Bank Loan Case Study

by

Subham Roy

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.
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.

The libraries for data analysis and visualization used in this project are Numpy & Pandas.

1. **Present** the overall approach of the analysis. Mention the problem statement and the analysis approach briefly

2. **Identify** the missing data and use appropriate method to deal with it. (Remove columns/or replace it with an appropriate value)
   Hint: *Note that in EDA, since it is not necessary to replace the missing value, but if you have to replace the missing value, what should be the approach. Clearly mention the approach.*

3. **Identify** if there are **outliers** in the dataset. Also, mention why do you think it is an outlier. Again, remember that for this exercise, it is not necessary to remove any data points.

4. **Identify** if there is data imbalance in the data. Find the ratio of data imbalance.
   Hint: *Since there are a lot of columns, you can run your analysis in loops for the appropriate columns and find the insights.*

5. Explain the **results of univariate, segmented univariate, bivariate analysis, etc.** in business terms.

6. **Find the top 10 correlation** for the Client with payment difficulties and all other cases (Target variable). Note that you have to find the top correlation by segmenting the data frame w.r.t to the target variable and then find the top correlation for each of the segmented data and find if any insight is there. Say, there are 5+1(target) variables in a dataset: Var1, Var2, Var3, Var4, Var5, Target. And if you have to find top 3 correlation, it can be: Var1 & Var2, Var2 & Var3, Var1 & Var3. Target variable will not feature in this correlation as it is a categorical variable and not a continuous variable which is increasing or decreasing.

7. **Include visualizations** and **summarize** the most important results in the presentation. You are free to choose the graphs which explain the numerical/categorical variables. Insights should explain why the variable is important for differentiating the clients with payment difficulties with all other cases.

Approach and Tech Used

For this project I used Jupyter Notebook (Anaconda) to run my queries and charts.

The notebook extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results. The Jupyter notebook combines two components:

A web application: a browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output.

Notebook documents: a representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media representations of objects.

This project helped me in understanding the tables at a much-detailed manner and helped to improve my strength in extracting data from tables in a more efficient manner.

Datasets

First, we imported all the libraries needed:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Next, we read the dataset files given to us:

Dataset 1 - "application_data.csv"

df_application = pd.read_csv('application_data.csv')
df_application.head()

Output:

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

Dataset 2 - "previous_application.csv"

df_previous_application = pd.read_csv('previous_application.csv')
df_previous_application.head()

Output:

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

Cleaning the data

Dataset 1 - "application_data.csv"

We find out the number of null values in the dataset:

First, we find out the null values in dataframe:

df_application.isnull().sum()

Output:

```
SK_ID_CURR                        0
TARGET                            0
NAME_CONTRACT_TYPE                0
CODE_GENDER                       0
FLAG_OWN_CAR                      0
FLAG_OWN_REALTY                   0
CNT_CHILDREN                      0
AMT_INCOME_TOTAL                  0
AMT_CREDIT                        0
AMT_ANNUITY                      12
AMT_GOODS_PRICE                 278
NAME_TYPE_SUITE                1292
NAME_INCOME_TYPE                  0
NAME_EDUCATION_TYPE               0
NAME_FAMILY_STATUS                0
NAME_HOUSING_TYPE                 0
REGION_POPULATION_RELATIVE        0
DAYS_BIRTH                        0
DAYS_EMPLOYED                     0
DAYS_REGISTRATION                 0
DAYS_ID_PUBLISH                   0
OWN_CAR_AGE                  202929
FLAG_MOBIL                        0
FLAG_EMP_PHONE                    0
FLAG_WORK_PHONE                   0
FLAG_CONT_MOBILE                  0
FLAG_PHONE                        0
FLAG_EMAIL                        0
OCCUPATION_TYPE               96391
CNT_FAM_MEMBERS                   2
```

Checking Percentage of Null Value's in dataframe usning function

def Missing_Values(dataframe):

*return round((dataframe.isnull().sum()*100/len(dataframe)).sort_values(ascending = False),3)*

Missing_Values(df_application)

Output:

```
COMMONAREA_MEDI             69.872
COMMONAREA_AVG              69.872
COMMONAREA_MODE             69.872
NONLIVINGAPARTMENTS_MODE    69.433
NONLIVINGAPARTMENTS_AVG     69.433
NONLIVINGAPARTMENTS_MEDI    69.433
FONDKAPREMONT_MODE          68.386
LIVINGAPARTMENTS_MODE       68.355
LIVINGAPARTMENTS_AVG        68.355
LIVINGAPARTMENTS_MEDI       68.355
FLOORSMIN_AVG               67.849
FLOORSMIN_MODE              67.849
FLOORSMIN_MEDI              67.849
YEARS_BUILD_MEDI            66.498
YEARS_BUILD_MODE            66.498
YEARS_BUILD_AVG             66.498
OWN_CAR_AGE                 65.991
LANDAREA_MEDI               59.377
LANDAREA_MODE               59.377
LANDAREA_AVG                59.377
BASEMENTAREA_MEDI           58.516
```

Then I stored it in a variable which contains more than 50% missing values:

Null=Missing_Values(df_application)[Missing_Values(df_application) > 50]

Print(Null)

Output:

COMMONAREA_MEDI	69.872
COMMONAREA_AVG	69.872
COMMONAREA_MODE	69.872
NONLIVINGAPARTMENTS_MODE	69.433
NONLIVINGAPARTMENTS_AVG	69.433
NONLIVINGAPARTMENTS_MEDI	69.433
FONDKAPREMONT_MODE	68.386
LIVINGAPARTMENTS_MODE	68.355
LIVINGAPARTMENTS_AVG	68.355
LIVINGAPARTMENTS_MEDI	68.355
FLOORSMIN_AVG	67.849
FLOORSMIN_MODE	67.849
FLOORSMIN_MEDI	67.849
YEARS_BUILD_MEDI	66.498
YEARS_BUILD_MODE	66.498
YEARS_BUILD_AVG	66.498
OWN_CAR_AGE	65.991
LANDAREA_MEDI	59.377
LANDAREA_MODE	59.377
LANDAREA_AVG	59.377
BASEMENTAREA_MEDI	58.516
BASEMENTAREA_AVG	58.516
BASEMENTAREA_MODE	58.516
EXT_SOURCE_1	56.381
NONLIVINGAREA_MODE	55.179
NONLIVINGAREA_AVG	55.179
NONLIVINGAREA_MEDI	55.179
ELEVATORS_MEDI	53.296
ELEVATORS_AVG	53.296
ELEVATORS_MODE	53.296

Dropping all those values since these have lots of missing data and it will disrupt the data:

Null.index

Output:

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

Droping all the colums having missing values >50%:

df_application.drop(columns=Null.index ,inplace=True)
df_application.head()

Output:

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

Checking the missing values above 40%:

Missing_Values(df_application)[Missing_Values(df_application)>40]

Output:

```
FLOORSMAX_AVG                    49.761
FLOORSMAX_MODE                   49.761
FLOORSMAX_MEDI                   49.761
YEARS_BEGINEXPLUATATION_AVG      48.781
YEARS_BEGINEXPLUATATION_MODE     48.781
YEARS_BEGINEXPLUATATION_MEDI     48.781
TOTALAREA_MODE                   48.269
EMERGENCYSTATE_MODE              47.398
dtype: float64
```

Checking the percentage of missing values:

Null_1 = Missing_Values(df_application)[Missing_Values(df_application)>40]

Null_1

Output:

```
FLOORSMAX_AVG                   49.761
FLOORSMAX_MODE                  49.761
FLOORSMAX_MEDI                  49.761
YEARS_BEGINEXPLUATATION_AVG     48.781
YEARS_BEGINEXPLUATATION_MODE    48.781
YEARS_BEGINEXPLUATATION_MEDI    48.781
TOTALAREA_MODE                  48.269
EMERGENCYSTATE_MODE             47.398
dtype: float64
```

Dropping the unnecessary columns from the datafame as they were not important:

df_application.drop(columns=Null_1.index ,inplace=True) print('Null_1 dropped from the df_application')

Output:

```
Null_1 dropped from the df_application
```

Checking the missing values in dataframe:

df_application.isnull().sum()

Output:

```
SK_ID_CURR                      0
TARGET                          0
NAME_CONTRACT_TYPE              0
CODE_GENDER                     0
FLAG_OWN_CAR                    0
FLAG_OWN_REALTY                 0
CNT_CHILDREN                    0
AMT_INCOME_TOTAL                0
AMT_CREDIT                      0
AMT_ANNUITY                    12
AMT_GOODS_PRICE               278
NAME_TYPE_SUITE              1292
NAME_INCOME_TYPE                0
NAME_EDUCATION_TYPE             0
NAME_FAMILY_STATUS              0
NAME_HOUSING_TYPE               0
REGION_POPULATION_RELATIVE      0
DAYS_BIRTH                      0
DAYS_EMPLOYED                   0
DAYS_REGISTRATION               0
DAYS_ID_PUBLISH                 0
FLAG_MOBIL                      0
FLAG_EMP_PHONE                  0
FLAG_WORK_PHONE                 0
FLAG_CONT_MOBILE                0
FLAG_PHONE                      0
FLAG_EMAIL                      0
OCCUPATION_TYPE             96391
CNT_FAM_MEMBERS                 2
REGION_RATING_CLIENT            0
```

Finding the value count of AMT_ANNUITY:

df_application.AMT_ANNUITY.value_counts()

Output:

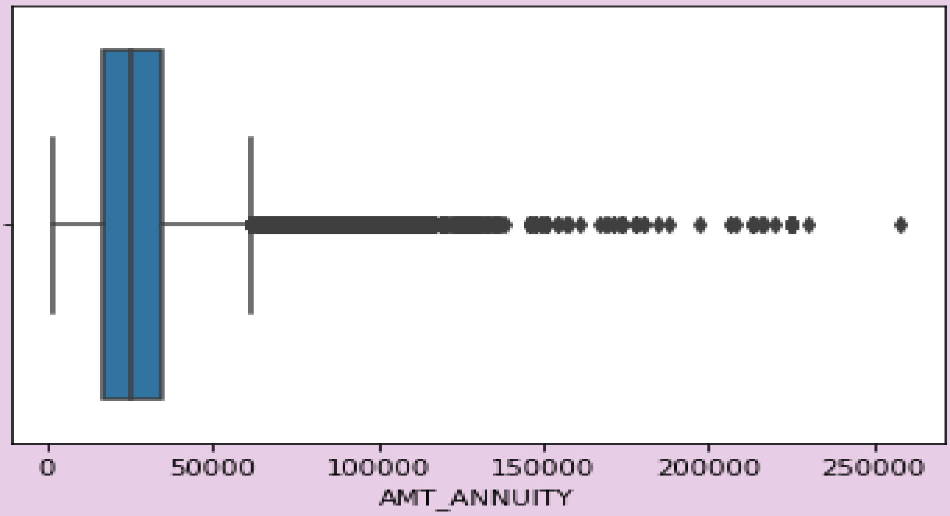
9000.0	6385
13500.0	5514
6750.0	2279
10125.0	2035
37800.0	1602
...	
4635.0	1
65209.5	1
70920.0	1
85792.5	1
51331.5	1

```
Name: AMT_ANNUITY, Length: 13672, dtype: int64
```

Ploting a Box plot for AMT_ANNUITY to check the outliers :

sns.boxplot(df_application.AMT_ANNUITY) plt.show()

Output:



Finding the null value count:

df_application.AMT_ANNUITY.median()

Output:

```
24903.0
```

Replacing the null values with median of AMT_ANNUITY, as we have outliers hence using mean will not be a correct imputation technique:

df_application['AMT_ANNUITY'] = df_application.AMT_ANNUITY.fillna(df_application.AMT_ANNUITY.median())

Checking the value count of AMT_GOODS_PRICE:

df_application.AMT_GOODS_PRICE.value_counts()

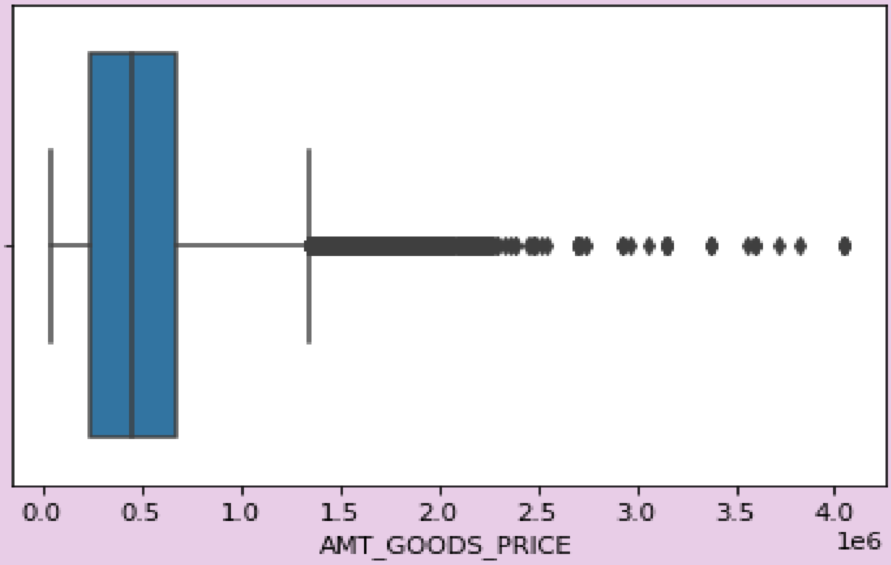
Output:

```
450000.0     26022
225000.0     25282
675000.0     24962
900000.0     15416
270000.0     11428
...
592452.0         1
1130125.5        1
362632.5         1
498856.5         1
1271875.5        1

Name: AMT_GOODS_PRICE, Length: 1002, dtype: int64
```

Checking for outliers in AMT_GOODS_PRICE as it is a continuous variable:

sns.boxplot(df_application['AMT_GOODS_PRICE']) plt.show()

Output:



Imputing null values with AMT_CREDIT based on the assumption that the amount of loan taken is equal to the amount of goods purchase:

df_application["AMT_GOODS_PRICE"] = df_application.AMT_GOODS_PRICE.fillna(df_application['AMT_GOODS_PRICE'] == df_application['AMT_CREDIT'])

Calculating the null value count:

df_application['NAME_TYPE_SUITE'].isna().sum()

Output:

1292

Replacing the null values by mode for this categorical variable:

df_application["NAME_TYPE_SUITE"] =
df_application.NAME_TYPE_SUITE.fillna("Unaccompanied")

Checking null value counts for the variable:

df_application.CNT_FAM_MEMBERS.isna().sum()

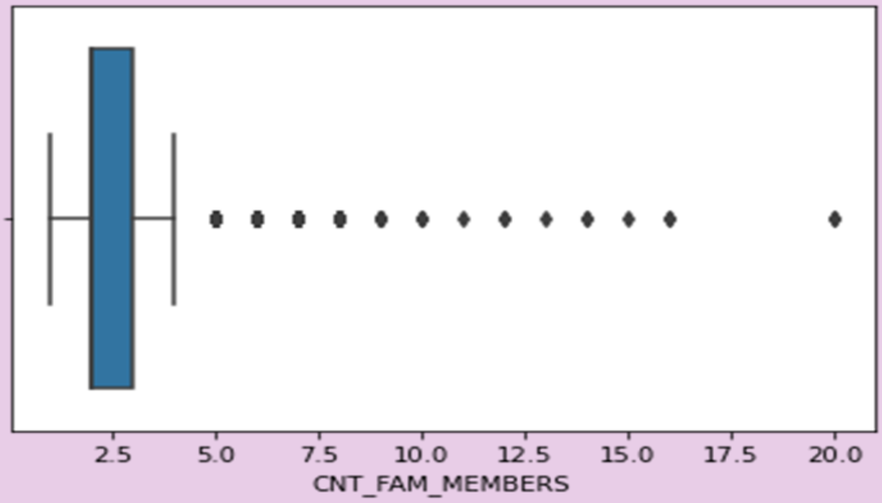
Output:

2

Plotting boxplot for variable:

sns.boxplot(df_application['CNT_FAM_MEMBERS'])

plt.show() Output:



This is a continuous variable and we can impute the mean/median inputting null values with Median due to the presence of outlier values:

df_application["CNT_FAM_MEMBERS"] =
df_application.CNT_FAM_MEMBERS.fillna(df_application.CNT_FAM_MEMBERS.median())

Percentage of each category present in "OCCUPATION_TYPE":

df_application["OCCUPATION_TYPE"].value_counts(normalize=True)*
100

Output:

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

Name: OCCUPATION_TYPE, dtype: float64

Finding the null value count:

df_application.OCCUPATION_TYPE.isnull().sum()

Output:

96391

Imputing null values with "Unknown" as using mode may distort the picture because of presence of large number of null values:

df_application["OCCUPATION_TYPE"] = df_application.OCCUPATION_TYPE.fillna("Unknown")

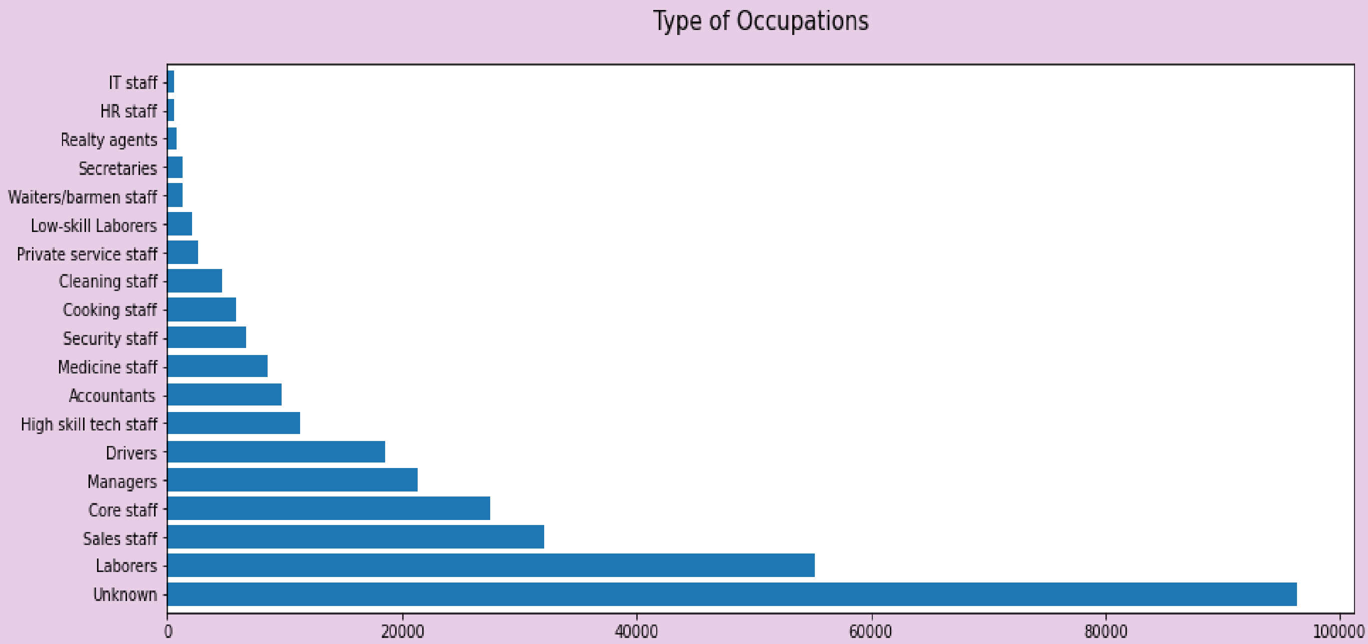
Plotting a bar graph for the variable Occupation Type to understand the distribution by various occupations:

plt.figure(figsize = [15,6])

df_application.OCCUPATION_TYPE.value_counts().plot.barh(width = .8)

plt.title("Type of Occupations", fontdict={"fontsize":15}, pad =20)

plt.show() Output:



Computing statistics for various numerical variables of number of queries to Credit Bureau about the client to understand their distribution:

df_application[["AMT_REQ_CREDIT_BUREAU_YEAR","AMT_REQ_CREDIT_BUREAU_QRT","AMT_REQ_CREDIT_BUREAU_MON","AMT_REQ_CREDIT_BUREAU_WEEK",

"AMT_REQ_CREDIT_BUREAU_DAY","AMT_REQ_CREDIT_BUREAU_HOUR"]].describe()

Output:

	AMT_REQ_CREDIT_BUREAU_YEAR	AMT_REQ_CREDIT_BUREAU_QRT	AMT_REQ_CREDIT_BUREAU_MON	AMT_REQ_CREDI
count	265992.000000	265992.000000	265992.000000	
mean	1.899974	0.265474	0.267395	
std	1.869295	0.794056	0.916002	
min	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	
50%	1.000000	0.000000	0.000000	
75%	3.000000	0.000000	0.000000	
max	25.000000	261.000000	27.000000	

Making a list of all variables pertaining to number of queries to Credit Bureau about the client:

AMT_REQ_CREDIT = ["AMT_REQ_CREDIT_BUREAU_YEAR","AMT_REQ_CREDIT_BUREAU_QRT","AMT_REQ_CREDIT_BUREAU_MON","AMT_REQ_CREDIT_BUREAU_WEEK",

"AMT_REQ_CREDIT_BUREAU_DAY","AMT_REQ_CREDIT_BUREAU_HOUR"]

Replacing the missing values with median values for the Credit Bureau list above:

df_application.fillna(df_application[AMT_REQ_CREDIT].median(),inplace = True)

Computing statistics for various numerical variables pertaining to client’s social surroundings:

df_application[['OBS_30_CNT_SOCIAL_CIRCLE','DEF_30_CNT_SOCIAL_CIRCLE','OBS_60_CNT_SOCIAL_CIRCLE','DEF_60_CNT_SOCIAL_CIRCLE']].describe()

Output:

	OBS_30_CNT_SOCIAL_CIRCLE	DEF_30_CNT_SOCIAL_CIRCLE	OBS_60_CNT_SOCIAL_CIRCLE	DEF_60_CNT_SOCIAL_CIRCLE
count	306490.000000	306490.000000	306490.000000	306490.00000
mean	1.422245	0.143421	1.405292	0.10004
std	2.400989	0.446698	2.379803	0.362291
min	0.000000	0.000000	0.000000	0.00000
25%	0.000000	0.000000	0.000000	0.00000
50%	0.000000	0.000000	0.000000	0.00000
75%	2.000000	0.000000	2.000000	0.00000
max	348.000000	34.000000	344.000000	24.00000

Making a list of all variables pertaining to client's social surroundings:

SOCIAL_CIRCLE = ['OBS_30_CNT_SOCIAL_CIRCLE','DEF_30_CNT_SOCIAL_CIRCLE','OBS_60_CNT_SOCIAL_CIRCLE','DEF_60_CNT_SOCIAL_CIRCLE']

Replacing the missing values with median values for the social surroundings list as above:

df_application.fillna(df_application[SOCIAL_CIRCLE].median(),inplace = True)

Finding the values for two variables of external sources:

df_application[['EXT_SOURCE_2','EXT_SOURCE_3']]

	EXT_SOURCE_2	EXT_SOURCE_3
0	0.262949	0.139376
1	0.622246	NaN
2	0.555912	0.729567
3	0.650442	NaN
4	0.322738	NaN
...	...	...
307506	0.681632	NaN
307507	0.115992	NaN
307508	0.535722	0.218859
307509	0.514163	0.661024
307510	0.708569	0.113922

307511 rows × 2 columns

Computing statistics for the columns pertaining to external sources:

df_application[['EXT_SOURCE_2','EXT_SOURCE_3']].describe()

Output:

	EXT_SOURCE_2	EXT_SOURCE_3
count	3.068510e+05	246546.000000
mean	5.143927e-01	0.510853
std	1.910602e-01	0.194844
min	8.173617e-08	0.000527
25%	3.924574e-01	0.370650
50%	5.659614e-01	0.535276
75%	6.636171e-01	0.669057
max	8.549997e-01	0.896010

Replacing the missing values with median values for external source variable:

```
df_application['EXT_SOURCE_2'] =
df_application.EXT_SOURCE_2.fillna(df_application['EXT_SOURCE_2'
].median())
```

Replacing the missing values with median values for external source variable:

```
df_application['EXT_SOURCE_3'] =
df_application.EXT_SOURCE_3.fillna(df_application['EXT_SOURCE_3'
].median())
```

Computing various statistics for the variable to understand about the values:

```
df_application.DAYS_LAST_PHONE_CHANGE.describe()
```

Output:

```
count    307510.000000
mean       -962.858788
std         826.808487
min       -4292.000000
25%       -1570.000000
50%        -757.000000
75%        -274.000000
max           0.000000

Name: DAYS_LAST_PHONE_CHANGE, dtype: float64
```

Finding the counts for various values of the variable:

df_application.DAYS_LAST_PHONE_CHANGE.value_counts(normalize =True)

Output:

```
0.0       0.122507
-1.0       0.009144
-2.0       0.007538
-3.0       0.005733
-4.0       0.004179

...
-3747.0    0.000003
-3999.0    0.000003
-3607.0    0.000003
-3915.0    0.000003
-3752.0    0.000003

Name: DAYS_LAST_PHONE_CHANGE, Length: 3773, dtype: float64
```

Imputing missing values with 0 which is the most occuring value:

```
df_application['DAYS_LAST_PHONE_CHANGE'] =
df_application.DAYS_LAST_PHONE_CHANGE.fillna(0)
```

Dataset 2 - "previous_application.csv"

We find out the number of null values in the dataset:

First, we find out the null values in dataframe:

```
df_previous_application.isnull().sum()
```

Output:

SK_ID_PREV	0
SK_ID_CURR	0
NAME_CONTRACT_TYPE	0
AMT_ANNUITY	372235
AMT_APPLICATION	0
AMT_CREDIT	1
AMT_DOWN_PAYMENT	895844
AMT_GOODS_PRICE	385515
WEEKDAY_APPR_PROCESS_START	0
HOUR_APPR_PROCESS_START	0
FLAG_LAST_APPL_PER_CONTRACT	0
NFLAG_LAST_APPL_IN_DAY	0
RATE_DOWN_PAYMENT	895844
RATE_INTEREST_PRIMARY	1664263
RATE_INTEREST_PRIVILEGED	1664263
NAME_CASH_LOAN_PURPOSE	0
NAME_CONTRACT_STATUS	0
DAYS_DECISION	0
NAME_PAYMENT_TYPE	0
CODE_REJECT_REASON	0
NAME_TYPE_SUITE	820405
NAME_CLIENT_TYPE	0
NAME_GOODS_CATEGORY	0
NAME_PORTFOLIO	0
NAME_PRODUCT_TYPE	0
CHANNEL_TYPE	0
SELLERPLACE_AREA	0
NAME_SELLER_INDUSTRY	0
CNT_PAYMENT	372230
NAME_YIELD_GROUP	0

