BANK LOAN CASE STUDY

Project Description:

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. Suppose you work for a consumer finance company which specialises in lending various types of loans to urban customers. You have to use EDA to analyse 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:

- If the applicant is likely to repay the loan, then not approving the loan results in a loss of business to the company.
- 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.

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:

1. **Approved:** The company has approved loan application
2. **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.
3. **Refused:** The company had rejected the loan (because the client does not meet their requirements etc.).
4. **Unused Offer:** Loan has been cancelled by the client but on different stages of the process.

Problem Statement: In this case study, you will use EDA to understand how consumer attributes and loan attributes influence the tendency of default. **Identification of such applicants using EDA** is the aim of this case study

It aims to **identify patterns** which indicate if a client has difficulty paying their instalments 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.

.

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.

To develop your understanding of the domain, you are advised to independently research a little about **risk analytics** – understanding the types of variables and their significance should be enough).

The dataset contains the following files:

1. **`application_data.csv`** contains all the information of the client at the time of application.
   The data is about wheather 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_descrption.csv` is data dictionary which describes the meaning of the** variables.

Approach:

Step 1: Loading the data, and removing unwanted columns

The data is read using pandas library and checked for correlation between each column. The columns that are least correlated are removed from the data.

Previous application:

- It is observed that, this dataset contains 1048575 rows and 37 columns. From the correlation heat map we find that [SELLERPLACE_AREA, NFLAG_INSURED_ON_APPROVAL, HOUR_APPR_PROCESS_START, NFLAG_LAST_APPL_IN_DAY] are the columns that are least correlated and therefore removed.

```
In [11]: prev_app.corr().style.background_gradient(cmap = 'coolwarm')
```

Out[11]:

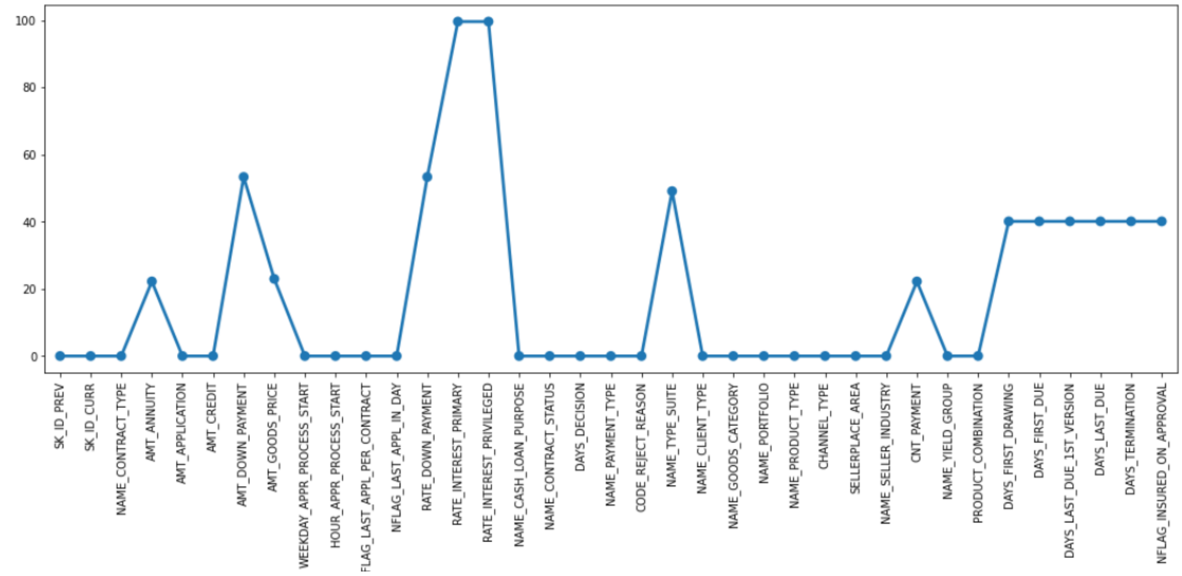
	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE
SK_ID_PREV	1.000000	0.000282	0.010379	0.002319	0.002710	-0.001844	0.013953
SK_ID_CURR	0.000282	1.000000	0.001330	0.002248	0.002347	-0.000854	0.002442
AMT_ANNUITY	0.010379	0.001330	1.000000	0.808841	0.816037	0.271141	0.820840
AMT_APPLICATION	0.002319	0.002248	0.808841	1.000000	0.975777	0.478458	0.999888
AMT_CREDIT	0.002710	0.002347	0.816037	0.975777	1.000000	0.297037	0.993112
AMT_DOWN_PAYMENT	-0.001844	-0.000854	0.271141	0.478458	0.297037	1.000000	0.478458
AMT_GOODS_PRICE	0.013953	0.002442	0.820840	0.999888	0.993112	0.478458	1.000000
HOUR_APPR_PROCESS_START	-0.002973	0.002559	-0.036402	-0.015063	-0.021828	0.016932	-0.046199
NFLAG_LAST_APPL_IN_DAY	-0.003193	0.000085	0.020574	0.004064	-0.025635	0.001763	-0.017536
RATE_DOWN_PAYMENT	-0.004247	0.000460	-0.102733	-0.072077	-0.187874	0.477225	-0.072077
RATE_INTEREST_PRIMARY	0.000597	0.033585	0.122145	0.090466	0.103305	0.017728	0.090466
RATE_INTEREST_PRIVILEGED	-0.027830	-0.007588	-0.202453	-0.202440	-0.204809	-0.129862	-0.202440
DAYS_DECISION	0.016653	-0.000299	0.277360	0.133063	0.133297	-0.024939	0.289011
SELLERPLACE_AREA	-0.000547	0.001095	-0.013897	-0.006703	-0.008462	0.003319	-0.014059
CNT_PAYMENT	0.013523	0.001306	0.393593	0.680530	0.674065	0.030964	0.672116
DAYS_FIRST_DRAWING	-0.001916	-0.001375	0.052182	0.073907	-0.037240	-0.002817	-0.024563
DAYS_FIRST_DUE	0.000397	0.001552	-0.054177	-0.049857	0.002247	-0.013457	-0.021526
DAYS_LAST_DUE_1ST_VERSION	0.002245	0.001604	-0.068995	-0.084641	0.043805	-0.000004	0.016382
DAYS_LAST_DUE	0.001469	0.001101	0.082231	0.172021	0.223995	-0.031953	0.210732
DAYS_TERMINATION	0.001628	0.000691	0.067730	0.147995	0.213626	-0.031032	0.208270
NFLAG_INSURED_ON_APPROVAL	0.003536	0.000648	0.282570	0.259439	0.264234	-0.043084	0.243824

- After removing the least correlated values, we find the missing values in the remaining columns and the columns having more than 40% missing values are removed.

```
SK_ID_PREV                     0.000000
SK_ID_CURR                     0.000000
NAME_CONTRACT_TYPE             0.000000
AMT_ANNUITY                   22.221491
AMT_APPLICATION                0.000000
AMT_CREDIT                     0.000000
AMT_DOWN_PAYMENT              53.348211
AMT_GOODS_PRICE               22.980235
WEEKDAY_APPR_PROCESS_START     0.000000
HOUR_APPR_PROCESS_START        0.000000
FLAG_LAST_APPL_PER_CONTRACT    0.000000
NFLAG_LAST_APPL_IN_DAY         0.000000
RATE_DOWN_PAYMENT             53.348211
RATE_INTEREST_PRIMARY         99.645137
RATE_INTEREST_PRIVILEGED      99.645137
NAME_CASH_LOAN_PURPOSE         0.000000
NAME_CONTRACT_STATUS           0.000000
DAYS_DECISION                  0.000000
NAME_PAYMENT_TYPE              0.000000
CODE_REJECT_REASON             0.000000
NAME_TYPE_SUITE               49.127626
NAME_CLIENT_TYPE               0.000000
NAME_GOODS_CATEGORY            0.000000
NAME_PORTFOLIO                 0.000000
NAME_PRODUCT_TYPE              0.000000
CHANNEL_TYPE                   0.000000
SELLERPLACE_AREA               0.000000
NAME_SELLER_INDUSTRY           0.000000
CNT_PAYMENT                   22.221205
NAME_YIELD_GROUP               0.000000
PRODUCT_COMBINATION            0.021362
DAYS_FIRST_DRAWING            40.121880
DAYS_FIRST_DUE                40.121880
DAYS_LAST_DUE_1ST_VERSION     40.121880
DAYS_LAST_DUE                 40.121880
DAYS_TERMINATION              40.121880
NFLAG_INSURED_ON_APPROVAL     40.121880
dtype: float64
```

- The graph with missing values is as follows:

```
plt.figure(figsize=(18,6))
x = prev_app.columns
y = prev_app.isnull().mean()*100
plt.xticks(rotation = 90)
sns.pointplot(x,y)
plt.show()
```



Application data:

- The shape of this data is (307511, 122) which indicates that the data has 307511 rows and 122 columns.
- Similar to the previous application data, the columns of this data are also correlated an d all the columns with least or no correlation are removed.

```
In [14]: app_data.corr().style.background_gradient(cmap = 'coolwarm')
```

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELA
SK_ID_CURR	1.000000	-0.002108	-0.001129	-0.001820	-0.000343	-0.000433	-0.000232	0.00
TARGET	-0.002108	1.000000	0.019187	-0.003982	-0.030369	-0.012817	-0.039645	-0.03
CNT_CHILDREN	-0.001129	0.019187	1.000000	0.012882	0.002145	0.021374	-0.001827	-0.02
AMT_INCOME_TOTAL	-0.001820	-0.003982	0.012882	1.000000	0.156870	0.191657	0.159610	0.07
AMT_CREDIT	-0.000343	-0.030369	0.002145	0.156870	1.000000	0.770138	0.986968	0.09
AMT_ANNUITY	-0.000433	-0.012817	0.021374	0.191657	0.770138	1.000000	0.775109	0.11
AMT_GOODS_PRICE	-0.000232	-0.039645	-0.001827	0.159610	0.986968	0.775109	1.000000	0.10
ION_POPULATION_RELATIVE	0.000849	-0.037227	-0.025573	0.074796	0.099738	0.118429	0.103520	1.00
DAYS_BIRTH	-0.001500	0.078239	0.330938	0.027261	-0.055436	0.009445	-0.053442	-0.02
DAYS_EMPLOYED	0.001366	-0.044932	-0.239818	-0.064223	-0.066838	-0.104332	-0.064842	-0.00
DAYS_REGISTRATION	-0.000973	0.041975	0.183395	0.027805	0.009621	0.038514	0.011565	-0.05

```
In [15]: # OWN_CAR_AGE, FLAG_MOBIL, FLAG_WORK_PHONE, FLAG_CONT_MOBILE, FLAG_PHONE, FLAG_EMAIL, HOUR_APPR_PROCESS_START, \
         # EXT_SOURCE_2, EXT_SOURCE_3, NONLIVINGAPARTMENTS_AVG, NONLIVINGAREA_AVG, FLAG_DOCUMENT_2, FLAG_DOCUMENT_4, \
         # FLAG_DOCUMENT_5, 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_MON,
         # AMT_REQ_CREDIT_BUREAU_QRT, AMT_REQ_CREDIT_BUREAU_YEAR
```

- The above mentioned columns are dropped from the app_data, the variable denoted for application data.

