved	**0**	4497	449700%
	1	315	31500%
Canceled	**0**	1107	110700%
	1	120	12000%
Refused	**0**	1113	111300%
	1	145	14500%
Unused offer	**0**	109	10900%
	1	10	1000%

Observation:

1. 90% of the client who cancelled the offers previously have repayed the loan. So, if the bank takes feedback from its clients and revise the inrest rate according to the client need it might help the bank in retaining its clients.
2. Previously 88% of the refused applicants, was seen to pay back loan in this tnure. Which can mean that they can be a future prospects to the bank.
3. Among the approved offers, around 92% of loan takers repayed the loan and only 8% of them tend to default.

Refusal reason should be recorded for further analysis as these clients could turn in to potential repaying customer.

In [147]:
```
bank_df.dtypes
```

Out[147]:
```
SK_ID_CURR                      int64
TARGET                          int64
NAME_CONTRACT_TYPE           category
CODE_GENDER                  category
FLAG_OWN_REALTY              category
CNT_CHILDREN                 category
AMT_INCOME_TOTAL              float64
AMT_CREDIT                    float64
AMT_ANNUITY                   float64
AMT_GOODS_PRICE               float64
NAME_TYPE_SUITE              category
NAME_INCOME_TYPE             category
NAME_EDUCATION_TYPE          category
NAME_FAMILY_STATUS           category
NAME_HOUSING_TYPE            category
REGION_POPULATION_RELATIVE    float64
DAYS_BIRTH                      int64
DAYS_EMPLOYED                   int64
DAYS_REGISTRATION             float64
DAYS_ID_PUBLISH                 int64
FLAG_MOBIL                      int64
OCCUPATION_TYPE              category
CNT_FAM_MEMBERS              category
REGION_RATING_CLIENT         category
REGION_RATING_CLIENT_W_CITY  category
WEEKDAY_APPR_PROCESS_START   category
HOUR_APPR_PROCESS_START       float64
REG_REGION_NOT_LIVE_REGION    float64
REG_REGION_NOT_WORK_REGION   category
LIVE_REGION_NOT_WORK_REGION  category
REG_CITY_NOT_LIVE_CITY       category
REG_CITY_NOT_WORK_CITY       category
LIVE_CITY_NOT_WORK_CITY      category
ORGANIZATION_TYPE            category
OBS_30_CNT_SOCIAL_CIRCLE      float64
DEF_30_CNT_SOCIAL_CIRCLE      float64
OBS_60_CNT_SOCIAL_CIRCLE      float64
DEF_60_CNT_SOCIAL_CIRCLE      float64
DAYS_LAST_PHONE_CHANGE        float64
FLAG_DOCUMENT_3               float64
AMT_REQ_CREDIT_BUREAU_HOUR    float64
AMT_REQ_CREDIT_BUREAU_DAY     float64
AMT_REQ_CREDIT_BUREAU_WEEK    float64
AMT_REQ_CREDIT_BUREAU_MON     float64
AMT_REQ_CREDIT_BUREAU_QRT     float64
AMT_REQ_CREDIT_BUREAU_YEAR    float64
INCOME_RANGE                 category
CREDIT_RANGE                 category
GOODS_PRICE_RANGE            category
AGE                           float64
AGE_GROUP                    category
YEARS_EMPLOYED                float64
EMPLOYEMENT_YEARS            category
dtype: object
```

In [148]:
```
bank_df = bank_df.astype('category')
```

In [149]:
```
labelencoder=LabelEncoder()
for col in bank_df.columns:
    bank_df[col] = labelencoder.fit_transform(bank_df[col])
```

Data preparation

For our further analysis we will be dividing the data into test and train sets

In [150]:
```
# Creating test and train dataset for analysis

x = bank_df.drop(['TARGET'], axis=1)
y = bank_df["TARGET"]

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=42, test
_size=0.1)
```

Classification Methods

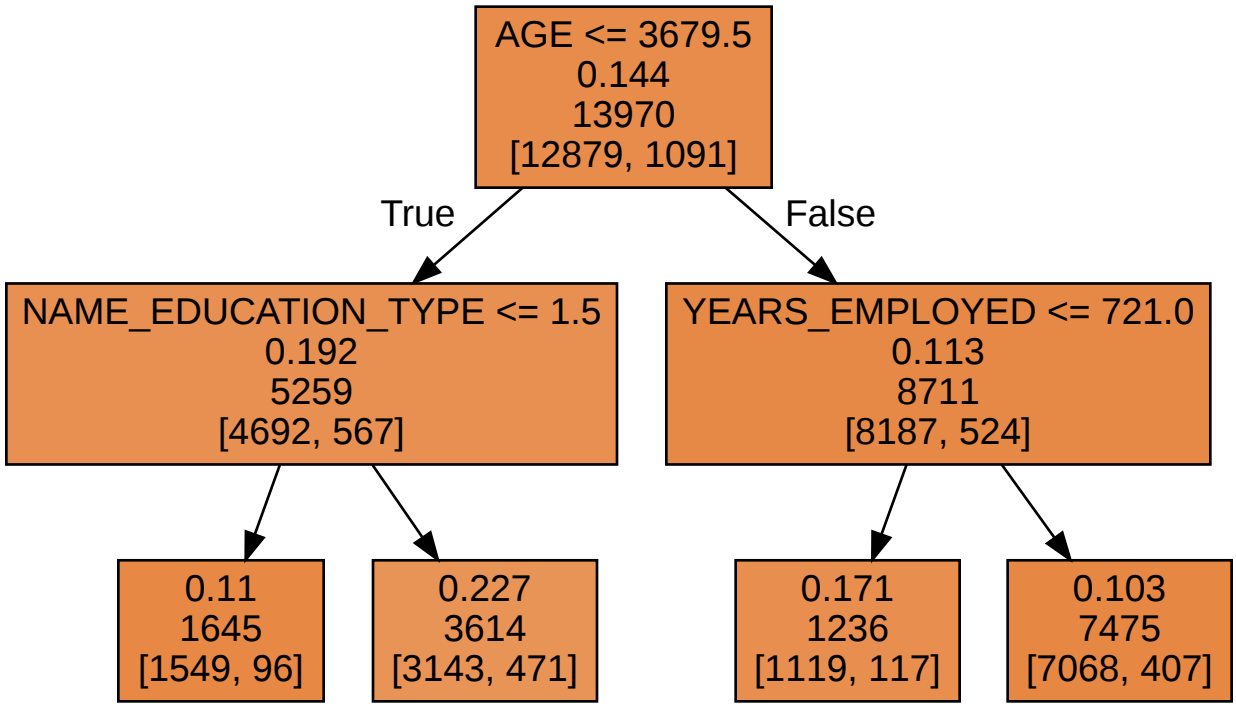
Developing a Decision Tree to determine the finest attributes

