Credit EDA Case Study

Analysis done by

1. Jyoti Gupta (jyoti.gupta1404@gmail.com)
2. Remya Ramchandran (mimmisudarsana@yahoo.com)

Overview

The loan providing companies find it hard to give loans to the people due to their insufficient or non-existent credit history. Because of that, some consumers use it as their advantage by becoming a defaulter. Here we are assuming that we are working for a consumer finance company which specializes in lending various types of loans to urban customers. We have to use EDA to analyze the patterns present in the data. This will ensure that the applicants capable of repaying the loan are not rejected.

When the company receives a loan application, the company has to decide for loan approval based on the applicant's profile. Two types of risks are associated with the bank's decision:

1. If the applicant is likely to repay the loan, then not approving the loan results in a loss of business to the company
2. If the applicant is not likely to repay the loan, i.e. he/she is likely to default, then approving the loan may lead to a financial loss for the company.

Business Objective

This case study aims to identify patterns which indicate if a client has difficulty paying their installments which may be used for taking actions such as denying the loan, reducing the amount of loan, lending (to risky applicants) at a higher interest rate, etc. This will ensure that the consumers capable of repaying the loan are not rejected. **Identification of such applicants using EDA is the aim of this case study.**

In other words, the company wants to understand the driving factors (or driver variables) behind loan default, i.e. the variables which are strong indicators of default. The company can utilize this knowledge for its portfolio and risk assessment.

Description

The data given below contains the information about the loan application at the time of applying for the loan. It contains two types of scenarios:

- **The client with payment difficulties:** he/she had late payment more than X days on at least one of the first Y instalments of the loan in our sample,
- **All other cases:** All other cases when the payment is paid on time.

When a client applies for a loan, there are four types of decisions that could be taken by the client/company):

- **Approved:** The Company has approved loan Application
- **Cancelled:** The client cancelled the application sometime during approval. Either the client changed her/his mind about the loan or in some cases due to a higher risk of the client he received worse pricing which he did not want.
- **Refused:** The company had rejected the loan (because the client does not meet their requirements etc.).
- **Unused offer:** Loan has been cancelled by the client but on different stages of the process.

Data Description

3 datasets files are explained below:

1. *'application_data.csv*' contains all the information of the client at the time of application.
   The data is about whether a **client has payment difficulties.**

2. '*previous_application.csv*' contains information about the client's previous loan data. It contains the data whether the previous application had been **Approved, Cancelled, Refused or Unused offer.**

3. *'columns_description.csv*' is data dictionary which describes the meaning of the variables.

Let's Start !

How we will perform EDA on Credit Score Case Study

- ❑ Analysis on Current Application Dataset
  - ✓ Understand the problem & Read/Examine the Dataset
  - ✓ Data Quality Check & Missing Values
  - ✓ Univariate Analysis
  - ✓ Bivariate Analysis
  - ✓ Multivariate Analysis

- ❑ Analysis on Previous Application Dataset
  - ✓ Understand the problem & Read/Examine the Dataset
  - ✓ Data Quality Check & Missing Values
  - ✓ Bivariate Analysis

- ❑ Merging Current & Previous Application Dataset
  - ✓ Bivariate Analysis
  - ✓ Multivariate Analysis

- ❑ What If We DO NOT Handle Missing Values

- ❑ Conclusions

Analysis on Current application Dataset

Understand the Problem
&
Read/Examine the Dataset

Import Libraries and Read Data

Imports

```
In [1]: # import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

Read The Data

```
In [2]: #read application data
app_data = pd.read_csv('application_data.csv')
```

Check the loaded Data

```
In [3]: #Check the loaded data
app_data.head()
```

Out[3]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_C
0	100002	1	Cash loans	M	N	Y	0	202500.0	40
1	100003	0	Cash loans	F	N	N	0	270000.0	129
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	13
3	100006	0	Cash loans	F	N	Y	0	135000.0	31
4	100007	0	Cash loans	M	N	Y	0	121500.0	51

5 rows × 122 columns

Examine the Distribution of the TARGET Column

The target is what we are asked to predict:

- **either 0 for the loan was repaid on time**

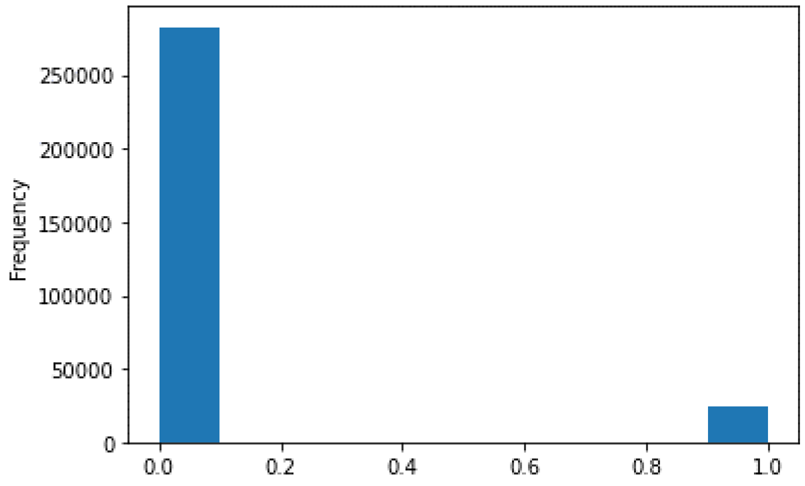
  OR
- **1 indicating the client had payment difficulties**

We can first examine the number of loans falling into each category.

From this information, we see there is a Data imbalance. There are 92% loans that were repaid on time and only 8% loans that were not repaid.

```
In [98]: app_data["TARGET"].value_counts(normalize=True)

Out[98]: 0    0.919271
         1    0.080729
         Name: TARGET, dtype: float64
```

```
In [97]: app_data['TARGET'].astype(int).plot.hist();
```



Data Quality Check
&
Missing Values

Identifying & Treating Missing Values

```
In [18]: # Data cleaning.

         null_count = app_data.isnull().sum().to_frame()
         for index, row in null_count.iterrows():
             print(index, row[0])
```

```
SK_ID_CURR 0
TARGET 0
NAME_CONTRACT_TYPE 0
CODE_GENDER 0
FLAG_OWN_CAR 0
FLAG_OWN_REALTY 0
CNT_CHILDREN 0
AMT_INCOME_TOTAL 0
AMT_CREDIT 0
AMT_ANNUITY 12
AMT_GOODS_PRICE 278
NAME_TYPE_SUITE 1292
NAME_INCOME_TYPE 0
NAME_EDUCATION_TYPE 0
NAME_FAMILY_STATUS 0
NAME_HOUSING_TYPE 0
REGION_POPULATION_RELATIVE 0
DAYS_BIRTH 0
DAYS_EMPLOYED 0
```

From the above analysis of null values, we could see there are some columns with significant amount of null values. Either a column is Numerical or Categorical, we can delete the observations having null values in the dataset or the column that is having more number of null values # i.e. more than half or 30%.

References for handling NULL Values - (https://medium.com/bycodegarage/a-comprehensive-guide-on-handling-missing-values-b1257a4866d1)

```
In [19]: #calculate the percentage of null values in columns.
         # Drop the columns with more than 30% of null values.

         cols_null = app_data.isnull().sum()/len(app_data)*100
         cols_null = cols_null[cols_null.values > 30.0]
         print(len(cols_null))
         print(cols_null)
         # fetch the columns with 30% or more null values.
         cols_null = list(cols_null[cols_null.values > 30.0].index)
         cols_null

         # Drop the columns:

         app_data.drop(columns=cols_null,axis=1,inplace=True)
```

All of these columns are dropped as these have high number of missing values

```
50
OWN_CAR_AGE                     65.990810
OCCUPATION_TYPE                 31.345545
EXT_SOURCE_1                    56.381073
APARTMENTS_AVG                  50.749729
BASEMENTAREA_AVG                58.515956
YEARS_BEGINEXPLUATATION_AVG     48.781019
YEARS_BUILD_AVG                 66.497784
COMMONAREA_AVG                  69.872297
ELEVATORS_AVG                   53.295980
ENTRANCES_AVG                   50.348768
FLOORSMAX_AVG                   49.760822
FLOORSMIN_AVG                   67.848630
LANDAREA_AVG                    59.376738
LIVINGAPARTMENTS_AVG            68.354953
LIVINGAREA_AVG                  50.193326
NONLIVINGAPARTMENTS_AVG         69.432963
NONLIVINGAREA_AVG               55.179164
APARTMENTS_MODE                 50.749729
BASEMENTAREA_MODE               58.515956
YEARS_BEGINEXPLUATATION_MODE    48.781019
YEARS_BUILD_MODE                66.497784
COMMONAREA_MODE                 69.872297
ELEVATORS_MODE                  53.295980
ENTRANCES_MODE                  50.348768
FLOORSMAX_MODE                  49.760822
FLOORSMIN_MODE                  67.848630
LANDAREA_MODE                   59.376738
LIVINGAPARTMENTS_MODE           68.354953
LIVINGAREA_MODE                 50.193326
NONLIVINGAPARTMENTS_MODE        69.432963
NONLIVINGAREA_MODE              55.179164
APARTMENTS_MEDI                 50.749729
BASEMENTAREA_MEDI               58.515956
YEARS_BEGINEXPLUATATION_MEDI    48.781019
YEARS_BUILD_MEDI                66.497784
COMMONAREA_MEDI                 69.872297
ELEVATORS_MEDI                  53.295980
ENTRANCES_MEDI                  50.348768
FLOORSMAX_MEDI                  49.760822
FLOORSMIN_MEDI                  67.848630
LANDAREA_MEDI                   59.376738
LIVINGAPARTMENTS_MEDI           68.354953
LIVINGAREA_MEDI                 50.193326
NONLIVINGAPARTMENTS_MEDI        69.432963
NONLIVINGAREA_MEDI              55.179164
FONDKAPREMONT_MODE              68.386172
HOUSETYPE_MODE                  50.176091
TOTALAREA_MODE                  48.268517
WALLSMATERIAL_MODE              50.840783
EMERGENCYSTATE_MODE             47.398304
dtype: float64
```

Identifying & Treating Missing Values

- **Missing values may not be present always as null. "XNA" is a missing value. Since CODE_GENDER is a categorical column replacing it with mode.**

```
app_data['CODE_GENDER'].value_counts()
```

```
F      202448
M      105059
XNA         4
Name: CODE_GENDER, dtype: int64
```

```
gender_mode = app_data.CODE_GENDER.mode()[0]
gender_mode
app_data.CODE_GENDER = app_data.CODE_GENDER.replace('XNA',gender_mode)
```

```
app_data['CODE_GENDER'].value_counts()
```

```
F    202452
M    105059
Name: CODE_GENDER, dtype: int64
```

- **"DAYS_LAST_PHONE_CHANGE" -- Null value is not replaced as there is only one record and doesn't seems to have an influence on the target variable.**

```
app_data['DAYS_LAST_PHONE_CHANGE'].isnull().sum()
```

```
1
```

Identifying & Treating Missing Values

- **Though we have handles missing values but we still see missing values in “AMT_ANNUITY”. There is a huge difference between min and max value. Till 75% data seems to have an increment in constant proportions. From 75 to max() again is a huge difference. For now we are not doing anything for these missing values as these are very less.**

```
#Handling missing values
app_data[app_data['AMT_ANNUITY'].isnull()].TARGET.value_counts()
```

```
0    12
Name: TARGET, dtype: int64
```

```
#There is a huge difference between min and max value.Till 75% data seems to have an increment in constant propotions.
#From 75 to max() again is a huge difference

app_data.AMT_ANNUITY.describe()
```

```
count    307499.000000
mean      27108.573909
std       14493.737315
min        1615.500000
25%       16524.000000
50%       24903.000000
75%       34596.000000
max      258025.500000
Name: AMT_ANNUITY, dtype: float64
```

- **Since there are outliers in the data, mean will be affected hence imputing with median values. (This can be left as null also as the amount of null value is very low)**

```
# Since there are outliers in the data, mean will be affected hence imputing with median values.
AMT_ANNUITY_FILL = app_data['AMT_ANNUITY'].median()
app_data['AMT_ANNUITY'].fillna(value = AMT_ANNUITY_FILL, inplace =True)
```

- **"NAME_TYPE_SUITE", this column doesn't seem to have significance on the target variable and hence keeping the null values as it is:**

```
# Check the data again.

app_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 73 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                 307511 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
```

Column Types

- Lets take a look at the number of columns of each datatype. Columns with "Object" datatype are "Categorical Columns" and columns with datatype "int64", "float64" are "Numerical Columns"

- We are also checking how many unique values are there in each categorical column.

- Most of the categorical variables have a relatively small number of unique entries than "ORGANIZATION_TYPE"

```
In [99]: # Number of each type of column
         app_data.dtypes.value_counts()

Out[99]: int64       40
         float64     22
         object      11
         category     1
         bool         1
         int32        1
         category     1
         dtype: int64
```

```
In [100]: # Number of unique classes in each object column
          app_data.select_dtypes('object').apply(pd.Series.nunique, axis = 0)

Out[100]: NAME_CONTRACT_TYPE             2
          CODE_GENDER                    2
          FLAG_OWN_CAR                   2
          FLAG_OWN_REALTY                2
          NAME_TYPE_SUITE                7
          NAME_INCOME_TYPE               8
          NAME_EDUCATION_TYPE            5
          NAME_FAMILY_STATUS             6
          NAME_HOUSING_TYPE              6
          WEEKDAY_APPR_PROCESS_START     7
          ORGANIZATION_TYPE             58
          dtype: int64
```

Handling Anomalies

- One problem we always want to be on the lookout for when doing EDA is anomalies within the data. These may be due to mis-typed numbers, errors in measuring equipment, or they could be valid but extreme measurements. One way to support anomalies quantitatively is by looking at the statistics of a column using the describe method.

- In given case study, DAYS_BIRTH column had negative values, hence we need to use abs() function to correct the values in this column and then again analyze the min(), max() and mean()

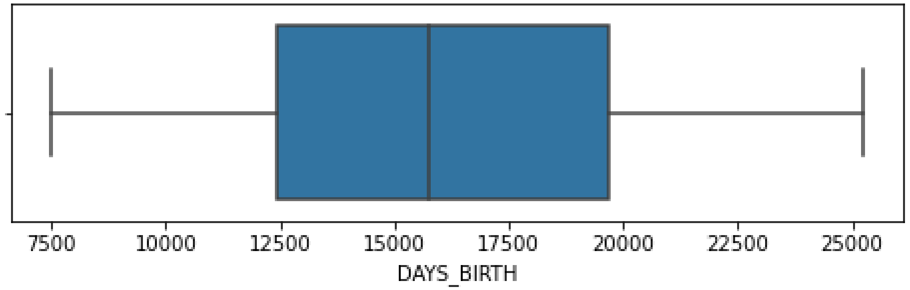
- If we draw the boxplot, then we see that there are no outliers for the age on either the high or low end.

```
app_data['DAYS_BIRTH'] = app_data['DAYS_BIRTH'].abs()
app_data['DAYS_BIRTH_YEAR'] = app_data['DAYS_BIRTH'].apply(lambda x: int(x/365) )
```

```
app_data['DAYS_BIRTH'].describe()
```

```
count    307511.000000
mean      16036.995067
std        4363.988632
min        7489.000000
25%       12413.000000
50%       15750.000000
75%       19682.000000
max       25229.000000
Name: DAYS_BIRTH, dtype: float64
```

```
plt.figure(figsize = [8,2])
sns.boxplot(app_data['DAYS_BIRTH'])

plt.show()
```



Handling Anomalies

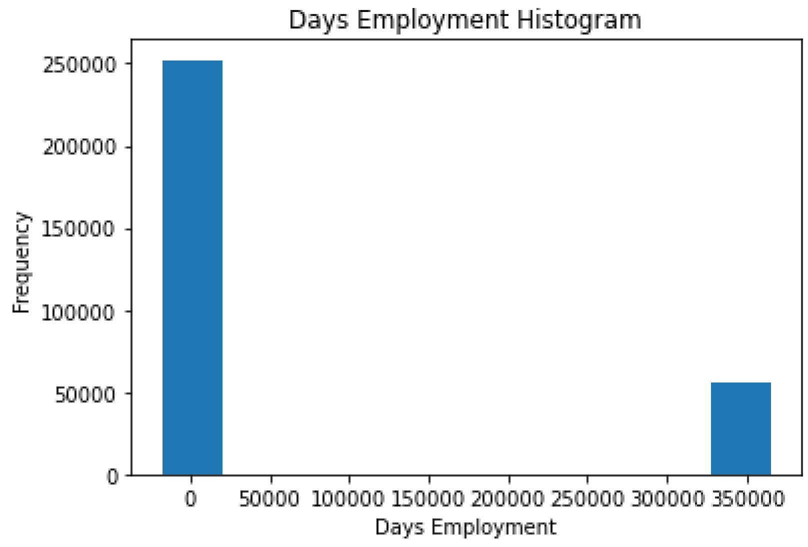
- **Lets check the "DAYS_EMPLOYED" column and analyze the result of describe() function.**

- **That doesn't look right! The maximum value (besides being positive) is about 1000 years!**

```
app_data['DAYS_EMPLOYED'].describe()
```

```
count    307511.000000
mean      63815.045904
std      141275.766519
min      -17912.000000
25%       -2760.000000
50%       -1213.000000
75%        -289.000000
max      365243.000000
Name: DAYS_EMPLOYED, dtype: float64
```

```
app_data['DAYS_EMPLOYED'].plot.hist(title = 'Days Employment Histogram');
plt.xlabel('Days Employment');
```



Handling Anomalies

- **Lets subset the anomalous clients and see if they tend to have higher or low rates of default than the rest of the clients.**

- **Well that is extremely interesting! It turns out that the anomalies have a lower rate of default.**

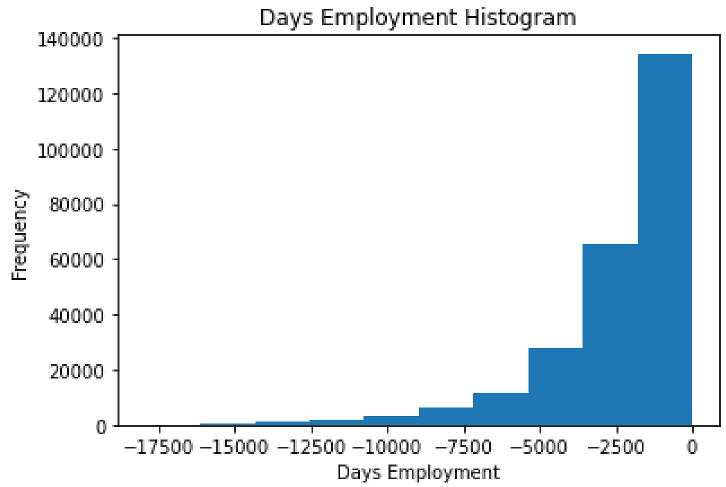
- **One of the safest approaches is just to set the anomalies to a missing value and then have them filled in using Imputation. In this case, since all the anomalies have the exact same value, we want to fill them in with the same value in case all of these loans share something in common. As a solution, we will fill in the anomalous values with not a number (np.nan) and then create a new boolean column indicating whether or not the value was anomalous.**

- **The distribution looks to be much more in line with what we would expect**

```
anom = app_data[app_data['DAYS_EMPLOYED'] == 365243]
non_anom = app_data[app_data['DAYS_EMPLOYED'] != 365243]
print('The non-anomalies default on %0.2f%% of loans' % (100 * non_anom['TARGET'].mean()))
print('The anomalies default on %0.2f%% of loans' % (100 * anom['TARGET'].mean()))
print('There are %d anomalous days of employment' % len(anom))
```

```
The non-anomalies default on 8.66% of loans
The anomalies default on 5.40% of loans
There are 55374 anomalous days of employment
```

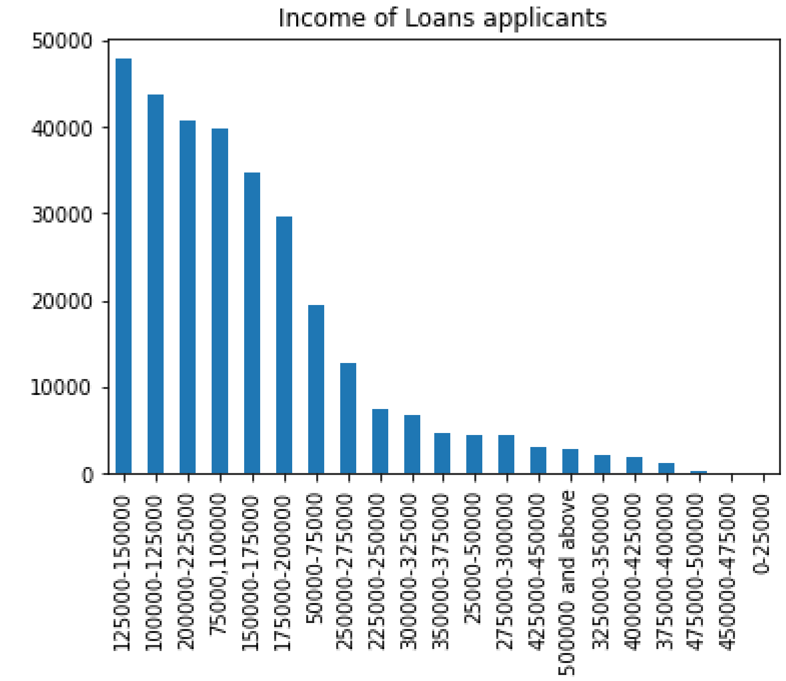
```
# Create an anomalous flag column
app_data['DAYS_EMPLOYED_ANOM'] = app_data["DAYS_EMPLOYED"] == 365243

# Replace the anomalous values with nan
app_data['DAYS_EMPLOYED'].replace({365243: np.nan}, inplace = True)

app_data['DAYS_EMPLOYED'].plot.hist(title = 'Days Employment Histogram');
plt.xlabel('Days Employment');
```



Handling Outliers & Creating Bins

- **First we corrected the format of "AMT_INCOME_TOTAL" column, then we checked the different values from 50th to 99th percentile to see the spread of the data.**
- **We see that last 1% of the population has very high income. We have also drawn the bar chart to see the different bins we created for this column.**

```
app_data.AMT_INCOME_BINS.value_counts().plot.bar()
plt.title('Income of Loans applicants')
plt.show()
```




