Bank Loan Case Study

BY: ADITI SAWANT

PROJECT DESCRIPTION:

The objective of this project is to analyse and evaluate the credit risk of potential loan applicants for a bank. This will help the bank to efficiently evaluate loan applications and minimize the risk of default. The dataset consists of various factors that affect the loan approval decision, such as credit score, income, loan amount, loan term, employment status, and other demographic information. The dataset also includes the loan approval status of each applicant. This project will help the bank to reduce the risk of default and streamline the loan approval process.

APPROACH

The project involves the following steps:

Data Cleaning and Pre-processing: This step involves cleaning and preparing the data for analysis. The data may contain missing values, outliers, or incorrect values that need to be addressed.

Exploratory Data Analysis (EDA): In this step, the data will be analysed to understand the relationships between variables and identify any patterns or trends that may exist.

The final step involves interpreting and explaining the insights to the stake holders. This will help the bank to reduce the risk of default and streamline the loan approval process.

TECH-STACK USED

For this project I used Jupyter Notebook(Anaconda) to run my queries and charts. It is used widely in data science, machine learning, and scientific computing.

I also used MS Word for representing all the content visible in the application and include input and output of the computation.

RESULTS

First we import all the libraries that are needed:

```
In [3]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt
        import scipy.stats as stats
        %matplotlib inline
```

Numpy is a powerful library in python that supports large, multi-dimensional arrays and matrices along with wide range of mathematical functions to operate on them.

Pandas is another popular library for data manipulation and analysis in python that provides data structures for efficiently storing and manipulating large datasets and functions for cleaning, transforming and analysing data.

Seaborn provides wide range of functions for creating informative statistical graphics.

Matplotlib provides wide range of tools for creating high-quality plots, graphs and charts. The pyplot module provides convenient interface for creating and customizing plots.

Scipy provides wide range of functions for mathematics, science and engineering. The stats module provides functions for statistical analysis, probability distributions and hypothesis testing.

%matplotlib inline command enables the display of matplotlib plots below the code cells that produces them.

Next we upload the dataset file given to us:

```
In [4]: df=pd.read_csv('e:/trainity/project 6/previous_application.csv')
        print(df.head())
```

```
   SK_ID_PREV  SK_ID_CURR NAME_CONTRACT_TYPE  AMT_ANNUITY  AMT_APPLICATION  \
0     2030495      271877     Consumer loans     1730.430          17145.0
1     2802425      108129         Cash loans    25188.615         607500.0
2     2523466      122040         Cash loans    15060.735         112500.0
3     2819243      176158         Cash loans    47041.335         450000.0
4     1784265      202054         Cash loans    31924.395         337500.0

   AMT_CREDIT  AMT_DOWN_PAYMENT  AMT_GOODS_PRICE WEEKDAY_APPR_PROCESS_START  \
0     17145.0               0.0          17145.0                   SATURDAY
1    679671.0               NaN         607500.0                   THURSDAY
2    136444.5               NaN         112500.0                    TUESDAY
3    470790.0               NaN         450000.0                     MONDAY
4    404055.0               NaN         337500.0                   THURSDAY

   HOUR_APPR_PROCESS_START  ... NAME_SELLER_INDUSTRY  CNT_PAYMENT  \
0                       15  ...         Connectivity         12.0
1                       11  ...                  XNA         36.0
2                       11  ...                  XNA         12.0
3                        7  ...                  XNA         12.0
4                        9  ...                  XNA         24.0

   NAME_YIELD_GROUP       PRODUCT_COMBINATION  DAYS_FIRST_DRAWING  \
0            middle  POS mobile with interest            365243.0
1        low_action          Cash X-Sell: low            365243.0
2              high         Cash X-Sell: high            365243.0
3            middle       Cash X-Sell: middle            365243.0
4              high         Cash Street: high                 NaN

  DAYS_FIRST_DUE DAYS_LAST_DUE_1ST_VERSION  DAYS_LAST_DUE DAYS_TERMINATION  \
0          -42.0                     300.0          -42.0            -37.0
1         -134.0                     916.0       365243.0         365243.0
```

After reading the csv file, the head() method was used to print the first 5 rows of the data frame.

To get the information about the data frame created above info() method is used as below:

In [5]:
```
df.info(null_counts=True)
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_14764\1982639406.py:1: FutureWarning:
stead
  df.info(null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   SK_ID_PREV                   1670214 non-null  int64
 1   SK_ID_CURR                   1670214 non-null  int64
 2   NAME_CONTRACT_TYPE           1670214 non-null  object
 3   AMT_ANNUITY                  1297979 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_DOWN_PAYMENT             774370 non-null   float64
 7   AMT_GOODS_PRICE              1284699 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object
 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64
 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object
 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64
 12  RATE_DOWN_PAYMENT            774370 non-null   float64
 13  RATE_INTEREST_PRIMARY        5951 non-null     float64
 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64
 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object
 16  NAME_CONTRACT_STATUS         1670214 non-null  object
 17  DAYS_DECISION                1670214 non-null  int64
 18  NAME_PAYMENT_TYPE            1670214 non-null  object
 19  CODE_REJECT_REASON           1670214 non-null  object
 20  NAME_TYPE_SUITE              849809 non-null   object
 21  NAME_CLIENT_TYPE             1670214 non-null  object
 22  NAME_GOODS_CATEGORY          1670214 non-null  object
 23  NAME_PORTFOLIO               1670214 non-null  object
 24  NAME_PRODUCT_TYPE            1670214 non-null  object
```

Further, describe() was used for understanding the distribution and basic statistical properties of data.

In [7]:
```
df.describe()
```

Out[7]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	HOUR_APPR_PROCESS_STA
count	1.670214e+06	1.670214e+06	1.297979e+06	1.670214e+06	1.670213e+06	7.743700e+05	1.284699e+06	1.670214e+
mean	1.923089e+06	2.783572e+05	1.595512e+04	1.752339e+05	1.961140e+05	6.697402e+03	2.278473e+05	1.248418e+
std	5.325980e+05	1.028148e+05	1.478214e+04	2.927798e+05	3.185746e+05	2.092150e+04	3.153966e+05	3.334028e+
min	1.000001e+06	1.000010e+05	0.000000e+00	0.000000e+00	0.000000e+00	-9.000000e-01	0.000000e+00	0.000000e+
25%	1.461857e+06	1.893290e+05	6.321780e+03	1.872000e+04	2.416050e+04	0.000000e+00	5.084100e+04	1.000000e+
50%	1.923110e+06	2.787145e+05	1.125000e+04	7.104600e+04	8.054100e+04	1.638000e+03	1.123200e+05	1.200000e+
75%	2.384280e+06	3.675140e+05	2.065842e+04	1.803600e+05	2.164185e+05	7.740000e+03	2.340000e+05	1.500000e+
max	2.845382e+06	4.562550e+05	4.180581e+05	6.905160e+06	6.905160e+06	3.060045e+06	6.905160e+06	2.300000e+

click to scroll output; double click to hide

Identifying the missing data:

Calculating the percentages of missing values in all columns:

```
In [3]: df.isnull().sum()/len(df)*100

Out[3]: SK_ID_PREV                      0.000000
        SK_ID_CURR                      0.000000
        NAME_CONTRACT_TYPE              0.000000
        AMT_ANNUITY                    22.286665
        AMT_APPLICATION                 0.000000
        AMT_CREDIT                      0.000060
        AMT_DOWN_PAYMENT               53.636480
        AMT_GOODS_PRICE                23.081773
        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              53.636480
        RATE_INTEREST_PRIMARY          99.643698
        RATE_INTEREST_PRIVILEGED       99.643698
        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                49.119754
        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                    22.286366
        NAME_YIELD_GROUP                0.000000
        PRODUCT_COMBINATION             0.020716
        DAYS_FIRST_DRAWING             40.298129
        DAYS_FIRST_DUE                 40.298129
        DAYS_LAST_DUE_1ST_VERSION      40.298129
```

After the checking the percentages of missing values, we will check for the columns where the missing percentage is higher than 40% and then the unnecessary columns will be dropped.

```
In [5]: null_percentages=df.isnull().mean()*100
        cols_to_drop=null_percentages[null_percentages>40].index
        cols_to_drop

Out[5]: Index(['AMT_DOWN_PAYMENT', 'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
               'RATE_INTEREST_PRIVILEGED', 'NAME_TYPE_SUITE', 'DAYS_FIRST_DRAWING',
               'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION', 'DAYS_LAST_DUE',
               'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROVAL'],
              dtype='object')

In [6]: len(cols_to_drop)

Out[6]: 11
```

Here the data frame shows that there are total 11 columns whose null percentage is higher than 40.

Further we will drop these columns.

```
In [7]: df=df.drop(cols_to_drop, axis=1)

In [8]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 26 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   SK_ID_PREV                   1670214 non-null  int64
 1   SK_ID_CURR                   1670214 non-null  int64
 2   NAME_CONTRACT_TYPE           1670214 non-null  object
 3   AMT_ANNUITY                  1297979 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_GOODS_PRICE              1284699 non-null  float64
 7   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object
 8   HOUR_APPR_PROCESS_START      1670214 non-null  int64
 9   FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object
 10  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64
 11  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object
 12  NAME_CONTRACT_STATUS         1670214 non-null  object
 13  DAYS_DECISION                1670214 non-null  int64
 14  NAME_PAYMENT_TYPE            1670214 non-null  object
 15  CODE_REJECT_REASON           1670214 non-null  object
 16  NAME_CLIENT_TYPE             1670214 non-null  object
 17  NAME_GOODS_CATEGORY          1670214 non-null  object
 18  NAME_PORTFOLIO               1670214 non-null  object
 19  NAME_PRODUCT_TYPE            1670214 non-null  object
 20  CHANNEL_TYPE                 1670214 non-null  object
 21  SELLERPLACE_AREA             1670214 non-null  int64
 22  NAME_SELLER_INDUSTRY         1670214 non-null  object
 23  CNT_PAYMENT                  1297984 non-null  float64
 24  NAME_YIELD_GROUP             1670214 non-null  object
 25  PRODUCT_COMBINATION          1669868 non-null  object
dtypes: float64(5), int64(6), object(15)
memory usage: 331.3+ MB
```

After cleaning the missing data from the 1st data frame we will now load the 2nd data frame

```
In [10]: df1=pd.read_csv('e:/trainity/project 6/application_data.csv')

In [11]: df1.head()

Out[11]:
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREI
0	100002	1	Cash loans	M	N	Y	0	202500.0	40659
1	100003	0	Cash loans	F	N	N	0	270000.0	129350
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	13500
3	100006	0	Cash loans	F	N	Y	0	135000.0	31268
4	100007	0	Cash loans	M	N	Y	0	121500.0	51300

5 rows × 122 columns

Further info() and describe() were used to get more details about the data frame.

In [15]:
```
df1.info(verbose=True, show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 122 columns):
 #    Column                        Non-Null Count   Dtype
---   ------                        --------------   -----
 0    SK_ID_CURR                    307511 non-null  int64
 1    TARGET                        307511 non-null  int64
 2    NAME_CONTRACT_TYPE            307511 non-null  object
 3    CODE_GENDER                   307511 non-null  object
 4    FLAG_OWN_CAR                  307511 non-null  object
 5    FLAG_OWN_REALTY               307511 non-null  object
 6    CNT_CHILDREN                  307511 non-null  int64
 7    AMT_INCOME_TOTAL              307511 non-null  float64
 8    AMT_CREDIT                    307511 non-null  float64
 9    AMT_ANNUITY                   307499 non-null  float64
 10   AMT_GOODS_PRICE               307233 non-null  float64
 11   NAME_TYPE_SUITE               306219 non-null  object
 12   NAME_INCOME_TYPE              307511 non-null  object
 13   NAME_EDUCATION_TYPE           307511 non-null  object
```

In [16]:
```
df1.describe()
```

Out[16]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIV
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000	3.072330e+05	307511.0000
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573909	5.383962e+05	0.0208
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737315	3.694465e+05	0.0138
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e+04	0.0002
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05	0.0100
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05	0.0188
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	0.0286
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	0.0725

8 rows × 106 columns

Checking null percentages of first 50 columns:

```
In [17]: df1.iloc[:,0:50].isnull().mean()*100

Out[17]: SK_ID_CURR                        0.000000
         TARGET                            0.000000
         NAME_CONTRACT_TYPE                0.000000
         CODE_GENDER                       0.000000
         FLAG_OWN_CAR                      0.000000
         FLAG_OWN_REALTY                   0.000000
         CNT_CHILDREN                      0.000000
         AMT_INCOME_TOTAL                  0.000000
         AMT_CREDIT                        0.000000
         AMT_ANNUITY                       0.003902
         AMT_GOODS_PRICE                   0.090403
         NAME_TYPE_SUITE                   0.420148
         NAME_INCOME_TYPE                  0.000000
         NAME_EDUCATION_TYPE               0.000000
         NAME_FAMILY_STATUS                0.000000
         NAME_HOUSING_TYPE                 0.000000
         REGION_POPULATION_RELATIVE        0.000000
         DAYS_BIRTH                        0.000000
         DAYS_EMPLOYED                     0.000000
         DAYS_REGISTRATION                 0.000000
         DAYS_ID_PUBLISH                   0.000000
         OWN_CAR_AGE                      65.990810
         FLAG_MOBIL                        0.000000
         FLAG_EMP_PHONE                    0.000000
         FLAG_WORK_PHONE                   0.000000
         FLAG_CONT_MOBILE                  0.000000
         FLAG_PHONE                        0.000000
         FLAG_EMAIL                        0.000000
         OCCUPATION_TYPE                  31.345545
         CNT_FAM_MEMBERS                   0.000650
         REGION_RATING_CLIENT              0.000000
         REGION_RATING_CLIENT_W_CITY       0.000000
         WEEKDAY_APPR_PROCESS_START        0.000000
```

Checking null percentages for next 50 columns:

```
In [18]: df1.iloc[:,50:100].isnull().mean()*100

Out[18]: 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
```

Checking null percentages of the remaining columns:

```
In [19]: df1.iloc[:,100:].isnull().mean()*100

Out[19]: FLAG_DOCUMENT_6                 0.000000
         FLAG_DOCUMENT_7                 0.000000
         FLAG_DOCUMENT_8                 0.000000
         FLAG_DOCUMENT_9                 0.000000
         FLAG_DOCUMENT_10                0.000000
         FLAG_DOCUMENT_11                0.000000
         FLAG_DOCUMENT_12                0.000000
         FLAG_DOCUMENT_13                0.000000
         FLAG_DOCUMENT_14                0.000000
         FLAG_DOCUMENT_15                0.000000
         FLAG_DOCUMENT_16                0.000000
         FLAG_DOCUMENT_17                0.000000
         FLAG_DOCUMENT_18                0.000000
         FLAG_DOCUMENT_19                0.000000
         FLAG_DOCUMENT_20                0.000000
         FLAG_DOCUMENT_21                0.000000
         AMT_REQ_CREDIT_BUREAU_HOUR     13.501631
         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
         dtype: float64
```

Checking the data frame after dropping the columns where null percentage was greater then 30:

```
In [22]: df1=df1.drop(col_to_drop, axis=1)

In [23]: df1.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 72 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   SK_ID_CURR                   307511 non-null  int64
 1   TARGET                       307511 non-null  int64
 2   NAME_CONTRACT_TYPE           307511 non-null  object
 3   CODE_GENDER                  307511 non-null  object
 4   FLAG_OWN_CAR                 307511 non-null  object
 5   FLAG_OWN_REALTY              307511 non-null  object
 6   CNT_CHILDREN                 307511 non-null  int64
 7   AMT_INCOME_TOTAL             307511 non-null  float64
 8   AMT_CREDIT                   307511 non-null  float64
 9   AMT_ANNUITY                  307499 non-null  float64
 10  AMT_GOODS_PRICE              307233 non-null  float64
 11  NAME_TYPE_SUITE              306219 non-null  object
 12  NAME_INCOME_TYPE             307511 non-null  object
 13  NAME_EDUCATION_TYPE          307511 non-null  object
 14  NAME_FAMILY_STATUS           307511 non-null  object
 15  NAME_HOUSING_TYPE            307511 non-null  object
 16  REGION_POPULATION_RELATIVE   307511 non-null  float64
 17  DAYS_BIRTH                   307511 non-null  int64
 18  DAYS_EMPLOYED                307511 non-null  int64
 19  DAYS_REGISTRATION            307511 non-null  float64
 20  DAYS_ID_PUBLISH              307511 non-null  int64
 21  FLAG_MOBIL                   307511 non-null  int64
 22  FLAG_EMP_PHONE               307511 non-null  int64
 23  FLAG_WORK_PHONE              307511 non-null  int64
 24  FLAG_CONT_MOBILE             307511 non-null  int64
 25  FLAG_PHONE                   307511 non-null  int64
```

Removing non relevant columns:

```
In [24]: nonrelevant=['FLAG_MOBIL','FLAG_EMP_PHONE','FLAG_WORK_PHONE','FLAG_CONT_MOBILE','FLAG_PHONE','FLAG_EMAIL','REGION_RATING_CLIENT'

In [25]: 1 len(nonrelevant)
Out[25]: 30

In [26]: df1=df1.drop(nonrelevant, axis=1)
         df1.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 42 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   SK_ID_CURR                  307511 non-null  int64
 1   TARGET                      307511 non-null  int64
 2   NAME_CONTRACT_TYPE          307511 non-null  object
 3   CODE_GENDER                 307511 non-null  object
 4   FLAG_OWN_CAR                307511 non-null  object
 5   FLAG_OWN_REALTY             307511 non-null  object
 6   CNT_CHILDREN                307511 non-null  int64
 7   AMT_INCOME_TOTAL            307511 non-null  float64
 8   AMT_CREDIT                  307511 non-null  float64
 9   AMT_ANNUITY                 307499 non-null  float64
 10  AMT_GOODS_PRICE             307233 non-null  float64
 11  NAME_TYPE_SUITE             306219 non-null  object
 12  NAME_INCOME_TYPE            307511 non-null  object
 13  NAME_EDUCATION_TYPE         307511 non-null  object
 14  NAME_FAMILY_STATUS          307511 non-null  object
 15  NAME_HOUSING_TYPE           307511 non-null  object
 16  REGION_POPULATION_RELATIVE  307511 non-null  float64
```

Checking the gender column for null values if any and then replacing the null values with the mode of the gender column and checking the annuity column and filling the null values with median of the annuity column:

```
In [24]: df1.CODE_GENDER.value_counts()
Out[24]: F      202448
         M      105059
         XNA         4
         Name: CODE_GENDER, dtype: int64

In [27]: mode_gender = df1[df1['CODE_GENDER'] != 'XNA']['CODE_GENDER'].mode()[0]
         df1.loc[df1['CODE_GENDER'] == 'XNA', 'CODE_GENDER'] = mode_gender

In [28]: df1.CODE_GENDER.value_counts()
Out[28]: F    202452
         M    105059
         Name: CODE_GENDER, dtype: int64

In [30]: df1.AMT_ANNUITY.isnull().sum()
Out[30]: 12

In [31]: df1.AMT_ANNUITY=df1.AMT_ANNUITY.fillna(df1['AMT_ANNUITY'].median())

In [32]: df1.AMT_ANNUITY.isnull().sum()
Out[32]: 0
```

On checking the values from contract type, income type and education type column, it was found that that the education type column had a value as Secondary / secondary special. This forward slash(/) is often used as a division operator which means that it can cause errors or incorrect results. So this / was replaced with 'or' using the lambda function.

```
In [35]: df1.NAME_EDUCATION_TYPE.value_counts()

Out[35]: Secondary / secondary special     218391
         Higher education                   74863
         Incomplete higher                  10277
         Lower secondary                     3816
         Academic degree                      164
         Name: NAME_EDUCATION_TYPE, dtype: int64

In [102]: df1.NAME_EDUCATION_TYPE=df1.NAME_EDUCATION_TYPE.apply(lambda x: x.replace('/','or'))
          df1.NAME_EDUCATION_TYPE.value_counts()

Out[102]: Secondary or secondary special    218391
          Higher education                   74863
          Incomplete higher                  10277
          Lower secondary                     3816
          Academic degree                      164
          Name: NAME_EDUCATION_TYPE, dtype: int64

In [103]: df1.NAME_FAMILY_STATUS.value_counts()

Out[103]: Married                 196432
          Single / not married     45444
          Civil marriage           29775
          Separated                19770
          Widow                    16088
          Unknown                      2
          Name: NAME_FAMILY_STATUS, dtype: int64

In [104]: df1.NAME_FAMILY_STATUS=df1.NAME_FAMILY_STATUS.apply(lambda x: x.replace('/','or'))
          df1.NAME_FAMILY_STATUS.value_counts()

Out[104]: Married                  196432
          Single or not married     45444
          Civil marriage            29775
          Separated                 19770
          Widow                     16088
```

Later on while checking the values from organization type column, there was a value as XNA. This value was deleted as it was of no use.