In [151]:
```
## Importing Decision Tree from the sklearn library for creating decision tree

from sklearn.tree import DecisionTreeClassifier

bank_dt = DecisionTreeClassifier(random_state = 50, max_depth = 2, min_samples_
leaf=4)
bank_dt.fit(x_train, y_train)
```

Out[151]:
```
DecisionTreeClassifier(max_depth=2, min_samples_leaf=4, random_state=50)
```

In [152]:
```
## Creating the decision tree

os.environ["PATH"] += os.pathsep + 'C:\\Users\\sindh\\anaconda3\\Lib\\site-packages\\graphviz'

dot1 = export_graphviz(bank_dt, out_file=None,label=all,
                           feature_names=x.columns,
                           filled=True, rounded=False, impurity=bool,
                           special_characters=False,leaves_parallel=False)
graph1 = graphviz.Source(dot1)
graph1
```

Out[152]:



Confirming on the parameter for analysis

Observation:

The decision tree shown above makes it abundantly evident that the parameter "EXT_SOURCE_3" is the most effective at illustrating the many applications in the dataset. The "EXT_SOURCE_3" decision node serves as the foundation for most classifications that can be applied to the dataset.

In [153]:
```
#LISTING THE APPROPRIATE FEATURE FOR ANALYSIS

list_feat = x.columns.values
important_feat = bank_dt.feature_importances_
index_sort = np.argsort(important_feat)

plt.figure(figsize=(11,19))
plt.barh(range(len(index_sort)), important_feat[index_sort], align='center', co
lor ="green")
plt.yticks(range(len(index_sort)), list_feat[index_sort])
plt.xlabel('Importance of Features')
plt.title('Important Features')
plt.draw()
plt.show()
```


Important Features
AGE
NAME_EDUCATION_TYPE
YEARS_EMPLOYED
NAME_CONTRACT_TYPE
CODE_GENDER
REGION_RATING_CLIENT_W_CITY
REGION_RATING_CLIENT
CNT_FAM_MEMBERS
OCCUPATION_TYPE
FLAG_MOBIL
DAYS_ID_PUBLISH
DAYS_REGISTRATION
DAYS_EMPLOYED
DAYS_BIRTH
REGION_POPULATION_RELATIVE
NAME_HOUSING_TYPE
NAME_FAMILY_STATUS
WEEKDAY_APPR_PROCESS_START
NAME_INCOME_TYPE
NAME_TYPE_SUITE
AMT_GOODS_PRICE
AMT_ANNUITY
AMT_CREDIT
AMT_INCOME_TOTAL
CNT_CHILDREN
FLAG_OWN_REALTY
EMPLOYEMENT_YEARS
HOUR_APPR_PROCESS_START
REG_REGION_NOT_LIVE_REGION
REG_REGION_NOT_WORK_REGION
AGE_GROUP
GOODS_PRICE_RANGE
CREDIT_RANGE
INCOME_RANGE
AMT_REQ_CREDIT_BUREAU_YEAR
AMT_REQ_CREDIT_BUREAU_QRT
AMT_REQ_CREDIT_BUREAU_MON
AMT_REQ_CREDIT_BUREAU_WEEK
AMT_REQ_CREDIT_BUREAU_DAY
AMT_REQ_CREDIT_BUREAU_HOUR
FLAG_DOCUMENT_3
DAYS_LAST_PHONE_CHANGE
DEF_60_CNT_SOCIAL_CIRCLE
OBS_60_CNT_SOCIAL_CIRCLE
DEF_30_CNT_SOCIAL_CIRCLE
OBS_30_CNT_SOCIAL_CIRCLE
ORGANIZATION_TYPE
LIVE_CITY_NOT_WORK_CITY
REG_CITY_NOT_WORK_CITY
REG_CITY_NOT_LIVE_CITY
LIVE_REGION_NOT_WORK_REGION
SK_ID_CURR
0.0
0.1
0.2
0.3
0.4
0.5
Importance of Features

Observation:

The significance of the abovementioned feature graph offers a clear image of all the characteristics that might play a significant influence. Additionally, it is demonstrated that the value "DAYS_EMPLOYED" is crucial compared to the other parameters. The decision tree diagram, where "DAYS_EMPLOYED" is the major decision node and the other parameters are divided based on it, also makes this point very clear.

Estimating and running predictions for findings

In [154]:
```
y_pred = bank_dt.predict(x_test)
```

In [155]:
```
print("Classification report from Decision Tree: \n", classification_report(y_test, y_pred))
```

```
Classification report from Decision Tree:
               precision    recall  f1-score   support

           0       0.92      1.00      0.96      1427
           1       0.00      0.00      0.00       126

    accuracy                           0.92      1553
   macro avg       0.46      0.50      0.48      1553
weighted avg       0.84      0.92      0.88      1553
```

In [156]:
```
print("Test Accuracy: {} Percent".format(round(bank_dt.score(x_test, y_test)*100, 2)))
```

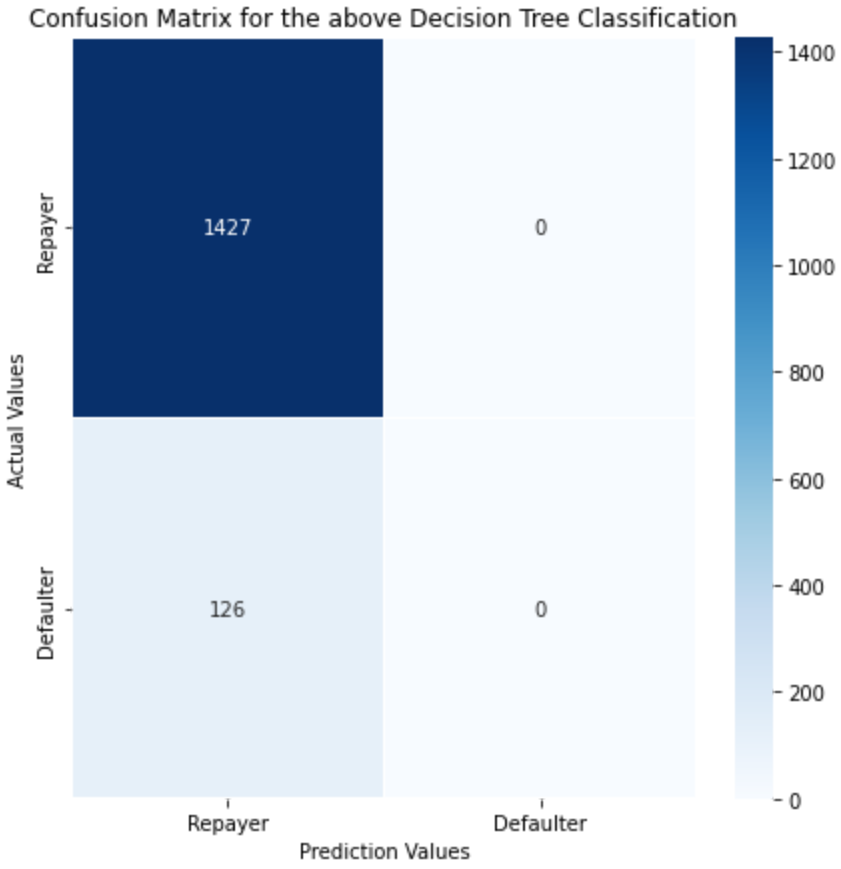
```
Test Accuracy: 91.89 Percent
```

Observation:

From the aforementioned test, we can observe that the decision tree test has an overall accuracy of 91.81%, proving that the model fits the data well.

