Problem Statement:

A consumer finance company specializes in lending various types of loans to urban customers. However, the company finds it hard to give loans to people with insufficient or non-existent credit history. Because of that, some consumers use it as their advantage by becoming a defaulter. The company needs to minimize the risk of losing money while lending to customers. When the company receives a loan application, the company has to decide for loan approval based on the applicant's profile. Two types of risks are associated with the bank's decision: • If the applicant is likely to repay the loan, then not approving the loan results in a loss of business to the company. • If the applicant is not likely to repay the loan, i.e. he/she is likely to default, then approving the loan may lead to a financial loss for the company.The data contains 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 aim is to identify patterns that indicate if a client has
difficulty paying their installments, which may be used for taking
actions such as denying the loan, reducing the amount of loan,
lending (to risky applicants) at a higher interest rate, etc. This
will ensure that the consumers capable of repaying the loan are not
rejected. Identification of such applicants using EDA is the aim of
this case study.
```

In [ ]:
```
# Analysis Approach:
we   analysis perform on three dataset application_data,columns_description,previous_a
We will perform exploratory data analysis (EDA) to understand the data, identify patte
  Data understanding:
      i)Importing correct libraries.
     ii)Checking each column, Index, header, footer etc .
    iii)Identifying data quality issues.

  Data Cleaning and Manipulation:
      i)Missing value imputation analysis.
     ii)Checking the structure and the metadata.
    iii)Changing datatypes to date, time, string, int, bool, etc for ease of
        analysis.

  Outlier check and data imbalance check:
     i)Checking the data for outliers that would cause the analysis biased
       to be
    ii)Checking for imbalances, ratio, percentage imbalance

  Data analysis:
    i)Business requirement-oriented analysis.
   ii)Correlation between columns.
  iii)Univariate analysis.
   iv)Bivariate analysis.
    v)Creating plots to understand the data better and find insight.
```

```
Conclusion and Presentation:
  i)Explains Business driven, type driven and data-driven metrics created
    in previous steps
 ii)Helps in getting better understanding about the data.
```

Tech-Stack Used:

For this project, we used Jupyter Notebook as the coding environment and Python libraries such as Pandas, Matplotlib, and Seaborn for data analysis and visualization.

Insights:

1.Data Cleaning: The initial data cleaning was done by checking for null values, duplicate records, and missing values. The
missing values were replaced by appropriate measures like mean, mode, or median. 2.Data Exploration: Exploratory data analysis was done by analyzing the distribution of variables, correlation between
variables, and creating visualizations using libraries like Matplotlib and Seaborn. 3.Data Preprocessing: Feature engineering was done to create new features from existing ones, and the categorical variables were encoded using label encoding or one-hot encoding. 4.Modeling: The data was split into training and testing sets, and various machine learning models were applied, including Logistic Regression, Random Forest, and XGBoost. The best model was selected based on the accuracy score. 5.Model Evaluation: The model was evaluated using various metrics like accuracy, precision, recall, and F1 score, and the model with the best performance was selected for prediction. 6.Prediction: The final model was used to predict the loan default probability for new customers, and the model's performance was validated by comparing the predicted and actual default rates.

Overall, the project provides a detailed analysis of the loan application data, explores the relationship between various features and loan default, and creates a predictive model to identify the customers who are at high risk of defaulting on their loans. The project is useful for banks and financial institutions to improve their loan approval process and reduce the risk of default.

Result:

```
Chances of client having payment difficulty
```

• All the below variables were established in analysis of Application dataframe as leading to default. Checked these against
the Approved loans which have defaults, and it proves to be correct • Medium income • 25-35 years olds, followed by 35-45 years age group • Male • Unemployed • Labourers, Salesman,

Drivers • Own House - No • Other IMPORTANT Factors to be considered • No of Bureau Hits in last week. Month etc – zero hits is good • Amount income not correspondingly equivalent to Good Bought – Income 'Low' and 'High' is a concern • Previous applications with Refused, Cancelled, Unused loans also have default which is a matter of concern. • This indicates that the financial company had Refused/Cancelled previous application but has approved the current and is facing default on these. • Credible Applications refused • Unused applications have lower loan amount. Is this the reason for no usage? • Female applicants should be given extra weightage as defaults are lesser. • Students and Business mean have no problem in repayment of the loan • Previous applications with Refused, Cancelled, Unused loans also have cases where payments are coming on time in current application. This indicates that possibly wrong decisions were done in those cases.

Indentify 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.

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
import warnings   #Filtering out the warnings
warnings.filterwarnings('ignore')
```

Load the Data

In [ ]:
```
# reading the application_data.csv file to dataframe using read_csv
app_data=pd.read_csv('D:\\New folder\\application_data.csv')

# reading the previous_application.csv file to dataframe using read_csv
pre_data = pd.read_csv('D:\\New folder\\previous_application.csv')
```

Checking the structure of the data

In [ ]:
```
#check the top five rows
app_data.head()
```

Out[ ]:

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

5 rows × 122 columns

Observations: The header row looks fine

In [ ]:
```
# check the bottom 5 rows
app_data.tail()
```

Out[ ]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_
307506	456251	0	Cash loans	M	N	
307507	456252	0	Cash loans	F	N	
307508	456253	0	Cash loans	F	N	
307509	456254	1	Cash loans	F	N	
307510	456255	0	Cash loans	F	N	

5 rows × 122 columns

Observations: The bottom rows looks fine. There are no junk values like page number and NaN values in bottom most row

In [ ]:
```
# check total no of rows and columns
app_data.shape
```

Out[ ]:
```
(307511, 122)
```

Observations: Dataframe has 307511 rows and 122 columns

3.Dealing with incorrect data types - app_data

In [ ]:
```
# checking the info of the dataframe
app_data.info(null_counts=True, verbose=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
 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   OWN_CAR_AGE                     104582 non-null  float64
 22   FLAG_MOBIL                      307511 non-null  int64
 23   FLAG_EMP_PHONE                  307511 non-null  int64
 24   FLAG_WORK_PHONE                 307511 non-null  int64
 25   FLAG_CONT_MOBILE                307511 non-null  int64
 26   FLAG_PHONE                      307511 non-null  int64
 27   FLAG_EMAIL                      307511 non-null  int64
 28   OCCUPATION_TYPE                 211120 non-null  object
 29   CNT_FAM_MEMBERS                 307509 non-null  float64
 30   REGION_RATING_CLIENT            307511 non-null  int64
 31   REGION_RATING_CLIENT_W_CITY     307511 non-null  int64
 32   WEEKDAY_APPR_PROCESS_START      307511 non-null  object
 33   HOUR_APPR_PROCESS_START         307511 non-null  int64
 34   REG_REGION_NOT_LIVE_REGION      307511 non-null  int64
 35   REG_REGION_NOT_WORK_REGION      307511 non-null  int64
 36   LIVE_REGION_NOT_WORK_REGION     307511 non-null  int64
 37   REG_CITY_NOT_LIVE_CITY          307511 non-null  int64
 38   REG_CITY_NOT_WORK_CITY          307511 non-null  int64
 39   LIVE_CITY_NOT_WORK_CITY         307511 non-null  int64
 40   ORGANIZATION_TYPE               307511 non-null  object
 41   EXT_SOURCE_1                    134133 non-null  float64
 42   EXT_SOURCE_2                    306851 non-null  float64
 43   EXT_SOURCE_3                    246546 non-null  float64
 44   APARTMENTS_AVG                  151450 non-null  float64
 45   BASEMENTAREA_AVG                127568 non-null  float64
 46   YEARS_BEGINEXPLUATATION_AVG     157504 non-null  float64
 47   YEARS_BUILD_AVG                 103023 non-null  float64
 48   COMMONAREA_AVG                  92646 non-null   float64
 49   ELEVATORS_AVG                   143620 non-null  float64
 50   ENTRANCES_AVG                   152683 non-null  float64
 51   FLOORSMAX_AVG                   154491 non-null  float64
 52   FLOORSMIN_AVG                   98869 non-null   float64
 53   LANDAREA_AVG                    124921 non-null  float64
 54   LIVINGAPARTMENTS_AVG            97312 non-null   float64
```

```
55   LIVINGAREA_AVG                153161 non-null  float64
56   NONLIVINGAPARTMENTS_AVG       93997 non-null   float64
57   NONLIVINGAREA_AVG             137829 non-null  float64
58   APARTMENTS_MODE               151450 non-null  float64
59   BASEMENTAREA_MODE             127568 non-null  float64
60   YEARS_BEGINEXPLUATATION_MODE  157504 non-null  float64
61   YEARS_BUILD_MODE              103023 non-null  float64
62   COMMONAREA_MODE               92646 non-null   float64
63   ELEVATORS_MODE                143620 non-null  float64
64   ENTRANCES_MODE                152683 non-null  float64
65   FLOORSMAX_MODE                154491 non-null  float64
66   FLOORSMIN_MODE                98869 non-null   float64
67   LANDAREA_MODE                 124921 non-null  float64
68   LIVINGAPARTMENTS_MODE         97312 non-null   float64
69   LIVINGAREA_MODE               153161 non-null  float64
70   NONLIVINGAPARTMENTS_MODE      93997 non-null   float64
71   NONLIVINGAREA_MODE            137829 non-null  float64
72   APARTMENTS_MEDI               151450 non-null  float64
73   BASEMENTAREA_MEDI             127568 non-null  float64
74   YEARS_BEGINEXPLUATATION_MEDI  157504 non-null  float64
75   YEARS_BUILD_MEDI              103023 non-null  float64
76   COMMONAREA_MEDI               92646 non-null   float64
77   ELEVATORS_MEDI                143620 non-null  float64
78   ENTRANCES_MEDI                152683 non-null  float64
79   FLOORSMAX_MEDI                154491 non-null  float64
80   FLOORSMIN_MEDI                98869 non-null   float64
81   LANDAREA_MEDI                 124921 non-null  float64
82   LIVINGAPARTMENTS_MEDI         97312 non-null   float64
83   LIVINGAREA_MEDI               153161 non-null  float64
84   NONLIVINGAPARTMENTS_MEDI      93997 non-null   float64
85   NONLIVINGAREA_MEDI            137829 non-null  float64
86   FONDKAPREMONT_MODE            97216 non-null   object
87   HOUSETYPE_MODE                153214 non-null  object
88   TOTALAREA_MODE                159080 non-null  float64
89   WALLSMATERIAL_MODE            151170 non-null  object
90   EMERGENCYSTATE_MODE           161756 non-null  object
91   OBS_30_CNT_SOCIAL_CIRCLE      306490 non-null  float64
92   DEF_30_CNT_SOCIAL_CIRCLE      306490 non-null  float64
93   OBS_60_CNT_SOCIAL_CIRCLE      306490 non-null  float64
94   DEF_60_CNT_SOCIAL_CIRCLE      306490 non-null  float64
95   DAYS_LAST_PHONE_CHANGE        307510 non-null  float64
96   FLAG_DOCUMENT_2               307511 non-null  int64
97   FLAG_DOCUMENT_3               307511 non-null  int64
98   FLAG_DOCUMENT_4               307511 non-null  int64
99   FLAG_DOCUMENT_5               307511 non-null  int64
100  FLAG_DOCUMENT_6               307511 non-null  int64
101  FLAG_DOCUMENT_7               307511 non-null  int64
102  FLAG_DOCUMENT_8               307511 non-null  int64
103  FLAG_DOCUMENT_9               307511 non-null  int64
104  FLAG_DOCUMENT_10              307511 non-null  int64
105  FLAG_DOCUMENT_11              307511 non-null  int64
106  FLAG_DOCUMENT_12              307511 non-null  int64
107  FLAG_DOCUMENT_13              307511 non-null  int64
108  FLAG_DOCUMENT_14              307511 non-null  int64
109  FLAG_DOCUMENT_15              307511 non-null  int64
110  FLAG_DOCUMENT_16              307511 non-null  int64
111  FLAG_DOCUMENT_17              307511 non-null  int64
112  FLAG_DOCUMENT_18              307511 non-null  int64
113  FLAG_DOCUMENT_19              307511 non-null  int64
114  FLAG_DOCUMENT_20              307511 non-null  int64
```

```
 115  FLAG_DOCUMENT_21              307511 non-null  int64
 116  AMT_REQ_CREDIT_BUREAU_HOUR    265992 non-null  float64
 117  AMT_REQ_CREDIT_BUREAU_DAY     265992 non-null  float64
 118  AMT_REQ_CREDIT_BUREAU_WEEK    265992 non-null  float64
 119  AMT_REQ_CREDIT_BUREAU_MON     265992 non-null  float64
 120  AMT_REQ_CREDIT_BUREAU_QRT     265992 non-null  float64
 121  AMT_REQ_CREDIT_BUREAU_YEAR    265992 non-null  float64
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
```

Observations: There are a lot of columns with null values but datatypes of these columns looks fine

In [ ]:
```
# checking the statistics summary of the dataframe
app_data.describe()
```

Out[ ]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNU
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.00(
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.50(
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.00(
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.00(
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.00(
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.50(

8 rows × 106 columns

4.Data quality check and missing values

#Inspecting Data Quality

In [ ]:
```
# checking the total number of missing values in each column
app_data.isnull().sum().sort_values(ascending=False)
```

Out[ ]:
```
COMMONAREA_MEDI             214865
COMMONAREA_AVG              214865
COMMONAREA_MODE             214865
NONLIVINGAPARTMENTS_MODE    213514
NONLIVINGAPARTMENTS_AVG     213514
                             ...
NAME_HOUSING_TYPE                0
NAME_FAMILY_STATUS               0
NAME_EDUCATION_TYPE              0
NAME_INCOME_TYPE                 0
SK_ID_CURR                       0
Length: 122, dtype: int64
```

In [ ]:
```
# checking the percentage of missing values in each column
(app_data.isnull().sum()/len(app_data)*100).sort_values(ascending=False)
```

Out[ ]:
```
COMMONAREA_MEDI             69.872297
COMMONAREA_AVG              69.872297
COMMONAREA_MODE             69.872297
NONLIVINGAPARTMENTS_MODE    69.432963
NONLIVINGAPARTMENTS_AVG     69.432963
                               ...
NAME_HOUSING_TYPE            0.000000
NAME_FAMILY_STATUS           0.000000
NAME_EDUCATION_TYPE          0.000000
NAME_INCOME_TYPE             0.000000
SK_ID_CURR                   0.000000
Length: 122, dtype: float64
```

Handling Missing values

Observations: Some columns in the dataset have missing values. Share of missing values varies a lot across these variables We are going to remove columns with high missing values percentage We are considering the threashold value as 50%.

