II. Import Libraries and set required parameters

```
# Import libraries
import numpy as np
print('numpy version\t:', np.__version__)
import pandas as pd
print('pandas version\t:', pd.__version__)
import matplotlib.pyplot as plt
import seaborn as sns
print('seaborn version\t:', sns.__version__)
from scipy import stats

import os

pd.set_option('display.max_columns', 200) # to display all the columns
pd.set_option('display.max_rows',150) # to display all rows of df series
pd.options.display.float_format = '{:.4f}'.format #set it to convert scientific noations such as 4.225108e+11 to 422510842796.00

import warnings
warnings.filterwarnings('ignore') # if there are any warning due to version mismatch, it will be ignored

import random
```

```
numpy version   : 1.21.6
pandas version  : 1.3.5
seaborn version : 0.11.2
```

1. Data Importing

```
# # Sample data to overcome Memory Error
# # Less RAM: Reduce the data: It's completely fine to take a sample of the data to work on this case study
# # Random Sampling to get a random sample of data from the complete data
# filename = "application_data.csv"# This file is available is the same location as the jupyter notebook

# # Count the number of rows in my file
# num_lines = sum(1 for i in open(filename))
# # The number of rows that I wanted to load
# size = num_lines//2

# # Create a random indices between these two numbers

# random.seed(10)
# skip_id = random.sample(range(1, num_lines), num_lines-size)

# df_app = pd.read_csv(filename, skiprows = skip_id)
```

```
# read data
df_app = pd.read_csv('application_data.csv')
```

Get some insights of data

```
# get shape of data (rows, columns)
print(df_app.shape)
```

```
(17474, 122)
```

```
df_app.dtypes.value_counts()
```

```
float64    85
int64      21
object     16
dtype: int64
```

```
# get some insights of data
df_app.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AM
0	100002	1	Cash loans	M	N	Y	0	
1	100003	0	Cash loans	F	N	N	0	
2	100004	0	Revolving loans	M	Y	Y	0	

```
df_app.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17474 entries, 0 to 17473
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(85), int64(21), object(16)
memory usage: 16.3+ MB
```

```
# get the count, size and Unique value in each column of application data
df_app.agg(['count','size','nunique'])
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDR
count	17474	17474	17474	17474	17474	17474	174
size	17474	17474	17474	17474	17474	17474	174
nunique	17474	2	2	2	2	2	

2. Data Quality Check and Missing Values

2.a. Find the percentage of missing values of the columns

```
# funcion to get null value
def column_wise_null_percentage(df):
    output = round(df.isnull().sum()/len(df.index)*100,2)
    return output
```

```
# get missign values of all columns
NA_col = column_wise_null_percentage(df_app)
NA_col
```

```
SK_ID_CURR                      0.0000
TARGET                          0.0000
NAME_CONTRACT_TYPE              0.0000
CODE_GENDER                     0.0000
FLAG_OWN_CAR                    0.0000
FLAG_OWN_REALTY                 0.0000
CNT_CHILDREN                    0.0000
AMT_INCOME_TOTAL                0.0000
AMT_CREDIT                      0.0000
AMT_ANNUITY                     0.0000
AMT_GOODS_PRICE                 0.0700
NAME_TYPE_SUITE                 0.4000
NAME_INCOME_TYPE                0.0000
NAME_EDUCATION_TYPE             0.0000
NAME_FAMILY_STATUS              0.0000
NAME_HOUSING_TYPE               0.0000
REGION_POPULATION_RELATIVE      0.0000
DAYS_BIRTH                      0.0000
DAYS_EMPLOYED                   0.0000
DAYS_REGISTRATION               0.0000
DAYS_ID_PUBLISH                 0.0000
OWN_CAR_AGE                    66.0700
FLAG_MOBIL                      0.0000
FLAG_EMP_PHONE                  0.0000
FLAG_WORK_PHONE                 0.0000
FLAG_CONT_MOBILE                0.0000
FLAG_PHONE                      0.0000
FLAG_EMAIL                      0.0000
OCCUPATION_TYPE                31.1800
CNT_FAM_MEMBERS                 0.0000
REGION_RATING_CLIENT            0.0000
REGION_RATING_CLIENT_W_CITY     0.0000
WEEKDAY_APPR_PROCESS_START      0.0000
HOUR_APPR_PROCESS_START         0.0000
REG_REGION_NOT_LIVE_REGION      0.0000
REG_REGION_NOT_WORK_REGION      0.0000
```

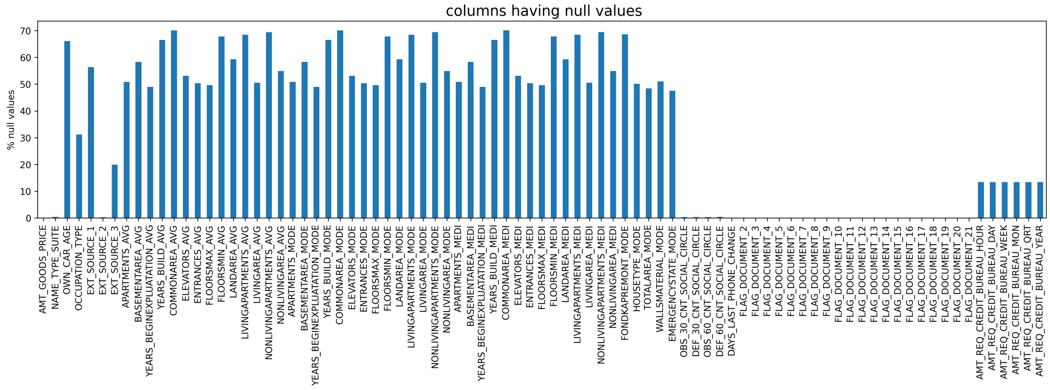
```
LIVE_REGION_NOT_WORK_REGION     0.0000
REG_CITY_NOT_LIVE_CITY          0.0000
REG_CITY_NOT_WORK_CITY          0.0000
LIVE_CITY_NOT_WORK_CITY         0.0000
ORGANIZATION_TYPE               0.0000
EXT_SOURCE_1                   56.3400
EXT_SOURCE_2                    0.2700
EXT_SOURCE_3                   19.8700
APARTMENTS_AVG                 50.8100
BASEMENTAREA_AVG               58.3400
YEARS_BEGINEXPLUATATION_AVG    48.9800
YEARS_BUILD_AVG                66.5000
COMMONAREA_AVG                 70.0600
ELEVATORS_AVG                  53.1400
ENTRANCES_AVG                  50.3400
FLOORSMAX_AVG                  49.6600
FLOORSMIN_AVG                  67.8300
LANDAREA_AVG                   59.2700
LIVINGAPARTMENTS_AVG           68.4600
LIVINGAREA_AVG                 50.5000
NONLIVINGAPARTMENTS_AVG        69.4500
NONLIVINGAREA_AVG              54.9300
```

```
# identify columns only with null values
NA_col = NA_col[NA_col>0]
NA_col
```

```
AMT_GOODS_PRICE                 0.0700
NAME_TYPE_SUITE                 0.4000
OWN_CAR_AGE                    66.0700
OCCUPATION_TYPE                31.1800
EXT_SOURCE_1                   56.3400
EXT_SOURCE_2                    0.2700
EXT_SOURCE_3                   19.8700
APARTMENTS_AVG                 50.8100
BASEMENTAREA_AVG               58.3400
YEARS_BEGINEXPLUATATION_AVG    48.9800
YEARS_BUILD_AVG                66.5000
COMMONAREA_AVG                 70.0600
ELEVATORS_AVG                  53.1400
ENTRANCES_AVG                  50.3400
FLOORSMAX_AVG                  49.6600
FLOORSMIN_AVG                  67.8300
LANDAREA_AVG                   59.2700
LIVINGAPARTMENTS_AVG           68.4600
LIVINGAREA_AVG                 50.5000
NONLIVINGAPARTMENTS_AVG        69.4500
NONLIVINGAREA_AVG              54.9300
APARTMENTS_MODE                50.8100
BASEMENTAREA_MODE              58.3400
YEARS_BEGINEXPLUATATION_MODE   48.9800
YEARS_BUILD_MODE               66.5000
COMMONAREA_MODE                70.0600
ELEVATORS_MODE                 53.1400
ENTRANCES_MODE                 50.3400
FLOORSMAX_MODE                 49.6600
FLOORSMIN_MODE                 67.8300
LANDAREA_MODE                  59.2700
LIVINGAPARTMENTS_MODE          68.4600
LIVINGAREA_MODE                50.5000
NONLIVINGAPARTMENTS_MODE       69.4500
NONLIVINGAREA_MODE             54.9300
APARTMENTS_MEDI                50.8100
BASEMENTAREA_MEDI              58.3400
YEARS_BEGINEXPLUATATION_MEDI   48.9800
YEARS_BUILD_MEDI               66.5000
COMMONAREA_MEDI                70.0600
ELEVATORS_MEDI                 53.1400
ENTRANCES_MEDI                 50.3400
FLOORSMAX_MEDI                 49.6600
FLOORSMIN_MEDI                 67.8300
LANDAREA_MEDI                  59.2700
LIVINGAPARTMENTS_MEDI          68.4600
LIVINGAREA_MEDI                50.5000
NONLIVINGAPARTMENTS_MEDI       69.4500
NONLIVINGAREA_MEDI             54.9300
FONDKAPREMONT_MODE             68.6300
HOUSETYPE_MODE                 50.1500
TOTALAREA_MODE                 48.3700
WALLSMATERIAL_MODE             50.9800
EMERGENCYSTATE_MODE            47.5000
OBS_30_CNT_SOCIAL_CIRCLE        0.3800
DEF_30_CNT_SOCIAL_CIRCLE        0.3800
OBS_60_CNT_SOCIAL_CIRCLE        0.3800
DEF_60_CNT_SOCIAL_CIRCLE        0.3900
```

```
# grafical representation of columns having % null values
plt.figure(figsize= (20,4),dpi=300)
```

```
NA_col.plot(kind = 'bar')
plt.title (' columns having null values')
plt.ylabel('% null values')
plt.show()
# plt.savefig('filename.png', dpi=300)
```