Building a confusion matrix for the classification decision tree mentioned above

In [157]:
```
## Building confusion matrix for the above decision tree classification

con_mat = confusion_matrix(y_test, y_pred)

x_labels = ["Repayer", "Defaulter"]
y_labels = ["Repayer", "Defaulter"]

fig_size = plt.subplots(figsize =(7,7))
sns.heatmap(con_mat, annot = True, linewidths=0.5, linecolor="white", fmt = ".0
f", cmap="Blues", xticklabels=x_labels, yticklabels=y_labels)
plt.xlabel("Prediction Values")
plt.ylabel("Actual Values")
plt.title('Confusion Matrix for the above Decision Tree Classification')
plt.show()
```



Observation:

The heatmap shown above illustrates the relationship between two variables, such as "Repayer" and "Defaulter," and plots their values in relation to predictions and actual values. The heatmap shows that defaulters are also repayers, as can be shown.

Logistic regression-based classification

In [158]:
```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(solver="lbfgs", max_iter=500)
log_reg.fit(x_train, y_train)



print("Test Accuracy: {} Percent".format(round(log_reg.score(x_test, y_test)*10
0,2)))
```

```
Test Accuracy: 91.89 Percent
```

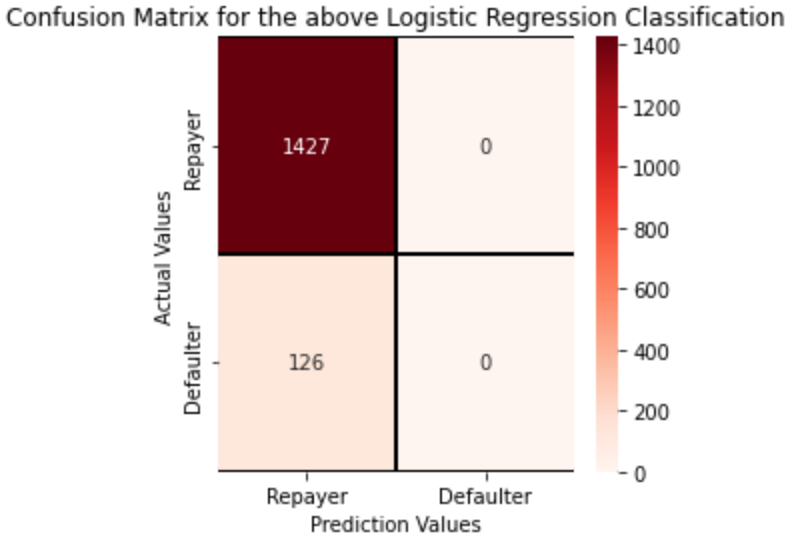
Building the confusion matrix for the abovementioned Logistic Regression

In [159]:
```
y_lr_pred = log_reg.predict(x_test)
print("Classification report from Logistic Regression: \n", classification_repo
rt(y_test, y_lr_pred))
```

```
Classification report from Logistic Regression:
               precision    recall  f1-score   support

           0       0.92      1.00      0.96      1427
           1       0.00      0.00      0.00       126

    accuracy                           0.92      1553
   macro avg       0.46      0.50      0.48      1553
weighted avg       0.84      0.92      0.88      1553
```

Building the confusion matrix for the logistic regression classification mentioned above

In [160]:
```
con_mat_lr = confusion_matrix(y_test, y_lr_pred)

x_labels = ["Repayer", "Defaulter"]
y_labels = ["Repayer", "Defaulter"]

fig_size = plt.subplots(figsize =(4,4))
sns.heatmap(con_mat_lr, annot = True, linewidths=0.2, linecolor="black", fmt =
".0f", cmap="Reds", xticklabels=x_labels, yticklabels=y_labels)
plt.xlabel("Prediction Values")
plt.ylabel("Actual Values")
plt.title('Confusion Matrix for the above Logistic Regression Classification')
plt.show()
```



CREATING A FUNCTION FOR CALCULATING THE CO-EFFICIENT OF DETERMINATION

In [161]:
```
# creating a function to create adjusted R-Squared

def adj_r2(x, y, model):
    r2 = model.score(x, y)
    n = x.shape[0]
    p = x.shape[1]
    adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)

    return adjusted_r2
```

In [162]:
```
print(adj_r2(x_train, y_train, log_reg))
```

```
0.9216122796522116
```

In [163]:
```
print(adj_r2(x_test, y_test, log_reg))
```

```
0.9160540888602705
```

PERFORMING L1 REGULARIZATION (LASSO MODEL)

In [164]:
```
from sklearn.linear_model import Lasso, LassoCV

lasso_cv = LassoCV(alphas = None, cv = 10, max_iter = 100000, normalize = True)
lasso_cv.fit(x_train, y_train)
```

Out[164]:
```
LassoCV(cv=10, max_iter=100000, normalize=True)
```

In [165]:
```
# best alpha parameter

alpha = lasso_cv.alpha_
alpha
```

Out[165]:
```
1.2239153929732719e-05
```

In [166]:
```
lasso = Lasso(alpha = lasso_cv.alpha_)
lasso.fit(x_train, y_train)
```

Out[166]:
```
Lasso(alpha=1.2239153929732719e-05)
```

In [167]:
```
lasso.score(x_train, y_train)
```

Out[167]:
```
0.03101587117617588
```

In [168]:
```
lasso.score(x_test, y_test)
```

Out[168]:
```
0.036463084284421576
```

In [169]:
```
print(adj_r2(x_train, y_train, lasso))
```

```
0.027395322588201543
```

In [170]:
```
print(adj_r2(x_test, y_test, lasso))
```

```
0.003060471206281501
```

PERFORMING L2 REGULARIZATION (Ridge Regression)

In [171]:
```
from sklearn.linear_model import Ridge, RidgeCV

alphas = np.random.uniform(0, 10, 50)
ridge_cv = RidgeCV(alphas = alphas, cv = 10, normalize = True)
ridge_cv.fit(x_train, y_train)
```