```
In [43]: df1=df1.loc[df1['ORGANIZATION_TYPE']!='XNA']

In [44]: df1.ORGANIZATION_TYPE.value_counts()

Out[44]: Business Entity Type 3    67992
         Self-employed             38412
         Other                     16683
         Medicine                  11193
         Business Entity Type 2    10553
         Government                10404
         School                     8893
         Trade: type 7              7831
         Kindergarten               6880
         Construction               6721
         Business Entity Type 1     5984
         Transport: type 4          5398
         Trade: type 3              3492
         Industry: type 9           3368
         Industry: type 3           3278
         Security                   3247
         Housing                    2958
         Industry: type 11          2704
         Military                   2634
         Bank                       2507
         Agriculture                2454
         Police                     2341
         Transport: type 2          2204
         Postal                     2157
         Security Ministries        1974
         Trade: type 2              1900
         Restaurant                 1811
         Services                   1575
         University                 1327
         Industry: type 7           1307
         Transport: type 3          1187
```

While checking the AMT_INCOME_TOTAL and the AMT_CREDIT column, it was found that there were different values which may cause a problem while plotting a chart. So a new columns named 'Income_range' and 'Credit_range' were created where the incomes and credits were grouped in a range respectively making it easy while plotting charts.

```
In [45]: df1.AMT_INCOME_TOTAL.value_counts()

Out[45]: 135000.0    30206
         112500.0    25161
         157500.0    22734
         180000.0    21805
         225000.0    18460
                     ...
         117324.0        1
         64584.0         1
         142897.5        1
         109170.0        1
         440100.0        1
         Name: AMT_INCOME_TOTAL, Length: 2266, dtype: int64
```

```
In [112]: i_limits=[0,25000,50000,75000,100000,125000,150000,175000,200000,225000,250000,275000,300000,325000,350000,375000,400000,425000,4
          i_labels = ['0-25000','25000-50000','50000-75000','75000,100000','100000-125000', '125000-150000', '150000-175000','175000-20000
          df1['Income_range']=pd.cut(df1['AMT_INCOME_TOTAL'],bins=i_limits,labels=i_labels)
```

```
In [113]: df1.Income_range.value_counts()

Out[113]: 125000-150000      39819
          200000-225000      35994
          100000-125000      34851
          75000,100000       29612
          150000-175000      29296
          175000-200000      25892
          50000-75000        12077
          250000-275000      11485
          225000-250000       6483
          300000-325000       6169
          350000-375000       4185
          275000-300000       3749
          425000-450000       2863
          500000 and above    2543
          25000-50000         1982
```

```
In [48]: df1.AMT_CREDIT.value_counts()

Out[48]: 450000.0     8764
         675000.0     7109
         180000.0     6850
         270000.0     6600
         225000.0     6394
                      ...
         1006888.5       1
         1689736.5       1
         296671.5        1
         495486.0        1
         743863.5        1
         Name: AMT_CREDIT, Length: 5331, dtype: int64
```

```
In [49]: c_limits = [0,150000,200000,250000,300000,350000,400000,450000,500000,550000,600000,650000,700000,750000,800000,850000,900000,10
         c_labels = ['0-150000', '150000-200000','200000-250000', '250000-300000', '300000-350000', '350000-400000','400000-450000','4500
         df1['Credit_range'] = pd.cut(df1.AMT_CREDIT,bins=c_limits,labels=c_labels)
```

```
In [50]: df1.Credit_range.value_counts()

Out[50]: 900000 and above    50717
         250000-300000       24961
         500000-550000       18434
         200000-250000       17502
         400000-450000       15854
         150000-200000       14867
         0-150000            13502
         300000-350000       13487
         650000-700000       12142
         450000-500000       11254
         750000-800000        9770
         550000-600000        9489
         800000-850000        9439
         850000-900000        9100
```

Further while checking it was found that the columns DAYS_BIRTH, DAYS_EMPLOYED, DAYS_ID_PUBLISH and DAYS_REGISTRATION had negative values. These values were converted into positive values using the np.abs function.

```
In [59]: df1.DAYS_REGISTRATION.value_counts()

Out[59]: -1.0        91
         -6.0        84
         -2.0        82
         -7.0        81
         -4.0        78
                     ..
         -15854.0     1
         -15524.0     1
         -13715.0     1
         -14494.0     1
         -12372.0     1
         Name: DAYS_REGISTRATION, Length: 14419, dtype: int64

In [60]: df1.DAYS_REGISTRATION=np.abs(df1.DAYS_REGISTRATION)
         df1.DAYS_REGISTRATION.value_counts()

Out[60]: 1.0         91
         6.0         84
         2.0         82
         7.0         81
         4.0         78
                     ..
         15854.0      1
         15524.0      1
         13715.0      1
         14494.0      1
         12372.0      1
         Name: DAYS_REGISTRATION, Length: 14419, dtype: int64
```

DAYS_BIRTH and DAYS_EMPLOYED columns had values in days instead of year. So they were converted to year by using the div function. This would make it easy for us while comparing variables during further analysis.

```
In [61]: df1.DAYS_BIRTH=df1.DAYS_BIRTH.div(365)
         df1.DAYS_BIRTH.value_counts()

Out[61]: 37.668493    43
         36.934247    42
         27.452055    41
         49.994521    41
         28.197260    40
                      ..
         67.443836     1
         61.126027     1
         64.309589     1
         67.843836     1
         66.627397     1
         Name: DAYS_BIRTH, Length: 16513, dtype: int64

In [62]: df1.DAYS_EMPLOYED=df1.DAYS_EMPLOYED.div(365)
         df1.DAYS_EMPLOYED.value_counts()