2.b. Identify and remove columns with high missing percentage (>50%)

```
# Get the column with null values more than 50%
NA_col_50 = NA_col[NA_col>50]
print("Number of columns having null value more than 50% :", len(NA_col_50.index))
print(NA_col_50)
```

```
Number of columns having null value more than 50% : 41
OWN_CAR_AGE                 66.0700
EXT_SOURCE_1                56.3400
APARTMENTS_AVG              50.8100
BASEMENTAREA_AVG            58.3400
YEARS_BUILD_AVG             66.5000
COMMONAREA_AVG              70.0600
ELEVATORS_AVG               53.1400
ENTRANCES_AVG               50.3400
FLOORSMIN_AVG               67.8300
LANDAREA_AVG                59.2700
LIVINGAPARTMENTS_AVG        68.4600
LIVINGAREA_AVG              50.5000
NONLIVINGAPARTMENTS_AVG     69.4500
NONLIVINGAREA_AVG           54.9300
APARTMENTS_MODE             50.8100
BASEMENTAREA_MODE           58.3400
YEARS_BUILD_MODE            66.5000
COMMONAREA_MODE             70.0600
ELEVATORS_MODE              53.1400
ENTRANCES_MODE              50.3400
FLOORSMIN_MODE              67.8300
LANDAREA_MODE               59.2700
LIVINGAPARTMENTS_MODE       68.4600
LIVINGAREA_MODE             50.5000
NONLIVINGAPARTMENTS_MODE    69.4500
NONLIVINGAREA_MODE          54.9300
APARTMENTS_MEDI             50.8100
BASEMENTAREA_MEDI           58.3400
YEARS_BUILD_MEDI            66.5000
COMMONAREA_MEDI             70.0600
ELEVATORS_MEDI              53.1400
ENTRANCES_MEDI              50.3400
FLOORSMIN_MEDI              67.8300
LANDAREA_MEDI               59.2700
LIVINGAPARTMENTS_MEDI       68.4600
LIVINGAREA_MEDI             50.5000
NONLIVINGAPARTMENTS_MEDI    69.4500
NONLIVINGAREA_MEDI          54.9300
FONDKAPREMONT_MODE          68.6300
HOUSETYPE_MODE              50.1500
WALLSMATERIAL_MODE          50.9800
dtype: float64
```

- Droped all columns from Dataframe for which missing value percentage are more than 50%.

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

```
# removed 41 columns having null percentage more than 50%.
df_app = df_app.drop(NA_col_50.index, axis =1)
df_app.shape
```

```
(17474, 81)
```

2.c. identify columns with less missing missing values (<15%)

```
# Get columns having <15% null values
NA_col_15 = NA_col[NA_col<15]
print("Number of columns having null value less than 15% :", len(NA_col_15.index))
print(NA_col_15)
```

```
Number of columns having null value less than 15% : 34
AMT_GOODS_PRICE               0.0700
NAME_TYPE_SUITE               0.4000
EXT_SOURCE_2                  0.2700
OBS_30_CNT_SOCIAL_CIRCLE      0.3800
DEF_30_CNT_SOCIAL_CIRCLE      0.3800
OBS_60_CNT_SOCIAL_CIRCLE      0.3800
DEF_60_CNT_SOCIAL_CIRCLE      0.3900
DAYS_LAST_PHONE_CHANGE        0.0100
FLAG_DOCUMENT_2               0.0100
FLAG_DOCUMENT_3               0.0100
FLAG_DOCUMENT_4               0.0100
FLAG_DOCUMENT_5               0.0100
FLAG_DOCUMENT_6               0.0100
FLAG_DOCUMENT_7               0.0100
FLAG_DOCUMENT_8               0.0100
FLAG_DOCUMENT_9               0.0100
FLAG_DOCUMENT_10              0.0100
FLAG_DOCUMENT_11              0.0100
FLAG_DOCUMENT_12              0.0100
FLAG_DOCUMENT_13              0.0100
FLAG_DOCUMENT_14              0.0100
FLAG_DOCUMENT_15              0.0100
FLAG_DOCUMENT_16              0.0100
FLAG_DOCUMENT_17              0.0100
FLAG_DOCUMENT_18              0.0100
FLAG_DOCUMENT_19              0.0100
FLAG_DOCUMENT_20              0.0100
FLAG_DOCUMENT_21              0.0100
AMT_REQ_CREDIT_BUREAU_HOUR   13.4000
AMT_REQ_CREDIT_BUREAU_DAY    13.4000
AMT_REQ_CREDIT_BUREAU_WEEK   13.4000
AMT_REQ_CREDIT_BUREAU_MON    13.4000
AMT_REQ_CREDIT_BUREAU_QRT    13.4000
AMT_REQ_CREDIT_BUREAU_YEAR   13.4000
dtype: float64
```

```
NA_col_15.index
```

```
Index(['AMT_GOODS_PRICE', 'NAME_TYPE_SUITE', 'EXT_SOURCE_2',
       '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_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',
       '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'],
      dtype='object')
```
```

- The columns having null values less than 15% are,

> 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE', 'EXT_SOURCE_2','OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE','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'

- These columns shall be imputed with suitable values which shall be explained subsequently.

```
# understand the insight of missing columns having <15% null values
df_app[NA_col_15.index].describe()
```

	AMT_GOODS_PRICE	EXT_SOURCE_2	OBS_30_CNT_SOCIAL_CIRCLE	DEF_30_CNT_SOCIAL_CIRCLE	OBS_60_CNT_S(
count	17462.0000	17426.0000	17407.0000	17407.0000	
mean	540701.5683	0.5152	1.4218	0.1482	
std	371331.5498	0.1899	2.3004	0.4520	
min	45000.0000	0.0000	0.0000	0.0000	
25%	238500.0000	0.3945	0.0000	0.0000	
50%	450000.0000	0.5650	0.0000	0.0000	
75%	684000.0000	0.6642	2.0000	0.0000	
max	4050000.0000	0.8550	25.0000	6.0000	

```
# identify unique values in the colums having <15% null value
df_app[NA_col_15.index].nunique().sort_values(ascending=False)
```

```
EXT_SOURCE_2                  15675
DAYS_LAST_PHONE_CHANGE         3009
AMT_GOODS_PRICE                 404
OBS_60_CNT_SOCIAL_CIRCLE         24
OBS_30_CNT_SOCIAL_CIRCLE         23
AMT_REQ_CREDIT_BUREAU_MON        18
AMT_REQ_CREDIT_BUREAU_YEAR       15
AMT_REQ_CREDIT_BUREAU_QRT         9
DEF_30_CNT_SOCIAL_CIRCLE          7
NAME_TYPE_SUITE                   7
DEF_60_CNT_SOCIAL_CIRCLE          6
AMT_REQ_CREDIT_BUREAU_WEEK        6
AMT_REQ_CREDIT_BUREAU_DAY         6
AMT_REQ_CREDIT_BUREAU_HOUR        3
FLAG_DOCUMENT_15                  2
FLAG_DOCUMENT_21                  2
FLAG_DOCUMENT_20                  2
FLAG_DOCUMENT_19                  2
FLAG_DOCUMENT_18                  2
FLAG_DOCUMENT_17                  2
FLAG_DOCUMENT_16                  2
FLAG_DOCUMENT_14                  2
FLAG_DOCUMENT_13                  2
FLAG_DOCUMENT_9                   2
FLAG_DOCUMENT_8                   2
FLAG_DOCUMENT_7                   2
FLAG_DOCUMENT_6                   2
FLAG_DOCUMENT_5                   2
FLAG_DOCUMENT_4                   2
FLAG_DOCUMENT_3                   2
FLAG_DOCUMENT_11                  2
FLAG_DOCUMENT_2                   1
FLAG_DOCUMENT_12                  1
FLAG_DOCUMENT_10                  1
dtype: int64
```

- **For analysis of imputation selecetd 7 varibles.**
  Continiuous variables:

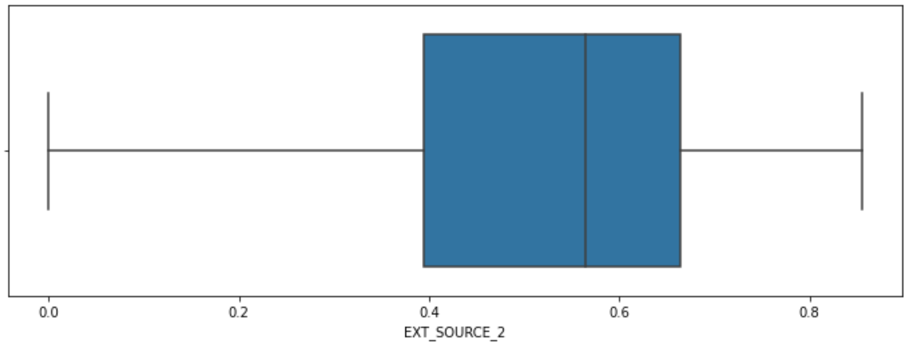
  > `'EXT_SOURCE_2','AMT_GOODS_PRICE'`

  Categorical variables:

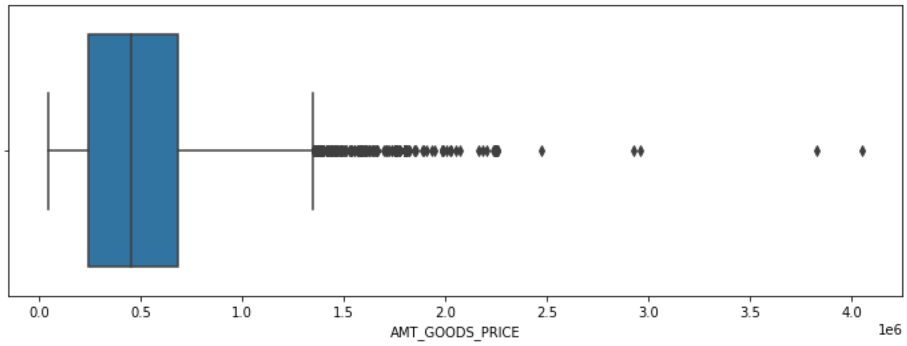
```
> 'OBS_30_CNT_SOCIAL_CIRCLE','OBS_60_CNT_SOCIAL_CIRCLE','DEF_60_CNT_SOCIAL_CIRCLE','DEF_30_CNT_SOCIAL_CIRCLE','NAME_TYPE_SU
```

Continous variable:

```
# Box plot for continuious variable
plt.figure(figsize=(12,4))
sns.boxplot(df_app['EXT_SOURCE_2'])
plt.show()
```




```
plt.figure(figsize=(12,4))
sns.boxplot(df_app['AMT_GOODS_PRICE'])
plt.show()
```



Inference from box plot:

- for 'EXT_SOURCE_2' there is no outliers present. And there is no significant diffence observed between mean and median. However data look to be right skewed. So missing values can be imputed with median value: 0.565
- for 'AMT_GOODS_PRICE' there is significant number of outlier present in the data. SO data should be imputed with median value: 450000

Categorical variables:

```
# identify maximum frequency values
print('Maximum Frequncy categorical values are,')
print('NAME_TYPE_SUITE: ',df_app['NAME_TYPE_SUITE'].mode()[0])
print('OBS_30_CNT_SOCIAL_CIRCLE:', df_app['OBS_30_CNT_SOCIAL_CIRCLE'].mode()[0])
print('DEF_30_CNT_SOCIAL_CIRCLE:', df_app['DEF_30_CNT_SOCIAL_CIRCLE'].mode()[0])
print('OBS_60_CNT_SOCIAL_CIRCLE:', df_app['OBS_60_CNT_SOCIAL_CIRCLE'].mode()[0])
print('DEF_60_CNT_SOCIAL_CIRCLE:', df_app['DEF_60_CNT_SOCIAL_CIRCLE'].mode()[0])
```

```
Maximum Frequncy categorical values are,
NAME_TYPE_SUITE:  Unaccompanied
OBS_30_CNT_SOCIAL_CIRCLE: 0.0
DEF_30_CNT_SOCIAL_CIRCLE: 0.0
OBS_60_CNT_SOCIAL_CIRCLE: 0.0
DEF_60_CNT_SOCIAL_CIRCLE: 0.0
```

For categorical vriable the value which should be imputed with maximum in frequency.
So the value to be imputed are:
NAME_TYPE_SUITE: Unaccompanied
OBS_30_CNT_SOCIAL_CIRCLE: 0.0
DEF_30_CNT_SOCIAL_CIRCLE: 0.0

OBS_60_CNT_SOCIAL_CIRCLE: 0.0

DEF_60_CNT_SOCIAL_CIRCLE: 0.0

```
# Remove unwanted columns from application dataset for better analysis.

unwanted=['FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE','FLAG_PHONE', 'FLAG_EMAIL',
          'REGION_RATING_CLIENT','REGION_RATING_CLIENT_W_CITY','FLAG_EMAIL','CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT',
          'REGION_RATING_CLIENT_W_CITY','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','EXT_SOURCE_2','EXT_SOURCE_3','YEARS_BEGINEXPLUATATION_AVG','FLOORSMAX_AVG','YEARS_BEGINEXPLUATATION_MODE',
          'FLOORSMAX_MODE','YEARS_BEGINEXPLUATATION_MEDI','FLOORSMAX_MEDI','TOTALAREA_MODE','EMERGENCYSTATE_MODE']

df_app.drop(labels=unwanted,axis=1,inplace=True)
```

```
df_app.shape
```

```
(17474, 42)
```

```
df_app.head()
```

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

There are some columns where the value is mentioned as 'XNA' which means 'Not Available'. So we have to find the number of rows and columns.

```
# For Code Gender column

print('CODE_GENDER: ',df_app['CODE_GENDER'].unique())
print('No of values: ',df_app[df_app['CODE_GENDER']=='XNA'].shape[0])

XNA_count = df_app[df_app['CODE_GENDER']=='XNA'].shape[0]
per_XNA = round(XNA_count/len(df_app.index)*100,3)

print('% of XNA Values:',  per_XNA)

print('maximum frequency data :', df_app['CODE_GENDER'].describe().top)
```

```
CODE_GENDER:  ['M' 'F']
No of values:  0
% of XNA Values: 0.0
maximum frequency data : F
```

Since, Female is having the majority and only 2 rows are having XNA values, we can impute those with Gender 'F' as there will be no impact on the dataset. Also there will no impact if we drop those rows.

```
# Dropping the XNA value in column 'CODE_GENDER' with "F" for the dataset

df_app = df_app.drop(df_app.loc[df_app['CODE_GENDER']=='XNA'].index)
df_app[df_app['CODE_GENDER']=='XNA'].shape
```

```
(0, 42)
```

```
# For Organization column
print('No of XNA values: ', df_app[df_app['ORGANIZATION_TYPE']=='XNA'].shape[0])

XNA_count = df_app[df_app['ORGANIZATION_TYPE']=='XNA'].shape[0]
per_XNA = round(XNA_count/len(df_app.index)*100,3)
```

```
print('% of XNA Values:',  per_XNA)

df_app['ORGANIZATION_TYPE'].describe()
```

```
No of XNA values:  3127
% of XNA Values: 17.895
count                      17474
unique                        58
top       Business Entity Type 3
freq                        3843
Name: ORGANIZATION_TYPE, dtype: object
```

So, for column 'ORGANIZATION_TYPE', we have total count of 153755 rows of which 27737 rows are having 'XNA' values. Which means 18% of the column is having this values.

```
# # Dropping the rows have 'XNA' values in the organization type column

# df_app = df_app.drop(df_app.loc[df_app['ORGANIZATION_TYPE']=='XNA'].index)
# df_app[df_app['ORGANIZATION_TYPE']=='XNA'].shape
```

2.d. Check the data type of all the columns and changed the data type.

```
df_app.head()
```

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

```
# Casting variable into numeric in the dataset

numeric_columns=['TARGET','CNT_CHILDREN','AMT_INCOME_TOTAL','AMT_CREDIT','AMT_ANNUITY','REGION_POPULATION_RELATIVE',
                 'DAYS_BIRTH','DAYS_EMPLOYED','DAYS_REGISTRATION','DAYS_ID_PUBLISH','HOUR_APPR_PROCESS_START',
                 'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY','REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY',
                'DAYS_LAST_PHONE_CHANGE']

df_app[numeric_columns]=df_app[numeric_columns].apply(pd.to_numeric)
df_app.head(5)
```

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

Following age/days columns are having -ve value, which needs to converted to +ve value.

```
'DAYS_BIRTH','DAYS_EMPLOYED','DAYS_REGISTRATION','DAYS_ID_PUBLISH','DAYS_LAST_PHONE_CHANGE',
```

```
# Converting '-ve' values into '+ve' Values
df_app['DAYS_BIRTH'] = df_app['DAYS_BIRTH'].abs()
df_app['DAYS_EMPLOYED'] = df_app['DAYS_EMPLOYED'].abs()
df_app['DAYS_REGISTRATION'] = df_app['DAYS_REGISTRATION'].abs()
df_app['DAYS_ID_PUBLISH'] = df_app['DAYS_ID_PUBLISH'].abs()
df_app['DAYS_LAST_PHONE_CHANGE'] = df_app['DAYS_LAST_PHONE_CHANGE'].abs()
```

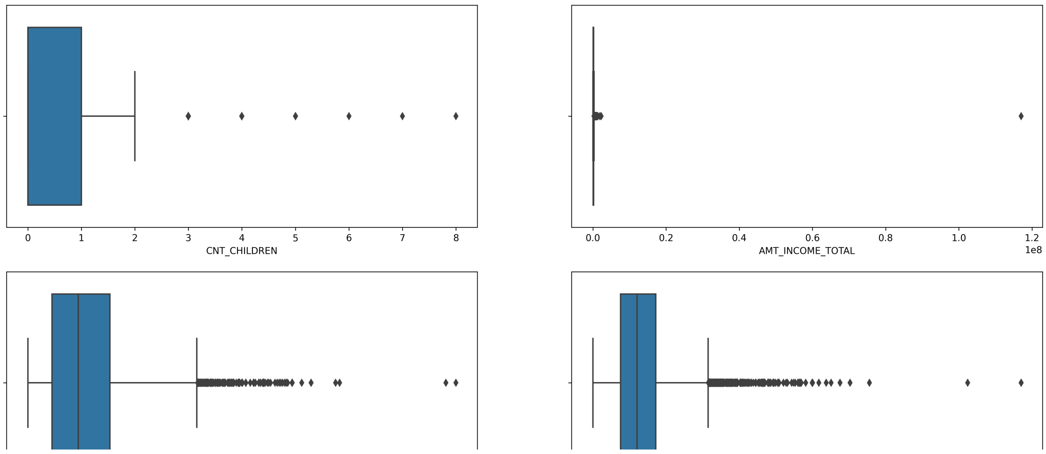
2.e Checking the outlier for numerical variables:

```
# describe numeric columns
df_app[numeric_columns].describe()
```

	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	REGION_POPULATION_RELATIVI
count	17474.0000	17474.0000	17474.0000	17474.0000	17474.0000	17474.000(
mean	0.0783	0.4203	174585.1023	601472.5508	27143.4901	0.020;
std	0.2686	0.7217	889002.9806	403978.0139	14494.7235	0.013{
min	0.0000	0.0000	25650.0000	45000.0000	2052.0000	0.000!
25%	0.0000	0.0000	112500.0000	270000.0000	16456.5000	0.010(
50%	0.0000	0.0000	146250.0000	517500.0000	25076.2500	0.018{
75%	0.0000	1.0000	202500.0000	813195.0000	34749.0000	0.028;
max	1.0000	8.0000	117000000.0000	4050000.0000	225000.0000	0.072!

```
# Box plot for selected columns
features = ['CNT_CHILDREN', 'AMT_INCOME_TOTAL','AMT_CREDIT','AMT_ANNUITY','DAYS_EMPLOYED', 'DAYS_REGISTRATION']

plt.figure(figsize = (20, 15), dpi=300)
for i in enumerate(features):
    plt.subplot(3, 2, i[0]+1)
    sns.boxplot(x = i[1], data = df_app)
plt.show()
```



From the above box plot and descibe analysis we found that following are the numeric columns are having outliers:

```
CNT_CHILDREN, AMT_INCOME_TOTAL,AMT_CREDIT,AMT_ANNUITY,DAYS_EMPLOYED, DAYS_REGISTRATION
```

- The first quartile almost missing for CNT_CHILDREN that means most of the data are present in the first quartile.
- There is single high value data point as outlier present in AMT_INCOME_TOTAL and Removal this point will dtrasticaly impact the box plot for further analysis.
- The first quartiles is slim compare to third quartile for AMT_CREDIT,AMT_ANNUITY, DAYS_EMPLOYED, DAYS_REGISTRATION. This mean data are skewed towards first quartile.

2.f. Bin Creation

Creating bins for continous variable categories column 'AMT_INCOME_TOTAL' and 'AMT_CREDIT'

```
bins = [0,100000,200000,300000,400000,500000,10000000000]
slot = ['<100000', '100000-200000','200000-300000','300000-400000','400000-500000', '500000 and above']