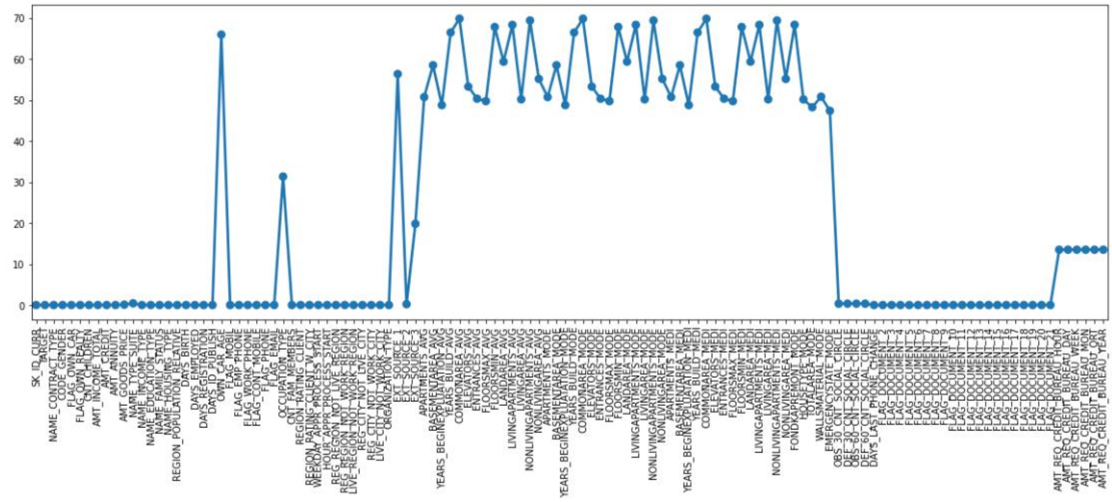
```
In [16]: unwanted_cols = ['OWN_CAR_AGE', 'FLAG_MOBIL', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE', 'FLAG_EMAIL', 'HOUR_APPR_PROCE
                          'EXT_SOURCE_2', 'EXT_SOURCE_3', 'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAREA_AVG', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMEN
         'FLAG_DOCUMENT_5', '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_MON', \
         'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR']

         app_data = app_data.drop(unwanted_cols, axis = 1)
```

- The missing values are then calculated and displayed. All the columns that has more than 50% missing values are removed from the dataset.

```
In [9]: app_data.isnull().mean()*100
```

```
Out[9]: SK_ID_CURR                     0.000000
        TARGET                         0.000000
        NAME_CONTRACT_TYPE             0.000000
        CODE_GENDER                    0.000000
        FLAG_OWN_CAR                   0.000000
                                         ...
        AMT_REQ_CREDIT_BUREAU_DAY     13.501631
        AMT_REQ_CREDIT_BUREAU_WEEK    13.501631
        AMT_REQ_CREDIT_BUREAU_MON     13.501631
        AMT_REQ_CREDIT_BUREAU_QRT     13.501631
        AMT_REQ_CREDIT_BUREAU_YEAR    13.501631
        Length: 122, dtype: float64
```

```
In [8]: plt.figure(figsize = (20,6))
        x = app_data.columns
        y = app_data.isnull().mean()*100
        plt.xticks(rotation = 90)
        sns.pointplot(x,y)
        plt.show()
```



__Step 2: Imputing the missing data with appropriate method.__

- After removing the unwanted columns from both the data sets, the shape of the data is as follows:

```
In [17]: print("Prev App", prev_app.shape)
         print("App Data", app_data.shape)

         Prev App (1048575, 33)
         App Data (307511, 90)
```

- Previous Application: The days columns are containing negative data are changed to positive data as days cannot be negative. The abs() method is used for the same. Also, the missing data is filled with median imputation.
- The categorical data is filled with “Unknown” as the value for ‘NAME_TYPE_SUITE’ and as “Cash” (mode imputation) for ‘PRODUCT_COMBINATION’ .
- Application data: The same steps are followed for application data as well.
  The numeric data and the categorical data are separated into different data frames.
  The days columns are changed to absolute values as days cannot be negative values.
  The missing values are imputed using IterativeImputer() from sklearn.

In [89]:
```
from sklearn.experimental import enable_iterative_imputer
```

In [90]:
```
from sklearn.impute import IterativeImputer
```

In [91]:
```
numeric_app_data.iloc[:,:] = IterativeImputer().fit_transform(numeric_app_data)
numeric_app_data.sample(5)
```

Out[91]:

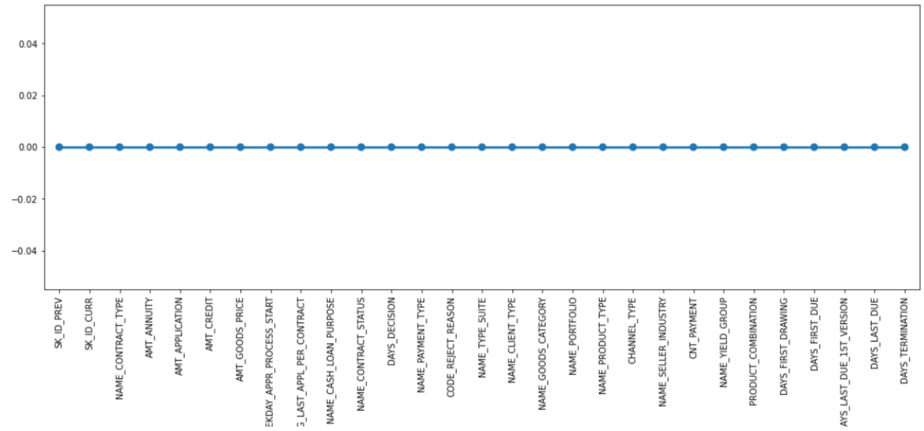
	TARGET	AMT_ANNUITY	AMT_GOODS_PRICE	CNT_FAM_MEMBERS	YEARS_BEGINEXPLUATATION_AVG	FLOORSMAX_AVG	YEARS_BEGINEXPLUATATI
1131	0.0	14508.0	225000.0	2.0	0.977722	0.225997	
68490	1.0	17019.0	225000.0	4.0	0.977773	0.225145	
22735	0.0	32895.0	1125000.0	2.0	0.985600	0.458300	
142858	1.0	33543.0	904500.0	3.0	0.978100	0.041700	
256727	0.0	67891.5	679500.0	2.0	0.978600	0.166700	

- The categorical data missing values are filled with “Unknown” values.

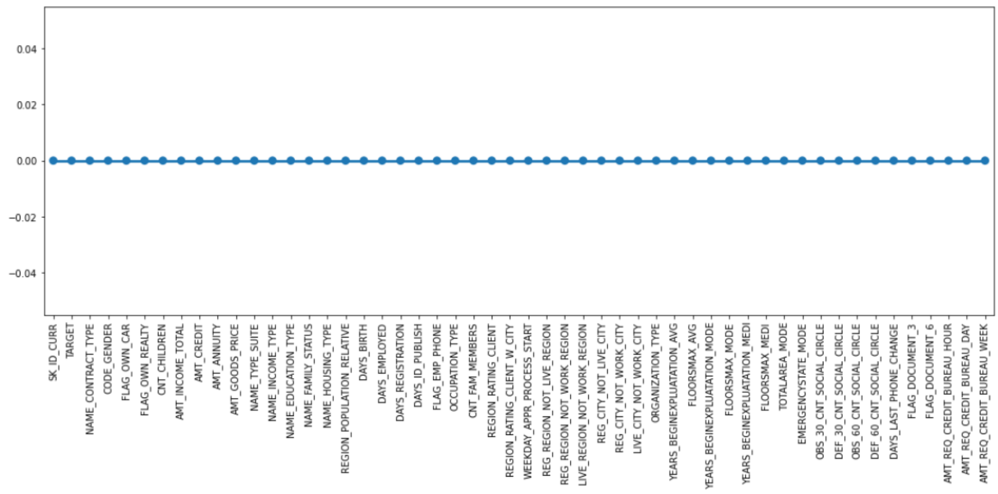
Step 3*: Checking for any remaining missing values in both the data sets.*

  - After the imputation of missing values with suitable methods the data sets are checked for any remaining missing values and a poinplot is plotted for both the datasets. The result is as follows:

In [65]:
```
plt.figure(figsize=(18,6))
x = prev_app.columns
y = prev_app.isnull().mean()*100
plt.xticks(rotation = 90)
sns.pointplot(x,y)
plt.show()
```



:
```
plt.figure(figsize=(18,6))
x = app_data.columns
y = app_data.isnull().mean()*100
plt.xticks(rotation = 90)
sns.pointplot(x,y)
plt.show()
```



Step 4: Data Set ready for Analysis

- After removing unwanted columns and imputing the missing values, the data sets are now ready for analysis.

Step 5: The links for working files:

1. https://drive.google.com/file/d/1Lu_6iq0sLDruDFpBrjXkYHzBbInjTNCJ/view?usp=sharing
2. https://drive.google.com/file/d/1x48y9IqxwNmlzTbfnwW7UFyMOsgSfFNs/view?usp=sharing
3. https://drive.google.com/file/d/1x48y9IqxwNmlzTbfnwW7UFyMOsgSfFNs/view?usp=sharing

Tech-Stack Used:

- **Jupyter Notebook :** Exploratory data analysis is done using jupyter note book
- **Microsoft Excel:** Excel is used to display csv files
- **Google Search Engine:** To clarify doubts and do research
- **Youtube:** For training classes

Insights:

1. **Problem Statement:** To identify patterns so that the person who can repay the loan is not rejected of the loan amount. The idea is to find patterns that are directly proportional to defaulting the repayment of loan, so that such persons are denied the loan or are charged with high rate of interest.
2. **Identify Missing columns and treat them correctly:**
   After removing the unwanted columns from both the data sets, the shape of the data is as follows:

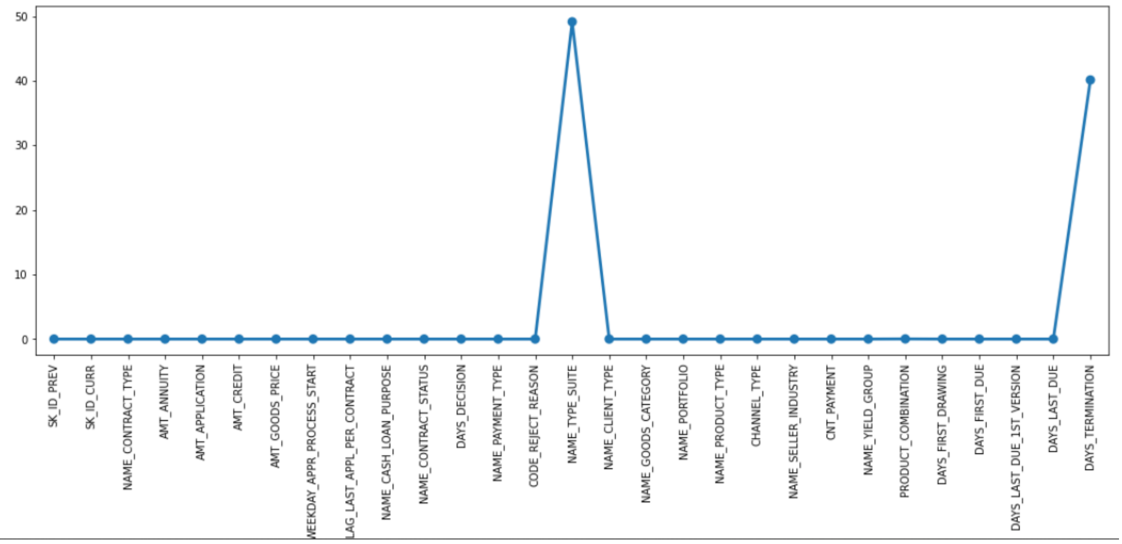
```
In [17]: print("Prev App", prev_app.shape)
         print("App Data", app_data.shape)

         Prev App (1048575, 33)
         App Data (307511, 90)
```

Previous Application: The days columns are containing negative data are changed to positive data as days cannot be negative. The abs() method is used for the same. Also, the missing data is filled with median imputation.

```
In [50]: prev_app['DAYS_LAST_DUE'] = abs(prev_app['DAYS_LAST_DUE'])

In [51]: prev_app['DAYS_LAST_DUE'].fillna(prev_app['DAYS_LAST_DUE'].median(), inplace = True)

In [52]: (prev_app['CNT_PAYMENT'] < 0).values.any()
Out[52]: False

prev_app['DAYS_FIRST_DRAWING'].fillna(prev_app['DAYS_FIRST_DRAWING'].median(), inplace = True)
```

The categorical data is filled with “Unknown” as the value for ‘NAME_TYPE_SUITE’ and as “Cash” (mode imputation) for ‘PRODUCT_COMBINATION’ . The code snippets are as follows:

In [54]:
```python
plt.figure(figsize=(18,6))
x = prev_app.columns
y = prev_app.isnull().mean()*100
plt.xticks(rotation = 90)
sns.pointplot(x,y)
plt.show()
```



In [55]:
```python
prev_app['NAME_TYPE_SUITE'].value_counts()
```

Out[55]:
```
Unaccompanied      318730
Family             134396
Spouse, partner     42160
Children            19957
Other_B             11084
Other_A              5707
Group of people      1401
Name: NAME_TYPE_SUITE, dtype: int64
```

```python
prev_app['NAME_TYPE_SUITE'].fillna("Unknown", inplace = True)
```

In [62]:
```python
prev_app['PRODUCT_COMBINATION'].value_counts()
```

Out[62]:
```
Cash                              178352
POS household with interest       166869
POS mobile with interest          139176
Cash X-Sell: middle                89806
Cash X-Sell: low                   80873
Card Street                        70951
POS industry with interest         62492
POS household without interest     52747
Card X-Sell                        50490
Cash Street: high                  37235
Cash X-Sell: high                  36813
Cash Street: middle                21616
Cash Street: low                   21166
POS mobile without interest        15181
POS other with interest            15072
POS industry without interest       7856
POS others without interest         1656
Name: PRODUCT_COMBINATION, dtype: int64
```

In [63]:
```python
prev_app['PRODUCT_COMBINATION'].mode()
```

Out[63]:
```
0    Cash
dtype: object
```

In [64]:
```python
prev_app['PRODUCT_COMBINATION'].fillna('Cash', inplace = True)
```

Application data: The same steps are followed for application data as well.

```
In [72]: # Getting all the numeric values to a new_df
numeric_app_data = app_data.select_dtypes(include = ['float', 'int64'])
numeric_app_data.sample(5)
```