Out[62]: 0.547945     156
         0.613699     152
         0.545205     151
         0.630137     151
         0.580822     150
                     ...
         38.249315      1
         32.402740      1
         27.879452      1
         25.915068      1
         23.819178      1
         Name: DAYS_EMPLOYED, Length: 12573, dtype: int64
```

Later on while checking the columns it was found that they weren't in numerical form, so their data type were converted to numeric using the 'apply(pd.to_numeric)' function.

Numeric data allows for mathematical operations such as addition, subtraction, multiplication, and division, which are necessary for many data analysis tasks. Numeric data can also be easily plotted, and visualized with graphs and charts. While non-numeric data such as text or categorical data can be informative.

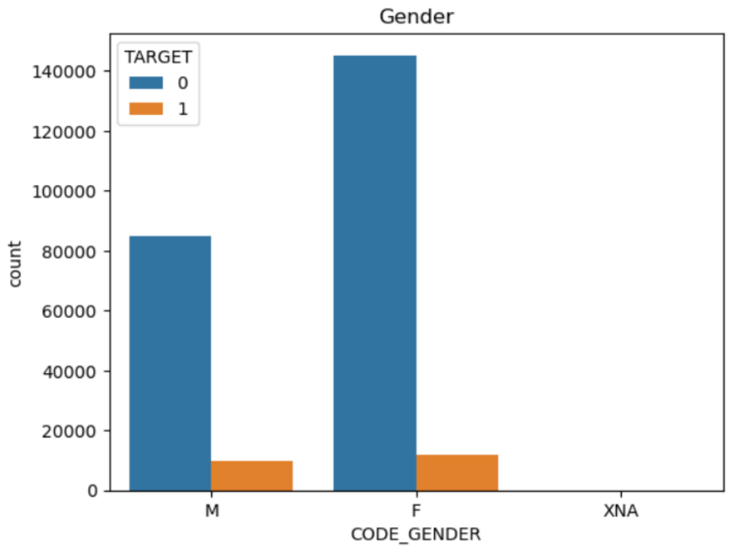
UNIVARIATE ANALYSIS

Univariate analysis is a statistical method that focuses on the analysis of a single variable at a time. Univariate analysis is commonly used to provide insight into the characteristics of a single variable, including the distribution of the variable, the presence of outliers, and the shape of the data.

Countplot and boxplot was used for univariate analysis of CODE_GENDER, Income_range, FLAG_OWN_CAR, NAME_EDUCATION_TYPE, FLAG_OWN_REALTY, ORGANIZATION_TYPE, Credit_range, NAME_INCOME_TYPE, NAME_CONTRACT_TYPE, AMT_CREDIT, AMT_INCOME_TOTAL, AMT_ANNUITY, DAYS_EMPLOYED, DAYS_REGISTRATION and CNT_CHILDREN.

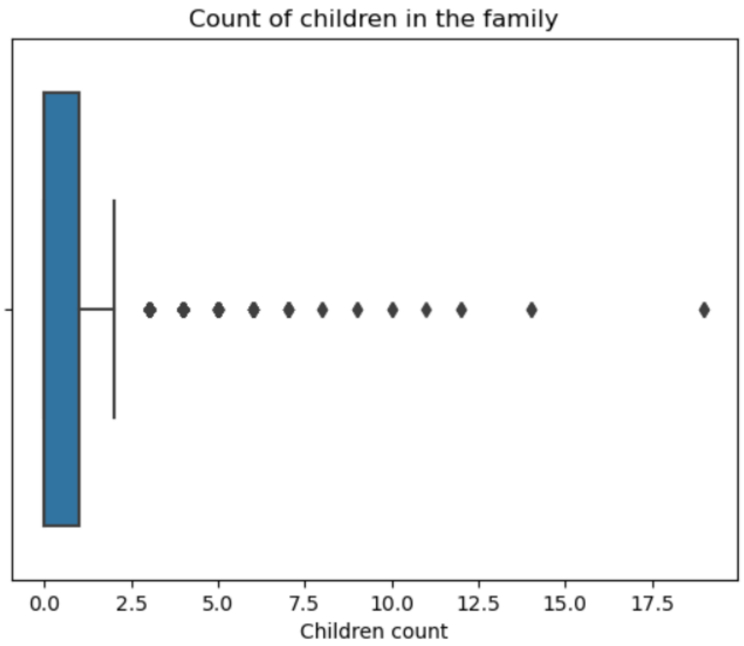
Following fig. shows univariate analysis for CODE_GENDER:

```
In [64]: #univariate analysis
         sns.countplot(x='CODE_GENDER', hue='TARGET', data=df1)
         plt.title('Gender')
         plt.show()
```



While plotting the boxplot for CNT_CHILDREN, it was found that it had outliers as shown below:

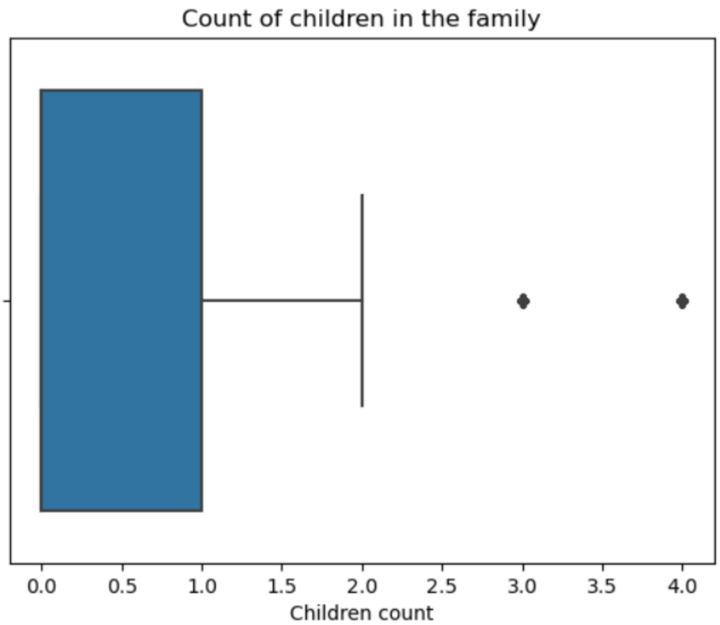
```
In [84]: sns.boxplot(x='CNT_CHILDREN', data=df1)
         plt.xlabel('Children count')
         plt.title('Count of children in the family')

Out[84]: Text(0.5, 1.0, 'Count of children in the family')
```



These outliers were then removed as shown below:

```
In [85]: df1=df1[~(df1.CNT_CHILDREN>=5)]
```

```
In [86]: sns.boxplot(x='CNT_CHILDREN', data=df1)
         plt.xlabel('Children count')
         plt.title('Count of children in the family')

Out[86]: Text(0.5, 1.0, 'Count of children in the family')
```



Later the data frame was divided into two data frames namely tg1 and tg0, where tg1 is client with payment difficulties and tg0 is all other.

```
In [87]: #dividing the data frame into two, tg1=client with payment difficulties and tg0=all other
         tg0=df1.loc[df1['TARGET']==0]
         tg1=df1.loc[df1['TARGET']==1]

In [88]: tg0.shape

Out[88]: (230197, 44)