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

```
bins = [0,100000,200000,300000,400000,500000,600000,700000,800000,900000,10000000000]
slot = ['<100000', '100000-200000','200000-300000','300000-400000','400000-500000', '500000-600000',
        '600000-700000','700000-800000','850000-900000','900000 and above']

df_app['AMT_CREDIT_RANGE']=pd.cut(df_app['AMT_CREDIT'],bins,labels=slot)
```

3. Analysis:

```
# Dividing the dataset into two dataset of  target=1(client with payment difficulties) and target=0(all other)

target0_df=df_app.loc[df_app["TARGET"]==0]
target1_df=df_app.loc[df_app["TARGET"]==1]
```

```
# insights from number of target values

percentage_defaulters= round(100*len(target1_df)/(len(target0_df)+len(target1_df)),2)

percentage_nondefaulters=round(100*len(target0_df)/(len(target0_df)+len(target1_df)),2)

print('Count of target0_df:', len(target0_df))
print('Count of target1_df:', len(target1_df))


print('Percentage of people who paid their loan are: ', percentage_nondefaulters, '%' )
print('Percentage of people who did not paid their loan are: ', percentage_defaulters, '%' )
```

```
Count of target0_df: 16106
Count of target1_df: 1368
Percentage of people who paid their loan are:  92.17 %
Percentage of people who did not paid their loan are:  7.83 %
```

```
# Calculating Imbalance percentage
```

```
# Since the majority is target0 and minority is target1

imb_ratio = round(len(target0_df)/len(target1_df),2)

print('Imbalance Ratio:', imb_ratio)
```

```
Imbalance Ratio: 11.77
```

The Imbalance ratio is 11.48

3.a Univariate analysis

Categorical Univariate Analysis in logarithmic scale for target=0 (client with no payment difficulties)

```
# Count plotting in logarithmic scale

def uniplot(df,col,title,hue =None):

    sns.set_style('whitegrid')
    sns.set_context('talk')
    plt.rcParams["axes.labelsize"] = 14
    plt.rcParams['axes.titlesize'] = 16
    plt.rcParams['axes.titlepad'] = 14


    temp = pd.Series(data = hue)
    fig, ax = plt.subplots()
    width = len(df[col].unique()) + 7 + 4*len(temp.unique())
    fig.set_size_inches(width , 8)
    plt.xticks(rotation=45)
    plt.yscale('log')
    plt.title(title)
    ax = sns.countplot(data = df, x= col, order=df[col].value_counts().index,hue = hue)

    plt.show()
```

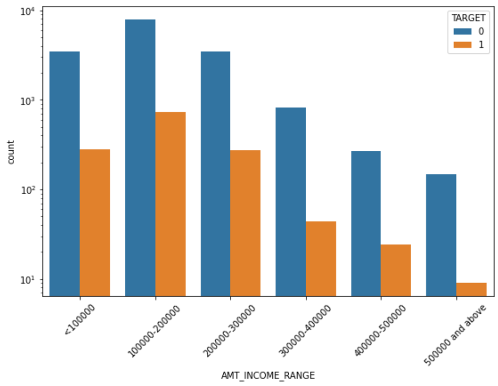
```
# Categoroical Univariate Analysis in logarithmic scale

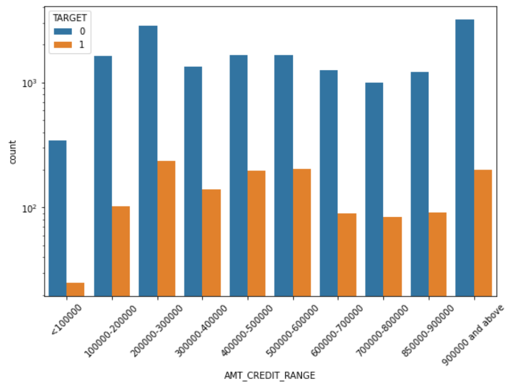
features = ['AMT_INCOME_RANGE', 'AMT_CREDIT_RANGE','NAME_INCOME_TYPE','NAME_CONTRACT_TYPE']
plt.figure(figsize = (20, 15))

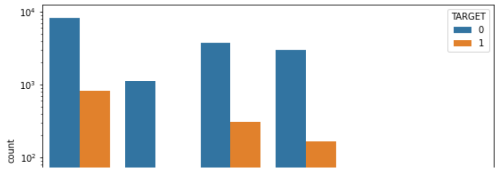
for i in enumerate(features):
    plt.subplot(2, 2, i[0]+1)
    plt.subplots_adjust(hspace=0.5)
    sns.countplot(x = i[1], hue = 'TARGET', data = df_app)

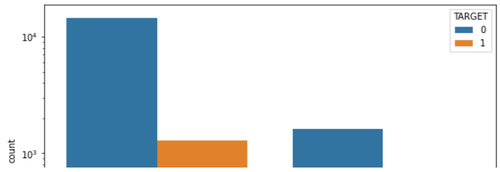
    plt.rcParams['axes.titlesize'] = 16

    plt.xticks(rotation = 45)
    plt.yscale('log')
```









Insights:

> AMT_INCOME_RANGE : * The people having 100000-200000 are havign higher number of loan and also having higher in defaulter * The income segment having >500000 are having less defaulter.

> AMT_CREDIT_RANGE: * The people having <100000 loan are less defaulter. * income having more thatn >100000 are almost equal % of loan defaulter

> NAME_INCOME_TYPE: * Student pensioner and business have higher percentage of loan repayment. * Working, State servent and Commercial associates have higher default percentage. * Maternity category is significantly higher problem in replayement.

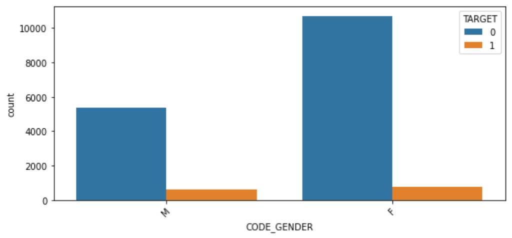
> NAME_CONTRACT_TYPE * For contract type 'cash loans' is having higher number of credits than 'Revolving loans' contract type. * From the above graphs we can see that the Revolving loans are small amount compared to Cash loans but the % of non payment for the revolving loans are comapritvely high.

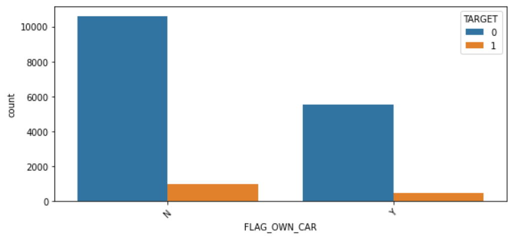
```
# Categoroical Univariate Analysis in Value scale

features = ['CODE_GENDER','FLAG_OWN_CAR']
plt.figure(figsize = (20, 10))

for i in enumerate(features):
    plt.subplot(2, 2, i[0]+1)
    plt.subplots_adjust(hspace=0.5)
    sns.countplot(x = i[1], hue = 'TARGET', data = df_app)

    plt.rcParams['axes.titlesize'] = 16
    plt.xticks(rotation = 45)
#     plt.yscale('log')
```





Insights:

> CODE_GENDER: * The % of defaulters are more in Male than Female

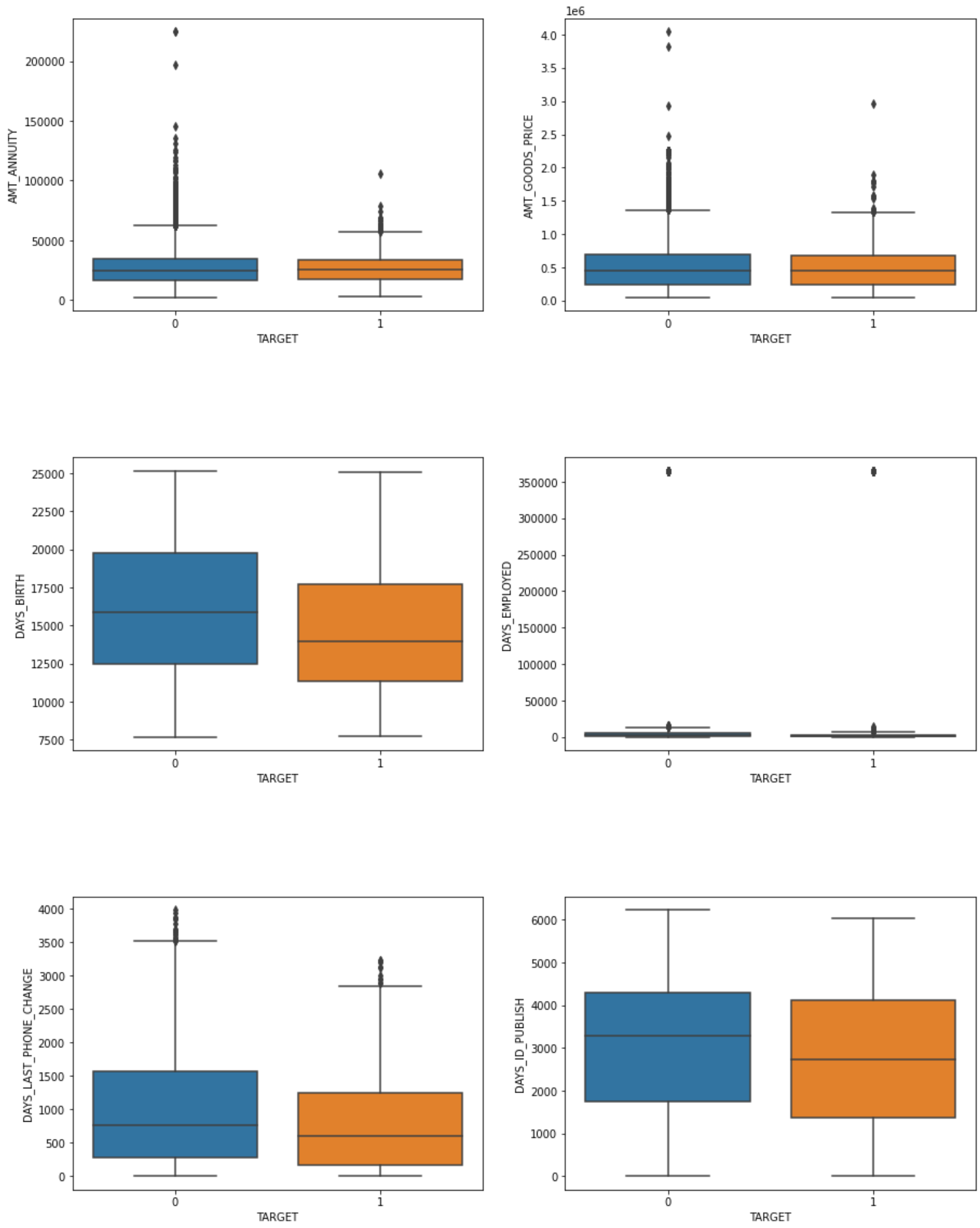
> FLAG_OWN_CAR: * The person owning car is having higher percentage of defaulter.