Out[72]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIVE	D
199734	331550	0	0	225000.0	542133.0	22918.5	468000.0	0.025164	
188403	318429	0	1	540000.0	490495.5	47781.0	454500.0	0.010006	
235319	372573	0	0	315000.0	1226511.0	35860.5	1071000.0	0.006629	
24007	127916	0	1	135000.0	592560.0	32274.0	450000.0	0.035792	
282519	427251	0	0	83250.0	1024290.0	30078.0	855000.0	0.020713	

```
In [73]: cols = [col for col in numeric_app_data.columns if (numeric_app_data[col] < 0).any()]
cols
```

```
Out[73]: ['DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'DAYS_LAST_PHONE_CHANGE']
```

```
In [74]: for col in cols:
    app_data[col] = abs(app_data[col])
    numeric_app_data[col] = abs(numeric_app_data[col])
app_data.sample(5)
```

The numeric data and the categorical data are separated into different data frames.
The days columns are changed to absolute values as days cannot be negative values.
The missing values are imputed using IterativeImputer() from sklearn.

```
In [89]: from sklearn.experimental import enable_iterative_imputer
```

```
In [90]: from sklearn.impute import IterativeImputer
```

```
In [91]: numeric_app_data.iloc[:,:] = IterativeImputer().fit_transform(numeric_app_data)
numeric_app_data.sample(5)
```

Out[91]:

	TARGET	AMT_ANNUITY	AMT_GOODS_PRICE	CNT_FAM_MEMBERS	YEARS_BEGINEXPLUATATION_AVG	FLOORSMAX_AVG	YEARS_BEGINEXPLUATATI
1131	0.0	14508.0	225000.0	2.0	0.977722	0.225997	
68490	1.0	17019.0	225000.0	4.0	0.977773	0.225145	
22735	0.0	32895.0	1125000.0	2.0	0.985600	0.458300	
142858	1.0	33543.0	904500.0	3.0	0.978100	0.041700	
256727	0.0	67891.5	679500.0	2.0	0.978600	0.166700	

The categorical data missing values are filled with “Unknown” values.

```
In [95]: app_data['NAME_TYPE_SUITE'].value_counts()

Out[95]: Unaccompanied      248526
         Family              40149
         Spouse, partner     11370
         Children             3267
         Other_B              1770
         Other_A               866
         Group of people       271
         Name: NAME_TYPE_SUITE, dtype: int64
```

```
In [96]: app_data['NAME_TYPE_SUITE'].fillna("Unknown", inplace = True)
```

```
In [97]: app_data['EMERGENCYSTATE_MODE'].value_counts()

Out[97]: No     159428
         Yes      2328
         Name: EMERGENCYSTATE_MODE, dtype: int64
```

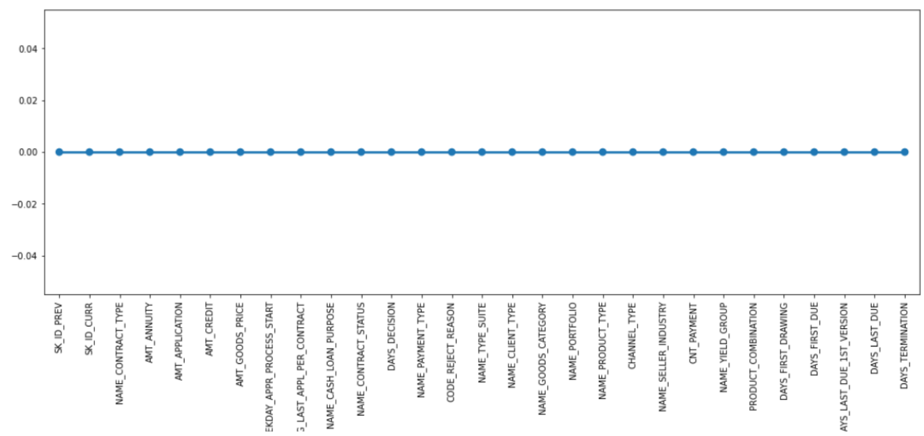
```
In [98]: app_data['EMERGENCYSTATE_MODE'].fillna('Unknown', inplace = True)
```

```
In [99]: app_data['OCCUPATION_TYPE'].value_counts()

Out[99]: Laborers                 55186
         Sales staff              32102
         Core staff               27570
         Managers                 21371
         Drivers                  18603
         High skill tech staff    11380
         Accountants               9813
         Medicine staff            8537
         Security staff            6721
         Cooking staff             5946
         Cleaning staff            4653
         Private service staff     2652
         Low-skill Laborers        2093
         Waiters/barmen staff      1348
         Secretaries               1305
         Realty agents              751
```

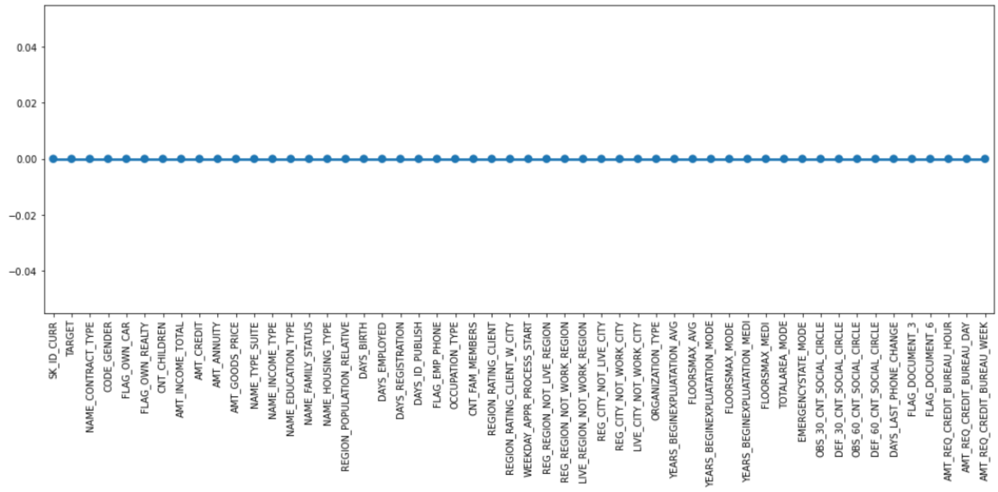
Checking for any remaining missing values in both the data sets.

a. After the imputation of missing values with suitable methods the data sets are checked for any remaining missing values and a pointplot is plotted for both the datasets. The result is as follows:

```
In [65]: plt.figure(figsize=(18,6))
         x = prev_app.columns
         y = prev_app.isnull().mean()*100
         plt.xticks(rotation = 90)
         sns.pointplot(x,y)
         plt.show()
```



```
plt.figure(figsize=(18,6))
x = app_data.columns
y = app_data.isnull().mean()*100
plt.xticks(rotation = 90)
sns.pointplot(x,y)
plt.show()
```



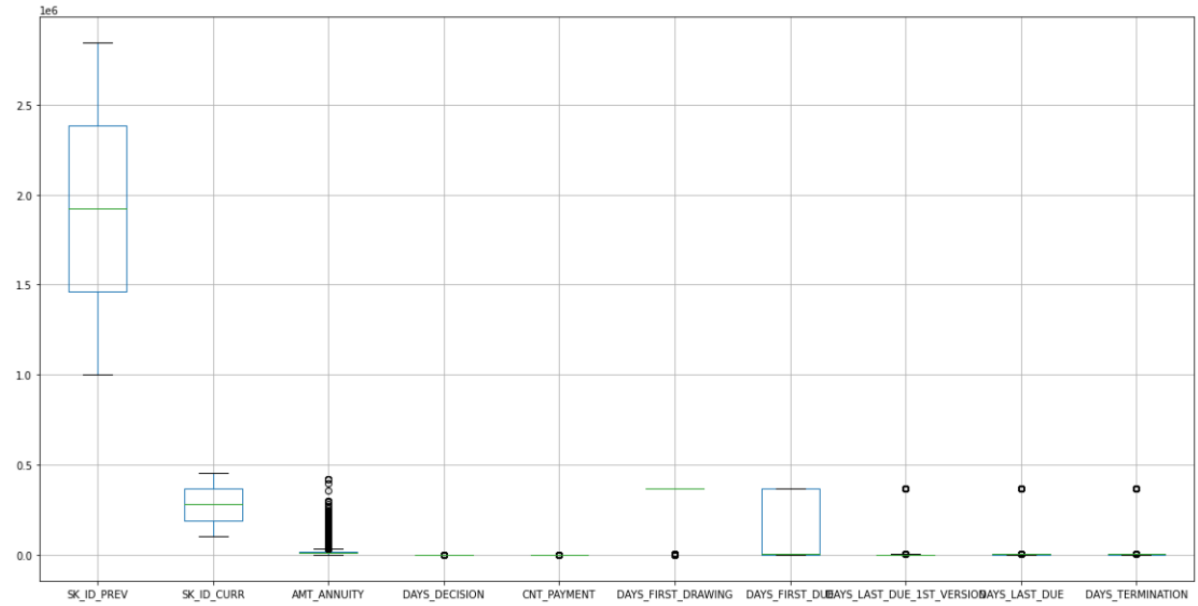
3. Identify the outliers in the dataset

Outliers are those data values which lie well beyond a certain limit. Extreme outliers have a great impact on the mean value of that column.

Boxplots are used to detect the outliers

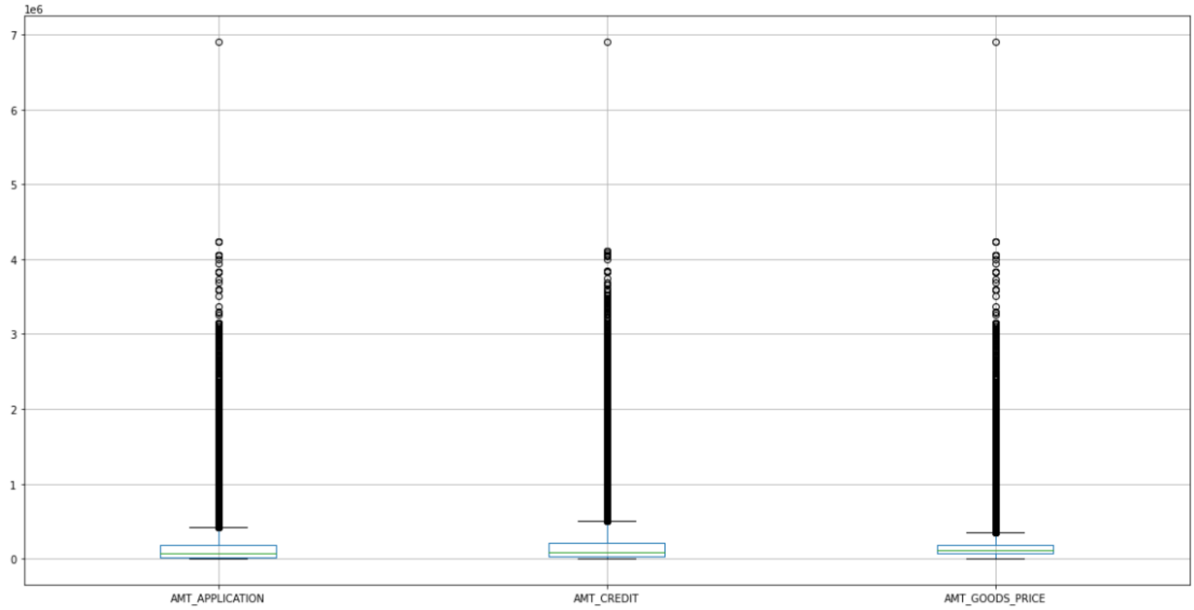
a. *Previous application data*: The columns are divided into two categories for better plotting of the graphs. In the **first figure**, we observe that the columns DAYS_LAST_DUE_1ST_VERSION, DAYS_LAST_DUE, DAYS_TERMINATION are the outliers which correspond to 0.4million which equates to 1,095.89 years and is clearly an outlier.

```
cols = [col for col in prev_app.columns if col not in ['AMT_APPLICATION', 'AMT_CREDIT', 'AMT_GOODS_PRICE']]
not_three_col = prev_app[cols]
plt.figure(figsize = (20,10))
plt.xticks(rotation = 90)
not_three_col.boxplot()
plt.show()
```



In the **second figure**, we observe that the values at 7 million are clearly the outliers.

```
plt.figure(figsize = (20,10))
plt.xticks(rotation = 90)
prev_app[['AMT_APPLICATION', 'AMT_CREDIT', 'AMT_GOODS_PRICE']].boxplot()
plt.show()
```