In [89]: tg1.shape

Out[89]: (21820, 44)
```

BIVARIATE ANALYSIS

Bivariate analysis is a statistical method that involves the analysis of the relationship between two variables. It aims to understand how changes in one variable affect changes in another variable. In bivariate analysis, the variables are usually measured on a continuous or categorical scale.

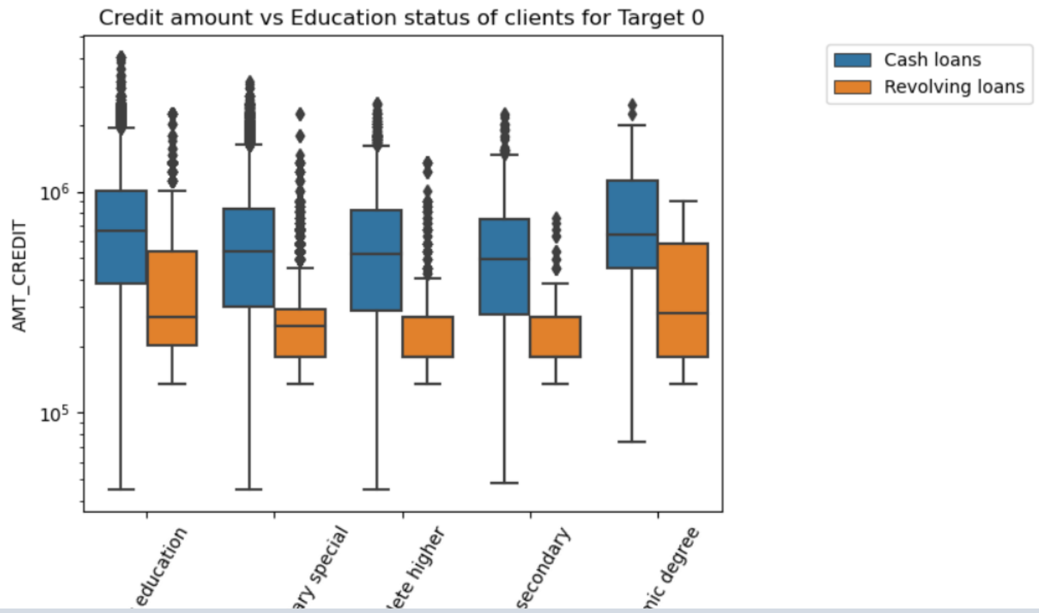
The primary goal of bivariate analysis is to explore the relationship between two variables and determine the strength and direction of the relationship.

Boxplot was used to calculate bivariate analysis. The following variables were used:

- Credit amount vs education status of client for target 0 and target 1
- Credit amount vs family status of clients for target 0 and target 1
- Income amount vs education status of clients for target 0 and target 1
- Total income status vs family status for target 0 and target 1

Following fig. shows bivariate analysis for credit amount vs education status of client for target 0

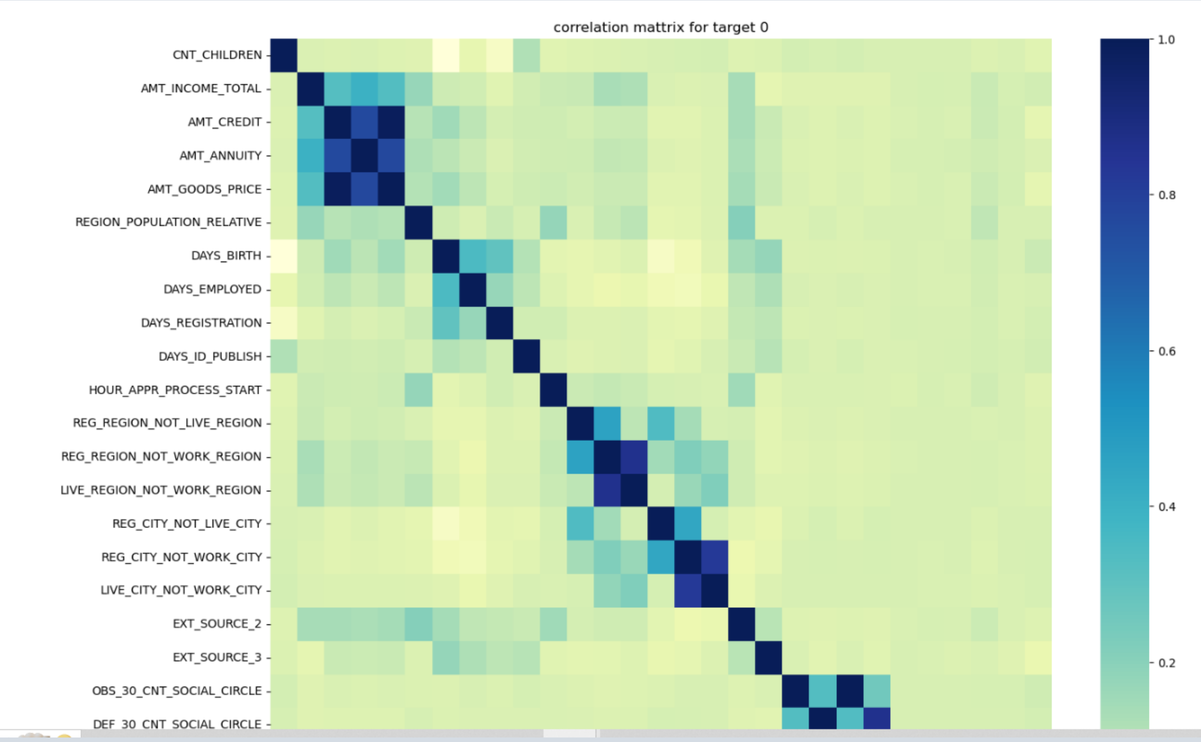
```
In [90]: # bivariate analysis for target 0
         sns.boxplot(x=tg0.NAME_EDUCATION_TYPE, y=tg0.AMT_CREDIT, hue=tg0.NAME_CONTRACT_TYPE, data=tg0)
         plt.title('Credit amount vs Education status of clients for Target 0')
         plt.xticks(rotation=60)
         plt.legend(loc='upper right', bbox_to_anchor=(1.5,1.0))
         plt.yscale('log')
         plt.show()
```



MULTIVARIATE ANALYSIS

Multivariate analysis is a statistical method that involves the analysis of more than two variables simultaneously to determine the relationship between them. It aims to understand how multiple variables affect each other and identify patterns and relationships within a dataset. The primary goal of multivariate analysis is to examine the relationship between multiple variables and determine their collective influence on an outcome.

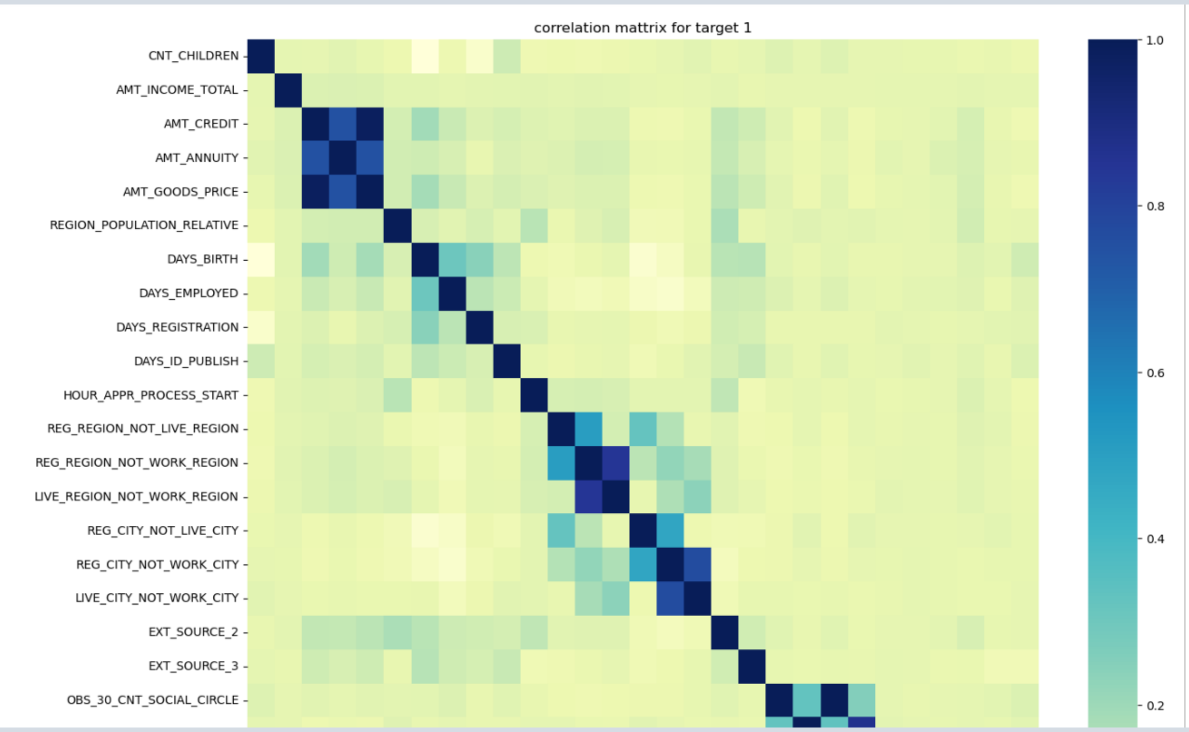
Heatmap was used to show correlation between variables. The following fig. shows the correlation for all variables for target 0:



Top 10 correlation for the client with all other:

```
In [100]: top10_other = corr_tg0.unstack().sort_values(kind='quicksort').drop_duplicates().nlargest(10)

In [101]: top10_other

Out[101]: CNT_CHILDREN                CNT_CHILDREN                1.000000
          OBS_30_CNT_SOCIAL_CIRCLE    OBS_60_CNT_SOCIAL_CIRCLE    0.998491
          AMT_GOODS_PRICE             AMT_CREDIT                  0.986727
          DEF_30_CNT_SOCIAL_CIRCLE    DEF_60_CNT_SOCIAL_CIRCLE    0.861431
          REG_REGION_NOT_WORK_REGION  LIVE_REGION_NOT_WORK_REGION 0.860428
          REG_CITY_NOT_WORK_CITY      LIVE_CITY_NOT_WORK_CITY     0.820781
          AMT_GOODS_PRICE             AMT_ANNUITY                 0.766930
          AMT_CREDIT                  AMT_ANNUITY                 0.762100
          REG_REGION_NOT_WORK_REGION  REG_REGION_NOT_LIVE_REGION  0.461709
          REG_CITY_NOT_LIVE_CITY      REG_CITY_NOT_WORK_CITY      0.442738
          dtype: float64
```

The following fig. shows the correlation for all variables for target 1:



Top 10 correlation for the client with payment difficulties:

```
In [111]: top10_difficulties = corr_tg1.unstack().sort_values(kind='quicksort').drop_duplicates().nlargest(10)
          top10_difficulties

Out[111]: CNT_CHILDREN                 CNT_CHILDREN                   1.000000
          OBS_60_CNT_SOCIAL_CIRCLE     OBS_30_CNT_SOCIAL_CIRCLE       0.998287
          AMT_CREDIT                   AMT_GOODS_PRICE                0.982800
          DEF_60_CNT_SOCIAL_CIRCLE     DEF_30_CNT_SOCIAL_CIRCLE       0.867575
          REG_REGION_NOT_WORK_REGION   LIVE_REGION_NOT_WORK_REGION    0.846866
          REG_CITY_NOT_WORK_CITY       LIVE_CITY_NOT_WORK_CITY        0.768170
          AMT_GOODS_PRICE              AMT_ANNUITY                    0.749432
          AMT_ANNUITY                  AMT_CREDIT                     0.748769
          REG_REGION_NOT_LIVE_REGION   REG_REGION_NOT_WORK_REGION     0.506738
          REG_CITY_NOT_LIVE_CITY       REG_CITY_NOT_WORK_CITY         0.478381
          dtype: float64
```

After individual analysis, both the data frames were merged for further analysis as shown in the following fig.:

```
In [112]: #Merging the 2 data frames together
          merged_df=pd.merge(df,df1,on='SK_ID_CURR')
          merged_df.head()
```

Out[112]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE_x	AMT_ANNUITY_x	AMT_APPLICATION	AMT_CREDIT_x	AMT_GOODS_PRICE_x	WEEKDAY_APPR_PRO
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	17145.0	
1	1696966	271877	Consumer loans	68258.655	1800000.0	1754721.0	1800000.0	
2	2154916	271877	Consumer loans	12417.390	108400.5	119848.5	108400.5	
3	2802425	108129	Cash loans	25188.615	607500.0	679671.0	607500.0	
4	1536272	108129	Cash loans	21709.125	450000.0	512370.0	450000.0	

5 rows × 69 columns

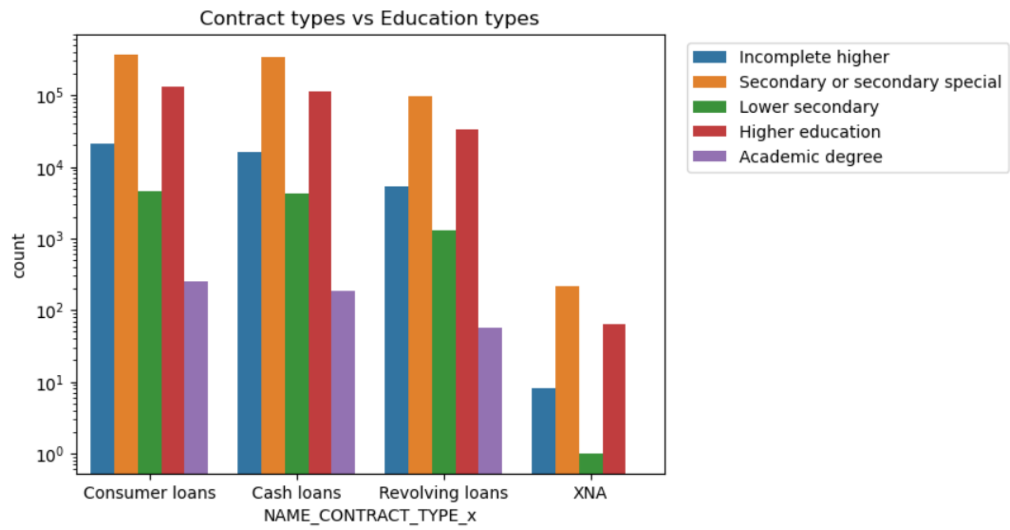
After merging the data frame, non-relevant columns were deleted and null percentages for columns were found as shown below:

```
In [116]: merged_df.isnull().mean()*100

Out[116]: SK_ID_PREV                     0.000000
          NAME_CONTRACT_TYPE_x           0.000000
          AMT_ANNUITY_x                 21.379311
          AMT_APPLICATION                0.000000
          AMT_CREDIT_x                   0.000000
          AMT_GOODS_PRICE_x             22.145566
          WEEKDAY_APPR_PROCESS_START_x   0.000000
          HOUR_APPR_PROCESS_START_x      0.000000
          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_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                   21.378960
          NAME_YIELD_GROUP               0.000000
          PRODUCT_COMBINATION            0.025711
          TARGET                         0.000000
          NAME_CONTRACT_TYPE_y           0.000000
          CODE_GENDER                    0.000000
          FLAG_OWN_CAR                   0.000000
          FLAG_OWN_REALTY                0.000000
          CNT_CHILDREN                   0.000000
          AMT_INCOME_TOTAL               0.000000
          AMT_CREDIT_y                   0.000000
```

After checking the null percentages, univariate analysis for NAME_CONTRACT_TYPE, PRODUCT_COMBINATION, NAME_CASH_LOAN_PURPOSE and NAME_INCOME_TYPE was performed.

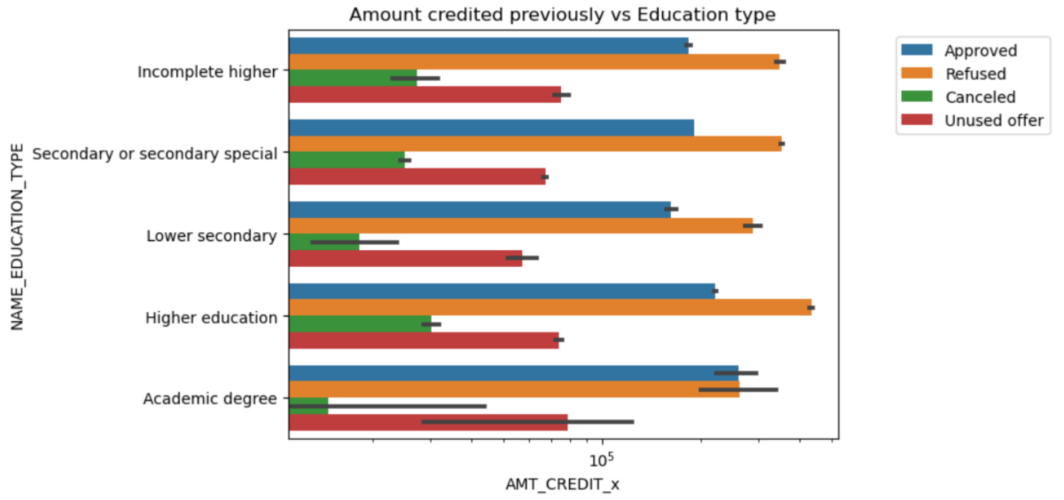
The univariate analysis for NAME_CONTRACT_TYPE is shown in the following fig.