Finding the percentage of null values for all variables:

Missing_Values(df_previous_application)

Output:

```
RATE_INTEREST_PRIVILEGED      99.644
RATE_INTEREST_PRIMARY         99.644
AMT_DOWN_PAYMENT              53.636
RATE_DOWN_PAYMENT             53.636
NAME_TYPE_SUITE               49.120
NFLAG_INSURED_ON_APPROVAL     40.298
DAYS_TERMINATION              40.298
DAYS_LAST_DUE                 40.298
DAYS_LAST_DUE_1ST_VERSION     40.298
DAYS_FIRST_DUE                40.298
DAYS_FIRST_DRAWING            40.298
AMT_GOODS_PRICE               23.082
AMT_ANNUITY                   22.287
CNT_PAYMENT                   22.286
PRODUCT_COMBINATION            0.021
AMT_CREDIT                     0.000
NAME_YIELD_GROUP               0.000
NAME_PORTFOLIO                 0.000
NAME_SELLER_INDUSTRY           0.000
SELLERPLACE_AREA               0.000
CHANNEL_TYPE                   0.000
NAME_PRODUCT_TYPE              0.000
SK_ID_PREV                     0.000
NAME_GOODS_CATEGORY            0.000
NAME_CLIENT_TYPE               0.000
CODE_REJECT_REASON             0.000
SK_ID_CURR                     0.000
DAYS_DECISION                  0.000
NAME_CONTRACT_STATUS           0.000
NAME_CASH_LOAN_PURPOSE         0.000
```

Finding all the variables with null value % >50%:

Null_2 = Missing_Values(df_previous_application)[Missing_Values(df_previous_application) > 50]

Null_2

Output:

```
RATE_INTEREST_PRIVILEGED    99.644
RATE_INTEREST_PRIMARY       99.644
AMT_DOWN_PAYMENT            53.636
RATE_DOWN_PAYMENT           53.636
dtype: float64
```

Retrieving Variable names with null values >50%:

Null_2.index

Output:

```
Index(['RATE_INTEREST_PRIVILEGED', 'RATE_INTEREST_PRIMARY', 'AMT_DOWN_PAYMENT',
'RATE_DOWN_PAYMENT'], dtype='object')
```

Dropping the unnecessary columns from the data fame as they are not important:

df_previous_application.drop(columns=Null_2.index ,inplace=True)
print('Null_2 dropped from the df_previous_application')

Output:

```
Null_2 dropped from the df_previous_application
```

Finding the percentage of null values for other variables:

Missing_Values(df_previous_application)

Output:

NAME_TYPE_SUITE	49.120
DAYS_FIRST_DRAWING	40.298
DAYS_TERMINATION	40.298
DAYS_LAST_DUE	40.298
DAYS_LAST_DUE_1ST_VERSION	40.298
DAYS_FIRST_DUE	40.298
NFLAG_INSURED_ON_APPROVAL	40.298
AMT_GOODS_PRICE	23.082
AMT_ANNUITY	22.287
CNT_PAYMENT	22.286
PRODUCT_COMBINATION	0.021
AMT_CREDIT	0.000
WEEKDAY_APPR_PROCESS_START	0.000
HOUR_APPR_PROCESS_START	0.000
NAME_CONTRACT_TYPE	0.000
AMT_APPLICATION	0.000
NAME_YIELD_GROUP	0.000
NAME_SELLER_INDUSTRY	0.000
SELLERPLACE_AREA	0.000
CHANNEL_TYPE	0.000
NAME_PRODUCT_TYPE	0.000
NAME_PORTFOLIO	0.000
NAME_GOODS_CATEGORY	0.000
NAME_CLIENT_TYPE	0.000
SK_ID_CURR	0.000
CODE_REJECT_REASON	0.000
NAME_PAYMENT_TYPE	0.000
DAYS_DECISION	0.000
NAME_CONTRACT_STATUS	0.000
NAME_CASH_LOAN_PURPOSE	0.000

Finding the count of various values for the variable:

df_previous_application.NAME_TYPE_SUITE.value_counts()

Output:

Unaccompanied	508970
Family	213263
Spouse, partner	67069
Children	31566
Other_B	17624
Other_A	9077
Group of people	2240

```
Name: NAME_TYPE_SUITE, dtype: int64
```

Imputing null values with mode:

df_previous_application['NAME_TYPE_SUITE'] = df_previous_application.NAME_TYPE_SUITE.fillna('Unaccompanied')

Imputing null values with AMT_CREDIT based on the assumption that the amount of good purchased is equal to the loan amount:

df_previous_application["AMT_GOODS_PRICE"] = df_previous_application.AMT_GOODS_PRICE.fillna(df_previous_application['AMT_GOODS_PRICE'] == df_previous_application['AMT_CREDIT'])

Imputing null values with median:

df_previous_application['AMT_ANNUITY'] = df_previous_application.AMT_ANNUITY.fillna(df_previous_application.AMT_ANNUITY.median())

Finding value counts for various product combinations:

df_previous_application.PRODUCT_COMBINATION.value_counts()

Output:

Cash	285990
POS household with interest	263622
POS mobile with interest	220670
Cash X-Sell: middle	143883
Cash X-Sell: low	130248
Card Street	112582

```
POS industry with interest         98833
POS household without interest     82908
Card X-Sell                        80582
Cash Street: high                  59639
Cash X-Sell: high                  59301
Cash Street: middle                34658
Cash Street: low                   33834
POS mobile without interest        24082
POS other with interest            23879
POS industry without interest      12602
POS others without interest         2555
Name: PRODUCT_COMBINATION, dtype: int64
```

Imputing NA values with mode:

df_previous_application['PRODUCT_COMBINATION'] = df_previous_application.PRODUCT_COMBINATION.fillna(df_previous_application.PRODUCT_COMBINATION.mode()[0])

Imputing null values with median:

df_previous_application['CNT_PAYMENT'] = df_previous_application.CNT_PAYMENT.fillna(df_previous_application.CNT_PAYMENT.median())

Finding values greater than 40% of null values:

Missing_Values(df_previous_application)

Output:

NFLAG_INSURED_ON_APPROVAL	40.298
DAYS_TERMINATION	40.298
DAYS_LAST_DUE	40.298
DAYS_LAST_DUE_1ST_VERSION	40.298
DAYS_FIRST_DUE	40.298
DAYS_FIRST_DRAWING	40.298
AMT_CREDIT	0.000
SELLERPLACE_AREA	0.000
NAME_PORTFOLIO	0.000
NAME_PRODUCT_TYPE	0.000
CHANNEL_TYPE	0.000
PRODUCT_COMBINATION	0.000
NAME_SELLER_INDUSTRY	0.000
CNT_PAYMENT	0.000
NAME_YIELD_GROUP	0.000
NAME_CLIENT_TYPE	0.000
NAME_GOODS_CATEGORY	0.000
SK_ID_PREV	0.000
SK_ID_CURR	0.000
CODE_REJECT_REASON	0.000
NAME_PAYMENT_TYPE	0.000
DAYS_DECISION	0.000
NAME_CONTRACT_STATUS	0.000
NAME_CASH_LOAN_PURPOSE	0.000
NFLAG_LAST_APPL_IN_DAY	0.000
FLAG_LAST_APPL_PER_CONTRACT	0.000
HOUR_APPR_PROCESS_START	0.000
WEEKDAY_APPR_PROCESS_START	0.000
AMT_GOODS_PRICE	0.000
AMT_APPLICATION	0.000

Standardizing Numerical values, changing data types and Creating buckets:

Dataset 1 - "application_data.csv"

Converting Days to Years to improve Readability:

```
df_application[["DAYS_BIRTH", "DAYS_EMPLOYED",
"DAYS_REGISTRATION", "DAYS_ID_PUBLISH",
"DAYS_LAST_PHONE_CHANGE"]] =
abs(df_application[["DAYS_BIRTH", "DAYS_EMPLOYED",
"DAYS_REGISTRATION", "DAYS_ID_PUBLISH",
"DAYS_LAST_PHONE_CHANGE"]])
```

Converting days to years up to 2 decimal places:

df_application['AGE_IN_YEARS'] = round(df_application['DAYS_BIRTH']/365,2)

df_application['EMPLOYMENT_YEARS'] = round(df_application['DAYS_EMPLOYED']/365,2)

Creating a Bucket for age:

df_application['AGE_IN_YEARS_RANGE'] = pd.cut(df_application['AGE_IN_YEARS'],bins=[0,20,25,30,35,40,45,50,55,60,65,70],labels=["0-20",'20-25','25-30','30-35','35-40','40-45','45-50','50-55','55-60','60-65','above 65'])

df_application[['AGE_IN_YEARS_RANGE','AGE_IN_YEARS']]

Output:

	AGE_IN_YEARS_RANGE	AGE_IN_YEARS
0	25-30	25.92
1	45-50	45.93
2	50-55	52.18
3	50-55	52.07
4	50-55	54.61
...	...	...
307506	25-30	25.55
307507	55-60	56.92
307508	40-45	41.00
307509	30-35	32.77
307510	45-50	46.18

307511 rows × 2 columns

Creating a Bucket for Employment years:

```
df_application['EMPLOYMENT_YEARS_RANGE'] =
pd.cut(df_application['EMPLOYMENT_YEARS'],bins=[0,5,10,15,20,25,30,35,40,45,50,55],labels=["0-5",'5-10','10-15','15-20','20-25','25-30','30-35','35-40','40-45','45-50','above 50'])
```

```
df_application[['EMPLOYMENT_YEARS_RANGE','EMPLOYMENT_YEARS']]
```

Output:

	EMPLOYMENT_YEARS_RANGE	EMPLOYMENT_YEARS
0	0-5	1.75
1	0-5	3.25
2	0-5	0.62
3	5-10	8.33
4	5-10	8.32
...	...	...
307506	0-5	0.65
307507	NaN	1000.67
307508	20-25	21.70
307509	10-15	13.11
307510	0-5	3.46

307511 rows × 2 columns

Converting to lakhs up to 2 decimal places:

```
df_application['AMT_INCOME_TOTAL_in_lakhs'] =
round(df_application['AMT_INCOME_TOTAL']/100000,2)
```

df_application['AMT_CREDIT_in_lakhs'] = round(df_application['AMT_CREDIT']/100000,2)

Creating buckets:

df_application['AMT_CREDIT_in_lakhs_Range'] = pd.cut(df_application['AMT_CREDIT_in_lakhs'],bins = [0,5,10,15,20,25,30,35,40,45], labels = ['0-5L','5-10L','10-15L','15-20L','20-25L','25-30L','30-35L','35-40L','Above 40L'])
df_application[['AMT_CREDIT_in_lakhs','AMT_INCOME_TOTAL_in_lakhs','AMT_CREDIT_in_lakhs_Range','AMT_CREDIT_in_lakhs']]

Output:

	AMT_CREDIT_in_lakhs	AMT_INCOME_TOTAL_in_lakhs	AMT_CREDIT_in_lakhs_Range	AMT_CREDIT_in_lakhs
0	4.07	2.02	0-5L	4.07
1	12.94	2.70	10-15L	12.94
2	1.35	0.68	0-5L	1.35
3	3.13	1.35	0-5L	3.13
4	5.13	1.22	5-10L	5.13
...	...	...	...	...
307506	2.55	1.58	0-5L	2.55
307507	2.70	0.72	0-5L	2.70
307508	6.78	1.53	5-10L	6.78
307509	3.70	1.71	0-5L	3.70
307510	6.75	1.58	5-10L	6.75

307511 rows × 4 columns

Creating a Bucket for AMT_INCOME_TOTAL:

df_application['AMT_INCOME_TOTAL_RANGE'] = pd.cut(df_application['AMT_INCOME_TOTAL_in_lakhs'],bins = [0,1,2,3,4,5,6,7,8,9,10,100], labels = ['0-1L','1-2L','2-3L','3-4L','4-5L','5-6L','6-7L','7-8L','8-9L','9-10L','Above 10L']) df_application[['AMT_INCOME_TOTAL_RANGE','AMT_INCOME_TOTAL']]

Output:

	AMT_INCOME_TOTAL_RANGE	AMT_INCOME_TOTAL
0	2-3L	202500.0
1	2-3L	270000.0
2	0-1L	67500.0
3	1-2L	135000.0
4	1-2L	121500.0
...	...	...
307506	1-2L	157500.0
307507	0-1L	72000.0
307508	1-2L	153000.0
307509	1-2L	171000.0
307510	1-2L	157500.0

307511 rows × 2 columns

Finding the count of various values:

df_application[['AMT_INCOME_TOTAL_RANGE','AMT_INCOME_TOTAL_in_lakhs']].value_counts()

Output:

```
AMT_INCOME_TOTAL_RANGE  AMT_INCOME_TOTAL_in_lakhs
1-2L                    1.35                         35763
                        1.12                         31053
                        1.58                         26580
                        1.80                         24725
0-1L                    0.90                         22501
                                                      ...
6-7L                    6.60                             1
                        6.57                             1
3-4L                    3.89                             1
6-7L                    6.48                             1
Above 10L               90.00                            1
Length: 571, dtype: int64
```

Removing rogue outlier values to prevent distortions in analysis:

```
df_application['EMPLOYMENT_YEARS'] = df_application.EMPLOYMENT_YEARS.replace(df_application.EMPLOYMENT_YEARS.max(),np.NaN)
```

Adding a column to understand the ratio:

```
df_application['Credit_Ratio'] = round(df_application.AMT_CREDIT/df_application.AMT_INCOME_TOTAL,2)
```

```
df_application['Credit_Ratio'].head()
```

Output:

```
0    2.01
1    4.79
2    2.00
3    2.32
4    4.22
```

```
Name: Credit_Ratio, dtype: float64
```

Dataset 2 - "previous_application.csv"

Converting to lakhs:

df_previous_application['AMT_ANNUITY_LAKHS'] = df_previous_application['AMT_ANNUITY']/100000

df_previous_application['AMT_APPLICATION_LAKHS'] = df_previous_application['AMT_APPLICATION']/100000
df_previous_application['AMT_CREDIT_LAKHS'] = df_previous_application['AMT_CREDIT']/100000

Converting days to absolute number:

df_previous_application[['DAYS_DECISION','DAYS_FIRST_DRAWING','DAYS_FIRST_DUE','DAYS_LAST_DUE_1ST_VERSION', 'DAYS_LAST_DUE','DAYS_TERMINATION']] = abs(df_previous_application[['DAYS_DECISION','DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION','DAYS_LAST_DUE', 'DAYS_TERMINATION']])

Converting days to years up to 2 decimal places:

df_previous_application[['DAYS_DECISION_YEARS','DAYS_FIRST_DRAWING_YEARS','DAYS_FIRST_DUE_YEARS','DAYS_LAST_DUE_1S

T_VERSION_YEARS',
'DAYS_LAST_DUE_YEARS','DAYS_TERMINATION_YEARS']] = round(df_previous_application[['DAYS_DECISION','DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION','DAYS_LAST_DUE', 'DAYS_TERMINATION']]/365,2)

Creating various buckets:

df_previous_application['AMT_CREDIT_LAKHS_Range']=pd.cut(df_application['AMT_INCOME_TOTAL_in_lakhs'], bins = [0,1,2,3,4,5,6,7,8,9,10,100], labels = ['0-1L','1-2L','2-3L','3-4L','4-5L','5-6L','6-7L','7-8L','8-9L','9-10L','Above 10L'])
df_previous_application['AMT_APPLICATION_LAKHS_Range'] = pd.cut(df_application['AMT_INCOME_TOTAL_in_lakhs'], bins = [0,1,2,3,4,5,6,7,8,9,10,100], labels = ['0-1L','1-2L','2-3L','3-4L','4-5L','5-6L','6-7L','7-8L','8-9L','9-10L','Above 10L'])

Making a list of all the flag variables:

list_Flag = ['FLAG_MOBIL','FLAG_EMP_PHONE','FLAG_WORK_PHONE','FLAG_CONT_MOBILE','FLAG_PHONE','FLAG_EMAIL']

list_Flag

Output:

```
['FLAG_MOBIL',

'FLAG_EMP_PHONE',

'FLAG_WORK_PHONE',
```

```
'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL']
```

Conversion of 0 to No and 1 to Yes for Flag variables for better understanding of the variables:

df_application['FLAG_MOBIL'] = df_application['FLAG_MOBIL'].apply(lambda x : 'YES' if x == 1 else 'NO')

df_application['FLAG_EMP_PHONE'] = df_application['FLAG_EMP_PHONE'].apply(lambda x : 'YES' if x == 1 else 'NO')

df_application['FLAG_WORK_PHONE'] = df_application['FLAG_WORK_PHONE'].apply(lambda x : 'YES' if x == 1 else 'NO')

df_application['FLAG_CONT_MOBILE'] = df_application['FLAG_CONT_MOBILE'].apply(lambda x : 'YES' if x == 1 else 'NO')

df_application['FLAG_PHONE'] = df_application['FLAG_PHONE'].apply(lambda x : 'YES' if x == 1 else 'NO')

df_application['FLAG_EMAIL'] = df_application['FLAG_EMAIL'].apply(lambda x : 'YES' if x == 1 else 'NO')

Analysis of Variables:

Univariate Analysis:

Dataset for "application_data.csv"

Numarical_Data = ['AMT_ANNUITY','AMT_GOODS_PRICE','AGE_IN_YEARS','EMPLOYMENT_YEARS','AMT_INCOME_TOTAL_in_lakhs', 'AMT_CREDIT_in_lakhs','CNT_FAM_MEMBERS','Credit_Ratio']

Categorical_Data =
['FLAG_OWN_CAR','FLAG_OWN_REALTY','NAME_TYPE_SUITE','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE',
'NAME_FAMILY_STATUS','NAME_HOUSING_TYPE','OCCUPATION_TYPE','AGE_IN_YEARS_RANGE','EMPLOYMENT_YEARS_RANGE','AMT_CREDIT_in_lakhs_Range','AMT_INCOME_TOTAL_RANGE']

def **Uni_Analysis_Numarical**(dataframe, column):

sns.set(style='darkgrid') plt.figure(figsize=(25, 5))

plt.subplot(1, 3, 1)

sns.boxplot(data=dataframe, x=column, orient='v').set(title='Box Plot')

plt.subplot(1, 3, 2)

sns.distplot(dataframe[column].dropna()).set(title='Distplot')
plt.show()

def **Uni_Analysis_Categorcal**(dataframe, column):

sns.set(style='darkgrid') plt.figure(figsize = [12,5])

dataframe[column].value_counts().plot.barh(width = 0.8)

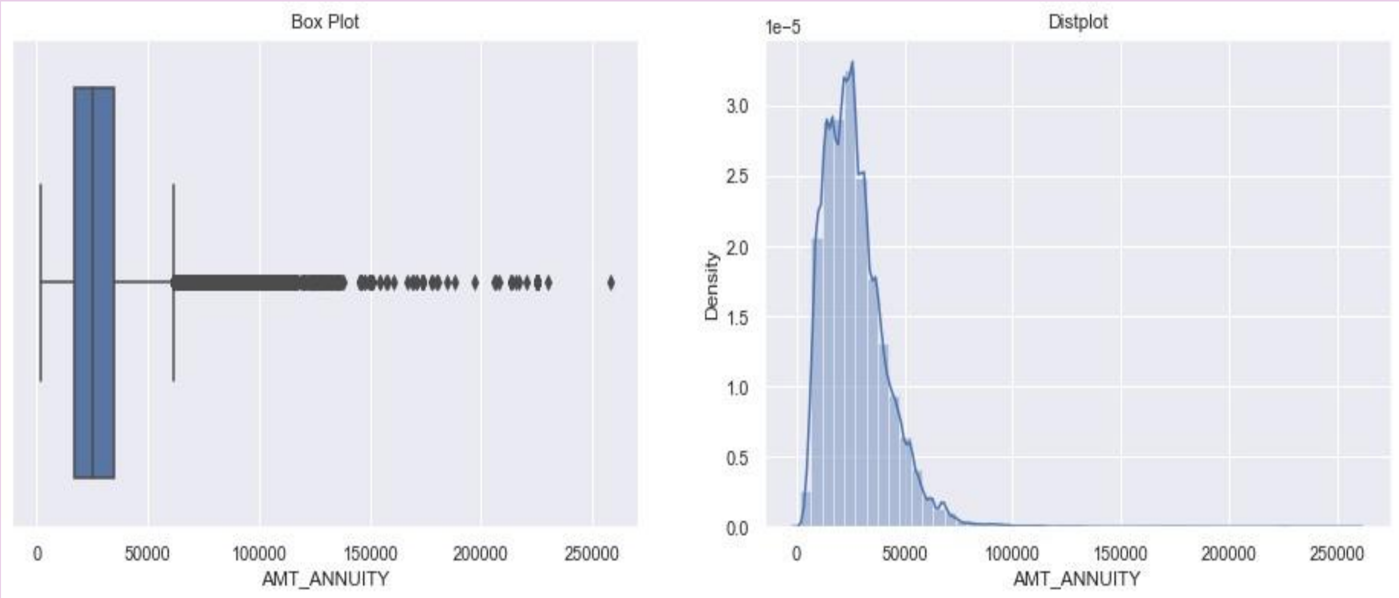
plt.title(column)

plt.show()

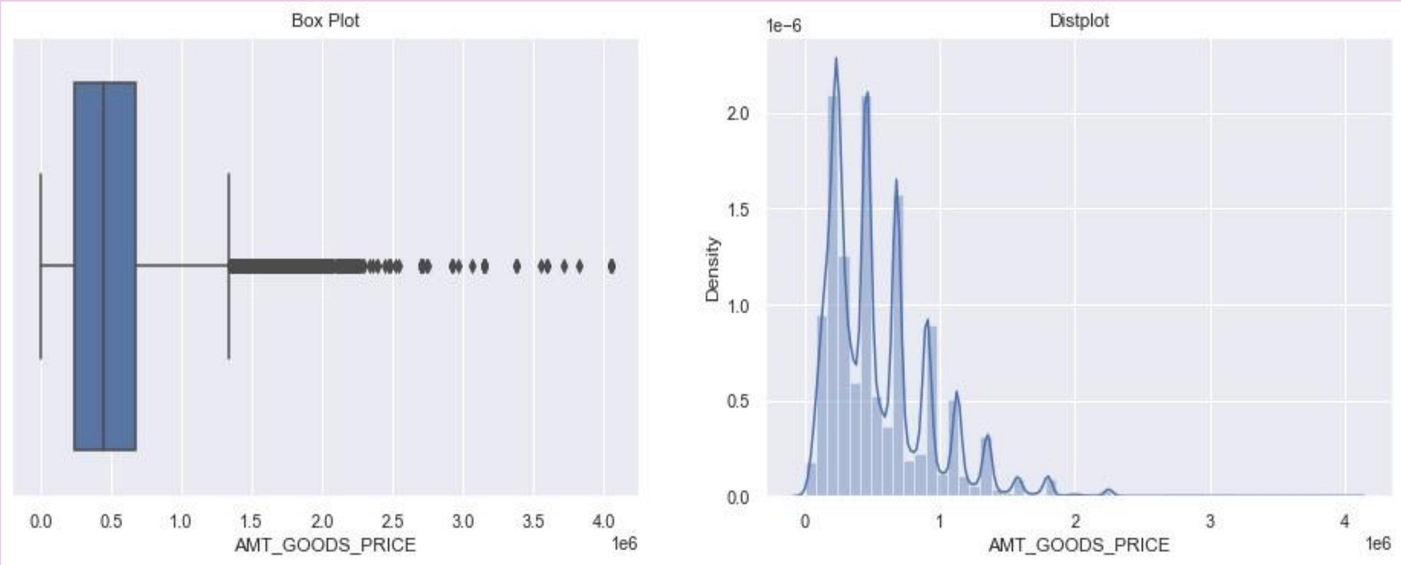
for i in Numarical_Data:

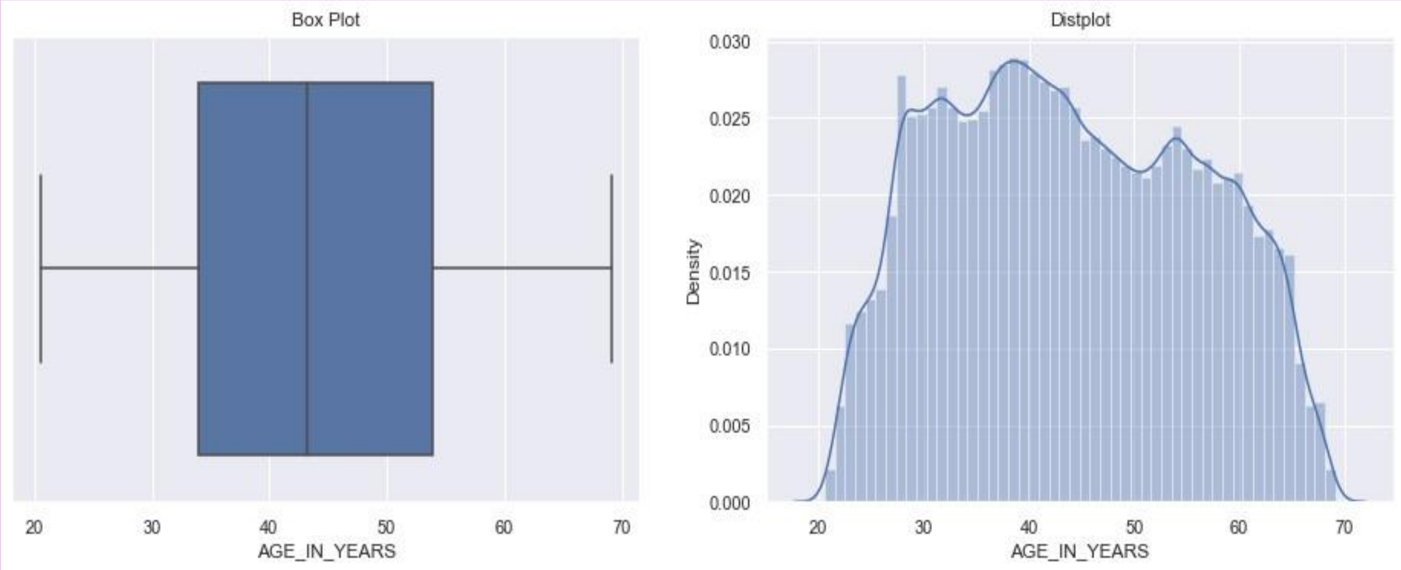
Uni_Analysis_Numarical(df_application,i)