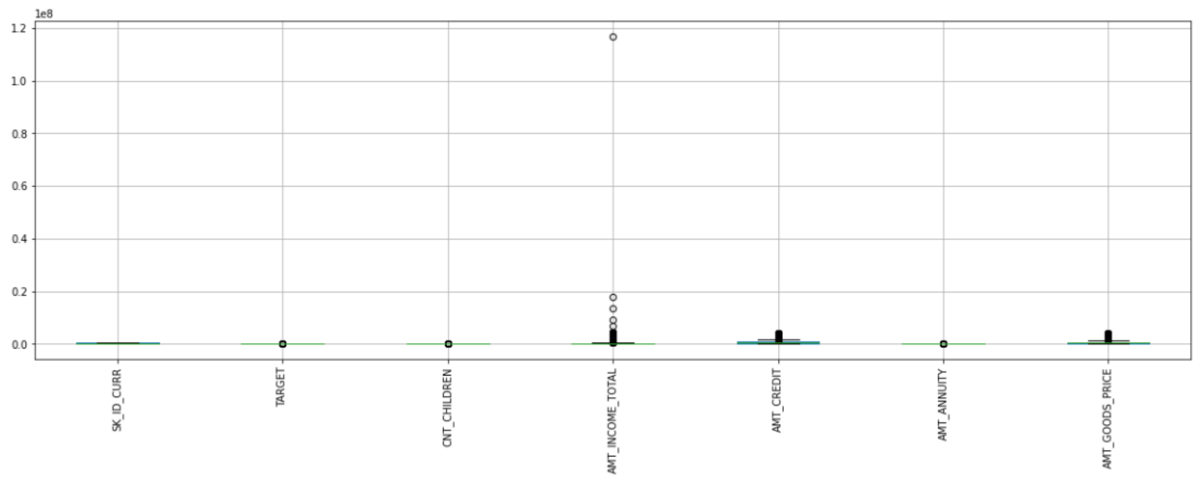


b. *Application data:*

a. In the **first figure**, we observe that $1.2*10^8$ is a clear outlier in the column AMT_INCOME_TOTAL

```
plt.figure(figsize = (20,6))
# plt.xticks(rotation = 90)
app_data.iloc[:,0:13].boxplot(rot = 90)
```

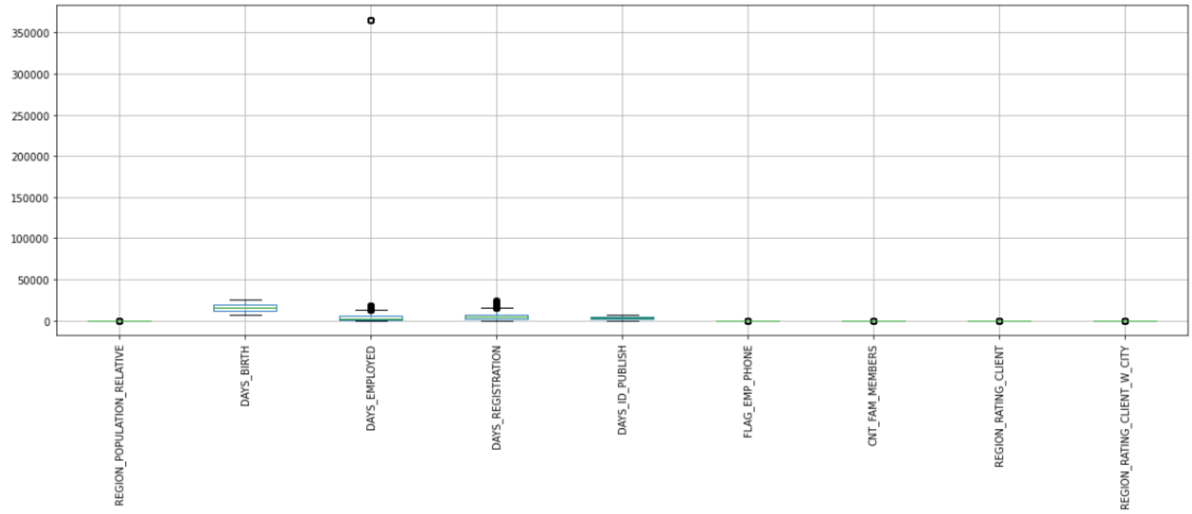
```
<matplotlib.axes._subplots.AxesSubplot at 0x1adacdc6700>
```



b. In the **second figure,** in the column DAYS_EMPLOYED we see an outlier at 37,500 days which equates to 102.75 years and a person cannot be employed for so long, indicating that it is an outlier.

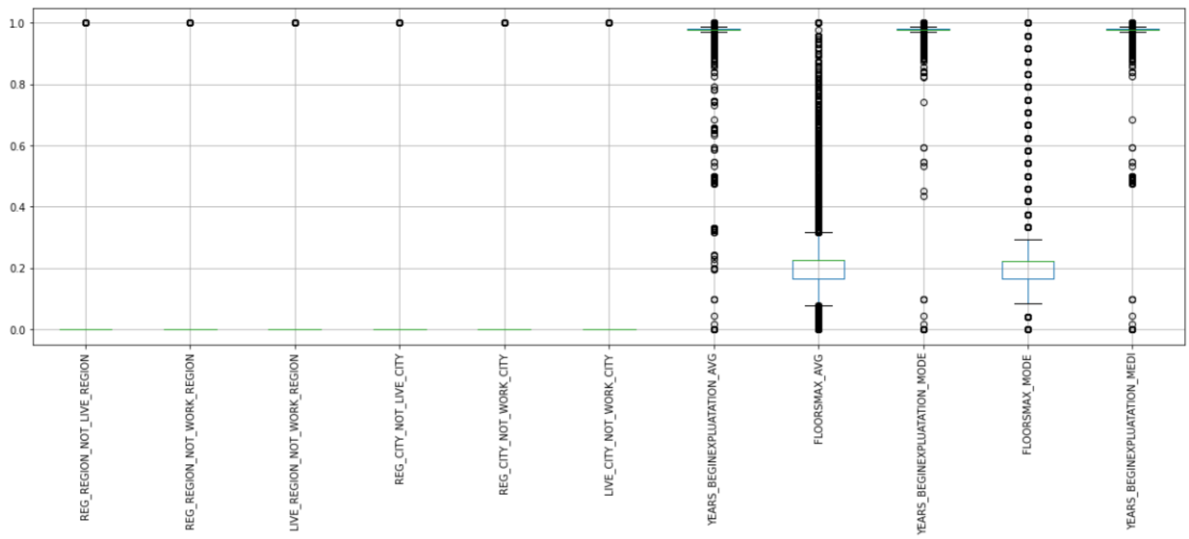
```
plt.figure(figsize = (20,6))
# plt.xticks(rotation = 90)
app_data.iloc[:,13:26].boxplot(rot = 90)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1adaf91b550>
```



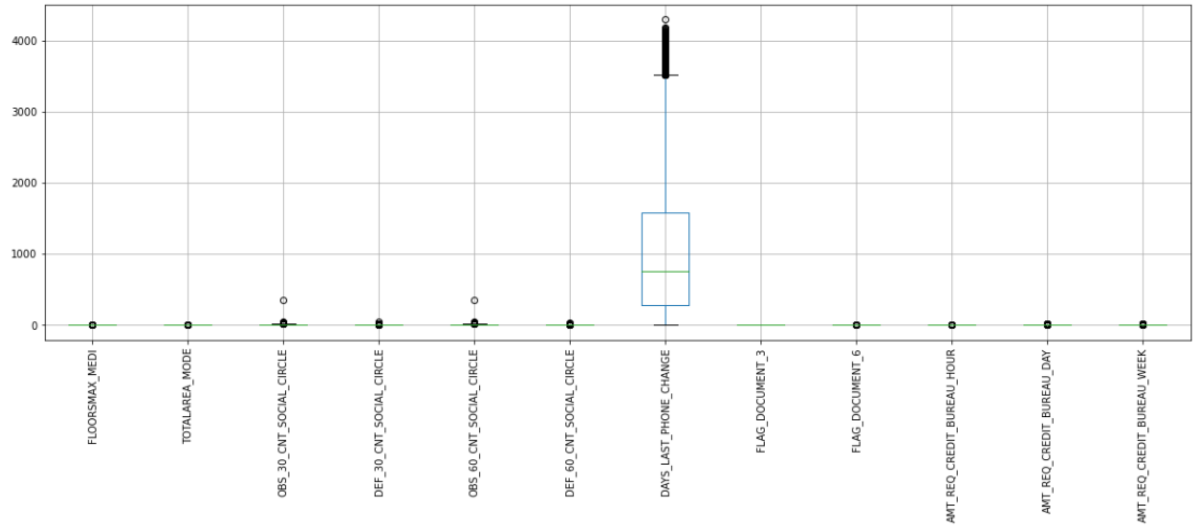
c. In the **third figure**, columns containing regions depict categorical values and therefore cannot be considered as outliers. Outliers can be observed in the last three columns of the figure.

```
plt.figure(figsize = (20,6))
# plt.xticks(rotation = 90)
app_data.iloc[:,26:39].boxplot(rot = 90)
plt.show()
```



d. In the fourth figure, the column DAYS_LAST_PHONE_CHANGE has outliers in the range 3,500 days to 4,500 days which correspond to 9.5 to 12.5 years approximately which is well outside the IQR range.

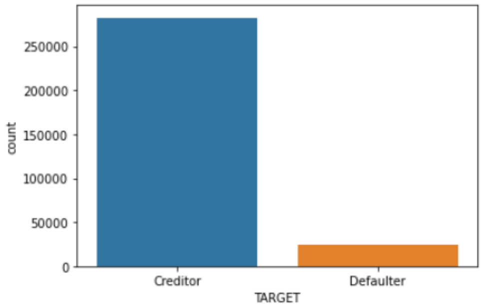
```
plt.figure(figsize = (20,6))
# plt.xticks(rotation = 90)
app_data.iloc[:,39:52].boxplot(rot = 90)
plt.show()
```



4. **<u>Identifying the data imbalance among columns:</u>**

Data imbalance occurs when the number of a particular type of data corresponds to 95% while the second type of data is only 5%. The disadvantages of data imbalance are:

```
sns.countplot(x = 'TARGET', data = app_data)
plt.xticks([0.0, 1.0], ['Creditor', 'Defaulter'])
plt.show()
```




```
print("Percentage of Creditors:",app_data['TARGET'].value_counts()[0.0]*100/len(app_data['TARGET']))
print("Percentage of Defaulters:",app_data['TARGET'].value_counts()[1.0]*100/len(app_data['TARGET']))
```

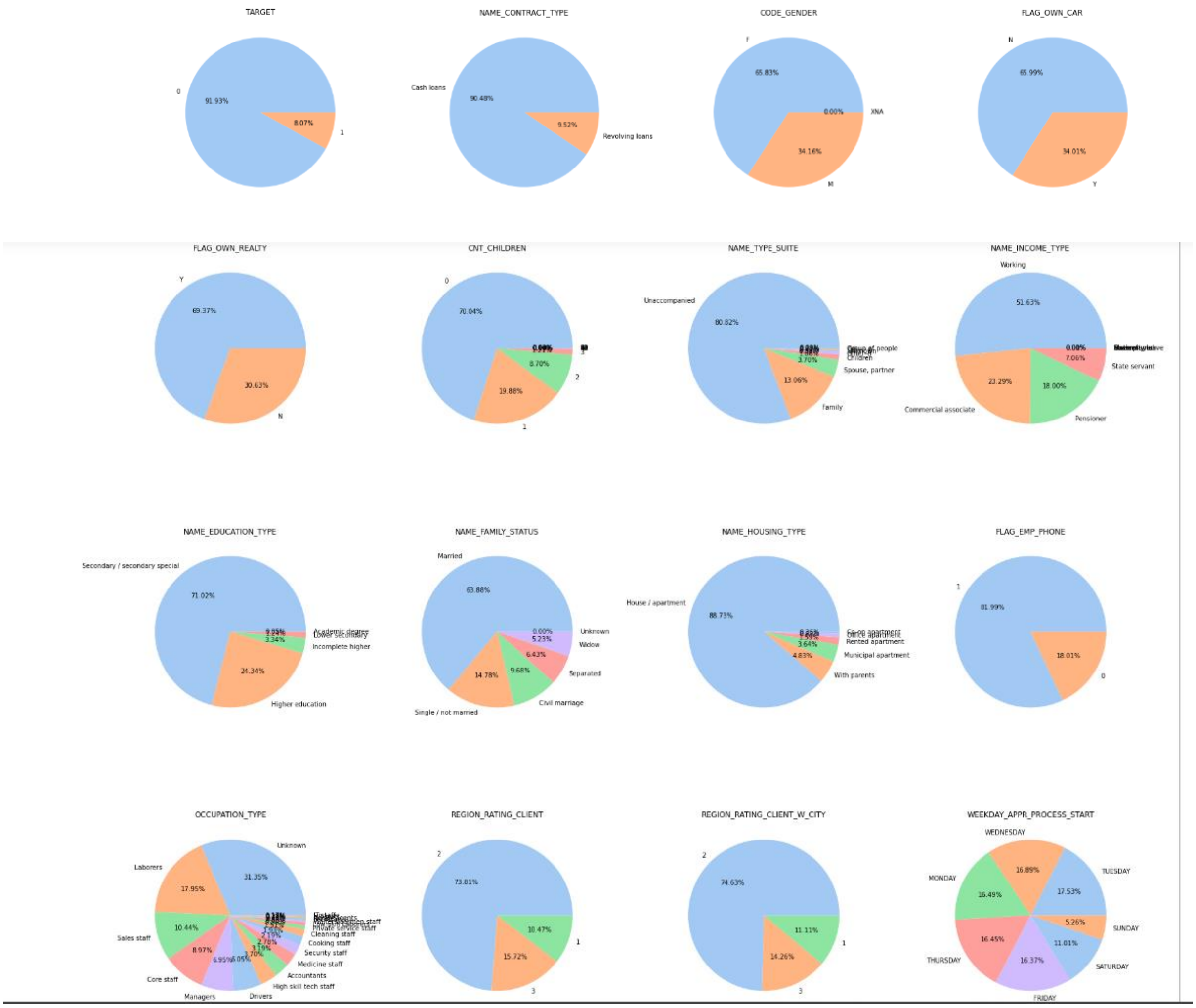
```
Percentage of Creditors: 91.92711805431351
Percentage of Defaulters: 8.072881945686495
```

The percentage of Creditors are approximately 92% and the defaulters are approximately 8%.