Out[171]:
```
RidgeCV(alphas=array([1.14211121, 3.81973139, 6.77577552, 0.0510736 , 9.3540432
,
       6.44770634, 4.99264509, 4.94204534, 8.27154351, 3.1397489 ,
       2.60337586, 2.20935024, 8.270013  , 4.78186443, 2.25281638,
       7.3048386 , 6.31655819, 0.93714888, 9.95001881, 3.96630535,
       6.31622642, 5.10008329, 5.88683519, 1.13044605, 3.43472638,
       4.45672041, 8.01580091, 9.59634823, 5.37465736, 4.53721292,
       0.03343189, 2.96407588, 5.19143017, 0.14945474, 3.71354887,
       8.86331957, 3.84334754, 0.26562984, 5.34238949, 8.45926221,
       5.88199976, 5.08133596, 0.03119717, 1.83248923, 2.94647752,
       5.09201188, 1.94836237, 8.39048323, 9.18026162, 9.80624254]),
        cv=10, normalize=True)
```

In [172]:
```
# best alpha parameter

alpha = ridge_cv.alpha_
alpha
```

Out[172]:
```
0.14945473551021937
```

In [173]:
```
ridge = Ridge(alpha = ridge_cv.alpha_)
ridge.fit(x_train, y_train)
```

Out[173]:
```
Ridge(alpha=0.14945473551021937)
```

In [174]:
```
ridge.score(x_train, y_train)
```

Out[174]:
```
0.031023564995534825
```

In [175]:
```
ridge.score(x_test, y_test)
```

Out[175]:
```
0.03615028602814241
```

In [176]:
```
print(adj_r2(x_train, y_train, ridge))
```

```
0.027403045155035333
```

In [177]:
```
print(adj_r2(x_test, y_test, ridge))
```

```
0.002736829277117936
```

Classification based on Linear Regression

In [178]:
```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(x_train, y_train)



print("Test Accuracy: {} Percent".format(round(log_reg.score(x_test, y_test)*10
0,2)))
```

```
Test Accuracy: 91.89 Percent
```

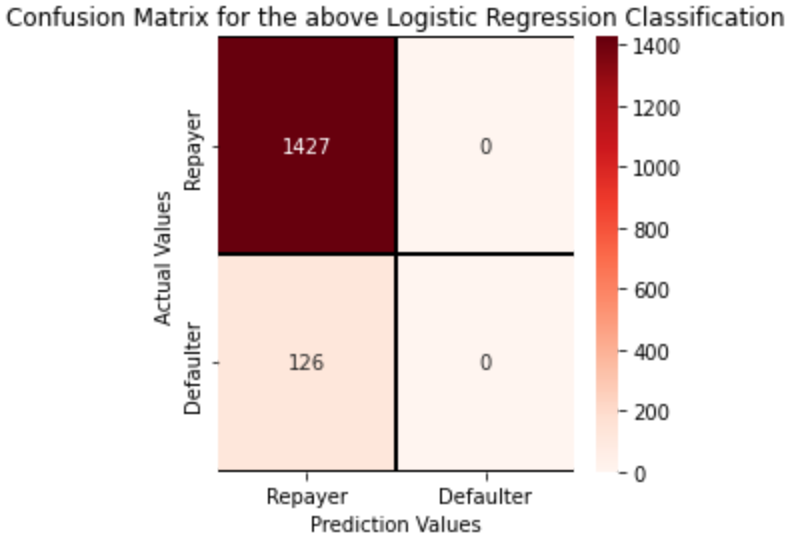
Constructing confusion matrix for the above Linear Regression

In [179]:
```
y_ln_pred = lin_reg.predict(x_test)
print("Classification report from Logistic Regression: \n", classification_repo
rt(y_test, y_lr_pred))
```

```
Classification report from Logistic Regression:
               precision    recall  f1-score   support

           0       0.92      1.00      0.96      1427
           1       0.00      0.00      0.00       126

    accuracy                           0.92      1553
   macro avg       0.46      0.50      0.48      1553
weighted avg       0.84      0.92      0.88      1553
```

Constructing confusion matrix for the above Linear Regression Classification

In [180]:
```
con_mat_ln = confusion_matrix(y_test, y_lr_pred)

x_labels = ["Repayer", "Defaulter"]
y_labels = ["Repayer", "Defaulter"]

fig_size = plt.subplots(figsize =(4,4))
sns.heatmap(con_mat_ln, annot = True, linewidths=0.2, linecolor="black", fmt =
".0f", cmap="Reds", xticklabels=x_labels, yticklabels=y_labels)
plt.xlabel("Prediction Values")
plt.ylabel("Actual Values")
plt.title('Confusion Matrix for the above Logistic Regression Classification')
plt.show()
```



In [181]:
```
# creating a function to create adjusted R-Squared

def adju_r2(x, y, model):
    r2 = model.score(x, y)
    n = x.shape[0]
    p = x.shape[1]
    adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)

    return adjusted_r2
```

In [182]:
```
print(adju_r2(x_train, y_train, lin_reg))
```

```
0.027398721493724132
```

In [183]:
```
print(adju_r2(x_test, y_test, lin_reg))
```

```
0.0026240465788027745
```

In [184]:
```
from sklearn.linear_model import Lasso, LassoCV

lasso_cv = LassoCV(alphas = None, cv = 10, max_iter = 100000, normalize = True)
lasso_cv.fit(x_train, y_train)
```

Out[184]:
```
LassoCV(cv=10, max_iter=100000, normalize=True)
```

In [185]:
```
# best alpha parameter

alpha = lasso_cv.alpha_
alpha
```

Out[185]:
```
1.2239153929732719e-05
```

In [186]:
```
lasso = Lasso(alpha = lasso_cv.alpha_)
lasso.fit(x_train, y_train)
```

Out[186]:
```
Lasso(alpha=1.2239153929732719e-05)
```

In [187]:
```
lasso.score(x_train, y_train)
```

Out[187]:
```
0.03101587117617588
```

In [188]:
```
lasso.score(x_test, y_test)
```

Out[188]:
```
0.036463084284421576
```

In [189]:
```
print(adju_r2(x_train, y_train, lasso))
```

```
0.027395322588201543
```

In [190]:
```
print(adju_r2(x_test, y_test, lasso))
```

```
0.003060471206281501
```

Ridge Regression

In [191]:
```
from sklearn.linear_model import Ridge, RidgeCV

alphas1 = np.random.uniform(0, 10, 50)
ridg_cv = RidgeCV(alphas = alphas1, cv = 10, normalize = True)
ridg_cv.fit(x_train, y_train)
```