```
app_data.AMT_INCOME_TOTAL.describe()
```

```
count    3.075110e+05
mean     1.687979e+05
std      2.371231e+05
min      2.565000e+04
25%      1.125000e+05
50%      1.471500e+05
75%      2.025000e+05
max      1.170000e+08
Name: AMT_INCOME_TOTAL, dtype: float64
```

```
#correcting the display of describe function
app_data['AMT_INCOME_TOTAL'].describe().apply("{0:.1f}".format)
```

```
count       307511.0
mean        168797.9
std         237123.1
min          25650.0
25%         112500.0
50%         147150.0
75%         202500.0
max      117000000.0
Name: AMT_INCOME_TOTAL, dtype: object
```

```
# To get a better understanding using quantile function.
app_data.AMT_INCOME_TOTAL.quantile([0.5,0.7,0.9,0.95,0.99])
```

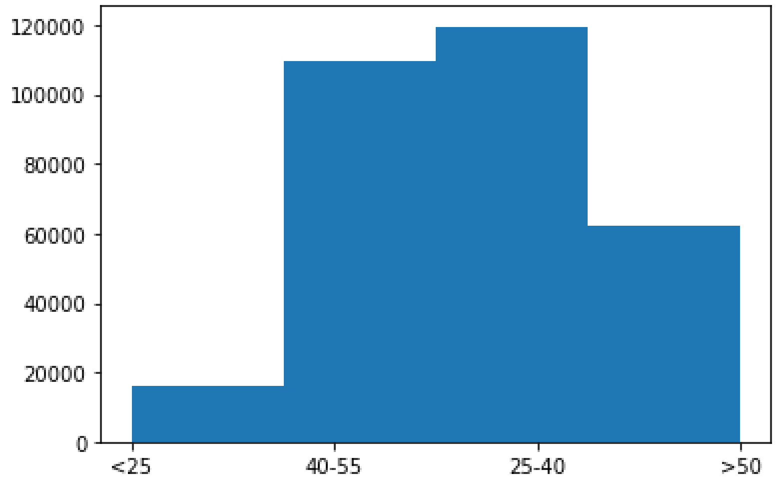
```
0.50    147150.0
0.70    180000.0
0.90    270000.0
0.95    337500.0
0.99    472500.0
Name: AMT_INCOME_TOTAL, dtype: float64
```

Handling Outliers & Creating Bins

- **Lets create bins for age column as well to analyze the population.**
- **We can clearly see that people with age between 25-40 years of age are highest to apply loans. 2nd highest is the people between age group 40-55.**

```
# Creating bins for ages
age_bins = [0,25,40,55,70]
age_slot = ['<25','25-40','40-55','>50']
app_data['DAYS_BIRTH_YEAR'] = pd.cut(app_data['DAYS_BIRTH_YEAR'],age_bins,labels=age_slot)
```

```
plt.hist(app_data['DAYS_BIRTH_YEAR'], bins = 4)
plt.show()
```

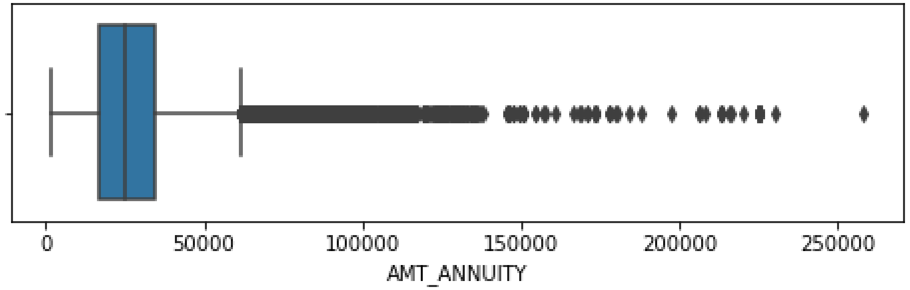


Univariate Analysis

Analyzing – "AMT_ANNUITY"

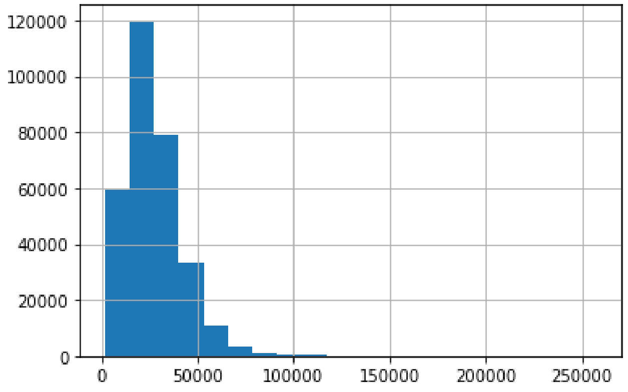
- **Lets draw a boxplot to analyze the spread of the data**

```
# Box plot to analyse the spread of data.
plt.figure(figsize = [8,2])
sns.boxplot(app_data['AMT_ANNUITY'])

plt.show()
```



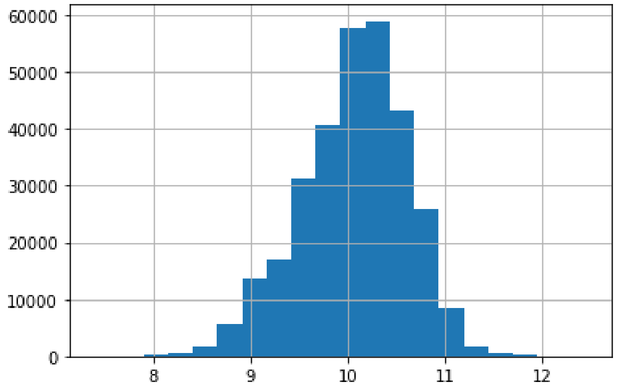

```
app_data['AMT_ANNUITY'].hist(bins=20)
```

<matplotlib.axes._subplots.AxesSubplot at 0x29149f6ff10>




```
app_data['AMT_ANNUITY_LOG'] = np.log(app_data['AMT_ANNUITY'])
app_data['AMT_ANNUITY_LOG'].hist(bins=20)
```

<matplotlib.axes._subplots.AxesSubplot at 0x29149ee3e20>



- **These Boxplot and histogram shows that there are many outliers in the data which means many people are applying for the higher amount of loan, which seems to be a possible case as we have population data so there can be few people with higher loan amount needs.**

- **However, when we use the natural logarithm (inverse exponential function) we can see that it turns out to be a normal distribution and it nullifies the effect of outliers in whole data set.**

- **As observed from the box plots there are outliers. Hence checking the percentile values. This is a possible case as there can be very few people with higher loan/EMI amount.**

```
# As observed from the box plots there are outliers.Hence checking the percentile values.
app_data.AMT_ANNUITY.quantile([0.5,0.7,0.9,0.95,0.99])
```

```
0.50    24903.0
0.70    32004.0
0.90    45954.0
0.95    53325.0
0.99    70006.5
Name: AMT_ANNUITY, dtype: float64
```

```
app_data[app_data['AMT_ANNUITY'] >= 70006.5]
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_
60	100071	0	Cash loans	F	N	Y	0	180000.0	1
112	100132	0	Cash loans	F	N	Y	0	202500.0	1
189	100219	0	Cash loans	M	N	Y	1	315000.0	2
191	100221	0	Cash loans	F	N	Y	0	225000.0	
485	100559	0	Cash loans	F	Y	Y	0	450000.0	2
...	...	...	...	...	...	...	...	...	
307002	455682	0	Cash loans	M	Y	N	0	546250.5	1
307055	455739	0	Cash loans	F	N	Y	0	112500.0	2
307069	455759	0	Cash loans	F	N	Y	0	130500.0	1
307165	455868	0	Cash loans	F	Y	Y	0	337500.0	1
307392	456125	0	Cash loans	F	N	Y	0	315000.0	1

3081 rows × 73 columns

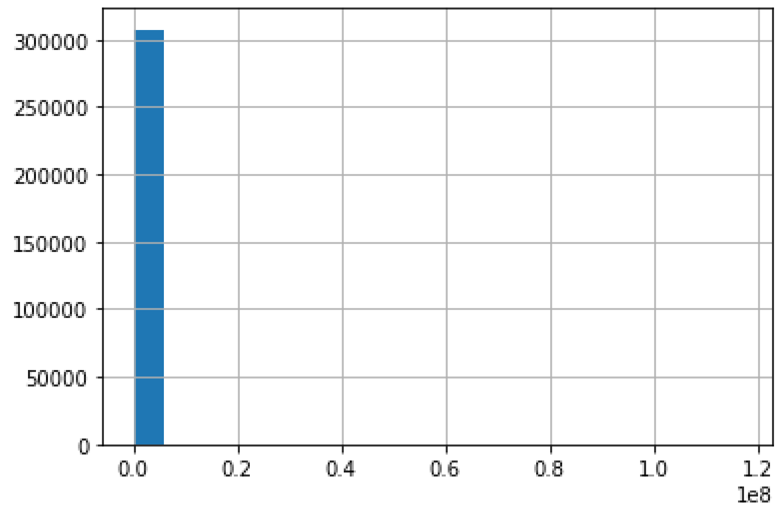
Analyzing – "AMT_INCOME_TOTAL"

Observation:

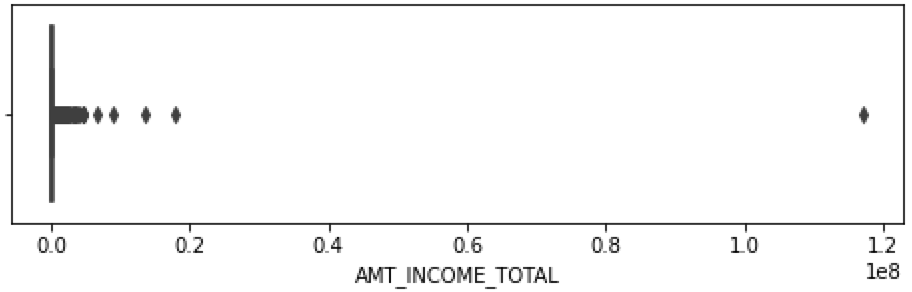
- **Lets draw a boxplot & histogram to analyze the spread of the data. We can observer that we have outliers available in this data. Also there are extreme outliers as well which mean there are few people with very high salary compared to the population.**

```
app_data['AMT_INCOME_TOTAL'].hist(bins=20)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x291110fabe0>
```




```
#Income variable has to be analysed as this may influence defaulter.
plt.figure(figsize = [8,2])
sns.boxplot(app_data['AMT_INCOME_TOTAL'])

plt.show()
```



Analyzing – "AMT_INCOME_TOTAL"

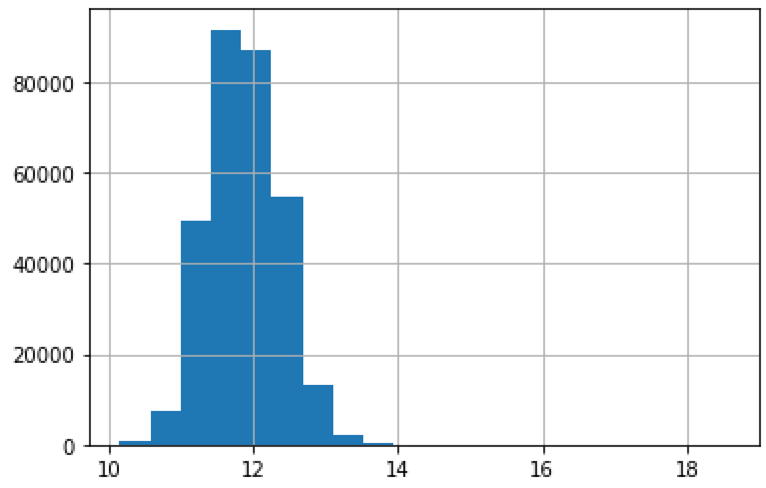
Observation:

- **Lets draw a boxplot & histogram to analyze the spread of the data. We can observe that we have outliers available in this data. Also there are extreme outliers as well which mean there are few people with very high salary compared to the population.**

- **If we use the natural logarithm (inverse exponential function) we can see that it turns out to be a normal distribution and it nullify the effect of outliers in whole data set.**

```
app_data['AMT_INCOME_TOTAL_LOG'] = np.log(app_data['AMT_INCOME_TOTAL'])
app_data['AMT_INCOME_TOTAL_LOG'].hist(bins=20)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2910fc51f40>
```



Analyzing – "NAME_CONTRACT_TYPE"

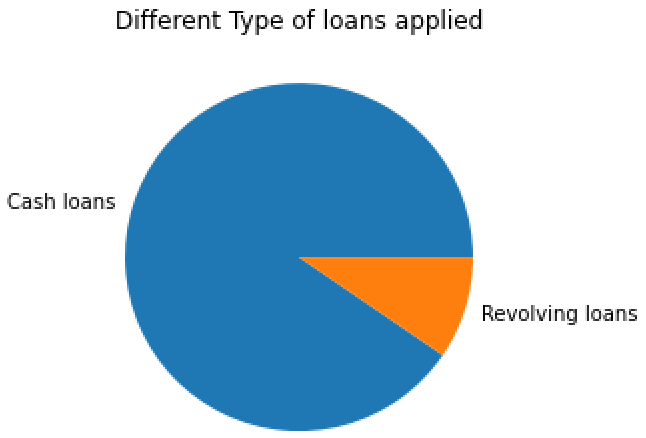
Observation:

- **Maximum people are applying for Cash Loans**

```
# NAME_CONTRACT_TYPE
app_data['NAME_CONTRACT_TYPE'].value_counts()
```

```
Cash loans         278232
Revolving loans     29279
Name: NAME_CONTRACT_TYPE, dtype: int64
```

```
app_data.NAME_CONTRACT_TYPE.value_counts(normalize= True).plot.pie(label = '')
plt.title('Different Type of loans applied')
plt.show()
```



Analyzing – “CODE_GENDER”

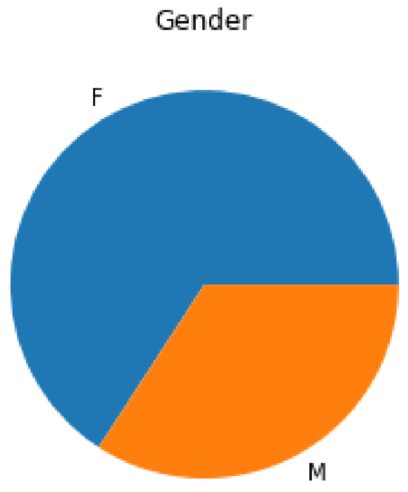
Observation:

- **Out of total population, Females are applying for loan higher than Male.**

```
# Gender.
app_data['CODE_GENDER'].value_counts()
```

```
F    202452
M    105059
Name: CODE_GENDER, dtype: int64
```

```
app_data.CODE_GENDER.value_counts(normalize= True).plot.pie(label = '')
plt.title('Gender')
plt.show()
```



Analyzing – "NAME_EDUCATION_TYPE"

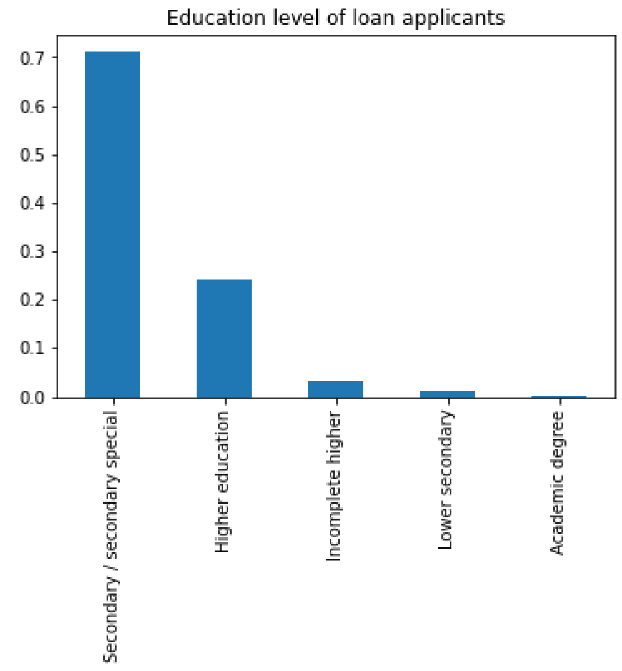
Observation:

- **Loans were mostly applied by "Secondary / secondary special" educated people**
- **People with education "Academic degree" and "Lower secondary" have rarely applied.**

```
# Education type.
app_data['NAME_EDUCATION_TYPE'].value_counts()
```

```
Secondary / secondary special    218391
Higher education                  74863
Incomplete higher                 10277
Lower secondary                    3816
Academic degree                     164
Name: NAME_EDUCATION_TYPE, dtype: int64
```

```
app_data.NAME_EDUCATION_TYPE.value_counts(normalize=True).plot.bar()
plt.title('Education level of loan applicants')
plt.show()
```

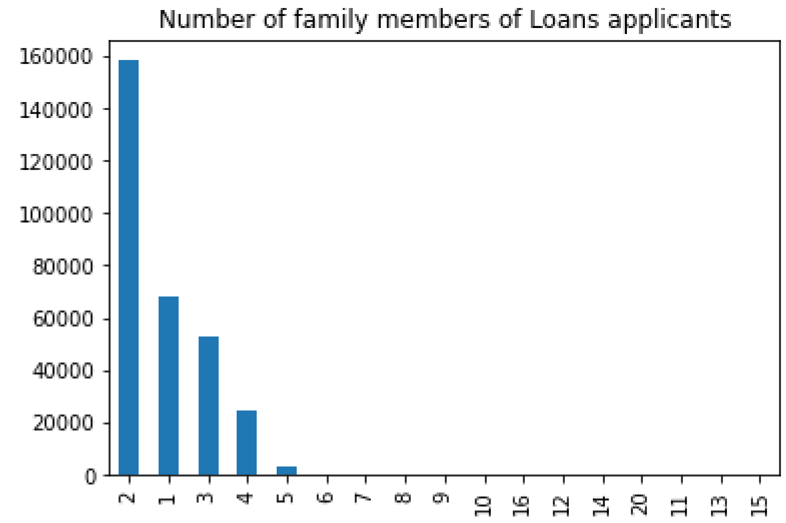


Analyzing – “CNT_FAM_MEMBERS”

Observation:

- **Most of the people who have applied for loans have 2 family members.**

```
# Family Members
app_data.CNT_FAM_MEMBERS.value_counts().plot.bar()
plt.title('Number of family members of Loans applicants')
plt.show()
```



Analyzing – "NAME_FAMILY_STATUS"

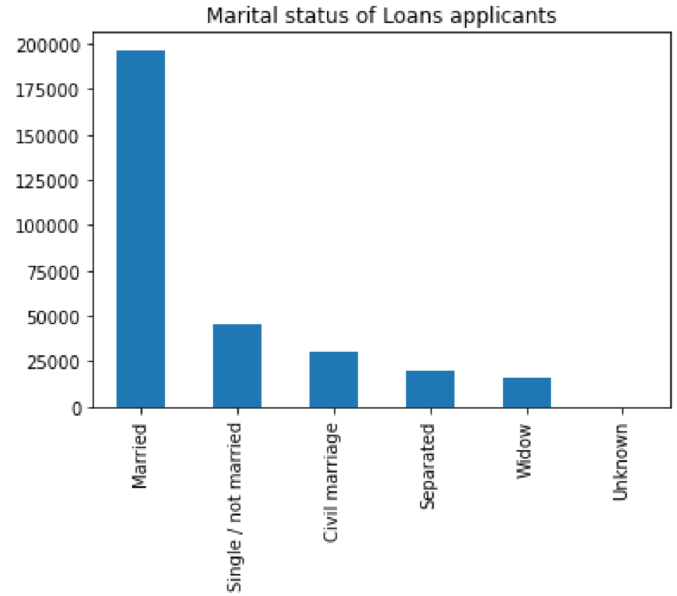
Observation:

- **Married people are applying for loans.**
- **Separated/widows seems to apply less for loans.**

```
app_data['NAME_FAMILY_STATUS'].value_counts()
```

```
Married                 196432
Single / not married     45444
Civil marriage           29775
Separated                19770
Widow                    16088
Unknown                      2
Name: NAME_FAMILY_STATUS, dtype: int64
```

```
app_data['NAME_FAMILY_STATUS'].value_counts().plot.bar()
plt.title('Marital status of Loans applicants')
plt.show()
```



Analyzing – "NAME_INCOME_TYPE"

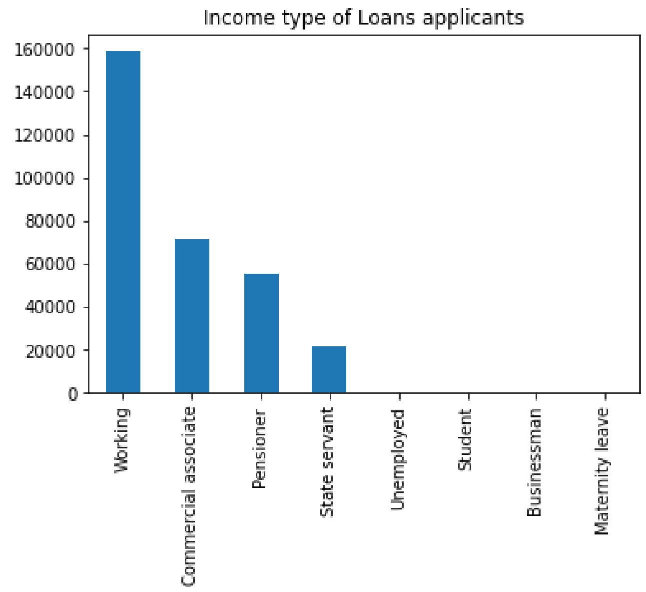
Observation:

- **Working people apply for more loans.**

```
#NAME_INCOME_TYPE.
app_data['NAME_INCOME_TYPE'].value_counts()
```

```
Working                 158774
Commercial associate     71617
Pensioner                55362
State servant            21703
Unemployed                  22
Student                     18
Businessman                 10
Maternity leave              5
Name: NAME_INCOME_TYPE, dtype: int64
```

```
app_data['NAME_INCOME_TYPE'].value_counts().plot.bar()
plt.title('Income type of Loans applicants')
plt.show()
```

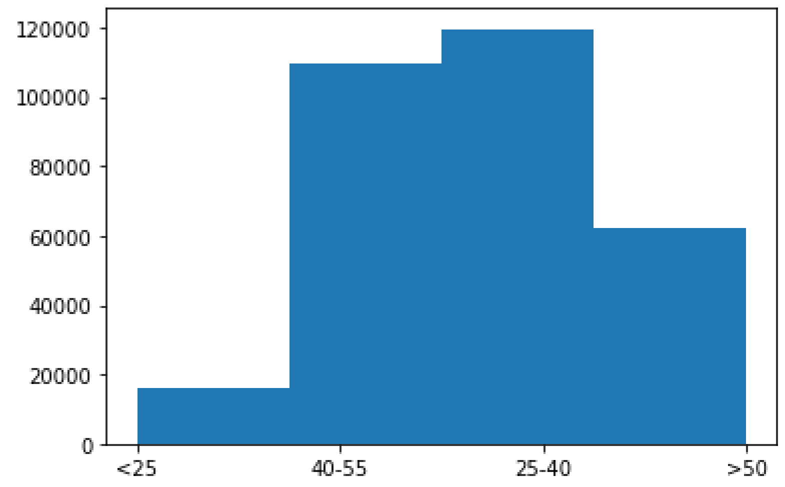


Analyzing – “DAYS_BIRTH_YEAR”

Observation:

- **We can clearly see that people with age between 25-40 years of age are highest to apply loans. 2nd highest is the people between age group 40-55.**

```
plt.hist(app_data['DAYS_BIRTH_YEAR'], bins = 4)
plt.show()
```



Bivariate Analysis

Correlations

Pearson's correlation coefficient is a statistical measure of the strength of a linear relationship between paired data. One of the effective way to try and understand the data is by looking for correlations between the features and the target. We can calculate the Pearson correlation coefficient between every variable and the target using the .corr dataframe method.

