Pranav Sanjay Vispute

[pranavvispute671@gmail.com (mailto:pranavvispute671@gmail.com)](mailto:pranavvispute671@gmail.com)

Loan Case Study

In [1]:
```
#importing all the required modules here
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import itertools
```

In [2]:
```
#to suppress warnings(taken from previous project)
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
#keeping the display values for better view
pd.set_option('display.max_columns', 75)
pd.set_option('display.max_rows', 75)
pd.set_option('display.width', 500)
pd.set_option('display.expand_frame_repr', False)
```

Part 1: Reading and Analyzing DataSet 1 "application_data.csv"

In [4]:
```
#1. importing the dataset
application_data=pd.read_csv("application_data.csv")
application_data
```

Out[4]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_R
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100004	0	Revolving loans	M	Y	
3	100006	0	Cash loans	F	N	
4	100007	0	Cash loans	M	N	
...	...	...	...	...	...	
307506	456251	0	Cash loans	M	N	
307507	456252	0	Cash loans	F	N	
307508	456253	0	Cash loans	F	N	
307509	456254	1	Cash loans	F	N	
307510	456255	0	Cash loans	F	N	

307511 rows × 122 columns

In [5]:
```
#. getting further knowledge like size, shape, columns, etc. from dataset
application_data.size
```

Out[5]: 37516342

In [6]:
```
application_data.shape
```

Out[6]: (307511, 122)

In [7]:
```
application_data.ndim
```

Out[7]: 2

In [8]:
```
application_data.columns
```

Out[8]:
```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'F
LAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY',
       ...
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21',
'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEE
K', 'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_Y
EAR'], dtype='object', length=122)
```

In [9]:
```
application_data.dtypes
```

Out[9]:
```
SK_ID_CURR                       int64
TARGET                           int64
NAME_CONTRACT_TYPE              object
CODE_GENDER                     object
FLAG_OWN_CAR                    object
                                 ...
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
Length: 122, dtype: object
```

In [10]:
```
application_data.info('all')
```

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

In [11]:
```
application_data.head(50)
```

Out[11]:

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

In [12]:
```
application_data.describe()
```

Out[12]:

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

8 rows × 106 columns

Overall inferences gained:

- the columns like 'days_registration' is supposed to have dates, but has negative values too. This shows that some cleaning is required
- The dataset is huge and has a lot of NaN values. In some cases the whole row is NaN.
- The dataset has 307511 rows and 122 columns

Cleaning and Scraping the data

(following steps from previous case study)

In [13]:
```
#Checking for %age of null values
round(application_data.isnull().mean()*100,4)
```

Out[13]:
```
SK_ID_CURR                     0.0000
TARGET                         0.0000
NAME_CONTRACT_TYPE             0.0000
CODE_GENDER                    0.0000
FLAG_OWN_CAR                   0.0000
                                ...
AMT_REQ_CREDIT_BUREAU_DAY     13.5016
AMT_REQ_CREDIT_BUREAU_WEEK    13.5016
AMT_REQ_CREDIT_BUREAU_MON     13.5016
AMT_REQ_CREDIT_BUREAU_QRT     13.5016
AMT_REQ_CREDIT_BUREAU_YEAR    13.5016
Length: 122, dtype: float64
```

We can see that there is a huge disparity in null percentages. Some columns have little to no null values. and some columns have too many null values. So we can filter out the columns by removing columns with null %age >=50

In [14]:
```
#defining a function to display null percentages
def null_vals(df):
    return round((df.isnull().sum()*100/len(df)).sort_values(ascending = False),2)
```

In [15]:
```
null_50s=null_vals(application_data)[null_vals(application_data)>50]
print(len(null_50s), "rows have more that 50% null values")
```

```
41 rows have more that 50% null values
```

In [16]:
```
null_50s
```

Out[16]:
```
COMMONAREA_MEDI             69.87
COMMONAREA_AVG              69.87
COMMONAREA_MODE             69.87
NONLIVINGAPARTMENTS_MODE    69.43
NONLIVINGAPARTMENTS_AVG     69.43
NONLIVINGAPARTMENTS_MEDI    69.43
FONDKAPREMONT_MODE          68.39
LIVINGAPARTMENTS_MODE       68.35
LIVINGAPARTMENTS_AVG        68.35
LIVINGAPARTMENTS_MEDI       68.35
FLOORSMIN_AVG               67.85
FLOORSMIN_MODE              67.85
FLOORSMIN_MEDI              67.85
YEARS_BUILD_MEDI            66.50
YEARS_BUILD_MODE            66.50
YEARS_BUILD_AVG             66.50
OWN_CAR_AGE                 65.99
LANDAREA_MEDI               59.38
LANDAREA_MODE               59.38
LANDAREA_AVG                59.38
BASEMENTAREA_MEDI           58.52
BASEMENTAREA_AVG            58.52
BASEMENTAREA_MODE           58.52
EXT_SOURCE_1                56.38
NONLIVINGAREA_MODE          55.18
NONLIVINGAREA_AVG           55.18
NONLIVINGAREA_MEDI          55.18
ELEVATORS_MEDI              53.30
ELEVATORS_AVG               53.30
ELEVATORS_MODE              53.30
WALLSMATERIAL_MODE          50.84
APARTMENTS_MEDI             50.75
APARTMENTS_AVG              50.75
APARTMENTS_MODE             50.75
ENTRANCES_MEDI              50.35
ENTRANCES_AVG               50.35
ENTRANCES_MODE              50.35
LIVINGAREA_AVG              50.19
LIVINGAREA_MODE             50.19
LIVINGAREA_MEDI             50.19
HOUSETYPE_MODE              50.18
dtype: float64
```

We can see that there are **41 columns** with more that 50% null values. These columns contain data about the client's residences. like builtup or carpet area. This kind of data wont contribute to the loan process, thus we can drop these columns.

In [17]:
```
application_data.drop(columns=null_50s.index, inplace=True)
application_data.shape
```

Out[17]:
```
(307511, 81)
```

We can see after removing 41 columns with more that 50% null values we are left with, 122-41=81 columns only!

Trying the same for null %>25

In [18]:
```
null_25_cols=null_vals(application_data)[null_vals(application_data)>25]
null_25_cols
```

Out[18]:
```
FLOORSMAX_AVG                     49.76
FLOORSMAX_MODE                    49.76
FLOORSMAX_MEDI                    49.76
YEARS_BEGINEXPLUATATION_AVG       48.78
YEARS_BEGINEXPLUATATION_MODE      48.78
YEARS_BEGINEXPLUATATION_MEDI      48.78
TOTALAREA_MODE                    48.27
EMERGENCYSTATE_MODE               47.40
OCCUPATION_TYPE                   31.35
dtype: float64
```

Here the column **"OCCUPATION_TYPE"** contains data that can **make or break** the loan application, so **we need it** in the dataset. All other columns can be safely removed.

In [19]:
```
null_25_cols.drop('OCCUPATION_TYPE', inplace=True) #removing the column from the list
null_25_cols
```

Out[19]:
```
FLOORSMAX_AVG                     49.76
FLOORSMAX_MODE                    49.76
FLOORSMAX_MEDI                    49.76
YEARS_BEGINEXPLUATATION_AVG       48.78
YEARS_BEGINEXPLUATATION_MODE      48.78
YEARS_BEGINEXPLUATATION_MEDI      48.78
TOTALAREA_MODE                    48.27
EMERGENCYSTATE_MODE               47.40
dtype: float64
```

In [20]:
```
application_data.drop(null_25_cols.index,axis=1, inplace = True)
application_data.shape
```

Out[20]:
```
(307511, 73)
```

Again the same for null%>15

In [21]:
```
null_15_cols=null_vals(application_data)[null_vals(application_data)>15]
null_15_cols
```

Out[21]:
```
OCCUPATION_TYPE     31.35
EXT_SOURCE_3        19.83
dtype: float64
```

We can see that like **"occupation_data"** was relevent, here **"ext_source"** is also important to the process, so we **exclude these two columns from dropping**

In [22]:
```
null_vals(application_data).head(80)
```

Out[22]:
```
OCCUPATION_TYPE                31.35
EXT_SOURCE_3                   19.83
AMT_REQ_CREDIT_BUREAU_YEAR     13.50
AMT_REQ_CREDIT_BUREAU_QRT      13.50
AMT_REQ_CREDIT_BUREAU_MON      13.50
AMT_REQ_CREDIT_BUREAU_WEEK     13.50
AMT_REQ_CREDIT_BUREAU_DAY      13.50
AMT_REQ_CREDIT_BUREAU_HOUR     13.50
NAME_TYPE_SUITE                 0.42
OBS_30_CNT_SOCIAL_CIRCLE        0.33
DEF_30_CNT_SOCIAL_CIRCLE        0.33
OBS_60_CNT_SOCIAL_CIRCLE        0.33
DEF_60_CNT_SOCIAL_CIRCLE        0.33
EXT_SOURCE_2                    0.21
AMT_GOODS_PRICE                 0.09
AMT_ANNUITY                     0.00
CNT_FAM_MEMBERS                 0.00
DAYS_LAST_PHONE_CHANGE          0.00
FLAG_DOCUMENT_17                0.00
FLAG_DOCUMENT_18                0.00
FLAG_DOCUMENT_21                0.00
FLAG_DOCUMENT_20                0.00
FLAG_DOCUMENT_19                0.00
FLAG_DOCUMENT_2                 0.00
FLAG_DOCUMENT_3                 0.00
FLAG_DOCUMENT_4                 0.00
FLAG_DOCUMENT_5                 0.00
FLAG_DOCUMENT_16                0.00
FLAG_DOCUMENT_6                 0.00
FLAG_DOCUMENT_7                 0.00
FLAG_DOCUMENT_8                 0.00
FLAG_DOCUMENT_9                 0.00
FLAG_DOCUMENT_10                0.00
FLAG_DOCUMENT_11                0.00
ORGANIZATION_TYPE               0.00
FLAG_DOCUMENT_13                0.00
FLAG_DOCUMENT_14                0.00
FLAG_DOCUMENT_15                0.00
FLAG_DOCUMENT_12                0.00
SK_ID_CURR                      0.00
LIVE_CITY_NOT_WORK_CITY         0.00
DAYS_REGISTRATION               0.00
NAME_CONTRACT_TYPE              0.00
CODE_GENDER                     0.00
FLAG_OWN_CAR                    0.00
FLAG_OWN_REALTY                 0.00
CNT_CHILDREN                    0.00
AMT_INCOME_TOTAL                0.00
AMT_CREDIT                      0.00
NAME_INCOME_TYPE                0.00
NAME_EDUCATION_TYPE             0.00
NAME_FAMILY_STATUS              0.00
NAME_HOUSING_TYPE               0.00
REGION_POPULATION_RELATIVE      0.00
DAYS_BIRTH                      0.00
DAYS_EMPLOYED                   0.00
DAYS_ID_PUBLISH                 0.00
REG_CITY_NOT_WORK_CITY          0.00
FLAG_MOBIL                      0.00
FLAG_EMP_PHONE                  0.00
FLAG_WORK_PHONE                 0.00
FLAG_CONT_MOBILE                0.00
FLAG_PHONE                      0.00
```

```
FLAG_EMAIL                     0.00
REGION_RATING_CLIENT           0.00
REGION_RATING_CLIENT_W_CITY    0.00
WEEKDAY_APPR_PROCESS_START     0.00
HOUR_APPR_PROCESS_START        0.00
REG_REGION_NOT_LIVE_REGION     0.00
REG_REGION_NOT_WORK_REGION     0.00
LIVE_REGION_NOT_WORK_REGION    0.00
TARGET                         0.00
REG_CITY_NOT_LIVE_CITY         0.00
dtype: float64
```

We can see from the column_description.csv that the columns EXT_SOURCE_2 and EXT_SOURCE_3 have normalized values. So we can safely drop them out as there is not linear correlation. And we know that **CORRELATION DOESN'T CAUSE CAUSATION**

In [23]:
```
#dropping the 2 columns
application_data.drop(['EXT_SOURCE_2','EXT_SOURCE_3'], axis=1,inplace=True)
application_data
```

Out[23]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_R
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100004	0	Revolving loans	M	Y	
3	100006	0	Cash loans	F	N	
4	100007	0	Cash loans	M	N	
...	...	...	...	...	...	
307506	456251	0	Cash loans	M	N	
307507	456252	0	Cash loans	F	N	
307508	456253	0	Cash loans	F	N	
307509	456254	1	Cash loans	F	N	
307510	456255	0	Cash loans	F	N	

307511 rows × 71 columns

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

Out[24]: (307511, 71)

Next thing to do is fill in values in important columns where null values exist.

In [25]:
```
null_vals(application_data).head(20)
```

Out[25]:
```
OCCUPATION_TYPE               31.35
AMT_REQ_CREDIT_BUREAU_YEAR    13.50
AMT_REQ_CREDIT_BUREAU_QRT     13.50
AMT_REQ_CREDIT_BUREAU_MON     13.50
AMT_REQ_CREDIT_BUREAU_WEEK    13.50
AMT_REQ_CREDIT_BUREAU_DAY     13.50
AMT_REQ_CREDIT_BUREAU_HOUR    13.50
NAME_TYPE_SUITE                0.42
DEF_30_CNT_SOCIAL_CIRCLE       0.33
OBS_60_CNT_SOCIAL_CIRCLE       0.33
DEF_60_CNT_SOCIAL_CIRCLE       0.33
OBS_30_CNT_SOCIAL_CIRCLE       0.33
AMT_GOODS_PRICE                0.09
AMT_ANNUITY                    0.00
CNT_FAM_MEMBERS                0.00
DAYS_LAST_PHONE_CHANGE         0.00
FLAG_DOCUMENT_8                0.00
FLAG_DOCUMENT_2                0.00
FLAG_DOCUMENT_3                0.00
FLAG_DOCUMENT_4                0.00
dtype: float64
```

Looking at the above data, we can easily notice that the columns OCCUPATION_TYPE, AMT_REQ_CREDIT_BUREAU_YEAR, AMT_REQ_CREDIT_BUREAU_QRT, AMT_REQ_CREDIT_BUREAU_MON, AMT_REQ_CREDIT_BUREAU_WEEK, AMT_REQ_CREDIT_BUREAU_DAY, AMT_REQ_CREDIT_BUREAU_HOUR, have considerable and similiar null percentages. Thus we need to fill in values for only these 7 columns. Other columns can be ignored.

1. Starting with column "OCCUPATION_TYPE"

In [26]:
```
application_data['OCCUPATION_TYPE'].value_counts(normalize=True)*100
```

Out[26]:
```
Laborers                 26.139636
Sales staff              15.205570
Core staff               13.058924
Managers                 10.122679
Drivers                   8.811576
High skill tech staff     5.390299
Accountants               4.648067
Medicine staff            4.043672
Security staff            3.183498
Cooking staff             2.816408
Cleaning staff            2.203960
Private service staff     1.256158
Low-skill Laborers        0.991379
Waiters/barmen staff      0.638499
Secretaries               0.618132
Realty agents             0.355722
HR staff                  0.266673
IT staff                  0.249147
Name: OCCUPATION_TYPE, dtype: float64
```

Since this column misses 31.35% of values, and the value type is not one like we can guess or calculate(being a profession), we can simply put in "Not_Known" in place of null

In [27]:
```
application_data["OCCUPATION_TYPE"]=application_data["OCCUPATION_TYPE"].fillna("Not_Know
```

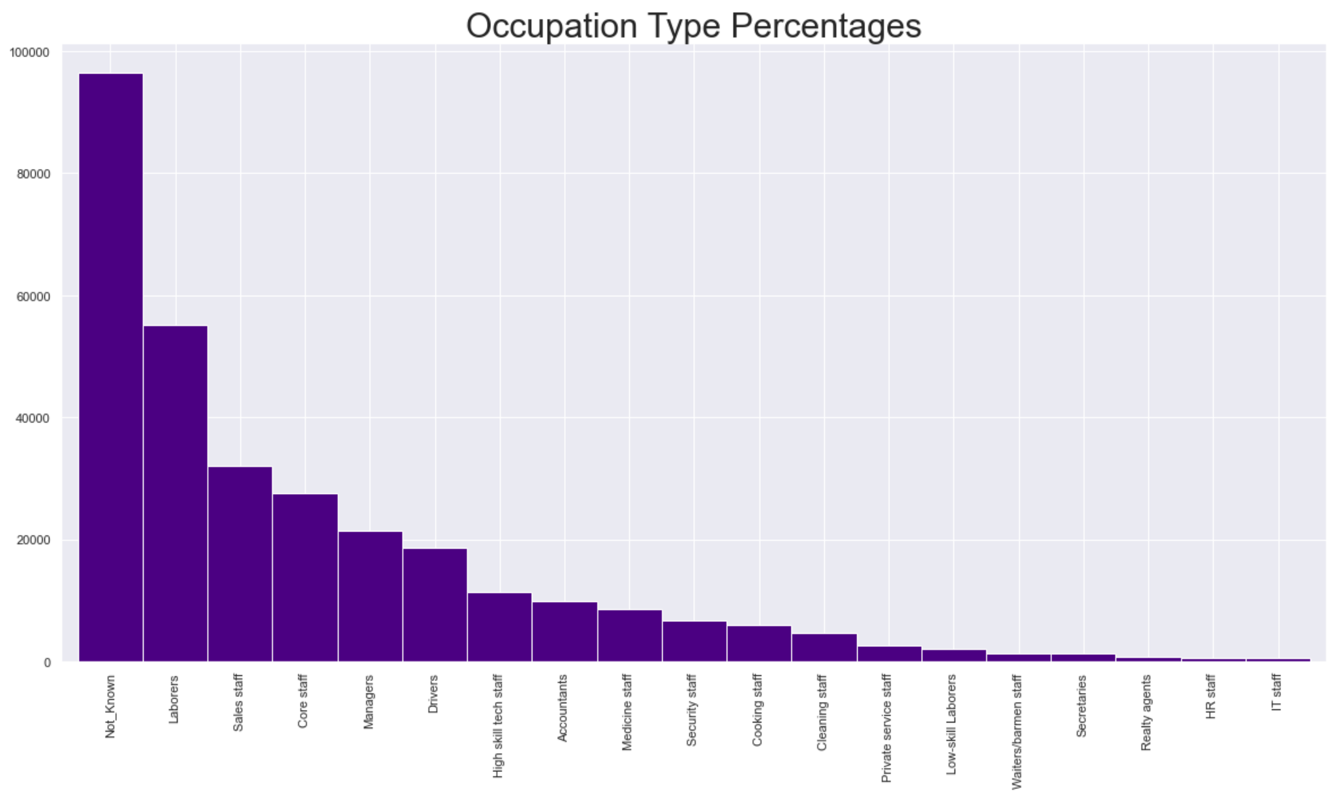
In [28]:
```
application_data["OCCUPATION_TYPE"].isnull().sum()
```

Out[28]: 0

Thus we have successfully replaced the occupation type of "missing" to "Not Known".

Now for the sake of curiosity, let's look at how many values are "Not_Known"

In [29]:
```
sns.set(style="darkgrid")
plt.figure(figsize=[20,10])
(application_data["OCCUPATION_TYPE"].value_counts()).plot.bar(color="indigo",width=1.0)
plt.title("Occupation Type Percentages", fontdict={"fontsize":30})
plt.show()
```



After the OCCUPATION_TYPE column, next are columns that represent number of requests done to a customer which is always a whole number, meaning we can't replace it with mean, as it would give a decimal number. We can use median here.

In [30]:
```
request_cols=["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"]
application_data.fillna(application_data[request_cols].median(), inplace=True)
```

In [31]:
```
null_vals(application_data).head(20)
```

Out[31]:
```
NAME_TYPE_SUITE              0.42
OBS_30_CNT_SOCIAL_CIRCLE     0.33
DEF_30_CNT_SOCIAL_CIRCLE     0.33
OBS_60_CNT_SOCIAL_CIRCLE     0.33
DEF_60_CNT_SOCIAL_CIRCLE     0.33
AMT_GOODS_PRICE              0.09
AMT_ANNUITY                  0.00
CNT_FAM_MEMBERS              0.00
DAYS_LAST_PHONE_CHANGE       0.00
FLAG_DOCUMENT_5              0.00
FLAG_DOCUMENT_9              0.00
FLAG_DOCUMENT_8              0.00
FLAG_DOCUMENT_7              0.00
FLAG_DOCUMENT_6              0.00
SK_ID_CURR                   0.00
FLAG_DOCUMENT_4              0.00
FLAG_DOCUMENT_11             0.00
FLAG_DOCUMENT_3              0.00
FLAG_DOCUMENT_2              0.00
FLAG_DOCUMENT_10             0.00
dtype: float64
```

As these columns have very less null %, we can choose to ignore the value addition

Now we can start optimizing present values in the dataset.

As stated before, columns like DAYS_BIRTH, DAYS_EMPLOYED, DAYS_REGISTRATION, DAYS_ID_PUBLISH, DAYS_LAST_PHONE_CHANGE, have negative values, thus we need to repair them.