In [ ]:
```
# removing columns with high missing values percentage
app_data= app_data.loc[:, (app_data.isnull().sum()/len(app_data)*100) < 50]   # here w
```

In [ ]:
```
# checking total no of rows and columns after removing high missing value columns
app_data.shape
```

Out[ ]:
```
(307511, 81)
```

Observations: 41 Columns had missing values percentage > 50%. once they are removed we now have 81 columns to work with

In [ ]:
```
# check the updated percentage of missing values in each column
round(app_data.isnull().sum()/len(app_data)*100,2)
```

Out[ ]:
```
SK_ID_CURR                     0.0
TARGET                         0.0
NAME_CONTRACT_TYPE             0.0
CODE_GENDER                    0.0
FLAG_OWN_CAR                   0.0
                              ...
AMT_REQ_CREDIT_BUREAU_DAY     13.5
AMT_REQ_CREDIT_BUREAU_WEEK    13.5
AMT_REQ_CREDIT_BUREAU_MON     13.5
AMT_REQ_CREDIT_BUREAU_QRT     13.5
AMT_REQ_CREDIT_BUREAU_YEAR    13.5
Length: 81, dtype: float64
```

Let's go through the list and verify the columns where the null value is less than 40% in a consecutive order.

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.

In [ ]:
```
# getting columns with null values greater than 0% and less than or equal to 40%
temp=app_data.columns[((app_data.isnull().sum()/len(app_data)*100) <= 40) & ((app_data
print(temp)
print(len(temp))
```

```
Index(['AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE', 'OCCUPATION_TYPE',
       'CNT_FAM_MEMBERS', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
       'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE',
       'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE',
       'DAYS_LAST_PHONE_CHANGE', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object')
18
```

Observation:We have 18 columns with null values larger than 0% but less than or equal to 40%.

AMT_ANNUITY : imputing missing values with median as there are outliers in AMT_ANNUITY AMT_GOODS_PRICE : imputing missing values with median as there are outliers in AMT_GOODS_PRICE NAME_TYPE_SUITE : imputing missing values with mode in NAME_TYPE_SUITE as it is categorical data OCCUPATION_TYPE : imputing missing values with 'Others' in OCCUPATION_TYPE as it is categorical data EXT_SOURCE_2 : deleting the column since higher percentage of missing values EXT_SOURCE_3 : deleting the column since higher percentage of missing values CNT_FAM_MEMBERS : deleting the row sonce the count of null values is only 2, and it wouldnt have much impact on overall data OBS_30_CNT_SOCIAL_CIRCLE : imputing missing values with median as there are outliers in OBS_30_CNT_SOCIAL_CIRCLE DEF_30_CNT_SOCIAL_CIRCLE : imputing missing values with median as there are outliers in DEF_30_CNT_SOCIAL_CIRCLE OBS_60_CNT_SOCIAL_CIRCLE : imputing missing values with median as there are outliers in OBS_60_CNT_SOCIAL_CIRCLE DEF_60_CNT_SOCIAL_CIRCLE : imputing missing values with median as there are outliers in DEF_60_CNT_SOCIAL_CIRCLE DAYS_LAST_PHONE_CHANGE : imputing missing values with 0 (meaning no data) AMT_REQ_CREDIT_BUREAU_HOUR : imputing missing values with median as there are outliers in AMT_REQ_CREDIT_BUREAU_HOUR AMT_REQ_CREDIT_BUREAU_DAY : imputing missing values

with median as there are outliers in AMT_REQ_CREDIT_BUREAU_DAY AMT_REQ_CREDIT_BUREAU_WEEK : imputing missing values with median as there are outliers in AMT_REQ_CREDIT_BUREAU_WEEK AMT_REQ_CREDIT_BUREAU_MON : imputing missing values with median as there are outliers in AMT_REQ_CREDIT_BUREAU_MON AMT_REQ_CREDIT_BUREAU_QRT : imputing missing values with median as there are outliers in AMT_REQ_CREDIT_BUREAU_QRT AMT_REQ_CREDIT_BUREAU_YEAR : imputing missing values with median as there are outliers in AMT_REQ_CREDIT_BUREAU_YEAR

In [ ]:

2.Handling Outlier values and NULL values

Observations: As a result of the above observation, the number of null value columns has been significantly decreased. Next we look for values that can be imputed. But, as noted in the case study, these columns do not need to be deleted or imputed.

AMT_ANNUITY Imputation

Observations: AMT_ANNUITY - Loan annuity, is a Numeric variable Missing values in percentage is 0.003902 We can use .describe boxplots to check for outliers

In [ ]:
```
# inspecting AMT_ANNUITY column
app_data[app_data['AMT_ANNUITY'].isnull()]
```

Out[ ]:

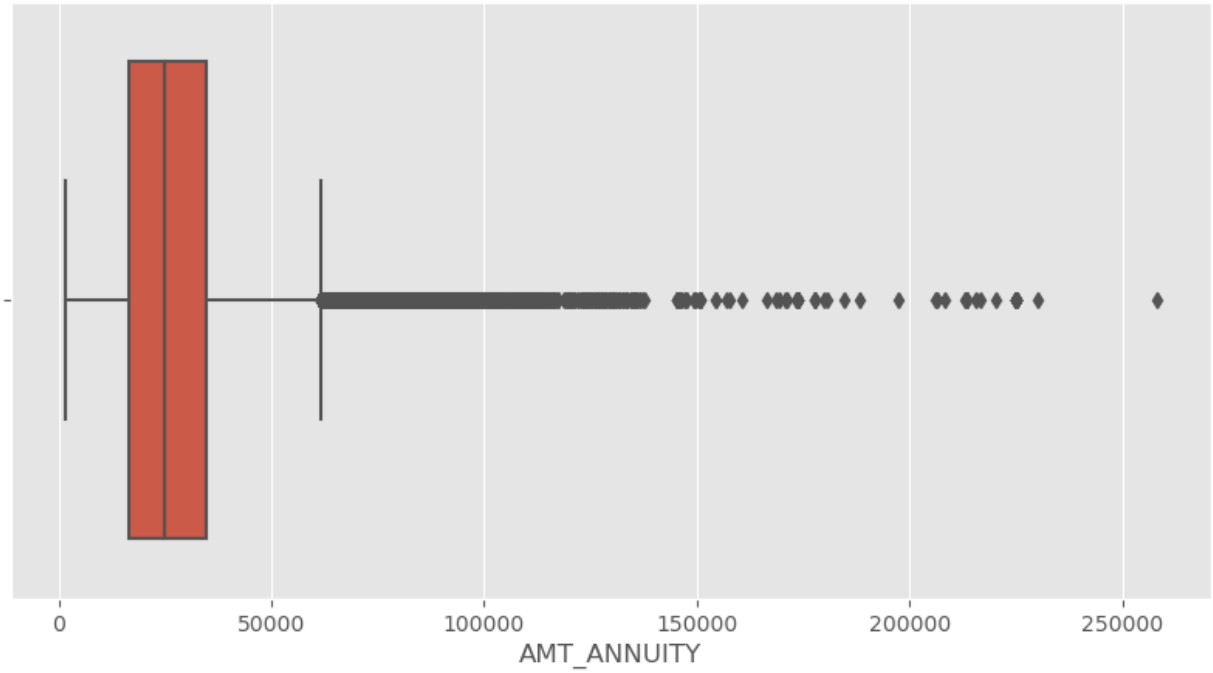
	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_
47531	155054	0	Cash loans	M	N	
50035	157917	0	Cash loans	F	N	
51594	159744	0	Cash loans	F	N	
55025	163757	0	Cash loans	F	N	
59934	169487	0	Cash loans	M	Y	
75873	187985	0	Cash loans	M	Y	
89343	203726	0	Cash loans	F	Y	
123872	243648	0	Cash loans	F	N	
207186	340147	0	Cash loans	M	N	
227939	364022	0	Cash loans	F	N	
239329	377174	0	Cash loans	F	N	
241835	379997	0	Cash loans	F	N	

12 rows × 81 columns

In [ ]:
```
# checking for outliers using statistical summary of AMT_ANNUITY column and boxplot
print(app_data['AMT_ANNUITY'].describe(percentiles=[0.1,0.25,0.5,0.75,0.99]))
print()
```

```
plt.style.use('ggplot')
plt.figure(figsize=[10,5])
sns.boxplot(app_data['AMT_ANNUITY'])
plt.show()
```

```
count    307499.000000
mean      27108.573909
std       14493.737315
min        1615.500000
10%       11074.500000
25%       16524.000000
50%       24903.000000
75%       34596.000000
99%       70006.500000
max      258025.500000
Name: AMT_ANNUITY, dtype: float64
```



Observations: We can readily observe from the statistics summary that

The 99th percentile is distant from the maximum value. The presence of outliers is clearly revealed by the boxplot.

We have to impute the median value 24903 in place of missing values

We will not do imputation in this example because it is not indicated in the case study.

AMT_GOODS_PRICE Imputation

Observations: AMT_GOODS_PRICE - For consumer loans it is the price of the goods for which the loan is given. Missing values in percentage is 0.090403 It is a Numeric variable hence we can use .describe boxplots to check for outliers

In [ ]:
```
# inspecting AMT_ANNUITY column
app_data[app_data['AMT_ANNUITY'].isnull()]
```

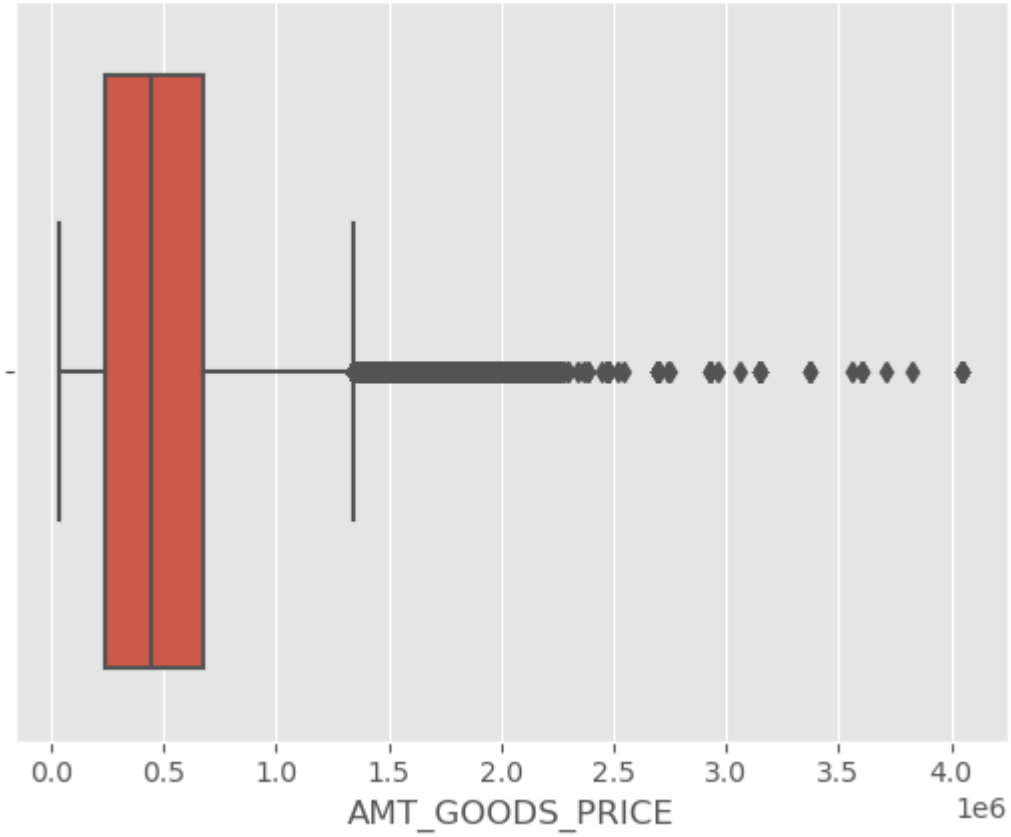
Out[ ]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_
47531	155054	0	Cash loans	M	N	
50035	157917	0	Cash loans	F	N	
51594	159744	0	Cash loans	F	N	
55025	163757	0	Cash loans	F	N	
59934	169487	0	Cash loans	M	Y	
75873	187985	0	Cash loans	M	Y	
89343	203726	0	Cash loans	F	Y	
123872	243648	0	Cash loans	F	N	
207186	340147	0	Cash loans	M	N	
227939	364022	0	Cash loans	F	N	
239329	377174	0	Cash loans	F	N	
241835	379997	0	Cash loans	F	N	

12 rows × 81 columns

In [ ]:
```
# check for outliers using statistical summary of AMT_GOODS_PRICE column and boxplot
print(round(app_data['AMT_GOODS_PRICE'].describe(percentiles=[0.1,0.25,0.5,0.75,0.99])
print()
sns.boxplot(app_data['AMT_GOODS_PRICE'])
plt.show()
```

```
count     307233.00
mean      538396.21
std       369446.46
min        40500.00
10%       180000.00
25%       238500.00
50%       450000.00
75%       679500.00
99%      1800000.00
max      4050000.00
Name: AMT_GOODS_PRICE, dtype: float64
```



Observations: We can readily observe from the statistics report that

99th percentile is far off from max value. This is a clear indicated in the boxplot well that outliers are present. Though there are values above 2000000 they cannot be treated as outliers as it could be a valid goods price We have to impute the median value 450000.00 in place of missing values

We will not do imputation in this example because it is not indicated in the case study.