Furthermore:

- **Positive values denote positive linear correlation**
- **Negative values denote negative linear correlation**
- **A value of 0 denotes no linear correlation**
- **The closer the value is to 1 or –1, the stronger the linear correlation.**

The correlation coefficient gives us an idea of possible relationships within the data. Some general interpretations of the absolute value of the correlation coefficent are:

- **.00-.19 "very weak"**
- **.20-.39 "weak"**
- **.40-.59 "moderate"**
- **.60-.79 "strong"**
- **.80-1.0 "very strong"**

Correlations

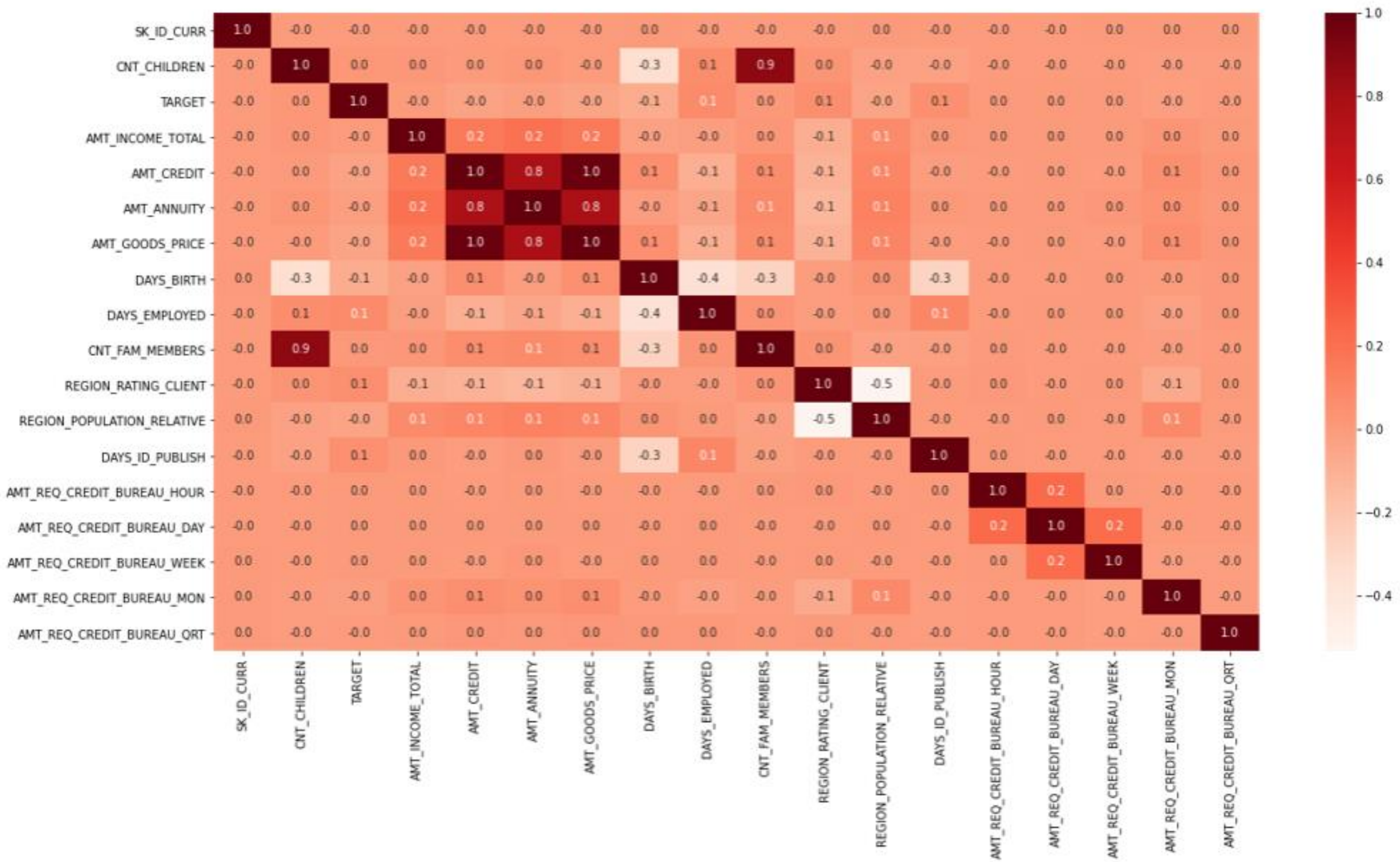
```
# Correlation of numeric variables
Def_corr = app_data[['SK_ID_CURR','CNT_CHILDREN','TARGET','AMT_INCOME_TOTAL','AMT_CREDIT','AMT_ANNUITY','AMT_GOODS_PRICE',
                    'DAYS_BIRTH','DAYS_EMPLOYED','CNT_FAM_MEMBERS','REGION_RATING_CLIENT','REGION_POPULATION_RELATIVE',
                    'DAYS_ID_PUBLISH','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']]
```

```
correlations = Def_corr.corr()['TARGET'].sort_values()
```

```
print('Most Positive Correlations:\n', correlations.tail(8))
print('\nMost Negative Correlations:\n', correlations.head(8))
```

```
Most Positive Correlations:
 AMT_REQ_CREDIT_BUREAU_HOUR    0.000930
AMT_REQ_CREDIT_BUREAU_DAY     0.002704
CNT_FAM_MEMBERS               0.009308
CNT_CHILDREN                  0.019187
DAYS_ID_PUBLISH               0.051457
REGION_RATING_CLIENT          0.058899
DAYS_EMPLOYED                 0.074958
TARGET                        1.000000
Name: TARGET, dtype: float64

Most Negative Correlations:
 DAYS_BIRTH                    -0.078239
AMT_GOODS_PRICE               -0.039645
REGION_POPULATION_RELATIVE    -0.037227
AMT_CREDIT                    -0.030369
AMT_ANNUITY                   -0.012815
AMT_REQ_CREDIT_BUREAU_MON     -0.012462
AMT_INCOME_TOTAL              -0.003982
SK_ID_CURR                    -0.002108
Name: TARGET, dtype: float64
```

```python
plt.figure(figsize=(20,10))
sns.heatmap(Def_corr.corr(),fmt='.1f',annot = True,cmap = "Reds")
plt.show()
```



Separating Dataset based on TARGET column

- **We have created 2 separate datasets “Defaulter” & “Non_Defaulter” using the “TARGET” column in original dataset to draw inferences**

```
Defaulter = app_data[app_data['TARGET'] == 1]
Defaulter.reset_index(inplace = True)
Defaulter

Non_Defaulter = app_data[app_data['TARGET'] == 0]
Non_Defaulter.reset_index(inplace = True)
Non_Defaulter
```

	index	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOT
0	1	100003	0	Cash loans	F	N	N	0	27000
1	2	100004	0	Revolving loans	M	Y	Y	0	6750
2	3	100006	0	Cash loans	F	N	Y	0	13500
3	4	100007	0	Cash loans	M	N	Y	0	12150
4	5	100008	0	Cash loans	M	N	Y	0	9900
...	...	...	...	...	...	...	...	...	
282681	307505	456249	0	Cash loans	F	N	Y	0	11250
282682	307506	456251	0	Cash loans	M	N	N	0	15750
282683	307507	456252	0	Cash loans	F	N	Y	0	7200
282684	307508	456253	0	Cash loans	F	N	Y	0	15300
282685	307510	456255	0	Cash loans	F	N	N	0	15750

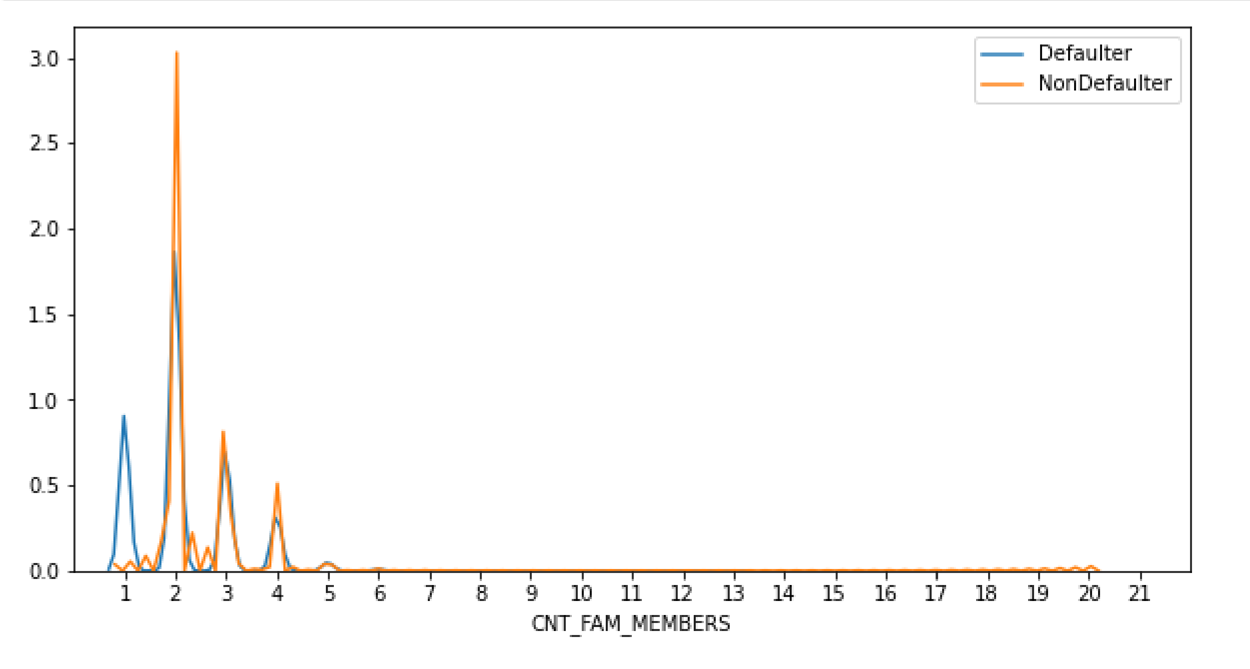
282686 rows × 76 columns

“CNT_FAM_MEMBERS” Vs “TARGET”

```
plt.figure(figsize=(20,10))
plt.xlim(0,22)
plt.xticks(range(1,22))
sns.distplot(Defaulter["CNT_FAM_MEMBERS"], hist=False, label="Defaulter")
sns.distplot(Non_Defaulter["CNT_FAM_MEMBERS"], hist=False, label="NonDefaulter")

plt.show()
```

Observation:

- **With given distribution plot we can see that people with 1 family members are tend to default the loan. People with 2 family members also have 60% changes to be defaulter.**
- **For families with more than 2 members have equal probability to be defaulter OR non-defaulter.**



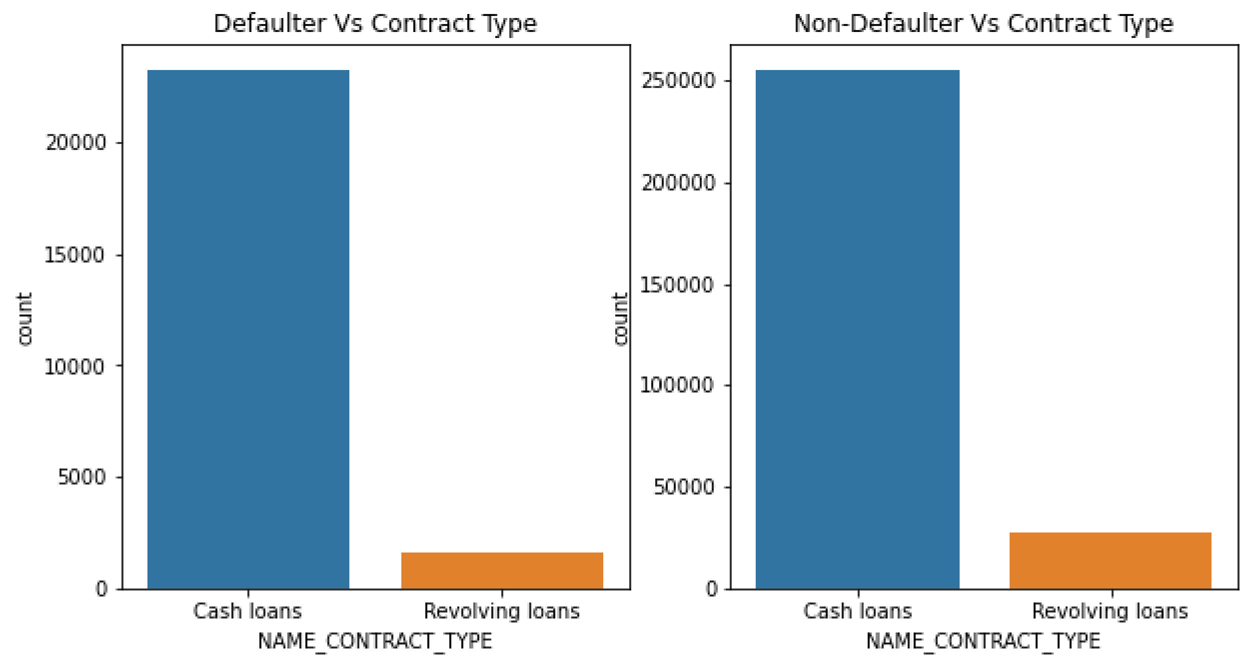
“NAME CONTRACT TYPE” Vs “TARGET”

Observation:

- By seeing this chart, we can clearly identify that there is a positive relation between “NAME_CONTRACT_TYPE” and “TARGET” column as cash loans are applied more hence higher probability to default/non-default in that category only

- Though we see that data difference in both the charts but that is totally normal as defaulter cases are far less than non-defaulter

```
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.title("Defaulter Vs Contract Type")
sns.countplot('NAME_CONTRACT_TYPE', data=Defaulter)
plt.subplot(1,2,2)
plt.title("Non-Defaulter Vs Contract Type")
sns.countplot('NAME_CONTRACT_TYPE', data=Non_Defaulter)
plt.show()
```



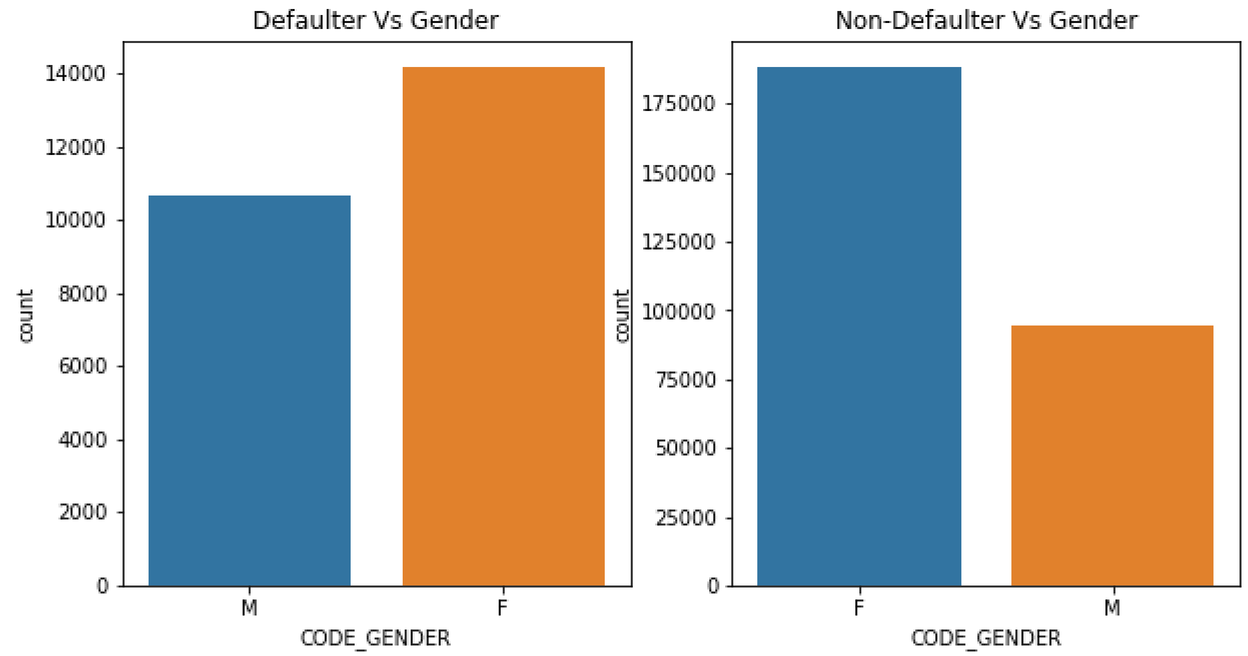
“CODE_GENDER” Vs “TARGET”

Observation:

- **There is a positive relation between “CODE_GENDER” and “TARGET” column as females tends to apply for loans more than males and females tends to default more than males.**

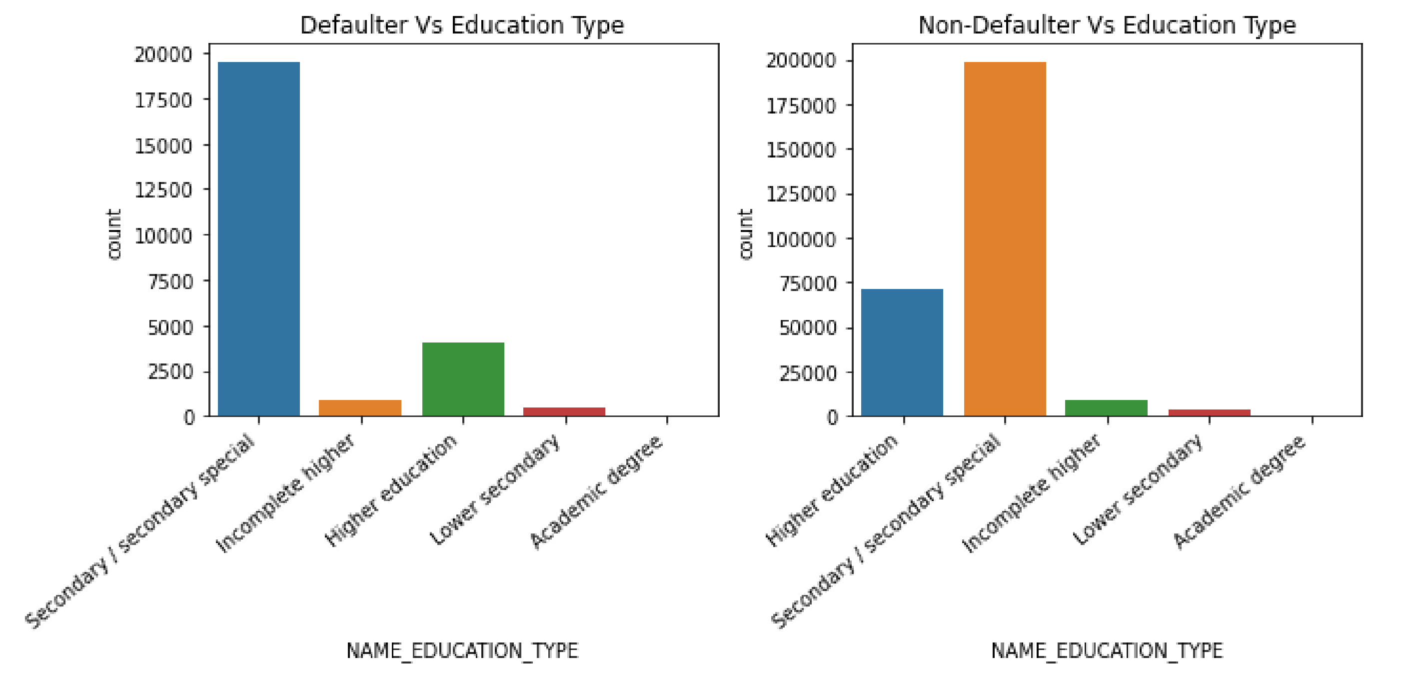
- **Though we see that data difference in both the charts but that is totally normal as defaulter cases are far less than non-defaulter**

```
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.title("Defaulter Vs Gender")
sns.countplot('CODE_GENDER', data=Defaulter)
plt.subplot(1,2,2)
plt.title("Non-Defaulter Vs Gender")
sns.countplot('CODE_GENDER', data=Non_Defaulter)
plt.show()
```



“NAME_EDUCATION_TYPE” Vs “TARGET”

```
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.title("Defaulter Vs Education Type")
ax=sns.countplot('NAME_EDUCATION_TYPE', data=Defaulter)
ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.subplot(1,2,2)
plt.title("Non-Defaulter Vs Education Type")
ax=sns.countplot('NAME_EDUCATION_TYPE', data=Non_Defaulter)
ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.show()
```