Out[191]:
```
RidgeCV(alphas=array([6.99555104, 5.9377745 , 4.05084633, 8.58579687, 2.3791640
4,
       7.03806948, 2.27664936, 7.0485508 , 0.58160436, 5.00715505,
       8.95218635, 4.28695479, 4.28128787, 4.63214456, 1.02637537,
       2.88358752, 8.57811033, 7.66964674, 9.98961493, 7.46221199,
       3.05821481, 7.55980398, 5.39864324, 6.61594107, 1.40899923,
       2.86611885, 7.21413605, 0.83745836, 2.61993817, 2.92199977,
       4.03861479, 4.86145023, 7.15339892, 5.77137816, 6.94927978,
       9.19614797, 0.86075852, 9.71414892, 3.78240883, 8.40932482,
       7.35984579, 9.3934196 , 2.82206826, 1.85031583, 2.80455107,
       4.7090223 , 1.10018037, 3.51850168, 3.189029  , 5.81740564]),
        cv=10, normalize=True)
```

In [192]:
```
# best alpha parameter

alpha1 = ridg_cv.alpha_
alpha1
```

Out[192]:
```
0.581604356410701
```

In [193]:
```
ridg = Ridge(alpha = ridg_cv.alpha_)
ridg.fit(x_train, y_train)
```

Out[193]:
```
Ridge(alpha=0.581604356410701)
```

In [194]:
```
ridg.score(x_train, y_train)
```

Out[194]:
```
0.031023491337985654
```

In [195]:
```
ridg.score(x_test, y_test)
```

Out[195]:
```
0.036172421044908476
```

In [196]:
```
print(adj_r2(x_train, y_train, ridg))
```

```
0.027402971222269312
```

In [197]:
```
print(adj_r2(x_test, y_test, ridg))
```

```
0.0027597316411319683
```

Classifying with Random Forest

In [198]:
```
from sklearn.ensemble import RandomForestClassifier

rand_for = RandomForestClassifier(n_estimators=100, random_state=42)
rand_for.fit(x_train, y_train)

print("Test Accuracy: {} Percent".format(round(rand_for.score(x_test, y_test)*1
00, 2)))
```

```
Test Accuracy: 91.89 Percent
```

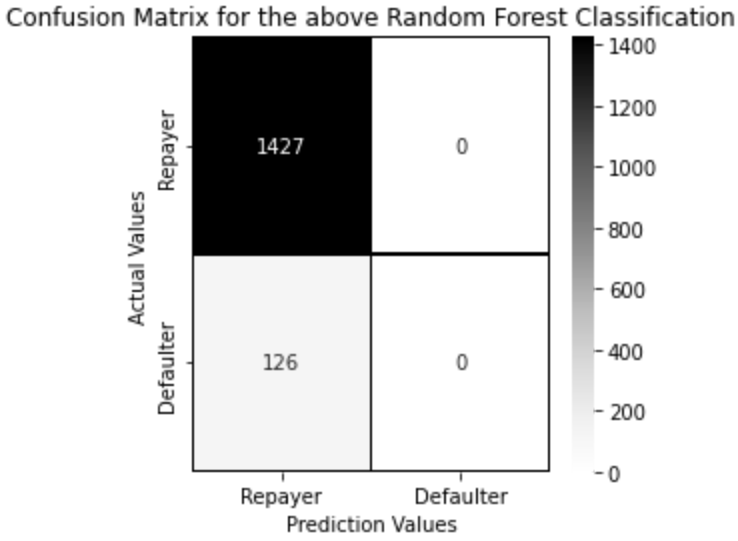
Classification based on Random Forest Method

In [199]:
```
y_rf_pred = rand_for.predict(x_test)
print("Classification report from Random Forest : \n", classification_report(y_
test, y_rf_pred))
```

```
Classification report from Random Forest :
               precision    recall  f1-score   support

           0       0.92      1.00      0.96      1427
           1       0.00      0.00      0.00       126

    accuracy                           0.92      1553
   macro avg       0.46      0.50      0.48      1553
weighted avg       0.84      0.92      0.88      1553
```

Constructing confusion matrix for the above Random Forest Classification

In [206]:
```
cm_rf = confusion_matrix(y_test, y_rf_pred)

x_labels = ["Repayer", "Defaulter"]
y_labels = ["Repayer", "Defaulter"]

fig_size = plt.subplots(figsize =(4,4))
sns.heatmap(cm_rf, annot = True, linewidths=0.2, linecolor="black", fmt = ".0
f", cmap="Greys", xticklabels=x_labels, yticklabels=y_labels)
plt.xlabel("Prediction Values")
plt.ylabel("Actual Values")
plt.title('Confusion Matrix for the above Random Forest Classification');
plt.show()
```



Support Vector Machines classification

In [200]:
```
from sklearn.svm import SVC

class_svm = SVC(random_state=42, gamma="auto")
class_svm.fit(x_train, y_train)

print("Test Accuracy: {} Percent".format(round(class_svm.score(x_test, y_test)*
100, 2)))
```

```
Test Accuracy: 91.89 Percent
```

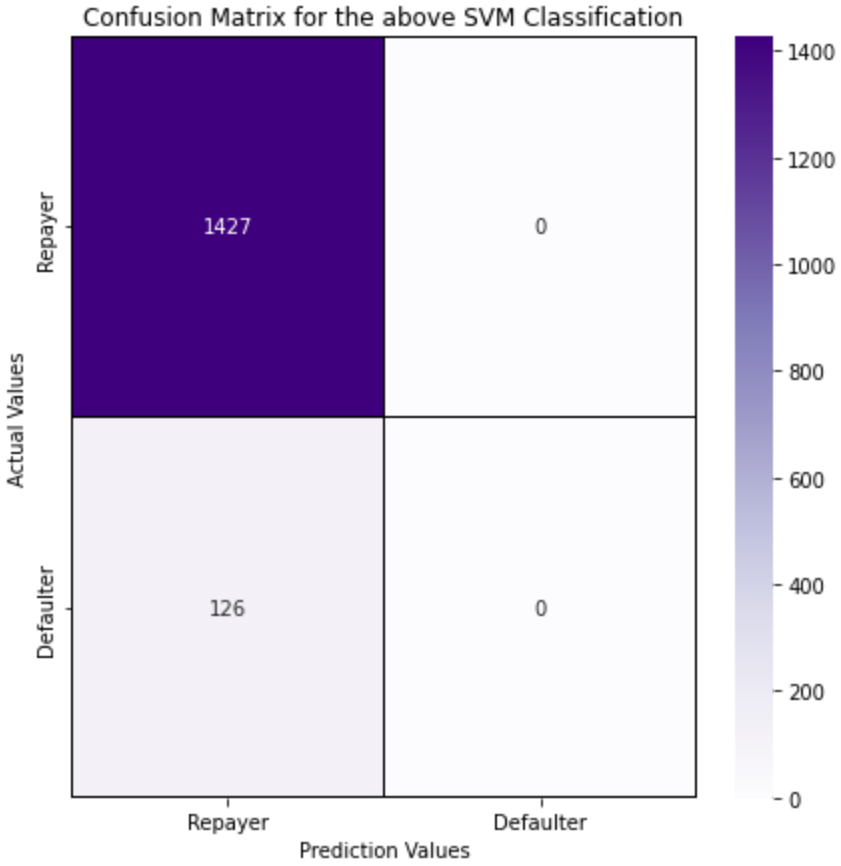
Classification based on Support Vector Machines