```
In [117]: #Contract types vs education types
          sns.countplot(x=merged_df.NAME_CONTRACT_TYPE_x, hue=merged_df.NAME_EDUCATION_TYPE, data=merged_df)
          plt.title('Contract types vs Education types')
          plt.yscale('log')
          plt.legend(loc='upper right', bbox_to_anchor=(1.6,1.0))

Out[117]: <matplotlib.legend.Legend at 0x249375f8070>
```

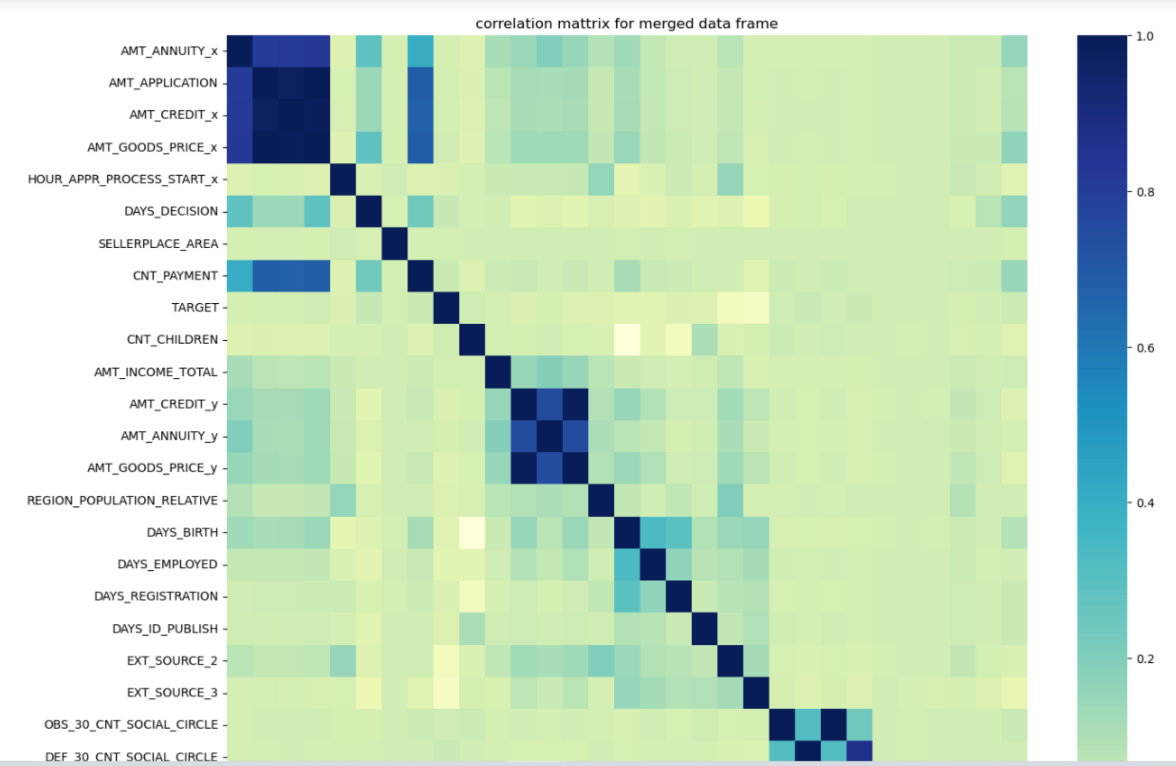


Bivariate analysis was performed on the merged data frame. Analysis for AMT_CREDIT_X and NAME_EDUCATION_TYPE is shown below:

```
In [124]: sns.barplot(x=merged_df.AMT_CREDIT_x, y=merged_df.NAME_EDUCATION_TYPE, hue=merged_df.NAME_CONTRACT_STATUS, data=merged_df)
          plt.xscale('log')
          plt.title('Amount credited previously vs Education type')
          plt.legend(loc='upper right', bbox_to_anchor=(1.4,1.0))
          plt.show()
```



After completion of the bivariate analysis, multivariate analysis was performed. The correlation is shown with the help of heatmap below:



Top 10 correlation for the merged data frame:

```
In [132]: top10=merged_df_corr.unstack().sort_values(kind='quicksort').drop_duplicates().nlargest(10)
          top10

Out[132]: AMT_ANNUITY_x            AMT_ANNUITY_x            1.000000
          AMT_APPLICATION          AMT_GOODS_PRICE_x        0.999849
          OBS_60_CNT_SOCIAL_CIRCLE OBS_30_CNT_SOCIAL_CIRCLE 0.998537
          AMT_CREDIT_x             AMT_GOODS_PRICE_x        0.992858
          AMT_CREDIT_y             AMT_GOODS_PRICE_y        0.985562
          AMT_CREDIT_x             AMT_APPLICATION          0.973441
          DEF_60_CNT_SOCIAL_CIRCLE DEF_30_CNT_SOCIAL_CIRCLE 0.862820
          AMT_GOODS_PRICE_x        AMT_ANNUITY_x            0.821127
          AMT_ANNUITY_x            AMT_CREDIT_x             0.818081
                                   AMT_APPLICATION          0.807590
          dtype: float64
```

INSIGHTS

After analysing the given data, I made following conclusions:

- Number of female applicants is higher than male applicants and number of females is more than males for not having payment difficulties. Maximum number of loans are approved for female clients.
- Number of applicants from the working type income category apply for loans the most.
- Most clients with the income range of 4.5 lc-4.75lc have difficulties while loan payment.
- Most clients with the income range of 1.2lc-1.5lc don't have difficulties while loan payment.
- Clients lying in the credit range 7lac-7.5lac are the least for having difficulties in payment.
- Clients lying in the 900000 and above are most capable of paying back loans.
- Trade : type 4, organization type have the least count of payment difficulties.
- Most cash loans applicants don't have payment difficulties.

- Applicants with no car are the most applying for loans.
- Family status of civil marriage of academic degree education have higher number of credits for target 0.
- Most clients from cash loans are from higher education of education status for target 0.
- Most number of all types of education and family lie in lower bound for target 1.
- Number of cash loans are maximum for contract status than revolving loans.
- Most cancelled loans are of product combination, cash and most refused loans are of product combination cash X-sell:low.
- Loans from purpose: repairs are most rejected.
- Students and pensioners have no unused offers so banks can consider providing more offers to them and are more likely to pay back.
- Housing type with parents or house or apartment or municipal apartment have successful payments.