The data imbalance in other columns are depicted through a pie chart and are as follows:

DATA IMBALANCE

In [115]:
```
figure, ax = plt.subplots(4,4,figsize=(30,30), facecolor='w', edgecolor='k')
figure.subplots_adjust(hspace = 0.5, wspace=0.001)
row_num = 0
col_num = 0
colors = sns.color_palette('pastel')[0:5]
for col in categoric_app_data.columns:
    ax[row_num, col_num].pie(x =categoric_app_data[col].value_counts().to_list(),\
            labels = categoric_app_data[col].value_counts().keys().to_list(),\
            autopct = '%.2f%%', colors=colors)
    ax[row_num, col_num].set_title(col)
    col_num += 1
    if col_num%4==0:
        col_num = 0
        row_num+=1
    if row_num == 4:
        break
plt.show()
```



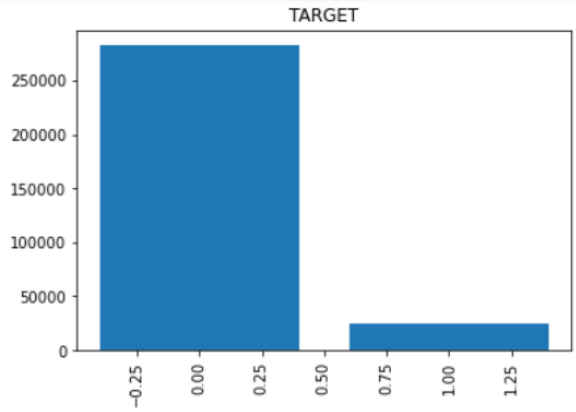
5. **<u>The results of Univariate, Segmented Univariate and Bivariate Analysis:</u>**
   <u>Definition Univariate Analysis</u>: Univariate analysis explores each variable in a data set, separately. It looks at the range of values, as well as the central tendency of the values. It describes the pattern of response to the variable. It describes each variable on its own. Descriptive statistics describe and summarize data.
   <u>Approach to perform Univariate Analysis</u>: To summarize and produce insights in each individual column, the dataset is divided into two: a. categorical columns b. numerical columns.
   For categorical data, countplot is used and for numerical data displot (a.k.a distribution plot) is used.

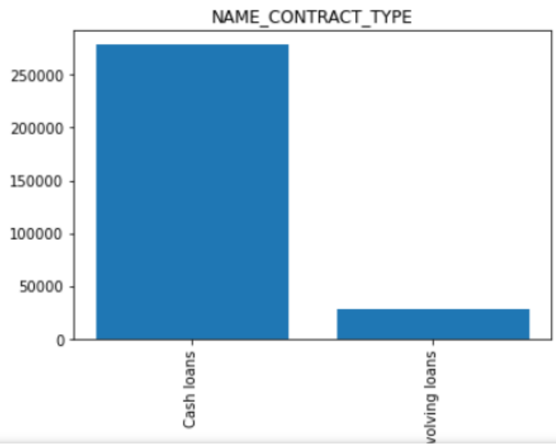
UNIVARIATE ANALYSIS

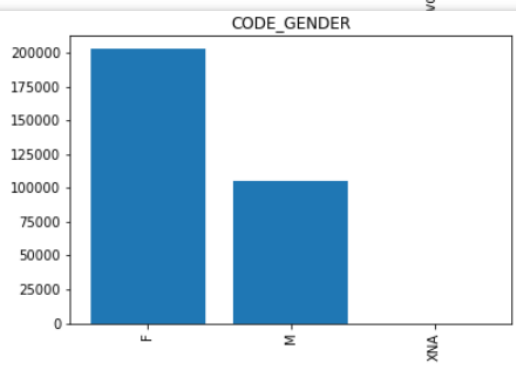
```
categoric_app_data = app_data.select_dtypes(exclude = ['float', 'int'])
categoric_app_data.sample(5)
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	NAME_TYPE_SUITE	NAME_
47026	154467	0	Cash loans	F	Y	Y	0	Unaccompanied	
74412	186295	0	Cash loans	F	N	Y	0	Unaccompanied	
273984	417595	1	Cash loans	M	Y	Y	1	Unaccompanied	
188401	318426	0	Cash loans	F	N	N	2	Unaccompanied	
236986	374493	0	Cash loans	F	N	Y	0	Unaccompanied	

```
for col in categoric_app_data.columns:
    counts = categoric_app_data[col].value_counts()
    plt.title(col)
    plt.bar(x = counts.keys().to_list(), height = counts.to_list())
    plt.xticks(rotation = 90)
    plt.show()
```





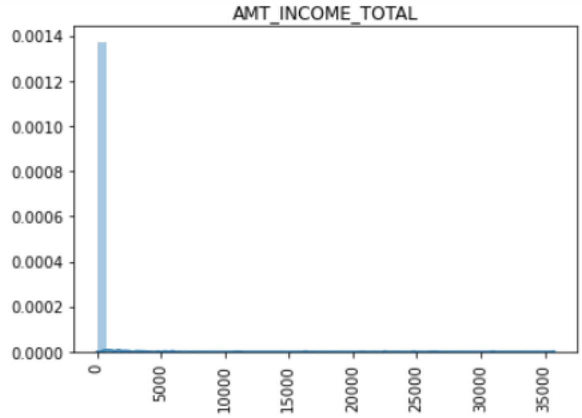


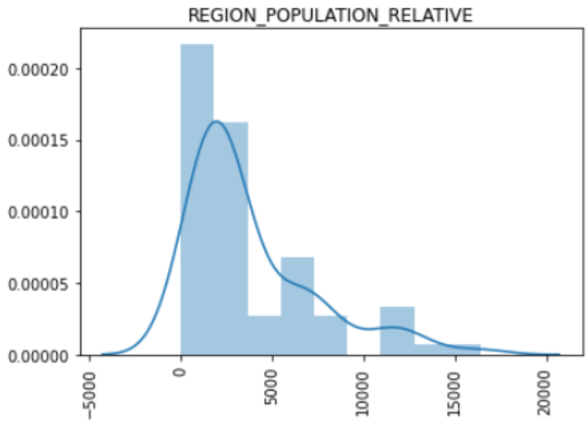
The list of other graphs is given in the working file:

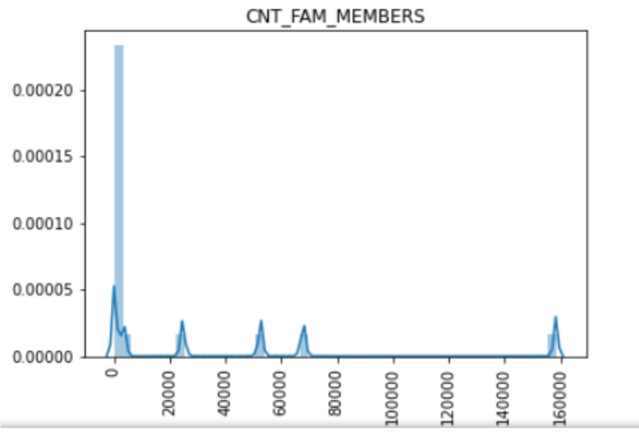
1. https://drive.google.com/file/d/1Lu_6iq0sLDruDFpBrjXkYHzBbInjTNCJ/view?usp=drive_link
2. https://drive.google.com/file/d/1x48y9IqxwNmlzTbfnwW7UFyMOsgSfFNs/view?usp=drive_link
3. https://drive.google.com/file/d/1x48y9IqxwNmlzTbfnwW7UFyMOsgSfFNs/view?usp=sharing

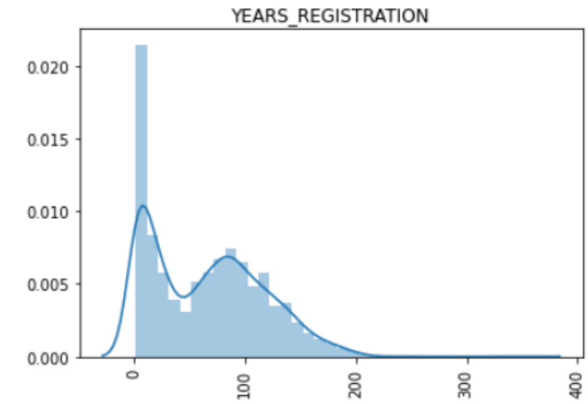
Similarly, on performing univariate analysis for numerical data using displot( ) from seaborn library, the following results are obtained:

```python
for col in num_app_data.columns:
    counts = num_app_data[col].value_counts()
    plt.title(col)
    sns.distplot(counts.to_list())
    plt.xticks(rotation = 90)
    plt.show()
```









The other graphs are given in the working file:

1. https://drive.google.com/file/d/1Lu_6iq0sLDruDFpBrjXkYHzBbInjTNCJ/view?usp=drive_link
2. https://drive.google.com/file/d/1x48y9IqxwNmlzTbfnwW7UFyMOsgSfFNs/view?usp=drive_link
3. https://drive.google.com/file/d/1x48y9IqxwNmlzTbfnwW7UFyMOsgSfFNs/view?usp=sharing

Segmented Univariate Analysis: Segmented Univariate Analysis is one of the simplest form of visualization to analyse data. In its name ‘Uni’ means one which itself describes that it considers only a single data variable for analysis. analysis. Segmented analysis here means that the data variable is analysed in subsets and is very useful as it can show the change metric in pattern across the different segments of the same variable.

```
numeric_app_data['AMT_INCOME_TOTAL(L)'] = app_data['AMT_INCOME_TOTAL']/100000
numeric_app_data['AMT_CREDIT (L)'] = app_data['AMT_CREDIT']/100000
numeric_app_data['AMT_ANNUITY (L)'] = app_data['AMT_ANNUITY']/100000
numeric_app_data['AMT_GOODS_PRICE (L)'] = app_data['AMT_GOODS_PRICE']/100000
```

```
numeric_app_data.drop('AMT_INCOME_TOTAL', inplace = True, axis = 1)
```

```
numeric_app_data.drop(['AMT_CREDIT','AMT_ANNUITY','AMT_GOODS_PRICE'], inplace = True, axis = 1)
```

```
numeric_app_data.sample(2)
```

	REGION_POPULATION_RELATIVE	CNT_FAM_MEMBERS	YEARS_BEGINEXPLUATATION_AVG	FLOORSMAX_AVG	YEARS_BEGINEXPLUATATION_MODE	FLC
66002	0.026392	2.0	0.9876	0.1250	0.9826	
78842	0.032561	2.0	0.9965	0.3167	0.9965	

```
bins = [0,1,2,3,4,5,6,7,8,9,10,1170]
Credit_bins = ['0L-1L',' 1L-2L',' 2L-3L','3L-4L','4L-5L','5L-6L','6L-7L','7L-8L','8L-9L','9L-10L','10L and above']
```

```
numeric_app_data['AMT_INCOME_TOTAL(L)'] = pd.cut(numeric_app_data['AMT_INCOME_TOTAL(L)'], bins, labels = Credit_bins)
```

```
numeric_app_data.sample(2)
```

The Amt_Income and Amt_Credit columns, Amt_Goods_Price columns are converted to bins like the picture shown above.

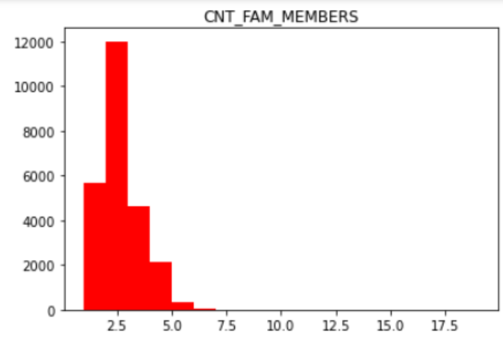
Approach to perform Segmented Univariate Analysis:

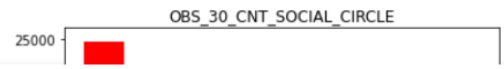
Target Column: The 'TARGET' column is selected. The value 0 is for Creditor and the value 1 is for Defaulter.

Problem Type: Select the type of problem, Classification

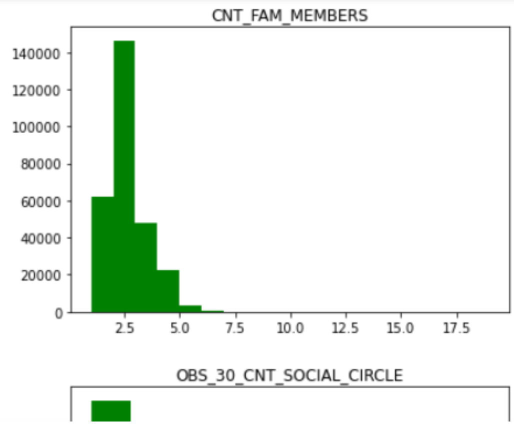
Numeric Variable (Optional): The numerical data is selected along with Target variable and the creditor and defaulters are shown in each category. In the following figures, the green colour is for Creditors and the red colour depicts Defaulters.