In [32]:
```
day_cols=["DAYS_BIRTH", "DAYS_EMPLOYED", "DAYS_REGISTRATION", "DAYS_ID_PUBLISH", "DAYS_L
application_data[day_cols].describe()
```

Out[32]:

	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_REGISTRATION	DAYS_ID_PUBLISH	DAYS_LAST_PHONE_CH
count	307511.000000	307511.000000	307511.000000	307511.000000	307510.0
mean	-16036.995067	63815.045904	-4986.120328	-2994.202373	-962.8
std	4363.988632	141275.766519	3522.886321	1509.450419	826.8
min	-25229.000000	-17912.000000	-24672.000000	-7197.000000	-4292.0
25%	-19682.000000	-2760.000000	-7479.500000	-4299.000000	-1570.0
50%	-15750.000000	-1213.000000	-4504.000000	-3254.000000	-757.0
75%	-12413.000000	-289.000000	-2010.000000	-1720.000000	-274.0
max	-7489.000000	365243.000000	0.000000	0.000000	0.0

Now we can use the abs() function to get the absolute values of negatives present in the columns

In [33]:
```
application_data[day_cols]=abs(application_data[day_cols])
```

In [34]:
```
application_data[day_cols].describe()
```

Out[34]:

	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_REGISTRATION	DAYS_ID_PUBLISH	DAYS_LAST_PHONE_CH
count	307511.000000	307511.000000	307511.000000	307511.000000	307510.0
mean	16036.995067	67724.742149	4986.120328	2994.202373	962.8
std	4363.988632	139443.751806	3522.886321	1509.450419	826.8
min	7489.000000	0.000000	0.000000	0.000000	0.0
25%	12413.000000	933.000000	2010.000000	1720.000000	274.0
50%	15750.000000	2219.000000	4504.000000	3254.000000	757.0
75%	19682.000000	5707.000000	7479.500000	4299.000000	1570.0
max	25229.000000	365243.000000	24672.000000	7197.000000	4292.0

As we are on the topic of dates, we need to also convert some values in days columns to years, as it makes more sense. like DAYS_BIRTH, DAYS_EMPLOYED can be converted to YEARS_BIRTH, YEARS_EMPLOYED

In [35]:
```
application_data['AGE']=application_data['DAYS_BIRTH']/365
bins = [0,20,25,30,35,40,45,50,55,60,100]
age_groups = ["0-20","20-25","25-30","30-35","35-40","40-45","45-50","50-55","55-60","6(

application_data["AGE_GROUPS"] = pd.cut(application_data["AGE"], bins=bins, labels=age_g
```

In [36]:
```
#Repeating the same procedure for DAYS_EMPLOYED

application_data["YEARS_EMPLOYED"] = application_data["DAYS_EMPLOYED"]/365
bins = [0,5,10,15,20,25,30,50]
year_slots = ["0-5","5-10","10-15","15-20","20-25","25-30","30 Above"]
application_data["EMPLOYEMENT_YEARS"] = pd.cut(application_data["YEARS_EMPLOYED"], bins=
```

In [37]:
```
application_data["EMPLOYEMENT_YEARS"].value_counts(normalize=True)*100
```

Out[37]:
```
0-5         54.061911
5-10        25.729074
10-15       10.926289
15-20        4.302854
20-25        2.476054
25-30        1.311996
30 Above     1.191822
Name: EMPLOYEMENT_YEARS, dtype: float64
```

columns like AMT_INCOME_TOTAL, AMT_CREDIT, AMT_GOODS_PRICE, have too high values, thus we can smooth out those values

In [38]:
```
#Converting in order of Lacs
application_data['AMT_CREDIT']=application_data['AMT_CREDIT']/100000
```

In [39]:
```
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-10I
```

In [40]:
```
application_data['AMT_CREDIT_RANGE']=pd.cut(application_data['AMT_CREDIT'],bins=bins,lal
```

In [41]:
```
round((application_data["AMT_CREDIT_RANGE"].value_counts(normalize = True)*100),2)
```

Out[41]:
```
2L-3L        17.82
10L Above    16.25
5L-6L        11.13
4L-5L        10.42
1L-2L         9.80
3L-4L         8.56
6L-7L         7.82
8L-9L         7.09
7L-8L         6.24
9L-10L        2.90
0-1L          1.95
Name: AMT_CREDIT_RANGE, dtype: float64
```

In [42]:
```
#Converting to order of Lacs
application_data['AMT_INCOME_TOTAL']=application_data['AMT_INCOME_TOTAL']/100000
```

In [43]:
```
bins = [0,1,2,3,4,5,6,7,8,9,10,11]
slot = ['0-1L','1L-2L', '2L-3L','3L-4L','4L-5L','5L-6L','6L-7L','7L-8L','8L-9L','9L-10L
```

In [44]:
```
application_data['AMT_INCOME_RANGE']=pd.cut(application_data['AMT_INCOME_TOTAL'],bins,la
```

In [45]:
```
round((application_data["AMT_INCOME_RANGE"].value_counts(normalize = True)*100),2)
```

Out[45]:
```
1L-2L        50.73
2L-3L        21.21
0-1L         20.73
3L-4L         4.78
4L-5L         1.74
5L-6L         0.36
6L-7L         0.28
8L-9L         0.10
7L-8L         0.05
9L-10L        0.01
Above 10L     0.01
Name: AMT_INCOME_RANGE, dtype: float64
```

In [46]:
```
#Converting in order of Lacs
application_data['AMT_GOODS_PRICE']=application_data['AMT_GOODS_PRICE']/100000
```

In [47]:
```
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-10I
```

In [48]:
```
application_data['AMT_GOODS_PRICE_RANGE']=pd.cut(application_data['AMT_GOODS_PRICE'],bir
```

In [49]:
```
round((application_data["AMT_GOODS_PRICE_RANGE"].value_counts(normalize = True)*100),2)
```

Out[49]:
```
2L-3L         20.43
4L-5L         18.54
6L-7L         13.03
Above 10L     11.11
1L-2L         10.73
8L-9L          6.99
3L-4L          6.91
5L-6L          4.27
0-1L           2.83
7L-8L          2.64
9L-10L         2.53
Name: AMT_GOODS_PRICE_RANGE, dtype: float64
```

Looking for OutLiers

In [50]:
```
application_data.describe()
```

Out[50]:

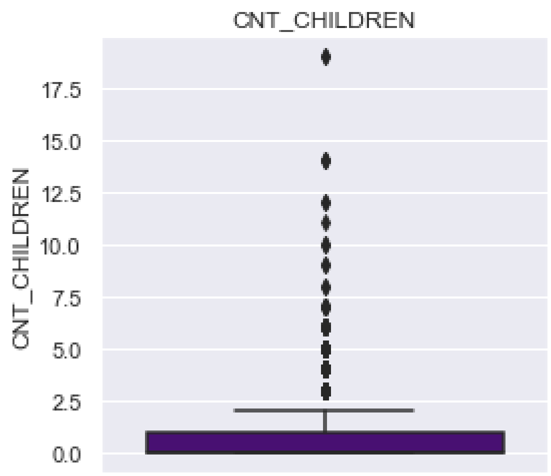
	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY
count	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000	307499.000000
mean	278180.518577	0.080729	0.417052	1.687979	5.990260	27108.573909
std	102790.175348	0.272419	0.722121	2.371231	4.024908	14493.737315
min	100002.000000	0.000000	0.000000	0.256500	0.450000	1615.500000
25%	189145.500000	0.000000	0.000000	1.125000	2.700000	16524.000000
50%	278202.000000	0.000000	0.000000	1.471500	5.135310	24903.000000
75%	367142.500000	0.000000	1.000000	2.025000	8.086500	34596.000000
max	456255.000000	1.000000	19.000000	1170.000000	40.500000	258025.500000

In the above describe output we can see that many columns have huge difference in max value and 75 percentile, which tells us that there are bound to be outliers in that range. Those columns are...

In [51]:
```
outliers=["CNT_CHILDREN","DAYS_REGISTRATION", "AMT_ANNUITY",
          "DAYS_BIRTH", "AMT_GOODS_PRICE","AMT_INCOME_TOTAL","DAYS_EMPLOYED","AMT_(
```

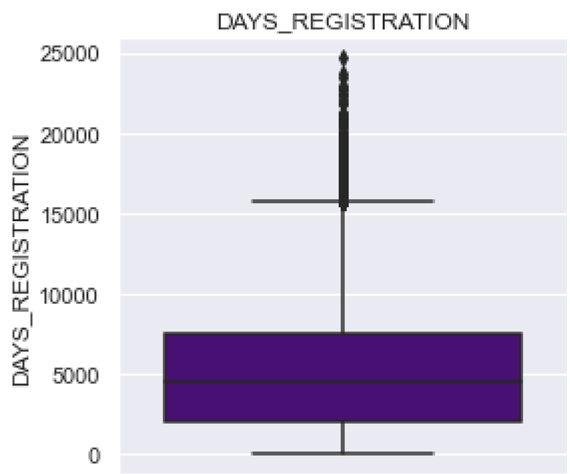
In [52]:
```
plt.figure(figsize=[4,4])
sns.boxplot(y=application_data["CNT_CHILDREN"], orient ="h", color="indigo")
plt.title("CNT_CHILDREN")
```

Out[52]: Text(0.5, 1.0, 'CNT_CHILDREN')



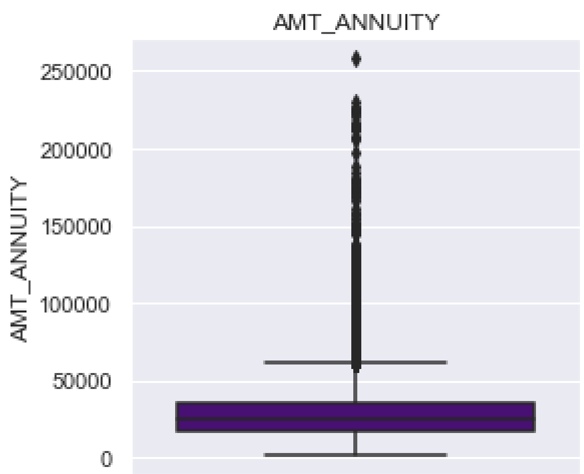
In [53]:
```
plt.figure(figsize=[4,4])
sns.boxplot(y=application_data["DAYS_REGISTRATION"], orient ="h", color="indigo")
plt.title("DAYS_REGISTRATION")
```

Out[53]: Text(0.5, 1.0, 'DAYS_REGISTRATION')



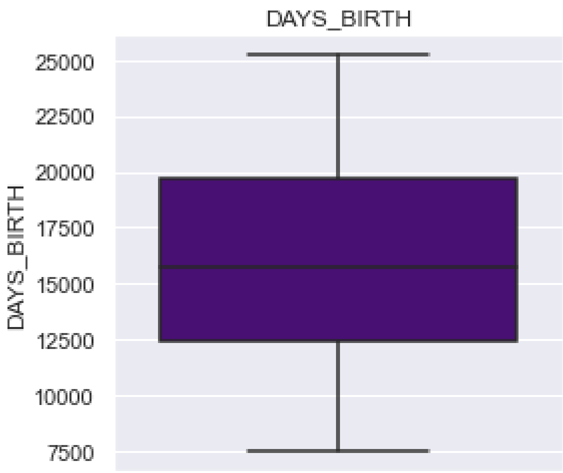
In [54]:
```
plt.figure(figsize=[4,4])
sns.boxplot(y=application_data["AMT_ANNUITY"], orient ="h", color="indigo")
plt.title("AMT_ANNUITY")
```

Out[54]: Text(0.5, 1.0, 'AMT_ANNUITY')



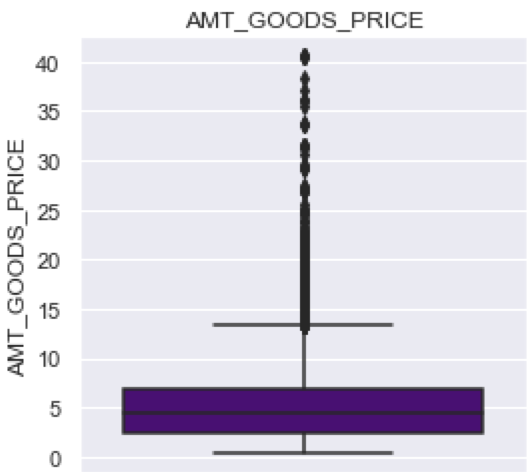
In [55]:
```
plt.figure(figsize=[4,4])
sns.boxplot(y=application_data["DAYS_BIRTH"], orient ="h", color="indigo")
plt.title("DAYS_BIRTH")
```

Out[55]: `Text(0.5, 1.0, 'DAYS_BIRTH')`



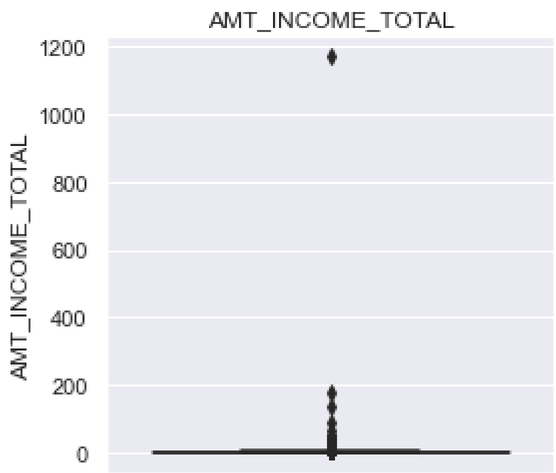
In [56]:
```
plt.figure(figsize=[4,4])
sns.boxplot(y=application_data["AMT_GOODS_PRICE"], orient ="h", color="indigo")
plt.title("AMT_GOODS_PRICE")
```

Out[56]: `Text(0.5, 1.0, 'AMT_GOODS_PRICE')`



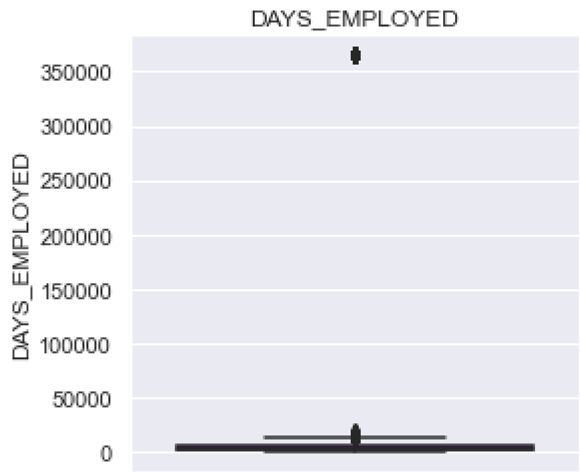
In [57]:
```
plt.figure(figsize=[4,4])
sns.boxplot(y=application_data["AMT_INCOME_TOTAL"], orient ="h", color="indigo")
plt.title("AMT_INCOME_TOTAL")
```

Out[57]: Text(0.5, 1.0, 'AMT_INCOME_TOTAL')



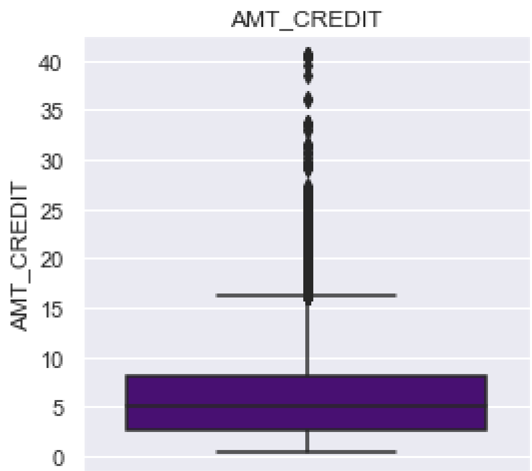
In [58]:
```
plt.figure(figsize=[4,4])
sns.boxplot(y=application_data["DAYS_EMPLOYED"], orient ="h", color="indigo")
plt.title("DAYS_EMPLOYED")
```

Out[58]: Text(0.5, 1.0, 'DAYS_EMPLOYED')



In [59]:
```
plt.figure(figsize=[4,4])
sns.boxplot(y=application_data["AMT_CREDIT"], orient ="h", color="indigo")
plt.title("AMT_CREDIT")
```

Out[59]: Text(0.5, 1.0, 'AMT_CREDIT')



Inferences

AMT_INCOME_TOTAL has a lot of outliers means some clients have very high income compared to others

AMT_ANNUITY, AMT_CREDIT, AMT_GOODS_PRICE, CNT_CHILDREN have some outliers

DAYS_EMPLOYED has outliers that outlie too much

DAYS_BIRTH is outlier free!

In [60]:
```
application_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 78 columns):
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   SK_ID_CURR                   307511 non-null  int64  
 1   TARGET                       307511 non-null  int64  
 2   NAME_CONTRACT_TYPE           307511 non-null  object 
 3   CODE_GENDER                  307511 non-null  object 
 4   FLAG_OWN_CAR                 307511 non-null  object 
 5   FLAG_OWN_REALTY              307511 non-null  object 
 6   CNT_CHILDREN                 307511 non-null  int64  
 7   AMT_INCOME_TOTAL             307511 non-null  float64
 8   AMT_CREDIT                   307511 non-null  float64
 9   AMT_ANNUITY                  307499 non-null  float64
 10  AMT_GOODS_PRICE              307233 non-null  float64
 11  NAME_TYPE_SUITE              306219 non-null  object 
 12  NAME_INCOME_TYPE             307511 non-null  object 
 13  NAME_EDUCATION_TYPE          307511 non-null  object 
 14  NAME_FAMILY_STATUS           307511 non-null  object 
 15  NAME_HOUSING_TYPE            307511 non-null  object 
 16  REGION_POPULATION_RELATIVE   307511 non-null  float64
 17  DAYS_BIRTH                   307511 non-null  int64  
 18  DAYS_EMPLOYED                307511 non-null  int64  
 19  DAYS_REGISTRATION            307511 non-null  float64
 20  DAYS_ID_PUBLISH              307511 non-null  int64  
 21  FLAG_MOBIL                   307511 non-null  int64  
 22  FLAG_EMP_PHONE               307511 non-null  int64  
 23  FLAG_WORK_PHONE              307511 non-null  int64  
 24  FLAG_CONT_MOBILE             307511 non-null  int64  
 25  FLAG_PHONE                   307511 non-null  int64  
 26  FLAG_EMAIL                   307511 non-null  int64  
 27  OCCUPATION_TYPE              307511 non-null  object 
 28  CNT_FAM_MEMBERS              307509 non-null  float64
 29  REGION_RATING_CLIENT         307511 non-null  int64  
 30  REGION_RATING_CLIENT_W_CITY  307511 non-null  int64  
 31  WEEKDAY_APPR_PROCESS_START   307511 non-null  object 
 32  HOUR_APPR_PROCESS_START      307511 non-null  int64  
 33  REG_REGION_NOT_LIVE_REGION   307511 non-null  int64  
 34  REG_REGION_NOT_WORK_REGION   307511 non-null  int64  
 35  LIVE_REGION_NOT_WORK_REGION  307511 non-null  int64  
 36  REG_CITY_NOT_LIVE_CITY       307511 non-null  int64  
 37  REG_CITY_NOT_WORK_CITY       307511 non-null  int64  
 38  LIVE_CITY_NOT_WORK_CITY      307511 non-null  int64  
 39  ORGANIZATION_TYPE            307511 non-null  object 
 40  OBS_30_CNT_SOCIAL_CIRCLE     306490 non-null  float64
 41  DEF_30_CNT_SOCIAL_CIRCLE     306490 non-null  float64
 42  OBS_60_CNT_SOCIAL_CIRCLE     306490 non-null  float64
 43  DEF_60_CNT_SOCIAL_CIRCLE     306490 non-null  float64
 44  DAYS_LAST_PHONE_CHANGE       307510 non-null  float64
 45  FLAG_DOCUMENT_2              307511 non-null  int64  
 46  FLAG_DOCUMENT_3              307511 non-null  int64  
 47  FLAG_DOCUMENT_4              307511 non-null  int64  
 48  FLAG_DOCUMENT_5              307511 non-null  int64  
 49  FLAG_DOCUMENT_6              307511 non-null  int64  
 50  FLAG_DOCUMENT_7              307511 non-null  int64  
 51  FLAG_DOCUMENT_8              307511 non-null  int64  
 52  FLAG_DOCUMENT_9              307511 non-null  int64  
 53  FLAG_DOCUMENT_10             307511 non-null  int64  
 54  FLAG_DOCUMENT_11             307511 non-null  int64  
 55  FLAG_DOCUMENT_12             307511 non-null  int64  
 56  FLAG_DOCUMENT_13             307511 non-null  int64  
 57  FLAG_DOCUMENT_14             307511 non-null  int64  
```

```
 58  FLAG_DOCUMENT_15             307511 non-null  int64
 59  FLAG_DOCUMENT_16             307511 non-null  int64
 60  FLAG_DOCUMENT_17             307511 non-null  int64
 61  FLAG_DOCUMENT_18             307511 non-null  int64
 62  FLAG_DOCUMENT_19             307511 non-null  int64
 63  FLAG_DOCUMENT_20             307511 non-null  int64
 64  FLAG_DOCUMENT_21             307511 non-null  int64
 65  AMT_REQ_CREDIT_BUREAU_HOUR   307511 non-null  float64
 66  AMT_REQ_CREDIT_BUREAU_DAY    307511 non-null  float64
 67  AMT_REQ_CREDIT_BUREAU_WEEK   307511 non-null  float64
 68  AMT_REQ_CREDIT_BUREAU_MON    307511 non-null  float64
 69  AMT_REQ_CREDIT_BUREAU_QRT    307511 non-null  float64
 70  AMT_REQ_CREDIT_BUREAU_YEAR   307511 non-null  float64
 71  AGE                          307511 non-null  float64
 72  AGE_GROUPS                   307511 non-null  category
 73  YEARS_EMPLOYED               307511 non-null  float64
 74  EMPLOYEMENT_YEARS            252135 non-null  category
 75  AMT_CREDIT_RANGE             307511 non-null  category
 76  AMT_INCOME_RANGE             307279 non-null  category
 77  AMT_GOODS_PRICE_RANGE        307233 non-null  category
dtypes: category(5), float64(20), int64(41), object(12)
memory usage: 172.7+ MB
```