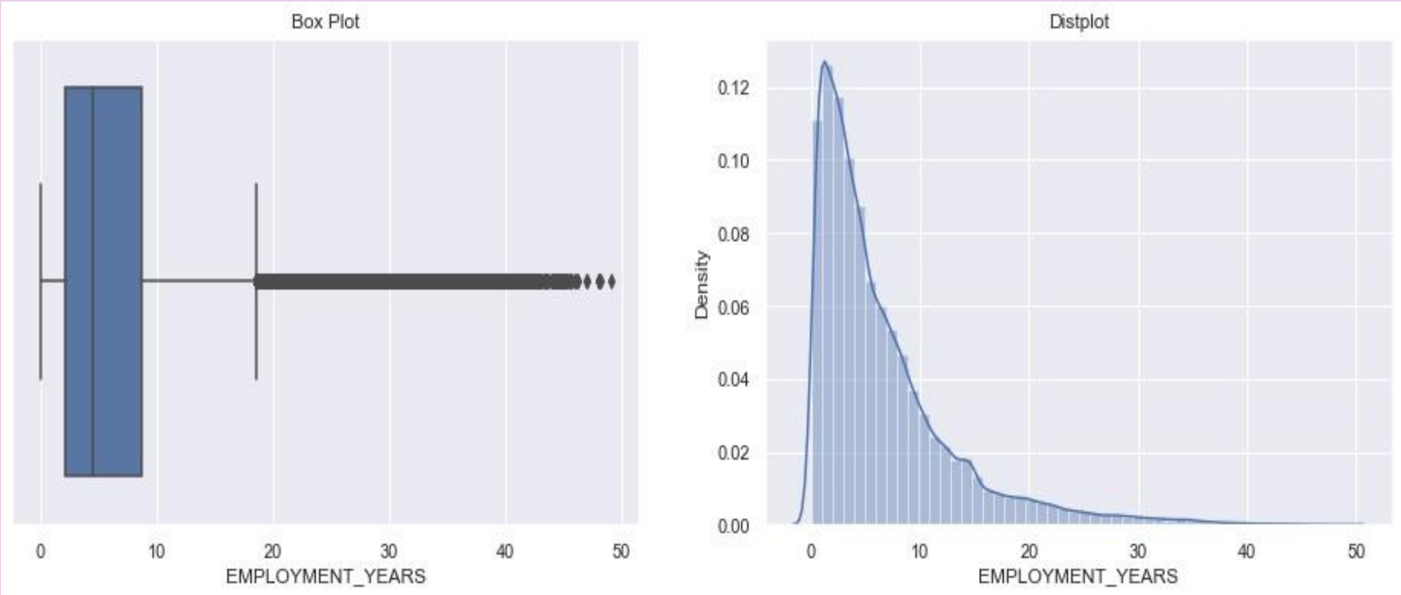
Output:

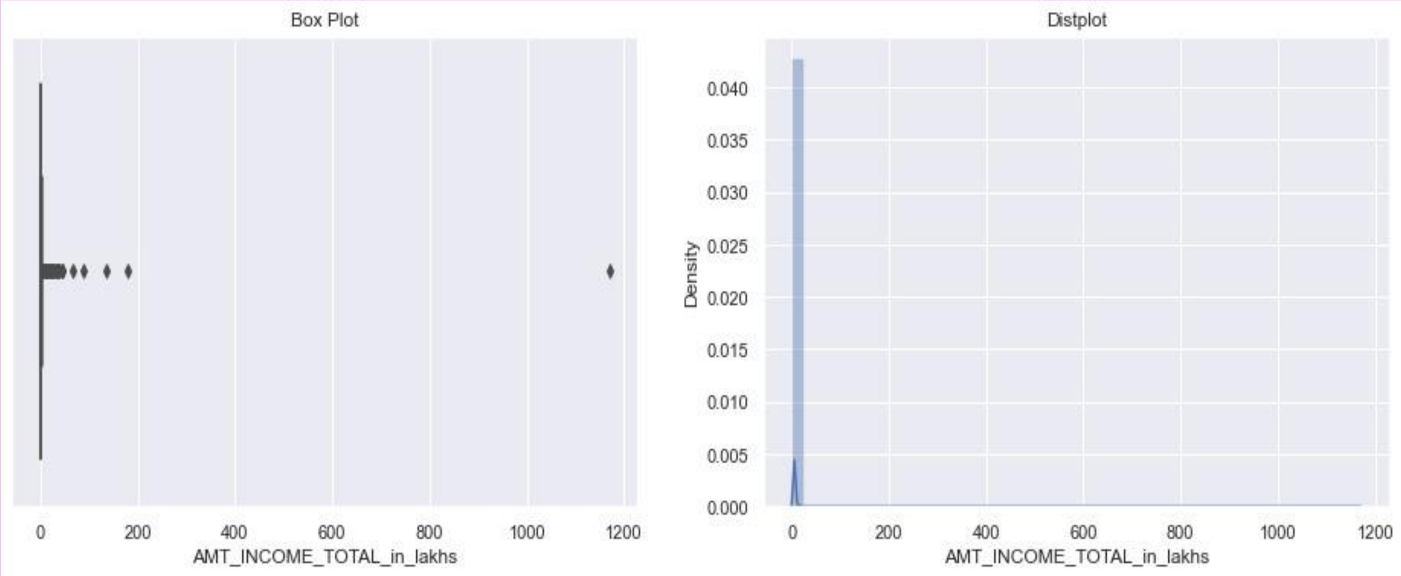


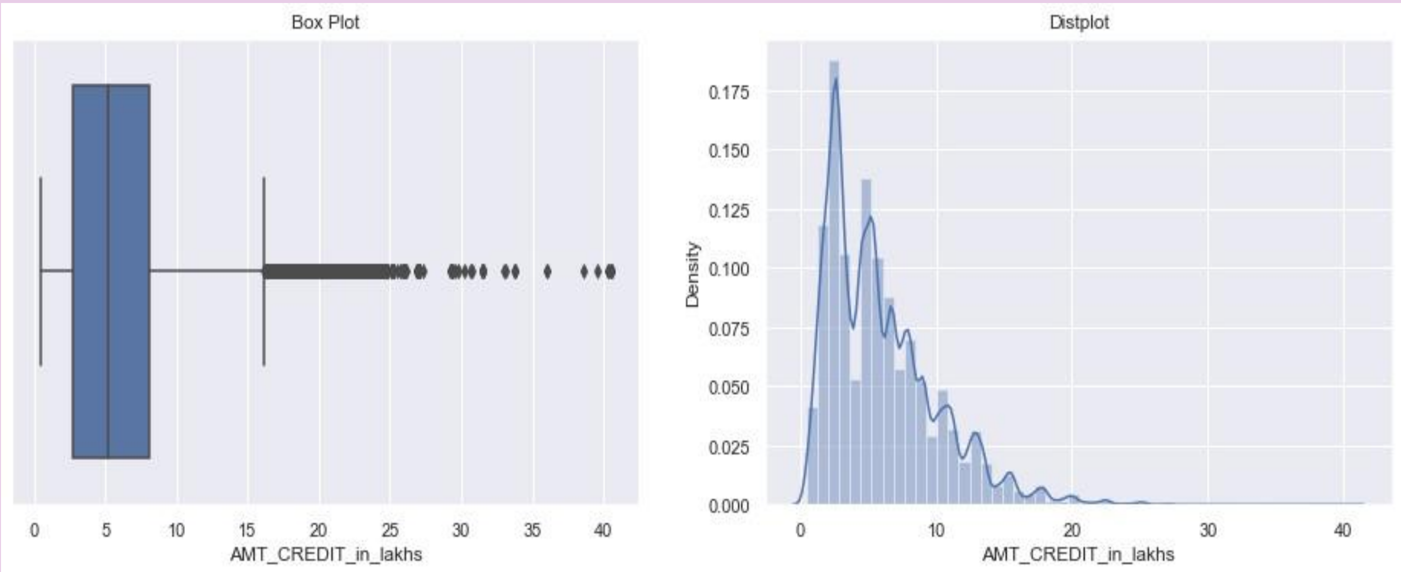
Box Plot
0
50000
100000
150000
200000
250000
AMT_ANNUITY
Distplot
1e−5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Density
0
50000
100000
150000
200000
250000
AMT_ANNUITY

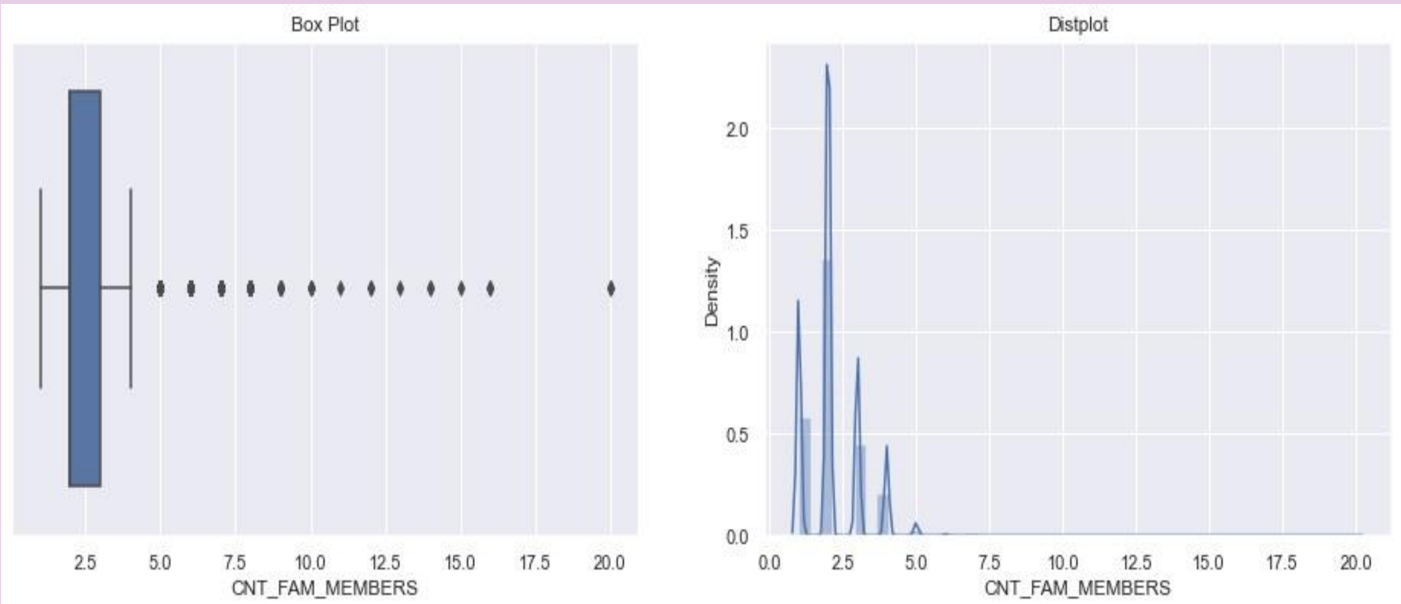

Box Plot
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
1e6
AMT_GOODS_PRICE
Distplot
1e−6
Density
0.0
0.5
1.0
1.5
2.0
0
1
2
3
4
1e6
AMT_GOODS_PRICE

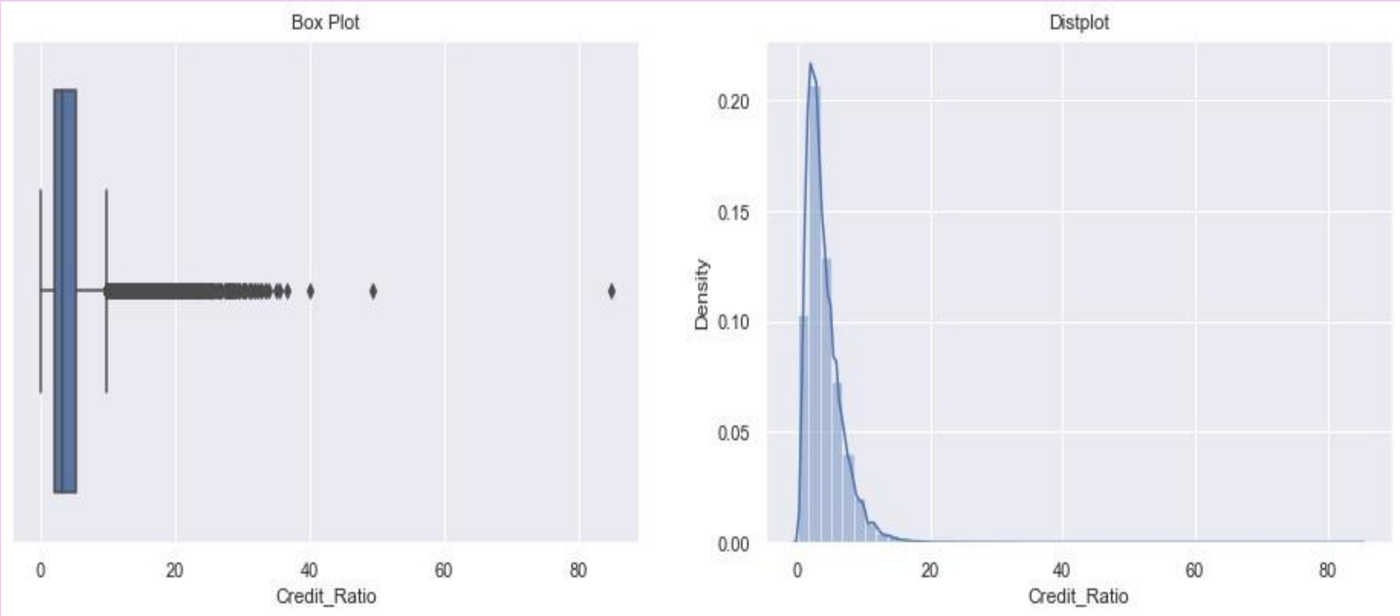

Box Plot
20
30
40
50
60
70
AGE_IN_YEARS
Distplot
Density
0.000
0.005
0.010
0.015
0.020
0.025
0.030
20
30
40
50
60
70
AGE_IN_YEARS


Box Plot
0
10
20
30
40
50
EMPLOYMENT_YEARS
Distplot
Density
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0
10
20
30
40
50
EMPLOYMENT_YEARS


Box Plot
AMT_INCOME_TOTAL_in_lakhs
Distplot
Density
AMT_INCOME_TOTAL_in_lakhs

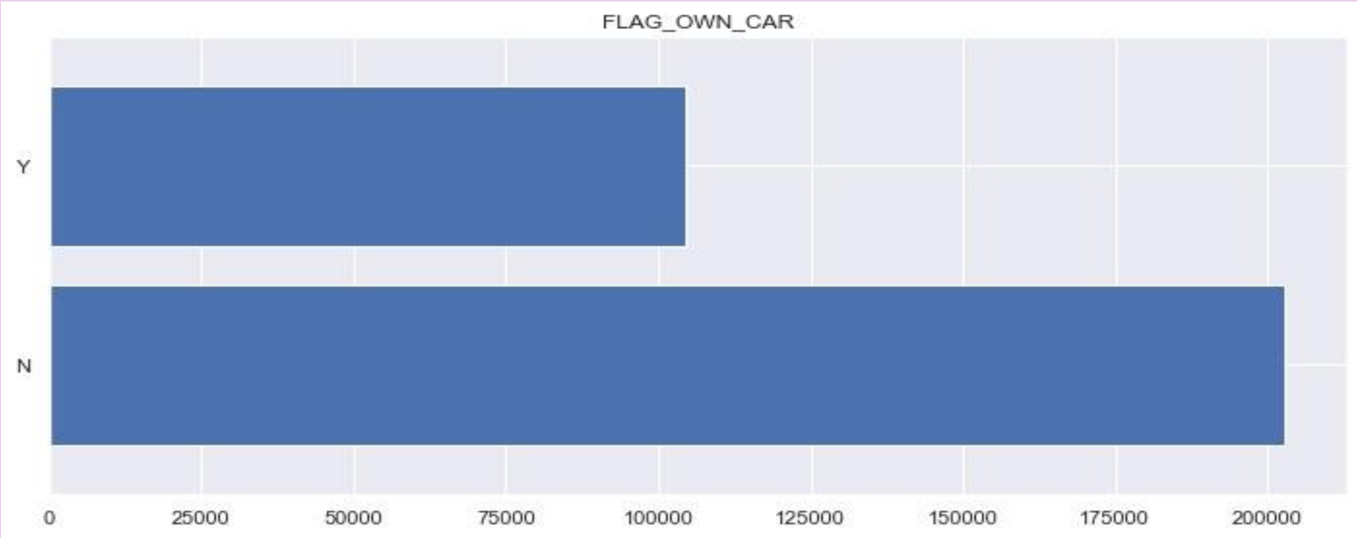

Box Plot
AMT_CREDIT_in_lakhs
Distplot
Density
AMT_CREDIT_in_lakhs

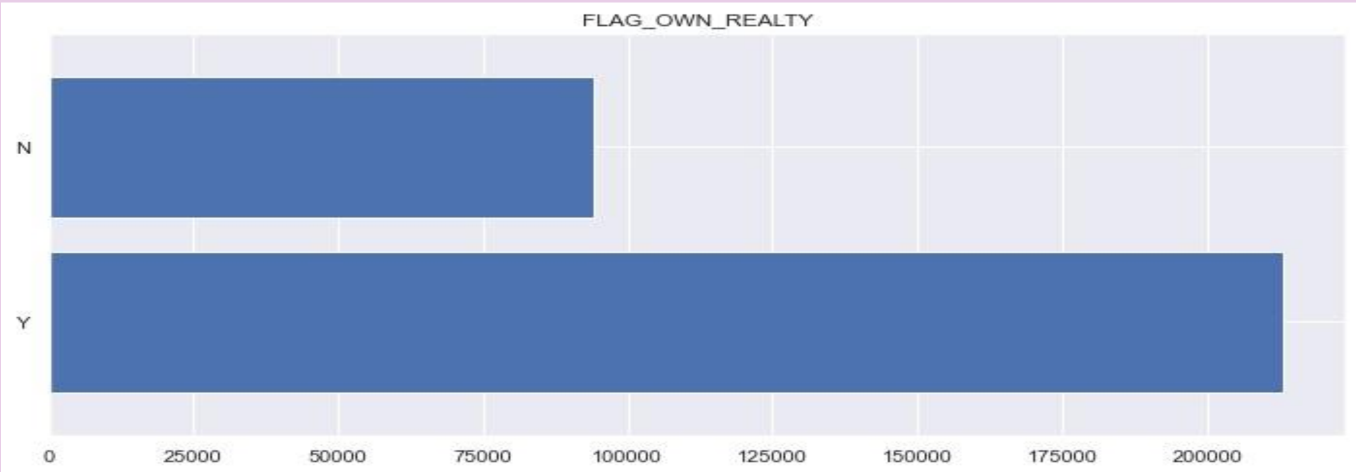

Box Plot
CNT_FAM_MEMBERS
Distplot
Density
CNT_FAM_MEMBERS

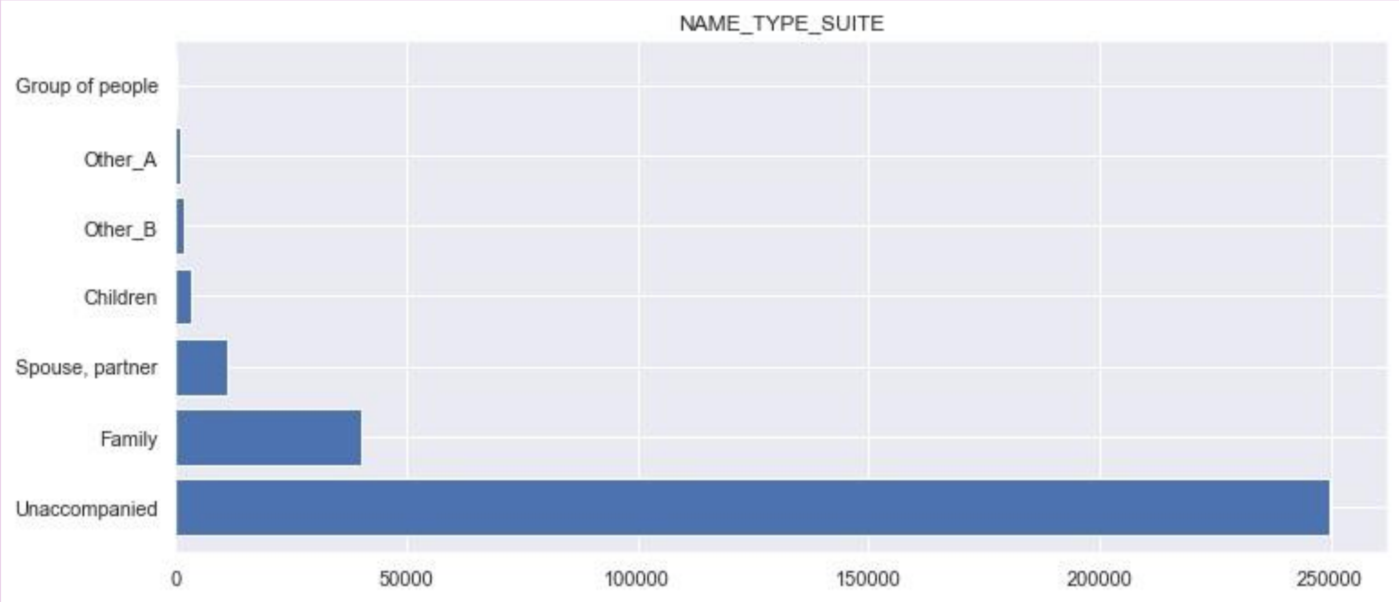


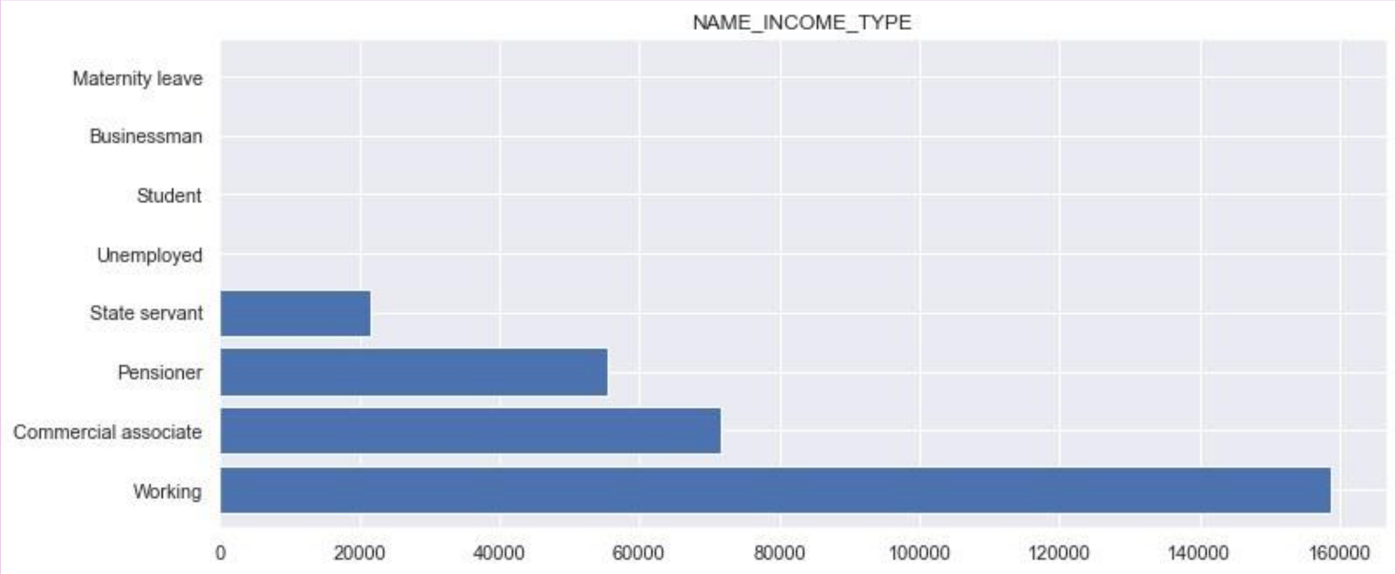

```
for i in Categorical_Data:
    Uni_Analysis_Categorcal(df_application,i)
```

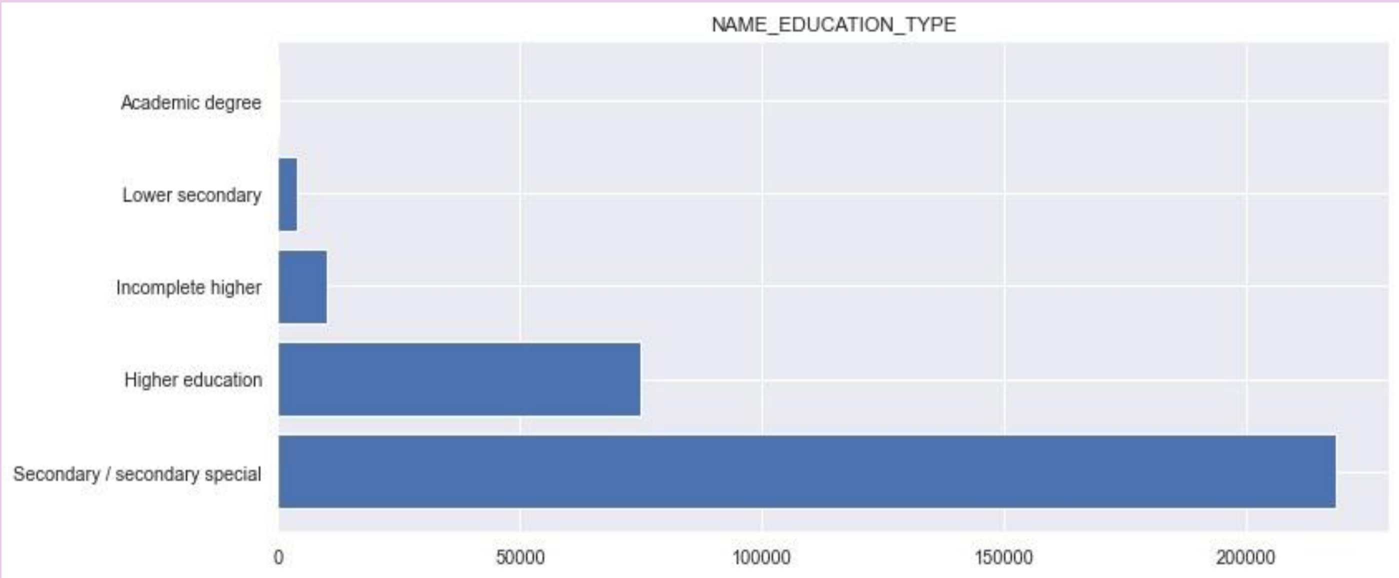
Output:

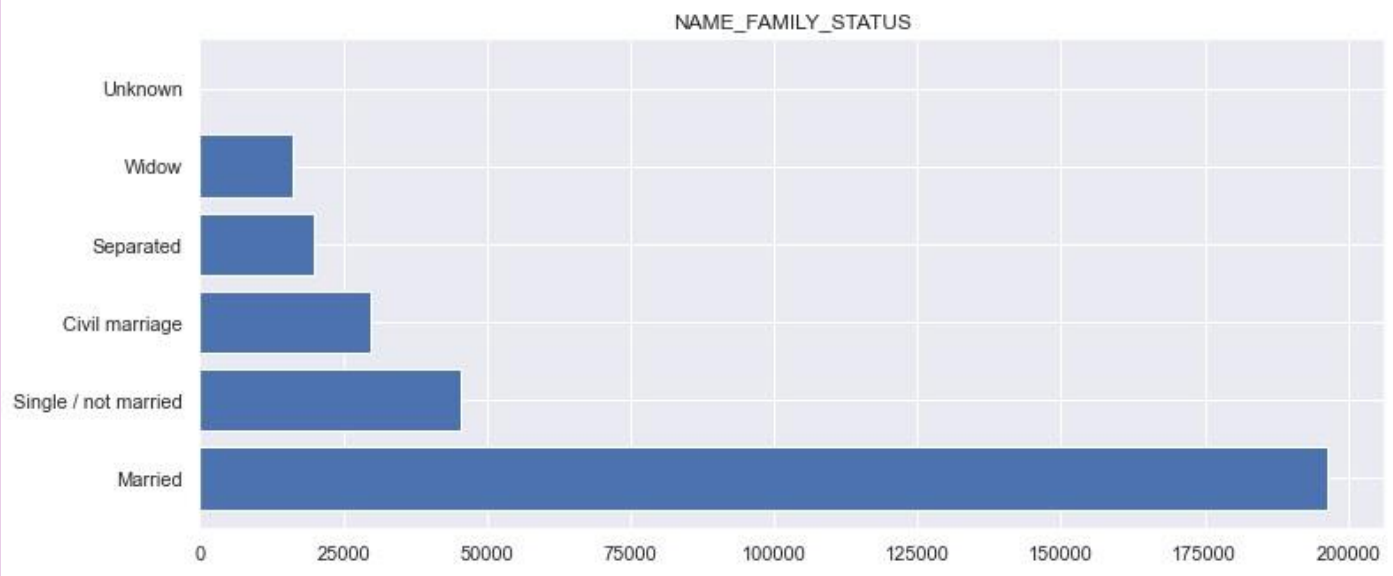

FLAG_OWN_CAR
Y
N
0
25000
50000
75000
100000
125000
150000
175000
200000

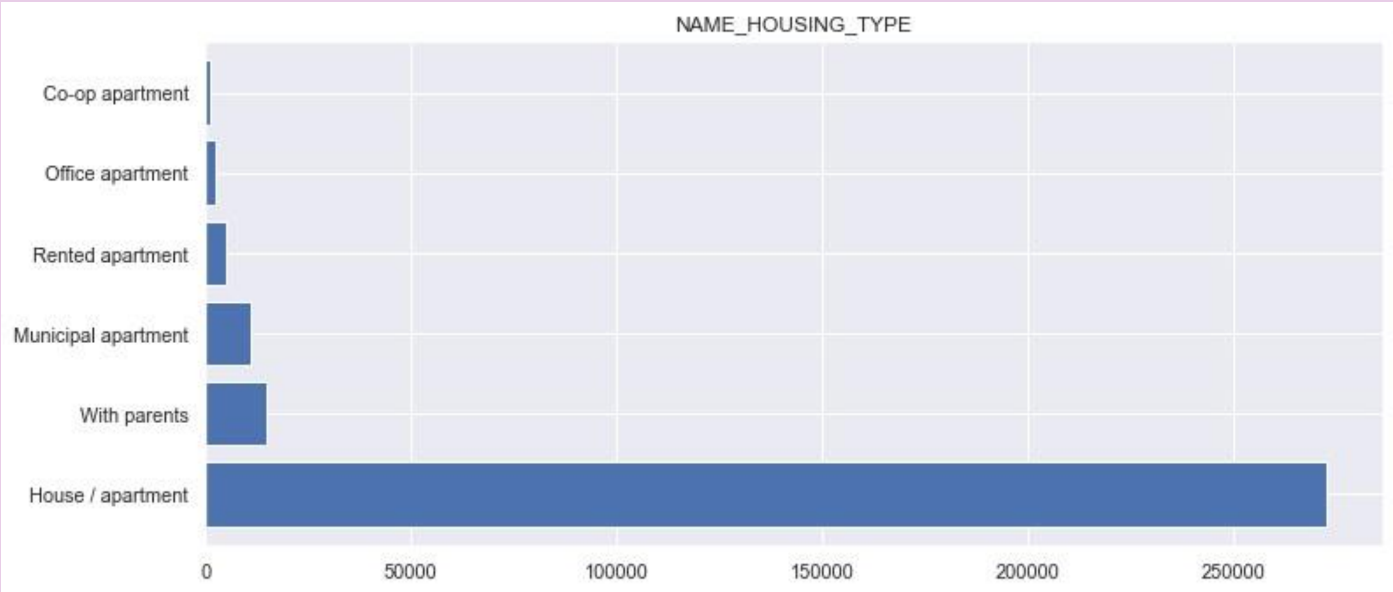

FLAG_OWN_REALTY
N
Y
0
25000
50000
75000
100000
125000
150000
175000
200000

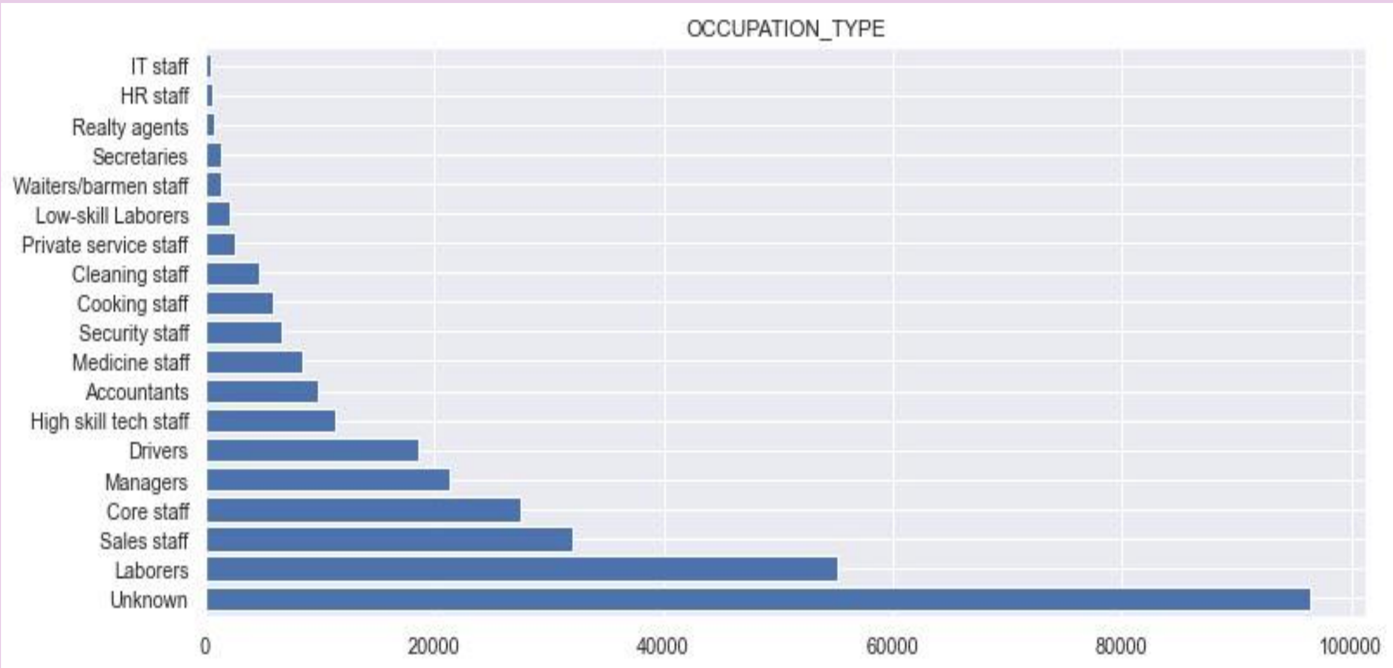

NAME_TYPE_SUITE
Group of people
Other_A
Other_B
Children
Spouse, partner
Family
Unaccompanied
0
50000
100000
150000
200000
250000

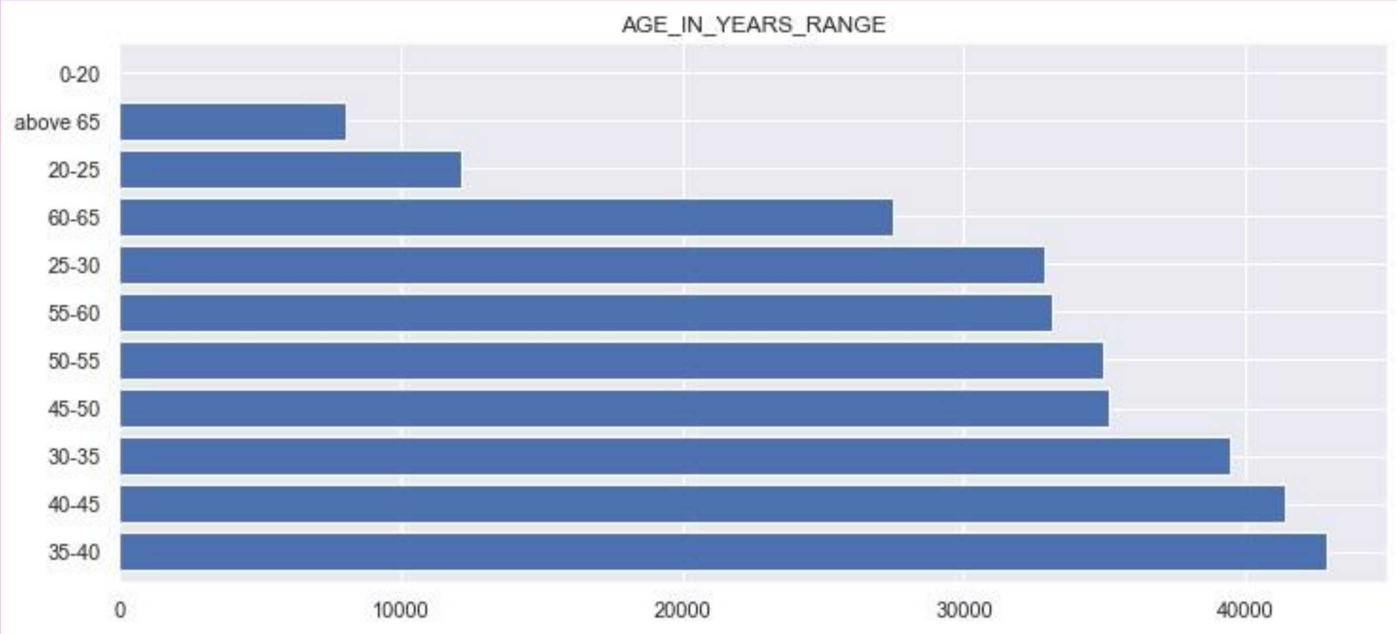

NAME_INCOME_TYPE
Maternity leave
Businessman
Student
Unemployed
State servant
Pensioner
Commercial associate
Working
0
20000
40000
60000
80000
100000
120000
140000
160000

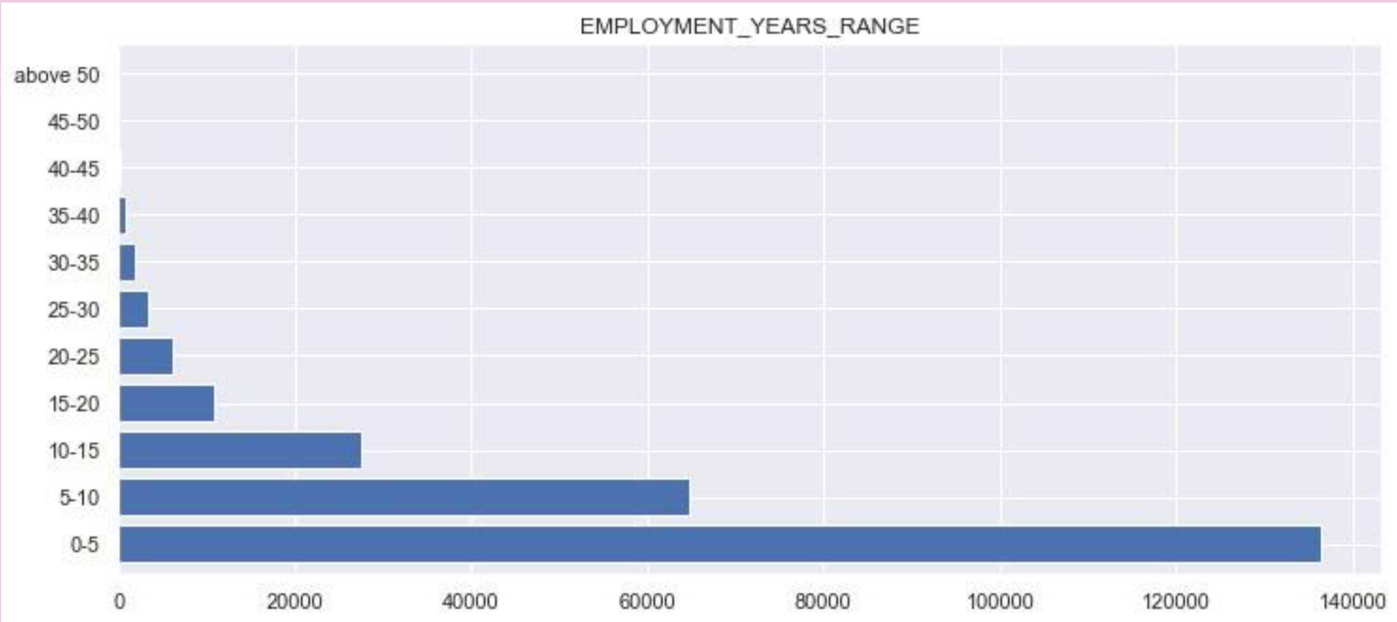

NAME_EDUCATION_TYPE
Academic degree
Lower secondary
Incomplete higher
Higher education
Secondary / secondary special
0
50000
100000
150000
200000

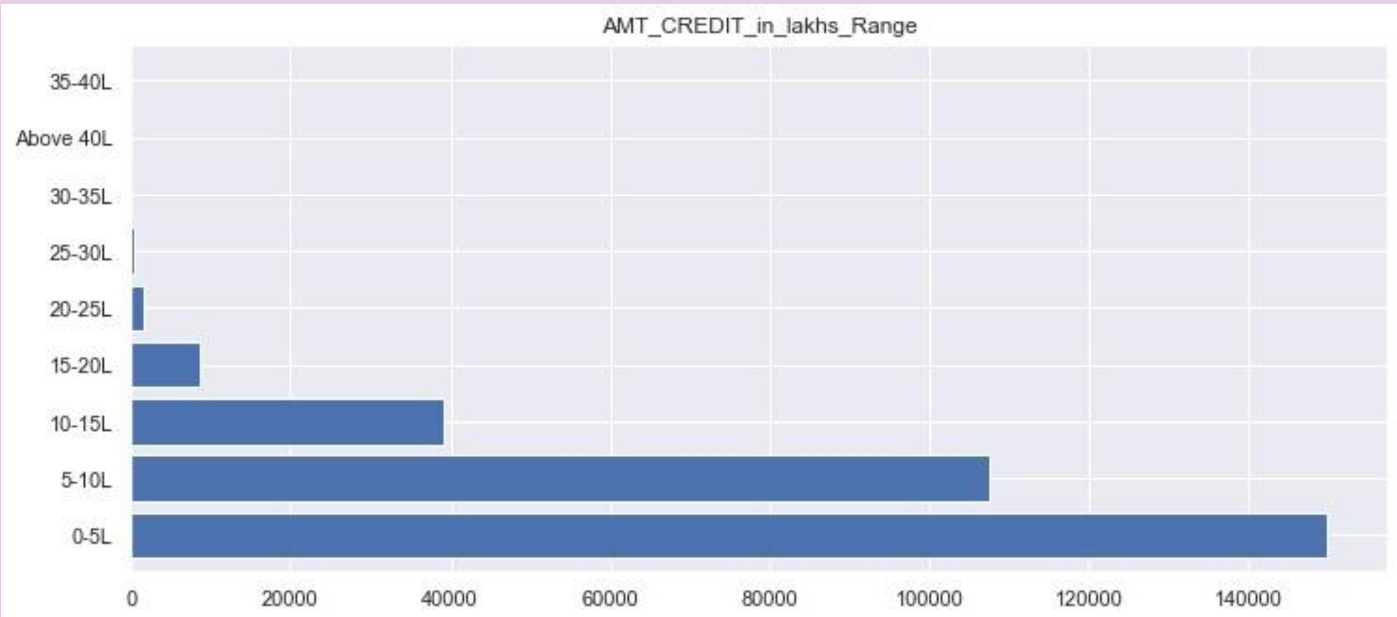

NAME_FAMILY_STATUS
Unknown
Widow
Separated
Civil marriage
Single / not married
Married
0
25000
50000
75000
100000
125000
150000
175000
200000

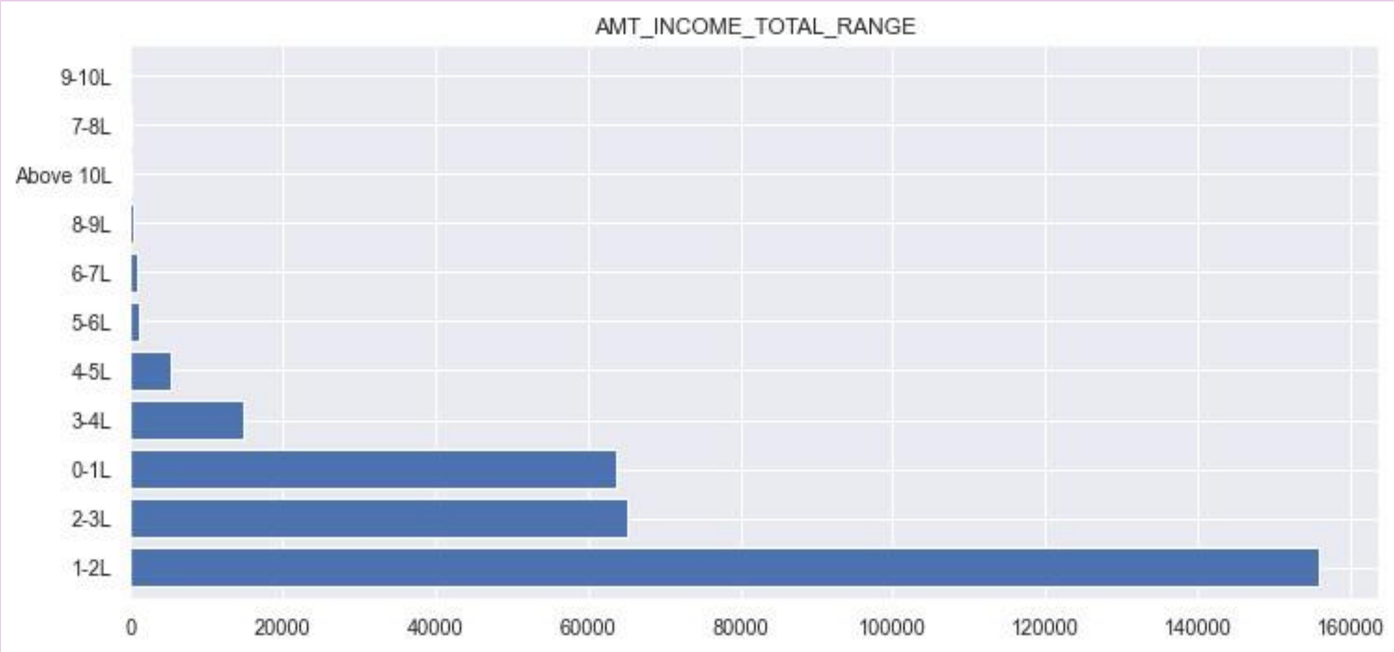

NAME_HOUSING_TYPE
Co-op apartment
Office apartment
Rented apartment
Municipal apartment
With parents
House / apartment
0
50000
100000
150000
200000
250000


OCCUPATION_TYPE
IT staff
HR staff
Realty agents
Secretaries
Waiters/barmen staff
Low-skill Laborers
Private service staff
Cleaning staff
Cooking staff
Security staff
Medicine staff
Accountants
High skill tech staff
Drivers
Managers
Core staff
Sales staff
Laborers
Unknown
0
20000
40000
60000
80000
100000


AGE_IN_YEARS_RANGE
0-20
above 65
20-25
60-65
25-30
55-60
50-55
45-50
30-35
40-45
35-40
0
10000
20000
30000
40000

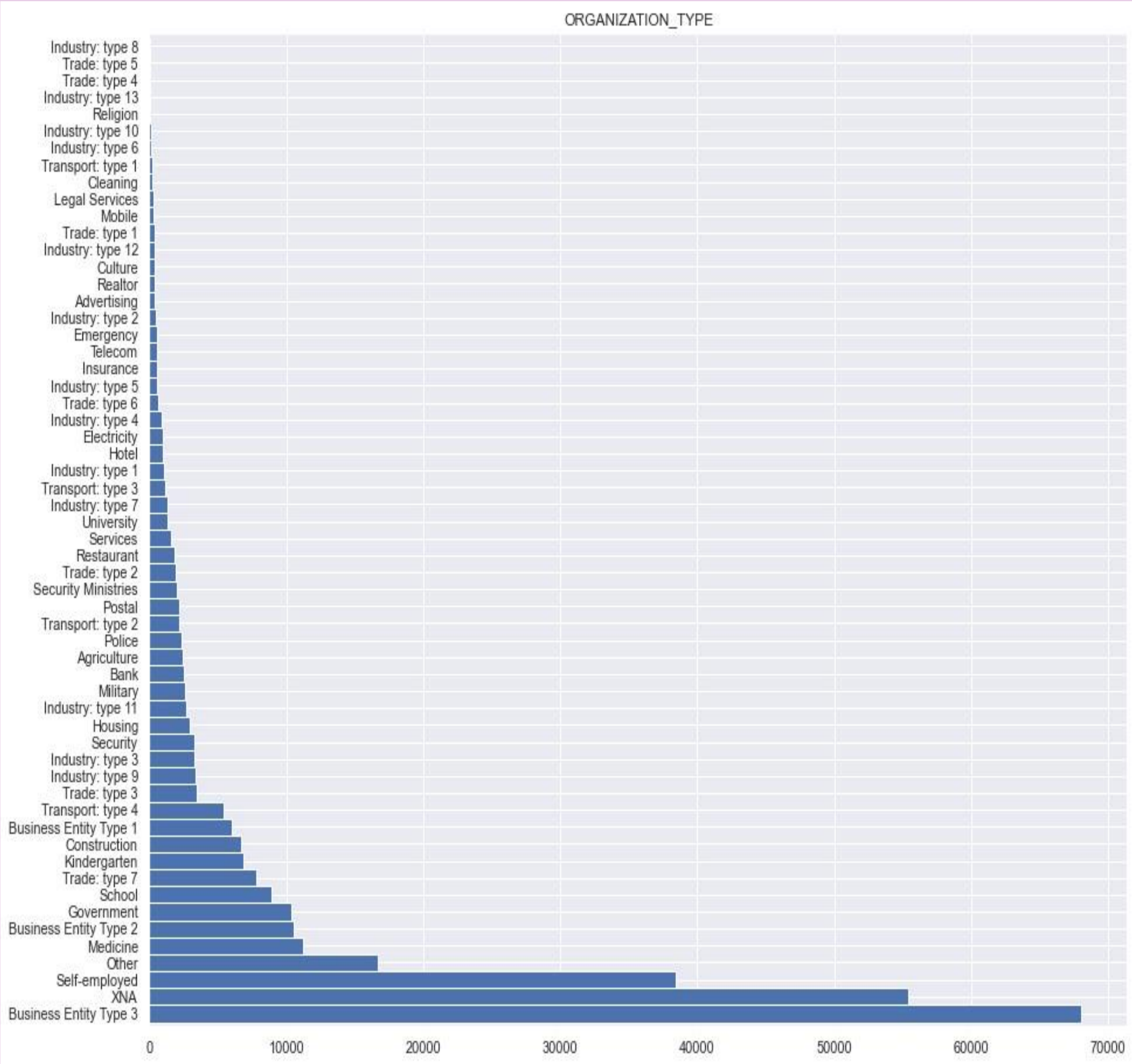

EMPLOYMENT_YEARS_RANGE
above 50
45-50
40-45
35-40
30-35
25-30
20-25
15-20
10-15
5-10
0-5
0
20000
40000
60000
80000
100000
120000
140000