NAME_TYPE_SUITE Imputation

Observations: NAME_TYPE_SUITE - Who was accompanying client when he was applying for the loan, is a categorical column

Missing values percentage is 0.420148

In [ ]:
```
# check for maximum repeated value in NAME_TYPE_SUITE
print(app_data['NAME_TYPE_SUITE'].value_counts())
print(app_data['NAME_TYPE_SUITE'].mode()[0])
```

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

Observations: Value counts, clearly states that the majority of NAME_TYPE_SUITE is under 'Unaccompanied' value. Its safe to assume that if NAME_TYPE_SUITE is null, either its not a necessary column to fill or client might not have anyone accompaning him while applying for loan. We can impute the mode value 'Unaccompanied' in place of missing values

We shall not do imputatin here because it is mentioned in the case study.

CNT_FAM_MEMBERS Imputation

Observations:

CNT_FAM_MEMBERS -The number of family members of the customer is a numerical variable.

Missing values percentage is 0.000650 Now We can use .describe boxplots to check for outliers

In [ ]:
```
# inspecting AMT_GOODS_PRICE column
app_data[app_data['CNT_FAM_MEMBERS'].isnull()]
```

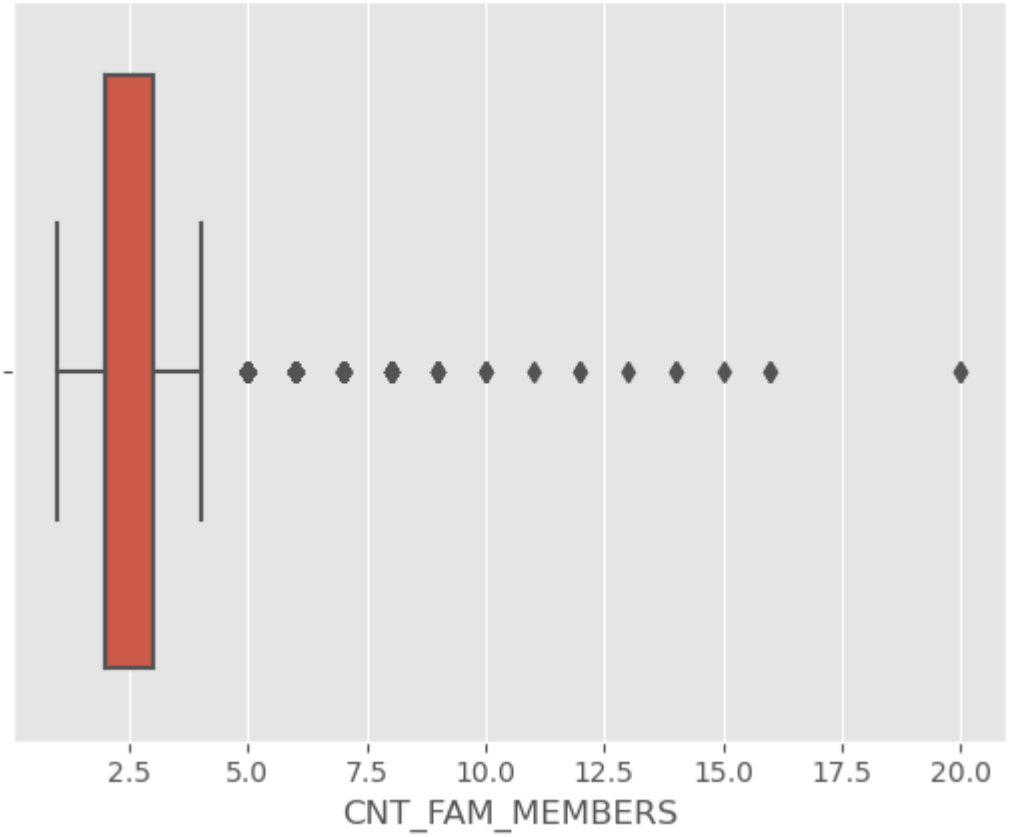
Out[ ]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_
41982	148605	0	Revolving loans	M	N	
187348	317181	0	Revolving loans	F	N	

2 rows × 81 columns

In [ ]:
```
# checking for outliers using statistical summary of CNT_FAM_MEMBERS column and boxplo
print(round(app_data['CNT_FAM_MEMBERS'].describe(percentiles=[0.1,0.25,0.5,0.75,0.99])
print()
sns.boxplot(app_data['CNT_FAM_MEMBERS'])
plt.show()
```

```
count    307509.00
mean          2.15
std           0.91
min           1.00
10%           1.00
25%           2.00
50%           2.00
75%           3.00
99%           5.00
max          20.00
Name: CNT_FAM_MEMBERS, dtype: float64
```



Observations:Looking at the statistical analysis, it is evident that 99th percentile is far off from max value. There are outlier but values aren't wrong. People can have any number of family members living with them. We only have two rows in which client family members are counted as null, thus deleting these columns wouldn't significantly alter the outcome.

While imputation is mentioned in the case study, we won't use it in this instance.

OCCUPATION_TYPE Imputation

Observations: OCCUPATION_TYPE is a categorical variable

In [ ]:
```
# checking for maximum repeated value in OCCUPATION_TYPE
print(app_data['OCCUPATION_TYPE'].value_counts())
print(app_data['OCCUPATION_TYPE'].mode()[0])
```

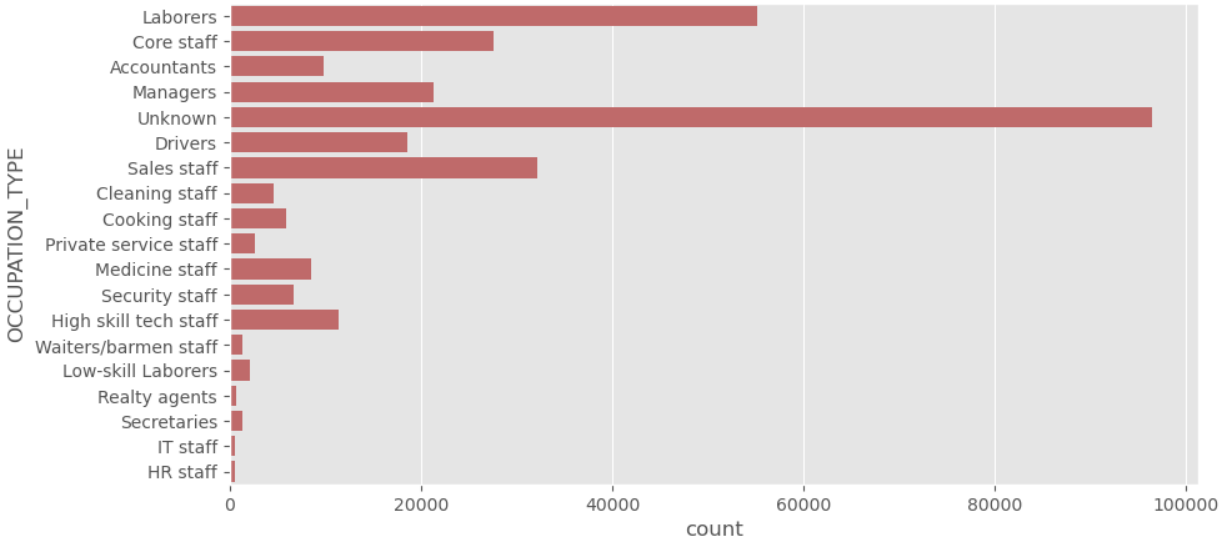
```
Laborers                 55186
Sales staff              32102
Core staff               27570
Managers                 21371
Drivers                  18603
High skill tech staff    11380
Accountants               9813
Medicine staff            8537
Security staff            6721
Cooking staff             5946
Cleaning staff            4653
Private service staff     2652
Low-skill Laborers        2093
Waiters/barmen staff      1348
Secretaries               1305
Realty agents              751
HR staff                   563
IT staff                   526
Name: OCCUPATION_TYPE, dtype: int64
Laborers
```

Observations:

The majority of those who asked for loans are labourers. In this case, it is recommended to establish a distinct column in the "OCCUPATION TYPE" column for null values. We wouldn't be imputing null values by columns' mode i.e. 'Labourers' since it will tip the scale to unbalance hence we categorise its value as 'Others'.

In [ ]:
```
# filling NaN values with others
app_data['OCCUPATION_TYPE'].fillna(value = 'Unknown', inplace = True)
```

In [ ]:
```
# plotting the count plot to see distribution of the column
plt.figure(figsize=[10,5])
sns.countplot(data = app_data, y = "OCCUPATION_TYPE", color = "indianred")
plt.show()
```

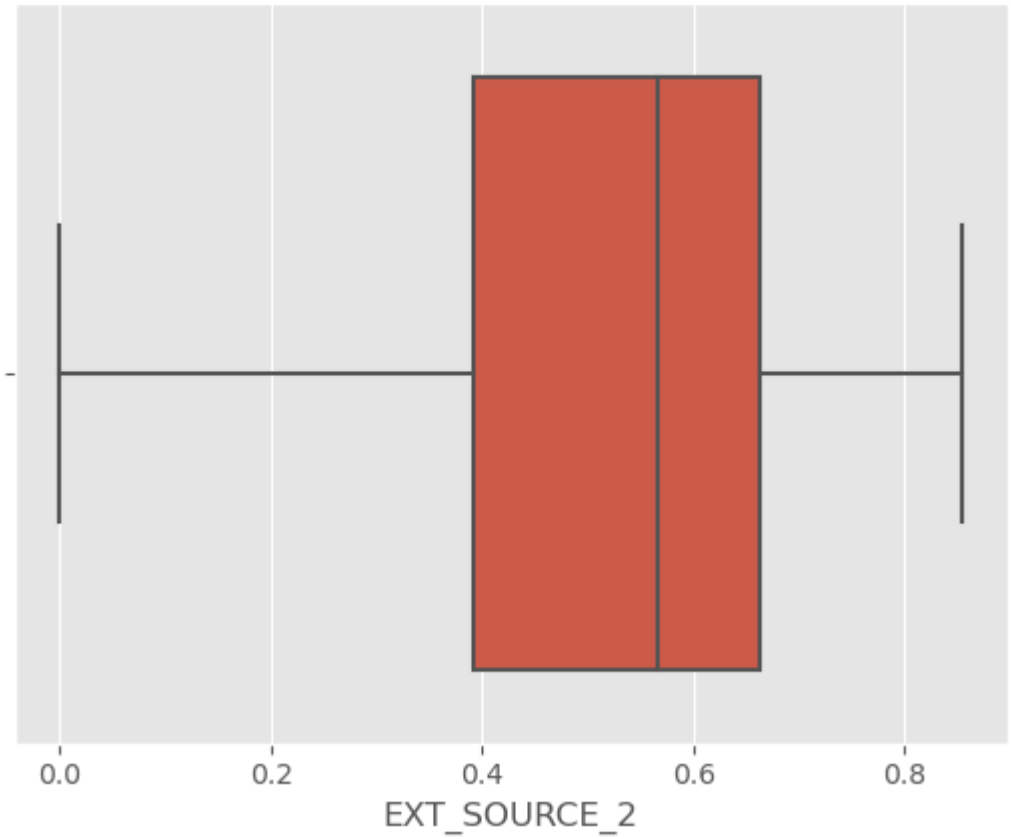


EXT_SOURCE_2 Imputation

Observations: EXT_SOURCE_2 - A numerical variable is the normalised score from an external data source. Missing percentage value is 0.214626 We can use .describe boxplots to check for outliers

In [ ]:
```
print(round(app_data['EXT_SOURCE_2'].describe(percentiles=[0.1,0.25,0.5,0.75,0.99]),4)
print()
sns.boxplot(app_data['EXT_SOURCE_2'])
plt.show()
```

```
count    306851.0000
mean          0.5144
std           0.1911
min           0.0000
10%           0.2157
25%           0.3925
50%           0.5660
75%           0.6636
99%           0.7828
max           0.8550
Name: EXT_SOURCE_2, dtype: float64
```



Observations:

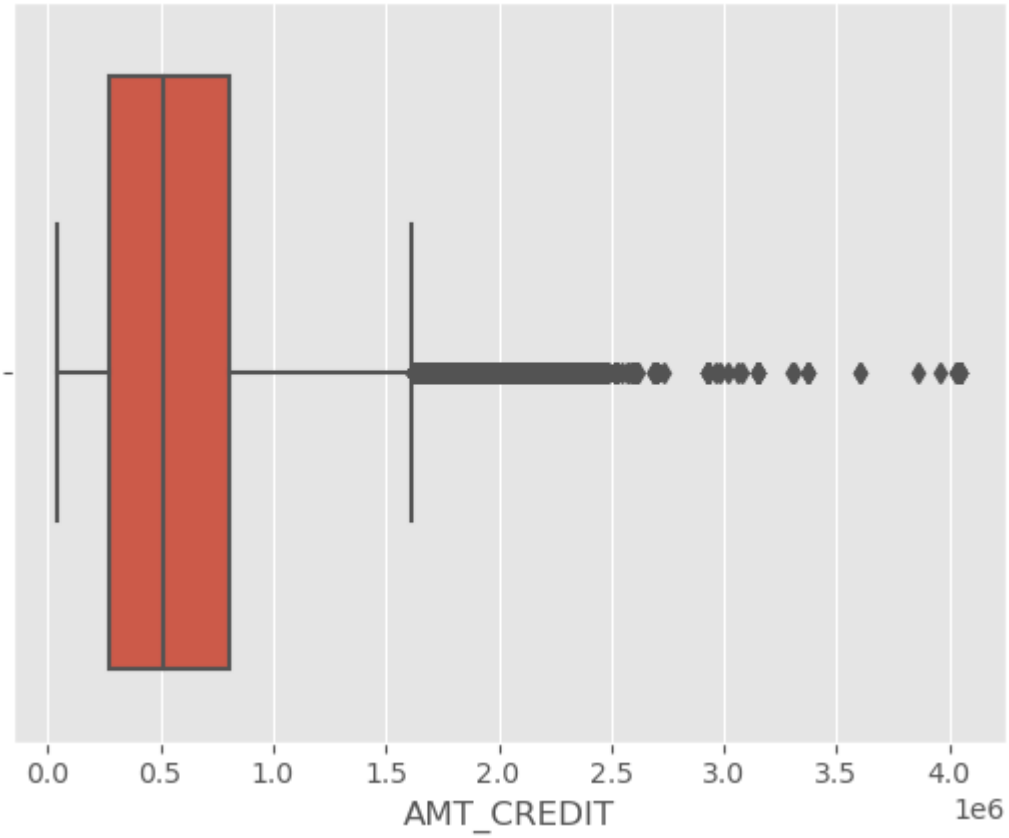
EXT_SOURCE_2 has no outliers The mean of the column can be used to impute the missing values. We won't be using imputation in this example because the case study already mentions it.