In [61]:

```
cat_cols=['NAME_CONTRACT_TYPE','CODE_GENDER','NAME_TYPE_SUITE','NAME_INCOME_TYPE','NAME_
                       'NAME_FAMILY_STATUS','NAME_HOUSING_TYPE','OCCUPATION_TYPE','WEEKD
                       'ORGANIZATION_TYPE','FLAG_OWN_REALTY','LIVE_CITY_NOT_WORK_CITY',
                       'REG_CITY_NOT_LIVE_CITY','REG_CITY_NOT_WORK_CITY','REG_REGION_NOT
                       'LIVE_REGION_NOT_WORK_REGION','REGION_RATING_CLIENT','WEEKDAY_APP
                       'REGION_RATING_CLIENT_W_CITY','CNT_CHILDREN','CNT_FAM_MEMBERS']
for i in cat_cols:
    application_data[i]=pd.Categorical(application_data[i])
```

In [62]:

```
len(cat_cols)
```

Out[62]:

```
21
```

In [63]:

```
application_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 78 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   SK_ID_CURR                   307511 non-null  int64
 1   TARGET                       307511 non-null  int64
 2   NAME_CONTRACT_TYPE           307511 non-null  category
 3   CODE_GENDER                  307511 non-null  category
 4   FLAG_OWN_CAR                 307511 non-null  object
 5   FLAG_OWN_REALTY              307511 non-null  category
 6   CNT_CHILDREN                 307511 non-null  category
 7   AMT_INCOME_TOTAL             307511 non-null  float64
 8   AMT_CREDIT                   307511 non-null  float64
 9   AMT_ANNUITY                  307499 non-null  float64
 10  AMT_GOODS_PRICE              307233 non-null  float64
 11  NAME_TYPE_SUITE              306219 non-null  category
 12  NAME_INCOME_TYPE             307511 non-null  category
 13  NAME_EDUCATION_TYPE          307511 non-null  category
 14  NAME_FAMILY_STATUS           307511 non-null  category
 15  NAME_HOUSING_TYPE            307511 non-null  category
 16  REGION_POPULATION_RELATIVE   307511 non-null  float64
 17  DAYS_BIRTH                   307511 non-null  int64
 18  DAYS_EMPLOYED                307511 non-null  int64
 19  DAYS_REGISTRATION            307511 non-null  float64
 20  DAYS_ID_PUBLISH              307511 non-null  int64
 21  FLAG_MOBIL                   307511 non-null  int64
 22  FLAG_EMP_PHONE               307511 non-null  int64
 23  FLAG_WORK_PHONE              307511 non-null  int64
 24  FLAG_CONT_MOBILE             307511 non-null  int64
 25  FLAG_PHONE                   307511 non-null  int64
 26  FLAG_EMAIL                   307511 non-null  int64
 27  OCCUPATION_TYPE              307511 non-null  category
 28  CNT_FAM_MEMBERS              307509 non-null  category
 29  REGION_RATING_CLIENT         307511 non-null  category
 30  REGION_RATING_CLIENT_W_CITY  307511 non-null  category
 31  WEEKDAY_APPR_PROCESS_START   307511 non-null  category
 32  HOUR_APPR_PROCESS_START      307511 non-null  int64
 33  REG_REGION_NOT_LIVE_REGION   307511 non-null  int64
 34  REG_REGION_NOT_WORK_REGION   307511 non-null  category
 35  LIVE_REGION_NOT_WORK_REGION  307511 non-null  category
 36  REG_CITY_NOT_LIVE_CITY       307511 non-null  category
 37  REG_CITY_NOT_WORK_CITY       307511 non-null  category
 38  LIVE_CITY_NOT_WORK_CITY      307511 non-null  category
 39  ORGANIZATION_TYPE            307511 non-null  category
 40  OBS_30_CNT_SOCIAL_CIRCLE     306490 non-null  float64
 41  DEF_30_CNT_SOCIAL_CIRCLE     306490 non-null  float64
 42  OBS_60_CNT_SOCIAL_CIRCLE     306490 non-null  float64
 43  DEF_60_CNT_SOCIAL_CIRCLE     306490 non-null  float64
 44  DAYS_LAST_PHONE_CHANGE       307510 non-null  float64
 45  FLAG_DOCUMENT_2              307511 non-null  int64
 46  FLAG_DOCUMENT_3              307511 non-null  int64
 47  FLAG_DOCUMENT_4              307511 non-null  int64
 48  FLAG_DOCUMENT_5              307511 non-null  int64
 49  FLAG_DOCUMENT_6              307511 non-null  int64
 50  FLAG_DOCUMENT_7              307511 non-null  int64
 51  FLAG_DOCUMENT_8              307511 non-null  int64
 52  FLAG_DOCUMENT_9              307511 non-null  int64
 53  FLAG_DOCUMENT_10             307511 non-null  int64
 54  FLAG_DOCUMENT_11             307511 non-null  int64
 55  FLAG_DOCUMENT_12             307511 non-null  int64
 56  FLAG_DOCUMENT_13             307511 non-null  int64
 57  FLAG_DOCUMENT_14             307511 non-null  int64
```

```
 58  FLAG_DOCUMENT_15             307511 non-null  int64
 59  FLAG_DOCUMENT_16             307511 non-null  int64
 60  FLAG_DOCUMENT_17             307511 non-null  int64
 61  FLAG_DOCUMENT_18             307511 non-null  int64
 62  FLAG_DOCUMENT_19             307511 non-null  int64
 63  FLAG_DOCUMENT_20             307511 non-null  int64
 64  FLAG_DOCUMENT_21             307511 non-null  int64
 65  AMT_REQ_CREDIT_BUREAU_HOUR   307511 non-null  float64
 66  AMT_REQ_CREDIT_BUREAU_DAY    307511 non-null  float64
 67  AMT_REQ_CREDIT_BUREAU_WEEK   307511 non-null  float64
 68  AMT_REQ_CREDIT_BUREAU_MON    307511 non-null  float64
 69  AMT_REQ_CREDIT_BUREAU_QRT    307511 non-null  float64
 70  AMT_REQ_CREDIT_BUREAU_YEAR   307511 non-null  float64
 71  AGE                          307511 non-null  float64
 72  AGE_GROUPS                   307511 non-null  category
 73  YEARS_EMPLOYED               307511 non-null  float64
 74  EMPLOYEMENT_YEARS            252135 non-null  category
 75  AMT_CREDIT_RANGE             307511 non-null  category
 76  AMT_INCOME_RANGE             307279 non-null  category
 77  AMT_GOODS_PRICE_RANGE        307233 non-null  category
dtypes: category(25), float64(19), int64(33), object(1)
memory usage: 131.7+ MB
```

As of now, the Data Preprocessing on application_data.csv is done. We can do the EDA on these as we please.

Dataset 2: previous_application.csv

repeating same for this dataset

In [64]:
```
previous_appl=pd.read_csv("previous_application.csv")
```

In [65]:
```
previous_appl.info()
```

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

In [66]:
```
previous_appl.head(10)
```

Out[66]:

	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

In [67]:
```
previous_appl.tail(10)
```

Out[67]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CR
1670204	1407146	198989	Cash loans	36598.095	450000.0	570
1670205	2815130	338803	Cash loans	14584.050	135000.0	182
1670206	2459206	238591	Cash loans	19401.435	180000.0	243
1670207	1662353	443544	Cash loans	12607.875	112500.0	112
1670208	1556789	209732	Cash loans	22299.390	315000.0	436
1670209	2300464	352015	Consumer loans	14704.290	267295.5	311
1670210	2357031	334635	Consumer loans	6622.020	87750.0	64
1670211	2659632	249544	Consumer loans	11520.855	105237.0	102
1670212	2785582	400317	Cash loans	18821.520	180000.0	191
1670213	2418762	261212	Cash loans	16431.300	360000.0	360

previous_appl.size

In [68]:
```
previous_appl.shape
```

Out[68]:
```
(1670214, 37)
```

In [69]:
```
previous_appl.describe()
```

Out[69]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMEN
count	1.670214e+06	1.670214e+06	1.297979e+06	1.670214e+06	1.670213e+06	7.743700e+0
mean	1.923089e+06	2.783572e+05	1.595512e+04	1.752339e+05	1.961140e+05	6.697402e+0
std	5.325980e+05	1.028148e+05	1.478214e+04	2.927798e+05	3.185746e+05	2.092150e+0
min	1.000001e+06	1.000010e+05	0.000000e+00	0.000000e+00	0.000000e+00	-9.000000e-0
25%	1.461857e+06	1.893290e+05	6.321780e+03	1.872000e+04	2.416050e+04	0.000000e+0
50%	1.923110e+06	2.787145e+05	1.125000e+04	7.104600e+04	8.054100e+04	1.638000e+0
75%	2.384280e+06	3.675140e+05	2.065842e+04	1.803600e+05	2.164185e+05	7.740000e+0
max	2.845382e+06	4.562550e+05	4.180581e+05	6.905160e+06	6.905160e+06	3.060045e+0

Just like last dataset, this also has negative values in days columns, so some cleaning needs to be done

In [70]:
```
null_vals(previous_appl)
```

Out[70]:
```
RATE_INTEREST_PRIVILEGED       99.64
RATE_INTEREST_PRIMARY          99.64
AMT_DOWN_PAYMENT               53.64
RATE_DOWN_PAYMENT              53.64
NAME_TYPE_SUITE                49.12
NFLAG_INSURED_ON_APPROVAL      40.30
DAYS_TERMINATION               40.30
DAYS_LAST_DUE                  40.30
DAYS_LAST_DUE_1ST_VERSION      40.30
DAYS_FIRST_DUE                 40.30
DAYS_FIRST_DRAWING             40.30
AMT_GOODS_PRICE                23.08
AMT_ANNUITY                    22.29
CNT_PAYMENT                    22.29
PRODUCT_COMBINATION             0.02
AMT_CREDIT                      0.00
NAME_YIELD_GROUP                0.00
NAME_PORTFOLIO                  0.00
NAME_SELLER_INDUSTRY            0.00
SELLERPLACE_AREA                0.00
CHANNEL_TYPE                    0.00
NAME_PRODUCT_TYPE               0.00
SK_ID_PREV                      0.00
NAME_GOODS_CATEGORY             0.00
NAME_CLIENT_TYPE                0.00
CODE_REJECT_REASON              0.00
SK_ID_CURR                      0.00
DAYS_DECISION                   0.00
NAME_CONTRACT_STATUS            0.00
NAME_CASH_LOAN_PURPOSE          0.00
NFLAG_LAST_APPL_IN_DAY          0.00
FLAG_LAST_APPL_PER_CONTRACT     0.00
HOUR_APPR_PROCESS_START         0.00
WEEKDAY_APPR_PROCESS_START      0.00
AMT_APPLICATION                 0.00
NAME_CONTRACT_TYPE              0.00
NAME_PAYMENT_TYPE               0.00
dtype: float64
```

In [71]:
```
p_nulls_50=null_vals(previous_appl)[null_vals(previous_appl)>50]
```

In [72]:
```
p_nulls_50
```

Out[72]:
```
RATE_INTEREST_PRIVILEGED    99.64
RATE_INTEREST_PRIMARY       99.64
AMT_DOWN_PAYMENT            53.64
RATE_DOWN_PAYMENT           53.64
dtype: float64
```

In [73]:
```
#dropping these columns
previous_appl.drop(columns = p_nulls_50.index, inplace=True)
```

In [74]:
```
# now checking for null % >15
p_nulls_15=null_vals(previous_appl)[null_vals(previous_appl)>15]
```

In [75]:
```
previous_appl[p_nulls_15.index]
```

Out[75]:

	NAME_TYPE_SUITE	DAYS_FIRST_DRAWING	DAYS_TERMINATION	DAYS_LAST_DUE	DAYS_LAST_I
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	
...	...	...	...	...	
1670209	NaN	365243.0	-351.0	-358.0	
1670210	Unaccompanied	365243.0	-1297.0	-1304.0	
1670211	Spouse, partner	365243.0	-1181.0	-1187.0	
1670212	Family	365243.0	-817.0	-825.0	
1670213	Family	365243.0	-423.0	-443.0	

1670214 rows × 10 columns

In [76]:
```
previous_appl.columns
```

Out[76]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY', 'AMT_APPLICATI
ON', '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_PURP
OSE', 'NAME_CONTRACT_STATUS', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE', 'CODE_REJECT_REASO
N', 'NAME_TYPE_SUITE', 'NAME_CLIENT_TYPE', 'NAME_GOODS_CATEGORY', 'NAME_PORTFOLIO', 'N
AME_PRODUCT_TYPE', 'CHANNEL_TYPE', 'SELLERPLACE_AREA',
       'NAME_SELLER_INDUSTRY', 'CNT_PAYMENT', 'NAME_YIELD_GROUP', 'PRODUCT_COMBINATIO
N', 'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION', 'DAYS_LAST_DU
E', 'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROVAL'],
      dtype='object')
```

In [77]:
```
not_needed=['WEEKDAY_APPR_PROCESS_START','HOUR_APPR_PROCESS_START','FLAG_LAST_APPL_PER_(

previous_appl.drop(not_needed, axis=1, inplace=True)
```

In [78]:
```
previous_appl.shape
```
Out[78]:
```
(1670214, 29)
```
In [79]:
```
#filling notknown in this categorical column
previous_appl["NAME_TYPE_SUITE"]=previous_appl["NAME_TYPE_SUITE"].fillna("NotKnown")