Univariate analysis Continuious variables:

```
# Univariate Analysis for continous variable

features = ['AMT_ANNUITY','AMT_GOODS_PRICE','DAYS_BIRTH','DAYS_EMPLOYED','DAYS_LAST_PHONE_CHANGE','DAYS_ID_PUBLISH']
plt.figure(figsize = (15, 20))

for i in enumerate(features):
    plt.subplot(3, 2, i[0]+1)
    plt.subplots_adjust(hspace=0.5)
    sns.boxplot(x = 'TARGET', y = i[1], data = df_app)
```



Inference:

- Days_Birth: The people having higher age are having higher probability of repayment.
- Some outliers are observed in In 'AMT_ANNUITY','AMT_GOODS_PRICE','DAYS_EMPLOYED', DAYS_LAST_PHONE_CHANGE in the dataset.
- Less outlier observed in Days_Birth and DAYS_ID_PUBLISH
- 1st quartile is smaller than third quartile in In 'AMT_ANNUITY','AMT_GOODS_PRICE', DAYS_LAST_PHONE_CHANGE.
- In DAYS_ID_PUBLISH: people changing ID in recent days are relativelty prone to be default.

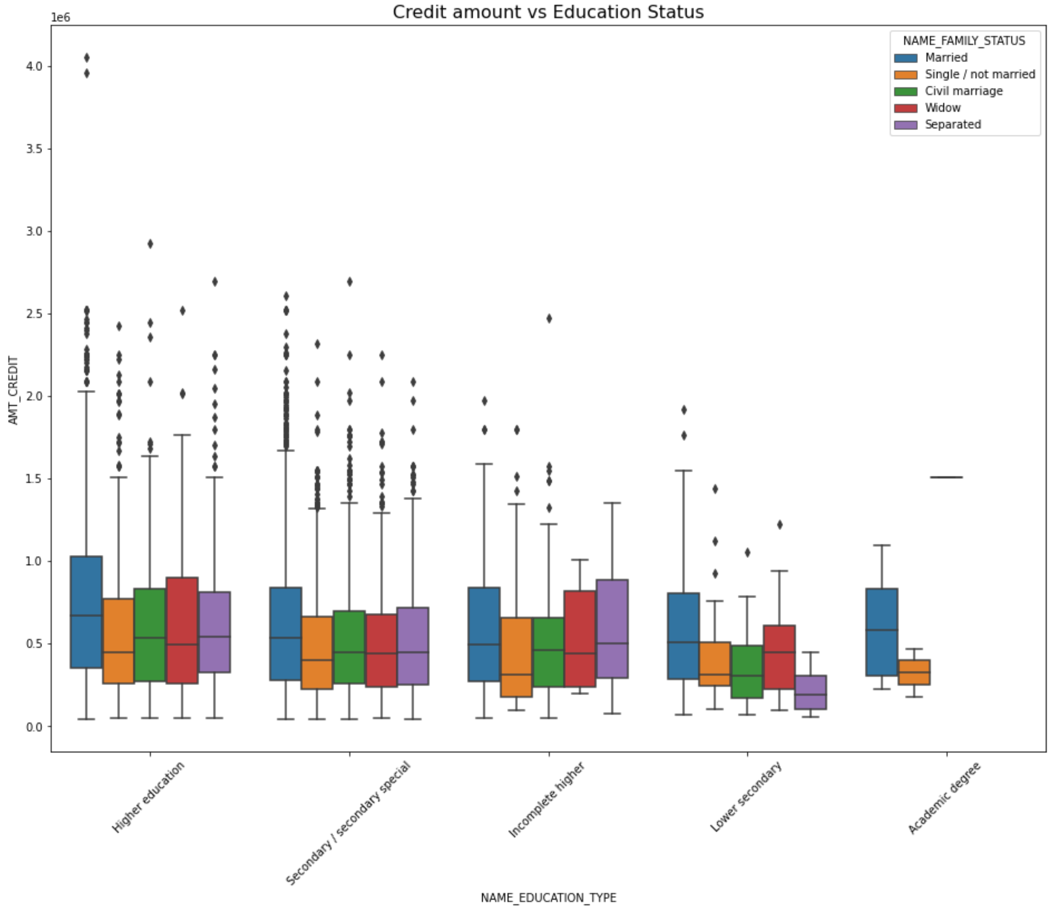
- There is single high value data point as outlier present in DAYS_EMPLOYED. Removal this point will drastically impact the box plot for further analysis.

3.b. Bivariate analysis for numerical variables

For Target 0

```
# Box plotting for Credit amount

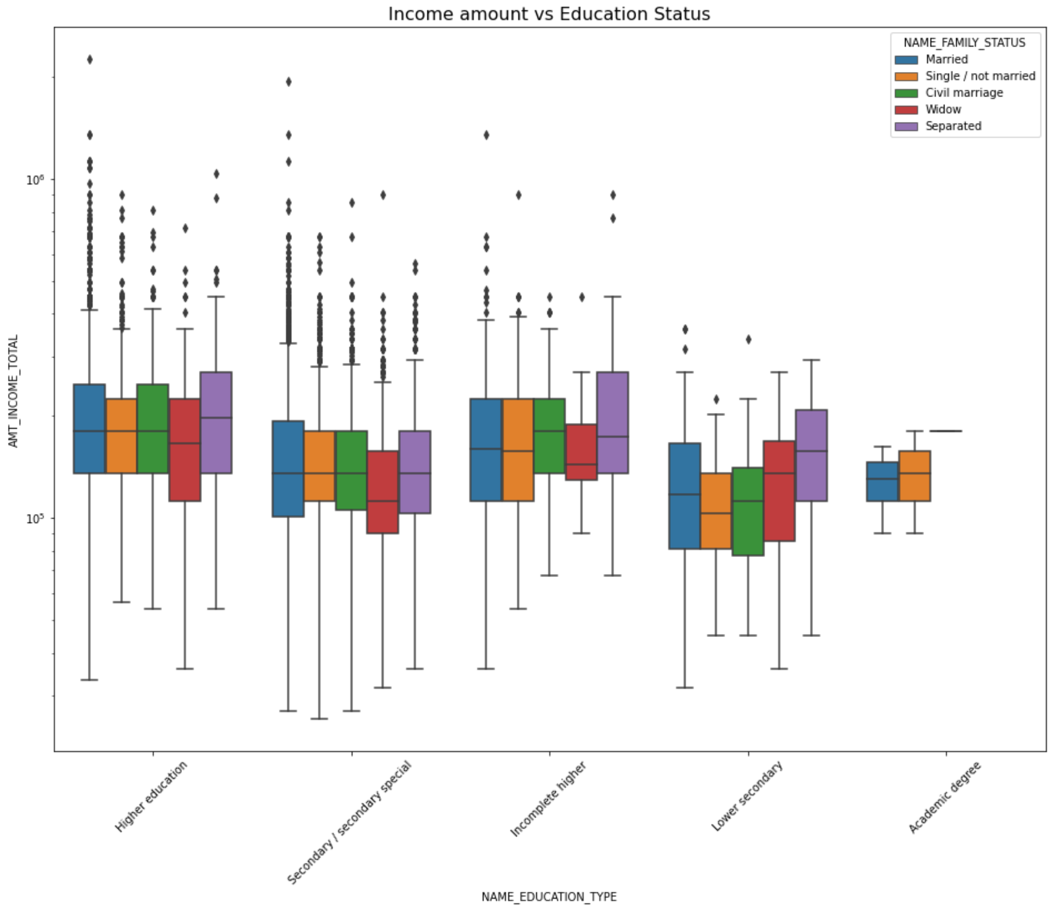
plt.figure(figsize=(16,12))
plt.xticks(rotation=45)
sns.boxplot(data =target0_df, x='NAME_EDUCATION_TYPE',y='AMT_CREDIT', hue ='NAME_FAMILY_STATUS',orient='v')
plt.title('Credit amount vs Education Status')
plt.show()
```



- Family status of 'civil marriage', 'marriage' and 'separated' of Academic degree education are having higher number of credits than others.
- Also, higher education of family status of 'marriage', 'single' and 'civil marriage' are having more outliers. Civil marriage for Academic degree is having most of the credits in the third quartile.

```
# Box plotting for Income amount in logarithmic scale

plt.figure(figsize=(16,12))
plt.xticks(rotation=45)
plt.yscale('log')
sns.boxplot(data =target0_df, x='NAME_EDUCATION_TYPE',y='AMT_INCOME_TOTAL', hue ='NAME_FAMILY_STATUS',orient='v')
plt.title('Income amount vs Education Status')
plt.show()
```

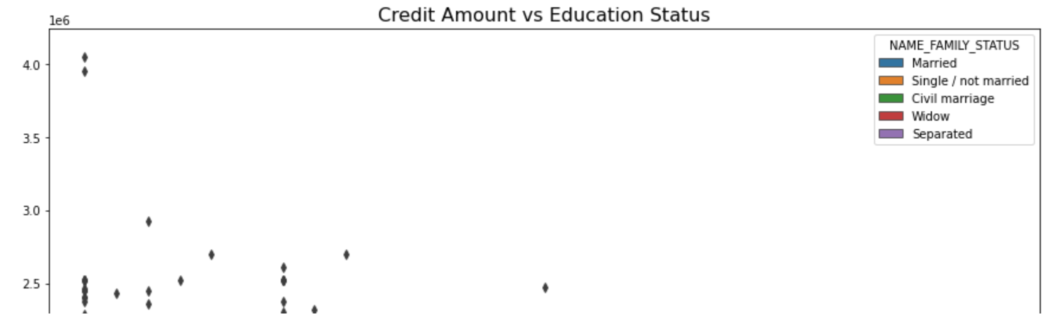


- In Education type 'Higher education' the income amount is mostly equal with family status. It does contain many outliers.
- Less outlier are having for Academic degree but there income amount is little higher that Higher education.
- Lower secondary of civil marriage family status are have less income amount than others.