In [201]:
```
y_svm_pred = class_svm.predict(x_test)
print("Classification report from SVM : \n", classification_report(y_test, y_sv
m_pred))
```

```
Classification report from SVM :
               precision    recall  f1-score   support

           0       0.92      1.00      0.96      1427
           1       0.00      0.00      0.00       126

    accuracy                           0.92      1553
   macro avg       0.46      0.50      0.48      1553
weighted avg       0.84      0.92      0.88      1553
```

Constructing confusion matrix for the above SVM Classification

In [202]:
```
cm_svm = confusion_matrix(y_test, y_svm_pred)

x_labels = ["Repayer", "Defaulter"]
y_labels = ["Repayer", "Defaulter"]

fig_size = plt.subplots(figsize =(7,7))
sns.heatmap(cm_svm, annot = True, linewidths=0.2, linecolor="black", fmt = ".0
f", cmap="Purples", xticklabels=x_labels, yticklabels=y_labels)
plt.xlabel("Prediction Values")
plt.ylabel("Actual Values")
plt.title('Confusion Matrix for the above SVM Classification')
plt.show()
```



Classification based on K - Nearest Neighbors

In [203]:
```
from sklearn.neighbors import KNeighborsClassifier

b_Kvalue = 0
b_score = 0

for i in range(1,10):
    k_near = KNeighborsClassifier(n_neighbors=i)
    k_near.fit(x_train, y_train)
    if k_near.score(x_test, y_test) > b_score:
        best_score = k_near.score(x_train, y_train)
        best_Kvalue = i

print("Fitting KNN Value: {}".format(best_Kvalue))
print("Test Accuracy: {} Percent".format(round(best_score*100,2)))
```

```
Fitting KNN Value: 9
Test Accuracy: 92.17 Percent
```

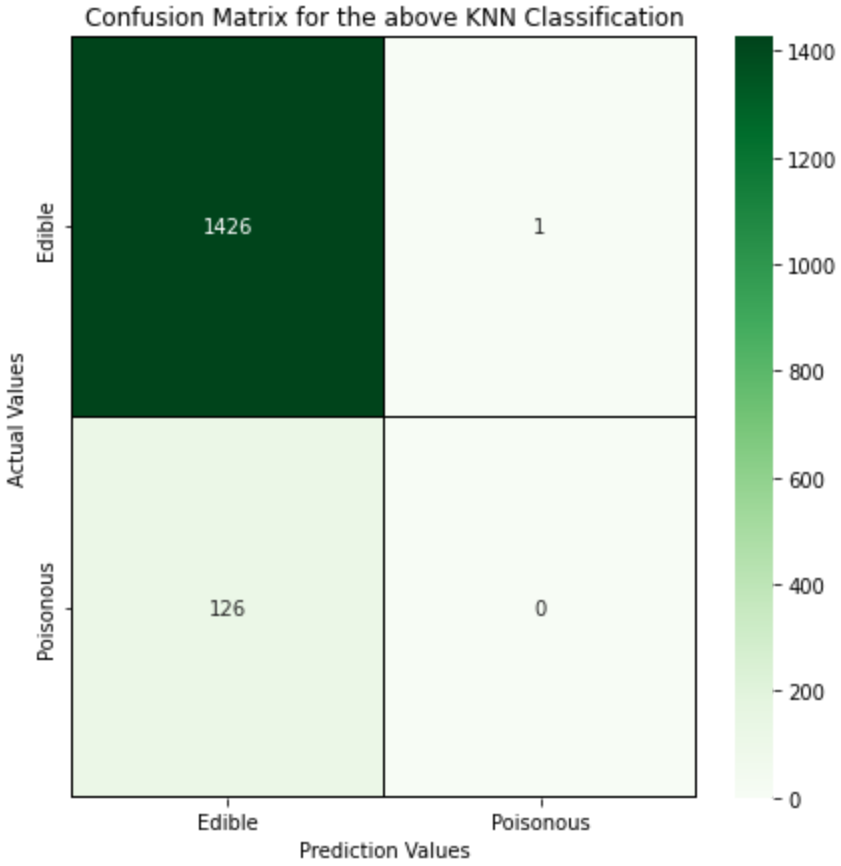
Constructing confusion matrix for the above KNN classification

In [204]:
```
y_knn_pred = k_near.predict(x_test)
print("Classification report from KNN : \n", classification_report(y_test, y_kn
n_pred))
```

```
Classification report from KNN :
               precision    recall  f1-score   support

           0       0.92      1.00      0.96      1427
           1       0.00      0.00      0.00       126

    accuracy                           0.92      1553
   macro avg       0.46      0.50      0.48      1553
weighted avg       0.84      0.92      0.88      1553
```

Constructing confusion matrix for the above KNN Classification

In [205]:
```
con_mat_knn = confusion_matrix(y_test, y_knn_pred)

x_labels = ["Edible", "Poisonous"]
y_labels = ["Edible", "Poisonous"]

fig_size = plt.subplots(figsize =(7,7))
sns.heatmap(con_mat_knn, annot = True, linewidths=0.2, linecolor="black", fmt =
".0f", cmap="Greens", xticklabels=x_labels, yticklabels=y_labels)
plt.xlabel("Prediction Values")
plt.ylabel("Actual Values")
plt.title('Confusion Matrix for the above KNN Classification')
plt.show()
```