null_vals(previous_appl)
```
Out[79]:
```
NFLAG_INSURED_ON_APPROVAL    40.30
DAYS_TERMINATION             40.30
DAYS_LAST_DUE                40.30
DAYS_LAST_DUE_1ST_VERSION    40.30
DAYS_FIRST_DUE               40.30
DAYS_FIRST_DRAWING           40.30
AMT_GOODS_PRICE              23.08
AMT_ANNUITY                  22.29
CNT_PAYMENT                  22.29
PRODUCT_COMBINATION           0.02
AMT_CREDIT                    0.00
NAME_PRODUCT_TYPE             0.00
NAME_YIELD_GROUP              0.00
NAME_SELLER_INDUSTRY          0.00
SELLERPLACE_AREA              0.00
CHANNEL_TYPE                  0.00
SK_ID_PREV                    0.00
NAME_PORTFOLIO                0.00
SK_ID_CURR                    0.00
NAME_CLIENT_TYPE              0.00
NAME_TYPE_SUITE               0.00
CODE_REJECT_REASON            0.00
NAME_PAYMENT_TYPE             0.00
DAYS_DECISION                 0.00
NAME_CONTRACT_STATUS          0.00
NAME_CASH_LOAN_PURPOSE        0.00
AMT_APPLICATION               0.00
NAME_CONTRACT_TYPE            0.00
NAME_GOODS_CATEGORY           0.00
dtype: float64
```
In [80]:
```
#converting negative days to positive days
pos_days= ['DAYS_DECISION','DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VI
previous_appl[pos_days]=abs(previous_appl[pos_days])
previous_appl[p_nulls_15.index].describe()
```
Out[80]:

	DAYS_FIRST_DRAWING	DAYS_TERMINATION	DAYS_LAST_DUE	DAYS_LAST_DUE_1ST_VERSION	DA
count	997149.000000	997149.000000	997149.000000	997149.000000	
mean	342340.056543	83505.775017	78152.730207	35163.363265	
std	88413.495220	152484.418802	148833.342466	106405.950190	
min	2.000000	2.000000	2.000000	0.000000	
25%	365243.000000	447.000000	455.000000	257.000000	
50%	365243.000000	1171.000000	1155.000000	741.000000	
75%	365243.000000	2501.000000	2418.000000	1735.000000	
max	365243.000000	365243.000000	365243.000000	365243.000000	

In [81]:
```
bins = [0,1*365,2*365,3*365,4*365,5*365,6*365,7*365,10*365]
ranges = ["1","2","3","4","5","6","7","above 7"]
previous_appl["YEARLY_DECISION"]=pd.cut(previous_appl["DAYS_DECISION"], bins, labels=ra
```

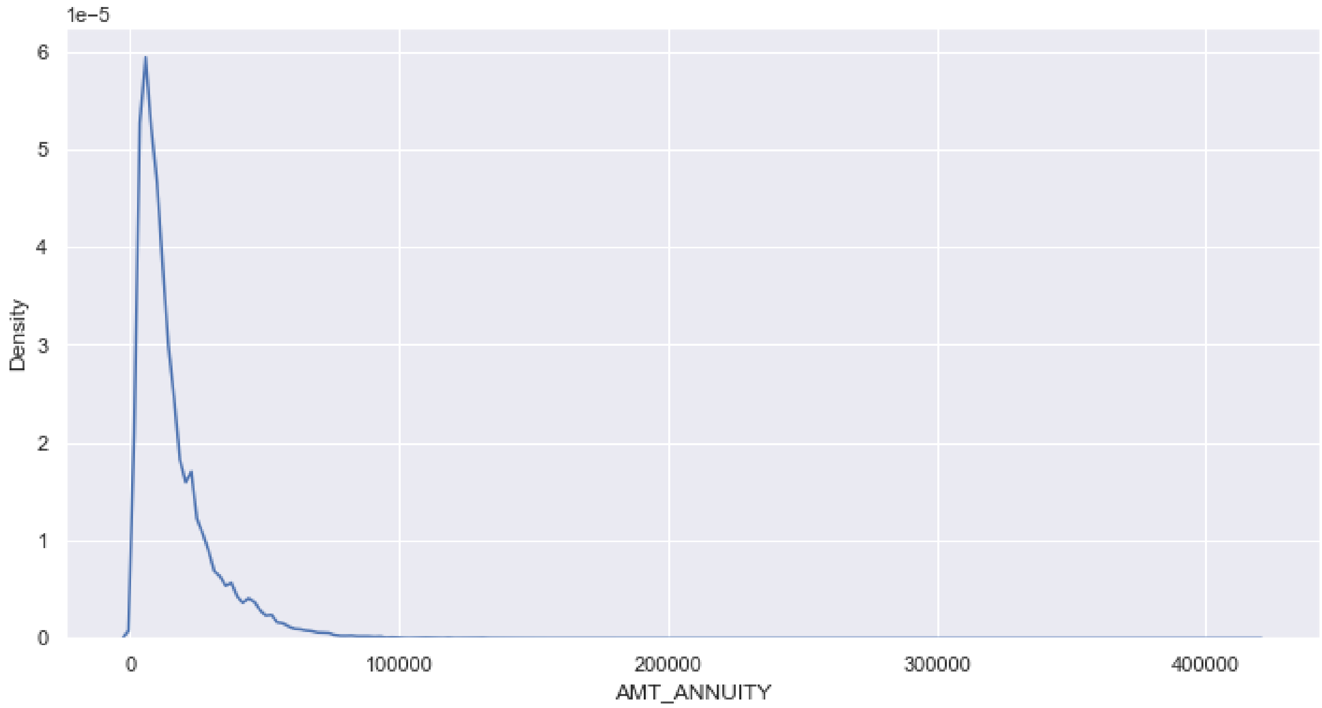
In [82]:
```
previous_appl["YEARLY_DECISION"].value_counts(normalize=True)*100
```

Out[82]:
```
1          34.351287
2          23.056806
3          12.855598
4           7.883181
5           6.128556
7           5.813806
above 7     5.060729
6           4.850037
Name: YEARLY_DECISION, dtype: float64
```

In [83]:
```
null_vals(previous_appl)
```

Out[83]:
```
NFLAG_INSURED_ON_APPROVAL    40.30
DAYS_TERMINATION             40.30
DAYS_LAST_DUE                40.30
DAYS_LAST_DUE_1ST_VERSION    40.30
DAYS_FIRST_DUE               40.30
DAYS_FIRST_DRAWING           40.30
AMT_GOODS_PRICE              23.08
AMT_ANNUITY                  22.29
CNT_PAYMENT                  22.29
PRODUCT_COMBINATION           0.02
AMT_CREDIT                    0.00
SK_ID_PREV                    0.00
CHANNEL_TYPE                  0.00
NAME_YIELD_GROUP              0.00
NAME_SELLER_INDUSTRY          0.00
SELLERPLACE_AREA              0.00
NAME_PORTFOLIO                0.00
NAME_PRODUCT_TYPE             0.00
SK_ID_CURR                    0.00
NAME_GOODS_CATEGORY           0.00
NAME_CLIENT_TYPE              0.00
NAME_TYPE_SUITE               0.00
CODE_REJECT_REASON            0.00
NAME_PAYMENT_TYPE             0.00
DAYS_DECISION                 0.00
NAME_CONTRACT_STATUS          0.00
NAME_CASH_LOAN_PURPOSE        0.00
AMT_APPLICATION               0.00
NAME_CONTRACT_TYPE            0.00
YEARLY_DECISION               0.00
dtype: float64
```

In [84]:
```
#Filling null values in the continuous vars AMT_ANNUITY, AMT_GOODS_PRICE
sns.set(style="darkgrid")
plt.figure(figsize=(12,6))
sns.kdeplot(previous_appl["AMT_ANNUITY"])
plt.show()
```

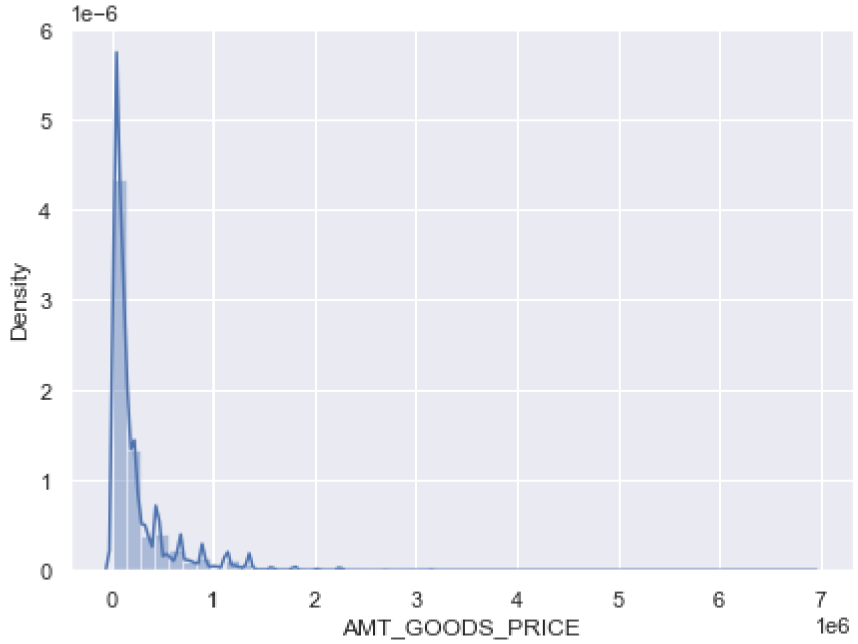


As the graph peaks at extreme left, mean can't be filled in, as there are a lot of outliers, Thus we can go with median filling

In [85]:
```
statsDF = pd.DataFrame()
statsDF['AMT_GOODS_PRICE_mode'] = previous_appl['AMT_GOODS_PRICE'].fillna(previous_appl
statsDF['AMT_GOODS_PRICE_median'] = previous_appl['AMT_GOODS_PRICE'].fillna(previous_app
statsDF['AMT_GOODS_PRICE_mean'] = previous_appl['AMT_GOODS_PRICE'].fillna(previous_appl
```

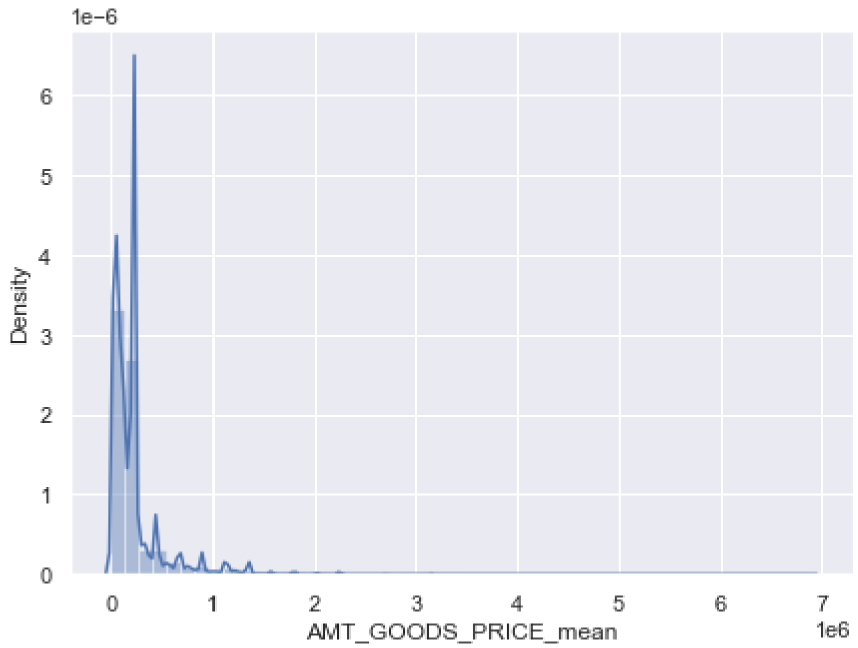
In [86]:
```
#Lets check the columns in plots
```

In [87]:
```
plt.figure(figsize=(7,5))
sns.distplot(previous_appl['AMT_GOODS_PRICE'][pd.notnull(previous_appl['AMT_GOODS_PRICE
```



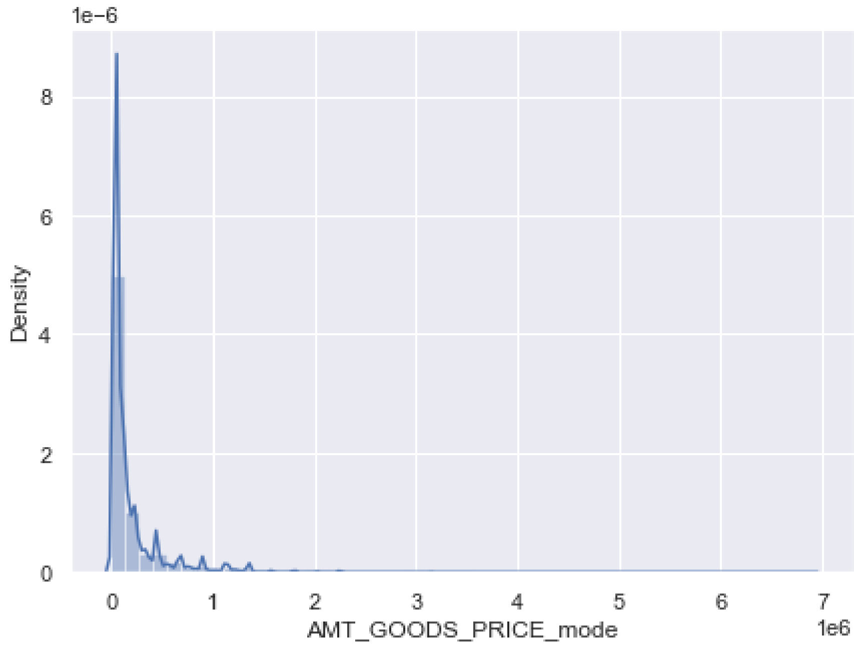
In [88]:
```
plt.figure(figsize=(7,5))
sns.distplot(statsDF["AMT_GOODS_PRICE_mean"])
```

Out[88]: <AxesSubplot:xlabel='AMT_GOODS_PRICE_mean', ylabel='Density'>



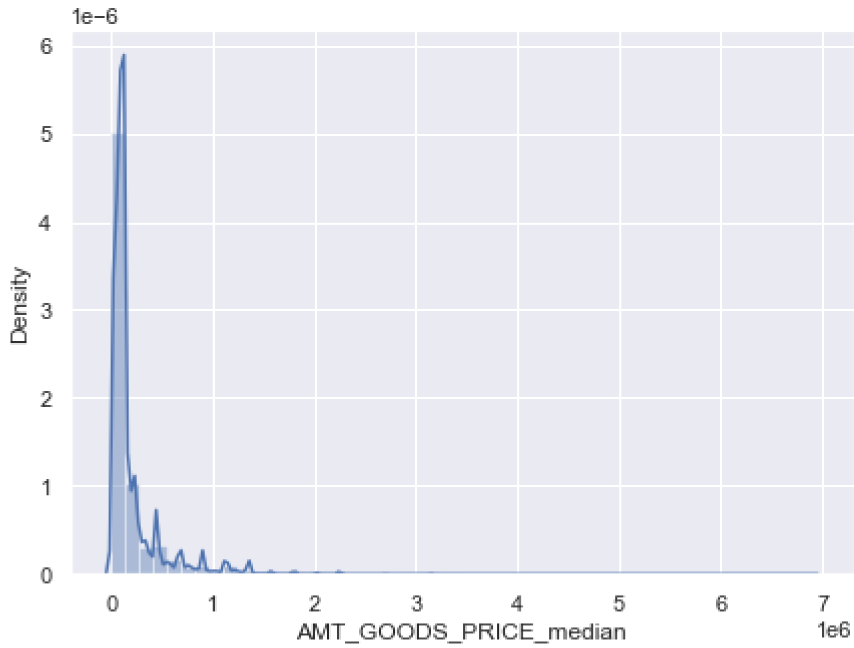
In [89]:
```
plt.figure(figsize=(7,5))
sns.distplot(statsDF["AMT_GOODS_PRICE_mode"])
```

Out[89]: <AxesSubplot:xlabel='AMT_GOODS_PRICE_mode', ylabel='Density'>



In [90]:
```
plt.figure(figsize=(7,5))
sns.distplot(statsDF["AMT_GOODS_PRICE_median"])
```

Out[90]: <AxesSubplot:xlabel='AMT_GOODS_PRICE_median', ylabel='Density'>



As we can see the mode graph is much similiar to the original graph, so we will go with mode replacement

In [91]:
```
#filling in mode in null values
previous_appl['AMT_GOODS_PRICE'].fillna(previous_appl['AMT_GOODS_PRICE'].mode()[0], inp
```

We can put in 0 in CNT_PAYMENT as NAME_CONTRACT_STATUS, as most of loans weren't started

In [92]:
```
previous_appl.loc[previous_appl['CNT_PAYMENT'].isnull(),'NAME_CONTRACT_STATUS'].value_co
```

Out[92]:
```
Canceled        305805
Refused          40897
Unused offer     25524
Approved             4
Name: NAME_CONTRACT_STATUS, dtype: int64
```

In [93]:
```
previous_appl["CNT_PAYMENT"].fillna(0, inplace=True)
```

In [94]:
```
#converting object columns to categorical columns
p_cat_cols= ['NAME_CASH_LOAN_PURPOSE','NAME_CONTRACT_STATUS','NAME_PAYMENT_TYPE',
                    'CODE_REJECT_REASON','NAME_CLIENT_TYPE','NAME_GOODS_CATEGORY','NAME_
                   'NAME_PRODUCT_TYPE','CHANNEL_TYPE','NAME_SELLER_INDUSTRY','NAME_YIELD
                    'NAME_CONTRACT_TYPE']
for i in p_cat_cols:
    previous_appl[i]=pd.Categorical(previous_appl[i])
```

Searching for Outliers

In [95]:
```
previous_appl.describe()
```

Out[95]:

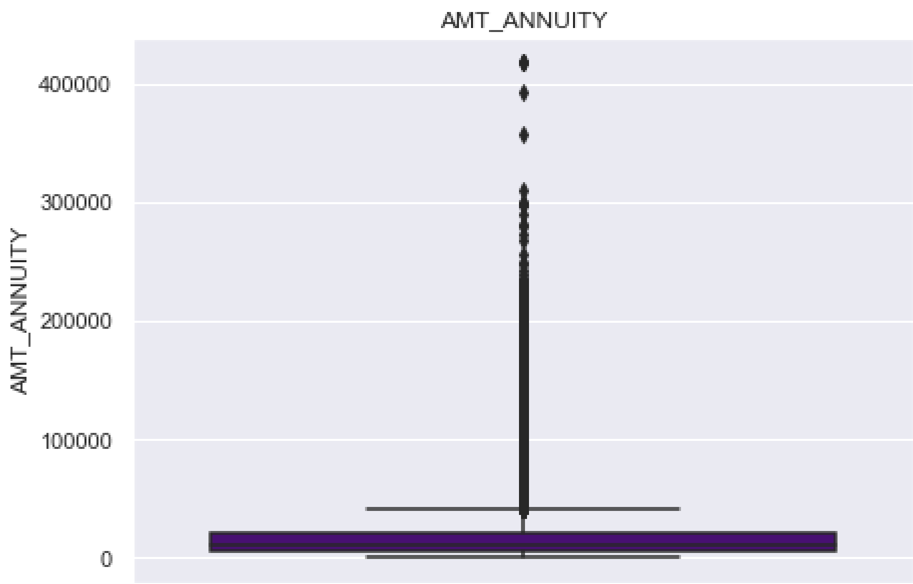
	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE
count	1.670214e+06	1.670214e+06	1.297979e+06	1.670214e+06	1.670213e+06	1.670214e+06
mean	1.923089e+06	2.783572e+05	1.595512e+04	1.752339e+05	1.961140e+05	1.856429e+05
std	5.325980e+05	1.028148e+05	1.478214e+04	2.927798e+05	3.185746e+05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

Just like application_data.csv, this csv file also has a huge difference in max value and 75 percentile. which means there are a lot of outliers

In [96]:
```
p_put_cols=['CHANNEL_TYPE','NAME_CONTRACT_STATUS',
                    'CODE_REJECT_REASON','NAME_CLIENT_TYPE','NAME_GOODS_CATEGORY','NAME_
                   ,'NAME_PAYMENT_TYPE','NAME_PRODUCT_TYPE','NAME_YIELD_GROUP','PRODUCT_
                    'NAME_CONTRACT_TYPE','NAME_SELLER_INDUSTRY','NAME_CASH_LOAN_PURPOSE
```

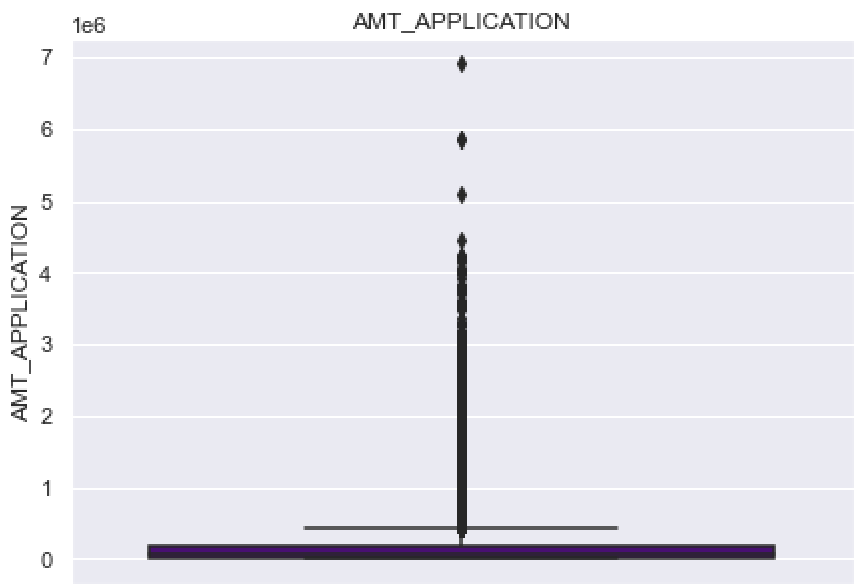
In [97]:
```
plt.figure(figsize=[7,5])
sns.boxplot(y=previous_appl["AMT_ANNUITY"], orient="h", color="indigo")
plt.title("AMT_ANNUITY")
```

Out[97]: Text(0.5, 1.0, 'AMT_ANNUITY')



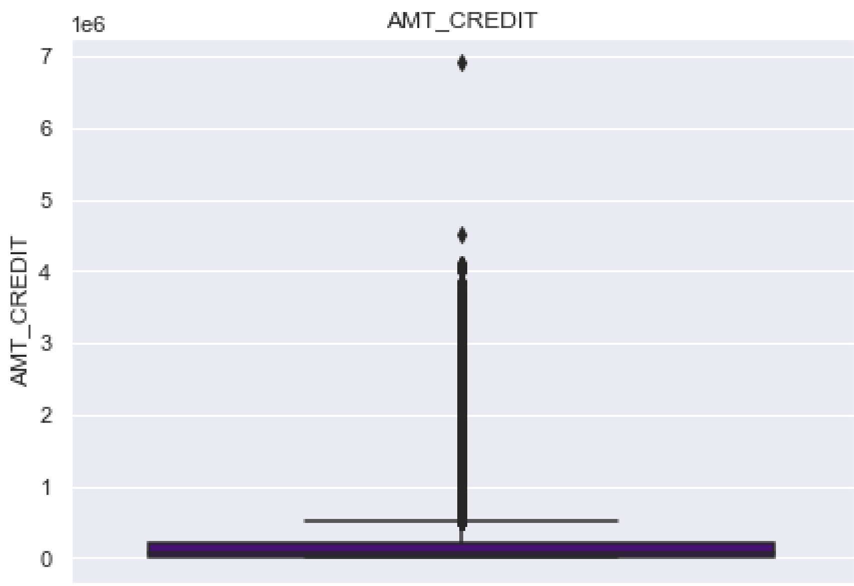
In [98]:
```
plt.figure(figsize=[7,5])
sns.boxplot(y=previous_appl["AMT_APPLICATION"], orient="h", color="indigo")
plt.title("AMT_APPLICATION")
```

Out[98]: Text(0.5, 1.0, 'AMT_APPLICATION')



In [99]:
```
plt.figure(figsize=[7,5])
sns.boxplot(y=previous_appl["AMT_CREDIT"], orient="h", color="indigo")
plt.title("AMT_CREDIT")
```

Out[99]: Text(0.5, 1.0, 'AMT_CREDIT')

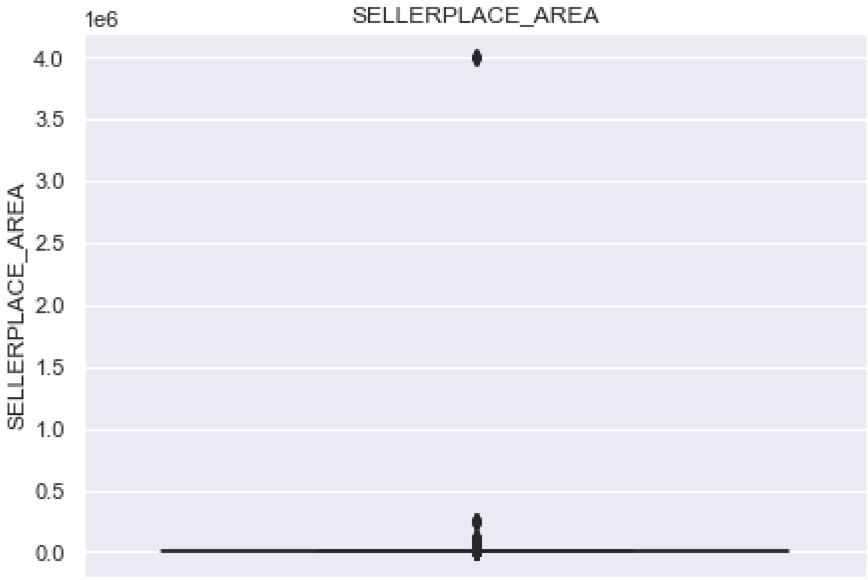


In [100]:
```
plt.figure(figsize=[7,5])
sns.boxplot(y=previous_appl["AMT_GOODS_PRICE"], orient="h", color="indigo")
plt.title("AMT_GOODS_PRICE")
```

Out[100]: Text(0.5, 1.0, 'AMT_GOODS_PRICE')

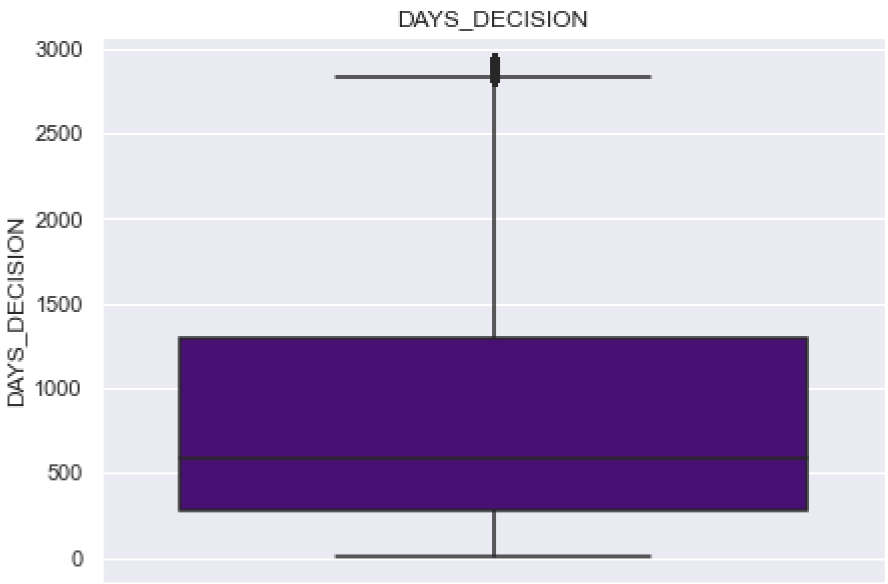
In [101]:
```
plt.figure(figsize=[7,5])
sns.boxplot(y=previous_appl["SELLERPLACE_AREA"], orient="h", color="indigo")
plt.title("SELLERPLACE_AREA")
```

Out[101]: Text(0.5, 1.0, 'SELLERPLACE_AREA')



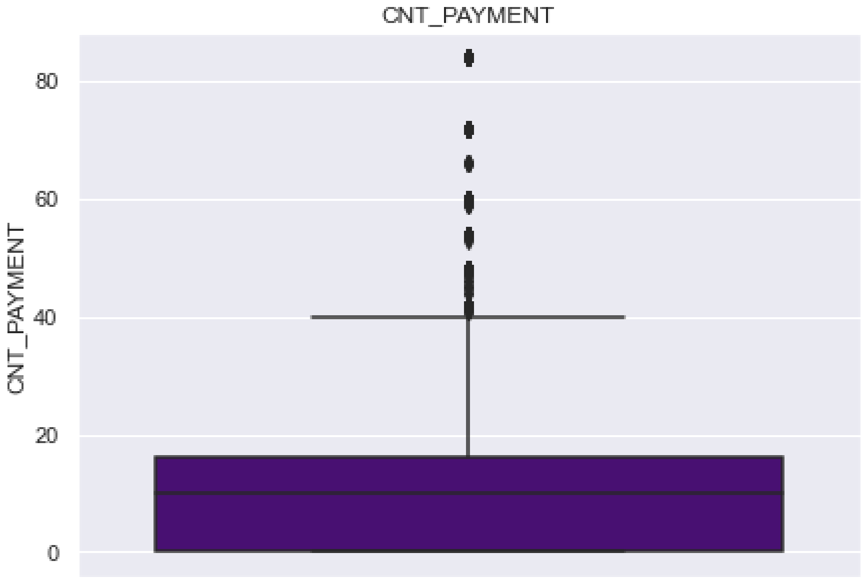
In [102]:
```
plt.figure(figsize=[7,5])
sns.boxplot(y=previous_appl["DAYS_DECISION"], orient="h", color="indigo")
plt.title("DAYS_DECISION")
```

Out[102]: Text(0.5, 1.0, 'DAYS_DECISION')



In [103]:
```
plt.figure(figsize=[7,5])
sns.boxplot(y=previous_appl["CNT_PAYMENT"], orient="h", color="indigo")
plt.title("CNT_PAYMENT")
```

Out[103]: Text(0.5, 1.0, 'CNT_PAYMENT')



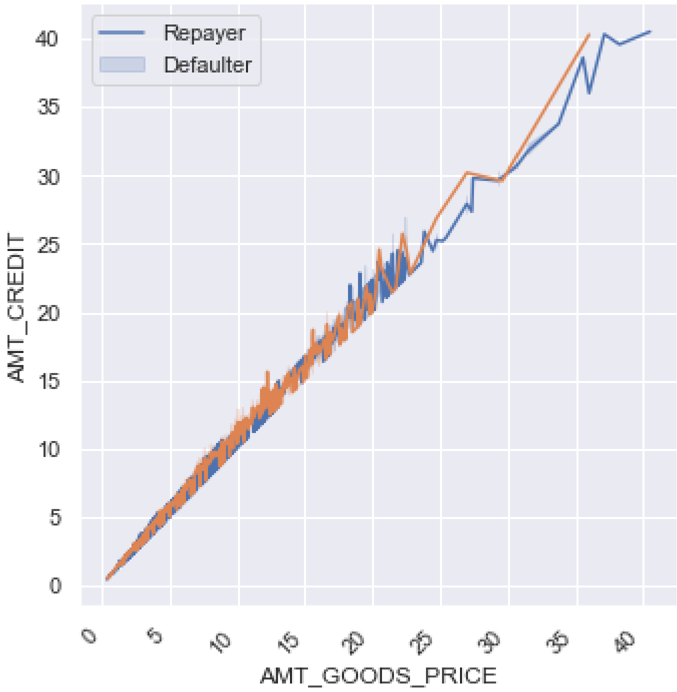
Now we can successfully start Exploratory Data Analysis

Numerical Bivariate Analysis

In [104]:
```
#defining function for plotting repetetive plots in bivariate numerical analysis
def bivar_n(x,y,df,hue,kind,labels):
    plt.figure(figsize=[15,15])
    sns.relplot(x=x, y=y, data=df, hue=hue,kind=kind,legend = False)
    plt.legend(labels=labels)
    plt.xticks(rotation=45, ha='right')
    plt.show()
```

In [105]:
```
# Looking for relationship between goodsprice and credit and analysing with repayment st
bivar_n('AMT_GOODS_PRICE','AMT_CREDIT',application_data,"TARGET", "line",['Repayer','De
```

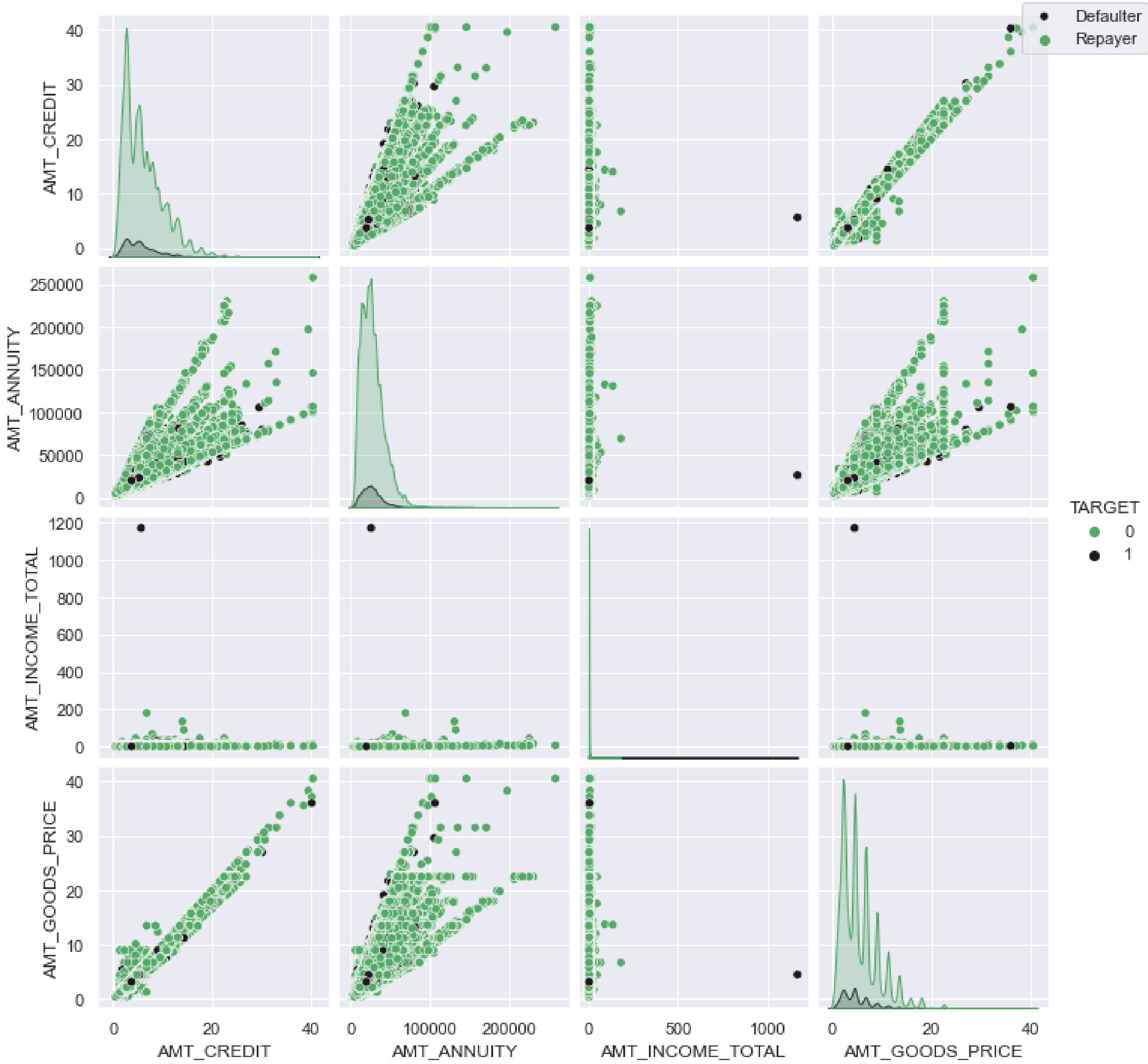
<Figure size 1080x1080 with 0 Axes>



From the graph we can infer that amount going above 25 lacs increases the defaulters

In [106]:

```
# Comparing amount to loan repayment status
amount = application_data[['AMT_CREDIT','TARGET','AMT_ANNUITY','AMT_INCOME_TOTAL', 'AMT_
amount = amount[(amount["AMT_GOODS_PRICE"].notnull()) & (amount["AMT_ANNUITY"].notnull()

ax=sns.pairplot(amount, hue="TARGET", palette=['g', 'k'])
ax.fig.legend(labels=['Defaulter', 'Repayer'])
plt.show()
```



Categorical Bivariate Analysis

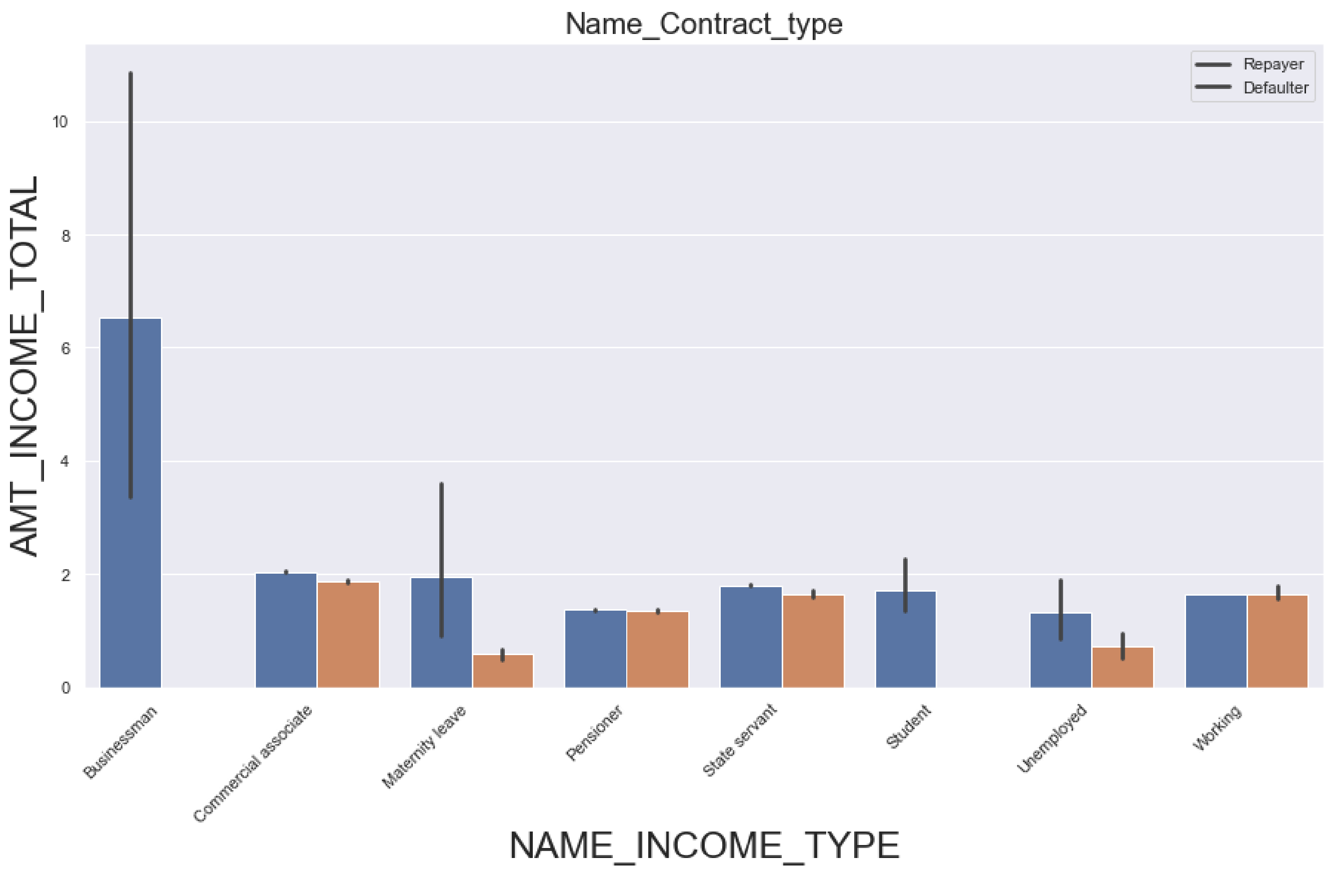
In [107]:
```
application_data.groupby('NAME_INCOME_TYPE')['AMT_INCOME_TOTAL'].describe()
```

Out[107]:

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

In [108]:
```
# function for plotting repetitive barplots in bivariate categorical analysis

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

    plt.figure(figsize=figsize)
    sns.barplot(x=x,y=y,data=df, hue=hue)
    plt.xlabel(x,fontsize = 25)
    plt.ylabel(y,fontsize = 25)
    plt.title("Name_Contract_type", fontsize = 20)
    plt.xticks(rotation=45, ha='right')
    plt.legend(labels = labels )
    plt.show()
```

In [109]:
```
# bar plot of IncomeType vs IncomeAmtRange
bivar_c("NAME_INCOME_TYPE", "AMT_INCOME_TOTAL", application_data, "TARGET",(15,8),["Repa
```



insights: Businessmen seem to have higher income, with ranges from just below 4 lacs to well above 10 lacs

Numerical Univariate Analysis

In [110]:
```
# bisecting the app_data dataframe based on Target value 0 and 1 for correlation and oth