For Target 1

```
# Box plotting for credit amount

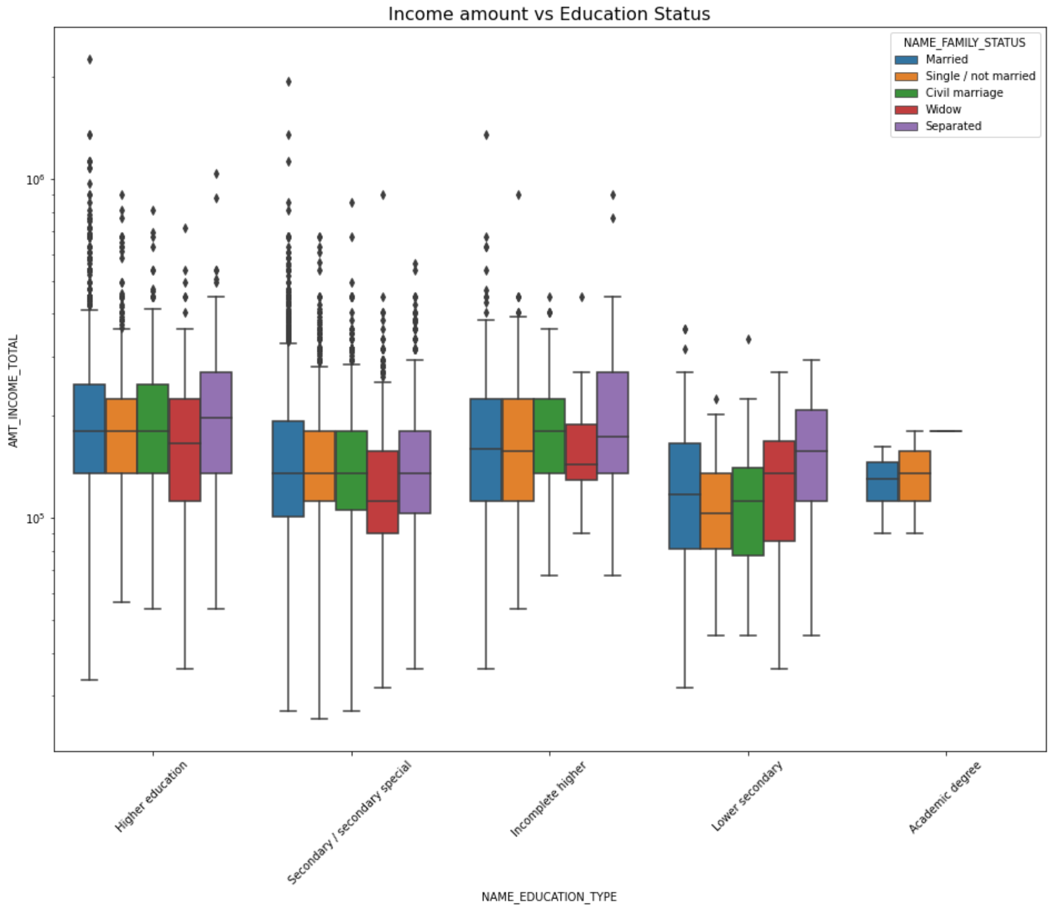
plt.figure(figsize=(15,10))
plt.xticks(rotation=45)
sns.boxplot(data =target0_df, x='NAME_EDUCATION_TYPE',y='AMT_CREDIT', hue ='NAME_FAMILY_STATUS',orient='v')
plt.title('Credit Amount vs Education Status')
plt.show()
```



- Observations are Quite similar with Target 0
- Family status of 'civil marriage', 'marriage' and 'separated' of Academic degree education are having higher number of credits than others.
- Most of the outliers are from Education type 'Higher education' and 'Secondary'.
- Civil marriage for Academic degree is having most of the credits in the third quartile.

```
# Box plotting for Income amount in logarithmic scale

plt.figure(figsize=(16,12))
plt.xticks(rotation=45)
plt.yscale('log')
sns.boxplot(data =target0_df, x='NAME_EDUCATION_TYPE',y='AMT_INCOME_TOTAL', hue ='NAME_FAMILY_STATUS',orient='v')
plt.title('Income amount vs Education Status')
plt.show()
```



- There is also have some similarity with Target0,
- Education type 'Higher education' the income amount is mostly equal with family status.
- Less outlier are having for Academic degree but there income amount is little higher that Higher education.
- Lower secondary are have less income amount than others.

3.c. Correlation:

Getting top 10 correlation between variables

```
# Top 10 correlated variables: target 0 dataaframe

corr = target0_df.corr()
corrdf = corr.where(np.triu(np.ones(corr.shape), k=1).astype(np.bool))
corrdf = corrdf.unstack().reset_index()
corrdf.columns = ['Var1', 'Var2', 'Correlation']
corrdf.dropna(subset = ['Correlation'], inplace = True)
corrdf['Correlation'] = round(corrdf['Correlation'], 2)
corrdf['Correlation'] = abs(corrdf['Correlation'])
corrdf.sort_values(by = 'Correlation', ascending = False).head(10)
```

	Var1	Var2	Correlation
649	OBS_60_CNT_SOCIAL_CIRCLE	OBS_30_CNT_SOCIAL_CIRCLE	1.0000
184	AMT_GOODS_PRICE	AMT_CREDIT	0.9900
680	DEF_60_CNT_SOCIAL_CIRCLE	DEF_30_CNT_SOCIAL_CIRCLE	0.8600
464	LIVE_REGION_NOT_WORK_REGION	REG_REGION_NOT_WORK_REGION	0.8500
557	LIVE_CITY_NOT_WORK_CITY	REG_CITY_NOT_WORK_CITY	0.8200
185	AMT_GOODS_PRICE	AMT_ANNUITY	0.7900
154	AMT_ANNUITY	AMT_CREDIT	0.7800
278	DAYS_EMPLOYED	DAYS_BIRTH	0.6200
433	REG_REGION_NOT_WORK_REGION	REG_REGION_NOT_LIVE_REGION	0.4800
153	AMT_ANNUITY	AMT_INCOME_TOTAL	0.4600

```
# Top 10 correlated variables: target 1 dataaframe

corr = target1_df.corr()
corrdf = corr.where(np.triu(np.ones(corr.shape), k=1).astype(np.bool))
corrdf = corrdf.unstack().reset_index()
corrdf.columns = ['Var1', 'Var2', 'Correlation']
corrdf.dropna(subset = ['Correlation'], inplace = True)
corrdf['Correlation'] = round(corrdf['Correlation'], 2)
corrdf['Correlation'] = abs(corrdf['Correlation'])
corrdf.sort_values(by = 'Correlation', ascending = False).head(10)
```

	Var1	Var2	Correlation
649	OBS_60_CNT_SOCIAL_CIRCLE	OBS_30_CNT_SOCIAL_CIRCLE	1.0000
184	AMT_GOODS_PRICE	AMT_CREDIT	0.9800
680	DEF_60_CNT_SOCIAL_CIRCLE	DEF_30_CNT_SOCIAL_CIRCLE	0.8900
464	LIVE_REGION_NOT_WORK_REGION	REG_REGION_NOT_WORK_REGION	0.7900
557	LIVE_CITY_NOT_WORK_CITY	REG_CITY_NOT_WORK_CITY	0.7800
185	AMT_GOODS_PRICE	AMT_ANNUITY	0.7700
154	AMT_ANNUITY	AMT_CREDIT	0.7600
278	DAYS_EMPLOYED	DAYS_BIRTH	0.5800
433	REG_REGION_NOT_WORK_REGION	REG_REGION_NOT_LIVE_REGION	0.5500
526	REG_CITY_NOT_WORK_CITY	REG_CITY_NOT_LIVE_CITY	0.4700

- From the above correlation analysis it is infered that the highest corelation (1.0) is between (OBS_60_CNT_SOCIAL_CIRCLE with OBS_30_CNT_SOCIAL_CIRCLE) and (FLOORSMAX_MEDI with FLOORSMAX_AVG) which is same for both the data set.

4. Read Previous Application data and merging with application data

```
# Reading the dataset of previous application

df_prev=pd.read_csv('previous_application.csv')
```

```
#explore the dataset
df_prev.columns
```

```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRICE',
       'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
       'FLAG_LAST_APPL_PER_CONTRACT', 'NFLAG_LAST_APPL_IN_DAY',
       'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
       'RATE_INTEREST_PRIVILEGED', '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')
```

```
# get shape of data (rows, columns)
df_prev.shape
```

```
(51192, 37)
```

```
# get the type of dataset
df_prev.dtypes
```

```
SK_ID_PREV                      int64
SK_ID_CURR                      int64
NAME_CONTRACT_TYPE             object
AMT_ANNUITY                   float64
AMT_APPLICATION               float64
AMT_CREDIT                    float64
AMT_DOWN_PAYMENT              float64
AMT_GOODS_PRICE               float64
WEEKDAY_APPR_PROCESS_START     object
HOUR_APPR_PROCESS_START       float64
FLAG_LAST_APPL_PER_CONTRACT    object
NFLAG_LAST_APPL_IN_DAY        float64
RATE_DOWN_PAYMENT             float64
RATE_INTEREST_PRIMARY         float64
RATE_INTEREST_PRIVILEGED      float64
NAME_CASH_LOAN_PURPOSE         object
NAME_CONTRACT_STATUS           object
DAYS_DECISION                 float64
NAME_PAYMENT_TYPE              object
CODE_REJECT_REASON             object
NAME_TYPE_SUITE                object
NAME_CLIENT_TYPE               object
NAME_GOODS_CATEGORY            object
NAME_PORTFOLIO                 object
NAME_PRODUCT_TYPE              object
CHANNEL_TYPE                   object
SELLERPLACE_AREA              float64
NAME_SELLER_INDUSTRY           object
CNT_PAYMENT                   float64
NAME_YIELD_GROUP               object
PRODUCT_COMBINATION            object
DAYS_FIRST_DRAWING            float64
DAYS_FIRST_DUE                float64
DAYS_LAST_DUE_1ST_VERSION     float64
DAYS_LAST_DUE                 float64
DAYS_TERMINATION              float64
NFLAG_INSURED_ON_APPROVAL     float64
dtype: object
```

```
# displaying the informtion of previous application dataset
df_prev.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51192 entries, 0 to 51191
Data columns (total 37 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   SK_ID_PREV                   51192 non-null  int64
 1   SK_ID_CURR                   51192 non-null  int64
 2   NAME_CONTRACT_TYPE           51192 non-null  object
 3   AMT_ANNUITY                  40351 non-null  float64
 4   AMT_APPLICATION              51192 non-null  float64
 5   AMT_CREDIT                   51192 non-null  float64
 6   AMT_DOWN_PAYMENT             25446 non-null  float64
 7   AMT_GOODS_PRICE              40185 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   51191 non-null  object
 9   HOUR_APPR_PROCESS_START      51191 non-null  float64
 10  FLAG_LAST_APPL_PER_CONTRACT  51191 non-null  object
 11  NFLAG_LAST_APPL_IN_DAY       51191 non-null  float64
 12  RATE_DOWN_PAYMENT            25445 non-null  float64
```

```
 13  RATE_INTEREST_PRIMARY       170 non-null    float64
 14  RATE_INTEREST_PRIVILEGED    170 non-null    float64
 15  NAME_CASH_LOAN_PURPOSE      51191 non-null  object
 16  NAME_CONTRACT_STATUS        51191 non-null  object
 17  DAYS_DECISION               51191 non-null  float64
 18  NAME_PAYMENT_TYPE           51191 non-null  object
 19  CODE_REJECT_REASON          51191 non-null  object
 20  NAME_TYPE_SUITE             26344 non-null  object
 21  NAME_CLIENT_TYPE            51191 non-null  object
 22  NAME_GOODS_CATEGORY         51191 non-null  object
 23  NAME_PORTFOLIO              51191 non-null  object
 24  NAME_PRODUCT_TYPE           51191 non-null  object
 25  CHANNEL_TYPE                51191 non-null  object
 26  SELLERPLACE_AREA            51191 non-null  float64
 27  NAME_SELLER_INDUSTRY        51191 non-null  object
 28  CNT_PAYMENT                 40350 non-null  float64
 29  NAME_YIELD_GROUP            51191 non-null  object
 30  PRODUCT_COMBINATION         51183 non-null  object
 31  DAYS_FIRST_DRAWING          31587 non-null  float64
 32  DAYS_FIRST_DUE              31587 non-null  float64
 33  DAYS_LAST_DUE_1ST_VERSION   31587 non-null  float64
 34  DAYS_LAST_DUE               31587 non-null  float64
 35  DAYS_TERMINATION            31587 non-null  float64
 36  NFLAG_INSURED_ON_APPROVAL   31587 non-null  float64
dtypes: float64(19), int64(2), object(16)
memory usage: 14.5+ MB
```

```
# Describing the previous application dataset
df_prev.describe()
```