Testing the model by predicting the values

In [207]:
```
## Predicting the test values, to test whether the trained model is predicting the values correctly
## IN the below dataframe "0" represents Edible mushroom and the value "1" indicates Poisonous

predict = bank_dt.predict(x_test)

print(predict[:100])
print(y_test[:100].values)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Observation:

From the above printed data arrays, we can see that the predicted and test values are not matching, which indicates that the model is not good at predicting the values.

Designing a Gradient Boosting Regression model

In [208]:
```
## Defining a offset parameter for obtaining the x and y train model

offset = int(x.shape[0] * 0.9)
x_train, y_train = x[:offset], y[:offset]
x_test, y_test = x[offset:], y[offset:]
```

In [209]:
```
## Defining the parameters for plotting train and test datasets

params = {'n_estimators': 500, 'max_depth': 4, 'min_samples_split': 2,
          'learning_rate': 0.01, 'loss': 'ls'}
clf = ensemble.GradientBoostingRegressor(**params)

clf.fit(x_train, y_train)
mse = mean_squared_error(y_test, clf.predict(x_test))
print("MSE: %.4f" % mse)
```

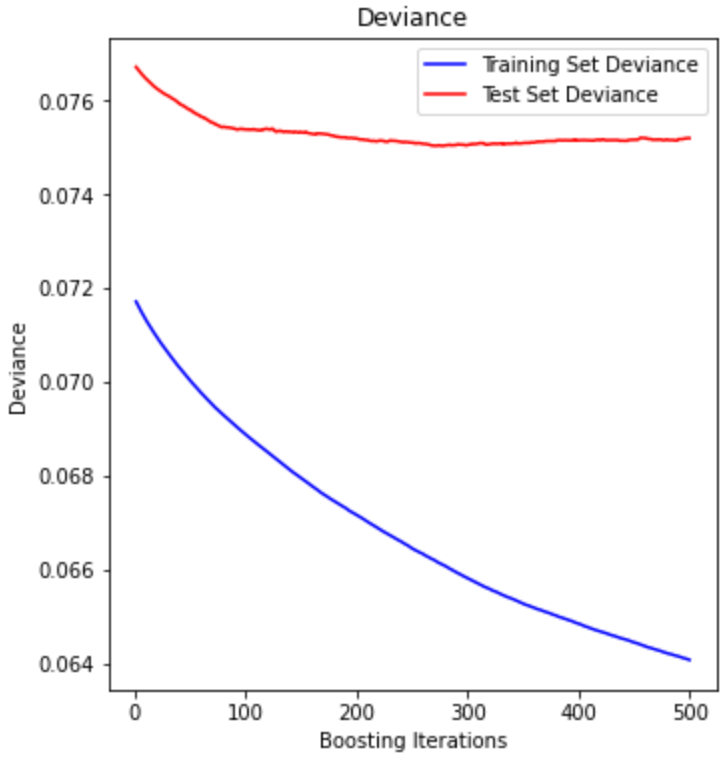
```
MSE: 0.0752
```

Observation:

The mean square error value obtained above, depicts that the regression line for test and train dataset varies by a good margin.

In [210]:
```
# compute and plot test set deviance

test_score = np.zeros((params['n_estimators'],), dtype=np.float64)

for i, y_pred in enumerate(clf.staged_predict(x_test)):
    test_score[i] = clf.loss_(y_test, y_pred)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('Deviance')
plt.plot(np.arange(params['n_estimators']) + 1, clf.train_score_, 'b-',
         label='Training Set Deviance')
plt.plot(np.arange(params['n_estimators']) + 1, test_score, 'r-',
         label='Test Set Deviance')
plt.legend(loc='upper right')
plt.xlabel('Boosting Iterations')
plt.ylabel('Deviance')
```

Out[210]:
```
Text(0, 0.5, 'Deviance')
```

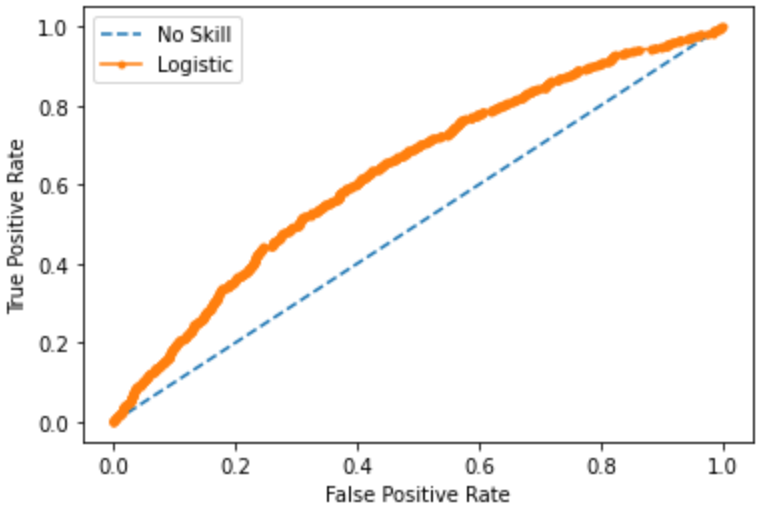


Observation:

We can see from the graph that the designed model is not accurate in predicting the values, as the train and test dataset is varying by a good margin. Indicating that the model is not a good fit.

Creating a metrics for ROC curve

In [211]:
```
# example of a roc curve for a predictive model
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from matplotlib import pyplot
# generate 2 class dataset
x = bank_df.drop(['TARGET'], axis=1)
y = bank_df["TARGET"]
# split into train/test sets
trainx, testx, trainy, testy = train_test_split(x, y, test_size=0.5, random_sta
te=2)
# fit a model
model = LogisticRegression(solver='lbfgs')
model.fit(trainx, trainy)
# predict probabilities
yhat = model.predict_proba(testx)
# retrieve just the probabilities for the positive class
pos_probs = yhat[:, 1]
# plot no skill roc curve
pyplot.plot([0, 1], [0, 1], linestyle='--', label='No Skill')
# calculate roc curve for model
fpr, tpr, _ = roc_curve(testy, pos_probs)
# plot model roc curve
pyplot.plot(fpr, tpr, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```



Observation:

The ROC (Reciever Operating Characteritics) curve, gives a value that indictes the Sensitivity, Specificity of a model designed. As the curve is not too close to the left corner of the graph. It indicates that the model is not a perfect fit. Indicating that it is not the most viable solution.

Calculating the AUC value

In [212]:
```
# Importing the required library
from sklearn.dummy import DummyClassifier
from sklearn.metrics import roc_auc_score
# no skill model, stratified random class predictions
model = DummyClassifier(strategy='stratified')
model.fit(trainx, trainy)
yhat = model.predict_proba(testx)
pos_probs = yhat[:, 1]
# calculate roc auc
roc_auc = roc_auc_score(testy, pos_probs)
print('No Skill ROC AUC %.3f' % roc_auc)
# skilled model
model = LogisticRegression(solver='lbfgs')
model.fit(trainx, trainy)
yhat = model.predict_proba(testx)
pos_probs = yhat[:, 1]
# calculate roc auc
roc_auc = roc_auc_score(testy, pos_probs)
print('Logistic ROC AUC %.3f' % roc_auc)
```