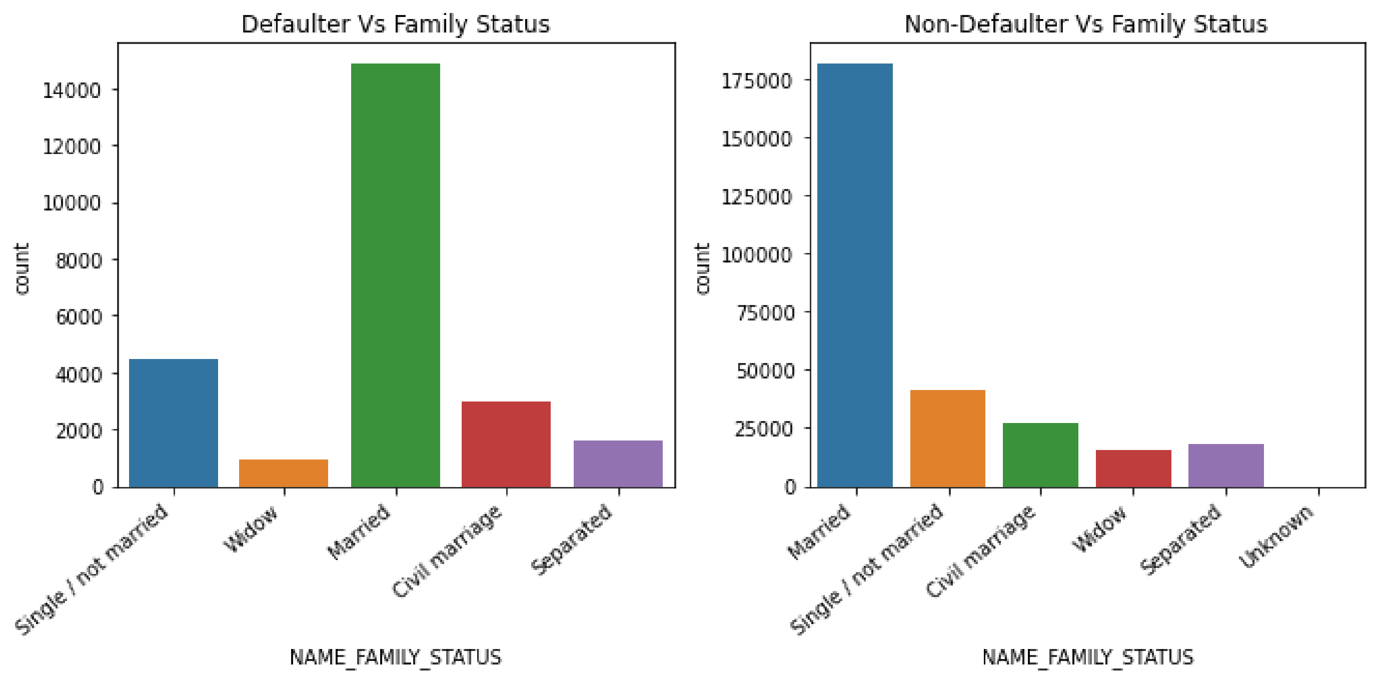


Observation:

- **By seeing this chart, we can clearly identify that there is a positive relation between “NAME_EDUCATION_TYPE” and “TARGET” column as people with secondary/secondary special education seems to apply for loans more and default as well**
- **Though we see that data difference in both the charts but that is totally normal as defaulter cases are far less than non-defaulter**

"NAME_FAMILY_STATUS" Vs "TARGET"

```
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.title("Defaulter Vs Family Status")
ax=sns.countplot('NAME_FAMILY_STATUS', data=Defaulter)
ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.subplot(1,2,2)
plt.title("Non-Defaulter Vs Family Status")
ax=sns.countplot('NAME_FAMILY_STATUS', data=Non_Defaulter)
ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.show()
```



Observation:

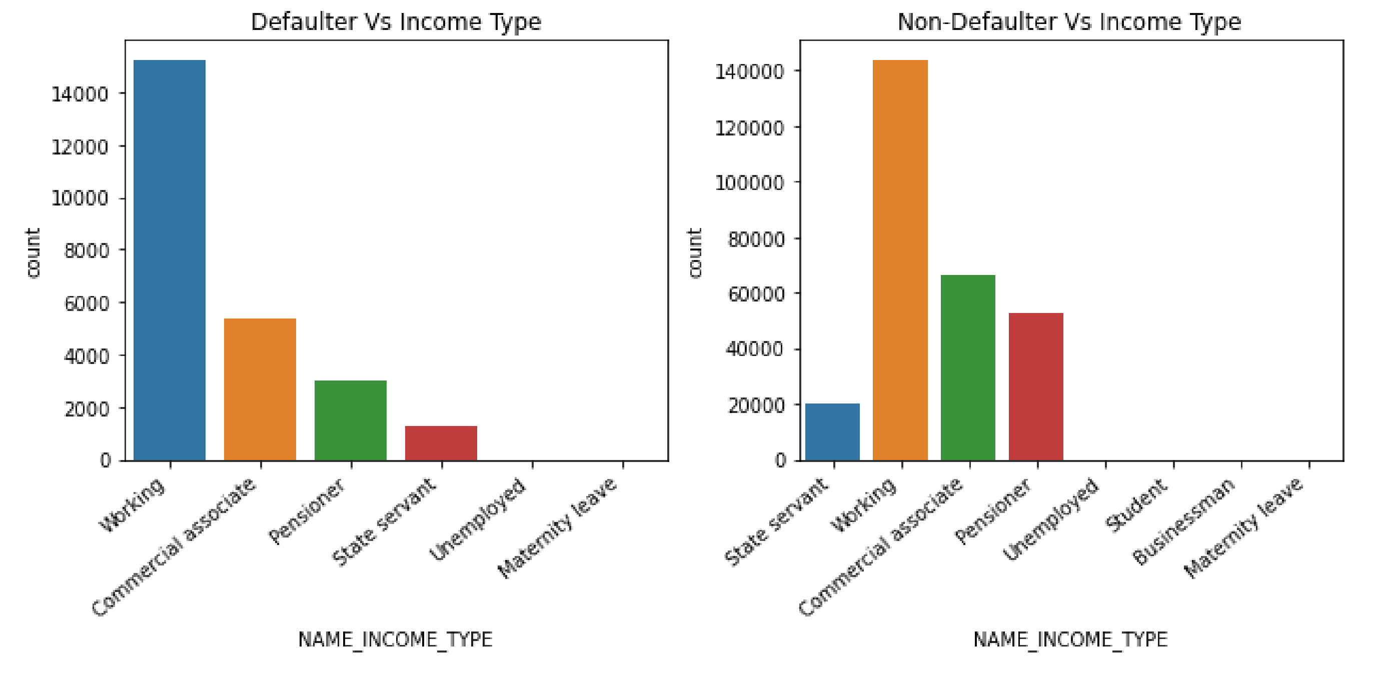
- **There is a positive relation between "NAME_FAMILY_STATUS" and "TARGET" column as married people seems to apply for loans more and default as well**
- **Though we see that data difference in both the charts but that is totally normal as defaulter cases are far less than non-defaulter**

“NAME_INCOME_TYPE” Vs “TARGET”

Observation:

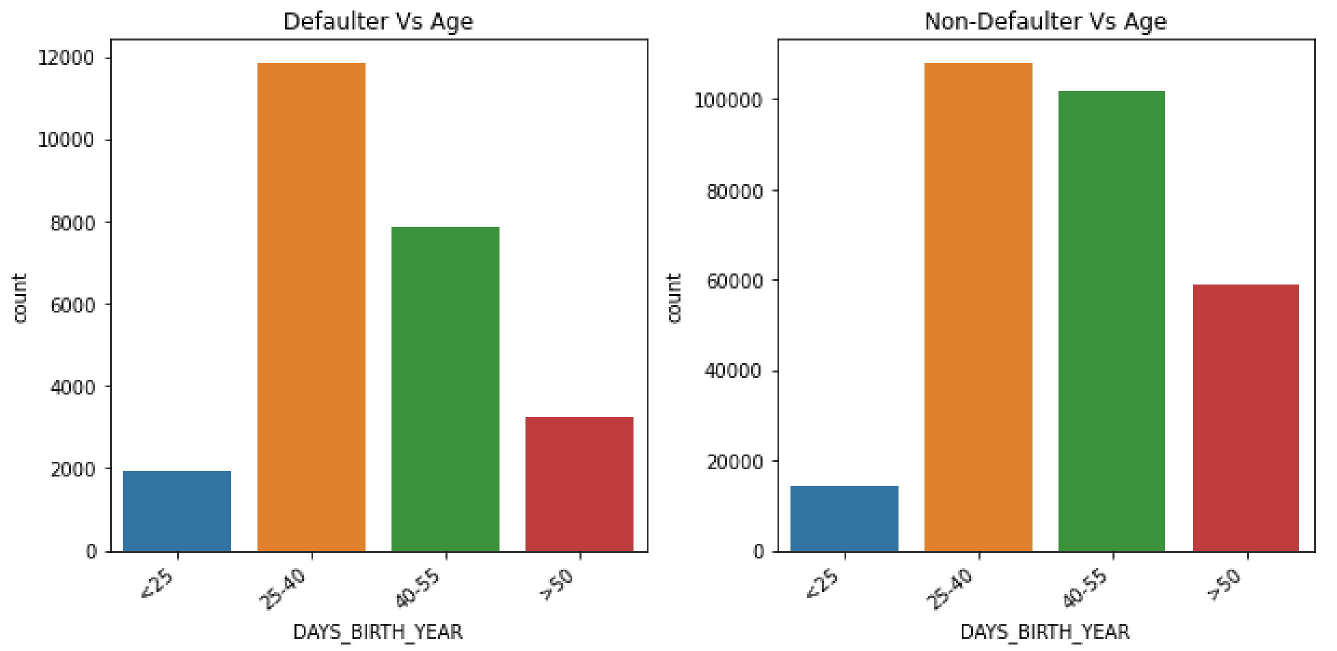
- **Again there is a positive relation between “NAME_INCOME_TYPE” and “TARGET” column as working people seems to apply for loans more and default as well**
- **Though we see that data difference in both the charts but that is totally normal as defaulter cases are far less than non-defaulter**
- **We also observe that there is no defaulter with education type as “Businessman” & “Student”, hence we should give loans to these people**

```
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.title("Defaulter Vs Income Type")
ax=sns.countplot('NAME_INCOME_TYPE', data=Defaulter)
ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.subplot(1,2,2)
plt.title("Non-Defaulter Vs Income Type")
ax=sns.countplot('NAME_INCOME_TYPE', data=Non_Defaulter)
ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.show()
```



“DAYS_BIRTH_YEAR” Vs “TARGET”

```
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.title("Defaulter Vs Age")
ax=sns.countplot('DAYS_BIRTH_YEAR', data=Defaulter)
ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.subplot(1,2,2)
plt.title("Non-Defaulter Vs Age")
ax=sns.countplot('DAYS_BIRTH_YEAR', data=Non_Defaulter)
ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.show()
```



Observation:

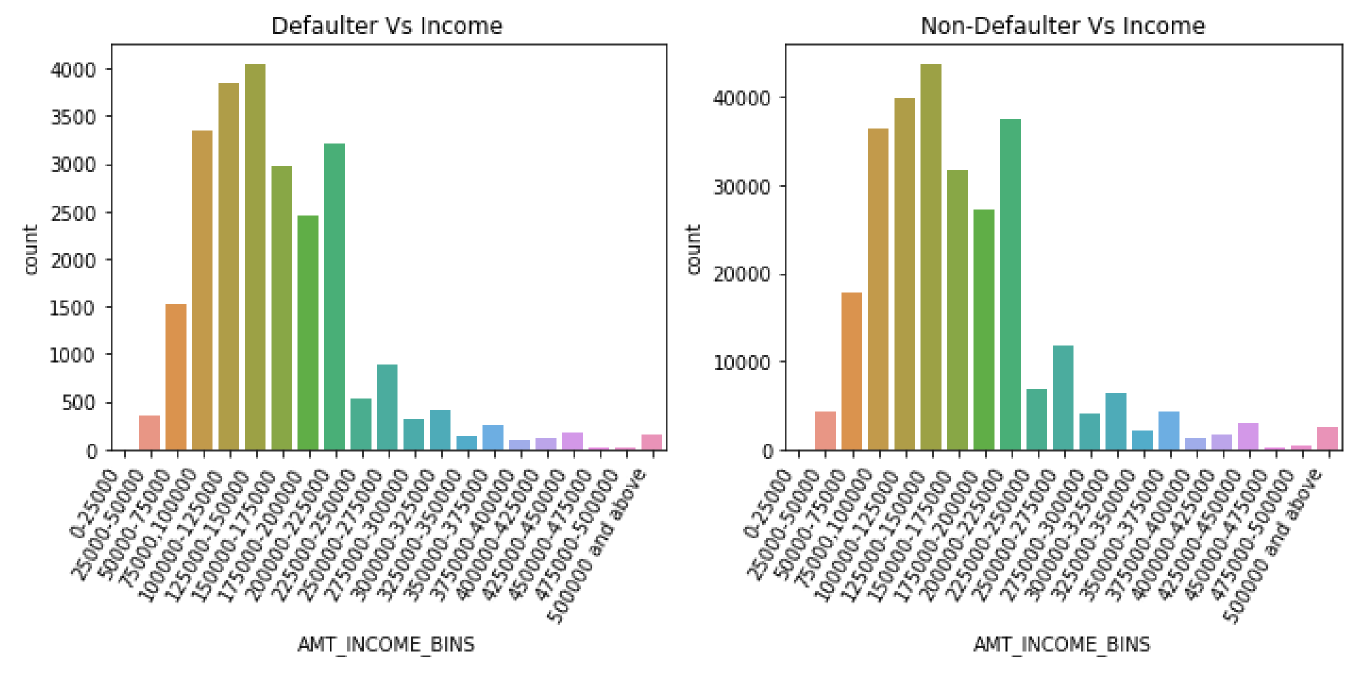
- As we can see there is a positive relation between “DAYS_BIRTH_YEAR” and “TARGET” column as people between age group of 25-40 tends to take more loans and default as well
- Though we see that data difference in both the charts but that is totally normal as defaulter cases are far less than non-defaulter

“AMT_INCOME_BINS” Vs “TARGET”

Observation:

- **Defaulters are mostly between income of 75,000 - 2,25,000**
- **As we can see there is a positive relation between “AMT_INCOME_BINS” and “TARGET” column as people who are earning more tend to apply for loans more and default as well**
- **Though we see that data difference in both the charts but that is totally normal as defaulter cases are far less than non-defaulter**

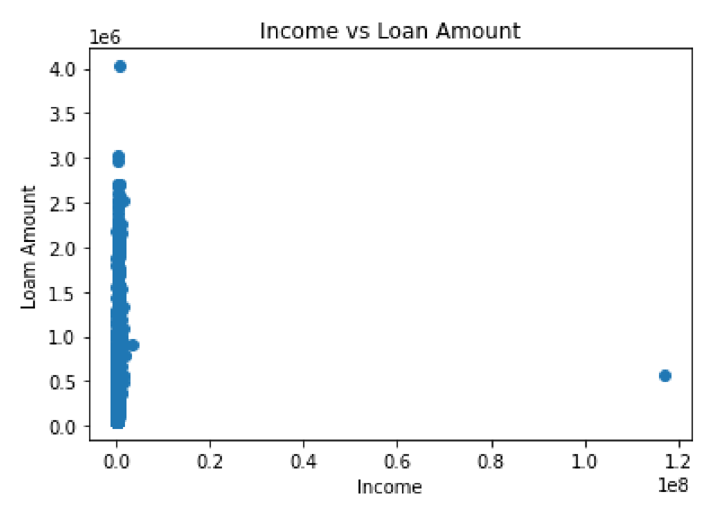
```
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.title("Defaulter Vs Income")
ax=sns.countplot('AMT_INCOME_BINS', data=Defaulter)
ax.set_xticklabels(ax.get_xticklabels(), rotation=60, ha="right")
plt.tight_layout()
plt.subplot(1,2,2)
plt.title("Non-Defaulter Vs Income")
ax=sns.countplot('AMT_INCOME_BINS', data=Non_Defaulter)
ax.set_xticklabels(ax.get_xticklabels(), rotation=60, ha="right")
plt.tight_layout()
plt.show()
```



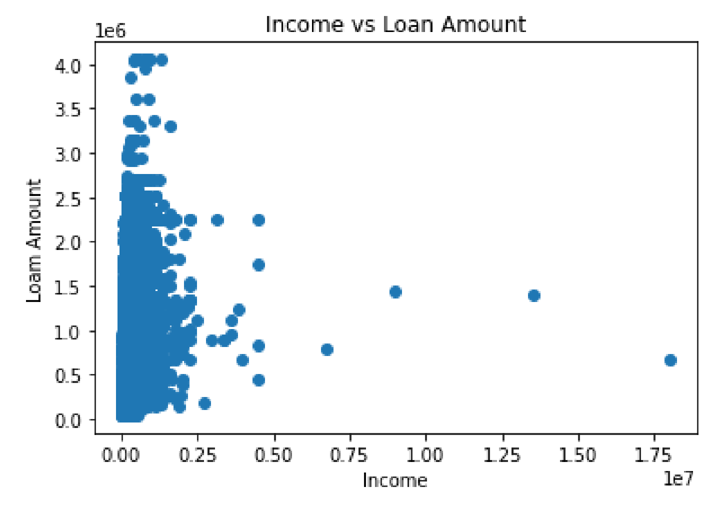
"AMT_INCOME_TOTAL" Vs "AMT_CREDIT"

Observation:

- **Trying to see if there is any relation between income and loan amount. so that we can see if income increases loan amount taken also increase.**
- **No inferences can be drawn from the above graph. Also as part of bivariate analysis between income vs target it was observed that defaulters are mostly between 75,0000 - 2,250000**




```
#Numeric - Numeric relation.
# Income vs Loan Amount
plt.scatter(Defaulter['AMT_INCOME_TOTAL'],Defaulter['AMT_CREDIT'])
plt.xlabel('Income')
plt.ylabel('Loam Amount')
plt.title('Income vs Loan Amount')
plt.show()

# Income vs Loan Amount
plt.scatter(Non_Defaulter['AMT_INCOME_TOTAL'],Non_Defaulter['AMT_CREDIT'])
plt.xlabel('Income')
plt.ylabel('Loam Amount')
plt.title('Income vs Loan Amount')
plt.show()
```



“AMT_INCOME_TOTAL” Vs “AMT_CREDIT”

Observation:

As we apply the log algorithm in total income and loan amount variables, we observe, from the scatter plot of Defaulters :-

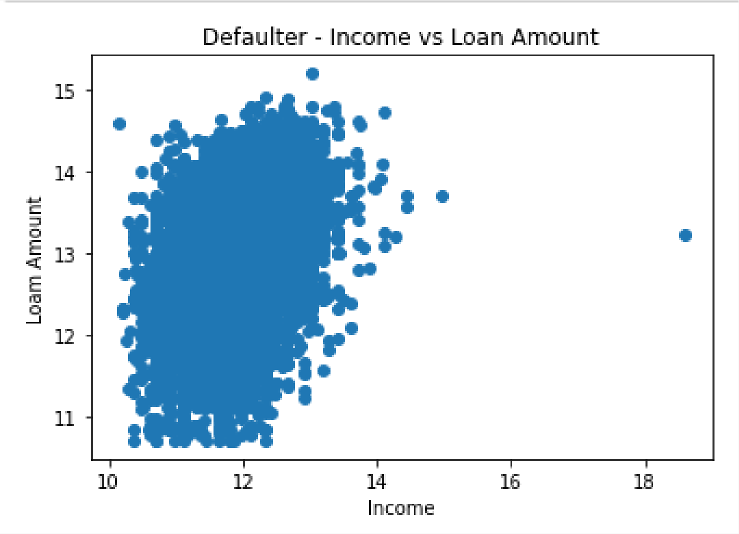
1 - The very last dot shows that a person with higher income took a higher amount loan and also defaulted the one

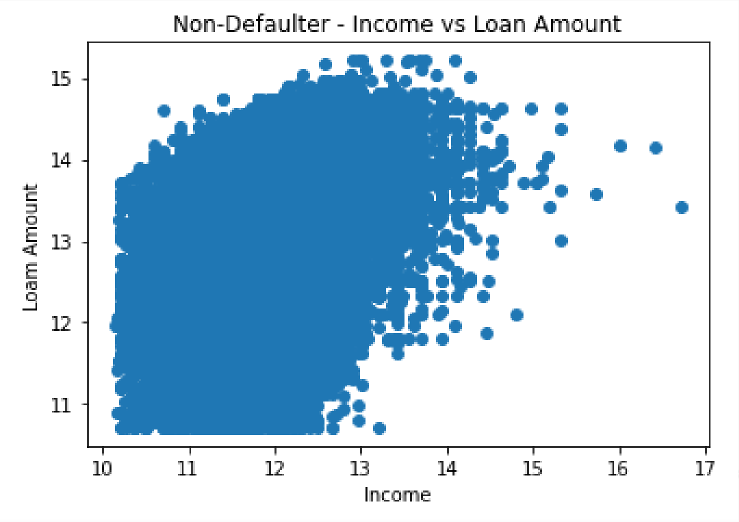
- **Also from the scatter plot of Non-defaulters, we can clearly see that as income of population increases**

  1 - People with a certain level of income on higher side, They don't tend to apply for loans frequently

  2 - Very few people with higher income are taking loans and that too of higher amount

```
# Income vs Loan Amount using log function to observe natural logarithm
plt.scatter(np.log(Defaulter['AMT_INCOME_TOTAL']),np.log(Defaulter['AMT_CREDIT']))
plt.xlabel('Income')
plt.ylabel('Loam Amount')
plt.title('Defaulter - Income vs Loan Amount')
plt.show()

# Income vs Loan Amount
plt.scatter(np.log(Non_Defaulter['AMT_INCOME_TOTAL']),np.log(Non_Defaulter['AMT_CREDIT']))
plt.xlabel('Income')
plt.ylabel('Loam Amount')
plt.title('Non-Defaulter - Income vs Loan Amount')
plt.show()
```





"AMT_CREDIT" Vs "TARGET"

Observation:

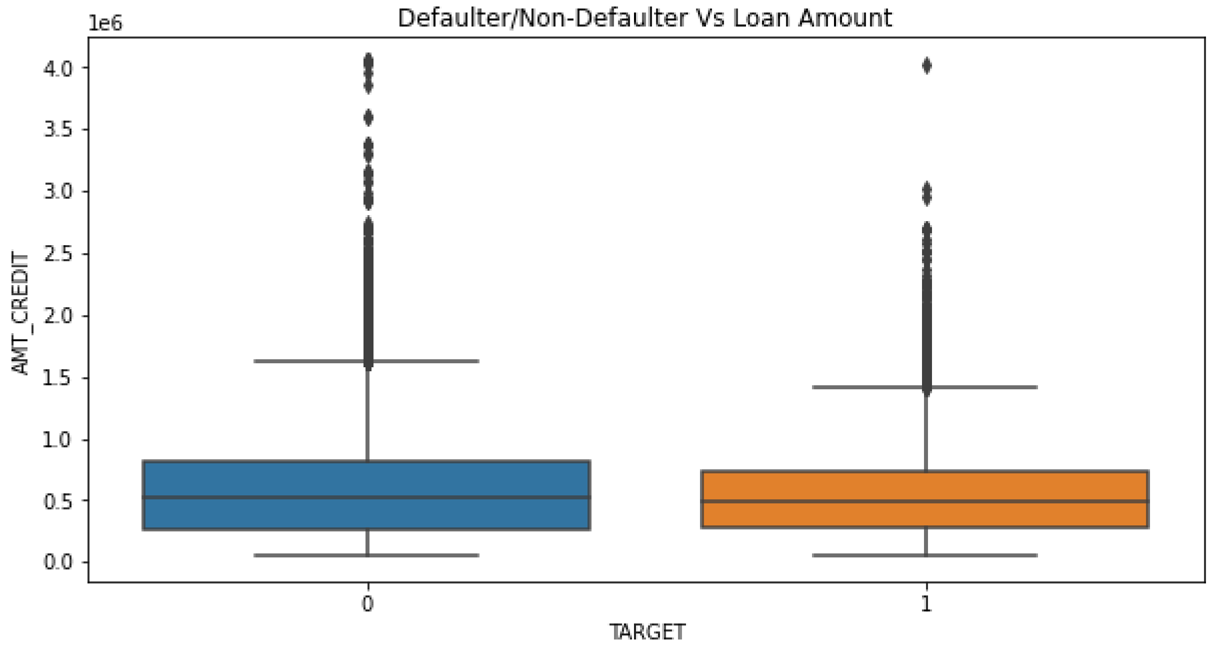
- **When we try to see the relation between loan amount and target variable (in Blue Boxplot), we see that many people from population tend to apply for higher amount of loan and repay as well.**
- **However there are very few people who are taking loan for the highest amount (in Orange Boxplot) and defaulting one as well**
- **We also tried to find mean, median and 75 percentile values of both the boxplot and we don't see a huge difference in these values for default and non-defaulter category.**

```
#Defaulter/non-defaulter vs loan_amount :  (Categorical vs numerical):
plt.figure(figsize=(10,5))
plt.title('Defaulter/Non-Defaulter Vs Loan Amount')
sns.boxplot(data=app_data,x='TARGET',y='AMT_CREDIT')
plt.show()
```

```
#function to find the 75th percentile.
def p75(x):
    return np.quantile(x, 0.75)
```

```
#calculate the mean, median and 75th percentile of loan amount with response
app_data.groupby("TARGET")["AMT_CREDIT"].aggregate(["mean","median",p75])
```

TARGET	mean	median	p75
0	602648.282002	517788.0	810000.0
1	557778.527674	497520.0	733315.5



Multivariate Analysis

"NAME_EDUCATION_TYPE" Vs "CODE_GENDER" Vs "TARGET"

Observation:

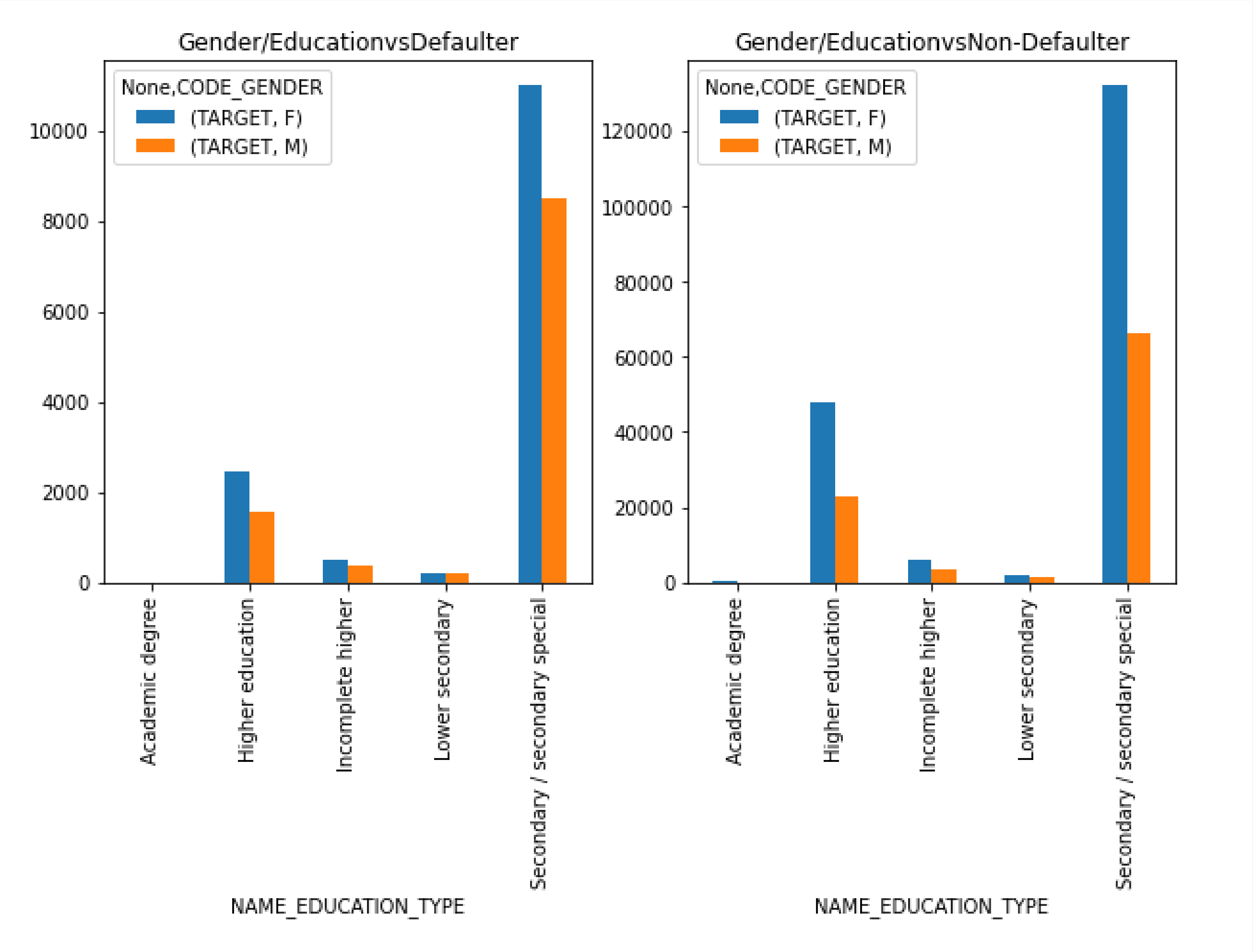
- **Count of females with education as secondary/secondary special are higher in both defaulter and non-defaulter list.**
- **Though we see that data difference in both the charts but that is totally normal as defaulter cases are far less than non-defaulter**

```
#Gender-Education- Target:

df = pd.DataFrame(app_data.groupby(['NAME_EDUCATION_TYPE','CODE_GENDER'])['TARGET'].sum())
print(df)

df1 = pd.DataFrame(Non_Defaulter.groupby(['NAME_EDUCATION_TYPE','CODE_GENDER'])['TARGET'].value_counts())
print(df1)

fig, ax = plt.subplots(ncols = 2,figsize=(10,5))
res = pd.pivot_table(df,index=['NAME_EDUCATION_TYPE'],columns=['CODE_GENDER'],values=['TARGET'])

ax[0].set(title ='Gender/EducationvsDefaulter')
res.plot(kind='bar',ax = ax[0])

res1 = pd.pivot_table(df1,index=['NAME_EDUCATION_TYPE'],columns=['CODE_GENDER'],values=['TARGET'])
ax[1].set(title ='Gender/EducationvsNon-Defaulter')
res1.plot(kind='bar',ax = ax[1])

plt.show()
```

“NAME_EDUCATION_TYPE” Vs “CODE_GENDER” Vs “TARGET”

Observation:

- **Count of females with education as secondary/secondary special are higher in both defaulter and non-defaulter list.**
- **Though we see that data difference in both the charts but that is totally normal as defaulter cases are far less than non-defaulter**



“NAME_FAMILY_STATUS” Vs “CODE_GENDER” Vs “TARGET”

Observation:

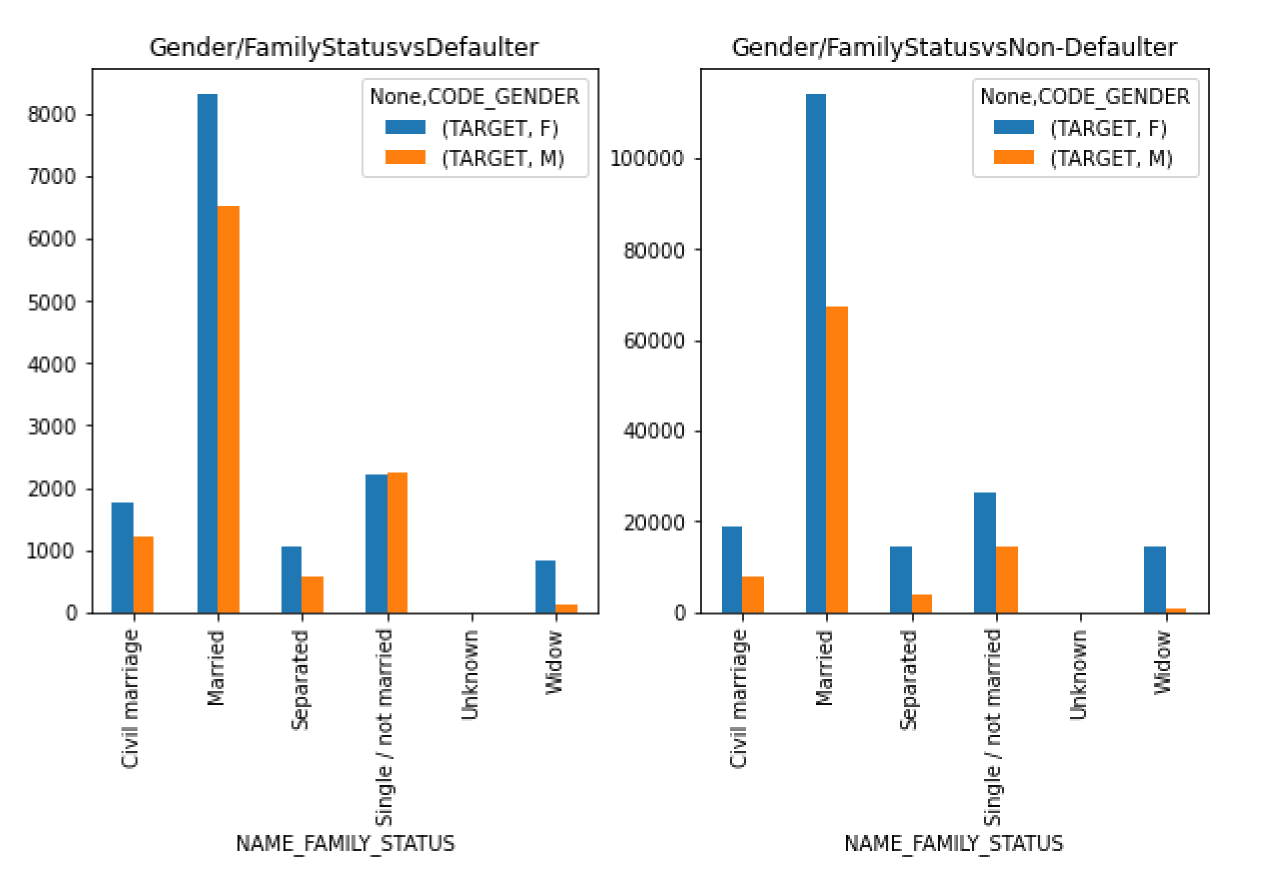
- **Count of females with family status as Married are higher in both defaulter and non-defaulter list.**
- **Though we see that data difference in both the charts but that is totally normal as defaulter cases are far less than non-defaulter**

```
#Gender-FamiyStatus-Target:

df = pd.DataFrame(app_data.groupby(['NAME_FAMILY_STATUS','CODE_GENDER'])['TARGET'].sum())
print(df)

df1 = pd.DataFrame(Non_Defaulter.groupby(['NAME_FAMILY_STATUS','CODE_GENDER'])['TARGET'].value_counts())
print(df1)

fig, ax = plt.subplots(ncols = 2,figsize=(10,5))
res = pd.pivot_table(df,index=['NAME_FAMILY_STATUS'],columns=['CODE_GENDER'],values=['TARGET'])
ax[0].set(title ='Gender/FamilyStatusvsDefaulter')
res.plot(kind='bar',ax=ax[0])

res1 = pd.pivot_table(df1,index=['NAME_FAMILY_STATUS'],columns=['CODE_GENDER'],values=['TARGET'])
ax[1].set(title ='Gender/FamilyStatusvsNon-Defaulter')
res1.plot(kind='bar',ax = ax[1])

plt.show()
```

"NAME_FAMILY_STATUS" Vs "CODE_GENDER" Vs "TARGET"

Observation:

- **Count of females with family status as Married are higher in both defaulter and non-defaulter list.**
- **Though we see that data difference in both the charts but that is totally normal as defaulter cases are far less than non-defaulter**

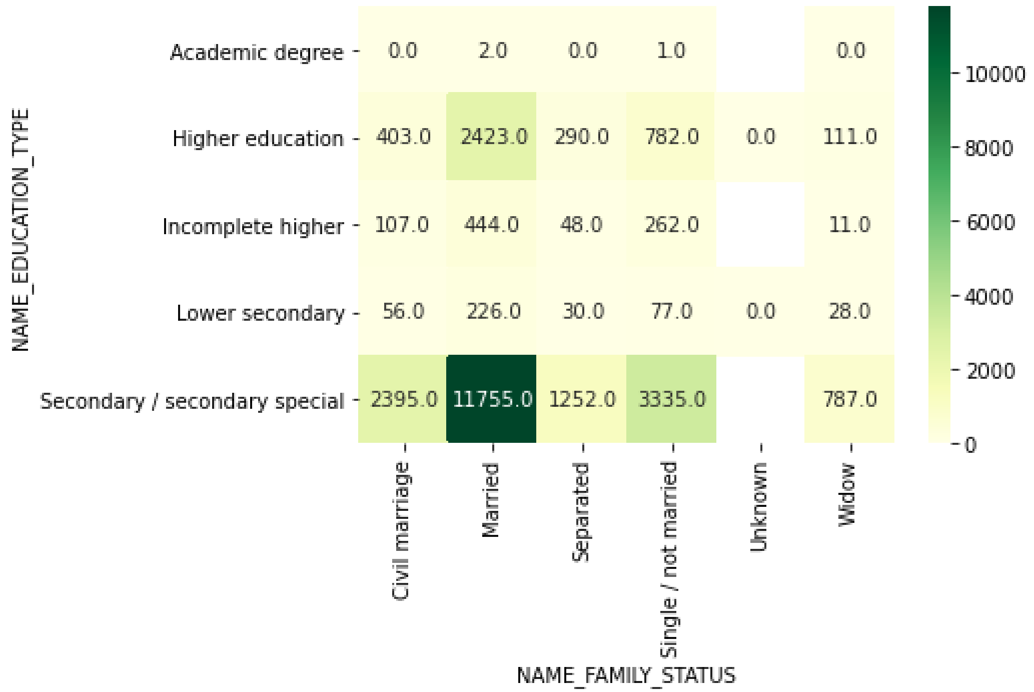


"NAME_FAMILY_STATUS" Vs "NAME_EDUCATION_TYPE" Vs "TARGET"

Observation:

- **Married People with secondary/secondary special education, tend to default more.**

```
#Education vs family status
res = pd.pivot_table(app_data,index = "NAME_EDUCATION_TYPE",columns = "NAME_FAMILY_STATUS",values = "TARGET", aggfunc=np.sum)
sns.heatmap(res,fmt='.1f',annot = True,cmap = "YlGn")
plt.show()
```

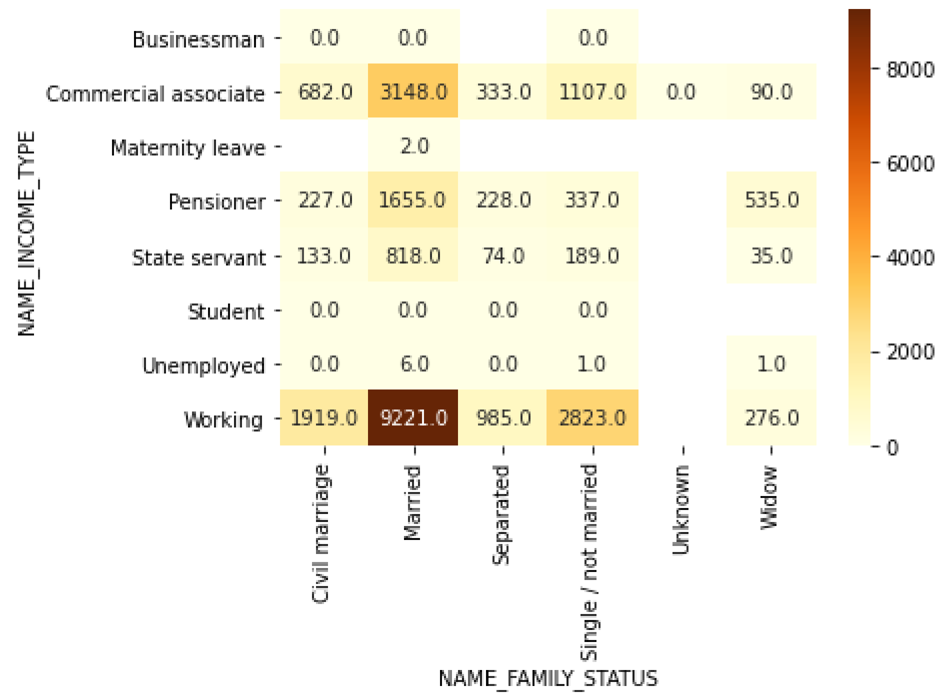


“NAME_FAMILY_STATUS” Vs “NAME_INCOME_TYPE” Vs “TARGET”

Observation:

- **Married People who are working as well, tend to default more.**

```python
#Income vs Family Status
res = pd.pivot_table(app_data,index = "NAME_INCOME_TYPE",columns = "NAME_FAMILY_STATUS",values = "TARGET", aggfunc=np.sum)
sns.heatmap(res,fmt='.1f', annot = True,cmap = "YlOrBr")
plt.show()
```



Analysis on Previous Application Dataset

Understand the Problem
&
Read/Examine the Dataset

Import Libraries and Read Data

Imports

```
# import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

Read The Data

```
#read application data
app_data = pd.read_csv('application_data.csv')
#read customer previous application data
prev_data = pd.read_csv('previous_application.csv')
```

Check the loaded Data

```
prev_data.head()
```

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

5 rows × 37 columns

Data Quality Check
&
Missing Values

Identifying & Treating Missing Values

- **We could see there are some columns with significant amount of null values. Either a column is Numerical or Categorical, we can delete the observations having null values in the dataset or the column that is having more number of null values # i.e. more than half or 30%.**

References for handling NULL Values - (https://medium.com/bycodegarage/a-comprehensive-guide-on-handling-missing-values-b1257a4866d1)

```
#Handling null values.
#calculate the percentage of null values in columns.
# Drop the columns with more than 30% of null values.

cols_null = prev_data.isnull().sum()/len(app_data)*100
cols_null = cols_null[cols_null.values > 30.0]
print(len(cols_null))
print(cols_null)
```

```
14
AMT_ANNUITY                  121.047702
AMT_DOWN_PAYMENT             291.320961
AMT_GOODS_PRICE              125.366247
RATE_DOWN_PAYMENT            291.320961
RATE_INTEREST_PRIMARY        541.204380
RATE_INTEREST_PRIVILEGED     541.204380
NAME_TYPE_SUITE              266.788830
CNT_PAYMENT                  121.046076
DAYS_FIRST_DRAWING           218.875097
DAYS_FIRST_DUE               218.875097
DAYS_LAST_DUE_1ST_VERSION    218.875097
DAYS_LAST_DUE                218.875097
DAYS_TERMINATION             218.875097
NFLAG_INSURED_ON_APPROVAL    218.875097
dtype: float64
```

```
# fetch the columns with 30% or more null values.
cols_null = list(cols_null[cols_null.values > 30.0].index)
cols_null
```

```
['AMT_ANNUITY',
 'AMT_DOWN_PAYMENT',
 'AMT_GOODS_PRICE',
 'RATE_DOWN_PAYMENT',
 'RATE_INTEREST_PRIMARY',
 'RATE_INTEREST_PRIVILEGED',
 'NAME_TYPE_SUITE',
 'CNT_PAYMENT',
 'DAYS_FIRST_DRAWING',
 'DAYS_FIRST_DUE',
 'DAYS_LAST_DUE_1ST_VERSION',
 'DAYS_LAST_DUE',
 'DAYS_TERMINATION',
 'NFLAG_INSURED_ON_APPROVAL']
```

```
# Drop the columns:
prev_data.drop(columns=cols_null,axis=1,inplace=True)
```

```
prev_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 23 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   SK_ID_PREV                   1670214 non-null  int64
 1   SK_ID_CURR                   1670214 non-null  int64
 2   NAME_CONTRACT_TYPE           1670214 non-null  object
 3   AMT_APPLICATION              1670214 non-null  float64
 4   AMT_CREDIT                   1670213 non-null  float64
 5   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object
 6   HOUR_APPR_PROCESS_START      1670214 non-null  int64
 7   FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object
 8   NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64
 9   NAME_CASH_LOAN_PURPOSE       1670214 non-null  object
 10  NAME_CONTRACT_STATUS         1670214 non-null  object
 11  DAYS_DECISION                1670214 non-null  int64
 12  NAME_PAYMENT_TYPE            1670214 non-null  object
 13  CODE_REJECT_REASON           1670214 non-null  object
 14  NAME_CLIENT_TYPE             1670214 non-null  object
 15  NAME_GOODS_CATEGORY          1670214 non-null  object
 16  NAME_PORTFOLIO               1670214 non-null  object
 17  NAME_PRODUCT_TYPE            1670214 non-null  object
 18  CHANNEL_TYPE                 1670214 non-null  object
 19  SELLERPLACE_AREA             1670214 non-null  int64
 20  NAME_SELLER_INDUSTRY         1670214 non-null  object
 21  NAME_YIELD_GROUP             1670214 non-null  object
 22  PRODUCT_COMBINATION          1669868 non-null  object
dtypes: float64(2), int64(6), object(15)
memory usage: 293.1+ MB
```