```
# Segmented Univariate Analysis of Numeric data
# figure, ax = plt.subplots(1,2)
for col in numeric_app_data.columns:
    minimum = int(numeric_app_data[col].min())
    maximum = int(numeric_app_data[col].max())
    d = int((maximum - minimum)/10)
    if d!=0:
        plt.title(col)
        plt.hist(numeric_app_data[col][numeric_app_data['TARGET']==1], bins = range(minimum, maximum, d),color = 'r')
        plt.show()
```





```
for col in numeric_app_data.columns:
    minimum = int(numeric_app_data[col].min())
    maximum = int(numeric_app_data[col].max())
    d = int((maximum - minimum)/10)
    if d!=0:
        plt.title(col)
        plt.hist(numeric_app_data[col][numeric_app_data['TARGET']==0], bins = range(minimum, maximum, d),color = 'g')
        plt.show()
```



The other graphs are given in the working file:

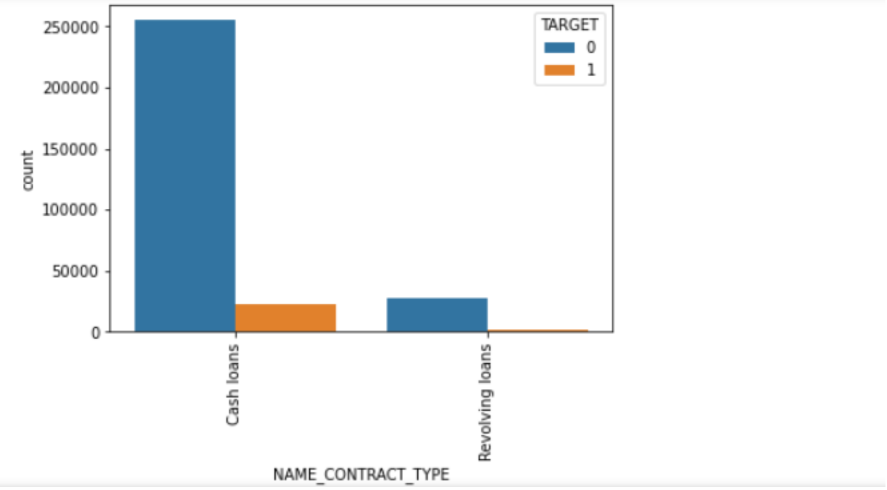
1. https://drive.google.com/file/d/1Lu_6iq0sLDruDFpBrjXkYHzBbInjTNCJ/view?usp=drive_link

2. https://drive.google.com/file/d/1x48y9IqxwNmlzTbfnwW7UFyMOsgSfFNs/view?usp=drive_link

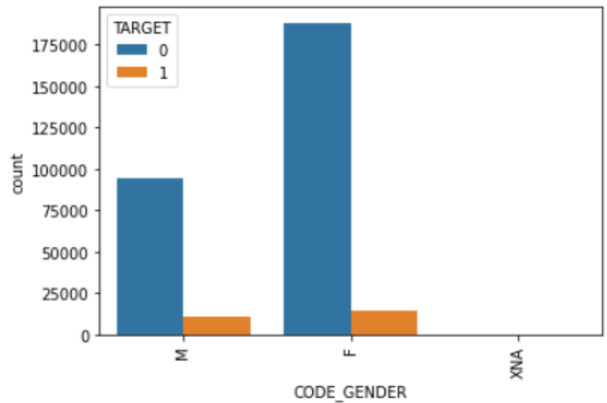
3. https://drive.google.com/file/d/1x48y9IqxwNmlzTbfnwW7UFyMOsgSfFNs/view?usp=sharing

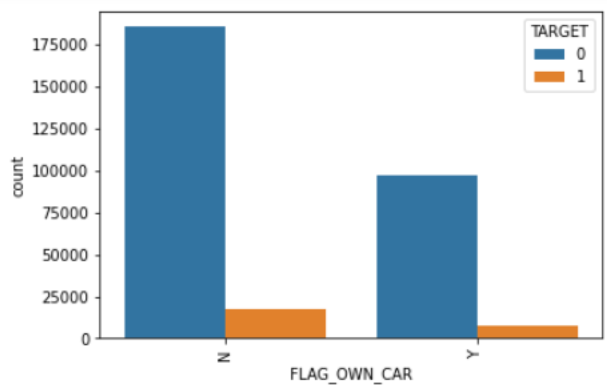
Categorical Variable (Optional): Similar to numerical data, the defaulters and creditors in each category are shown. In the following figure we observe that, the creditors are large in case of Cash Loans.

```python
for col in categoric_app_data.columns:
    if col != 'TARGET':
        sns.countplot(x = col, data = categoric_app_data, hue = 'TARGET')
        plt.xticks(rotation = 90)
        plt.show()
```

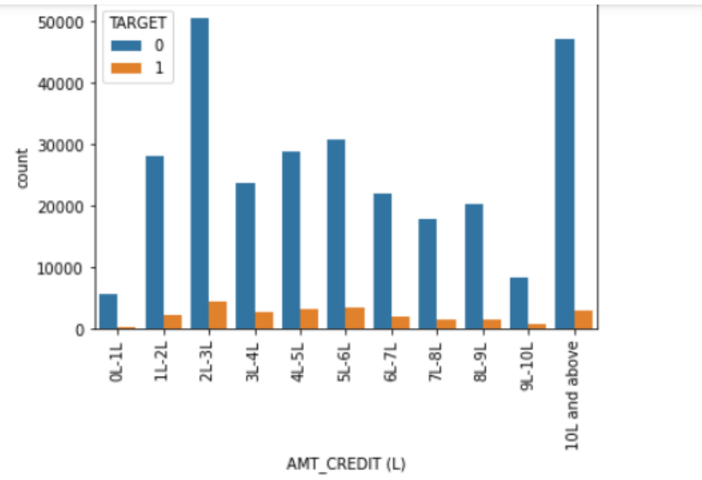


In the following figure, we observe that females are more committed to replaying the loan than the males





In the following figure, the Amt is turned to bins as shown in the previous code snippet and the following graph is produced.



Conclusion from Segmented Univariate Analysis:

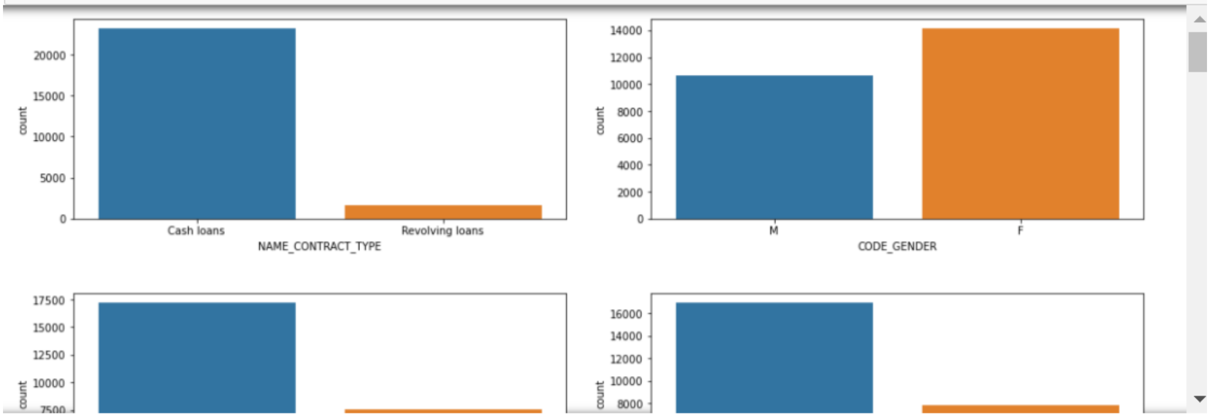
Creditors:

a. Numerical data: By observing the segmented analysis of numeric data we can draw that the clients who are around **the age of 40**, with years_id_publish around **12 years**, and count **of family members 2.5** are **more likely to repay the loan.**

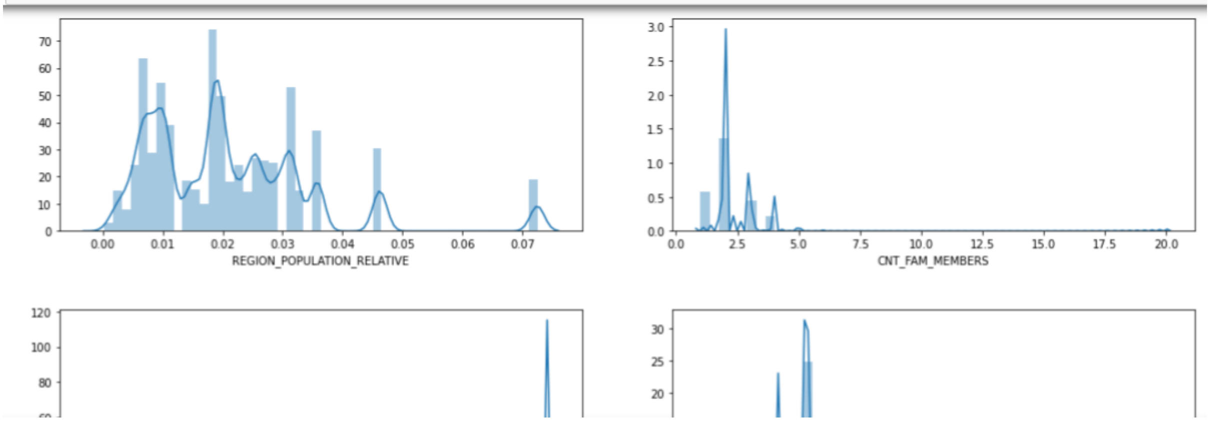
b. Categorical Data: From the graphs, we observe that, The **female** clients, with contract type **Cash Loans** who **do not own a car**, who **own a realty** and have **no children,** who are **Married** and **Working** and **own a Family,** have an **Amt_Credit** of 2L-3L and Amt_Income of **1L-**2L and are **not in Emergency** are **more likely** to **pay** the loan.

Defaulters:

```python
# Target 1 is Defaultor
figure, axs = plt.subplots(13,2, figsize = (16,48))
figure.tight_layout(pad=5.0)
row_num, col_num = 0, 0
for col in categoric_app_data.columns:
    if col != 'TARGET':
        sns.countplot(x = col, data = new_app_data, ax = axs[row_num, col_num])
        plt.title("Defaultors in "+col)
        plt.xticks(rotation = 45)
        col_num += 1
        if col_num%2 == 0:
            col_num = 0
            row_num += 1
        if row_num == 13:
            break
```




```python
# Target 1 is Defaultor for Numeric DataD
figure, axs = plt.subplots(13,2, figsize = (16,48))
figure.tight_layout(pad=5.0)
row_num, col_num = 0, 0
for col in numeric_app_data.columns:
    if col != 'TARGET':
        sns.distplot(numeric_app_data[col], ax = axs[row_num, col_num])
        plt.title("Defaultors in "+col)
        plt.xticks(rotation = 45)
        col_num += 1
        if col_num%2 == 0:
            col_num = 0
            row_num += 1
        if row_num == 13:
            break
```



a. Numerical Data: The above defaulters graph shows that the family members with **2** and the Region_Population_Relative of 0.02 with floor max avg nearly 0.25 with days last phone change around 0 are likely to default the repayment of loan

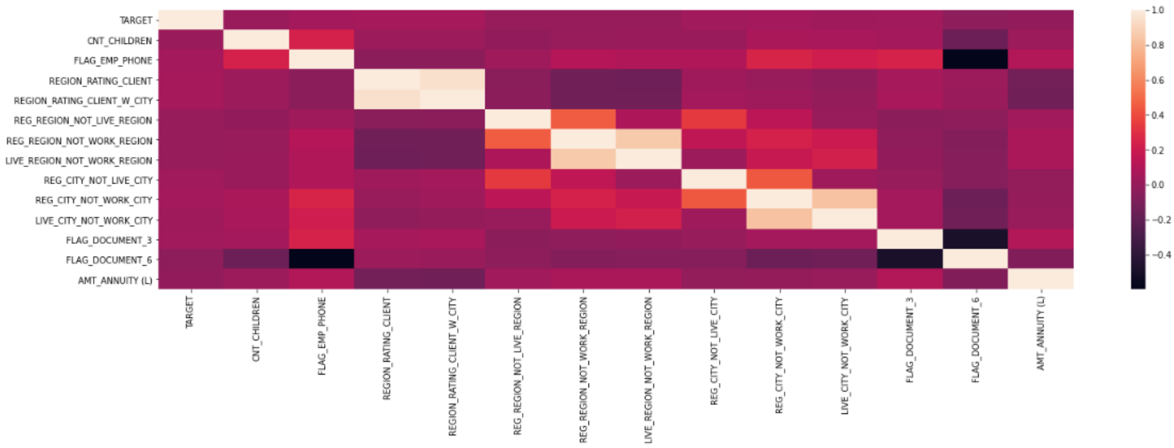
b. Categorical Data: From the graphs, we observe that, The **female** clients, with contract type **Cash Loans** who **do not own a car**, who **own a realty** and have **no children,** who are **Married** and **Working** and **own a Family** and whose application process started on Tuesday with unknown Emergency state in **Business field** are **more likely** to be a defaulter.

BIVARIATE ANALYSIS