cols_for_correlation = ['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_REALTY',
                        'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY',
                        'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'I
                        'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH',
                        'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'OCCUPATION_TYPE', 'CNT_
                        'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_START', 'HO
                        'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION', 'LIV
                        'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_N
                        'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'DAYS_LA
                        'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_
                        'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_I
```

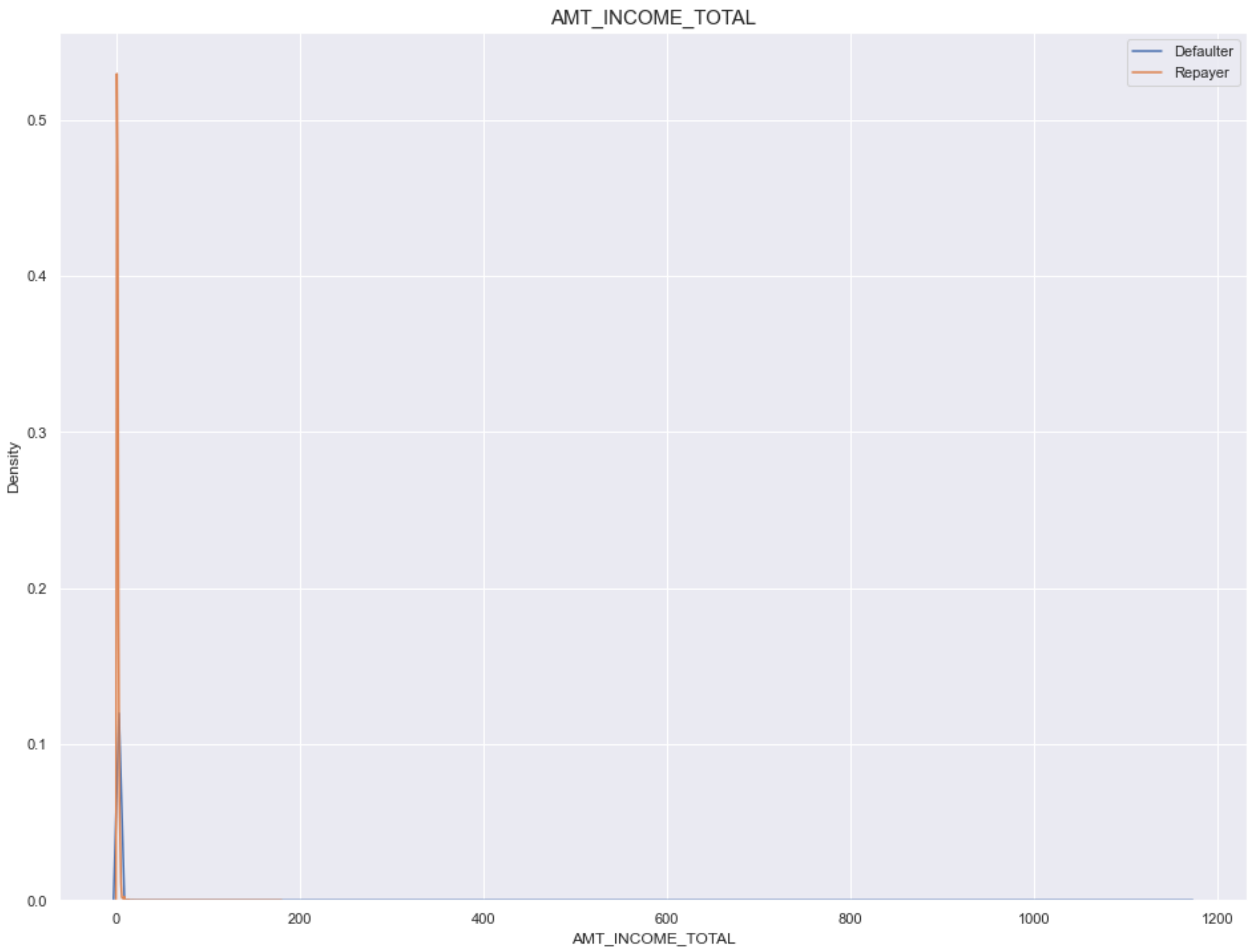
In [111]:
```
#Dataframe of repayers
Repayer_df = application_data.loc[application_data['TARGET']==0, cols_for_correlation]
```

In [112]:
```
# DF of defaulters
Defaulter_df = application_data.loc[application_data['TARGET']==1, cols_for_correlation]
```

In [113]:
```
amt=application_data[["AMT_INCOME_TOTAL", "AMT_CREDIT", "AMT_ANNUITY", "AMT_GOODS_PRICE'
```

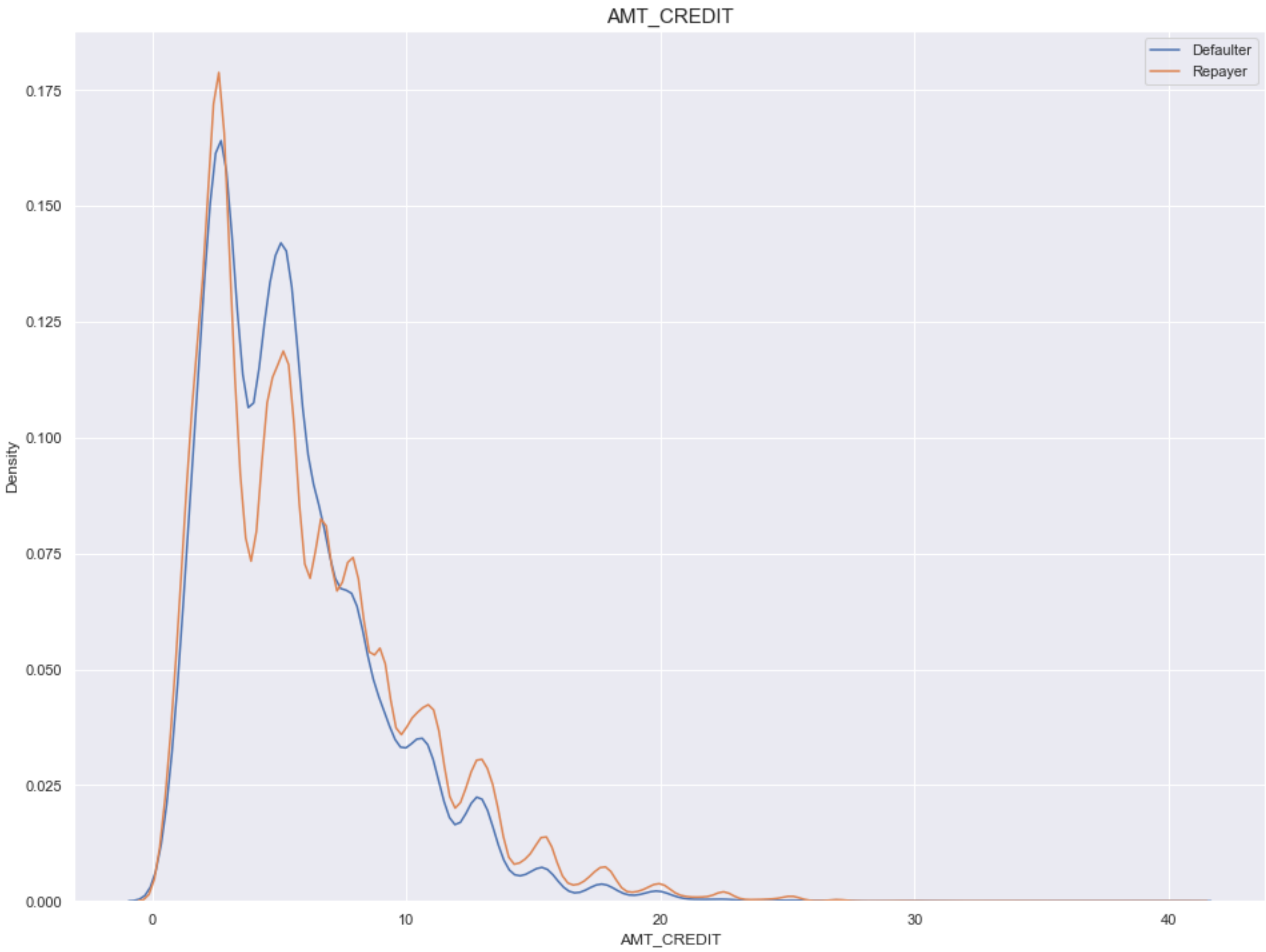
In [114]:
```
fig=plt.figure(figsize=(16,12))
sns.distplot(Defaulter_df["AMT_INCOME_TOTAL"], hist=False,label ="Defaulter")
sns.distplot(Repayer_df["AMT_INCOME_TOTAL"], hist=False, label ="Repayer")
plt.title("AMT_INCOME_TOTAL", fontdict={'fontsize' : 15, 'fontweight' : 5})
plt.legend()
```

Out[114]: <matplotlib.legend.Legend at 0x198811027f0>



In [115]:
```
fig=plt.figure(figsize=(16,12))
sns.distplot(Defaulter_df["AMT_CREDIT"], hist=False,label ="Defaulter")
sns.distplot(Repayer_df["AMT_CREDIT"], hist=False, label ="Repayer")
plt.title("AMT_CREDIT", fontdict={'fontsize' : 15, 'fontweight' : 5})
plt.legend()
```

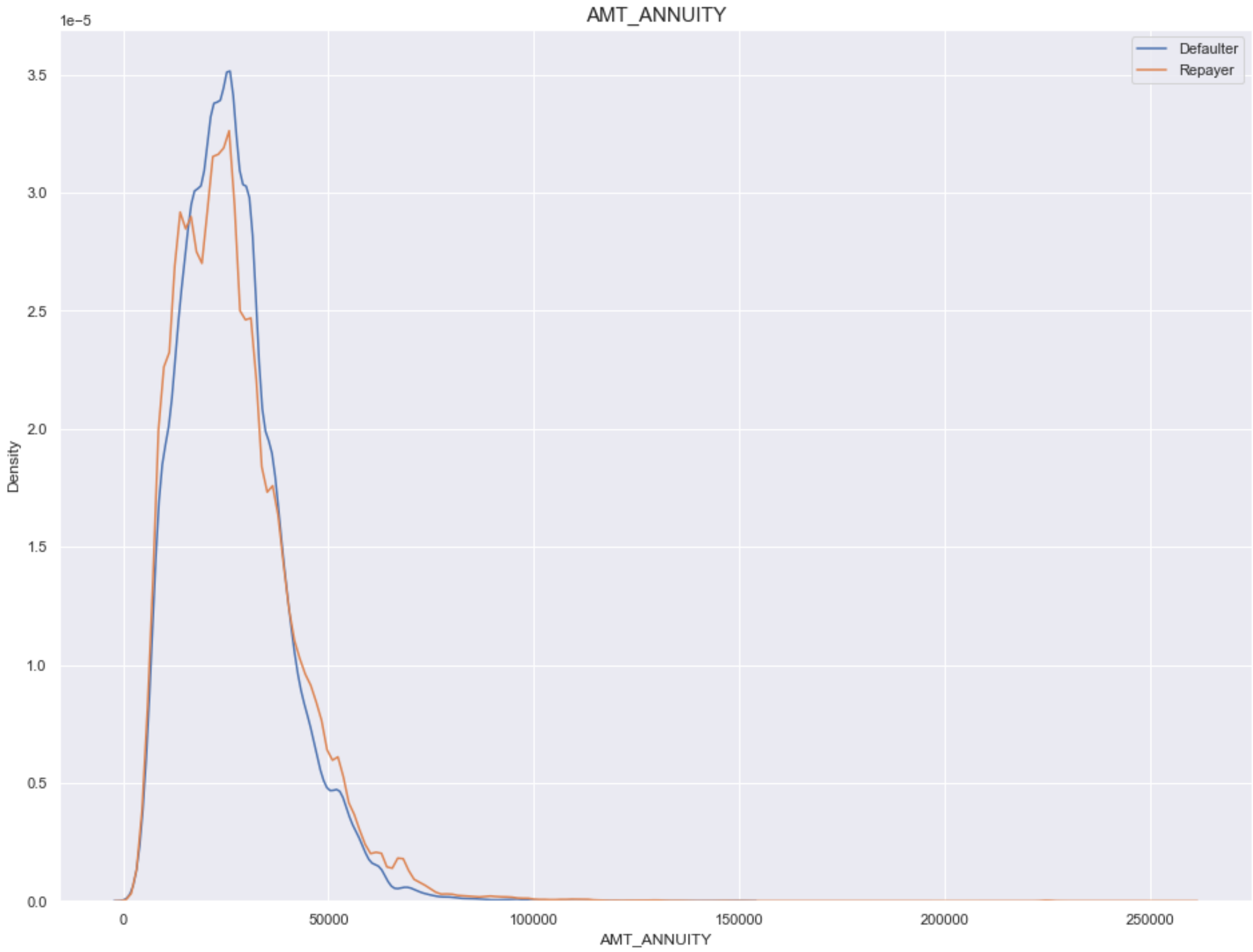
Out[115]: <matplotlib.legend.Legend at 0x1988115c850>



Insight: cred amount of most loans<10L

In [116]:
```
fig=plt.figure(figsize=(16,12))
sns.distplot(Defaulter_df["AMT_ANNUITY"], hist=False,label ="Defaulter")
sns.distplot(Repayer_df["AMT_ANNUITY"], hist=False, label ="Repayer")
plt.title("AMT_ANNUITY", fontdict={'fontsize' : 15, 'fontweight' : 5})
plt.legend()
```

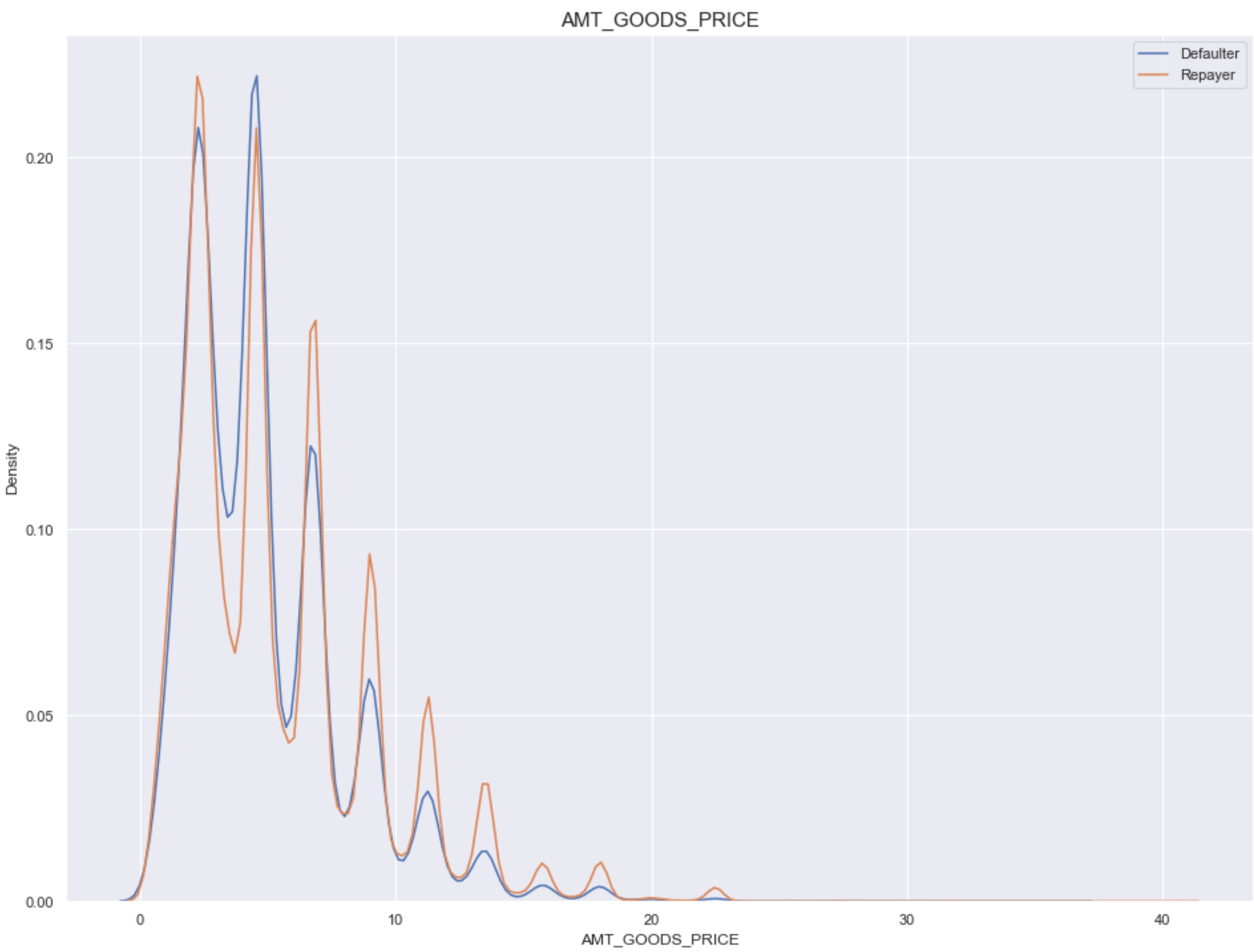
Out[116]: <matplotlib.legend.Legend at 0x198811d2220>



Insight: looking at the graph shows that most people pay annuity<50k for loan

In [117]:
```
fig=plt.figure(figsize=(16,12))
sns.distplot(Defaulter_df["AMT_GOODS_PRICE"], hist=False,label ="Defaulter")
sns.distplot(Repayer_df["AMT_GOODS_PRICE"], hist=False, label ="Repayer")
plt.title("AMT_GOODS_PRICE", fontdict={'fontsize' : 15, 'fontweight' : 5})
plt.legend()
```

Out[117]: <matplotlib.legend.Legend at 0x198812ab760>



Insight: goods price<10L for most loans

Numeric Variables Analysis

bisecting the application_data based on target value 0, 1 for correlation and other

In [118]:
```
len(cols_for_correlation)
```

Out[118]: 41

In [119]:

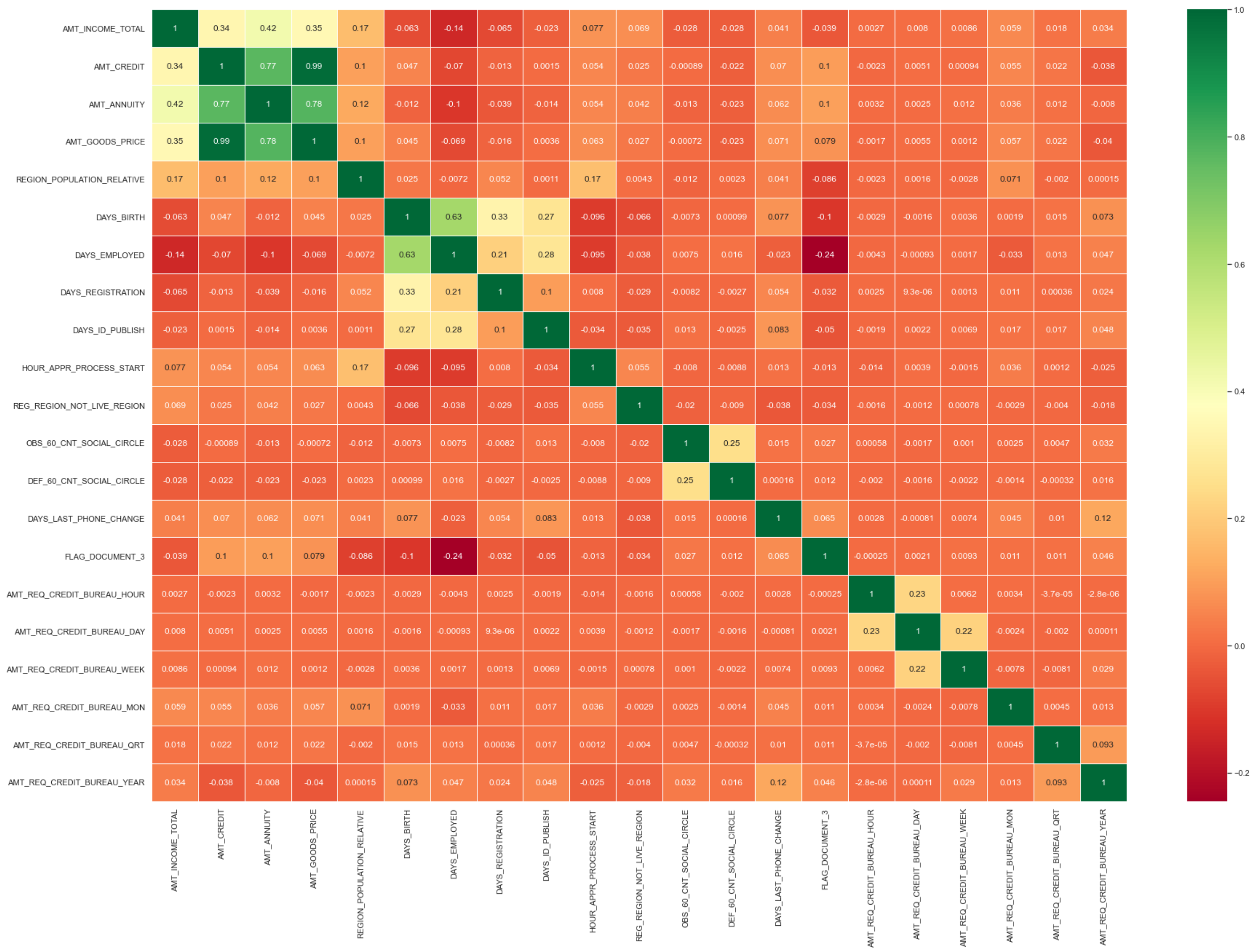
```
#correlation between numeric variables
corr_repayer = Repayer_df.corr()
corr_df_repayer = corr_repayer.where(np.triu(np.ones(corr_repayer.shape),k=1).astype(np
corr_df_repayer.columns =['VAR1','VAR2','Correlation']
corr_df_repayer.dropna(subset = ["Correlation"], inplace = True)
corr_df_repayer["Correlation"]=corr_df_repayer["Correlation"].abs()
corr_df_repayer.sort_values(by='Correlation', ascending=False, inplace=True)
corr_df_repayer.head(10)
```

Out[119]:

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

In [120]:

```
#plotting heatmap for linear correlation
fig=plt.figure(figsize=(30,20))
ax=sns.heatmap(Repayer_df.corr(), cmap="RdYlGn", annot=True, linewidth=0.5)
```



Insights: Credit amount is highly correlated to Goods_price_amount, loan_annuity, total_income

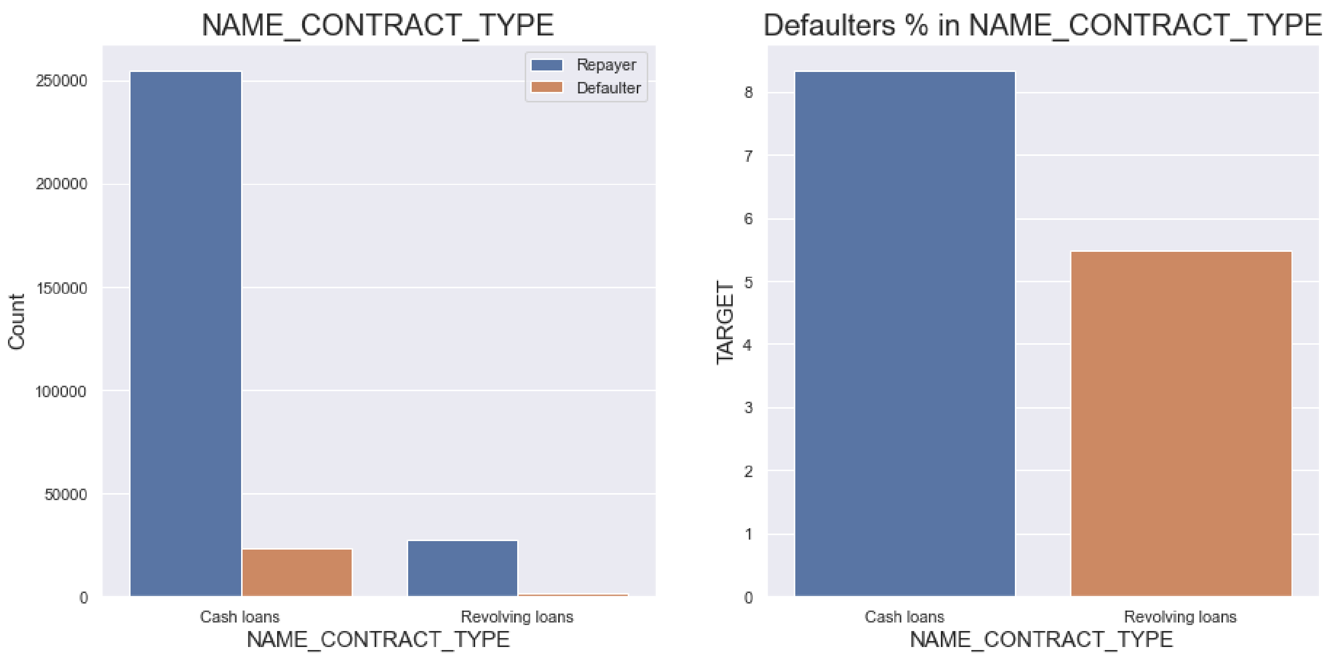
Categorical Variable Analysis

Segmented Univariate Analysis

In [121]:

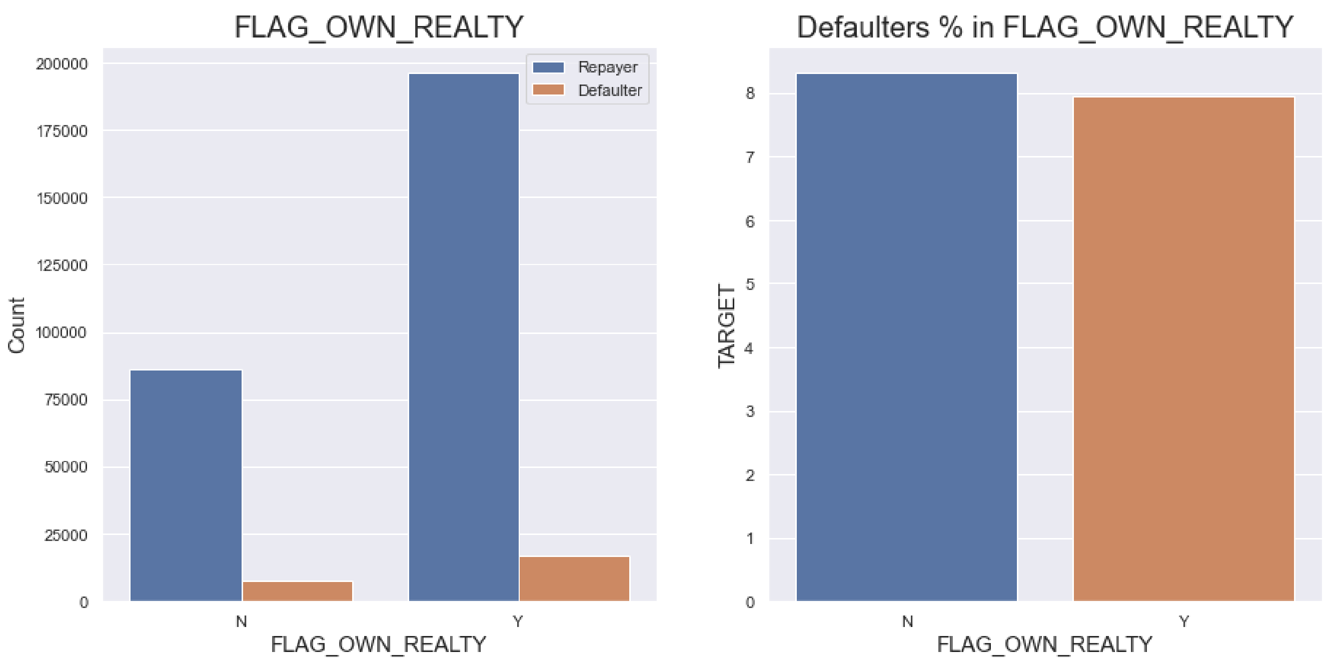
```python
# function to distinguish column from categorical or numerical
def data_type(dataset,col):
    if dataset[col].dtype == np.int64 or dataset[col].dtype == np.float64:
        return "numerical"
    if dataset[col].dtype == "category":
        return "categorical"

#function to perform analysis of single variable with target variable

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


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


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


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

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


# 1. Subplot 1: Count plot of the column

        s = sns.countplot(ax=ax1, x=col, data=dataset, hue=target_col)
        ax1.set_title(col, fontsize = 20)
        ax1.legend(['Repayer','Defaulter'])
        ax1.set_xlabel(col,fontdict={'fontsize' : 15, 'fontweight' : 3})

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

# 2. Subplot 2: Percentage of defaulters within the column

        s = sns.barplot(ax=ax2, x = col, y=target_col, data=target_1_percentage)
        ax2.set_title("Defaulters % in "+col, fontsize = 20)
        ax2.set_xlabel(col,fontdict={'fontsize' : 15, 'fontweight' : 3})
        ax2.set_ylabel(target_col,fontdict={'fontsize' : 15, 'fontweight' : 3})