AMT_CREDIT_in_lakhs_Range
35-40L
Above 40L
30-35L
25-30L
20-25L
15-20L
10-15L
5-10L
0-5L
0
20000
40000
60000
80000
100000
120000
140000




```
sns.set(style='darkgrid')
plt.figure(figsize = [15,12])

df_application['ORGANIZATION_TYPE'].value_counts().plot.barh(width = 1)
plt.title('ORGANIZATION_TYPE')

plt.show()
```

Output:

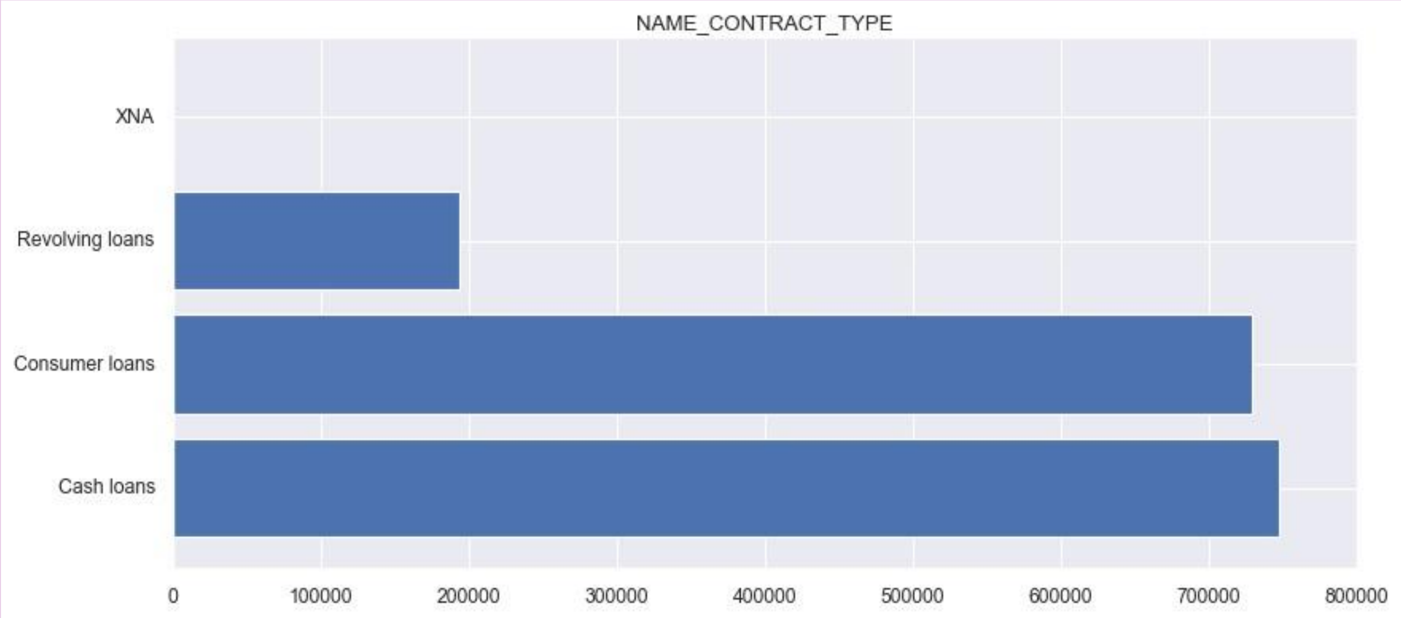


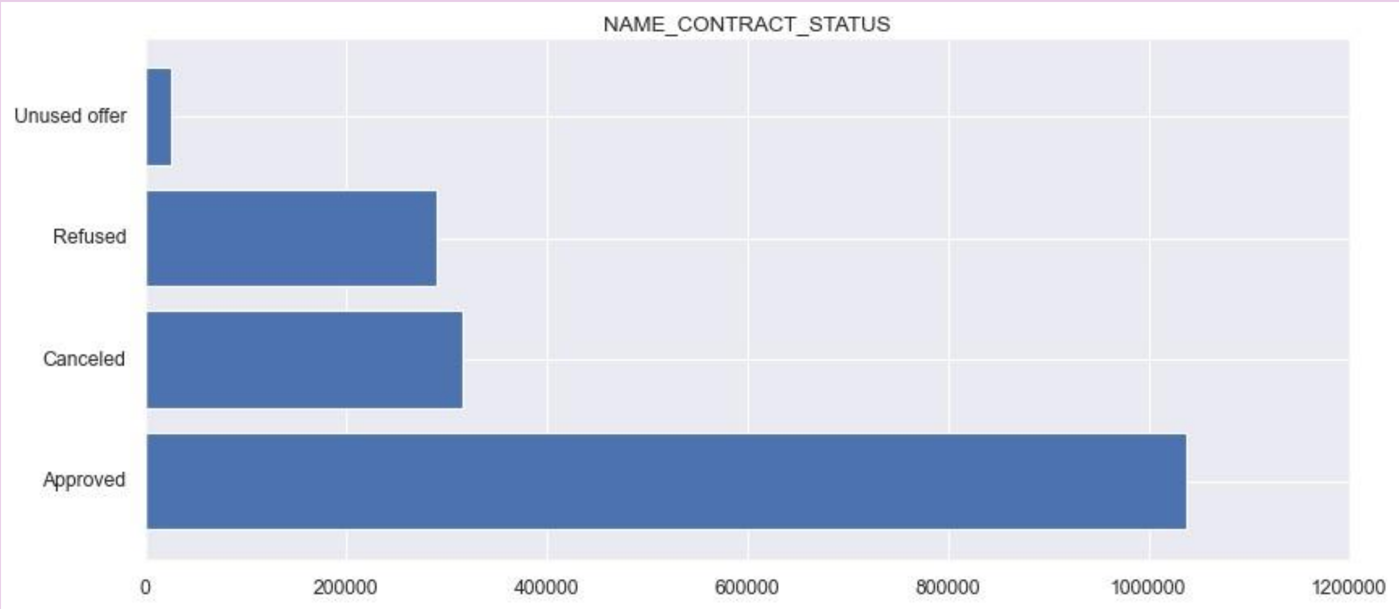
Dataset for "previous_application.csv"

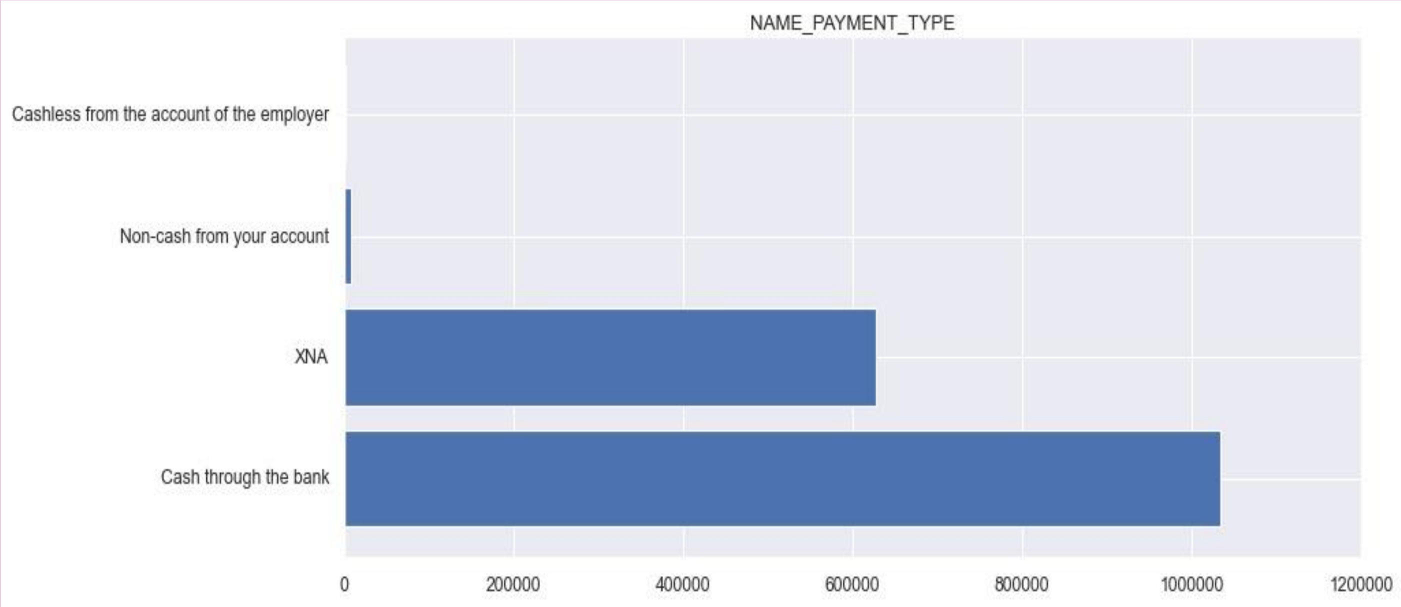
Categorical_Data_for_prev = ['NAME_CONTRACT_TYPE','NAME_CONTRACT_STATUS','NAME_PAYMENT_TYPE','NAME_TYPE_SUITE', 'NAME_CLIENT_TYPE','AMT_CREDIT_LAKHS_Range','AMT_APPLICATION_LAKHS_Range',] for i in Categorical_Data_for_prev:

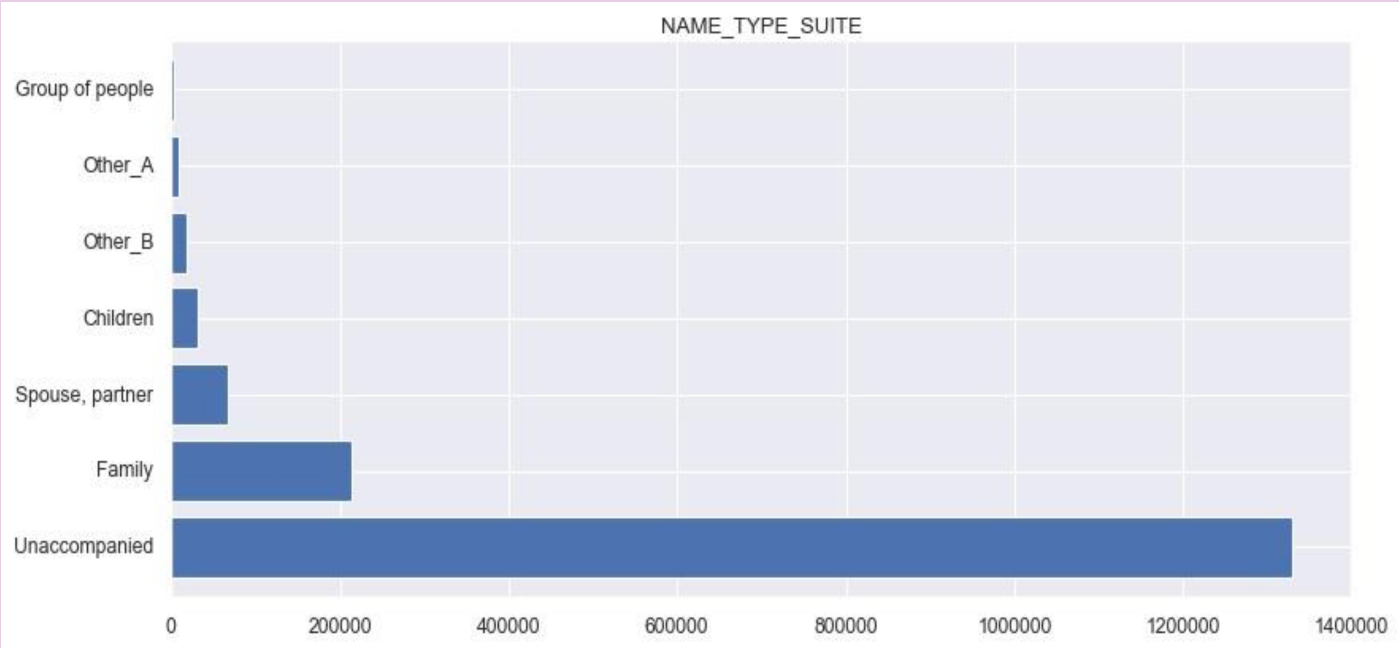
    Uni_Analysis_Categorcal(df_previous_application,i)

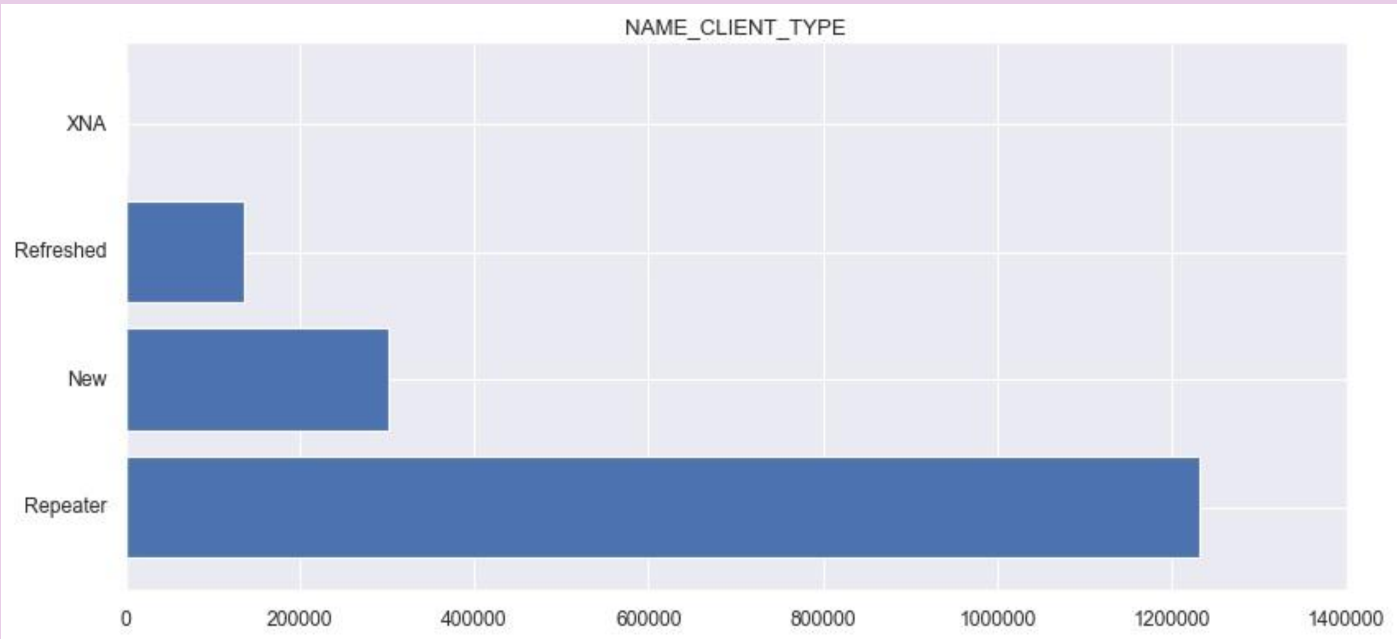
Output:

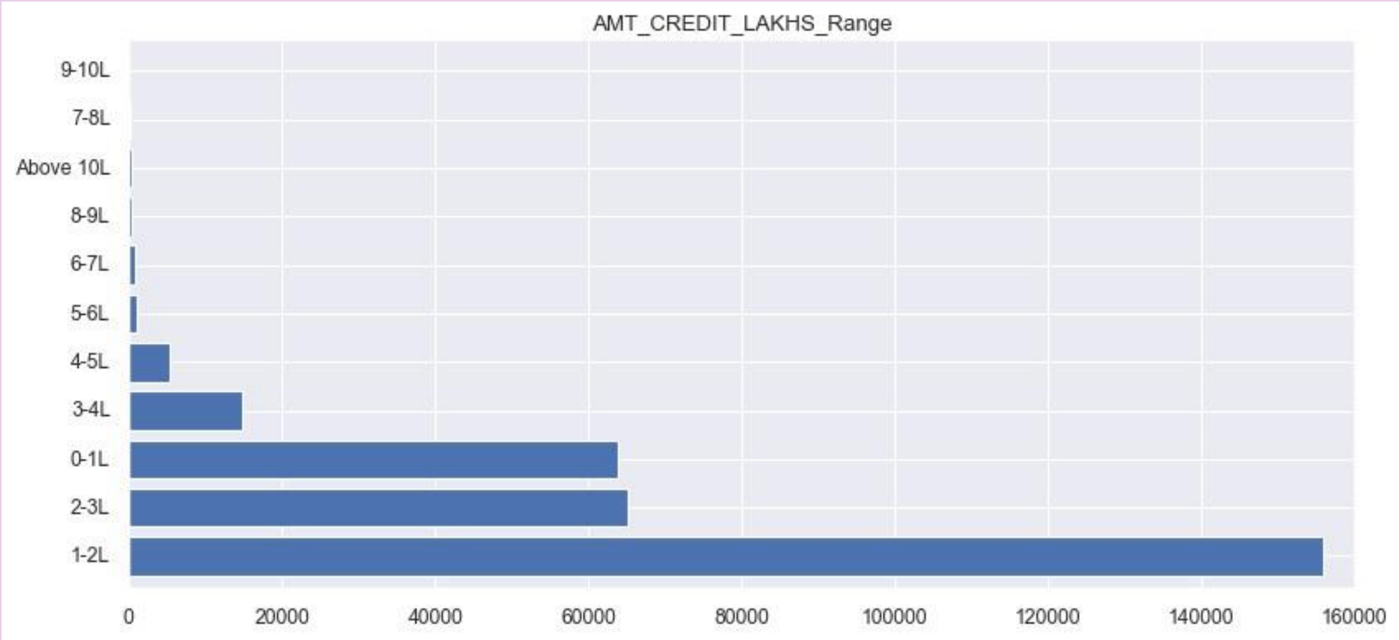


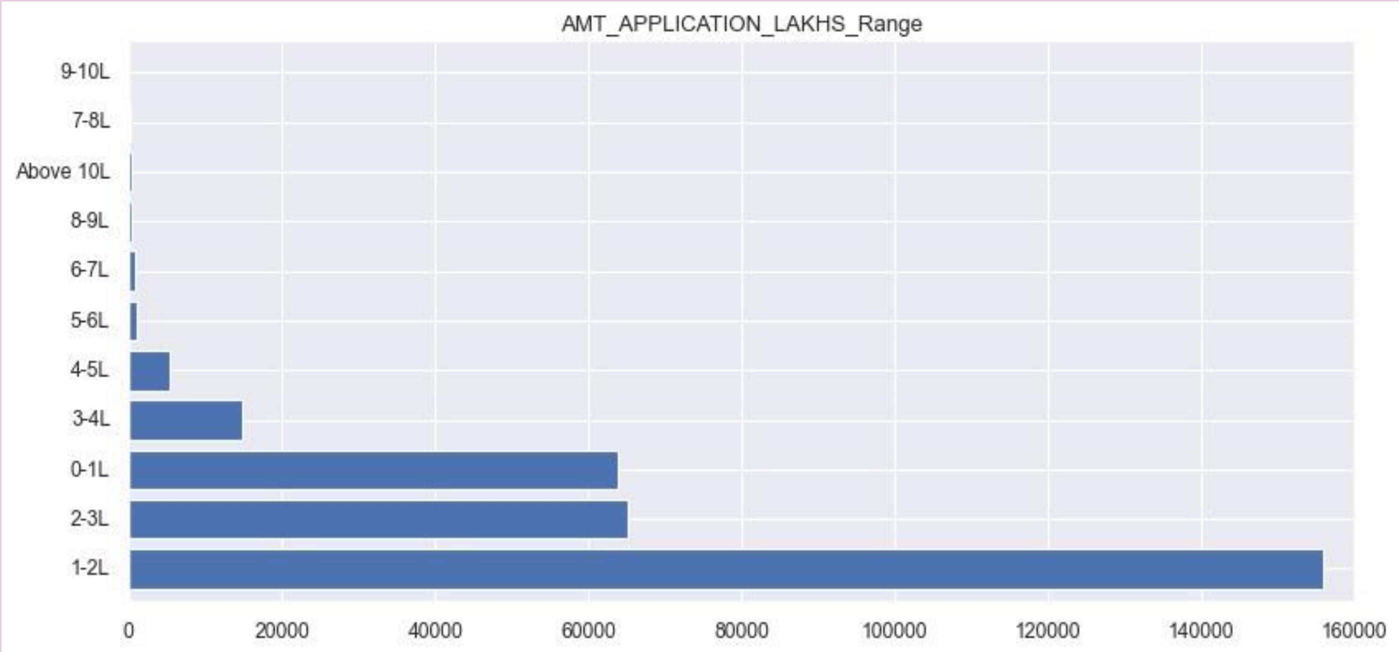



NAME_PAYMENT_TYPE
Cashless from the account of the employer
Non-cash from your account
XNA
Cash through the bank
0
200000
400000
600000
800000
1000000
1200000


NAME_TYPE_SUITE
Group of people
Other_A
Other_B
Children
Spouse, partner
Family
Unaccompanied
0
200000
400000
600000
800000
1000000
1200000
1400000


NAME_CLIENT_TYPE
XNA
Refreshed
New
Repeater
0
200000
400000
600000
800000
1000000
1200000
1400000





TARGET Analysis

Univariate Analysis of TARGET

Categorical_Data_1 =

['NAME_CONTRACT_TYPE','FLAG_OWN_CAR','FLAG_OWN_REALTY','NAME_TYPE_SUITE','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE',
'NAME_FAMILY_STATUS','NAME_HOUSING_TYPE','OCCUPATION_TYPE','AGE_IN_YEARS_RANGE',
'EMPLOYMENT_YEARS_RANGE','AMT_CREDIT_in_lakhs_Range','AMT_INCOME_TOTAL_RANGE']

Numarical_Data_1 =
['AMT_ANNUITY','AMT_GOODS_PRICE','CNT_FAM_MEMBERS','CNT_CHILDREN','Credit_Ratio']

Tagget_Variable_Payment_Difficulty =
df_application[df_application.TARGET == 1]

Tagget_Variable_All_Other = df_application[df_application.TARGET == 0]

Tagget_Variable_All_Other.CODE_GENDER.value_counts()

```
F      188278
M       94404
XNA         4

Name: CODE_GENDER, dtype: int64
```

Function to plot for categorical variables:

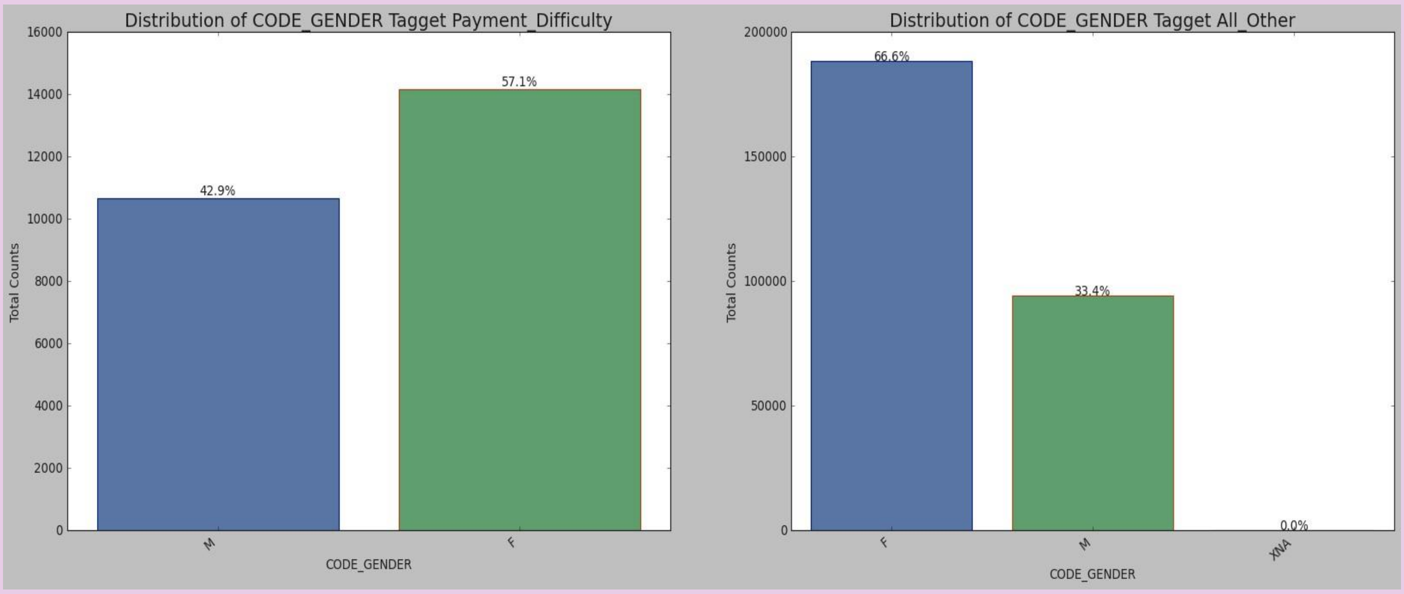
def **Tagget_categorical_Uni**(variable):

plt.style.use('classic')   sns.despine

fig,(ax1,ax2) = plt.subplots(1,2,figsize=(25,8))
sns.countplot(x=variable,data=Tagget_Variable_Payment_Difficulty,line

```
width=1,ax=ax1,edgecolor=sns.color_palette("dark", 3),hue =variable )
ax1.set_ylabel('Total Counts')
    ax1.set_title(f'Distribution of {variable} Tagget
Payment_Difficulty',fontsize=18)
    ax1.set_xticklabels(ax1.get_xticklabels(), rotation=40, ha="right")


    for p in ax1.patches:
ax1.annotate('{:.1f}%'.format((p.get_height()/len(Tagget_Variable_Paym
ent_Difficulty))*100), (p.get_x()+0.4, p.get_height()+100), ha='center')


    sns.countplot(x=variable,
data=Tagget_Variable_All_Other,ax=ax2,linewidth=1,edgecolor=sns.colo
r_palette("dark", 3),hue =variable )     ax2.set_ylabel('Total Counts')
    ax2.set_title(f'Distribution of {variable} Tagget All_Other',fontsize =
18,)
    ax2.set_xticklabels(ax2.get_xticklabels(), rotation=40, ha="right")


    for p in ax2.patches:
ax2.annotate('{:.1f}%'.format((p.get_height()/len(Tagget_Variable_All_O
ther))*100), (p.get_x()+0.4, p.get_height()+100), ha='center')


    plt.show()


Tagget_categorical_Uni('CODE_GENDER'):
```