AMT_CREDIT Imputation

Observations: AMT_CREDIT - Credit amount of the loan, is a numerical variable We can use .describe boxplots to check for outliers

In [ ]:
```
print(round(app_data['AMT_CREDIT'].describe(percentiles=[0.1,0.25,0.5,0.75,0.99]),2))
print()
sns.boxplot(app_data['AMT_CREDIT'])
plt.show()
```

```
count     307511.00
mean      599026.00
std       402490.78
min        45000.00
10%       180000.00
25%       270000.00
50%       513531.00
75%       808650.00
99%      1854000.00
max      4050000.00
Name: AMT_CREDIT, dtype: float64
```



Observations: The statistical summary makes it abundantly evident that the 99th percentile is far from the maximum value. Although there are outliers, values are correct. Individuals with better credit or greater salaries may be eligible for loans with larger amounts.
We can consider amount above 1854000.00 as outlier

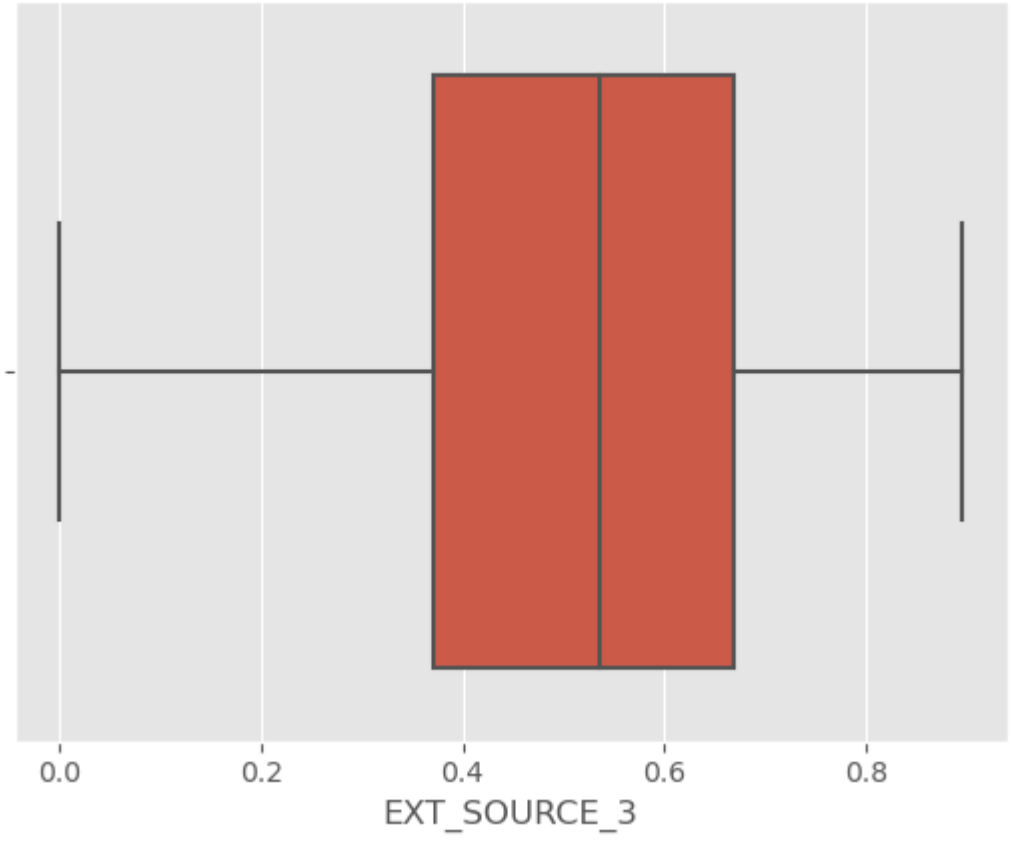
While imputation is mentioned in the case study, we won't use it in this instance.

EXT_SOURCE_3 Imputation

Observations: EXT_SOURCE_3 - Normalized score from external data source, is a numerical variable Missing values percentage is 19.825307 We can use .describe boxplots to check for outliers

In [ ]:
```
print(round(app_data['EXT_SOURCE_3'].describe(percentiles=[0.1,0.25,0.5,0.75,0.99]),4)
print()
sns.boxplot(app_data['EXT_SOURCE_3'])
plt.show()
```

```
count    246546.0000
mean          0.5109
std           0.1948
min           0.0005
10%           0.2276
25%           0.3706
50%           0.5353
75%           0.6691
99%           0.8328
max           0.8960
Name: EXT_SOURCE_3, dtype: float64
```



Observations: We see that EXT_SOURCE_3 has no outliers The number of null values is very high. We can drop EXT_SOURCE_3 since it has high percentage of null values and only use

EXT_SCORE2 for reference.

Number of enquiries to Credit Bureau Imputation

Observations: Columns with numbers of enquiries to Credit Bureau columns are: 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 These are numerical variables Missing percentage values is 13.50% This value refers to number of times client tried to access Bureau to access his credit score Null values suggests, applicant hasn't got any loan We can use .describe boxplots to check for outliers

In [ ]:
```
# checkimg the summary statistics for the columns
app_data[['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()
```

Out[ ]:

	AMT_REQ_CREDIT_BUREAU_YEAR	AMT_REQ_CREDIT_BUREAU_QRT	AMT_REQ_CREDIT_BUREAU_M
count	265992.000000	265992.000000	265992.000
mean	1.899974	0.265474	0.267
std	1.869295	0.794056	0.916
min	0.000000	0.000000	0.000
25%	0.000000	0.000000	0.000
50%	1.000000	0.000000	0.000
75%	3.000000	0.000000	0.000
max	25.000000	261.000000	27.000

In [ ]:
```
# checking the most recurring value for the columns
app_data[['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']].mode()
```

Out[ ]:

	AMT_REQ_CREDIT_BUREAU_YEAR	AMT_REQ_CREDIT_BUREAU_QRT	AMT_REQ_CREDIT_BUREAU_MON
0	0.0	0.0	0.0

Observations: AMT_REQ_CREDIT_BUREAU_YEAR Mean : 1.899974 Median : 1.000000 Mode : 0 AMT_REQ_CREDIT_BUREAU_QRT Mean : 0.265474 Median : 0 Mode : 0 AMT_REQ_CREDIT_BUREAU_MON Mean : 0.267395 Median : 0 Mode : 0 AMT_REQ_CREDIT_BUREAU_WEEK Mean : 0.034362 Median : 0 Mode : 0

AMT_REQ_CREDIT_BUREAU_DAY Mean : 0.007000 Median : 0 Mode : 0 AMT_REQ_CREDIT_BUREAU_HOUR Mean : 0.006402 Median : 0 Mode : 0

We don't impute any value as nullable values percentage is higher and it might introduce bias in the data

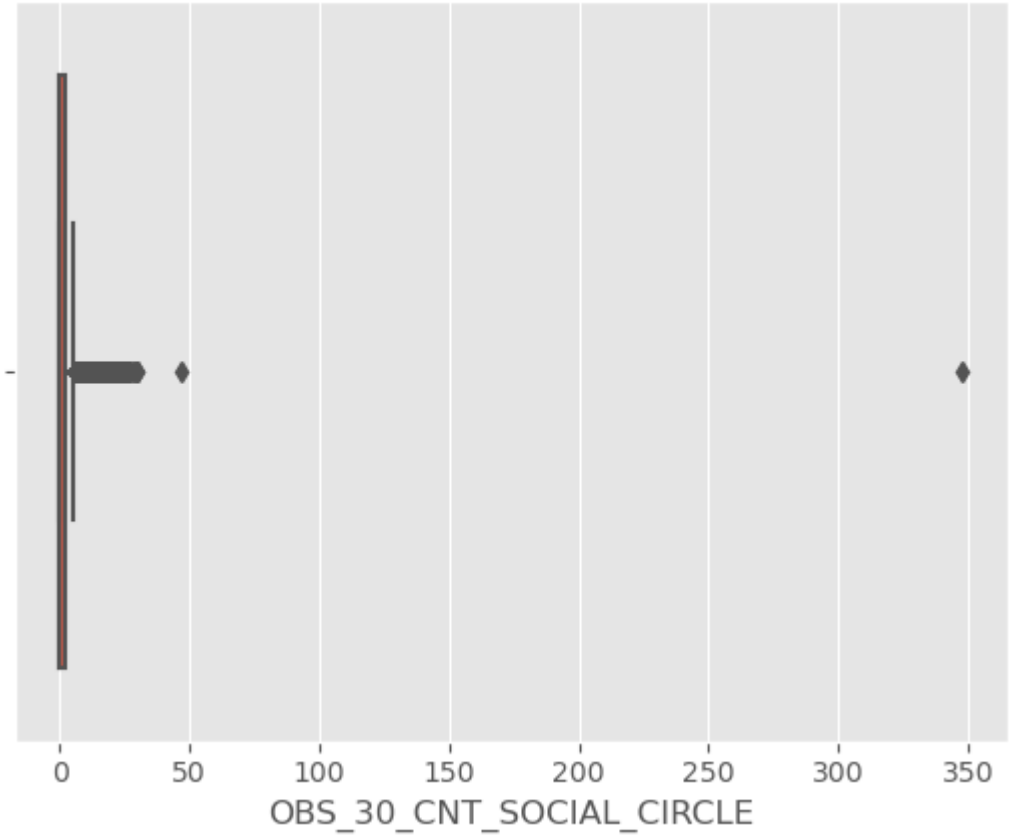
OBS_30_CNT_SOCIAL_CIRCLE Imputation

Observations:

OBS_30_CNT_SOCIAL_CIRCLE - Number of client's 30 DPD (days past due) default is a numerical column Missing values percentage is 0.33% We can use .describe boxplots to check for outliers

In [ ]:
```
print(round(app_data['OBS_30_CNT_SOCIAL_CIRCLE'].describe(percentiles=[0.1,0.25,0.5,0.
print()
sns.boxplot(app_data['OBS_30_CNT_SOCIAL_CIRCLE'])
plt.show()
```

```
count    306490.00
mean          1.42
std           2.40
min           0.00
10%           0.00
25%           0.00
50%           0.00
75%           2.00
99%          10.00
max         348.00
Name: OBS_30_CNT_SOCIAL_CIRCLE, dtype: float64
```



Observations:Looking at the statistical summary, we can clearly see that the 99th percentile is far off from max value. There are two outlier values near 50 and 350. Mode: 0.0 Mean: 1.42 Median: 0.0 We can impute values with median value since mean and median are close to each other. Missing value percentage is low hence there wouldn't be any unbalanced biases.

Inspecting incorrect/unknown data values

Observations: Certain columns, such as gender, marital status, and year of birth, allow us to utilise common sense to predict what the underlying values will be.

inspecting CODE_GENDER column

Observations:

Gender can be Male or Female.

In [ ]:
```
# checking the distribution of genders
app_data['CODE_GENDER'].value_counts()
```

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

Observations: There are roughly twice as many female applications as male applicants. XNA may imply that the client may not wish to specify the gender or it might have got missed while

entering application

In [ ]:
```
# checking the rows where gender is XNA
app_data[app_data['CODE_GENDER'] == 'XNA']
```

Out[ ]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_
35657	141289	0	Revolving loans	XNA	Y	
38566	144669	0	Revolving loans	XNA	N	
83382	196708	0	Revolving loans	XNA	N	
189640	319880	0	Revolving loans	XNA	Y	

4 rows × 81 columns

Observations: We can impute CODE_GENDER with 'F' Count of XNA is very low it wouldn't cause any unbalanced biases.