```
# Bivariate Analysis can be done in two ways - scatter plot and by correlation coeffiecient
```

```
plt.figure(figsize = (24, 6))
sns.heatmap(categoric_app_data.corr())
```

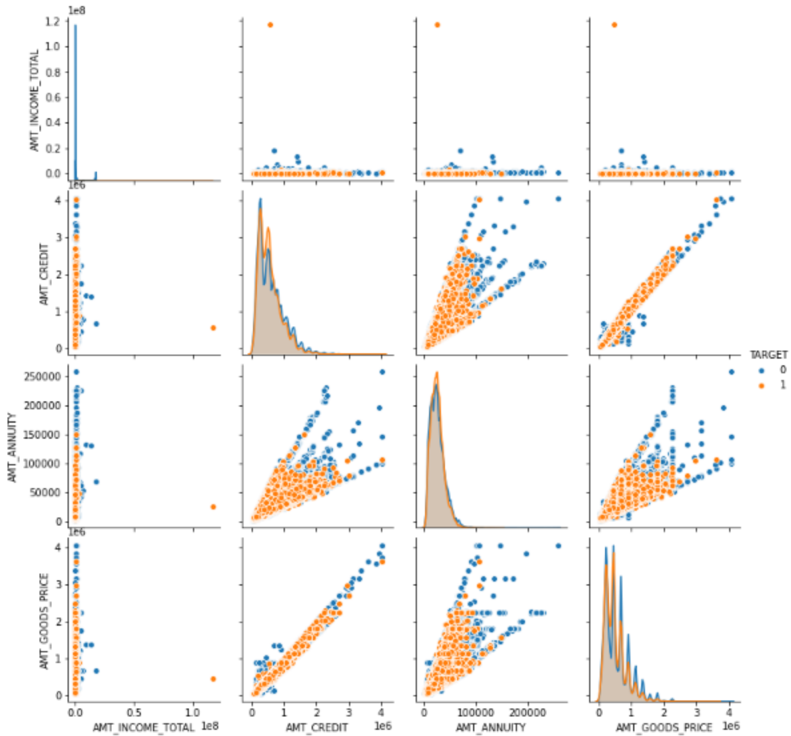
```
<matplotlib.axes._subplots.AxesSubplot at 0x2a27c1e9f70>
```



The heat map is plotted for Categorical data and pairplots are used to depict numerical data.

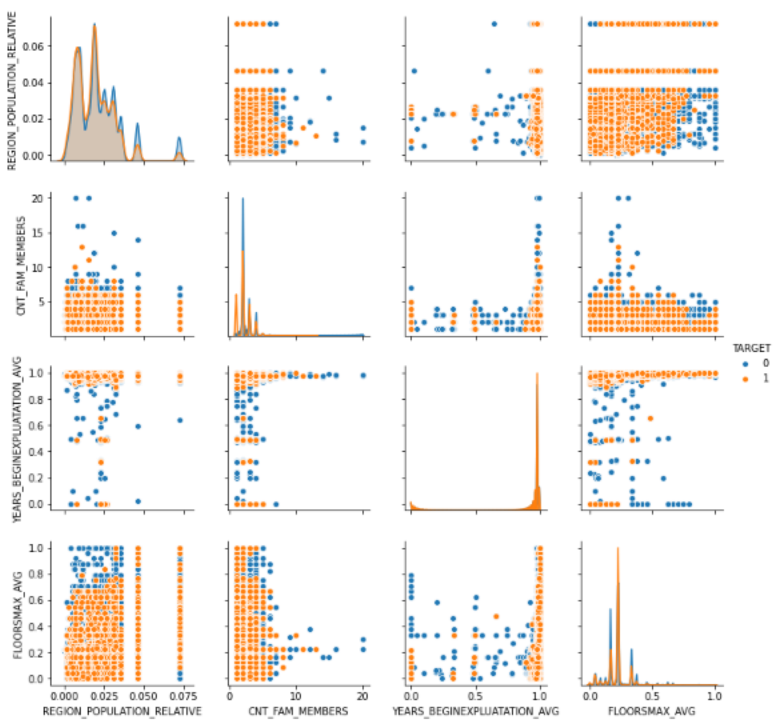
```
ndf = num_app_data[['AMT_INCOME_TOTAL','AMT_CREDIT','AMT_ANNUITY','AMT_GOODS_PRICE','TARGET']]
sns.pairplot(ndf, hue = 'TARGET')
```

```
<seaborn.axisgrid.PairGrid at 0x2a261afaa00>
```



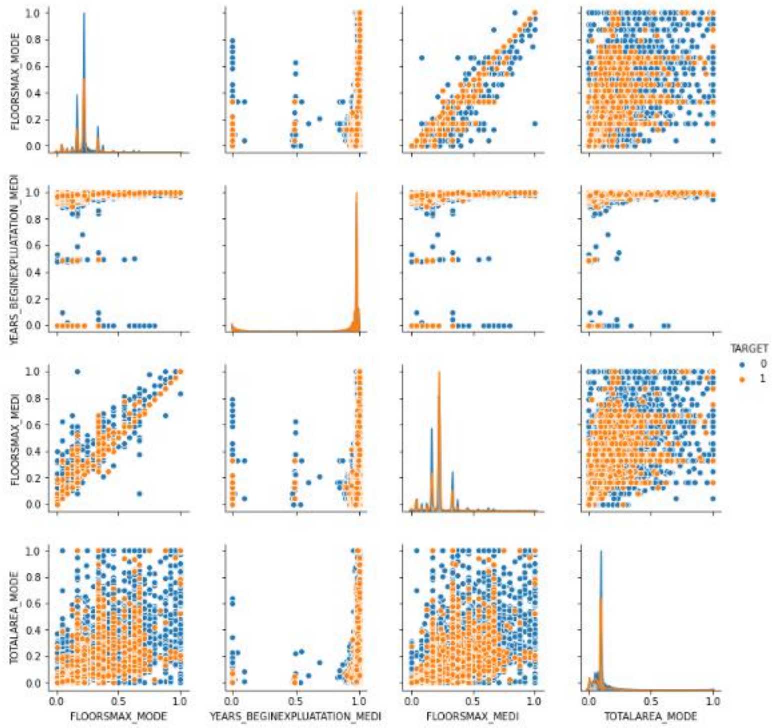

```
ndf = num_app_data[['REGION_POPULATION_RELATIVE','CNT_FAM_MEMBERS','YEARS_BEGINEXPLUATATION_AVG','FLOORSMAX_AVG','TARGET']]
sns.pairplot(ndf, hue = 'TARGET')
```

```
<seaborn.axisgrid.PairGrid at 0x2a266622f10>
```



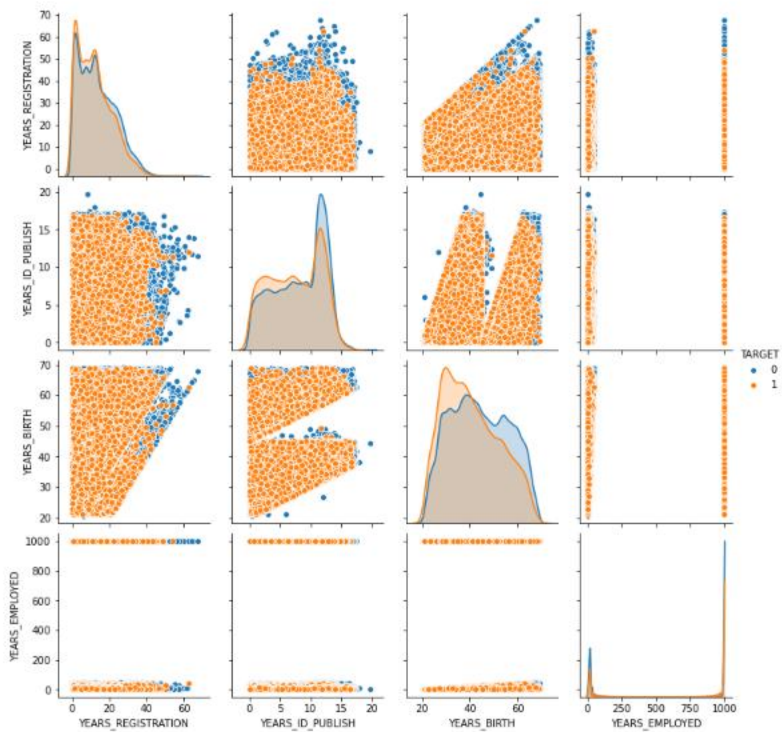
```
ndf = num_app_data[['FLOORSMAX_MODE','YEARS_BEGINEXPLUATATION_MEDI','FLOORSMAX_MEDI','TOTALAREA_MODE','TARGET']]
sns.pairplot(ndf, hue = 'TARGET')
```

<seaborn.axisgrid.PairGrid at 0x2a244aa7e20>




```
ndf = num_app_data[['YEARS_REGISTRATION','YEARS_ID_PUBLISH','YEARS_BIRTH','YEARS_EMPLOYED','TARGET']]
sns.pairplot(ndf, hue = 'TARGET')
```

<seaborn.axisgrid.PairGrid at 0x2a235262a00>



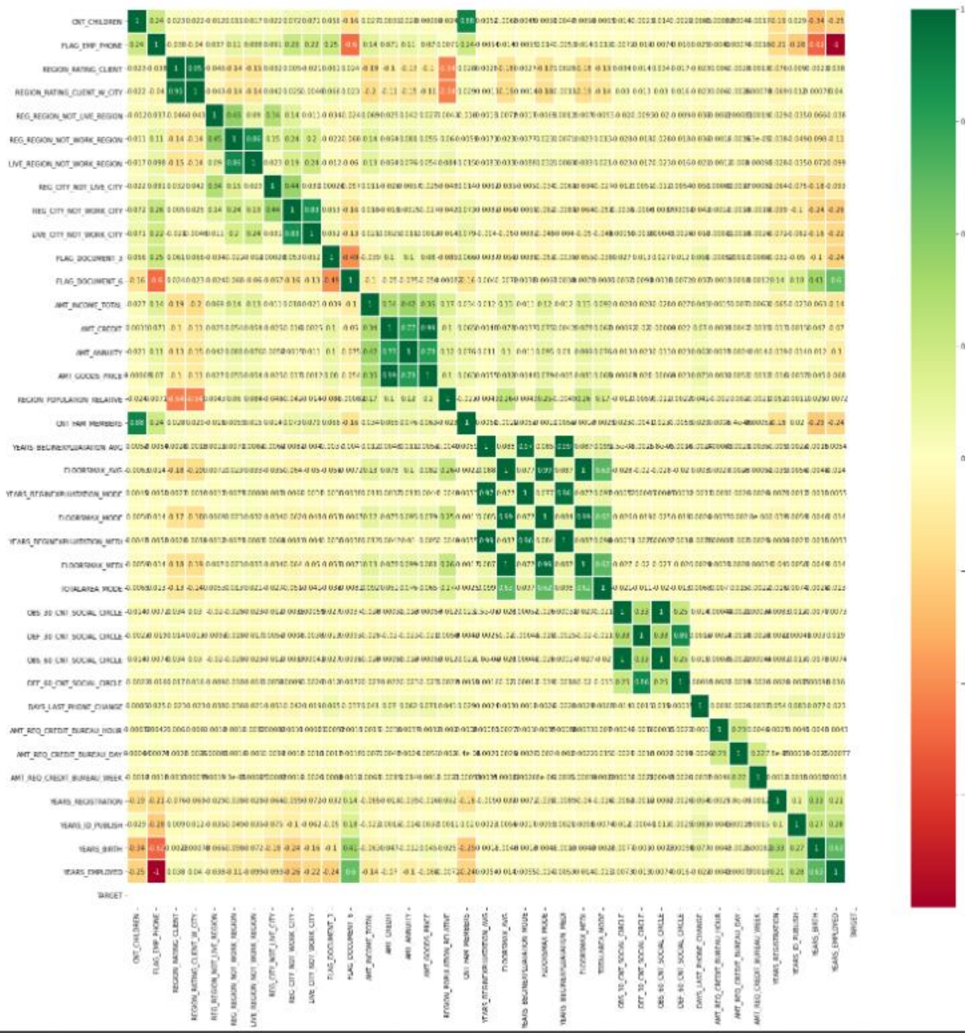
6. Top Correlations in different Scenarios