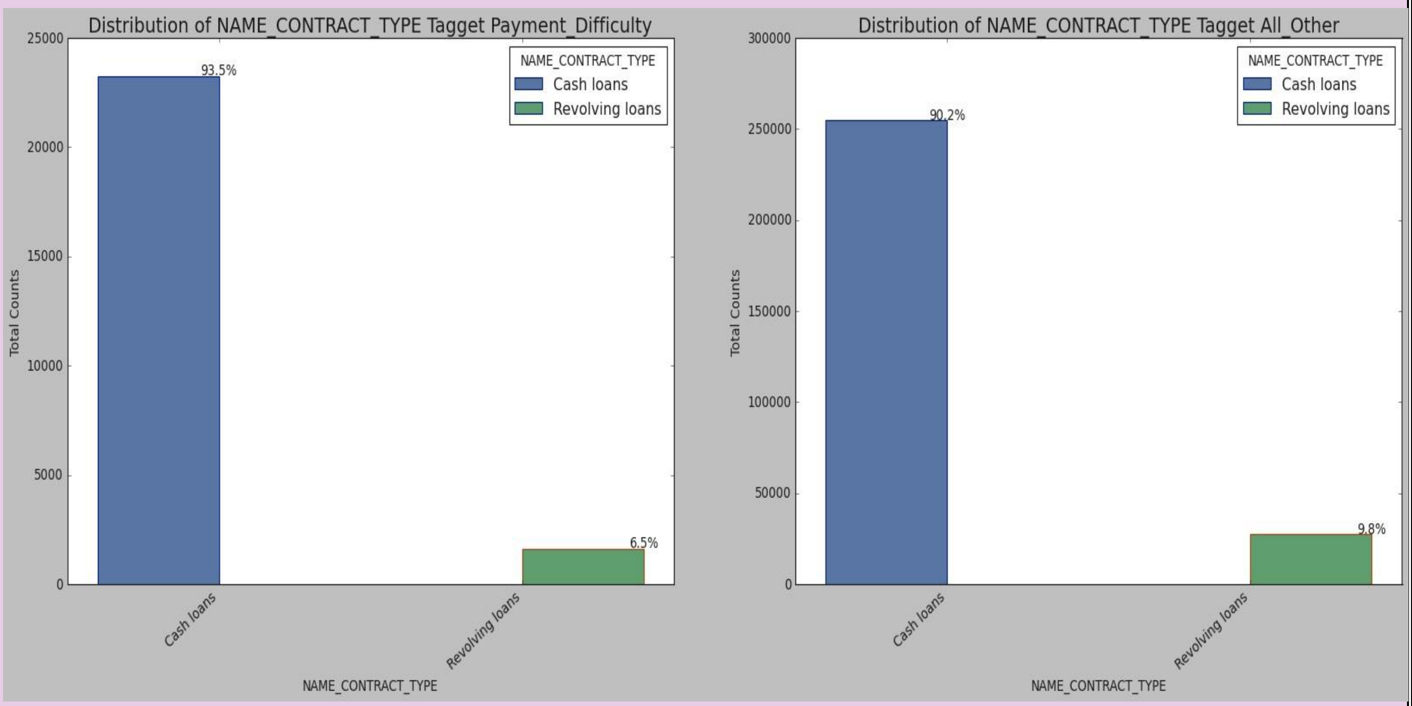
Output:

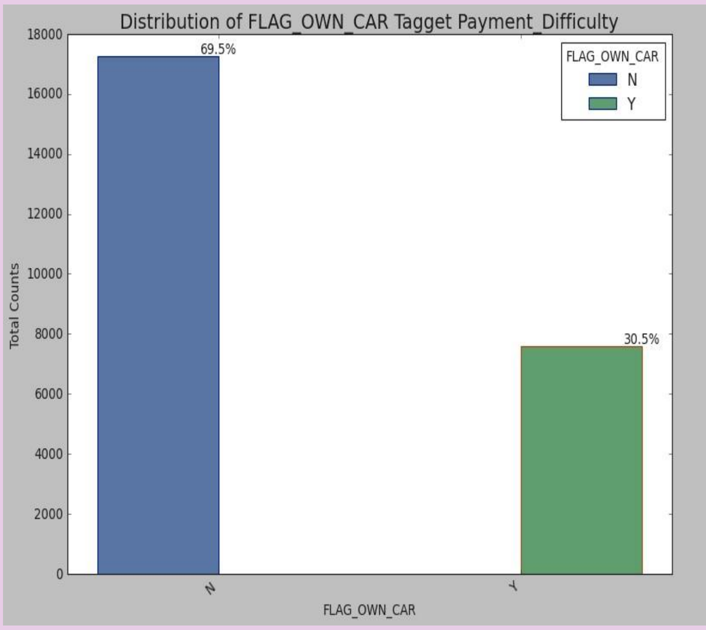


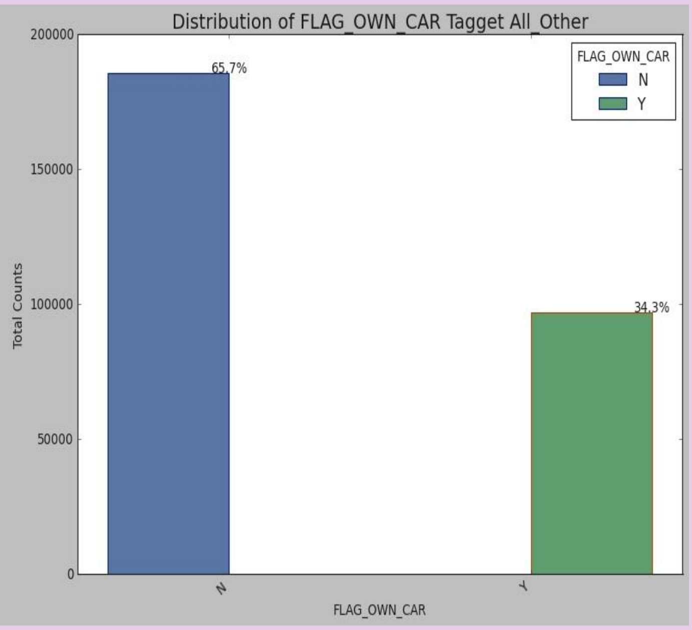
for i in Categorical_Data_1:

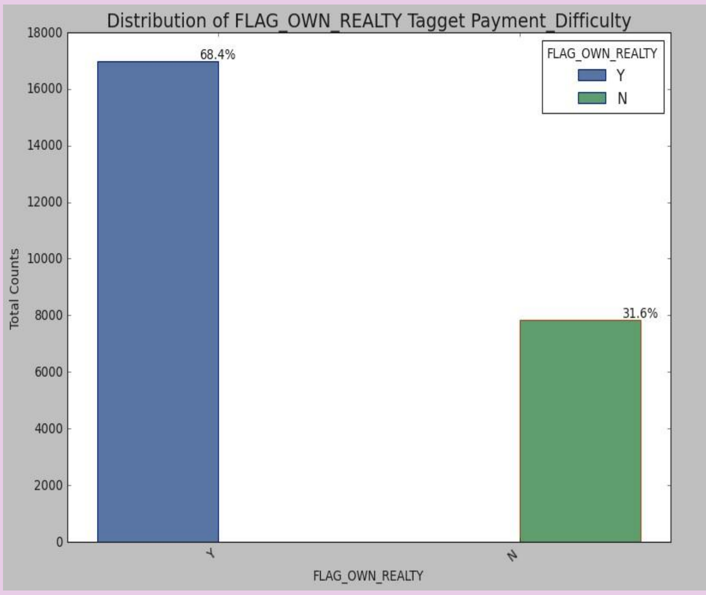
   Tagget_categorical_Uni(i)

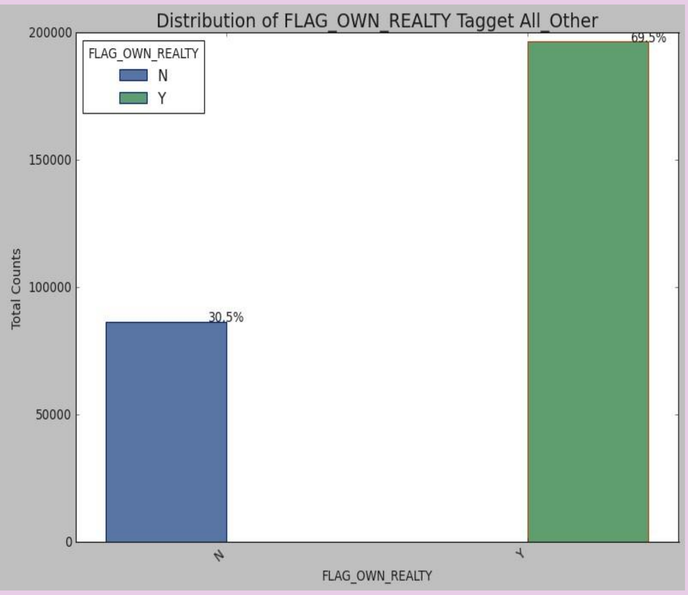
Output:

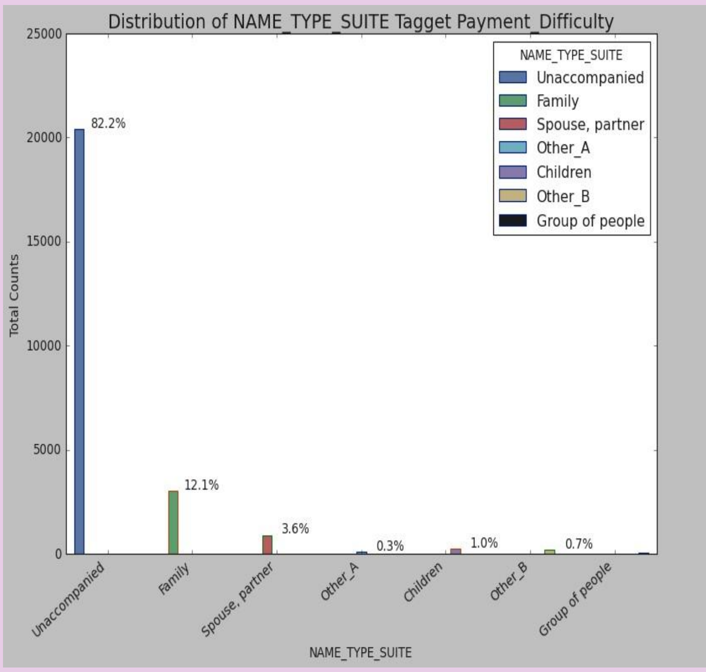


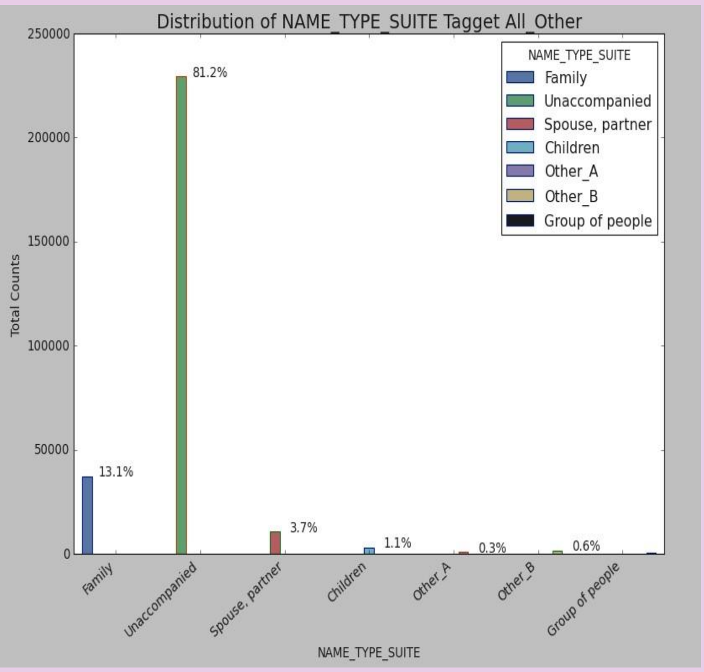

Distribution of FLAG_OWN_CAR Tagget Payment_Difficulty
Total Counts
FLAG_OWN_CAR
N
Y
69.5%
30.5%
FLAG_OWN_CAR

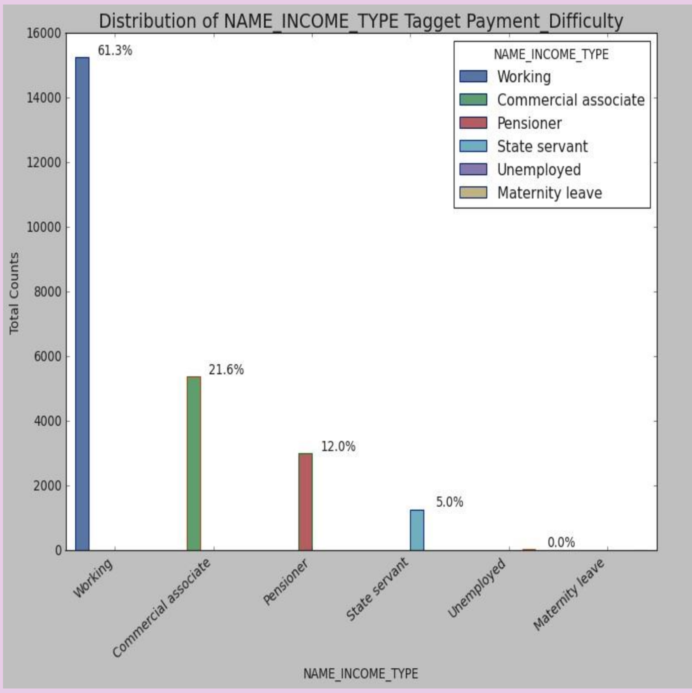

Distribution of FLAG_OWN_CAR Tagget All_Other
Total Counts
FLAG_OWN_CAR
N
Y
65.7%
34.3%
FLAG_OWN_CAR

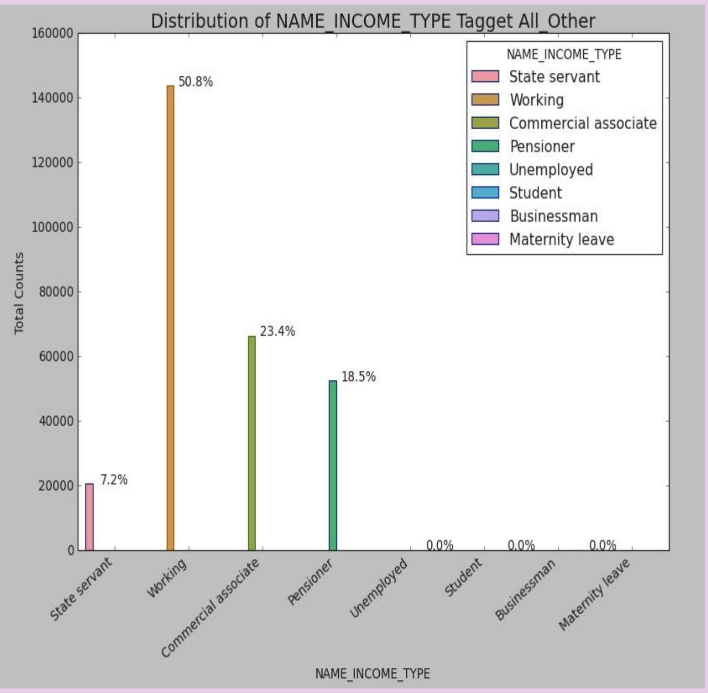

Distribution of FLAG_OWN_REALTY Tagget Payment_Difficulty
Total Counts
FLAG_OWN_REALTY
Y
N
68.4%
31.6%
FLAG_OWN_REALTY

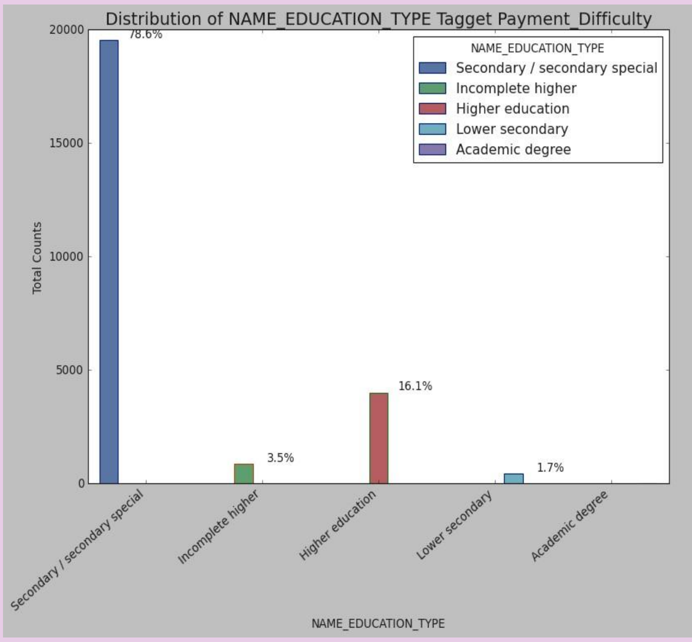

Distribution of FLAG_OWN_REALTY Tagget All_Other
Total Counts
FLAG_OWN_REALTY
N
Y
30.5%
69.5%
FLAG_OWN_REALTY

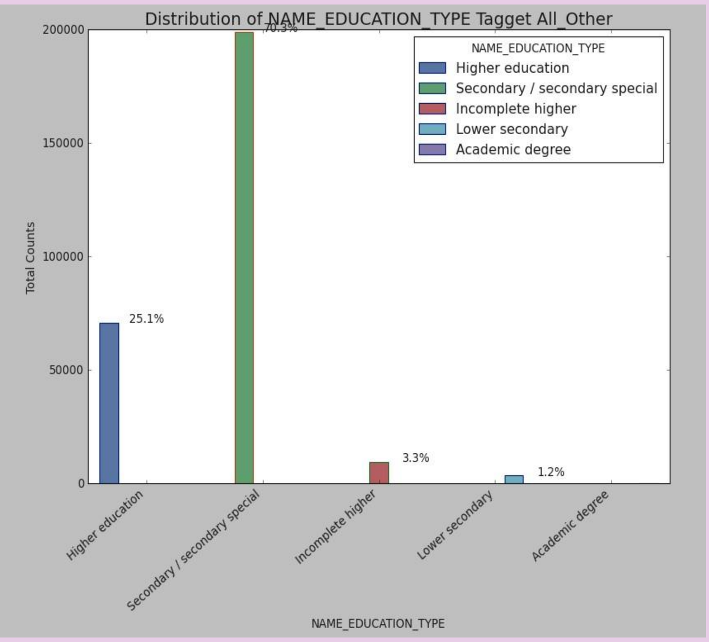

Distribution of NAME_TYPE_SUITE Tagget Payment_Difficulty
Total Counts
NAME_TYPE_SUITE
Unaccompanied
Family
Spouse, partner
Other_A
Children
Other_B
Group of people
82.2%
12.1%
3.6%
0.3%
1.0%
0.7%
NAME_TYPE_SUITE

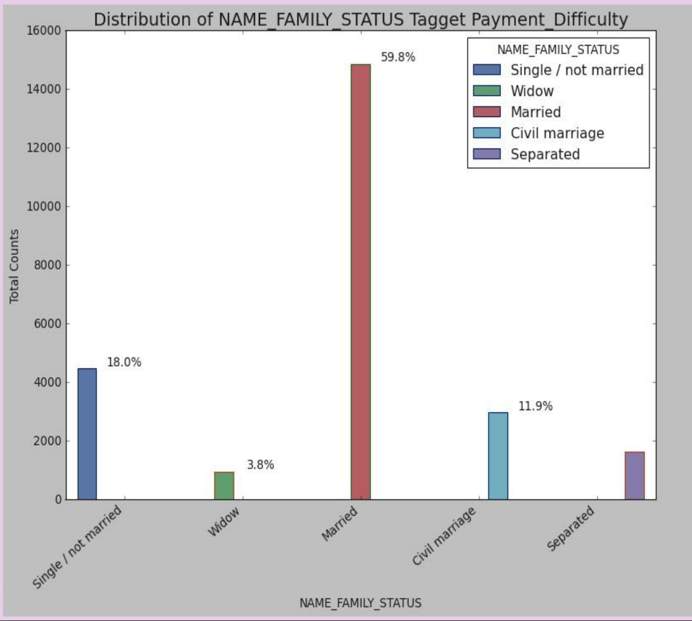

Distribution of NAME_TYPE_SUITE Tagget All_Other
Total Counts
NAME_TYPE_SUITE
Family
Unaccompanied
Spouse, partner
Children
Other_A
Other_B
Group of people
13.1%
81.2%
3.7%
1.1%
0.3%
0.6%
NAME_TYPE_SUITE

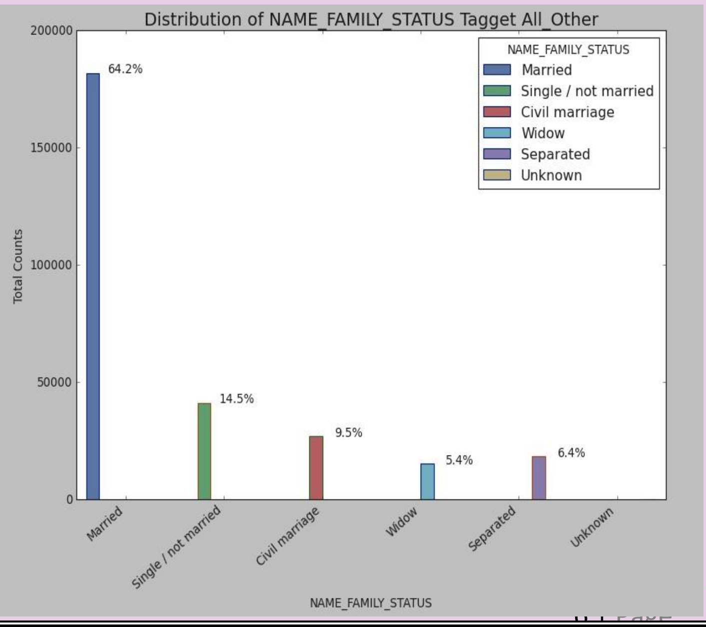

Distribution of NAME_INCOME_TYPE Tagget Payment_Difficulty
NAME_INCOME_TYPE
Working
Commercial associate
Pensioner
State servant
Unemployed
Maternity leave
61.3%
21.6%
12.0%
5.0%
0.0%
Total Counts
NAME_INCOME_TYPE

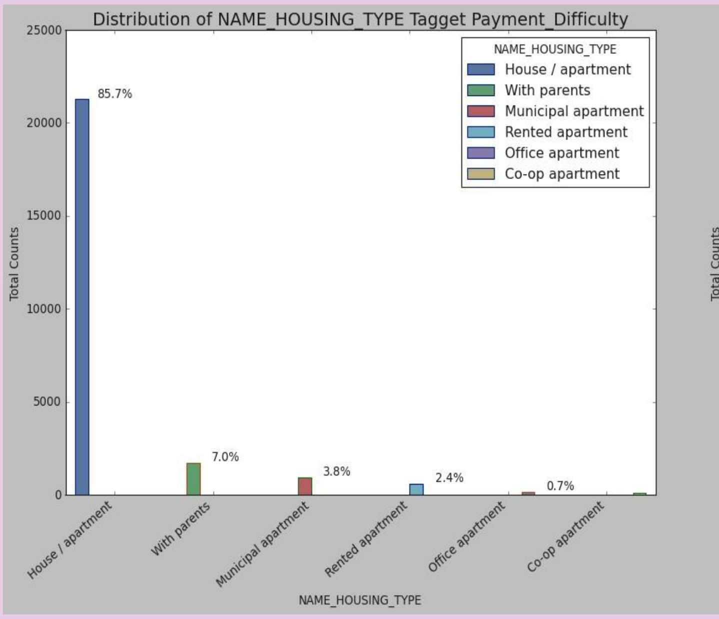

Distribution of NAME_INCOME_TYPE Tagget All_Other
NAME_INCOME_TYPE
State servant
Working
Commercial associate
Pensioner
Unemployed
Student
Businessman
Maternity leave
7.2%
50.8%
23.4%
18.5%
0.0%
0.0%
0.0%
Total Counts
NAME_INCOME_TYPE

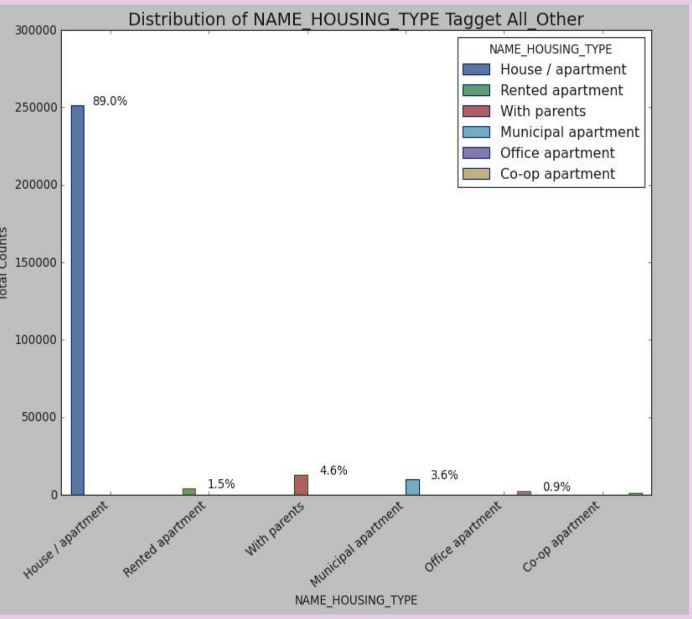

Distribution of NAME_EDUCATION_TYPE Tagget Payment_Difficulty
NAME_EDUCATION_TYPE
Secondary / secondary special
Incomplete higher
Higher education
Lower secondary
Academic degree
78.6%
3.5%
16.1%
1.7%
Total Counts
NAME_EDUCATION_TYPE