In [ ]:
```
# replacing XNA with F and then checking the count of males and females
app_data['CODE_GENDER'] = app_data['CODE_GENDER'].apply(lambda x: 'F' if x == 'XNA' el
app_data['CODE_GENDER'].value_counts()
```

Out[ ]:
```
F    202452
M    105059
Name: CODE_GENDER, dtype: int64
```

inspect DAYS_BIRTH column

Observations:

DAYS_BIRTH - Client's age in days at the time of application

In [ ]:
```
# checking statistical summary of the column
app_data['DAYS_BIRTH'].describe()
```

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

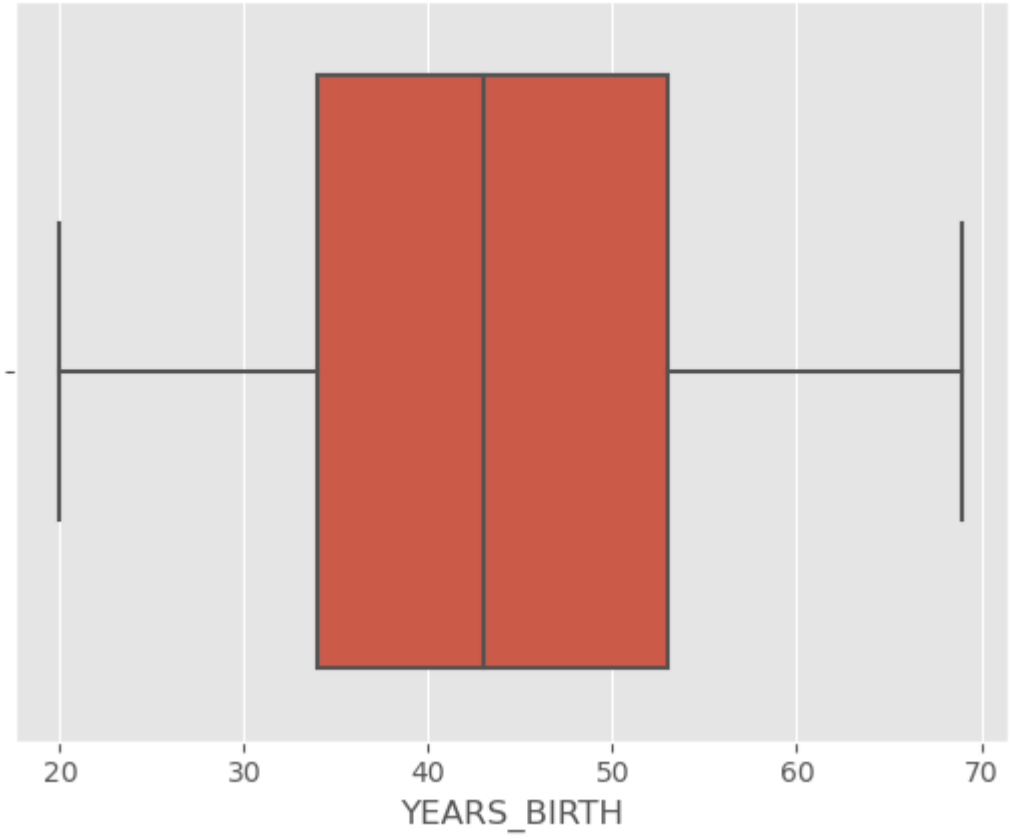
Observations: Date of birth cann't be negative Also have to convert days to years for data to make sense

In [ ]:
```
# Converting DAYS_BIRTH to positive days
app_data['DAYS_BIRTH'] = app_data['DAYS_BIRTH'].apply(lambda x: abs(x) if x < 0 else x
app_data['DAYS_BIRTH'].describe()
```

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

In [ ]:
```
# creating a new column YEARS_BIRTH for ease of analysis
app_data['YEARS_BIRTH'] = app_data['DAYS_BIRTH'].apply(lambda x: (x//365))
print(app_data['YEARS_BIRTH'].describe())
sns.boxplot(data=app_data, x='YEARS_BIRTH')
plt.show()
```

```
count    307511.000000
mean         43.435968
std          11.954593
min          20.000000
25%          34.000000
50%          43.000000
75%          53.000000
max          69.000000
Name: YEARS_BIRTH, dtype: float64
```



Observations: Min age of applicant is 21 Max age of applicant is 69 Mean and median is very close to each other i.e 43 There are no outliers Most of the applicants are in the age group between 34 to 54

inspecting NAME_FAMILY_STATUS column

Observations:

NAME_FAMILY_STATUS - Family status of the client It is a categorical column

In [ ]:
```
# checking the NAME_FAMILY_STATUS of applicants
app_data['NAME_FAMILY_STATUS'].value_counts()
```

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

Observations: Unknown value indicates that the value wasn't specified by applicant or data was missed while processing the application

In [ ]:
```
# checking the rows where NAME_FAMILY_STATUS is Unknown
app_data[app_data['NAME_FAMILY_STATUS'] == 'Unknown']
```

Out[ ]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_
41982	148605	0	Revolving loans	M	N	
187348	317181	0	Revolving loans	F	N	

2 rows × 82 columns

Observations: Data looks valid We can impute missing values with mode if the column i.e. 'Married'

In [ ]:
```
# replacing 'Unknown' with 'Married'
app_data['NAME_FAMILY_STATUS'] = app_data['NAME_FAMILY_STATUS'].apply(lambda x: 'Marri
app_data['NAME_FAMILY_STATUS'].value_counts()
```

Out[ ]:
```
Married                 196434
Single / not married     45444
Civil marriage           29775
Separated                19770
Widow                    16088
Name: NAME_FAMILY_STATUS, dtype: int64
```

Observations: Married people are the ones mostly applying for loan followed by Single or Not married Widows are the ones who have less number of application for a loan

inspecting DAYS_EMPLOYED column

Observations:

DAYS_EMPLOYED - How many days before the application the person started current employment

In [ ]:
```
# checking statistical summary of the column
app_data['DAYS_EMPLOYED'].describe()
```

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

Observations: Days of employement cann't be negative Also have to convert days to years for data to make sense

In [ ]:
```
# Converting DAYS_EMPLOYED to positive days and checking statistical values
app_data['DAYS_EMPLOYED'] = app_data['DAYS_EMPLOYED'].apply(lambda x: abs(x) if x < 0
print(app_data['DAYS_EMPLOYED'].describe())
print()
print(app_data['DAYS_EMPLOYED'].value_counts().head())
print()
print(app_data['DAYS_EMPLOYED'].value_counts(normalize=True).head())
```

```
count    307511.000000
mean      67724.742149
std      139443.751806
min           0.000000
25%         933.000000
50%        2219.000000
75%        5707.000000
max      365243.000000
Name: DAYS_EMPLOYED, dtype: float64

365243    55374
200         156
224         152
230         151
199         151
Name: DAYS_EMPLOYED, dtype: int64

365243    0.180072
200       0.000507
224       0.000494
230       0.000491
199       0.000491
Name: DAYS_EMPLOYED, dtype: float64
```

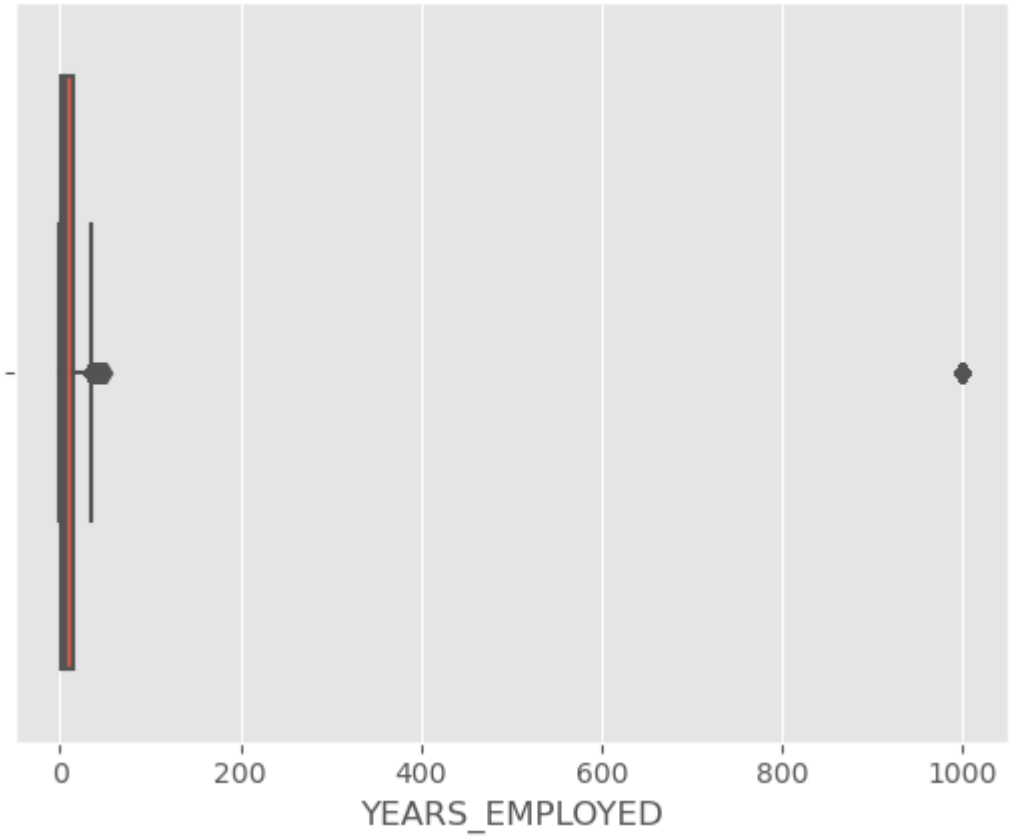
Observations: Min employement days of applicant is 0 Max employement days of applicant is 365243, which when divided by 365 comes close to 1000 years, which is physically impossible to be employed for an applicant 55374 records have days as 365243 This value is present for 18% of the data and cannot be an anamoly There are outliers

In [ ]:
```
# checking type of income for columns where employement days is 365243
app_data[app_data['DAYS_EMPLOYED'] == 365243].NAME_INCOME_TYPE.value_counts()
```

Observations: For 'Pensioners' or 'Unemployed' DAYS_EMPLOYED value is taken as 1000 years In order to avoid having our findings skewed, we must take this circumstance into account while performing computations with this column.

In [ ]:
```
# creating a new column YEARS_EMPLOYED for ease of analysis
app_data['YEARS_EMPLOYED'] = app_data['DAYS_EMPLOYED'].apply(lambda x: (x//365))
print(app_data['YEARS_EMPLOYED'].describe())
sns.boxplot(data=app_data, x='YEARS_EMPLOYED')
plt.show()
```

```
count    307511.000000
mean        185.021521
std         381.972190
min           0.000000
25%           2.000000
50%           6.000000
75%          15.000000
max        1000.000000
Name: YEARS_EMPLOYED, dtype: float64
```



Observations: Min employement years of applicant is 0 Max employement years of applicant is 1000 which is physically impossible to be employed for an applicant There are outliers but as discussed above they are 'Pentioners' or 'Unemployed'

inspecting DAYS_REGISTRATION column

Observations:

DAYS_REGISTRATION - How many days before the application did client change his registration

In [ ]:
```
# checking statistical summary of the column
print(app_data['DAYS_REGISTRATION'].describe())
print()
print(app_data['DAYS_REGISTRATION'].value_counts())
print()
print(app_data['DAYS_REGISTRATION'].value_counts(normalize=True))
```

```
count    307511.000000
mean      -4986.120328
std        3522.886321
min      -24672.000000
25%       -7479.500000
50%       -4504.000000
75%       -2010.000000
max           0.000000
Name: DAYS_REGISTRATION, dtype: float64

-1.0        113
-7.0         98
-6.0         96
-4.0         92
-2.0         92
           ...
-15581.0      1
-15031.0      1
-14804.0      1
-15008.0      1
-14798.0      1
Name: DAYS_REGISTRATION, Length: 15688, dtype: int64

-1.0        0.000367
-7.0        0.000319
-6.0        0.000312
-4.0        0.000299
-2.0        0.000299
              ...
-15581.0    0.000003
-15031.0    0.000003
-14804.0    0.000003
-15008.0    0.000003
-14798.0    0.000003
Name: DAYS_REGISTRATION, Length: 15688, dtype: float64
```

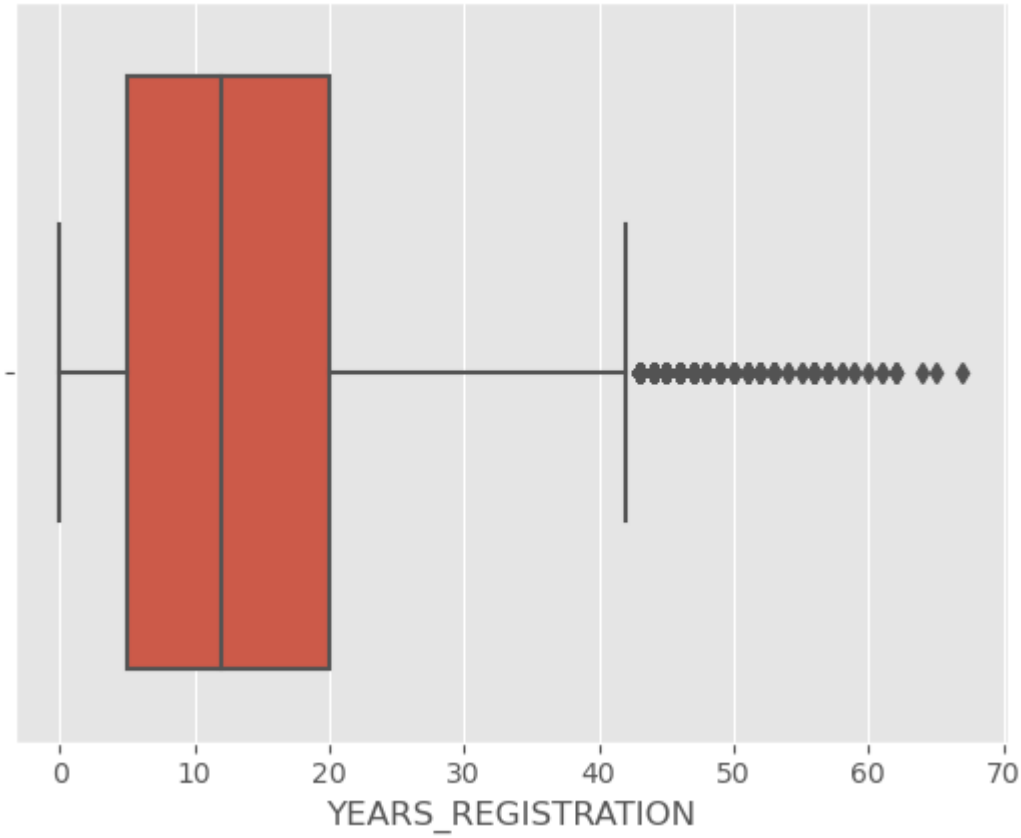
Observations: Days of registration cann't be negative Also have to convert days to years for data to make sense