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


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

        if ylog:
            ax1.set_yscale('log')
            ax1.set_ylabel("Count (log)",fontdict={'fontsize' : 15, 'fontweight' : 3})
        else:
            ax1.set_ylabel("Count",fontdict={'fontsize' : 15, 'fontweight' : 3})


        plt.show()
```

```
In [122]: # contracttype vs loanrepay status
univar(application_data, "NAME_CONTRACT_TYPE","TARGET", False, False,True)
```



- Revolving loans represent a very small portion of total loans
- CashLoan defaulter are 8-9% and Revolving loan defaulters are ~5.4%

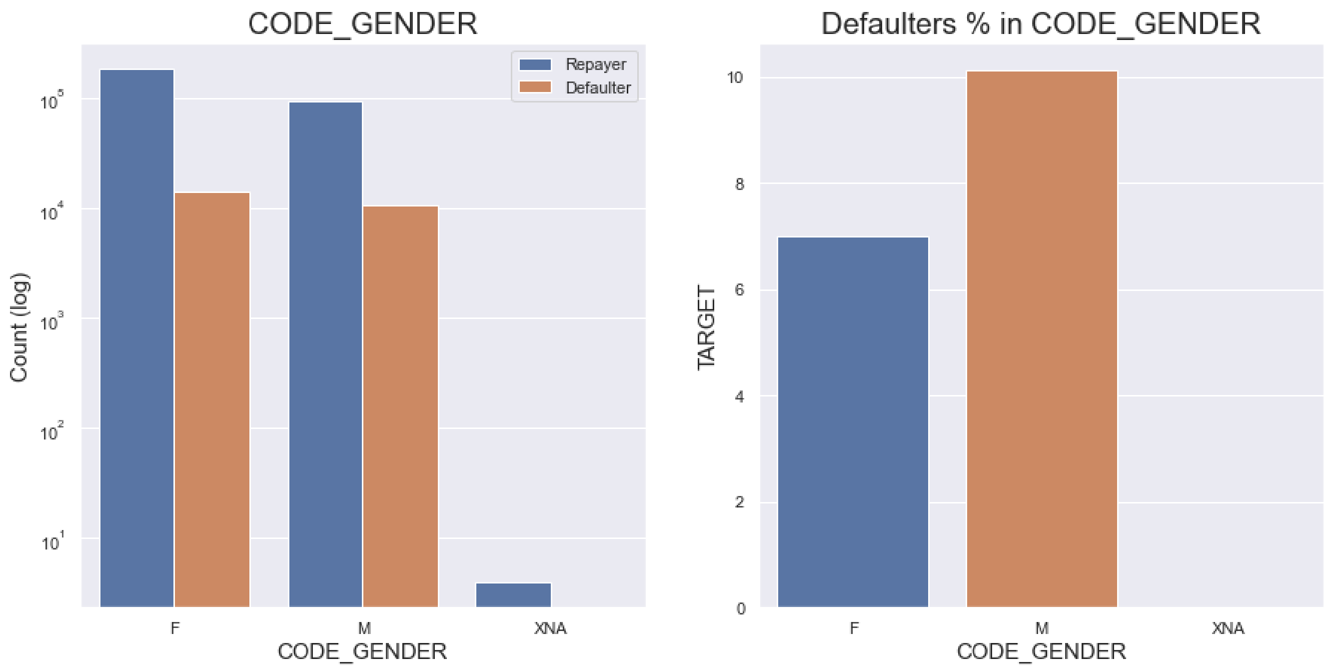
```
In [123]: #3 real estate ownership vs loan repay status

univar(application_data,"FLAG_OWN_REALTY","TARGET",False,False,True)
```



Real estate owners : non owners= 2:1

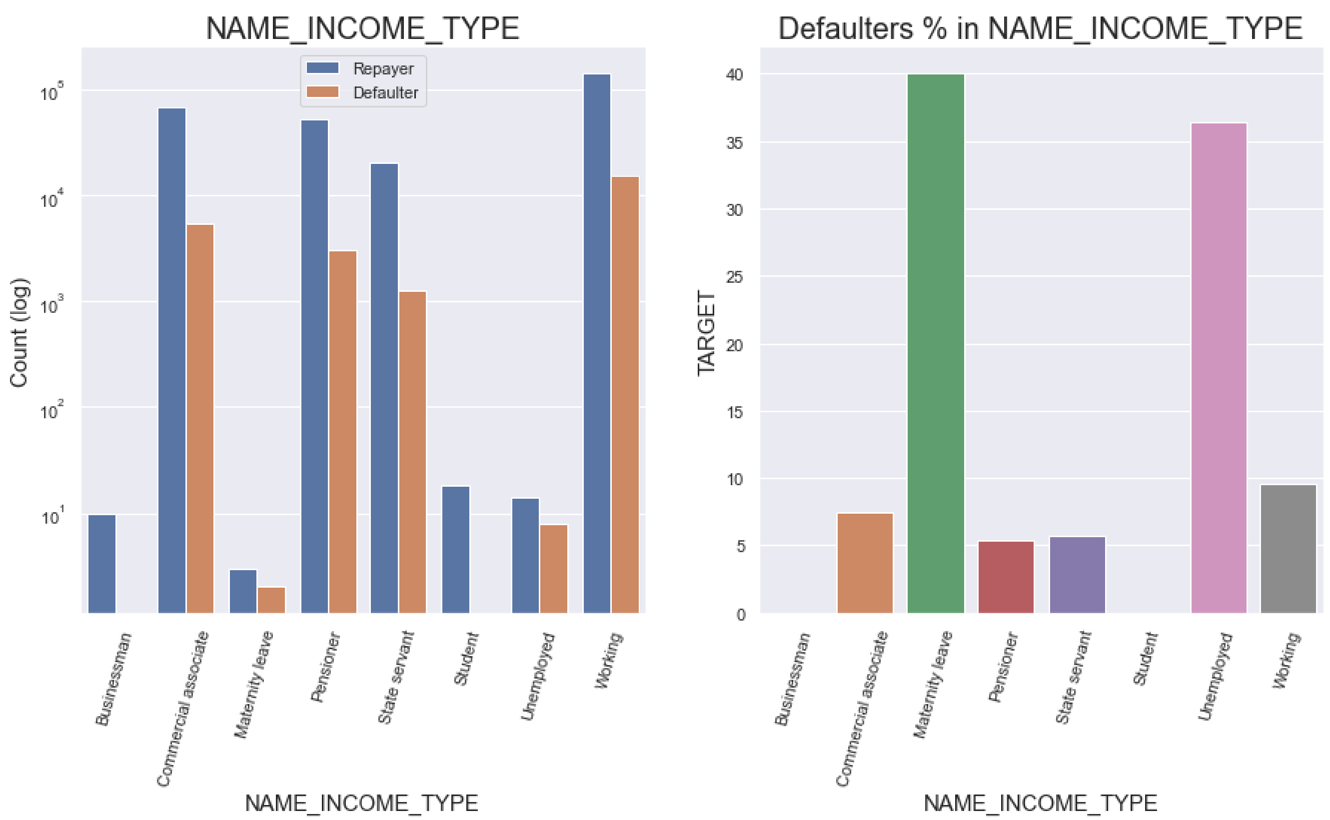
no correlation between owning real estate and violating loans

```
In [124]: #Gender vs Loan repayment status
univar(application_data, "CODE_GENDER", "TARGET", True, False, True)
```



10.2% men are defaulters against 7% women

```
In [125]: #income type vs Loan Repayment status
univar(application_data, "NAME_INCOME_TYPE", "TARGET", True, True, True)
```

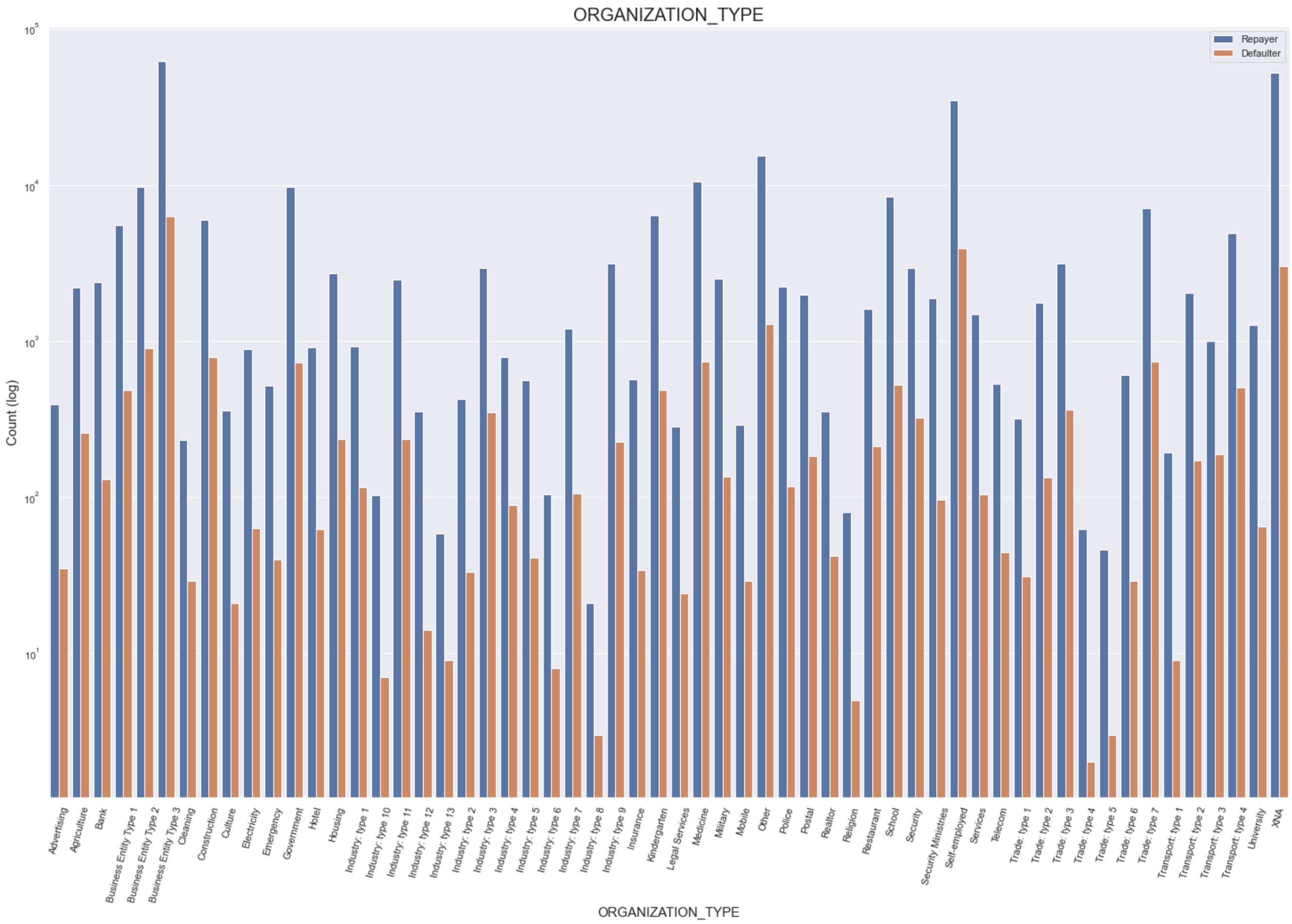


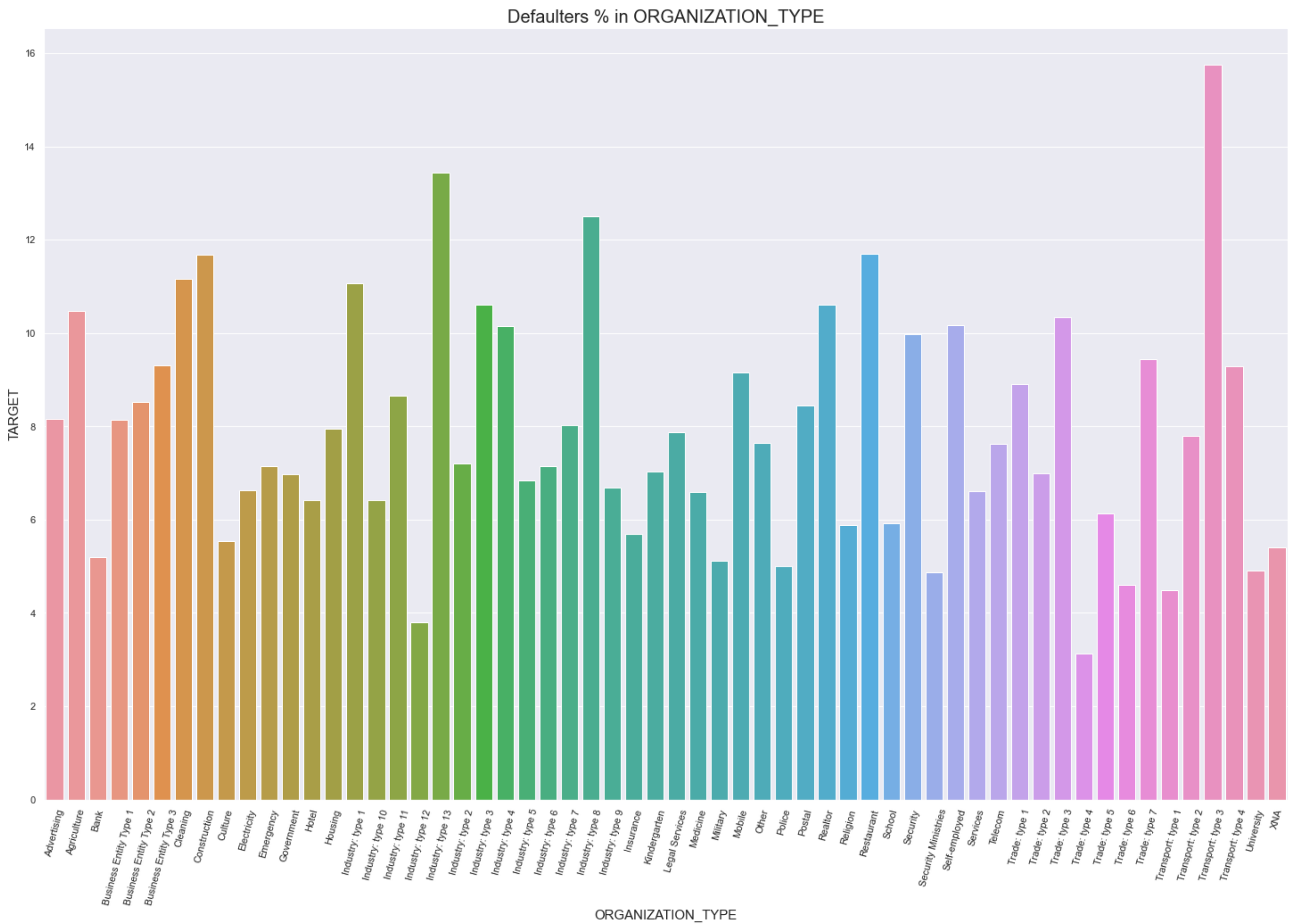
BUSINESSMEN and STUDENTS have no default record, thus loans can be safely given to these categories

Women on MATERNITYLEAVE have very high chance to default a loan

Obviously UNEMPLOYED also tend to violate a loan

In [126]:
```
# loanrepay status vs organization type
univar(application_data, "ORGANIZATION_TYPE", "TARGET", True, True, False)
```





Insights: Org type

- TradeType 4 and 5 are the safest to lend loans to

- Majority of people are from BUSINESS ENTITY TYPE 3

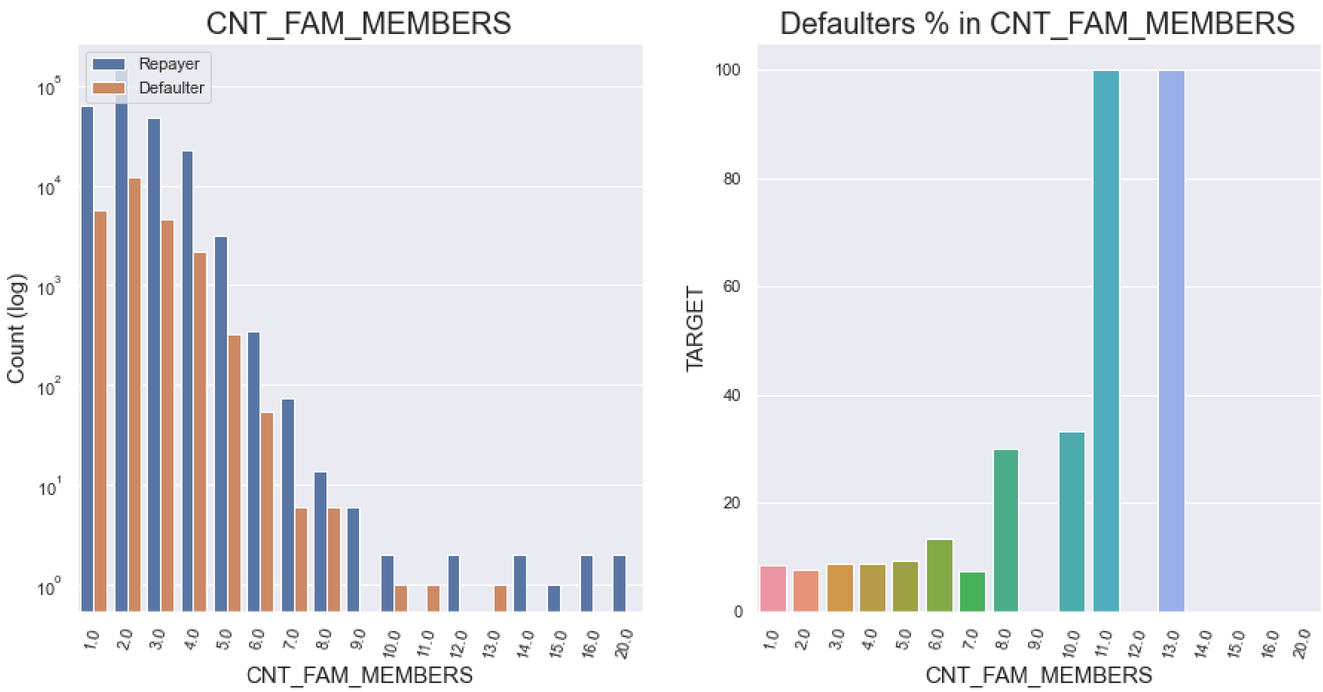
- people who are SELFEMPLOYED are risky to give loans, to counter the defaulting, higher interest can be imposed

In [127]:
```
# Region Rating vs loan repayment status
univar(application_data, "REGION_RATING_CLIENT", "TARGET", True, False, True)
```

People from REGION RATED 1 are safest to give loan to

People from REGION RATED 3 are most probable to default ~ 11.2-11.5%

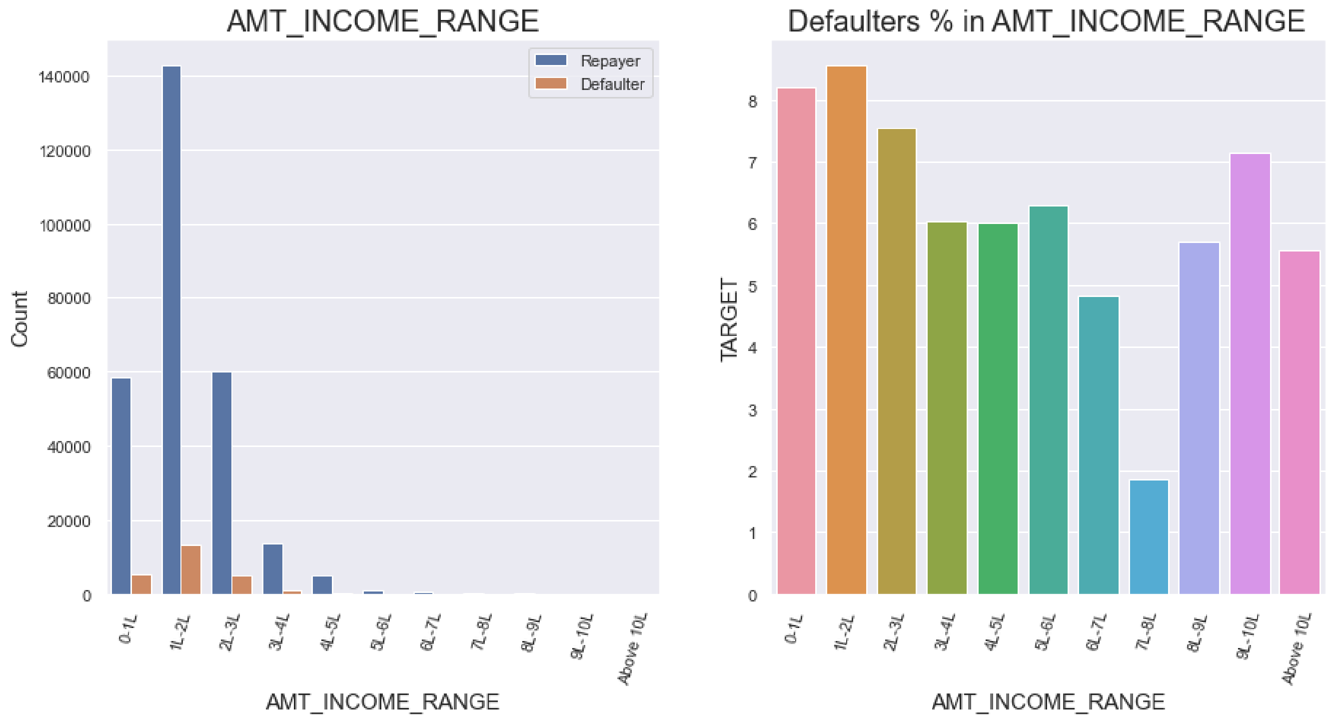
In [128]:
```
# number of family members vs loan repay status
univar(application_data, "CNT_FAM_MEMBERS", "TARGET", True, True, True)
```



most of the clients have family member count of the average, 1-4 people, representing nuclear family type in cities

people living in joint family of members>8 are riskiest to give loans to, which is obvious as more family members means more finances required

In [129]:
```
# Most Important, Income range vs Loan repay status
univar(application_data, "AMT_INCOME_RANGE", "TARGET", False, True, True)
```

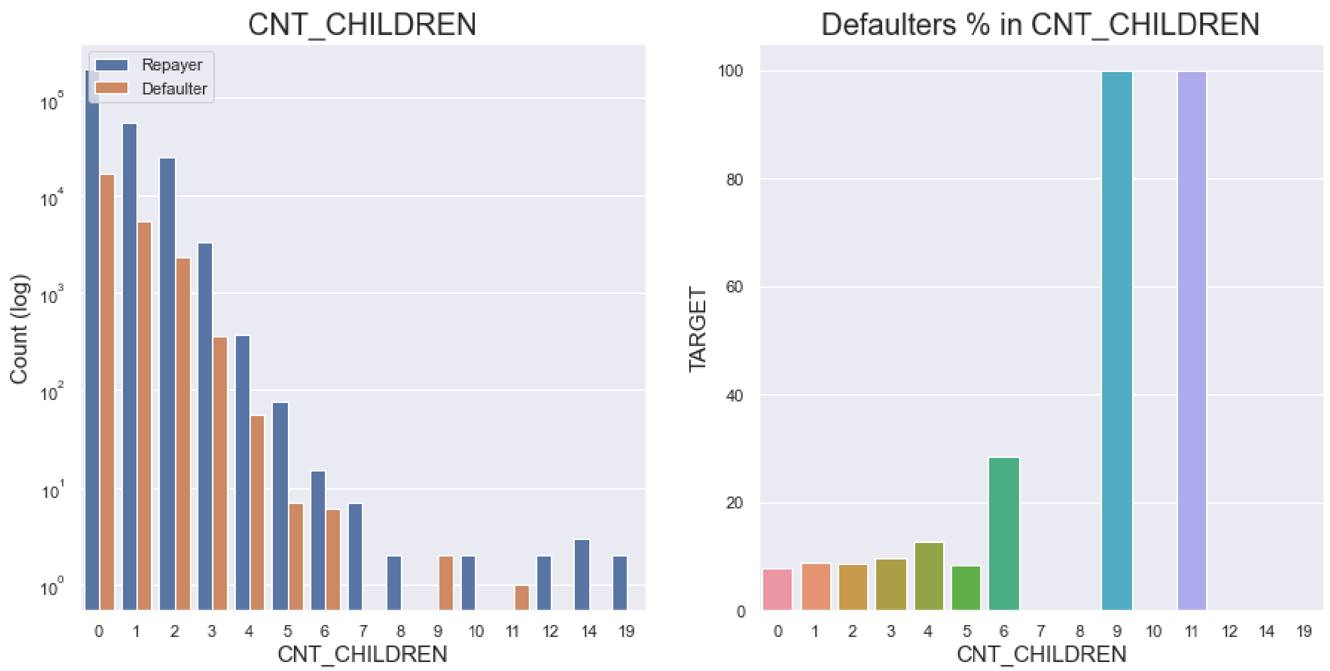


Most of the clients have income less that 3L,

clients of this range have high risk of defaulting

Whereas clients having income in range 7-8 lakh have least risk

In [130]:
```
#number of children vs loan repy status
univar(application_data, "CNT_CHILDREN", "TARGET", True, False, True)
```



This also follows the trend of more fam members=more risk

People of 9 and 11 children have 100% default rate

while majority of clients have no children

Merged df Analysis

In [131]:
```
loan_df=pd.merge(application_data, previous_appl, how="inner", on="SK_ID_CURR")
```

In [132]:
```
loan_df.head()
```

Out[132]:

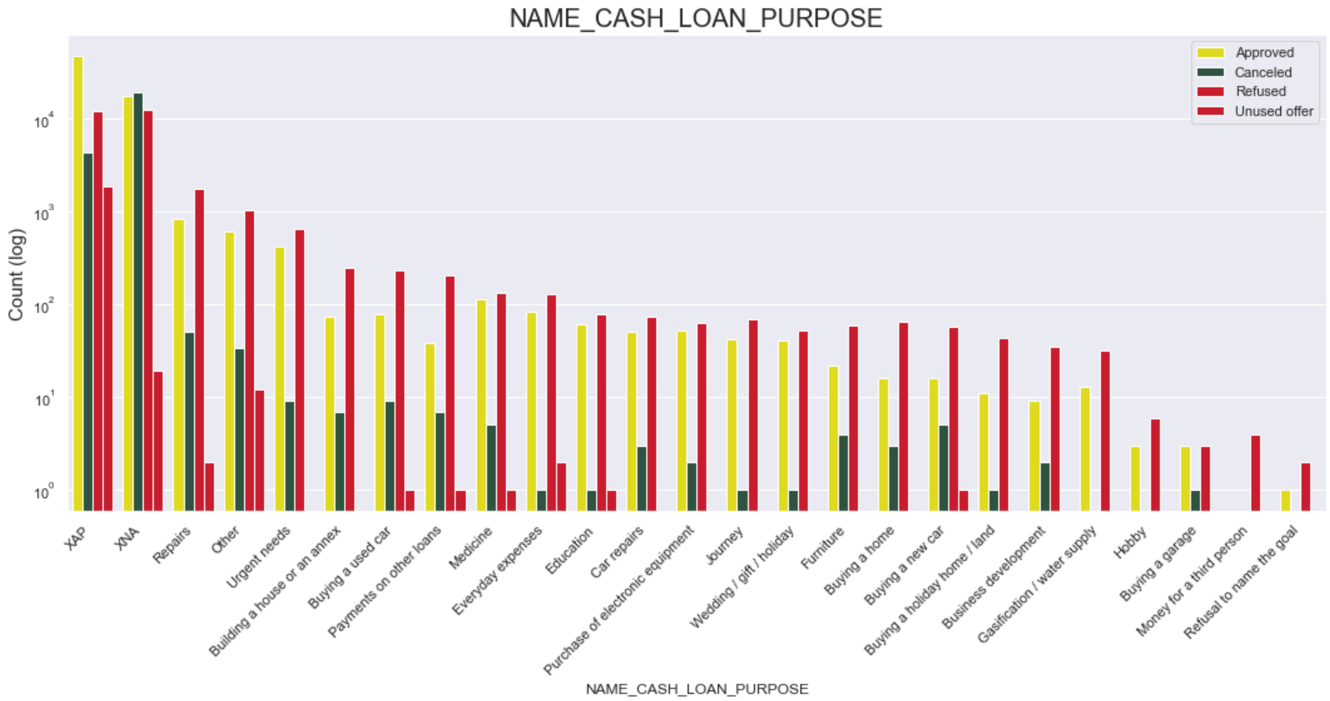
	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REAL
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100003	0	Cash loans	F	N	
3	100003	0	Cash loans	F	N	
4	100004	0	Revolving loans	M	Y	

5 rows × 107 columns

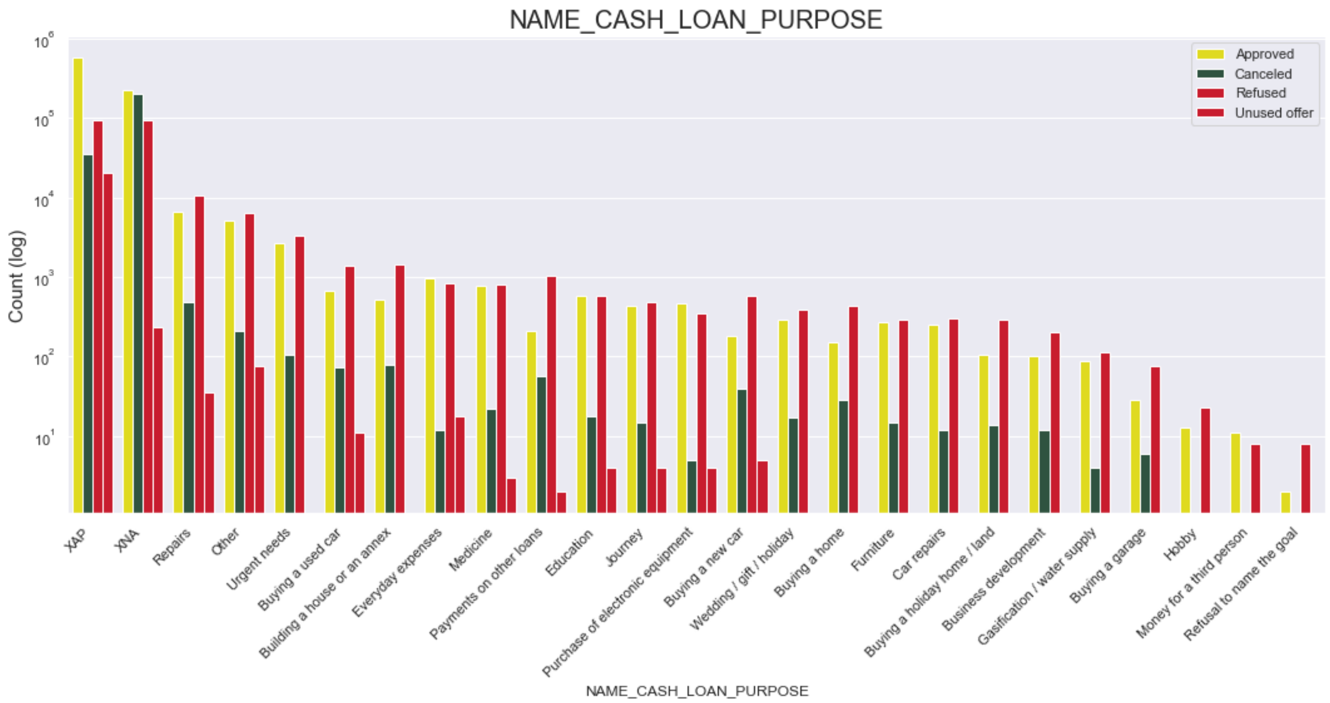
In [133]:
```
#dividing loan_df into 0 and 1 for repayer and defaulter