```
prev_data.shape
```

```
(1670214, 23)
```

Identifying & Treating Missing Values

- **We could see there are some columns with significant amount of null values. Either a column is Numerical or Categorical, we can delete the observations having null values in the dataset or the column that is having more number of null values # i.e. more than half or 30%.**
- **References for handling NULL Values - (https://medium.com/bycodegarage/a-comprehensive-guide-on-handling-missing-values-b1257a4866d1)**
- **Only “AMT_CREDIT" & "PRODUCT_COMBINATION" column seems to have null values and number is very low, hence we don’t need to handle these anymore**

```
#Handling null values.
#calculate the percentage of null values in columns.
# Drop the columns with more than 30% of null values.

cols_null = prev_data.isnull().sum()/len(app_data)*100
cols_null = cols_null[cols_null.values > 30.0]
print(len(cols_null))
print(cols_null)
```

```
14
AMT_ANNUITY                  121.047702
AMT_DOWN_PAYMENT             291.320961
AMT_GOODS_PRICE              125.366247
RATE_DOWN_PAYMENT            291.320961
RATE_INTEREST_PRIMARY        541.204380
RATE_INTEREST_PRIVILEGED     541.204380
NAME_TYPE_SUITE              266.788830
CNT_PAYMENT                  121.046076
DAYS_FIRST_DRAWING           218.875097
DAYS_FIRST_DUE               218.875097
DAYS_LAST_DUE_1ST_VERSION    218.875097
DAYS_LAST_DUE                218.875097
DAYS_TERMINATION             218.875097
NFLAG_INSURED_ON_APPROVAL    218.875097
dtype: float64
```

```
# fetch the columns with 30% or more null values.
cols_null = list(cols_null[cols_null.values > 30.0].index)
cols_null
```

```
['AMT_ANNUITY',
 'AMT_DOWN_PAYMENT',
 'AMT_GOODS_PRICE',
 'RATE_DOWN_PAYMENT',
 'RATE_INTEREST_PRIMARY',
 'RATE_INTEREST_PRIVILEGED',
 'NAME_TYPE_SUITE',
 'CNT_PAYMENT',
 'DAYS_FIRST_DRAWING',
 'DAYS_FIRST_DUE',
 'DAYS_LAST_DUE_1ST_VERSION',
 'DAYS_LAST_DUE',
 'DAYS_TERMINATION',
 'NFLAG_INSURED_ON_APPROVAL']
```

```
# Drop the columns:
prev_data.drop(columns=cols_null,axis=1,inplace=True)
```

```
prev_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 23 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   SK_ID_PREV                   1670214 non-null  int64
 1   SK_ID_CURR                   1670214 non-null  int64
 2   NAME_CONTRACT_TYPE           1670214 non-null  object
 3   AMT_APPLICATION              1670214 non-null  float64
 4   AMT_CREDIT                   1670213 non-null  float64
 5   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object
 6   HOUR_APPR_PROCESS_START      1670214 non-null  int64
 7   FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object
 8   NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64
 9   NAME_CASH_LOAN_PURPOSE       1670214 non-null  object
 10  NAME_CONTRACT_STATUS         1670214 non-null  object
 11  DAYS_DECISION                1670214 non-null  int64
 12  NAME_PAYMENT_TYPE            1670214 non-null  object
 13  CODE_REJECT_REASON           1670214 non-null  object
 14  NAME_CLIENT_TYPE             1670214 non-null  object
 15  NAME_GOODS_CATEGORY          1670214 non-null  object
 16  NAME_PORTFOLIO               1670214 non-null  object
 17  NAME_PRODUCT_TYPE            1670214 non-null  object
 18  CHANNEL_TYPE                 1670214 non-null  object
 19  SELLERPLACE_AREA             1670214 non-null  int64
 20  NAME_SELLER_INDUSTRY         1670214 non-null  object
 21  NAME_YIELD_GROUP             1670214 non-null  object
 22  PRODUCT_COMBINATION          1669868 non-null  object
dtypes: float64(2), int64(6), object(15)
memory usage: 293.1+ MB
```

```
prev_data.shape
```

```
(1670214, 23)
```

Identifying & Treating Missing Values

- **Missing values may not be present always as null. "XNA" & "XAP" is also a missing value. Since NAME_CASH_LOAN_PURPOSE is a categorical column and number of missing rows is again more than 30%, hence deleting these for further analysis.**

```
# Checking few categorical columns for null values.
prev_data.NAME_CASH_LOAN_PURPOSE.value_counts()
```

```
XAP                                 922661
XNA                                 677918
Repairs                              23765
Other                                15608
Urgent needs                          8412
Buying a used car                     2888
Building a house or an annex          2693
Everyday expenses                     2416
Medicine                              2174
Payments on other loans               1931
Education                             1573
Journey                               1239
Purchase of electronic equipment      1061
Buying a new car                      1012
Wedding / gift / holiday               962
Buying a home                          865
Car repairs                            797
Furniture                              749
Buying a holiday home / land           533
Business development                   426
Gasification / water supply            300
Buying a garage                        136
Hobby                                   55
Money for a third person                25
Refusal to name the goal                15
Name: NAME_CASH_LOAN_PURPOSE, dtype: int64
```

```
# Following the rule to drop rows if more than 30% contains null values.
prev_data = prev_data[-prev_data['NAME_CASH_LOAN_PURPOSE'].isin(['XAP','XNA'])]
```

Identifying & Treating Missing Values

```
# Following the rule to drop rows if more than 30% contains null values.
prev_data = prev_data[-prev_data['NAME_CASH_LOAN_PURPOSE'].isin(['XAP','XNA'])]
```

```
# Check data
prev_data.NAME_CASH_LOAN_PURPOSE.value_counts()
```

```
Repairs                             23765
Other                               15608
Urgent needs                         8412
Buying a used car                    2888
Building a house or an annex         2693
Everyday expenses                    2416
Medicine                             2174
Payments on other loans              1931
Education                            1573
Journey                              1239
Purchase of electronic equipment     1061
Buying a new car                     1012
Wedding / gift / holiday              962
Buying a home                         865
Car repairs                           797
Furniture                             749
Buying a holiday home / land          533
Business development                  426
Gasification / water supply           300
```

- **"NAME_CONTRACT_TYPE" – No handling required in this column.**

```
prev_data.NAME_CONTRACT_TYPE.value_counts()
```

```
Cash loans    69635
Name: NAME_CONTRACT_TYPE, dtype: int64
```

Identifying & Treating Missing Values

- **"NAME_CLIENT_TYPE" has "XNA" values, hence replacing this with mode. As we checked the value counts of this column, we can see that fresh loans are proportionally low than repeater.**

```
prev_data.NAME_CLIENT_TYPE.value_counts()
```

```
Repeater     56256
New           9964
Refreshed     3362
XNA             53
Name: NAME_CLIENT_TYPE, dtype: int64
```

```
# Replace XNA with most occuring value --- see if it can be replaced with repeater.
client_type = prev_data.NAME_CLIENT_TYPE.mode()[0]
prev_data.NAME_CLIENT_TYPE = prev_data.NAME_CLIENT_TYPE.replace('XNA',client_type)
```

```
prev_data.NAME_CLIENT_TYPE.value_counts()
```

```
Repeater     56309
New           9964
Refreshed     3362
Name: NAME_CLIENT_TYPE, dtype: int64
```

Identifying & Treating Missing Values

- **"NAME_PAYMENT_TYPE" has "XNA" values, hence replacing this with mode and drawing horizontal bar plot for the same. We can see that mostly loans are applied via "cash through the bank".**

```
prev_data.NAME_PAYMENT_TYPE.value_counts()
```

```
Cash through the bank                          63835
XNA                                             5416
Non-cash from your account                       320
Cashless from the account of the employer         64
Name: NAME_PAYMENT_TYPE, dtype: int64
```

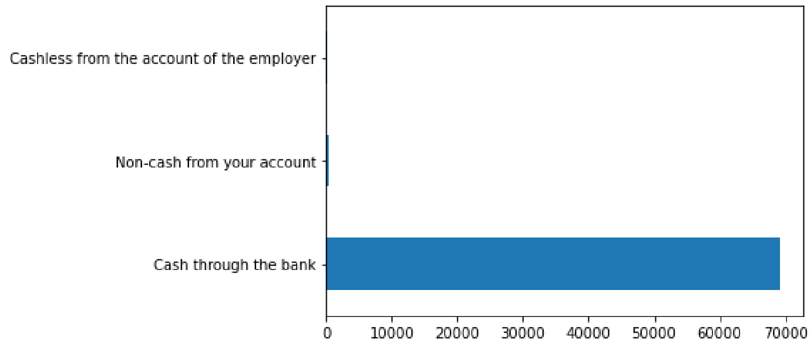
Replace XNA with most occuring value

```
payment_type = prev_data.NAME_PAYMENT_TYPE.mode()[0]
payment_type
prev_data.NAME_PAYMENT_TYPE = prev_data.NAME_PAYMENT_TYPE.replace('XNA',payment_type)
```

```
prev_data.NAME_PAYMENT_TYPE.value_counts()
```

```
Cash through the bank                          69251
Non-cash from your account                       320
Cashless from the account of the employer         64
Name: NAME_PAYMENT_TYPE, dtype: int64
```

```
prev_data.NAME_PAYMENT_TYPE.value_counts().plot.barh()
plt.show()
```



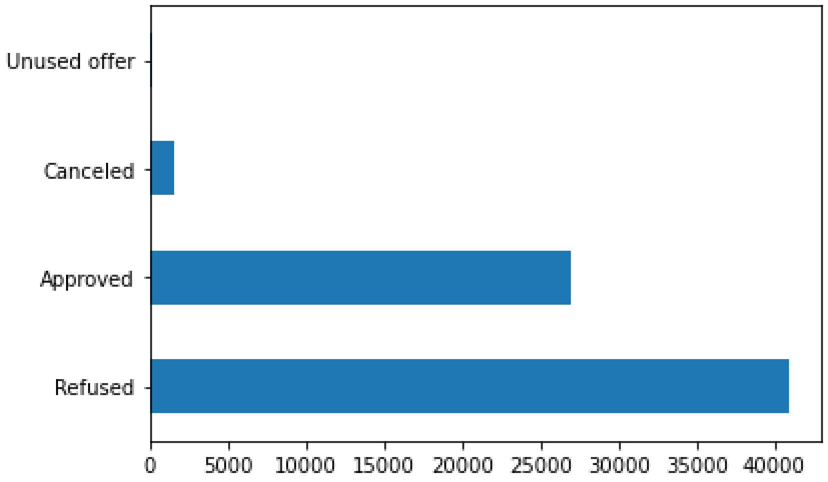
Identifying & Treating Missing Values

- **"NAME_CONTRACT_STATUS" has primarily 4 values and this is the variable on which we need to perform our analysis by looking at other variables along with it.**

```
prev_data.NAME_CONTRACT_STATUS.value_counts()
```

```
Refused         40858
Approved        26933
Canceled         1639
Unused offer      205
Name: NAME_CONTRACT_STATUS, dtype: int64
```

```
prev_data.NAME_CONTRACT_STATUS.value_counts().plot.barh()
plt.show()
```



Bivariate Analysis

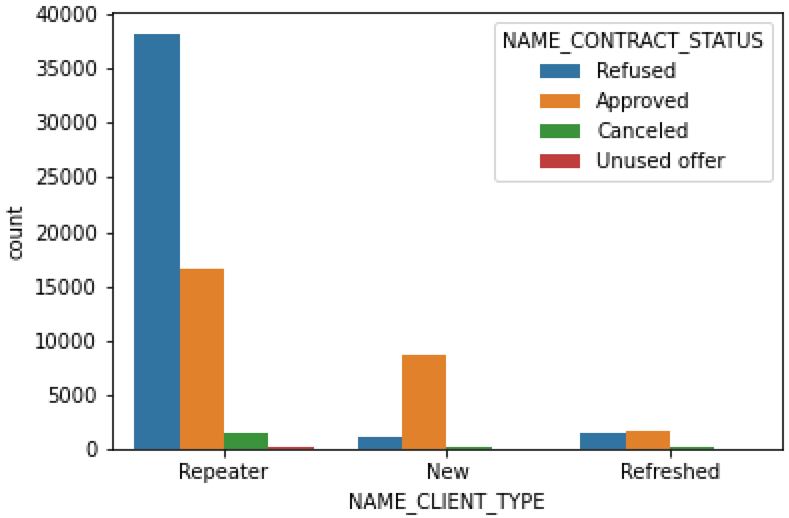
"NAME CLIENT TYPE" Vs "NAME CONTRACT STATUS"

Observation:

- **By seeing this chart, we can clearly identify that fresh loans are getting approved easily but in case of a repeater, rejection rate is high.**

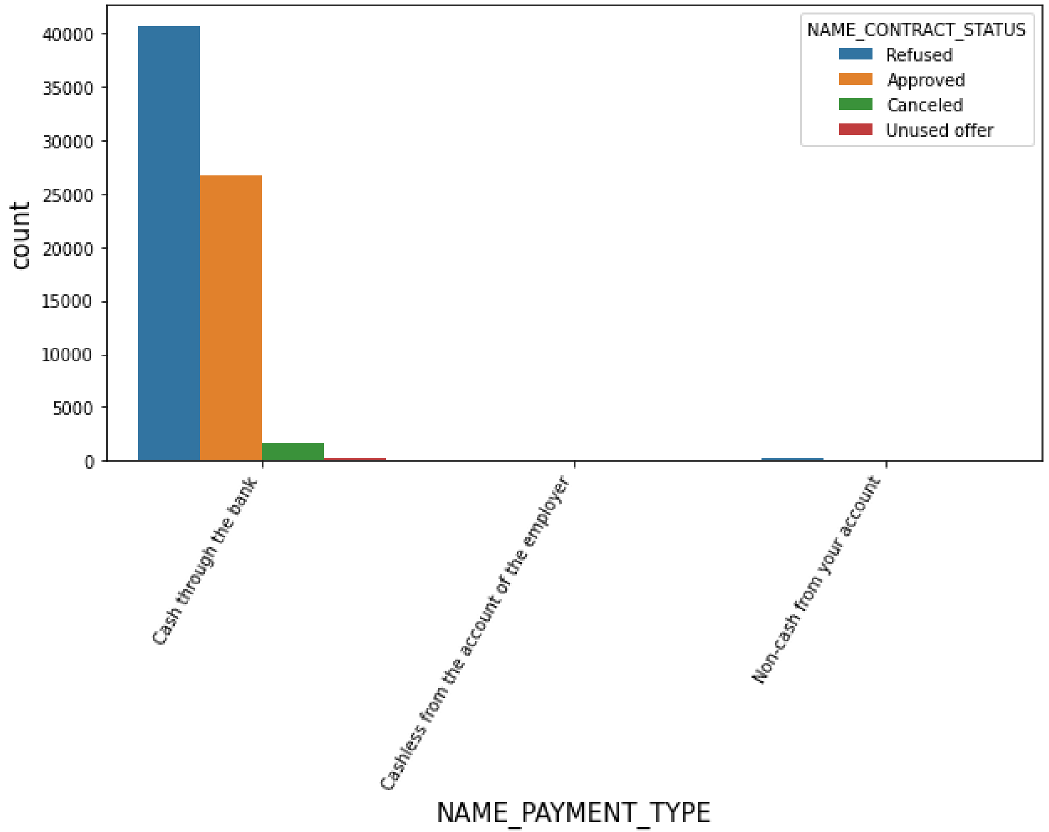
```
# Create a bi-variate analysis for different contract Status Vs client type
sns.countplot(x='NAME_CLIENT_TYPE', hue='NAME_CONTRACT_STATUS', data=prev_data)
```

<matplotlib.axes._subplots.AxesSubplot at 0x22f8179e4c0>



“NAME_PAYMENT_TYPE” Vs “NAME_CONTRACT_STATUS”

```
plt.figure(figsize=(10,5))
ax=sns.countplot(x='NAME_PAYMENT_TYPE', hue='NAME_CONTRACT_STATUS', data=prev_data)
ax.set_xticklabels(ax.get_xticklabels(), rotation=60, ha="right")
plt.show()
```

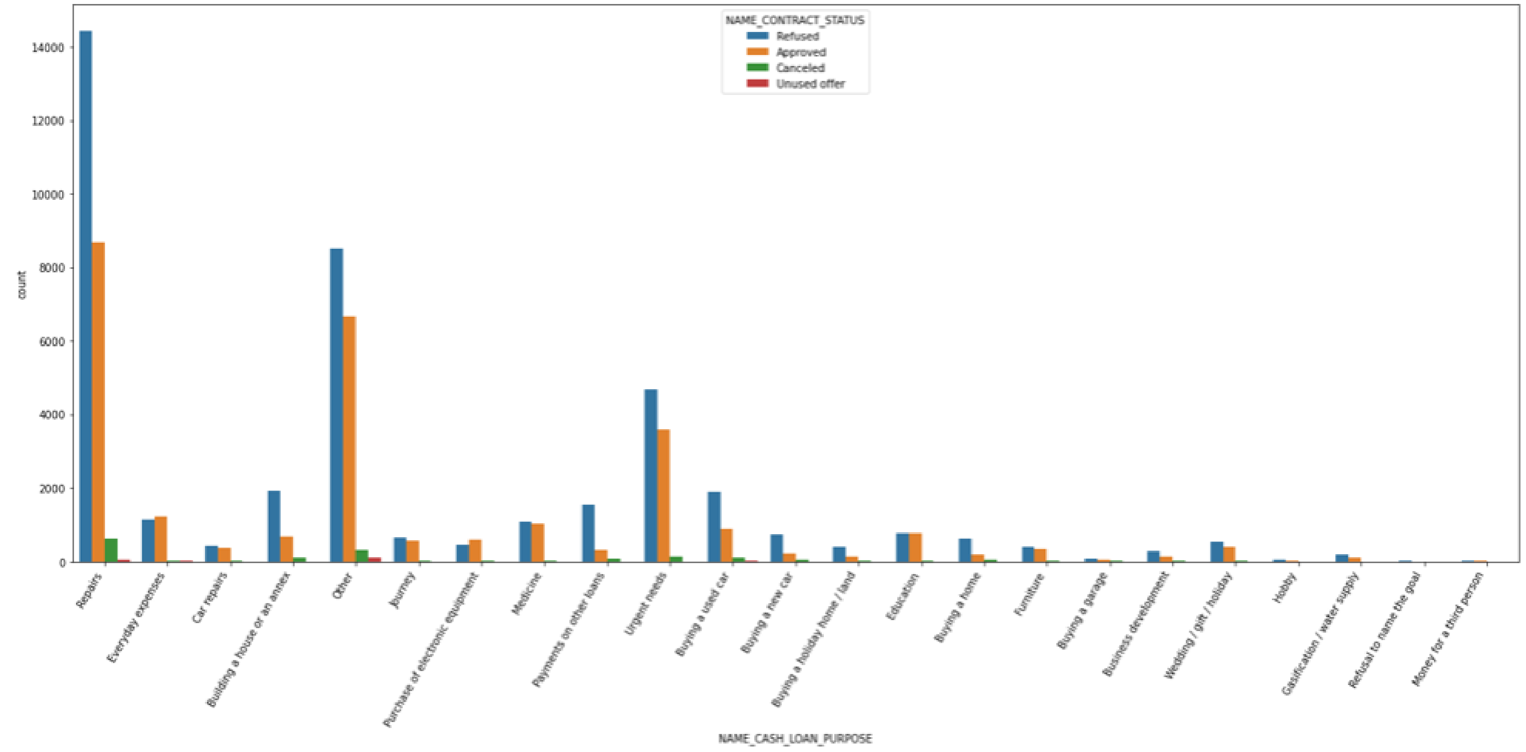


Observation:

- **We know the mostly people are getting loans via “Cash through the bank”, however when we try to related this variable with “NAME_CONTRACT_STATUS”, we see that rejection rate is higher than approval.**

“NAME_CASH_LOAN_PURPOSE” Vs “NAME_CONTRACT_STATUS”

```
plt.figure(figsize=(20,10))
ax=sns.countplot(x='NAME_CASH_LOAN_PURPOSE', hue='NAME_CONTRACT_STATUS', data=prev_data)
ax.set_xticklabels(ax.get_xticklabels(), rotation=60, ha="right")
plt.tight_layout()
plt.show()
```



Observation:

- Mostly people are applying for loan for “Repairs”, “Others” and “Urgent Needs” and also rejection rate in all these categories are higher than approval.
- Maximum loans are getting rejected which were taken for “Repairs”. The proportion of rejection is quite higher than approval.

Merging Current & Previous application Dataset

Merging “app_data” & “prev_data” to create a new one “cust_data”

- **We are merging both the dataset of current application (app_data) and previous application (prev_data) and making a new dataset (cust_data).**
- **There are few columns which are common in both datasets, hence to differentiate between them, we are using suffixes to be “_x”. This will add “_” at the end of column name in first dataset and in the other dataset, it will add “_x” at the end just in that column only.**

```
#cust_data = app_data
cust_data = app_data.merge(prev_data,how='inner',on='SK_ID_CURR',suffixes='_x')
```

```
#check the information about datatypes and null values
cust_data.info(verbose=True, null_counts=True)
```

```
 1   TARGET                      59413 non-null  int64
 2   NAME_CONTRACT_TYPE_         59413 non-null  object
 3   CODE_GENDER                 59413 non-null  object
 4   FLAG_OWN_CAR                59413 non-null  object
 5   FLAG_OWN_REALTY             59413 non-null  object
 6   CNT_CHILDREN                59413 non-null  int64
 7   AMT_INCOME_TOTAL            59413 non-null  float64
 8   AMT_CREDIT_                 59413 non-null  float64
 9   AMT_ANNUITY                 59406 non-null  float64
 10  AMT_GOODS_PRICE             59354 non-null  float64
 11  NAME_TYPE_SUITE             59218 non-null  object
 12  NAME_INCOME_TYPE            59413 non-null  object
 13  NAME_EDUCATION_TYPE         59413 non-null  object
 14  NAME_FAMILY_STATUS          59413 non-null  object
 15  NAME_HOUSING_TYPE           59413 non-null  object
 16  REGION_POPULATION_RELATIVE  59413 non-null  float64
 17  DAYS_BIRTH                  59413 non-null  int64
 18  DAYS_EMPLOYED               59413 non-null  int64
 19  DAYS_REGISTRATION           59413 non-null  float64
```

Merging “app_data” & “prev_data” to create a new one “cust_data”

- **Lets read top rows of this new dataset and check the detailed information about this dataset**