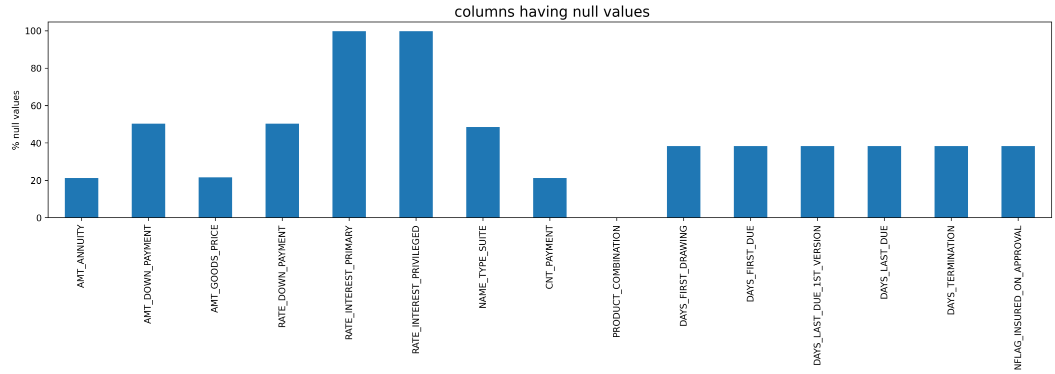
	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOOD
count	51192.0000	51192.0000	40351.0000	51192.0000	51192.0000	25446.0000	40
mean	1922177.8510	278964.4087	15442.8001	168369.1932	188023.0428	6523.5078	2145
std	535495.1780	102705.8720	14496.7863	281583.2967	307855.3473	17307.8811	3018
min	1000001.0000	100007.0000	0.0000	0.0000	0.0000	0.0000	
25%	1457113.5000	190097.7500	6103.7325	22005.0000	26049.3750	0.0000	49
50%	1920694.5000	279232.5000	10849.8600	71095.5000	78552.0000	1570.5000	103
75%	2388982.0000	368430.0000	19609.8525	180000.0000	197820.0000	7830.0000	225
max	2845367.0000	456254.0000	234478.3950	3826372.5000	4104351.0000	1035000.0000	3826

```
# Finding percentage of null values columns
NA_col_pre = column_wise_null_percentage(df_prev)
```

```
# identify columns only with null values
NA_col_pre = NA_col_pre[NA_col_pre>0]
NA_col_pre
```

```
AMT_ANNUITY                  21.1800
AMT_DOWN_PAYMENT             50.2900
AMT_GOODS_PRICE              21.5000
RATE_DOWN_PAYMENT            50.2900
RATE_INTEREST_PRIMARY        99.6700
RATE_INTEREST_PRIVILEGED     99.6700
NAME_TYPE_SUITE              48.5400
CNT_PAYMENT                  21.1800
PRODUCT_COMBINATION           0.0200
DAYS_FIRST_DRAWING           38.3000
DAYS_FIRST_DUE               38.3000
DAYS_LAST_DUE_1ST_VERSION    38.3000
DAYS_LAST_DUE                38.3000
DAYS_TERMINATION             38.3000
NFLAG_INSURED_ON_APPROVAL    38.3000
dtype: float64
```

```
# grafical representation of columns having % null values
plt.figure(figsize= (20,4),dpi=300)
NA_col_pre.plot(kind = 'bar')
plt.title (' columns having null values')
plt.ylabel('% null values')
plt.show()
```




```
# Get the column with null values more than 50%
NA_col_pre = NA_col_pre[NA_col_pre>50]
print("Number of columns having null value more than 50% :", len(NA_col_pre.index))
print(NA_col_pre)
```

```
Number of columns having null value more than 50% : 4
AMT_DOWN_PAYMENT             50.2900
RATE_DOWN_PAYMENT            50.2900
RATE_INTEREST_PRIMARY        99.6700
RATE_INTEREST_PRIVILEGED     99.6700
dtype: float64
```

- Droped all columns from Dataframe for which missing value percentage are more than 50%.

    'AMT_DOWN_PAYMENT', 'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY','RATE_INTEREST_PRIVILEGED'

```
# removed 4 columns having null percentage more than 50%.
df_prev = df_prev.drop(NA_col_pre.index, axis =1)
df_prev.shape
```

```
(51192, 33)
```

```
# Merging the Application dataset with previous appliaction dataset

df_comb = pd.merge(left=df_app,right=df_prev,how='inner',on='SK_ID_CURR',suffixes='_x')
df_comb.shape
```

```
(2387, 76)
```

```
df_comb.head()
```

K_ID_CURR	TARGET	NAME_CONTRACT_TYPE_	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INC
100007	0	Cash loans	M	N	Y	0	1
100009	0	Cash loans	F	Y	Y	1	1
100012	0	Revolving loans	M	N	Y	0	1
100026	0	Cash loans	F	N	N	1	4
100027	0	Cash loans	F	N	Y	0	[illegible]

```
# Renaming the column names after merging from combined df
```

```
df_comb = df_comb.rename({'NAME_CONTRACT_TYPE_' : 'NAME_CONTRACT_TYPE','AMT_CREDIT_':'AMT_CREDIT','AMT_ANNUITY_':'AMT_ANNUITY',
                          'WEEKDAY_APPR_PROCESS_START_' : 'WEEKDAY_APPR_PROCESS_START',
                          'HOUR_APPR_PROCESS_START_':'HOUR_APPR_PROCESS_START','NAME_CONTRACT_TYPEx':'NAME_CONTRACT_TYPE_PREV',
                          'AMT_CREDITx':'AMT_CREDIT_PREV','AMT_ANNUITYx':'AMT_ANNUITY_PREV',
                          'WEEKDAY_APPR_PROCESS_STARTx':'WEEKDAY_APPR_PROCESS_START_PREV',
                          'HOUR_APPR_PROCESS_STARTx':'HOUR_APPR_PROCESS_START_PREV'}, axis=1)


# Removing unwanted columns from cmbined df for analysis

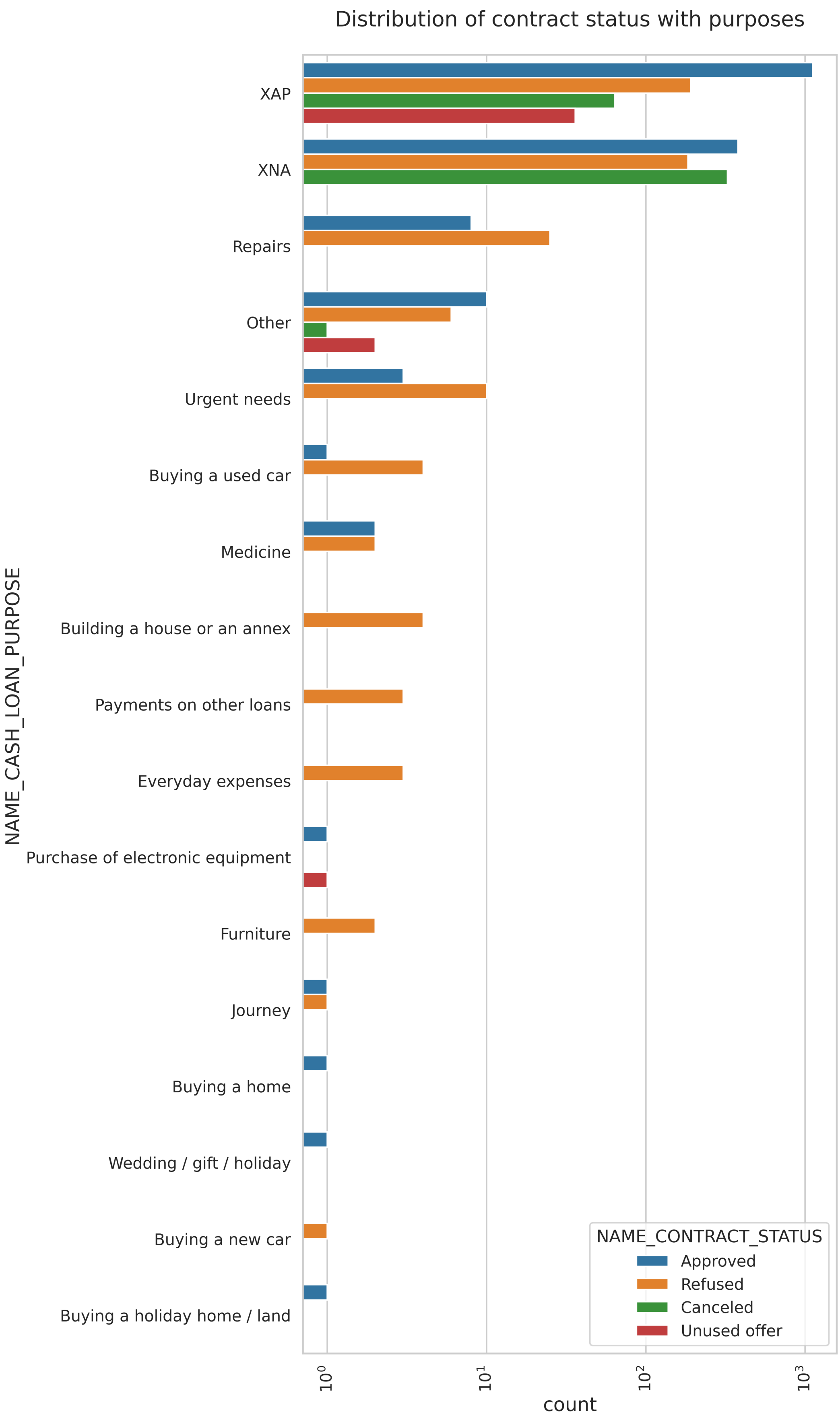
df_comb.drop(['SK_ID_CURR','WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START','REG_REGION_NOT_LIVE_REGION',
              'REG_REGION_NOT_WORK_REGION','LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
              'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY','WEEKDAY_APPR_PROCESS_START_PREV',
              'HOUR_APPR_PROCESS_START_PREV', 'FLAG_LAST_APPL_PER_CONTRACT','NFLAG_LAST_APPL_IN_DAY'],axis=1,inplace=True)
```