L1=loan_df[loan_df["TARGET"]==0]#Defaulter

L0=loan_df[loan_df["TARGET"]==1]#Repayer
```

In [134]:
```
#function for repetetive plots in univar categorical analysis
def univar_c_merge(col,df,hue,palette,ylog,figsize):
    plt.figure(figsize=figsize)
    ax=sns.countplot(x=col, data=df,hue= hue,palette= palette,order=df[col].value_counts


    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 [135]:
```
univar_c_merge("NAME_CASH_LOAN_PURPOSE", L0, "NAME_CONTRACT_STATUS", ["#fff800","#29594
```



In [136]:
```
univar_c_merge("NAME_CASH_LOAN_PURPOSE", L1, "NAME_CONTRACT_STATUS", ["#fff800","#29594
```



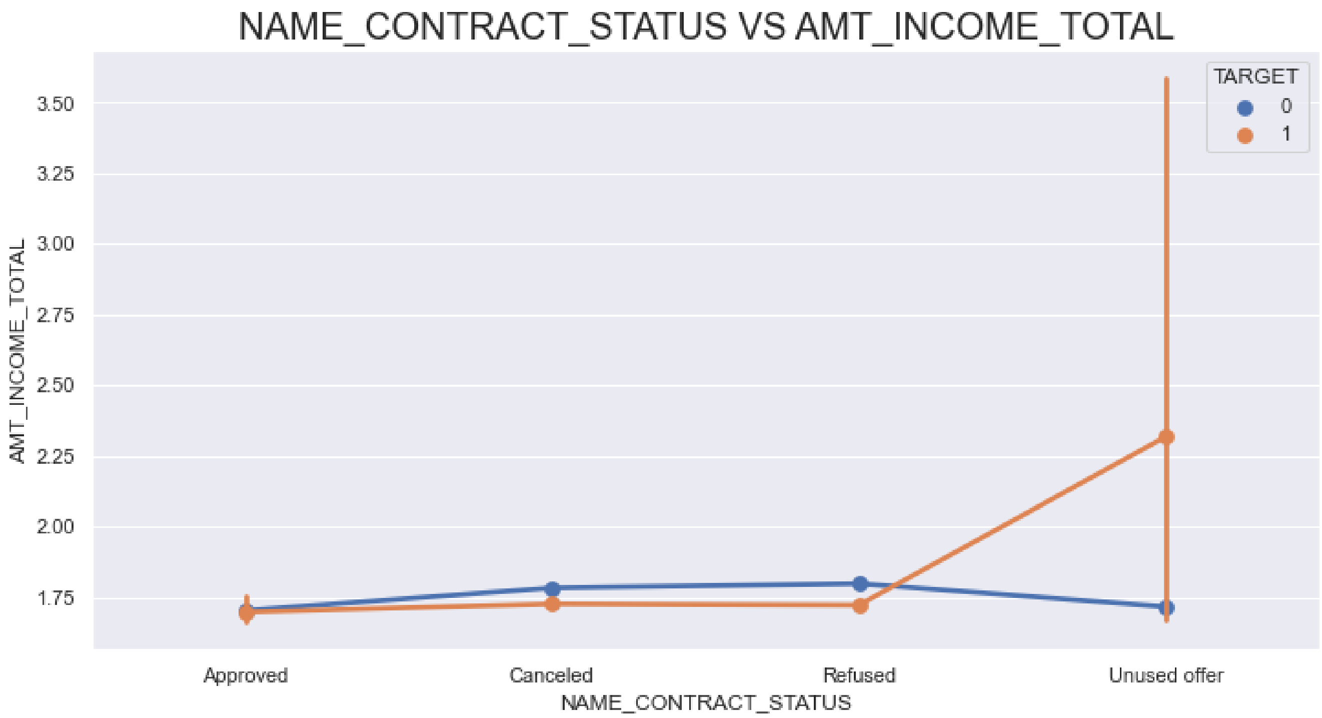
Insights

- People mostly dont mention loan purpose(high unknown values)
- Loans for repairs are majorly not given, meaning Bank considers this as a threat and bank either disapproves the loan, or imposes high interest rate. Due to higher iterest rate, the applicant withdraws

the application

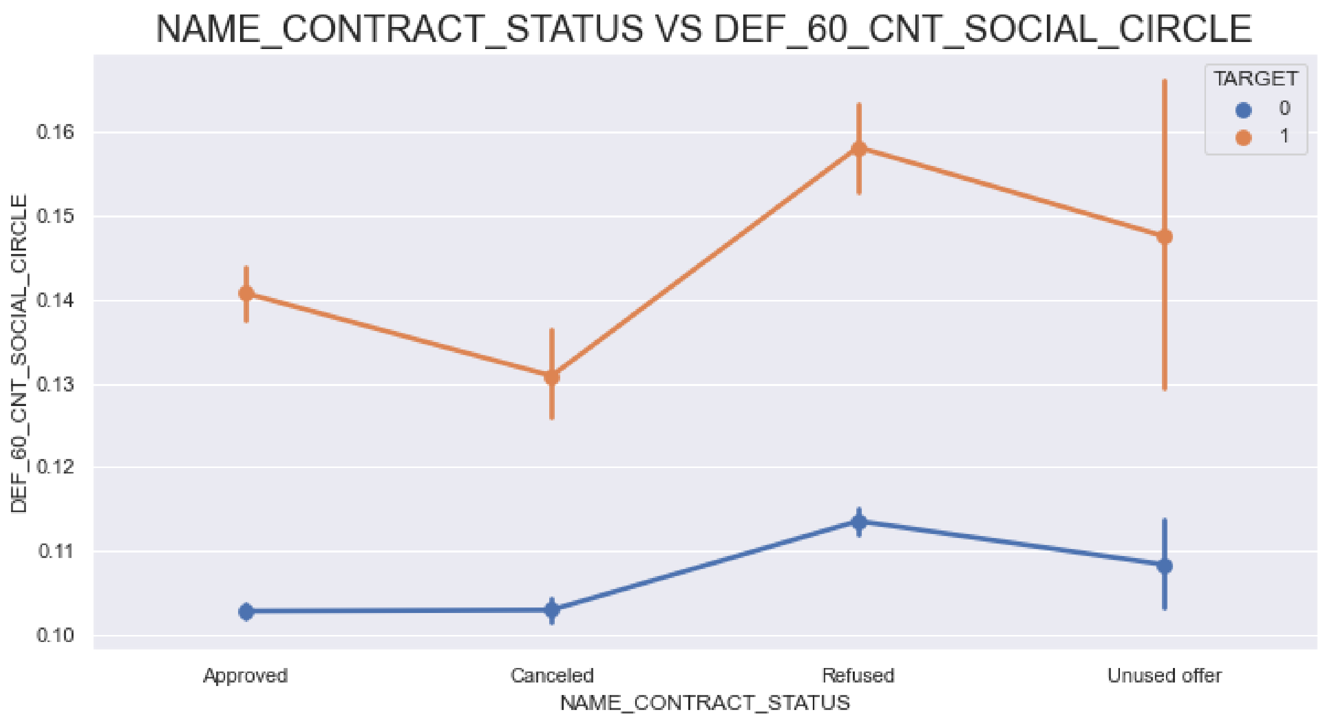
In [137]:
```
#function to plot pointplot
def pointplt(df, hue, x, y):
    plt.figure(figsize=(12,6))
    sns.pointplot(x=x,y=y, hue=hue, data=df)
    plt.title(x+" VS "+y, fontsize=20)
```

In [138]:
```
#relationship in income total and contant status
pointplt(loan_df, "TARGET", "NAME_CONTRACT_STATUS", "AMT_INCOME_TOTAL")
```



some people have defaulted after they have not used an earlier offer

In [139]:
```
#name contract vs being in social circle for 60 days
pointplt(loan_df,"TARGET","NAME_CONTRACT_STATUS",'DEF_60_CNT_SOCIAL_CIRCLE')
```



no insights obtained from this graph

In [143]:
```
#contract status vs loan repay status for opportunity
univar_c_merge("NAME_CONTRACT_STATUS",loan_df, "TARGET", ['y','r'], False,(14,8))
r=loan_df.groupby("NAME_CONTRACT_STATUS")["TARGET"]
df=pd.concat([r.value_counts(), round(r.value_counts(normalize=True).mul(100),2)], axis=
df["Percent"] = df['Percent'].astype(str)+"%"
df
```

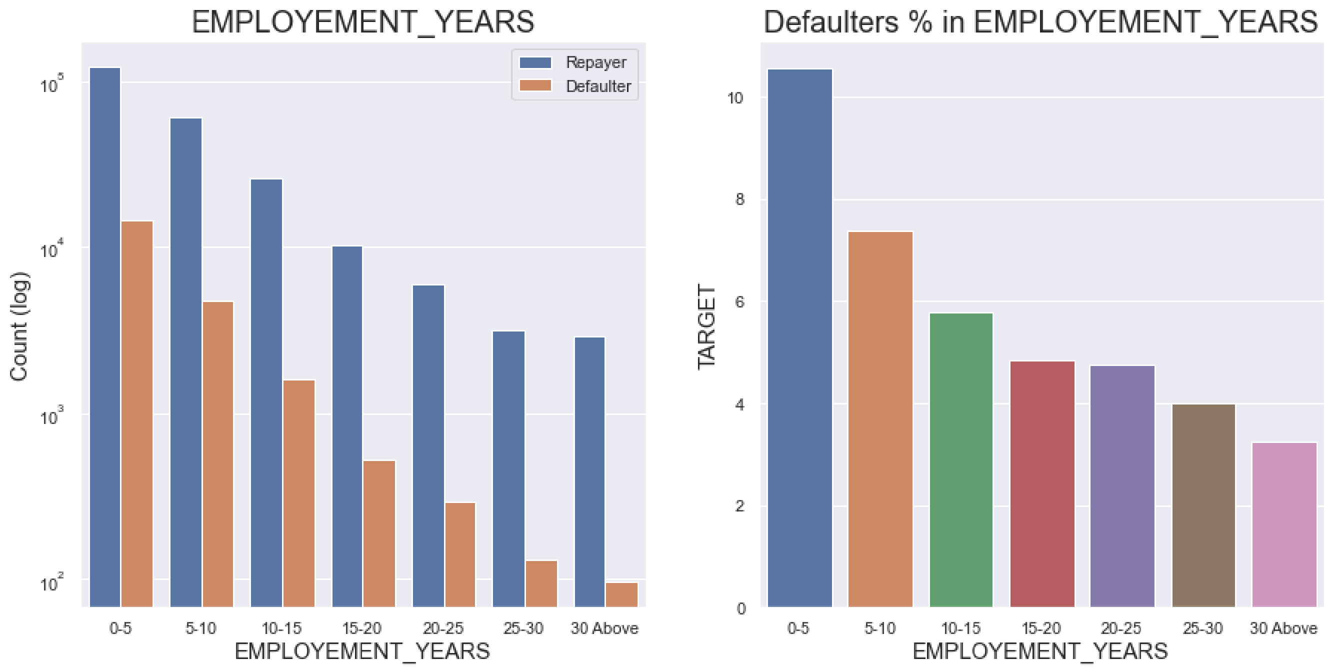
Out[143]:

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

80% of refused clients paid back the current loan

more the 90% of cancelled clients repayed the loan, this can be a business opportunity

```
In [144]: univar(application_data, "EMPLOYEMENT_YEARS", "TARGET", True, False, True)
```



People who have recently joined workforce tend to default loans

People abve 30 years working mostly repay loans

With number of employment increasing, tendency to default loans also decreases

Conclusion

After analyzing both the datasets, we can identify the following patterns and some inferences can be done

Attributes that determine the eligibility

- ORGANIZATION_TYPE: Trade Type 4 and 5 have less that 3% default rate
- AMT_INCOME_TOTAL: people earning more than 7L do not default loans
- DAYS_BIRTH: people above age of 50 do not default loans
- NAME_EDUCATION_TYPE: Having a degree dramatically reduces the chances of defaulting
- DAYS_EMPLOYED: people with 40+ working years have less that 1% default rates, the people who have recently become employed are risky to lend loans to

Business Opportunities with higher interest rates

- AMT_INCOME: As most of the population has income lesser than 3L, loans can be given to them at higher interest rates, meaning a business opportunity
- people getting loan in ranges of 3-6L have higher chance of defaulting, thus, imposing more interest should be better(AMT_INCOME)

- People living in rented houses or with parents tend to take loans, so offering these loans should be beneficial(NAME_HOUSING_TYPE)
- People having 4-9 children have highest loan default rates, so higher interet rates can pull in business(CNT_CHILDREN, CNT_FAM_MEMBERS)

Avoid at any costs!!

- People taking loans for "REPAIRS" purpose in "NAME_CASH_PURPOSE" have highest default rates. Bank avoides giving them loans and SHOULD avoid in future as well
- As the loan amount reaches 25L, defaulter increase, so maximum caution should be excercised while lending such huge amount of loan
- Also it is pretty obvious that UNEMPLOYED client mostly defaults the loan, thus not refusing such client is good for the bank
- People with children ~ 9-11, should be avoided at all costs, as they are sure to default.

Suggestions

- Some refused clients have repayed past loans, so they should be reconsidered for giving loans
- People from REGION_RATING = 1 and 2 should be given more weightage in giving loans, as they do not tend to default
- Reconsider while giving loan to women on MATERNITY LEAVE, as they also tend to default more
- Students have no chance of defaulting the loans, thus they should be given more priority than others. Also if possible, interest rates can be reduced for STUDENTS, as this can help us rake in more clients. Same can be said for BUSINESSMEN

In [ ]:

```

```