```
#check top 5 rows of this combined dataset
cust_data.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CRE
0	100034	0	Revolving loans	M	N	Y	0	90000.0	180
1	100035	0	Cash loans	F	N	Y	0	292500.0	665
2	100039	0	Cash loans	M	Y	N	1	360000.0	733
3	100046	0	Revolving loans	M	Y	Y	0	180000.0	540
4	100046	0	Revolving loans	M	Y	Y	0	180000.0	540

5 rows × 144 columns

```
cust_data.info(verbose=True, null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59413 entries, 0 to 59412
Data columns (total 144 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   SK_ID_CURR                     59413 non-null  int64
 1   TARGET                         59413 non-null  int64
 2   NAME_CONTRACT_TYPE_            59413 non-null  object
 3   CODE_GENDER                    59413 non-null  object
 4   FLAG_OWN_CAR                   59413 non-null  object
 5   FLAG_OWN_REALTY                59413 non-null  object
 6   CNT_CHILDREN                   59413 non-null  int64
 7   AMT_INCOME_TOTAL               59413 non-null  float64
 8   AMT_CREDIT_                    59413 non-null  float64
 9   AMT_ANNUITY                    59406 non-null  float64
 10  AMT_GOODS_PRICE                59354 non-null  float64
 11  NAME_TYPE_SUITE                59218 non-null  object
 12  NAME_INCOME_TYPE               59413 non-null  object
 13  NAME_EDUCATION_TYPE            59413 non-null  object
```

Bivariate Analysis

"NAME_CASH_LOAN_PURPOSE" Vs "TARGET"

- **Lets analyze "NAME_CASH_LOAN_PURPOSE" with "TARGET"**

```
Loan_purpose = pd.DataFrame(cust_data.groupby(['NAME_CASH_LOAN_PURPOSE'])['TARGET'].value_counts())
Loan_purpose
```

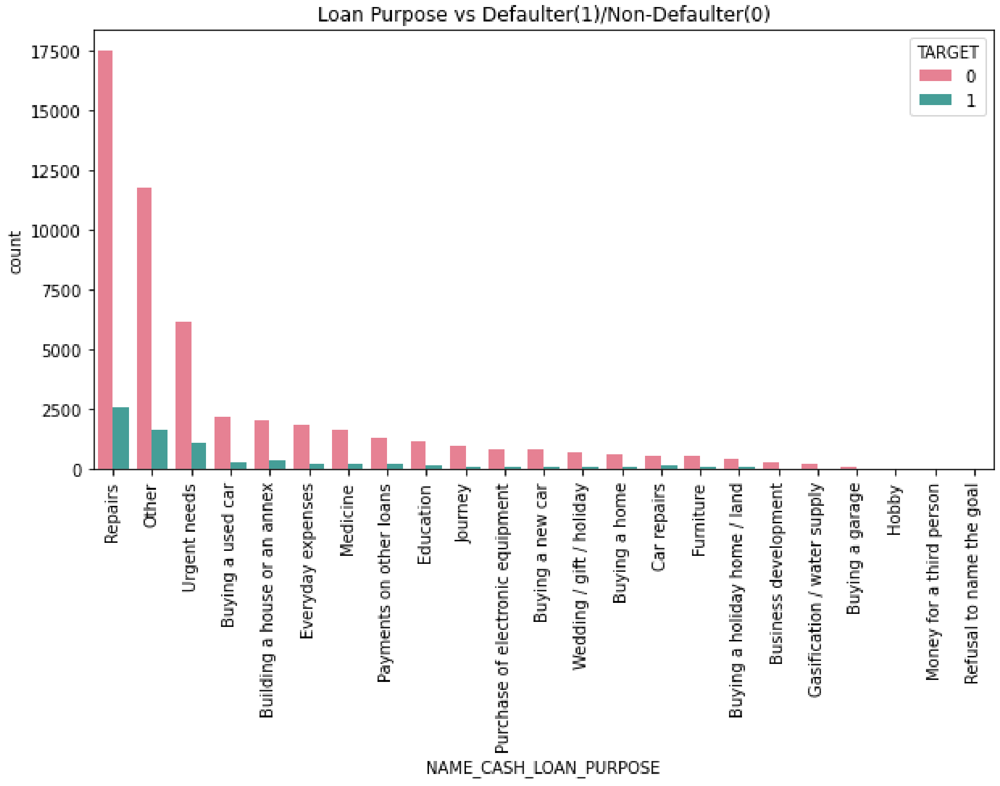
		TARGET
NAME_CASH_LOAN_PURPOSE	**TARGET**	
Building a house or an annex	**0**	2020
	1	324
Business development	**0**	313
	1	46
Buying a garage	**0**	109
	1	7
Buying a holiday home / land	**0**	408
	1	55
Buying a home	**0**	617
	1	84
Buying a new car	**0**	806
	1	80
Buying a used car	**0**	2151
	1	318
Car repairs	**0**	564
	1	127
Education	**0**	1194
	1	140
Everyday expenses	**0**	1836
	1	216
Furniture	**0**	575
	1	85
Gasification / water supply	**0**	206
	1	45
Hobby	**0**	36
	1	9

“NAME_CASH_LOAN_PURPOSE” Vs “TARGET”

Observation:

- **We see that people who are taking loans for “Repairs” are turning out to be a defaulter.**
- **We also analyzed that this is the category for which maximum loan requests are getting rejected as well. Seems our decision has been right in this direction.**

```
plt.figure(figsize = (10,5))
sns.countplot(data = cust_data, x= 'NAME_CASH_LOAN_PURPOSE',
              order=cust_data['NAME_CASH_LOAN_PURPOSE'].value_counts().index,hue = 'TARGET',palette='husl')
plt.xticks(rotation=90)
plt.title('Loan Purpose vs Defaulter(1)/Non-Defaulter(0)')
plt.show()
```

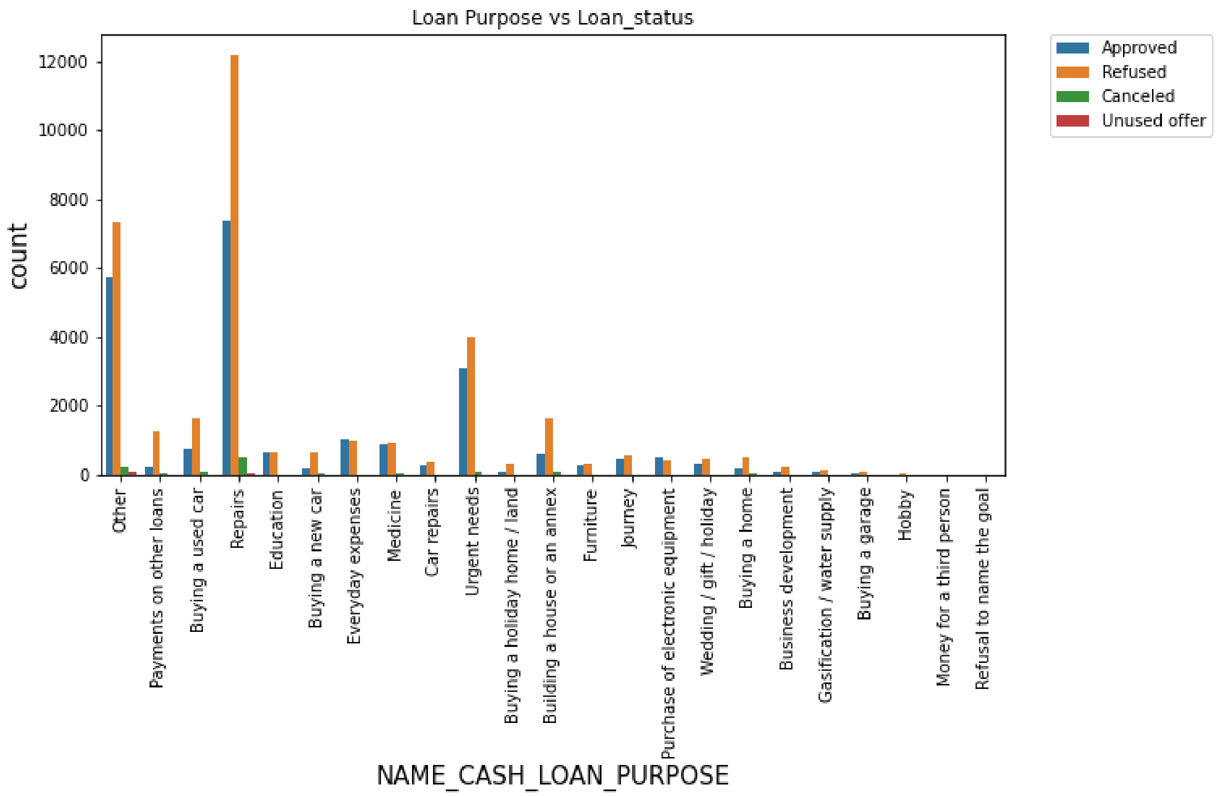


"NAME_CASH_LOAN_PURPOSE" Vs "NAME_CONTRACT_STATUS"

Observation:

- **Mostly people are applying for loan for "Repairs", "Others" and "Urgent Needs" and also rejection rate in all these categories are higher than approval.**
- **Maximum loans are getting rejected which were taken for "Repairs". The proportion of rejection is quite higher than approval.**
- **For Education, #of approved and rejected applications are same**
- **For "Everyday Expense" and "Purchase of electronic equipment", approvals are higher than rejection.**

```
# Purpose of loans vs Loan status
plt.figure(figsize = (10,5))
sns.countplot(data = cust_data, x= 'NAME_CASH_LOAN_PURPOSE',hue = 'NAME_CONTRACT_STATUS')
plt.xticks(rotation=90)
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.title('Loan Purpose vs Loan_status')
plt.show()
```



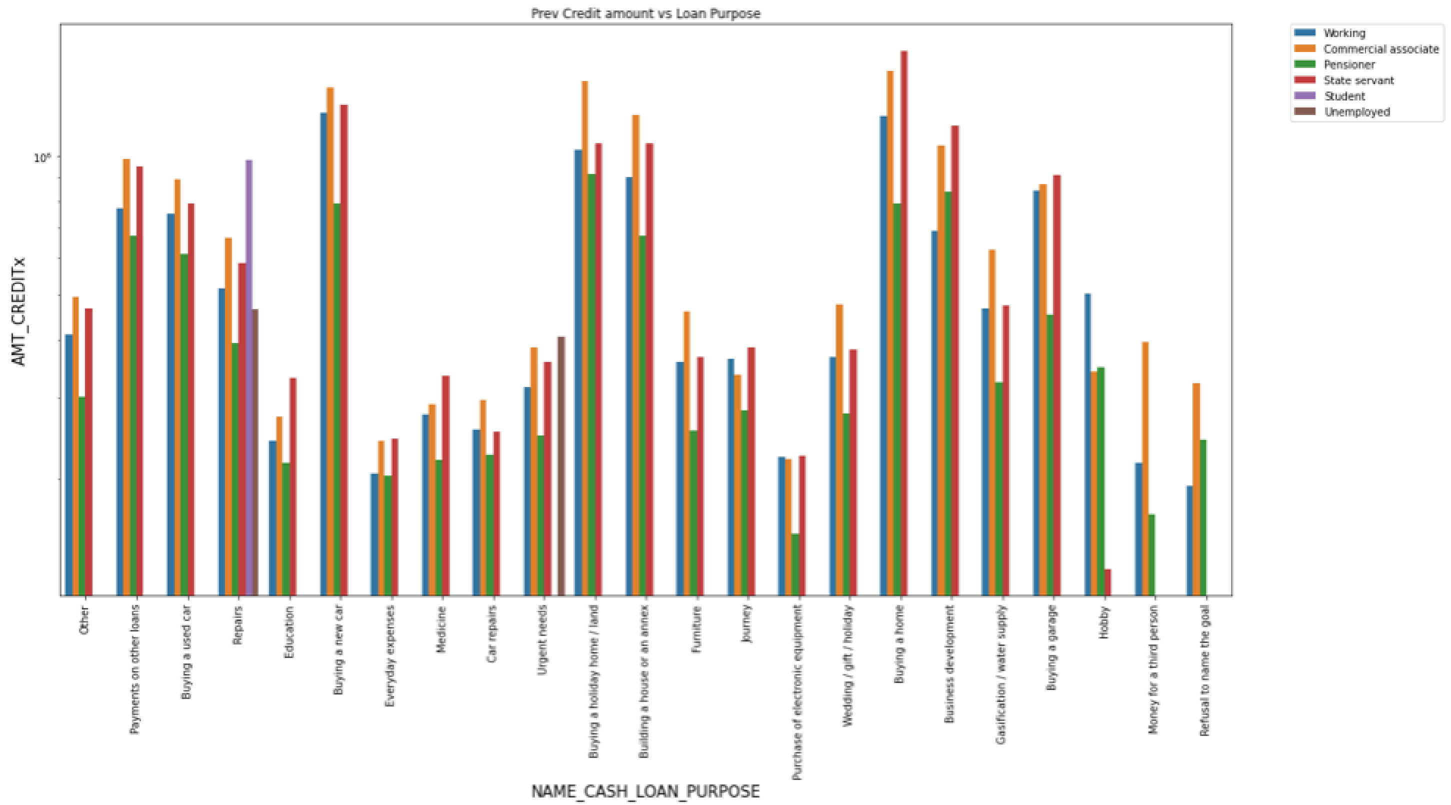
Multivariate Analysis

“NAME_CASH_LOAN_PURPOSE” Vs “NAME_INCOME_TYPE” Vs “AMT_CREDIT”

Observation:

- **The credit amount of Loan purposes like 'Buying a holiday 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.**

```
plt.figure(figsize=(20,10))
plt.rcParams["axes.labelsize"] = 15

plt.xticks(rotation=90)
plt.yscale('log')
sns.barplot(data =cust_data, x='NAME_CASH_LOAN_PURPOSE',hue='NAME_INCOME_TYPE',y='AMT_CREDITx',orient='v', ci=False)
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.title('Prev Credit amount vs Loan Purpose')
plt.show()
```



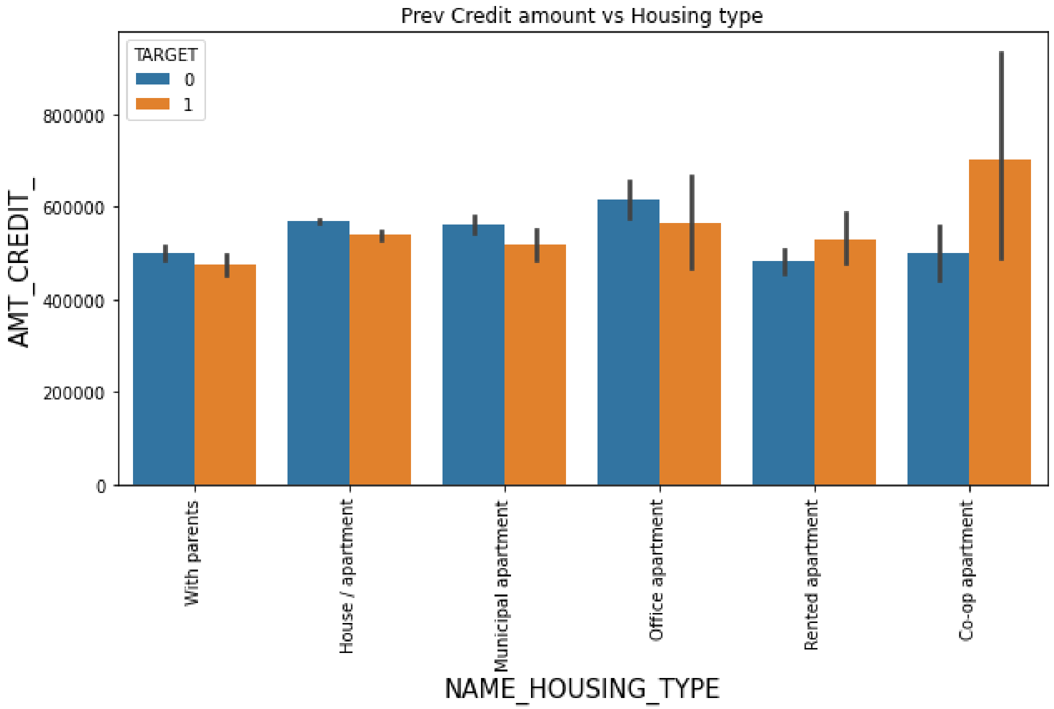
"AMT_CREDIT" Vs "TARGET" Vs "NAME_HOUSING_TYPE"

Observation:

- **Here for Housing type, office apartment is having higher credit of target 0 (Non-Defaulter) and co-op apartment is having higher credit of target 1 (Defaulter).**

- **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/Apartment or municipal apartment for successful payments**

```
plt.figure(figsize=(10,5))
plt.xticks(rotation=90)
sns.barplot(data =cust_data, y='AMT_CREDIT_',hue='TARGET',x='NAME_HOUSING_TYPE')
plt.title('Prev Credit amount vs Housing type')
plt.show()
```



WHAT IF WE DO NOT HANDLE MISSING VALUES?

Lets Observe

- There are many ways to handle missing values in a dataset while performing Exploratory Data Analysis. If there are significant number of NULL values, its advisable that we drop all those rows/columns and then start EDA. Sometime we also perform imputations to handle these missing/NULL values. For numerical columns, we can choose mean, median Or quantiles values. For categorical, we go for mode values

- However it is impossible to know ahead of time, if these columns will be helpful to derive any strong evidences, hence we will do a quick analysis to see, how inferences changes when we DO NOT handle missing values in current application dataset

- As we have already performed detailed EDA on the given datasets after handling missing values, hence we will only analyze correlation between different variables and see how this inference is different

Import Libraries & Read Data

Imports

In [1]:
```
# import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

Read The Data

In [2]:
```
#read application data
app_data = pd.read_csv('application_data.csv')
```

Check the loaded Data

In [3]:
```
#Check the loaded data
app_data.head()
```

Out[3]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_C
0	100002	1	Cash loans	M	N	Y	0	202500.0	40
1	100003	0	Cash loans	F	N	N	0	270000.0	129
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	13
3	100006	0	Cash loans	F	N	Y	0	135000.0	31
4	100007	0	Cash loans	M	N	Y	0	121500.0	51

5 rows × 122 columns

Correlations

Pearson's correlation coefficient is a statistical measure of the strength of a linear relationship between paired data. One of the effective way to try and understand the data is by looking for correlations between the features and the target. We can calculate the Pearson correlation coefficient between every variable and the target using the .corr dataframe method.

Furthermore:

- Positive values denote positive linear correlation
- Negative values denote negative linear correlation
- A value of 0 denotes no linear correlation
- The closer the value is to 1 or –1, the stronger the linear correlation.

The correlation coefficient gives us an idea of possible relationships within the data. Some general interpretations of the absolute value of the correlation coefficent are:

- .00-.19 "very weak"
- .20-.39 "weak"
- .40-.59 "moderate"
- .60-.79 "strong"
- .80-1.0 "very strong"

Correlations

Observation:

- Let's take a look at some of more significant correlations: The DAYS_BIRTH is the most positive correlation. (except for TARGET because the correlation of a variable with itself is always 1!) Looking at the documentation, DAYS_BIRTH is the age in days of the client at the time of the loan in negative days (for whatever reason!). The correlation is positive, but the value of this feature is actually negative, meaning that as the client gets older, they are less likely to default on their loan (i.e. the target == 0). That's a little confusing, so we will take the absolute value of the feature and then the correlation will be negative.