** Performing univariate analysis**

```
# Distribution of contract status in logarithmic scale
# Distribution of contract status in logarithmic scale

sns.set_style('whitegrid')
sns.set_context('talk')

plt.figure(figsize=(10,25),dpi = 300)
plt.rcParams["axes.labelsize"] = 20
plt.rcParams['axes.titlesize'] = 22
plt.rcParams['axes.titlepad'] = 30
plt.xticks(rotation=90)
plt.xscale('log')
plt.title('Distribution of contract status with purposes')
ax = sns.countplot(data = df_comb, y= 'NAME_CASH_LOAN_PURPOSE',
                   order=df_comb['NAME_CASH_LOAN_PURPOSE'].value_counts().index,hue = 'NAME_CONTRACT_STATUS')
```



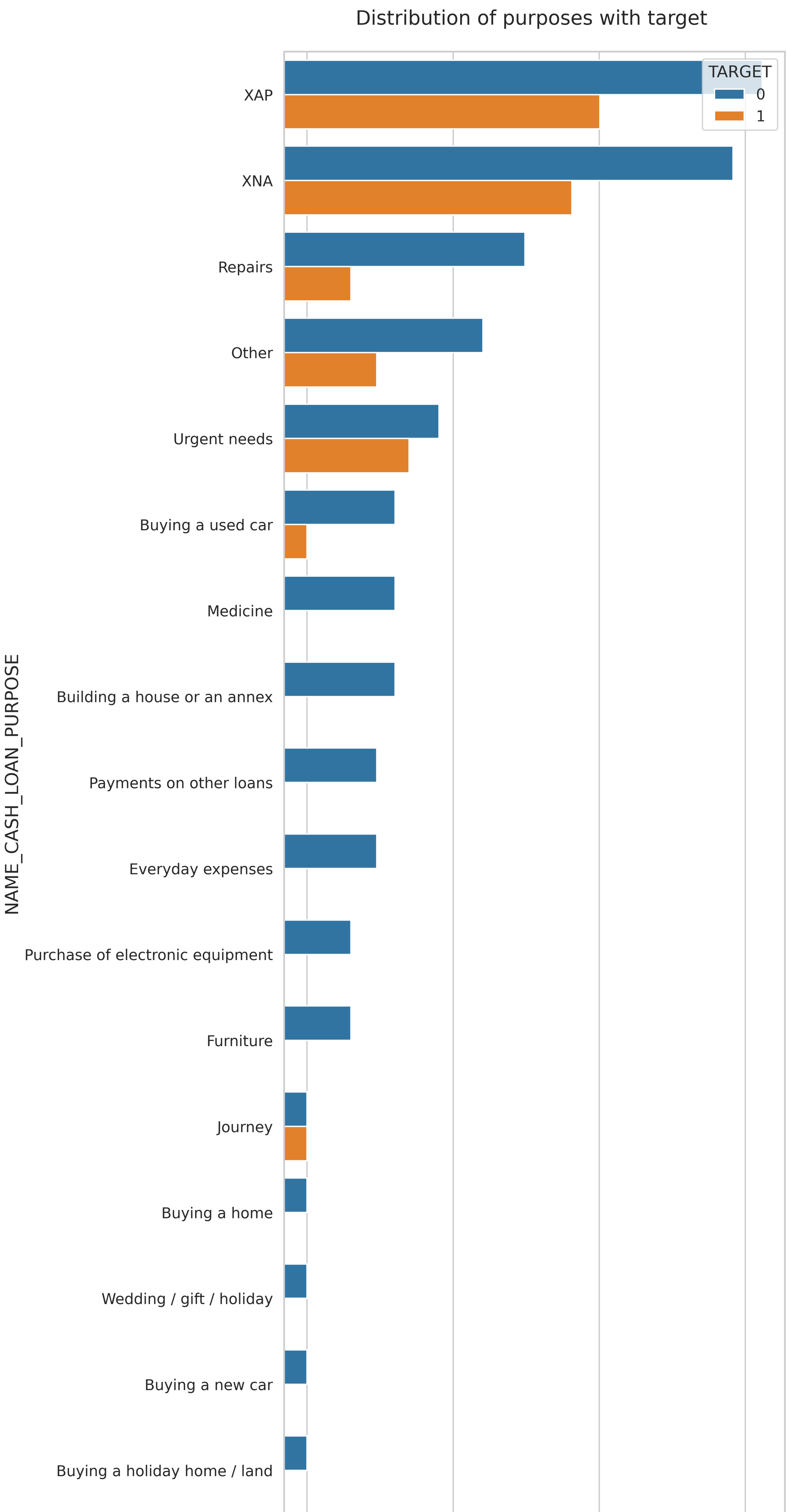
Points to be concluded from above plot:

Most rejection of loans came from purpose 'repairs'. For education purposes we have equal number of approves and rejection Payign other loans and buying a new car is having significant higher rejection than approves.

```
# Distribution of contract status

sns.set_style('whitegrid')
sns.set_context('talk')

plt.figure(figsize=(10,30),dpi = 300)
plt.rcParams["axes.labelsize"] = 20
plt.rcParams['axes.titlesize'] = 22
plt.rcParams['axes.titlepad'] = 30
plt.xticks(rotation=90)
plt.xscale('log')
plt.title('Distribution of purposes with target ')
ax = sns.countplot(data = df_comb, y= 'NAME_CASH_LOAN_PURPOSE',
                   order=df_comb['NAME_CASH_LOAN_PURPOSE'].value_counts().index,hue = 'TARGET')
```



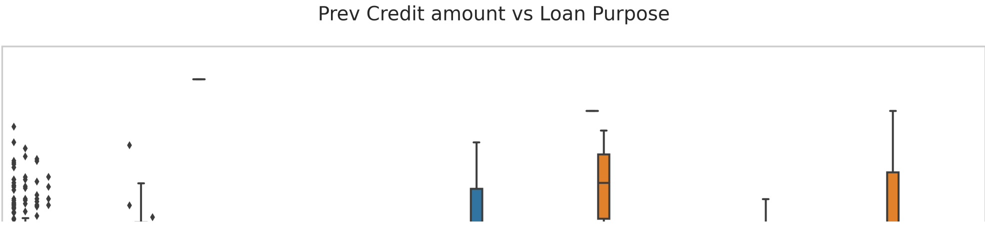
Few points we can conclude from abpve plot:

Loan purposes with 'Repairs' are facing more difficulites in payment on time. There are few places where loan payment is significant higher than facing difficulties. They are 'Buying a garage', 'Business developemt', 'Buying land','Buying a new car' and 'Education' Hence we can focus on these purposes for which the client is having for minimal payment difficulties.

Bivariate analysis

```
# Box plotting for Credit amount in logarithmic scale

plt.figure(figsize=(20,15),dpi = 300)
plt.xticks(rotation=90)
plt.yscale('log')
sns.boxplot(data =df_comb, x='NAME_CASH_LOAN_PURPOSE',hue='NAME_INCOME_TYPE',y='AMT_CREDIT_PREV',orient='v')
plt.title('Prev Credit amount vs Loan Purpose')
plt.show()
```



From the above we can conclude some points-

The credit amount of Loan purposes like 'Buying a home','Buying a land','Buying a new car' and'Building a house' is higher. Income type of state servants have a significant amount of credit applied Money for third person or a Hobby is having less credits applied for.

```
# Box plotting for Credit amount prev vs Housing type in logarithmic scale

plt.figure(figsize=(15,15),dpi = 150)
plt.xticks(rotation=90)
sns.barplot(data =df_comb, y='AMT_CREDIT_PREV',hue='TARGET',x='NAME_HOUSING_TYPE')
plt.title('Prev Credit amount vs Housing type')
plt.show()
```

Here for Housing type, office appartment is having higher credit of target 0 and co-op apartment is having higher credit of target 1. So, we can conclude that bank should avoid giving loans to the housing type of co-op apartment as they are having difficulties in payment. Bank can focus mostly on housing type with parents or House\appartment or miuncipal appartment for successful payments.

6. Conclusion/Recomendation:

1. Banks should focus more on contract type 'Student' ,'pensioner' and 'Businessman' with housing 'type other than 'Co-op apartment' for successful payments.

2. Banks should focus less on income type 'Working' as they are having most number of unsuccessful payments.

3. In loan purpose 'Repairs':

> a. Although having higher number of rejection in loan purposes with 'Repairs' there are observed difficulties in payment on time.
> b. There are few places where loan payment is delay is significantly high.
> c. Bank should keep continue to caution while giving loan for this purpose.

4. Bank should avoid giving loans to the housing type of co-op apartment as they are having difficulties in payment.

5. Bank can focus mostly on housing type 'with parents' , 'House\apartment' and 'municipal apartment' for successful payments.

✓ 2s completed at 3:49 PM