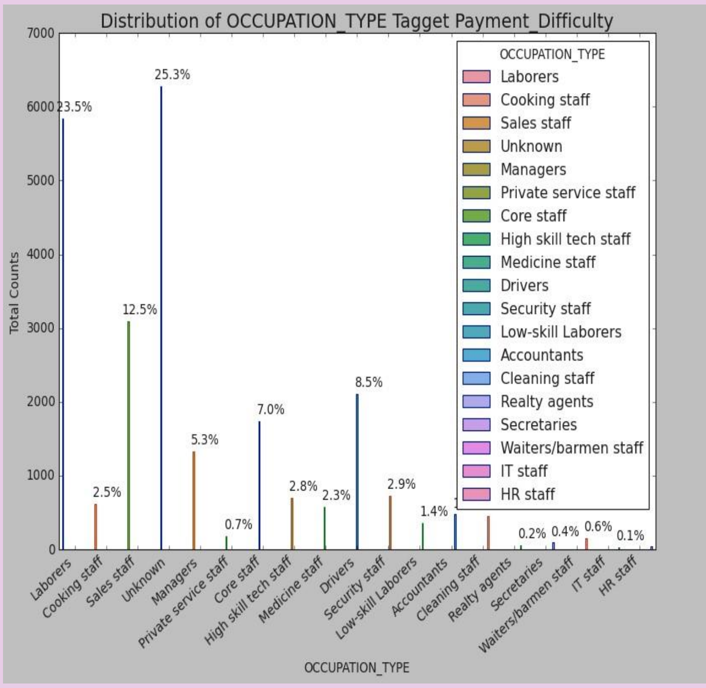

Distribution of NAME_EDUCATION_TYPE Tagget All_Other
NAME_EDUCATION_TYPE
Higher education
Secondary / secondary special
Incomplete higher
Lower secondary
Academic degree
25.1%
70.3%
3.3%
1.2%
Total Counts
NAME_EDUCATION_TYPE

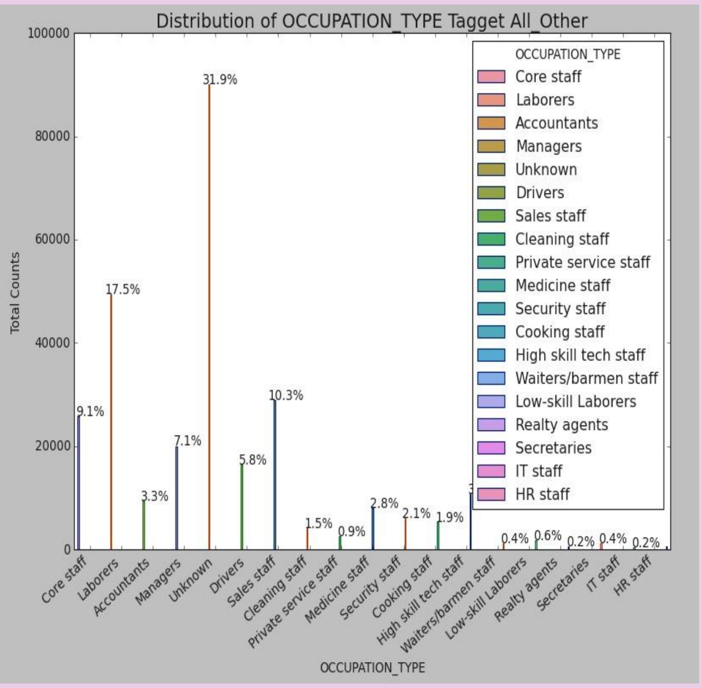

Distribution of NAME_FAMILY_STATUS Tagget Payment_Difficulty
NAME_FAMILY_STATUS
Single / not married
Widow
Married
Civil marriage
Separated
18.0%
3.8%
59.8%
11.9%
Total Counts
NAME_FAMILY_STATUS

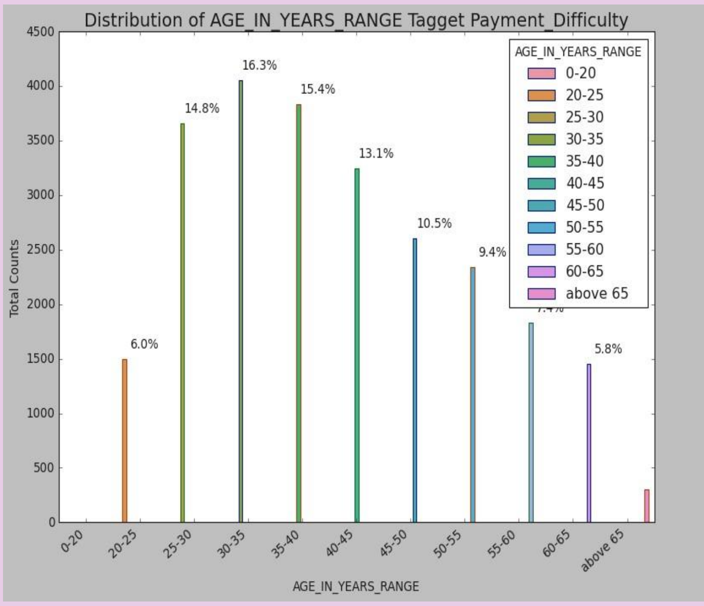

Distribution of NAME_FAMILY_STATUS Tagget All_Other
NAME_FAMILY_STATUS
Married
Single / not married
Civil marriage
Widow
Separated
Unknown
64.2%
14.5%
9.5%
5.4%
6.4%
Total Counts
NAME_FAMILY_STATUS

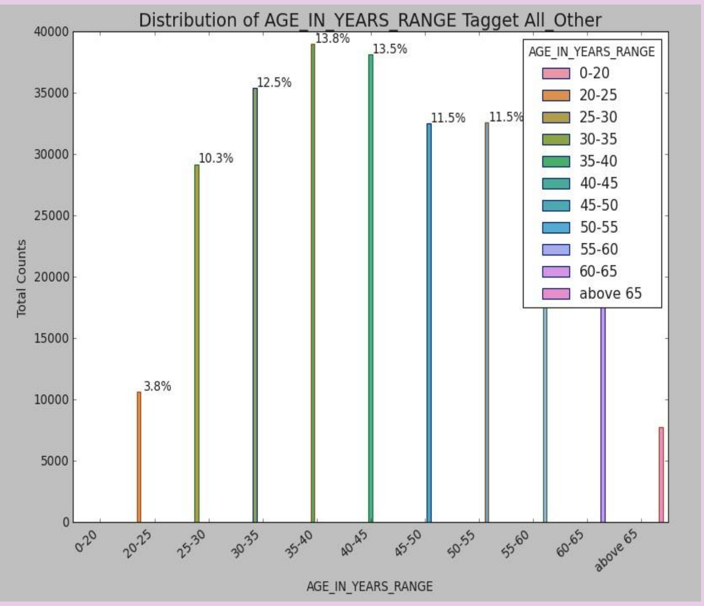

Distribution of NAME_HOUSING_TYPE Tagget Payment_Difficulty
NAME_HOUSING_TYPE
House / apartment
With parents
Municipal apartment
Rented apartment
Office apartment
Co-op apartment
Total Counts
85.7%
7.0%
3.8%
2.4%
0.7%
NAME_HOUSING_TYPE

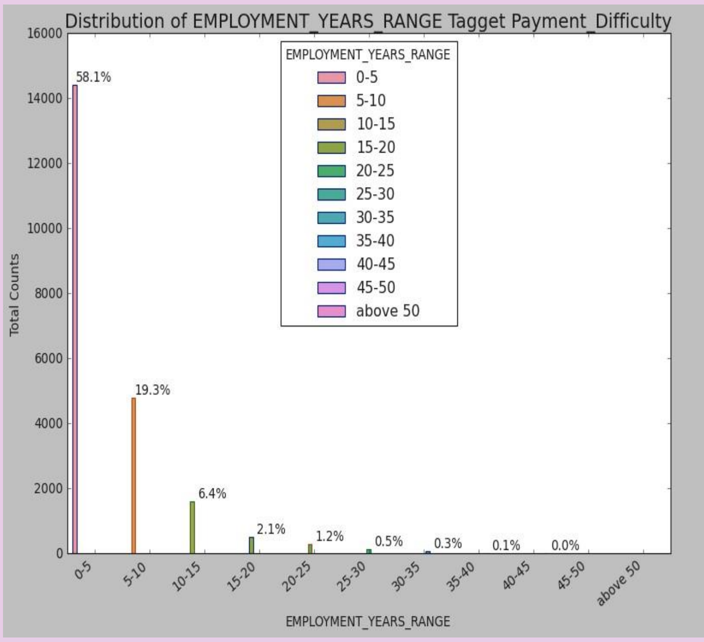

Distribution of NAME_HOUSING_TYPE Tagget All_Other
NAME_HOUSING_TYPE
House / apartment
Rented apartment
With parents
Municipal apartment
Office apartment
Co-op apartment
Total Counts
89.0%
1.5%
4.6%
3.6%
0.9%
NAME_HOUSING_TYPE

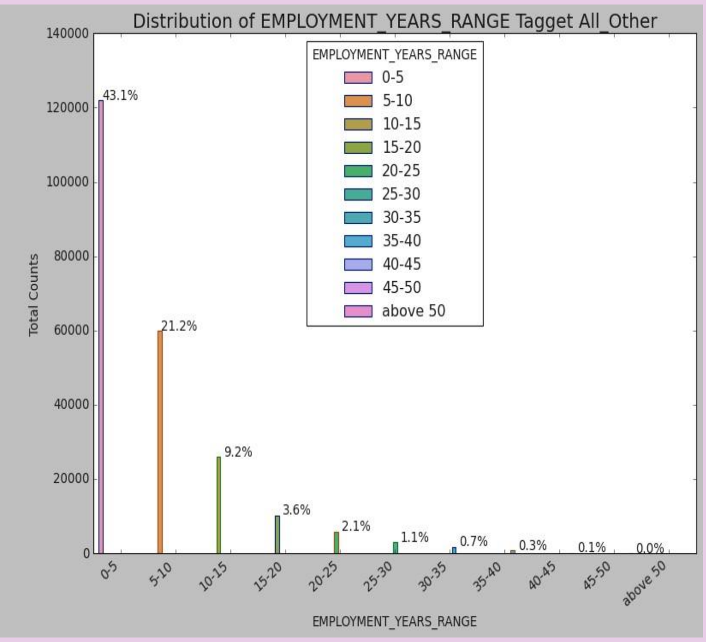

Distribution of OCCUPATION_TYPE Tagget Payment_Difficulty
OCCUPATION_TYPE
Laborers
Cooking staff
Sales staff
Unknown
Managers
Private service staff
Core staff
High skill tech staff
Medicine staff
Drivers
Security staff
Low-skill Laborers
Accountants
Cleaning staff
Realty agents
Secretaries
Waiters/barmen staff
IT staff
HR staff
Total Counts
23.5%
2.5%
12.5%
25.3%
5.3%
0.7%
7.0%
2.8%
2.3%
8.5%
2.9%
1.4%
0.2%
0.4%
0.6%
0.1%
OCCUPATION_TYPE

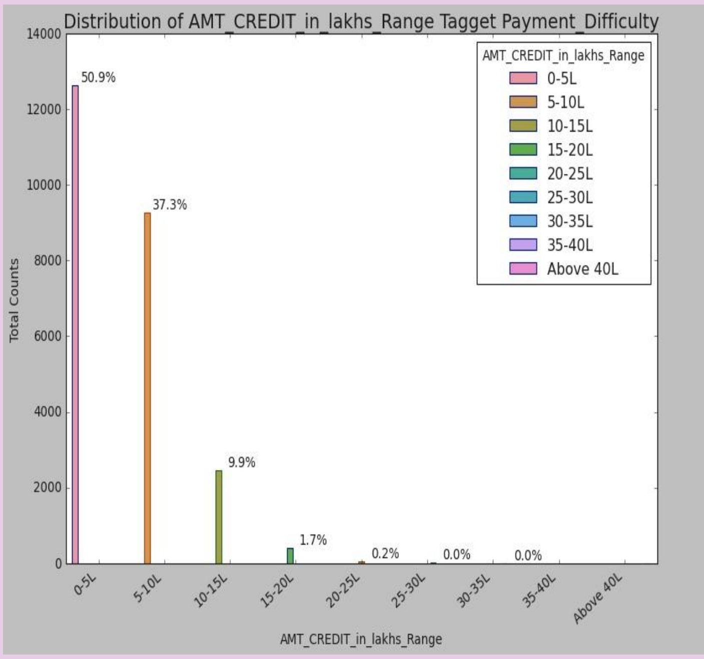

Distribution of OCCUPATION_TYPE Tagget All_Other
OCCUPATION_TYPE
Core staff
Laborers
Accountants
Managers
Unknown
Drivers
Sales staff
Cleaning staff
Private service staff
Medicine staff
Security staff
Cooking staff
High skill tech staff
Waiters/barmen staff
Low-skill Laborers
Realty agents
Secretaries
IT staff
HR staff
Total Counts
9.1%
17.5%
3.3%
7.1%
31.9%
5.8%
10.3%
1.5%
0.9%
2.8%
2.1%
1.9%
0.4%
0.6%
0.2%
0.4%
0.2%
OCCUPATION_TYPE

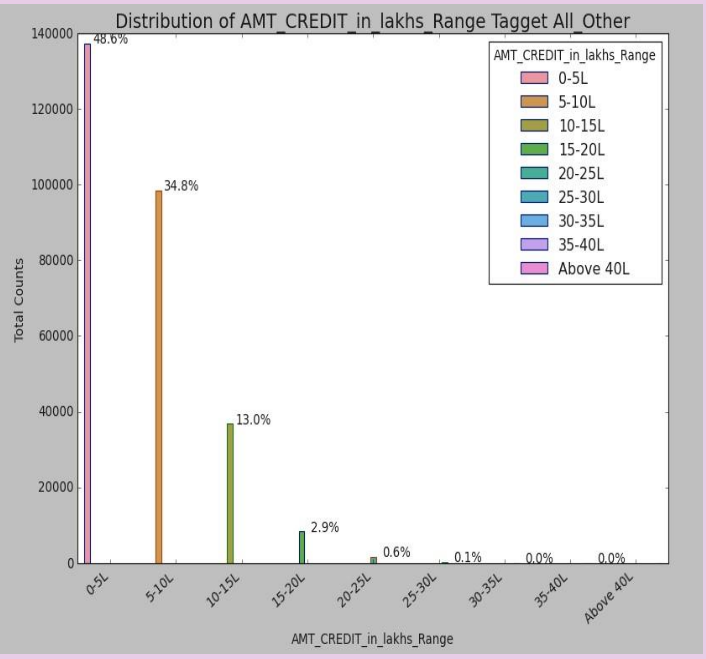

Distribution of AGE_IN_YEARS_RANGE Tagget Payment_Difficulty
Total Counts
AGE_IN_YEARS_RANGE
0-20
20-25
25-30
30-35
35-40
40-45
45-50
50-55
55-60
60-65
above 65
6.0%
14.8%
16.3%
15.4%
13.1%
10.5%
9.4%
5.8%

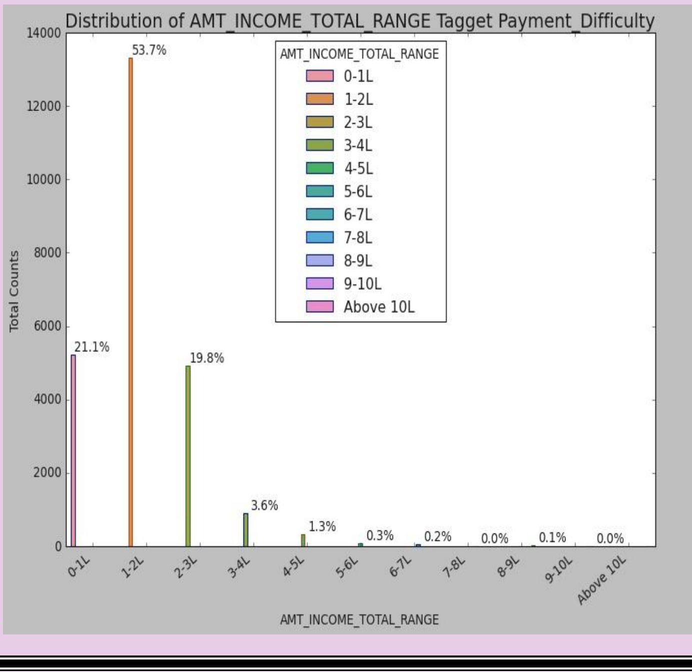

Distribution of AGE_IN_YEARS_RANGE Tagget All_Other
Total Counts
AGE_IN_YEARS_RANGE
0-20
20-25
25-30
30-35
35-40
40-45
45-50
50-55
55-60
60-65
above 65
3.8%
10.3%
12.5%
13.8%
13.5%
11.5%
11.5%

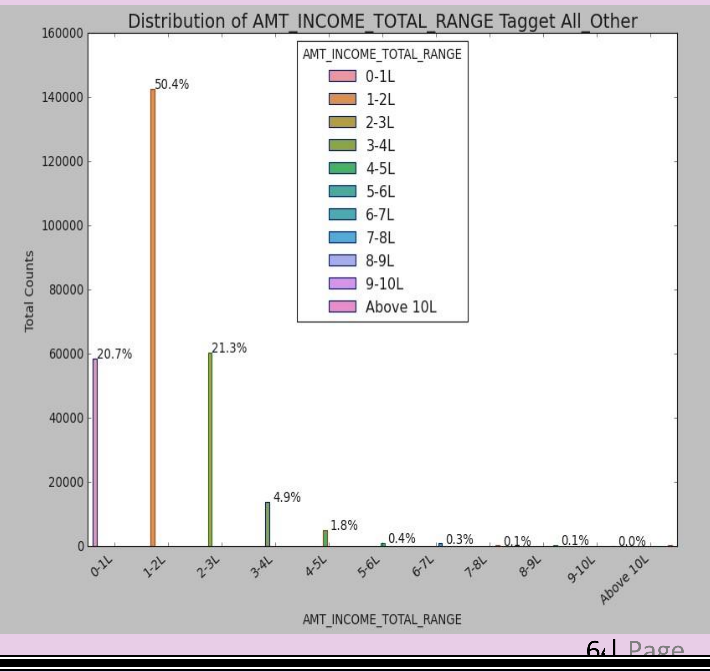

Distribution of EMPLOYMENT_YEARS_RANGE Tagget Payment_Difficulty
Total Counts
EMPLOYMENT_YEARS_RANGE
0-5
5-10
10-15
15-20
20-25
25-30
30-35
35-40
40-45
45-50
above 50
58.1%
19.3%
6.4%
2.1%
1.2%
0.5%
0.3%
0.1%
0.0%


Distribution of EMPLOYMENT_YEARS_RANGE Tagget All_Other
Total Counts
EMPLOYMENT_YEARS_RANGE
0-5
5-10
10-15
15-20
20-25
25-30
30-35
35-40
40-45
45-50
above 50
43.1%
21.2%
9.2%
3.6%
2.1%
1.1%
0.7%
0.3%
0.1%
0.0%


Distribution of AMT_CREDIT_in_lakhs_Range Tagget Payment_Difficulty
Total Counts
AMT_CREDIT_in_lakhs_Range
0-5L
5-10L
10-15L
15-20L
20-25L
25-30L
30-35L
35-40L
Above 40L
50.9%
37.3%
9.9%
1.7%
0.2%
0.0%
0.0%


Distribution of AMT_CREDIT_in_lakhs_Range Tagget All_Other
Total Counts
AMT_CREDIT_in_lakhs_Range
0-5L
5-10L
10-15L
15-20L
20-25L
25-30L
30-35L
35-40L
Above 40L
48.6%
34.8%
13.0%
2.9%
0.6%
0.1%
0.0%
0.0%

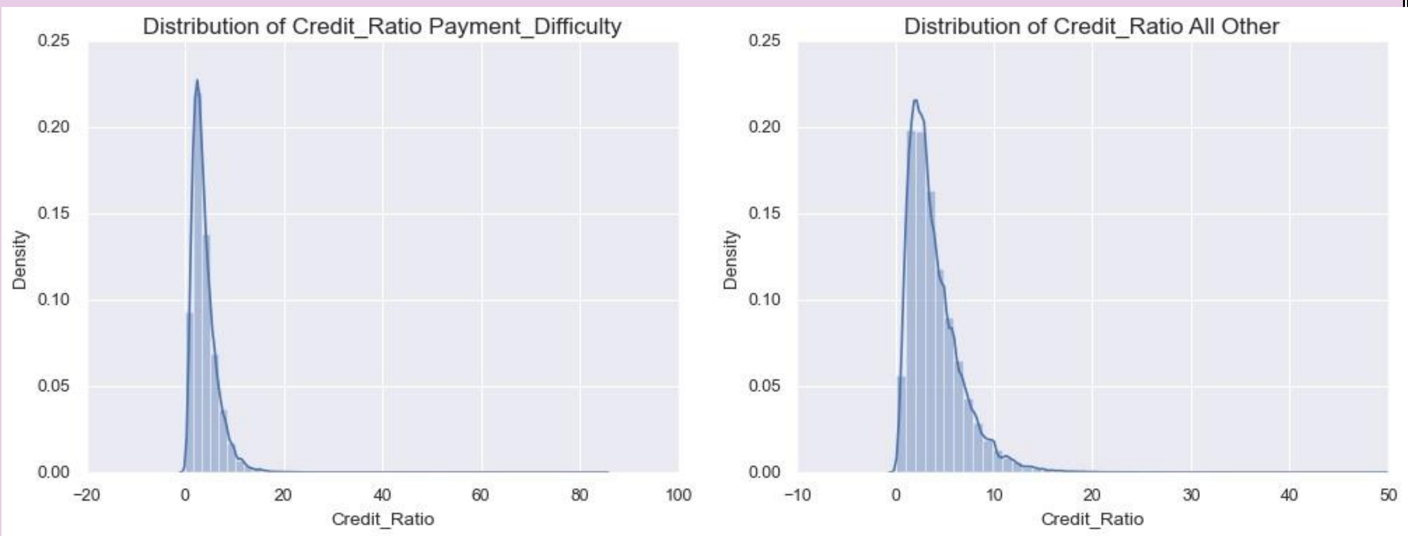

Distribution of AMT_INCOME_TOTAL_RANGE Tagget Payment_Difficulty
Total Counts
AMT_INCOME_TOTAL_RANGE
0-1L
1-2L
2-3L
3-4L
4-5L
5-6L
6-7L
7-8L
8-9L
9-10L
Above 10L
21.1%
53.7%
19.8%
3.6%
1.3%
0.3%
0.2%
0.0%
0.1%
0.0%


Distribution of AMT_INCOME_TOTAL_RANGE Tagget All_Other
Total Counts
AMT_INCOME_TOTAL_RANGE
0-1L
1-2L
2-3L
3-4L
4-5L
5-6L
6-7L
7-8L
8-9L
9-10L
Above 10L
20.7%
50.4%
21.3%
4.9%
1.8%
0.4%
0.3%
0.1%
0.1%
0.0%

Function to plot for categorical variables:

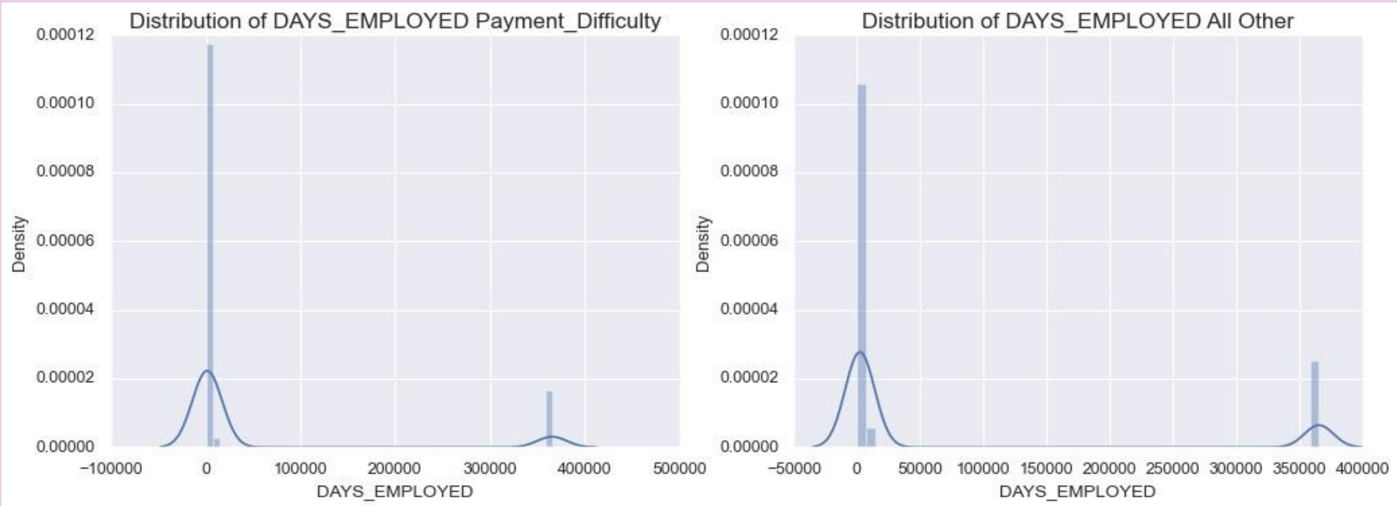
```
def Tagget_Numarical_Uni(variable):

    sns.set(style='darkgrid')
plt.figure(figsize=(15, 5))

    plt.subplot(1, 2, 1)
    sns.distplot(Tagget_Variable_Payment_Difficulty[variable].dropna())
plt.title(f'Distribution of {variable} Payment_Difficulty',fontsize=15)
plt.xlabel(variable)

    plt.subplot(1, 2, 2)
    sns.distplot(Tagget_Variable_All_Other[variable].dropna())
plt.title(f'Distribution of {variable} All Other',fontsize=15)
plt.xlabel(variable)

    plt.show()



Tagget_Numarical_Uni('Credit_Ratio')
```

Output:



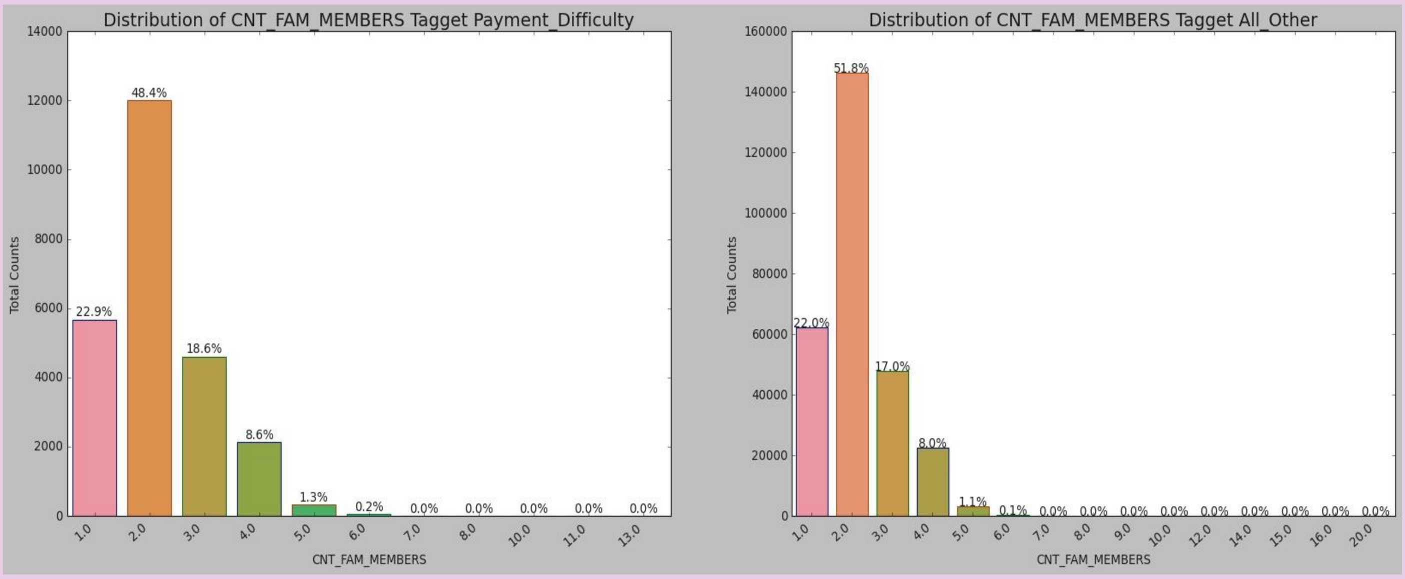
Tagget_Numarical_Uni('DAYS_EMPLOYED')

Output:



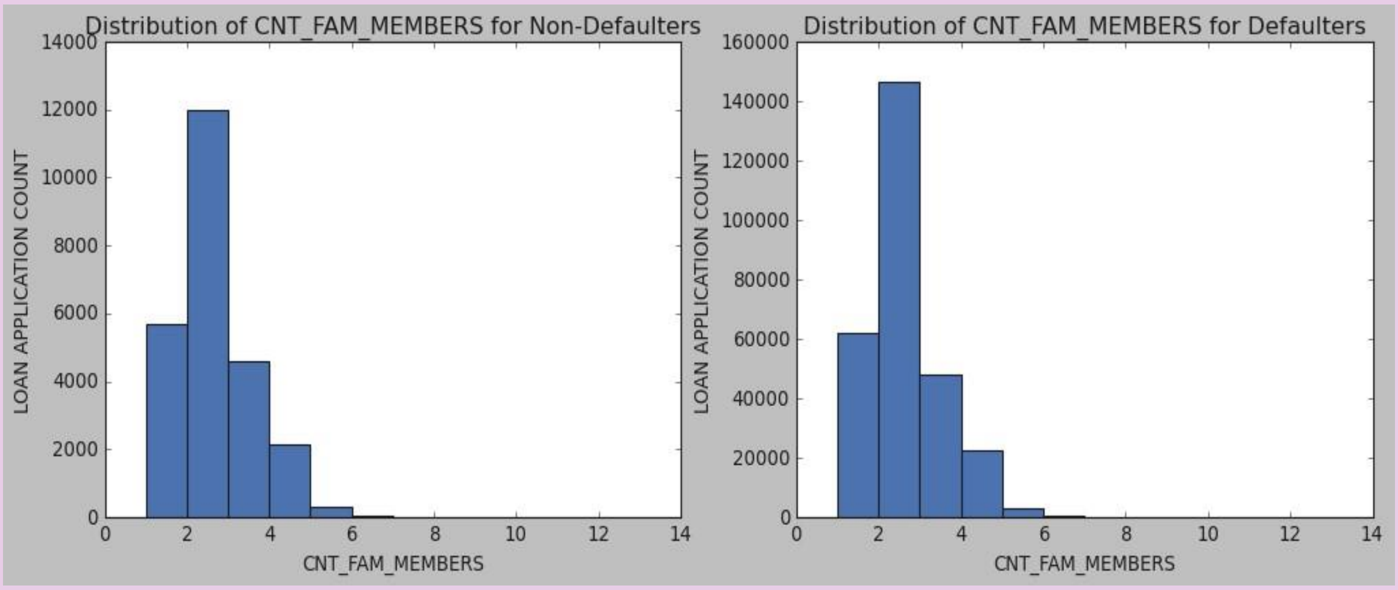
Tagget_categorical_Uni('CNT_FAM_MEMBERS')

Output:




```
plt.figure(figsize=(15,5))

plt.subplot(1, 2, 1)
Tagget_Variable_Payment_Difficulty['CNT_FAM_MEMBERS'].plot.hist(
bins=range(15))
plt.title('Distribution of CNT_FAM_MEMBERS for
NonDefaulters',fontsize=15)
plt.xlabel('CNT_FAM_MEMBERS')

plt.subplot(1, 2, 2)
Tagget_Variable_All_Other['CNT_FAM_MEMBERS'].plot.hist(bins=ran
ge(15))
plt.title(f'Distribution of CNT_FAM_MEMBERS for
Defaulters',fontsize=15)
plt.xlabel('CNT_FAM_MEMBERS')

plt.show()
```

Output:



Bivariate Analysis of TARGET

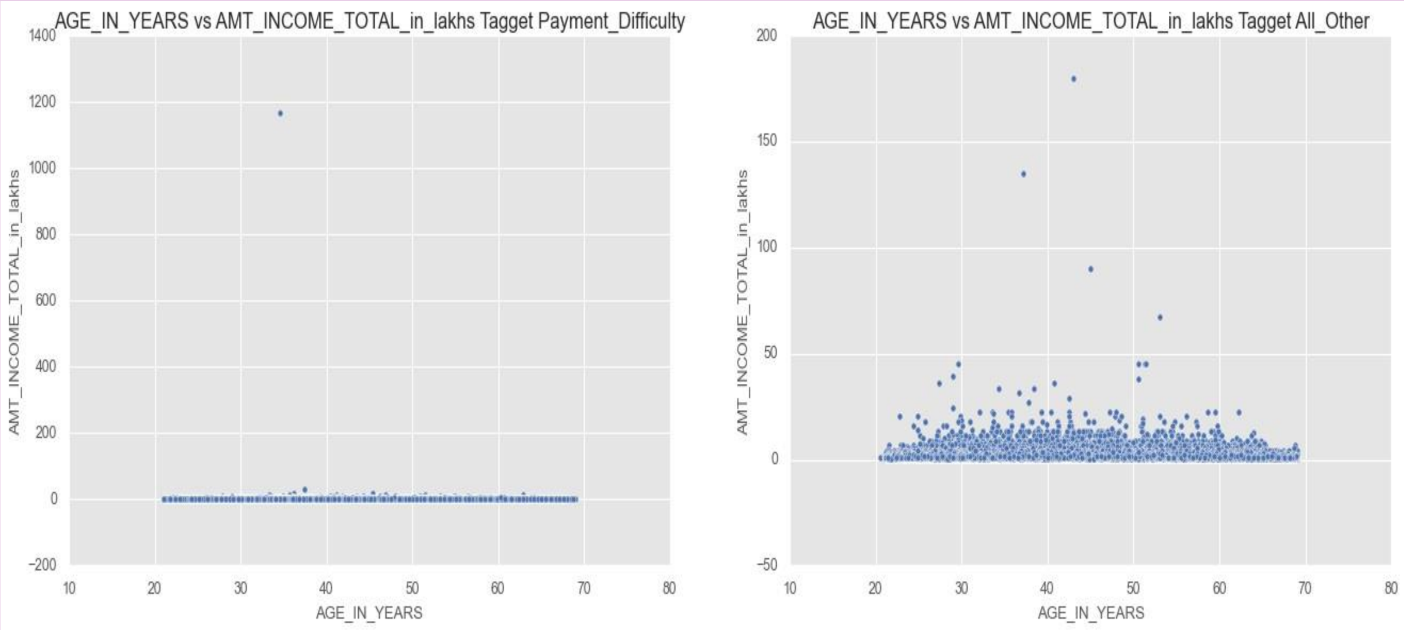
sns.scatterplot(x=Tagget_Variable_Payment_Difficulty.AMT_ANNUITY, y = Tagget_Variable_Payment_Difficulty.AMT_GOODS_PRICE, data=Tagget_Variable_Payment_Difficulty,hue = 'TARGET')

Output:


```
def Tagget_Numarical_Bi(variable_1, variable_2):
    # other thems of plot seaborn-colorblind,seaborn-dark-palette ,classic

    plt.style.use('ggplot')
    sns.despine
    fig,(ax1,ax2) = plt.subplots(1,2,figsize=(20,6))

    sns.scatterplot(x=variable_1,
y=variable_2,data=Tagget_Variable_Payment_Difficulty,ax=ax1)
ax1.set_xlabel(variable_1)        ax1.set_ylabel(variable_2)
    ax1.set_title(f'{variable_1} vs {variable_2} Tagget
Payment_Difficulty',fontsize=15)

    sns.scatterplot(x=variable_1,
y=variable_2,data=Tagget_Variable_All_Other,ax=ax2)
ax2.set_xlabel(variable_1)        ax2.set_ylabel(variable_2)
    ax2.set_title(f'{variable_1} vs {variable_2} Tagget
All_Other',fontsize=15)
```

    plt.show()
Tagget_Numarical('AGE_IN_YEARS','AMT_INCOME_TOTAL_in_lakhs')

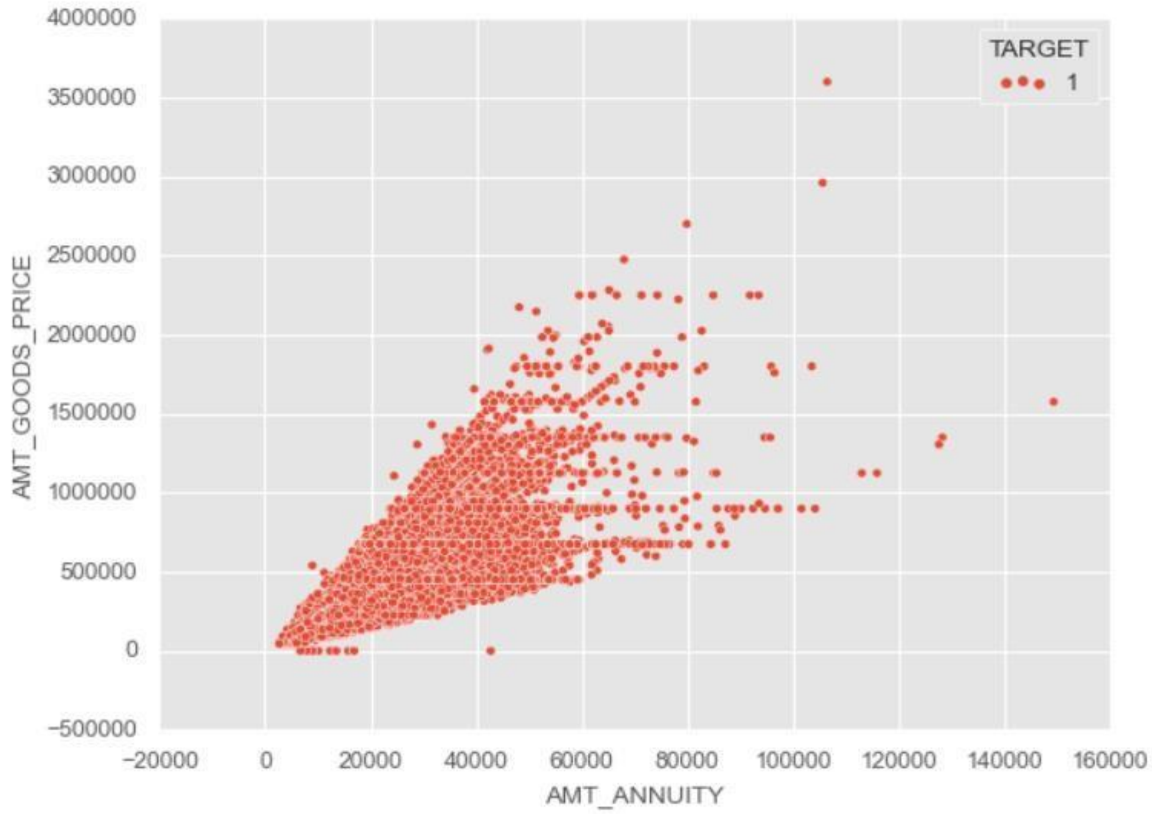
Output:



sns.scatterplot(x=Tagget_Variable_Payment_Difficulty.AMT_ANNUITY, y = Tagget_Variable_Payment_Difficulty.AMT_GOODS_PRICE, data=Tagget_Variable_Payment_Difficulty,hue = 'TARGET')

Output:

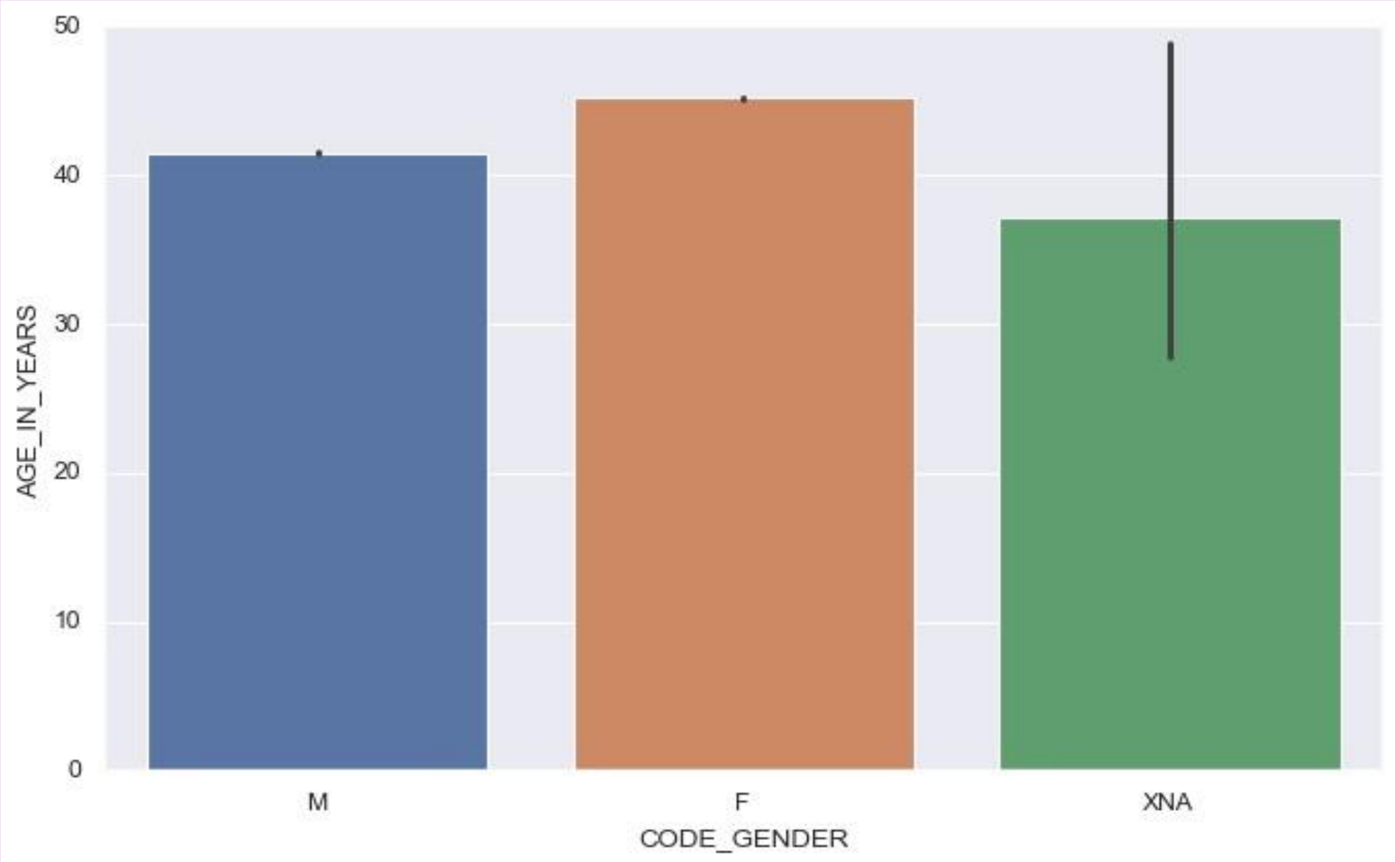
<AxesSubplot:xlabel='AMT_ANNUITY', ylabel='AMT_GOODS_PRICE'>



plt.figure(figsize = [10,6]) sns.set(style='darkgrid')
sns.barplot(x = df_application.CODE_GENDER,y = df_application.AGE_IN_YEARS)

plt.show()

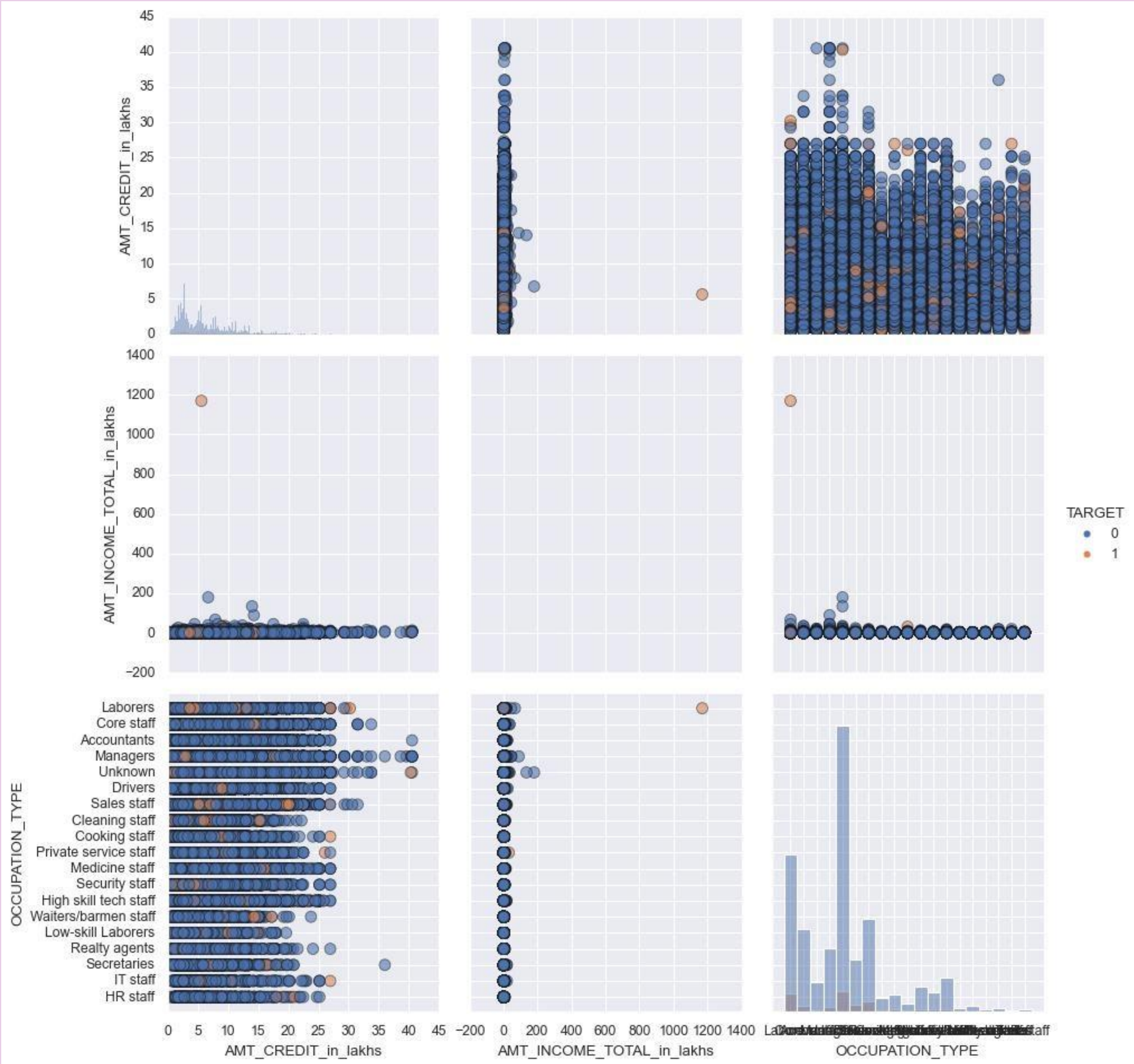
Output:



sns.pairplot(df_application,vars = ['AMT_CREDIT_in_lakhs','AMT_INCOME_TOTAL_in_lakhs','OCCUPATION_TYPE'],diag_kind = 'hist', hue = 'TARGET',plot_kws = {'alpha': 0.6, 's': 80, 'edgecolor': 'k'},size = 4)

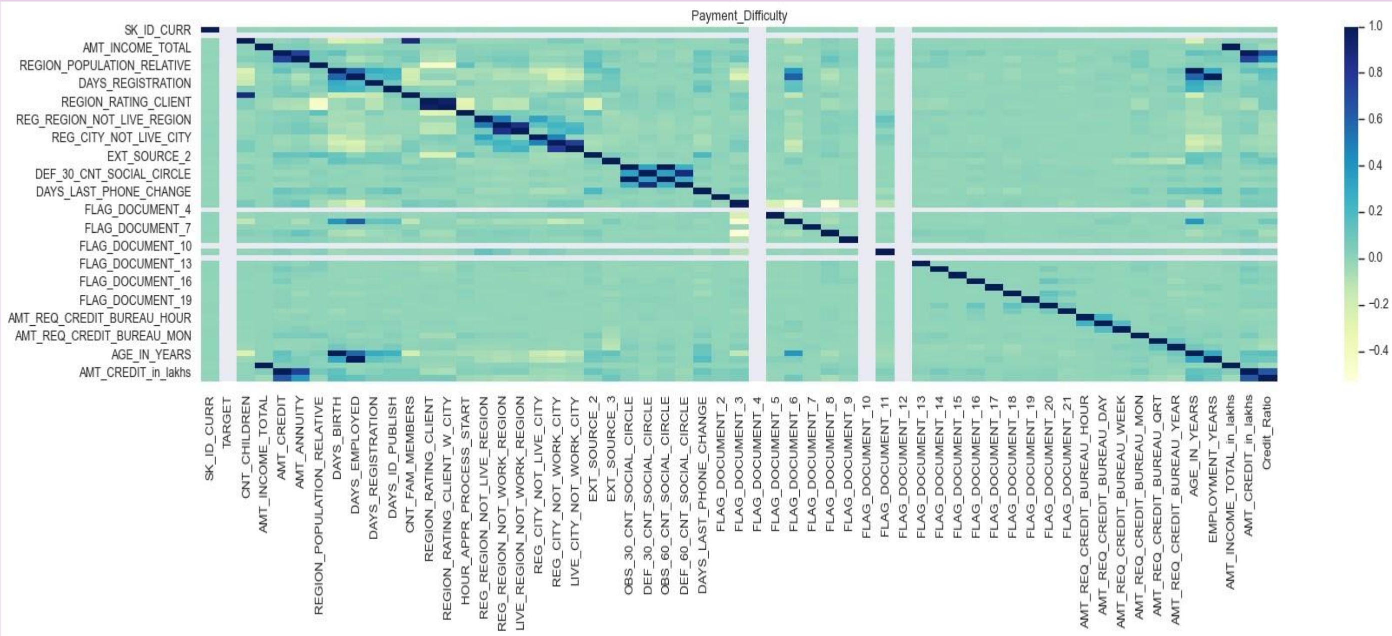
plt.show()

Output:




```
plt.figure(figsize=(25, 5))
sns.heatmap(Tagget_Variable_Payment_Difficulty.corr(),cmap="YlGnBu
")
plt.title('Payment_Difficulty')

plt.show()
```

Output:



df_3=Tagget_Variable_Payment_Difficulty[['CNT_CHILDREN','AMT_ANNUITY','AMT_GOODS_PRICE','AGE_IN_YEARS','EMPLOYMENT_YEARS','AMT_INCOME_TOTAL_in_lakhs','AMT_CREDIT_in_lakhs','CNT_FAM_MEMBERS','FLAG_OWN_CAR','FLAG_OWN_REALTY','NAME_TYPE_SUITE','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE','NAME_FAMILY_STATUS','OCCUPATION_TYPE','NAME_HOUSING_TYPE']]

plt.figure(figsize=(25, 5))
sns.heatmap(df_3.corr(method = 'pearson'),cmap = 'YlGnBu', annot=True)
plt.title('Payment_Difficulty')

plt.show()

Output:


Payment_Difficulty
CNT_CHILDREN 1 0.031 -0.26 -0.033 0.0048 -0.0017 0.89
AMT_ANNUITY 0.031 1 0.014 0.049 0.046 0.75 0.076
AGE_IN_YEARS -0.26 0.014 1 0.31 -0.0031 0.14 -0.2
EMPLOYMENT_YEARS -0.033 0.049 0.31 1 4.6e-05 0.11 0.0016
AMT_INCOME_TOTAL_in_lakhs 0.0048 0.046 -0.0031 4.6e-05 1 0.038 0.0067
AMT_CREDIT_in_lakhs -0.0017 0.75 0.14 0.11 0.038 1 0.051
CNT_FAM_MEMBERS 0.89 0.076 -0.2 0.0016 0.0067 0.051 1
CNT_CHILDREN AMT_ANNUITY AGE_IN_YEARS EMPLOYMENT_YEARS AMT_INCOME_TOTAL_in_lakhs AMT_CREDIT_in_lakhs CNT_FAM_MEMBERS
0.90
0.75
0.60
0.45
0.30
0.15
0.00
-0.15