```
# Top 10 Correlations for Defaulter
defaulter_mixed_data = mixed_data[mixed_data['TARGET'] == 1]
dmd = defaulter_mixed_data.corr().unstack().abs().sort_values().dropna()
dmd
```

```
REGION_RATING_CLIENT           REGION_RATING_CLIENT_W_CITY    0.956637
YEARS_BEGINEXPLUATATION_MODE   YEARS_BEGINEXPLUATATION_MEDI   0.978080
YEARS_BEGINEXPLUATATION_MEDI   YEARS_BEGINEXPLUATATION_MODE   0.978080
YEARS_BEGINEXPLUATATION_MODE   YEARS_BEGINEXPLUATATION_AVG    0.980472
YEARS_BEGINEXPLUATATION_AVG    YEARS_BEGINEXPLUATATION_MODE   0.980472
AMT_CREDIT                     AMT_GOODS_PRICE                0.983115
AMT_GOODS_PRICE                AMT_CREDIT                     0.983115
FLOORSMAX_MODE                 FLOORSMAX_AVG                  0.986790
FLOORSMAX_AVG                  FLOORSMAX_MODE                 0.986790
FLOORSMAX_MODE                 FLOORSMAX_MEDI                 0.989356
FLOORSMAX_MEDI                 FLOORSMAX_MODE                 0.989356
YEARS_BEGINEXPLUATATION_MEDI   YEARS_BEGINEXPLUATATION_AVG    0.996125
YEARS_BEGINEXPLUATATION_AVG    YEARS_BEGINEXPLUATATION_MEDI   0.996125
FLOORSMAX_AVG                  FLOORSMAX_MEDI                 0.997231
FLOORSMAX_MEDI                 FLOORSMAX_AVG                  0.997231
OBS_30_CNT_SOCIAL_CIRCLE       OBS_60_CNT_SOCIAL_CIRCLE       0.998269
OBS_60_CNT_SOCIAL_CIRCLE       OBS_30_CNT_SOCIAL_CIRCLE       0.998269
FLAG_EMP_PHONE                 YEARS_EMPLOYED                 0.999705
YEARS_EMPLOYED                 FLAG_EMP_PHONE                 0.999705
CNT_CHILDREN                   CNT_CHILDREN                   1.000000
```

```python
#Correlations for Creditor

creditor_mixed_data = mixed_data[mixed_data['TARGET'] == 0]
cmd = creditor_mixed_data.corr().unstack().abs().sort_values().dropna()

cmd
```

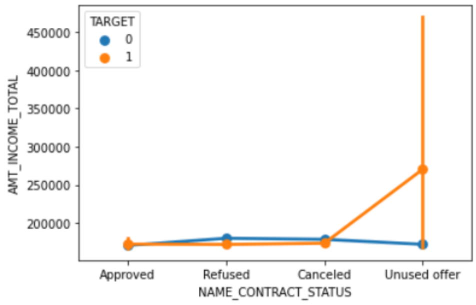
```
CNT_CHILDREN                    CNT_FAM_MEMBERS                 0.878570
CNT_FAM_MEMBERS                 CNT_CHILDREN                    0.878570
REGION_RATING_CLIENT            REGION_RATING_CLIENT_W_CITY     0.950149
REGION_RATING_CLIENT_W_CITY     REGION_RATING_CLIENT            0.950149
YEARS_BEGINEXPLUATATION_MEDI    YEARS_BEGINEXPLUATATION_MODE    0.962061
YEARS_BEGINEXPLUATATION_MODE    YEARS_BEGINEXPLUATATION_MEDI    0.962061
                                YEARS_BEGINEXPLUATATION_AVG     0.971029
YEARS_BEGINEXPLUATATION_AVG     YEARS_BEGINEXPLUATATION_MODE    0.971029
FLOORSMAX_AVG                   FLOORSMAX_MODE                  0.985592
FLOORSMAX_MODE                  FLOORSMAX_AVG                   0.985592
AMT_CREDIT                      AMT_GOODS_PRICE                 0.987254
AMT_GOODS_PRICE                 AMT_CREDIT                      0.987254
FLOORSMAX_MEDI                  FLOORSMAX_MODE                  0.988146
FLOORSMAX_MODE                  FLOORSMAX_MEDI                  0.988146
YEARS_BEGINEXPLUATATION_MEDI    YEARS_BEGINEXPLUATATION_AVG     0.993582
YEARS_BEGINEXPLUATATION_AVG     YEARS_BEGINEXPLUATATION_MEDI    0.993582
FLOORSMAX_AVG                   FLOORSMAX_MEDI                  0.997018
FLOORSMAX_MEDI                  FLOORSMAX_AVG                   0.997018
OBS_60_CNT_SOCIAL_CIRCLE        OBS_30_CNT_SOCIAL_CIRCLE        0.998508
OBS_30_CNT_SOCIAL_CIRCLE        OBS_60_CNT_SOCIAL_CIRCLE        0.998508
```



7. **After merging previous application and application data**

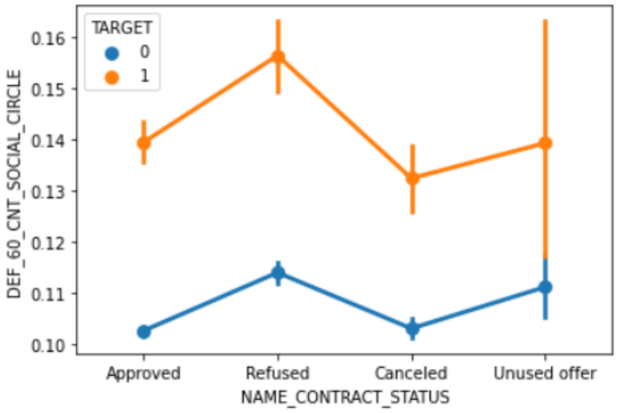
In [20]:
```
# plotting the relationship between income total and contact status
sns.pointplot(hue = 'TARGET', x = 'NAME_CONTRACT_STATUS', y = 'AMT_INCOME_TOTAL', data = loan_df)
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x25a05056430>



In [21]:
```
sns.pointplot(data = loan_df,hue = "TARGET",x = "NAME_CONTRACT_STATUS",y = 'DEF_60_CNT_SOCIAL_CIRCLE')
```

<matplotlib.axes._subplots.AxesSubplot at 0x25a053deac0>

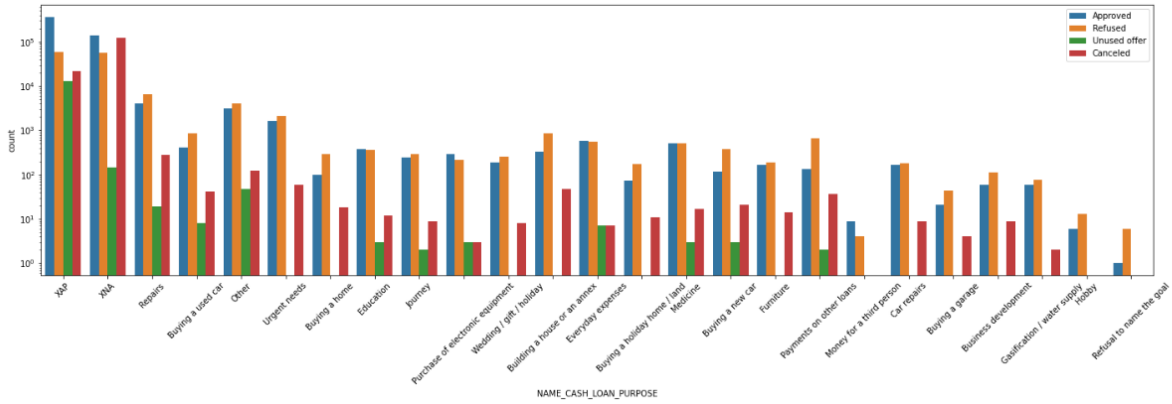


The above figures show relation between Contract Status and Creditors, defaulters (**Bivariate Analysis**). It shows that for Unused Offer, the people having high income are likely to default more.

In the below figures we see the relationship (**Bivariate Analysis**) of NAME_CASH_LOAN_PURPOSE and NAME_CONTRACT_STATUS for variables with creditors and defaulters.

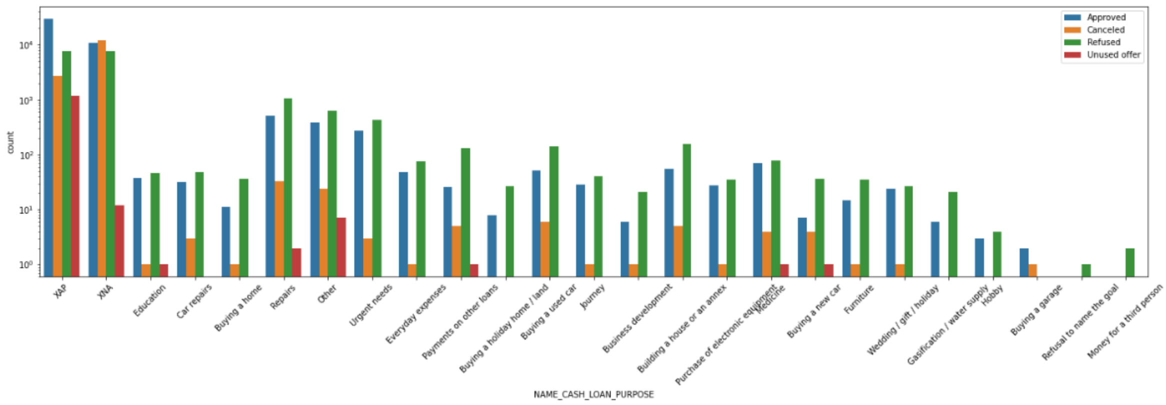
```
plt.figure(figsize = (24, 6))
sns.countplot('NAME_CASH_LOAN_PURPOSE', data = L0, hue = 'NAME_CONTRACT_STATUS')
plt.yscale('log')
plt.legend(loc = 'upper right')
plt.xticks(rotation = 45)
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24]),
 <a list of 25 Text major ticklabel objects>)
```




```
plt.figure(figsize = (24, 6))
sns.countplot('NAME_CASH_LOAN_PURPOSE', data = L1, hue = 'NAME_CONTRACT_STATUS')
plt.yscale('log')
plt.legend(loc = 'upper right')
plt.xticks(rotation = 45)
```

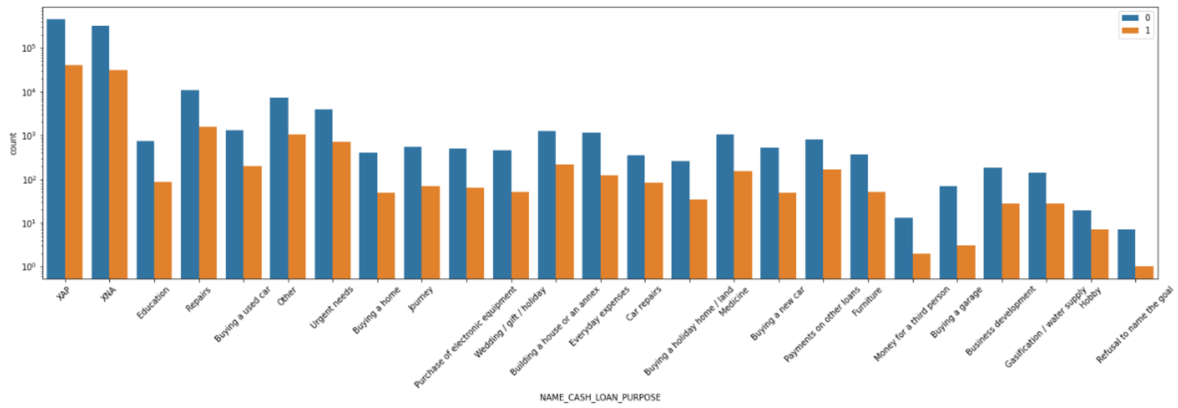
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24]),
 <a list of 25 Text major ticklabel objects>)
```



The below graph shows **Segmented Univariate Analysis** for the merged data.

```
plt.figure(figsize = (24, 6))
sns.countplot('NAME_CASH_LOAN_PURPOSE', data = loan_df, hue = 'TARGET')
plt.yscale('log')
plt.legend(loc = 'upper right')
plt.xticks(rotation = 45)
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24]),
 <a list of 25 Text major ticklabel objects>)
```



Result:

1. As a Data Analyst, we have mined insights on which type of people should be awarded the loans and who should not be given
2. There is high correlation between Amt_Credit and Amt_Annuity and Amt_Goods_Price.
3. We also see that female with Cash Loans and having 2 children with Amount Credit of **2L-3L** are highest defaulters.
4. The data imbalance is shown using pie charts.
5. In the process of solving this project, I learnt show casing the data using different charts of seaborn, using correlation maps, finding relation and dependency between different features, univariate analysis, segmented univariate analysis, bivariate analysis.