In [ ]:
```
# Converting DAYS_REGISTRATION to positive days and checking statistical values
app_data['DAYS_REGISTRATION'] = app_data['DAYS_REGISTRATION'].apply(lambda x: abs(x) i
print(app_data['DAYS_REGISTRATION'].describe())
```

```
count    307511.000000
mean       4986.120328
std        3522.886321
min           0.000000
25%        2010.000000
50%        4504.000000
75%        7479.500000
max       24672.000000
Name: DAYS_REGISTRATION, dtype: float64
```

In [ ]:
```
# creating a new column YEARS_REGISTRATION for ease of analysis
app_data['YEARS_REGISTRATION'] = app_data['DAYS_REGISTRATION'].apply(lambda x: (x//365
print(app_data['YEARS_REGISTRATION'].describe())
sns.boxplot(data=app_data, x='YEARS_REGISTRATION')
plt.show()
```

```
count    307511.000000
mean         13.168683
std           9.646841
min           0.000000
25%           5.000000
50%          12.000000
75%          20.000000
max          67.000000
Name: YEARS_REGISTRATION, dtype: float64
```



Observations: Min age of applicant is 0 Max age of applicant is 67 Mean and median is very close to each other There are outliers but they are not wrong values Most of the applicants are in the resigration years range between 5 to 20

inspecting DAYS_ID_PUBLISH column

Observations:

DAYS_ID_PUBLISH - How many days before the application did client change the identity document with which he applied for the loan

In [ ]:
```
# checking statistical summary of the column
print(app_data['DAYS_ID_PUBLISH'].describe())
print()
print(app_data['DAYS_ID_PUBLISH'].value_counts().head())
print()
print(app_data['DAYS_ID_PUBLISH'].value_counts(normalize=True).head())
```

```
count    307511.000000
mean      -2994.202373
std        1509.450419
min       -7197.000000
25%       -4299.000000
50%       -3254.000000
75%       -1720.000000
max           0.000000
Name: DAYS_ID_PUBLISH, dtype: float64

-4053    169
-4095    162
-4046    161
-4417    159
-4256    158
Name: DAYS_ID_PUBLISH, dtype: int64

-4053    0.000550
-4095    0.000527
-4046    0.000524
-4417    0.000517
-4256    0.000514
Name: DAYS_ID_PUBLISH, dtype: float64
```

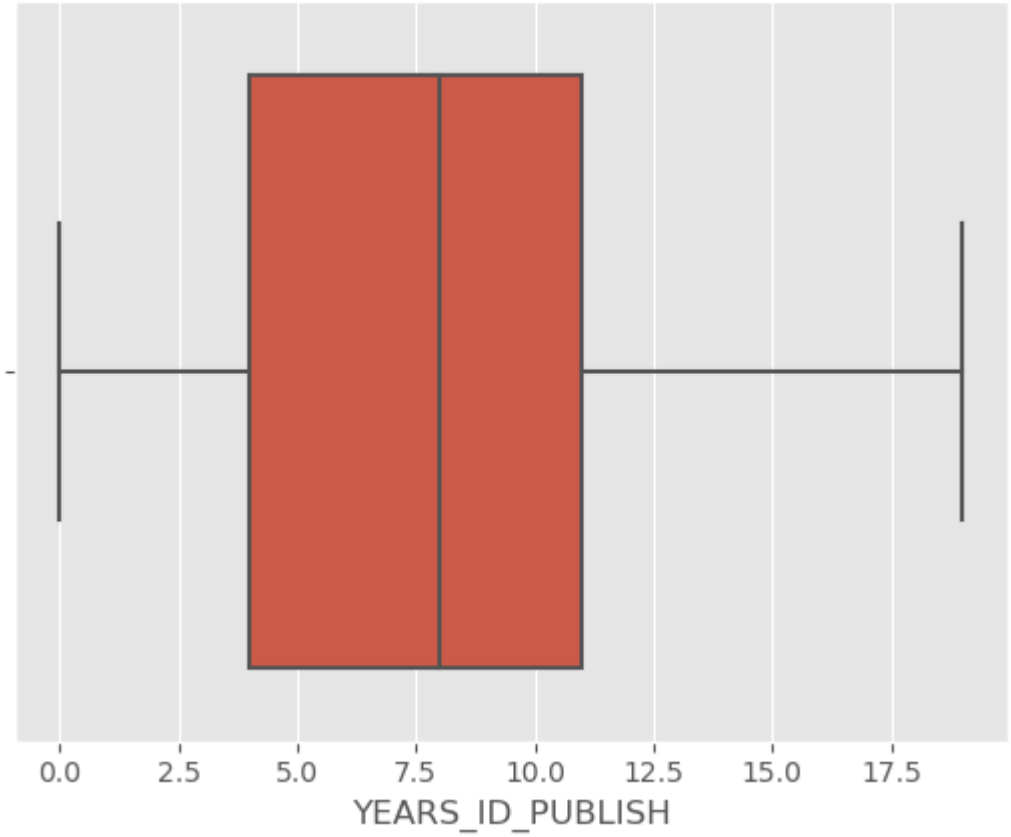
Observations: Days of ID Change cann't be negative Also have to convert days to years for data to make sense

In [ ]:
```
# Converting DAYS_ID_PUBLISH to positive days and checking statistical values
app_data['DAYS_ID_PUBLISH'] = app_data['DAYS_ID_PUBLISH'].apply(lambda x: abs(x) if x
print(app_data['DAYS_ID_PUBLISH'].describe())
```

```
count    307511.000000
mean       2994.202373
std        1509.450419
min           0.000000
25%        1720.000000
50%        3254.000000
75%        4299.000000
max        7197.000000
Name: DAYS_ID_PUBLISH, dtype: float64
```

In [ ]:
```
# creating a new column YEARS_ID_PUBLISH for ease of analysis
app_data['YEARS_ID_PUBLISH'] = app_data['DAYS_ID_PUBLISH'].apply(lambda x: (x//365))
print(app_data['YEARS_ID_PUBLISH'].describe())
sns.boxplot(data=app_data, x='YEARS_ID_PUBLISH')
plt.show()
```

```
count    307511.000000
mean          7.713474
std           4.134515
min           0.000000
25%           4.000000
50%           8.000000
75%          11.000000
max          19.000000
Name: YEARS_ID_PUBLISH, dtype: float64
```



Observations: Min age of applicant is 0 Max age of applicant is 19 Mean and median is very close to each other There are no outliers Most of the applicants who have changed ID, fall in ranges between 5 to 20 years

inspecting DAYS_LAST_PHONE_CHANGE column

Observations:

DAYS_LAST_PHONE_CHANGE - How many days before application did client change phone

In [ ]:
```
# checking statistical summary of the column
print(app_data['DAYS_LAST_PHONE_CHANGE'].describe())
print()
print(app_data['DAYS_LAST_PHONE_CHANGE'].value_counts().head())
print()
print(app_data['DAYS_LAST_PHONE_CHANGE'].value_counts(normalize=True).head())
```

```
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

 0.0    37672
-1.0     2812
-2.0     2318
-3.0     1763
-4.0     1285
Name: DAYS_LAST_PHONE_CHANGE, dtype: int64

 0.0    0.122507
-1.0    0.009144
-2.0    0.007538
-3.0    0.005733
-4.0    0.004179
Name: DAYS_LAST_PHONE_CHANGE, dtype: float64
```

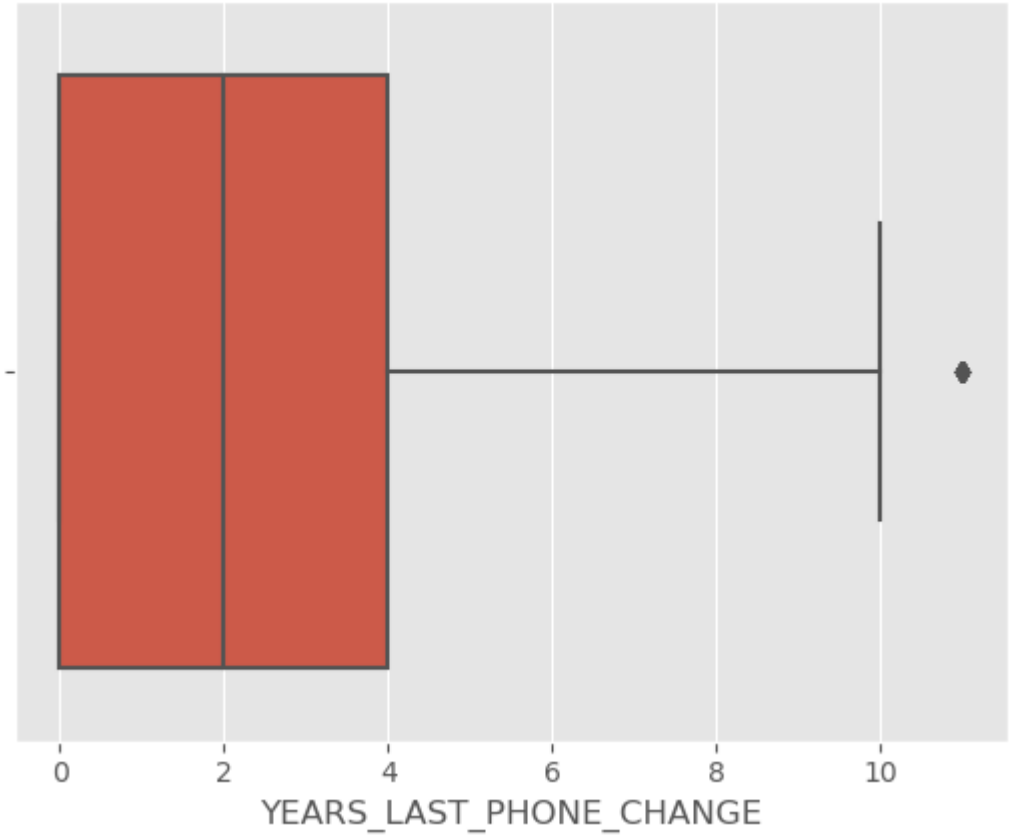
Observations: Days of ID Change cann't be negative Also have to convert days to years for data to make sense

In [ ]:
```
# Converting DAYS_LAST_PHONE_CHANGE to positive days and checking statistical values
app_data['DAYS_LAST_PHONE_CHANGE'] = app_data['DAYS_LAST_PHONE_CHANGE'].apply(lambda x
print(app_data['DAYS_LAST_PHONE_CHANGE'].describe())
```

```
count    307510.000000
mean        962.858788
std         826.808487
min           0.000000
25%         274.000000
50%         757.000000
75%        1570.000000
max        4292.000000
Name: DAYS_LAST_PHONE_CHANGE, dtype: float64
```

In [ ]:
```
# creating a new column YEARS_LAST_PHONE_CHANGE for ease of analysis
app_data['YEARS_LAST_PHONE_CHANGE'] = app_data['DAYS_LAST_PHONE_CHANGE'].apply(lambda
print(app_data['YEARS_LAST_PHONE_CHANGE'].describe())
sns.boxplot(data=app_data, x='YEARS_LAST_PHONE_CHANGE')
plt.show()
```

```
count    307510.000000
mean          2.225115
std           2.193678
min           0.000000
25%           0.000000
50%           2.000000
75%           4.000000
max          11.000000
Name: YEARS_LAST_PHONE_CHANGE, dtype: float64
```



Observations: Min age of applicant is 0 Max age of applicant is 11 Mean and median is very close to each other There is an outlier at 11 Most of the applicants who have changed ID, fall in ranges between 0 to 4 years

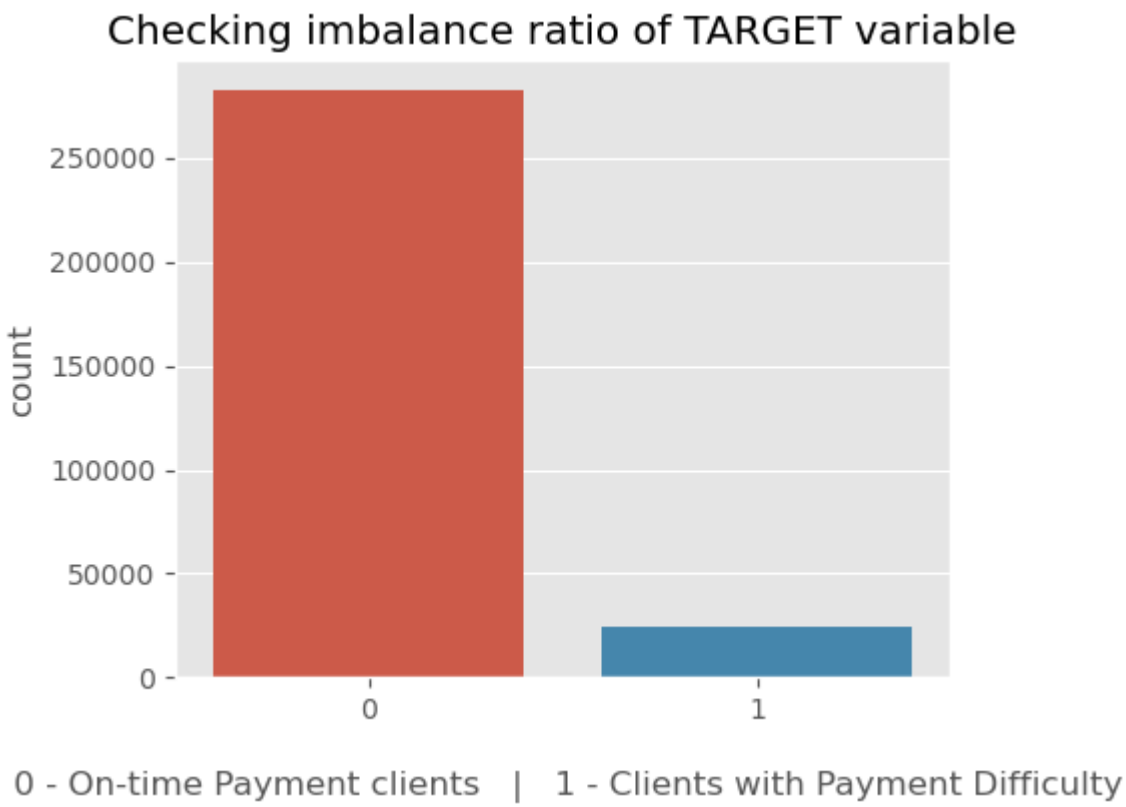
5.Data Analysis

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.