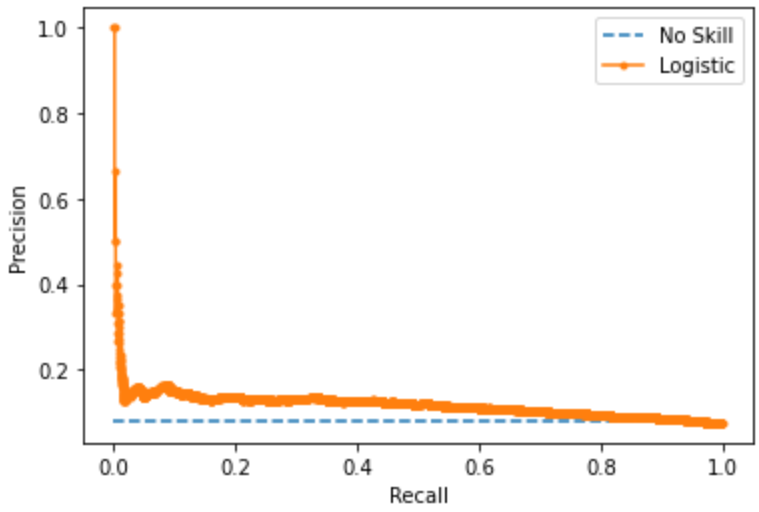
```
No Skill ROC AUC 0.502
Logistic ROC AUC 0.634
```

Observation:

The AUC value for the above ROC curve is "0.678", showing that it is the best model to predict the values.

Creating the metrics for plotting the PR (Precision-Recall) Curve

In [213]:
```
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
from matplotlib import pyplot
# generate 2 class dataset
x = bank_df.drop(['TARGET'], axis=1)
y = bank_df["TARGET"]

trainx, testx, trainy, testy = train_test_split(x, y, test_size=0.5, random_sta
te=2)
# fit a model
model = LogisticRegression(solver='lbfgs')
model.fit(trainx, trainy)
# predict probabilities
yhat = model.predict_proba(testx)
# retrieve just the probabilities for the positive class
pos_probs = yhat[:, 1]
# calculate the no skill line as the proportion of the positive class
no_skill = len(y[y==1]) / len(y)
# plot the no skill precision-recall curve
pyplot.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
# calculate model precision-recall curve
precision, recall, _ = precision_recall_curve(testy, pos_probs)
# plot the model precision-recall curve
pyplot.plot(recall, precision, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```



Obserevation:

precision-recall curve shows tradeoff that happens between precision and recall for different thresholds.In this curve as the precision value increases the recall opportunities diminish. From the above graph we can see that the precision of the values are very less and the recall features are more, again proving that it is not a good fit.

In [214]:
```
# example of a precision-recall auc for a predictive model
# no skill model, stratified random class predictions
model = DummyClassifier(strategy='stratified')
model.fit(trainx, trainy)
yhat = model.predict_proba(testx)
pos_probs = yhat[:, 1]
# calculate the precision-recall auc
precision, recall, _ = precision_recall_curve(testy, pos_probs)
auc_score = auc(recall, precision)
print('No Skill PR AUC: %.3f' % auc_score)
# fit a model
model = LogisticRegression(solver='lbfgs')
model.fit(trainx, trainy)
yhat = model.predict_proba(testx)
pos_probs = yhat[:, 1]
# calculate the precision-recall auc
precision, recall, _ = precision_recall_curve(testy, pos_probs)
auc_score = auc(recall, precision)
print('Logistic PR AUC: %.3f' % auc_score)
```

```
No Skill PR AUC: 0.120
Logistic PR AUC: 0.120
```

Observation:

From the above, AUC values for the PR curve we can see that the no skill AUC values are around "0.119", where the logistic AUC value is "0.167", indicating that it is better than the no skill value. However, the model still proves to be inefficient.

Plotting Lorenz curve

In [215]:
```
X1 = np.append(testy,y_pred)

def gini(arr):
    ## first sort
    sort_arr = arr.copy()
    sort_arr.sort()
    n1 = arr.size
    coef1_ = 2. / n1
    const1_ = (n1 + 1.) / n1
    weighted_sum1 = sum([(i+1)*yi for i, yi in enumerate(sort_arr)])
    return coef1_*weighted_sum1/(sort_arr.sum()) - const1_
```

In [216]:
```
gini(X1)
```

Out[216]:
```
0.8952363337233658
```

OBSERVATION:

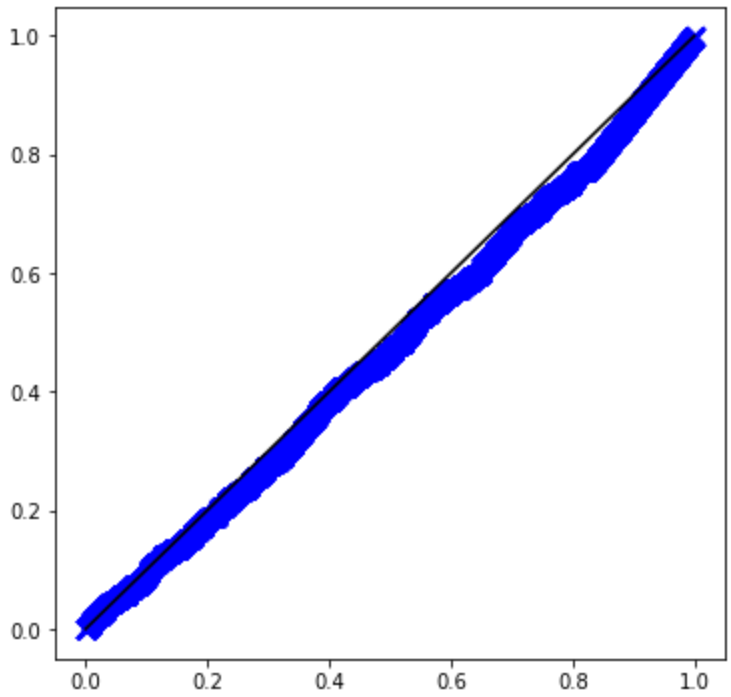
We are using gini index to measure TARGET distribution across the applications.The gi ni co-efficient of the Target is "0.9", which represents perfect equality.

In [217]:
```
X_lorenz1 = X1.cumsum() / X1.sum()
X_lorenz1 = np.insert(X_lorenz1, 0, 0)
X_lorenz1[0], X_lorenz1[-1]
```

Out[217]:
```
(0.0, 1.0000000000000009)
```

In [218]:
```
fig, ax = plt.subplots(figsize=[6,6])
## scatter plot of Lorenz curve
ax.scatter(np.arange(X_lorenz1.size)/(X_lorenz1.size-1), X_lorenz1,
           marker='x', color='blue', s=80)
## line plot of equality
ax.plot([0,1], [0,1], color='k')
```

Out[218]:
```
[<matplotlib.lines.Line2D at 0x7f12ed57ab20>]
```



OBSERVATION:

In the Lorenz curve, we can see that the closeness in the lorenz curve to the equal d istribution line. It has the lesser variation in the distribution.