```python
correlations = app_data.corr()['TARGET'].sort_values()

# Display correlations
print('Most Positive Correlations:\n', correlations.tail(15))
print('\nMost Negative Correlations:\n', correlations.head(15))
```

```
Most Positive Correlations:
 DEF_60_CNT_SOCIAL_CIRCLE       0.031276
DEF_30_CNT_SOCIAL_CIRCLE       0.032248
LIVE_CITY_NOT_WORK_CITY        0.032518
OWN_CAR_AGE                    0.037612
DAYS_REGISTRATION              0.041975
FLAG_DOCUMENT_3                0.044346
REG_CITY_NOT_LIVE_CITY         0.044395
FLAG_EMP_PHONE                 0.045982
REG_CITY_NOT_WORK_CITY         0.050994
DAYS_ID_PUBLISH                0.051457
DAYS_LAST_PHONE_CHANGE         0.055218
REGION_RATING_CLIENT           0.058899
REGION_RATING_CLIENT_W_CITY    0.060893
DAYS_BIRTH                     0.078239
TARGET                         1.000000
Name: TARGET, dtype: float64

Most Negative Correlations:
 EXT_SOURCE_3                 -0.178919
EXT_SOURCE_2                 -0.160472
EXT_SOURCE_1                 -0.155317
DAYS_EMPLOYED                -0.044932
FLOORSMAX_AVG                -0.044003
FLOORSMAX_MEDI               -0.043768
FLOORSMAX_MODE               -0.043226
AMT_GOODS_PRICE              -0.039645
REGION_POPULATION_RELATIVE   -0.037227
ELEVATORS_AVG                -0.034199
ELEVATORS_MEDI               -0.033863
FLOORSMIN_AVG                -0.033614
FLOORSMIN_MEDI               -0.033394
LIVINGAREA_AVG               -0.032997
LIVINGAREA_MEDI              -0.032739
Name: TARGET, dtype: float64
```

Correlations

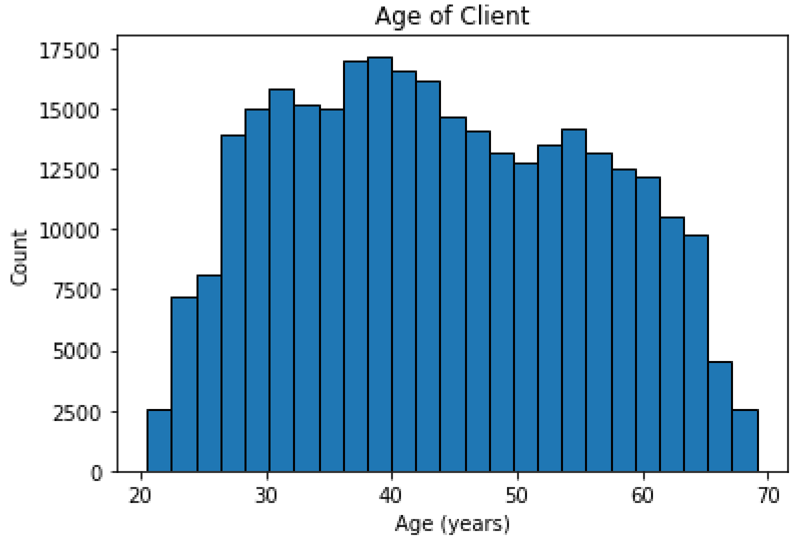
Observation:

- As the client gets older, there is a negative linear relationship with the target meaning that as clients get older, they tend to repay their loans on time more often.
- Let's start looking at this variable. First, we can make a histogram of the age. We will put the x axis in years to make the plot a little more understandable.
- By itself, the distribution of age does not tell us much other than that there are no outliers as all the ages are reasonable. To visualize the effect of the age on the target, we will next make a kernel density estimation plot (KDE) colored by the value of the target. A kernel density estimate plot shows the distribution of a single variable and can be thought of as a smoothed histogram

```
# Find the correlation of the positive days since birth and target
app_data['DAYS_BIRTH'] = abs(app_data['DAYS_BIRTH'])
app_data['DAYS_BIRTH'].corr(app_data['TARGET'])
```

```
-0.07823930830982712
```

```
plt.hist(app_data['DAYS_BIRTH'] / 365, edgecolor = 'k', bins = 25)
plt.title('Age of Client'); plt.xlabel('Age (years)'); plt.ylabel('Count');
```

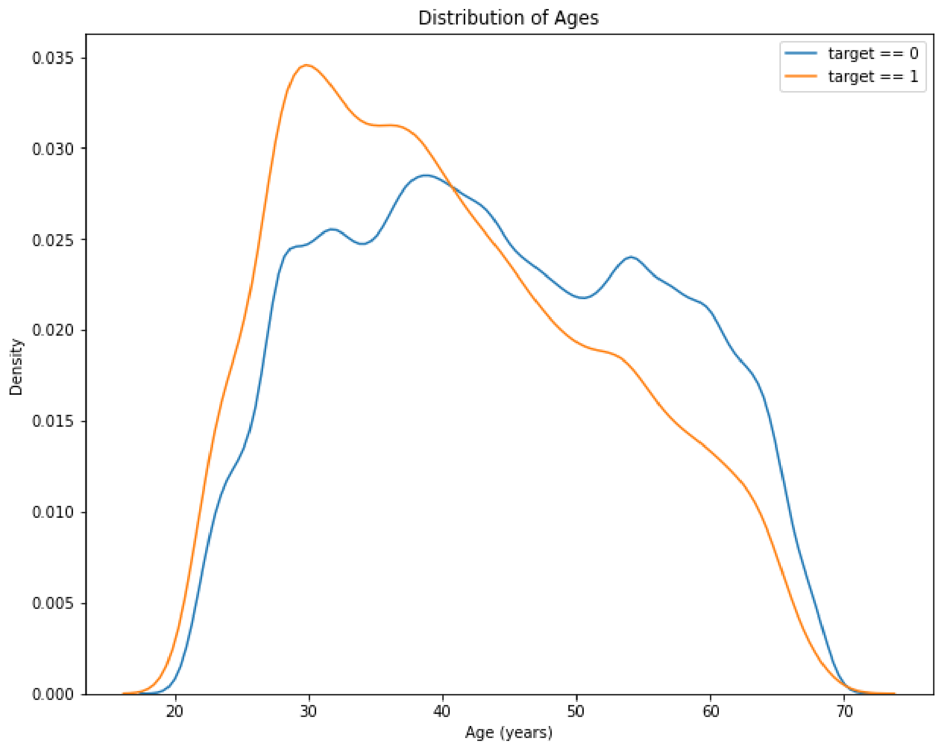


"DAYS_BIRTH" Vs "TARGET"

Observation:

- The target == 1 curve skews towards the younger end of the range.
- Let's look at this relationship in another way: average failure to repay loans by age bracket.
- To make this graph, first we cut the age category into bins of 5 years each. Then, for each bin, we calculate the average value of the target, which tells us the ratio of loans that were not repaid in each age category.

```
plt.figure(figsize = (10, 8))

# KDE plot of loans that were repaid on time
sns.kdeplot(app_data.loc[app_data['TARGET'] == 0, 'DAYS_BIRTH'] / 365, label = 'target == 0')

# KDE plot of loans which were not repaid on time
sns.kdeplot(app_data.loc[app_data['TARGET'] == 1, 'DAYS_BIRTH'] / 365, label = 'target == 1')

# Labeling of plot
plt.xlabel('Age (years)'); plt.ylabel('Density'); plt.title('Distribution of Ages');
```



Creating Bins for “DAYS_BIRTH”

- **Let's look at this relationship in another way: average failure to repay loans by age bracket.**
- **To make this graph, first we cut the age category into bins of 5 years each. Then, for each bin, we calculate the average value of the target, which tells us the ratio of loans that were not repaid in each age category.**

```
# Age information into a separate dataframe
age_data = app_data[['TARGET', 'DAYS_BIRTH']]
age_data['YEARS_BIRTH'] = age_data['DAYS_BIRTH'] / 365

# Bin the age data
age_data['YEARS_BINNED'] = pd.cut(age_data['YEARS_BIRTH'], bins = np.linspace(20, 70, num = 11))
age_data.head(10)
```

	TARGET	DAYS_BIRTH	YEARS_BIRTH	YEARS_BINNED
0	1	9461	25.920548	(25.0, 30.0]
1	0	16765	45.931507	(45.0, 50.0]
2	0	19046	52.180822	(50.0, 55.0]
3	0	19005	52.068493	(50.0, 55.0]
4	0	19932	54.608219	(50.0, 55.0]
5	0	16941	46.413699	(45.0, 50.0]
6	0	13778	37.747945	(35.0, 40.0]
7	0	18850	51.643836	(50.0, 55.0]
8	0	20099	55.065753	(55.0, 60.0]
9	0	14469	39.641096	(35.0, 40.0]

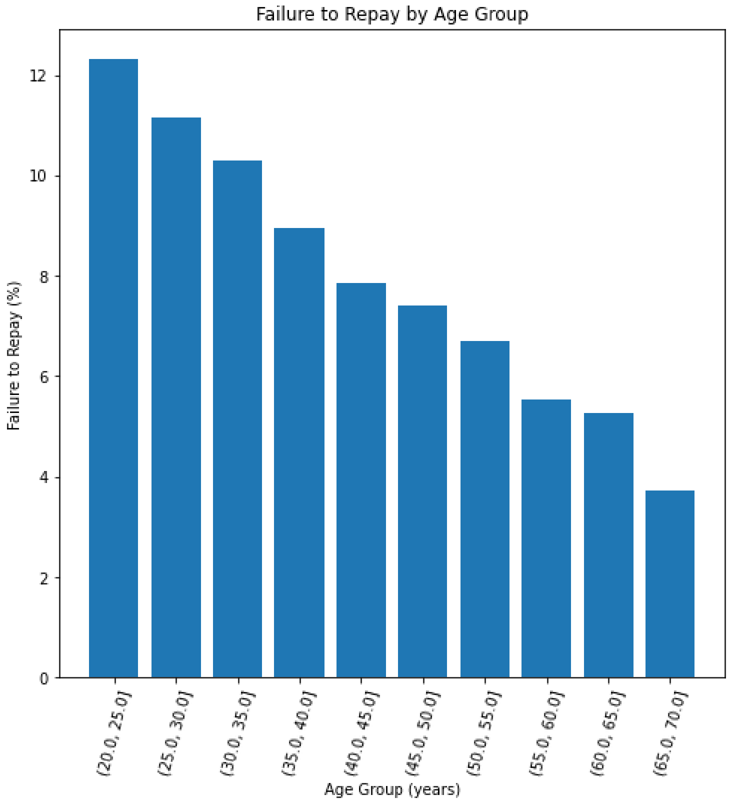
```
# Group by the bin and calculate averages
age_groups  = age_data.groupby('YEARS_BINNED').mean()
age_groups
```

	TARGET	DAYS_BIRTH	YEARS_BIRTH
YEARS_BINNED			
(20.0, 25.0]	0.123036	8532.795625	23.377522
(25.0, 30.0]	0.111436	10155.219250	27.822518
(30.0, 35.0]	0.102814	11854.848377	32.479037
(35.0, 40.0]	0.089414	13707.908253	37.555913
(40.0, 45.0]	0.078491	15497.661233	42.459346
(45.0, 50.0]	0.074171	17323.900441	47.462741
(50.0, 55.0]	0.066968	19196.494791	52.593136
(55.0, 60.0]	0.055314	20984.262742	57.491131
(60.0, 65.0]	0.052737	22780.547460	62.412459
(65.0, 70.0]	0.037270	24292.614340	66.555108

“AGE” Vs “TARGET”

- There is a clear trend: younger applicants are more likely to not repay the loan! The rate of failure to repay is above 10% for the youngest three age groups and below 5% for the oldest age group.
- This is information that could be directly used by the bank: because younger clients are less likely to repay the loan, maybe they should be provided with more guidance or financial planning tips. This does not mean the bank should discriminate against younger clients, but it would be smart to take precautionary measures to help younger clients pay on time

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

# Graph the age bins and the average of the target as a bar plot
plt.bar(age_groups.index.astype(str), 100 * age_groups['TARGET'])

# Plot labeling
plt.xticks(rotation = 75); plt.xlabel('Age Group (years)'); plt.ylabel('Failure to Repay (%)')
plt.title('Failure to Repay by Age Group');
```



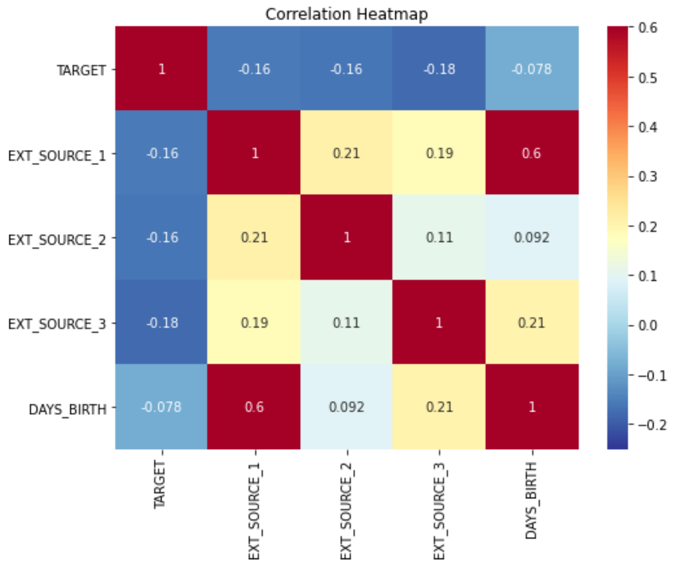
Correlations

- The 3 variables with the strongest negative correlations with the target are EXT_SOURCE_1, EXT_SOURCE_2, and EXT_SOURCE_3. According to the documentation, these features represent a "normalized score from external data source". At this moment, we are not sure what this exactly means, but it may be a cumulative sort of credit rating made using numerous sources of data.
- First, we can show the correlations of the EXT_SOURCE features with the target and with each other.
- All three EXT_SOURCE features have negative correlations with the target, indicating that as the value of the EXT_SOURCE increases, the client is more likely to repay the loan. We can also see that DAYS_BIRTH is positively correlated with EXT_SOURCE_1 indicating that maybe one of the factors in this score is the client age.
- Next we can look at the distribution of each of these features colored by the value of the target. This will let us visualize the effect of this variable on the target.

```
# Extract the EXT_SOURCE variables and show correlations
ext_data = app_data[['TARGET', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_BIRTH']]
ext_data_corrs = ext_data.corr()
ext_data_corrs
```

	TARGET	EXT_SOURCE_1	EXT_SOURCE_2	EXT_SOURCE_3	DAYS_BIRTH
TARGET	1.000000	-0.155317	-0.160472	-0.178919	-0.078239
EXT_SOURCE_1	-0.155317	1.000000	0.213982	0.186846	0.600610
EXT_SOURCE_2	-0.160472	0.213982	1.000000	0.109167	0.091996
EXT_SOURCE_3	-0.178919	0.186846	0.109167	1.000000	0.205478
DAYS_BIRTH	-0.078239	0.600610	0.091996	0.205478	1.000000

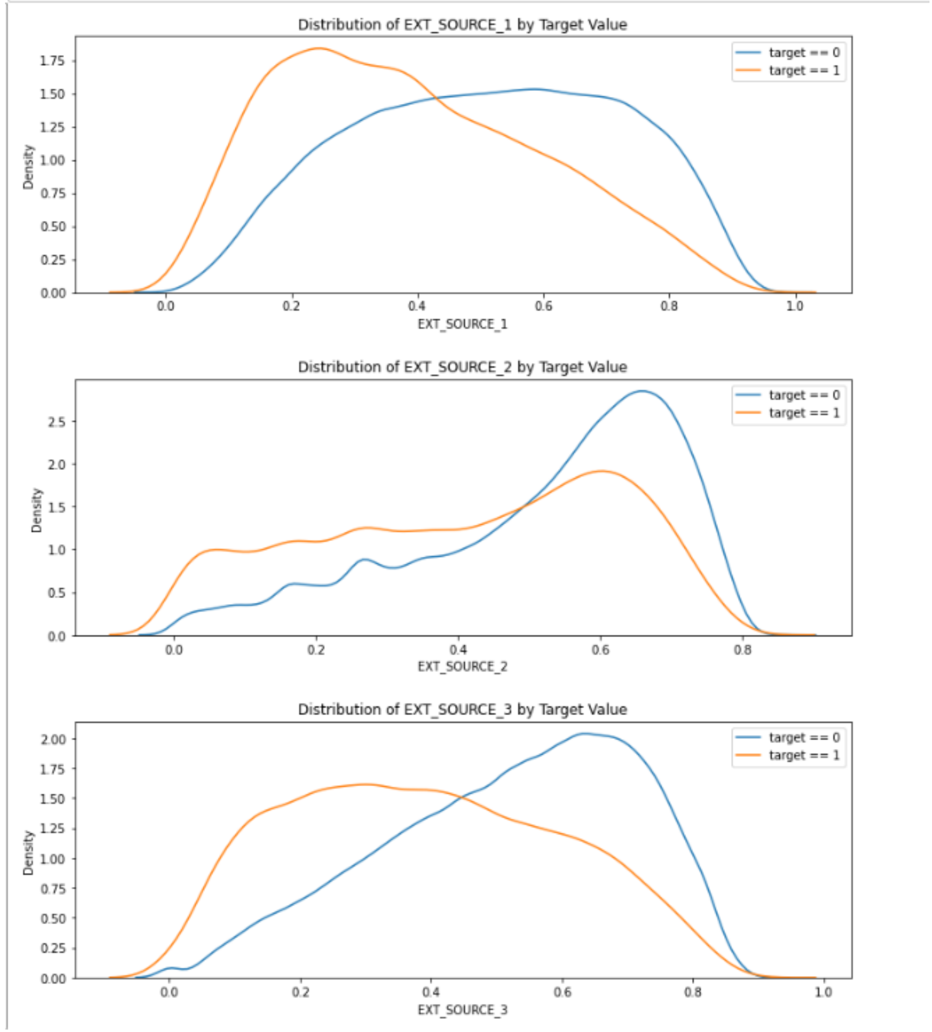
```
plt.figure(figsize = (8, 6))

# Heatmap of correlations
sns.heatmap(ext_data_corrs, cmap = plt.cm.RdYlBu_r, vmin = -0.25, annot = True, vmax = 0.6)
plt.title('Correlation Heatmap');
```



Distribution of “EXT_SOURCE” Vs “TARGET”

- Next we can look at the distribution of each of these features colored by the value of the target. This will let us visualize the effect of this variable on the target.
- EXT_SOURCE_3 displays the greatest difference between the values of the target. We can clearly see that this feature has some relationship to the likelihood of an applicant to repay a loan.

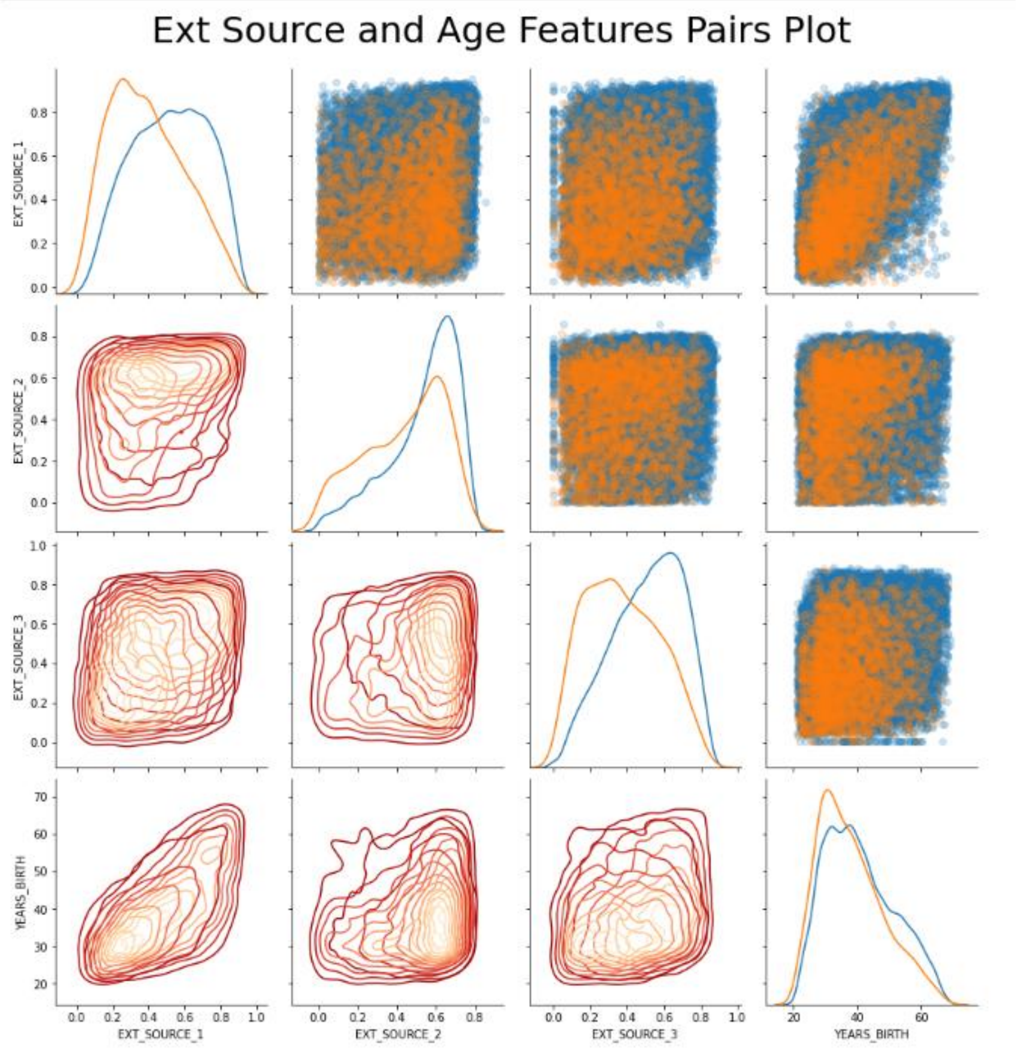
```python
plt.figure(figsize = (10, 12))

# iterate through the sources
for i, source in enumerate(['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']):

    # create a new subplot for each source
    plt.subplot(3, 1, i + 1)
    # plot repaid loans
    sns.kdeplot(app_data.loc[app_data['TARGET'] == 0, source], label = 'target == 0')
    # plot loans that were not repaid
    sns.kdeplot(app_data.loc[app_data['TARGET'] == 1, source], label = 'target == 1')

    # Label the plots
    plt.title('Distribution of %s by Target Value' % source)
    plt.xlabel('%s' % source); plt.ylabel('Density');

plt.tight_layout(h_pad = 2.5)
```



“YEARS_BIRTH” Vs “EXT_SOURCE” Vs “TARGET”

- In this plot, the red indicates loans that were not repaid and the blue are loans that are paid. We can see the different relationships within the data.

```
# Copy the data for plotting
plot_data = ext_data.drop(columns = ['DAYS_BIRTH']).copy()

# Add in the age of the client in years
plot_data['YEARS_BIRTH'] = age_data['YEARS_BIRTH']

# Drop na values and limit to first 100000 rows
plot_data = plot_data.dropna().loc[:100000, :]

# Function to calculate correlation coefficient between two columns
def corr_func(x, y, **kwargs):
    r = np.corrcoef(x, y)[0][1]
    ax = plt.gca()
    ax.annotate("r = {:.2f}".format(r),
                xy=(.2, .8), xycoords=ax.transAxes,
                size = 20)

# Create the pairgrid object
grid = sns.PairGrid(data = plot_data, size = 3, diag_sharey=False,
                    hue = 'TARGET',
                    vars = [x for x in list(plot_data.columns) if x != 'TARGET'])

# Upper is a scatter plot
grid.map_upper(plt.scatter, alpha = 0.2)

# Diagonal is a histogram
grid.map_diag(sns.kdeplot)

# Bottom is density plot
grid.map_lower(sns.kdeplot, cmap = plt.cm.OrRd_r);

plt.suptitle('Ext Source and Age Features Pairs Plot', size = 32, y = 1.05);
```



Conclusions

Conclusions – After Handling NULL Values

Bank should focus to give loans

- Bank should try to give loans to older people than younger people for successful payments.
- Banks should focus more on contract type 'Student' , 'pensioner' and 'Businessman' with housing type other than 'Co-op apartment' for successful payments.
- Get as much as clients from housing type 'With parents' as they are having least number of unsuccessful payments.

Bank should focus to avoid giving loans

- Bank should avoid approving loans for people with 1 family member, as they tend to default more.
- Banks should avoid giving loans on income type 'Working' and family status as 'married' as they are having most number of unsuccessful payments.
- Also with loan purpose 'Repair' is having higher number of unsuccessful payments.
- Married People with secondary/secondary special education, tend to default more, hence these people should be avoided to give loans

Conclusion – without handling NULL Values

- Older applicants are more likely to repay the loan on time. This does not mean the bank should discriminate against younger clients, but it would be smart to take precautionary measures to help younger clients pay on time

- Credit score obtained by multiple sources helps to identify if the applicant will be able to reply the loan on time or not. Higher credit score lead to strong capability of applicants to pay on time.