Checking Imbalance for target column 'TARGET'

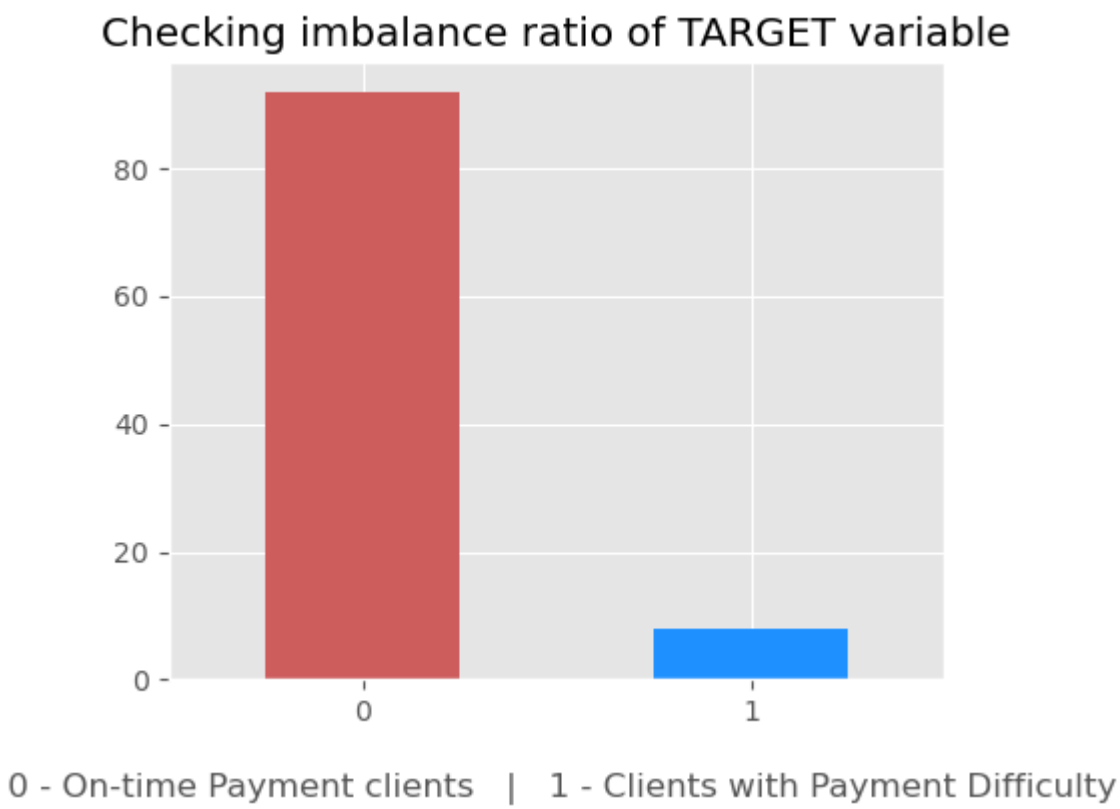
In [ ]:
```
# checking the imbalance using countplot
plt.style.use('ggplot')
plt.figure(figsize = [5,4])

sns.countplot(data=app_data, x='TARGET')
plt.title("Checking imbalance ratio of TARGET variable")
plt.xlabel("\n 0 - On-time Payment clients   |   1 - Clients with Payment Difficulty")
plt.show()
```



In [ ]:
```
# checking exact Target 0 to Target 1 ratio
app_data[app_data.TARGET==0].shape[0]/app_data[app_data.TARGET==1].shape[0]
```

Out[ ]:
```
11.387150050352467
```

In [ ]:
```
# checking the imbalance using normalization
plt.style.use('ggplot')
plt.figure(figsize = [5,4])
(app_data['TARGET'].value_counts(normalize=True)*100).plot.bar(color=['indianred', 'dc
plt.title("Checking imbalance ratio of TARGET variable")
plt.xticks(rotation = 0)
plt.xlabel("\n 0 - On-time Payment clients   |   1 - Clients with Payment Difficulty")
plt.show()
```



Observations:

1 in every 11 applicant has payment difficulty.

Creating new dataframe with TARGET value

Observations: TARGET column has 2 values 1 implies client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in our sample 0 - all other cases

In [ ]:
```
# inspecting TARGET column
print(app_data['TARGET'].value_counts())
print()
print(app_data['TARGET'].value_counts(normalize=True))
```

```
0    282686
1     24825
Name: TARGET, dtype: int64

0    0.919271
1    0.080729
Name: TARGET, dtype: float64
```

Observations: 91.92% of the clients have on-time payment 8.07% of clients have difficulty with payment We can split the data to two columns to get better insights

In [ ]:
```
# creating new dataframe with TARGET value
df0 = app_data[app_data['TARGET'] == 0]
df1 = app_data[app_data['TARGET'] == 1]
```

Univariate analysis

In [ ]:
```
# checking all columns with object type data and storing it in a list
obj_plot=list(app_data.columns[app_data.dtypes=="object"])
obj_plot
```

Out[ ]:
```
['NAME_CONTRACT_TYPE',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'NAME_TYPE_SUITE',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'OCCUPATION_TYPE',
 'WEEKDAY_APPR_PROCESS_START',
 'ORGANIZATION_TYPE',
 'EMERGENCYSTATE_MODE']
```

Analysis of object columns

In [ ]:
```
# checking each columns underlying groups
for i in obj_plot:
    # for clients with payment difficulties
    print("-------------------------------------------------")
    print(f"{i} for clients with payment difficulties")
    print(df1[i].value_counts().sort_values(ascending = False))
    # for on-time payment clients
    print(f"\n{i} for on-time payment clients")
    print(df0[i].value_counts().sort_values(ascending = False))
```

```
--------------------------------------------------
NAME_CONTRACT_TYPE for clients with payment difficulties
Cash loans         23221
Revolving loans     1604
Name: NAME_CONTRACT_TYPE, dtype: int64

NAME_CONTRACT_TYPE for on-time payment clients
Cash loans         255011
Revolving loans     27675
Name: NAME_CONTRACT_TYPE, dtype: int64
--------------------------------------------------
CODE_GENDER for clients with payment difficulties
F    14170
M    10655
Name: CODE_GENDER, dtype: int64

CODE_GENDER for on-time payment clients
F    188282
M     94404
Name: CODE_GENDER, dtype: int64
--------------------------------------------------
FLAG_OWN_CAR for clients with payment difficulties
N    17249
Y     7576
Name: FLAG_OWN_CAR, dtype: int64

FLAG_OWN_CAR for on-time payment clients
N    185675
Y     97011
Name: FLAG_OWN_CAR, dtype: int64
--------------------------------------------------
FLAG_OWN_REALTY for clients with payment difficulties
Y    16983
N     7842
Name: FLAG_OWN_REALTY, dtype: int64

FLAG_OWN_REALTY for on-time payment clients
Y    196329
N     86357
Name: FLAG_OWN_REALTY, dtype: int64
--------------------------------------------------
NAME_TYPE_SUITE for clients with payment difficulties
Unaccompanied      20337
Family              3009
Spouse, partner      895
Children             241
Other_B              174
Other_A               76
Group of people       23
Name: NAME_TYPE_SUITE, dtype: int64

NAME_TYPE_SUITE for on-time payment clients
Unaccompanied      228189
Family              37140
Spouse, partner     10475
Children             3026
Other_B              1596
Other_A               790
Group of people       248
Name: NAME_TYPE_SUITE, dtype: int64
```

```
--------------------------------------------------
NAME_INCOME_TYPE for clients with payment difficulties
Working                 15224
Commercial associate     5360
Pensioner                2982
State servant            1249
Unemployed                  8
Maternity leave             2
Name: NAME_INCOME_TYPE, dtype: int64

NAME_INCOME_TYPE for on-time payment clients
Working                 143550
Commercial associate     66257
Pensioner                52380
State servant            20454
Student                     18
Unemployed                  14
Businessman                 10
Maternity leave              3
Name: NAME_INCOME_TYPE, dtype: int64
--------------------------------------------------
NAME_EDUCATION_TYPE for clients with payment difficulties
Secondary / secondary special    19524
Higher education                  4009
Incomplete higher                  872
Lower secondary                    417
Academic degree                      3
Name: NAME_EDUCATION_TYPE, dtype: int64

NAME_EDUCATION_TYPE for on-time payment clients
Secondary / secondary special    198867
Higher education                  70854
Incomplete higher                  9405
Lower secondary                    3399
Academic degree                     161
Name: NAME_EDUCATION_TYPE, dtype: int64
--------------------------------------------------
NAME_FAMILY_STATUS for clients with payment difficulties
Married                 14850
Single / not married     4457
Civil marriage           2961
Separated                1620
Widow                     937
Name: NAME_FAMILY_STATUS, dtype: int64

NAME_FAMILY_STATUS for on-time payment clients
Married                 181584
Single / not married     40987
Civil marriage           26814
Separated                18150
Widow                    15151
Name: NAME_FAMILY_STATUS, dtype: int64
--------------------------------------------------
NAME_HOUSING_TYPE for clients with payment difficulties
House / apartment      21272
With parents            1736
Municipal apartment      955
Rented apartment         601
Office apartment         172
Co-op apartment           89
```

```
Name: NAME_HOUSING_TYPE, dtype: int64

NAME_HOUSING_TYPE for on-time payment clients
House / apartment      251596
With parents            13104
Municipal apartment     10228
Rented apartment         4280
Office apartment         2445
Co-op apartment          1033
Name: NAME_HOUSING_TYPE, dtype: int64
------------------------------------------------
OCCUPATION_TYPE for clients with payment difficulties
Unknown                  6278
Laborers                 5838
Sales staff              3092
Drivers                  2107
Core staff               1738
Managers                 1328
Security staff            722
High skill tech staff     701
Cooking staff             621
Medicine staff            572
Accountants               474
Cleaning staff            447
Low-skill Laborers        359
Private service staff     175
Waiters/barmen staff      152
Secretaries                92
Realty agents              59
HR staff                   36
IT staff                   34
Name: OCCUPATION_TYPE, dtype: int64

OCCUPATION_TYPE for on-time payment clients
Unknown                  90113
Laborers                 49348
Sales staff              29010
Core staff               25832
Managers                 20043
Drivers                  16496
High skill tech staff    10679
Accountants               9339
Medicine staff            7965
Security staff            5999
Cooking staff             5325
Cleaning staff            4206
Private service staff     2477
Low-skill Laborers        1734
Secretaries               1213
Waiters/barmen staff      1196
Realty agents              692
HR staff                   527
IT staff                   492
Name: OCCUPATION_TYPE, dtype: int64
------------------------------------------------
WEEKDAY_APPR_PROCESS_START for clients with payment difficulties
TUESDAY      4501
WEDNESDAY    4238
FRIDAY       4101
THURSDAY     4098
```

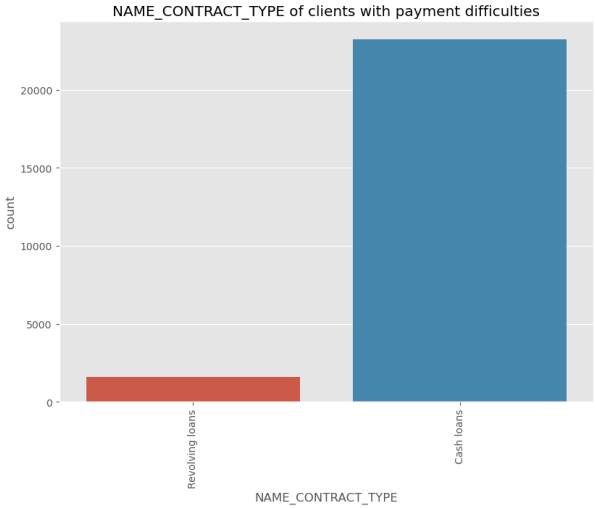
```
MONDAY       3934
SATURDAY     2670
SUNDAY       1283
Name: WEEKDAY_APPR_PROCESS_START, dtype: int64

WEEKDAY_APPR_PROCESS_START for on-time payment clients
TUESDAY      49400
WEDNESDAY    47696
MONDAY       46780
THURSDAY     46493
FRIDAY       46237
SATURDAY     31182
SUNDAY       14898
Name: WEEKDAY_APPR_PROCESS_START, dtype: int64
--------------------------------------------------
ORGANIZATION_TYPE for clients with payment difficulties
Business Entity Type 3    6323
Self-employed             3908
XNA                       2990
Other                     1275
Business Entity Type 2     900
Construction               785
Trade: type 7              740
Medicine                   737
Government                 726
School                     526
Transport: type 4          501
Business Entity Type 1     487
Kindergarten               484
Trade: type 3              361
Industry: type 3           348
Security                   324
Agriculture                257
Housing                    235
Industry: type 11          234
Industry: type 9           225
Restaurant                 212
Transport: type 3          187
Postal                     182
Transport: type 2          172
Military                   135
Trade: type 2              133
Bank                       130
Police                     117
Industry: type 1           115
Industry: type 7           105
Services                   104
Security Ministries         96
Industry: type 4            89
University                  65
Electricity                 63
Hotel                       62
Telecom                     44
Realtor                     42
Industry: type 5            41
Emergency                   40
Advertising                 35
Insurance                   34
Industry: type 2            33
Trade: type 1               31
```

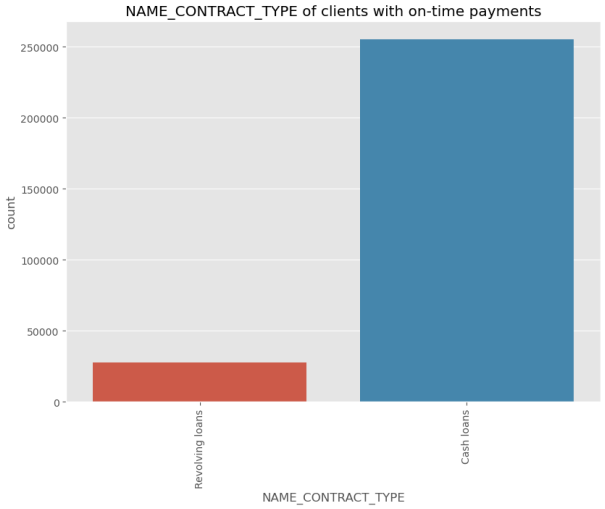
```
Cleaning                   29
Mobile                     29
Trade: type 6              29
Legal Services             24
Culture                    21
Industry: type 12          14
Industry: type 13           9
Transport: type 1           9
Industry: type 6            8
Industry: type 10           7
Religion                    5
Trade: type 5               3
Industry: type 8            3
Trade: type 4               2
Name: ORGANIZATION_TYPE, dtype: int64

ORGANIZATION_TYPE for on-time payment clients
Business Entity Type 3    61669
XNA                       52384
Self-employed             34504
Other                     15408
Medicine                  10456
Government                 9678
Business Entity Type 2     9653
School                     8367
Trade: type 7              7091
Kindergarten               6396
Construction               5936
Business Entity Type 1     5497
Transport: type 4          4897
Industry: type 9           3143
Trade: type 3              3131
Industry: type 3           2930
Security                   2923
Housing                    2723
Military                   2499
Industry: type 11          2470
Bank                       2377
Police                     2224
Agriculture                2197
Transport: type 2          2032
Postal                     1975
Security Ministries        1878
Trade: type 2              1767
Restaurant                 1599
Services                   1471
University                 1262
Industry: type 7           1202
Transport: type 3          1000
Industry: type 1            924
Hotel                       904
Electricity                 887
Industry: type 4            788
Trade: type 6               602
Insurance                   563
Industry: type 5            558
Telecom                     533
Emergency                   520
Industry: type 2            425
Advertising                 394
```

```
Culture                    358
Industry: type 12          355
Realtor                    354
Trade: type 1              317
Mobile                     288
Legal Services             281
Cleaning                   231
Transport: type 1          192
Industry: type 6           104
Industry: type 10          102
Religion                    80
Trade: type 4               62
Industry: type 13           58
Trade: type 5               46
Industry: type 8            21
Name: ORGANIZATION_TYPE, dtype: int64
--------------------------------------------------
EMERGENCYSTATE_MODE for clients with payment difficulties
No     11104
Yes      223
Name: EMERGENCYSTATE_MODE, dtype: int64

EMERGENCYSTATE_MODE for on-time payment clients
No     148324
Yes      2105
Name: EMERGENCYSTATE_MODE, dtype: int64
```

In [ ]:

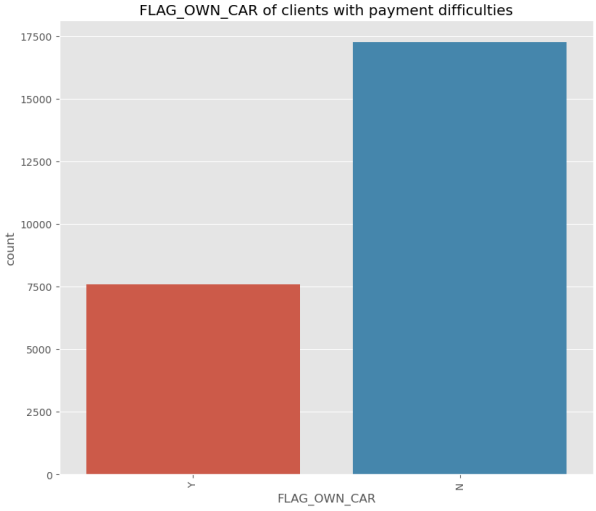
```python
# Plotting a count plot on TARGET's object columns
c_plot=['NAME_CONTRACT_TYPE','CODE_GENDER','FLAG_OWN_CAR','FLAG_OWN_REALTY','NAME_INCO
for i in c_plot:
    plt.style.use('ggplot')
    plt.figure(figsize = [18,8])
    # for clients with payment difficulties
    plt.subplot(1,2,1)
    plt.title(f'{i} of clients with payment difficulties')
    sns.countplot(data=df1, x =i, order = sorted(df1[i].unique(), reverse = True))
    plt.xticks(rotation = 90)
    # for on-time payment clients
    plt.subplot(1,2,2)
    plt.title(f'{i} of clients with on-time payments')
    sns.countplot(data=df0, x =i, order = sorted(df1[i].unique(), reverse = True))
    plt.xticks(rotation = 90)
    plt.tight_layout(pad = 4)
    plt.show()
```

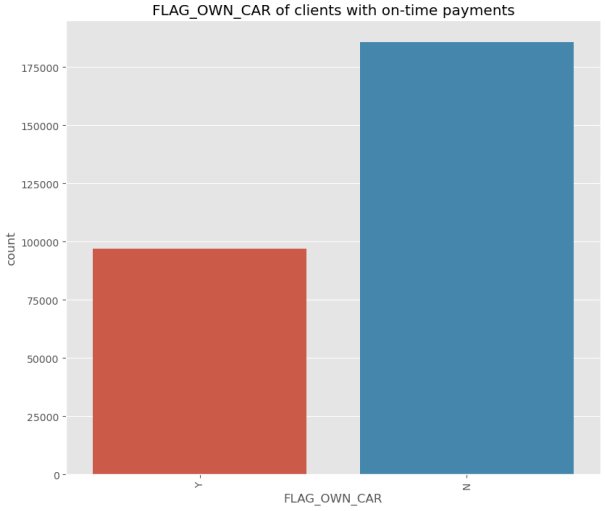

NAME_CONTRACT_TYPE of clients with payment difficulties
count
0
5000
10000
15000
20000
Revolving loans
Cash loans
NAME_CONTRACT_TYPE

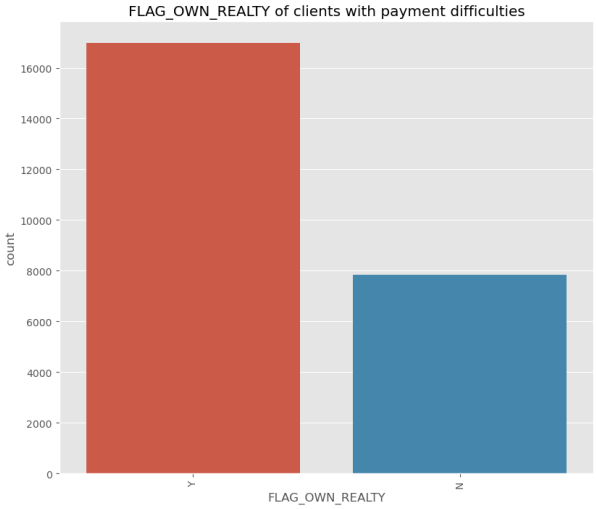

NAME_CONTRACT_TYPE of clients with on-time payments
count
0
50000
100000
150000
200000
250000
Revolving loans
Cash loans
NAME_CONTRACT_TYPE

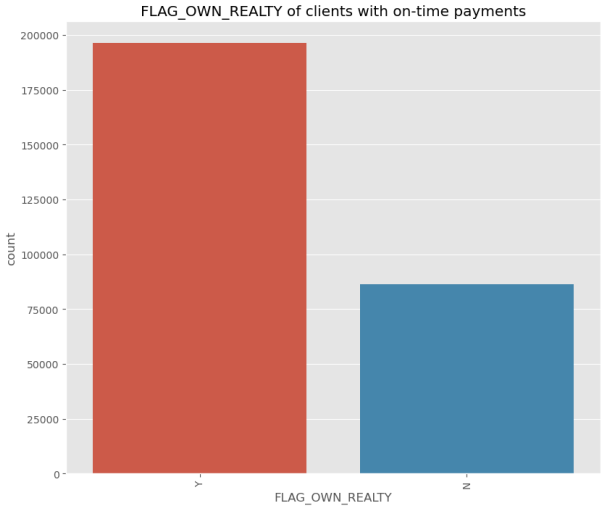

CODE_GENDER of clients with payment difficulties
count
0
2000
4000
6000
8000
10000
12000
14000
M
F
CODE_GENDER

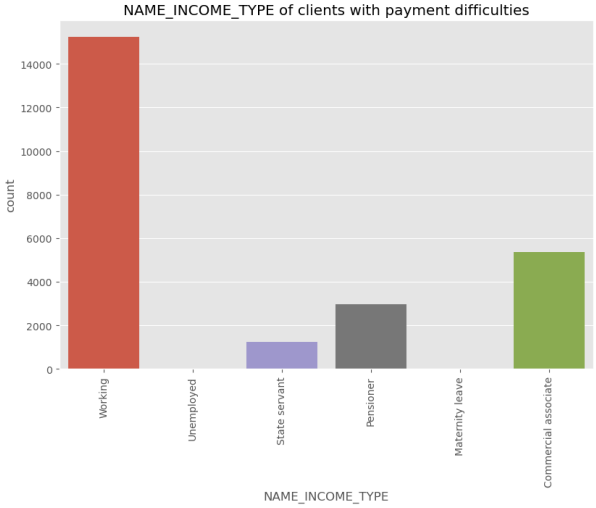

CODE_GENDER of clients with on-time payments
count
0
25000
50000
75000
100000
125000
150000
175000
M
F
CODE_GENDER

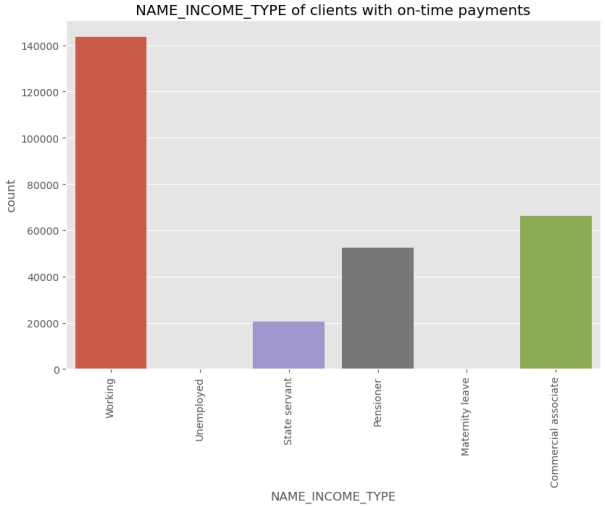

FLAG_OWN_CAR of clients with payment difficulties
count
0
2500
5000
7500
10000
12500
15000
17500
Y
N
FLAG_OWN_CAR

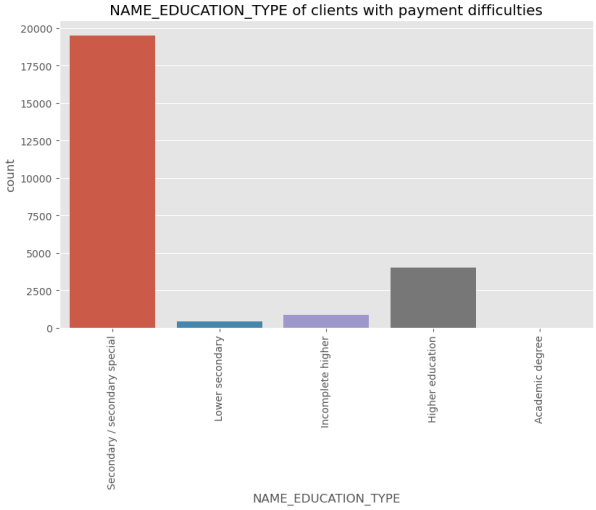

FLAG_OWN_CAR of clients with on-time payments
count
0
25000
50000
75000
100000
125000
150000
175000
Y
N
FLAG_OWN_CAR

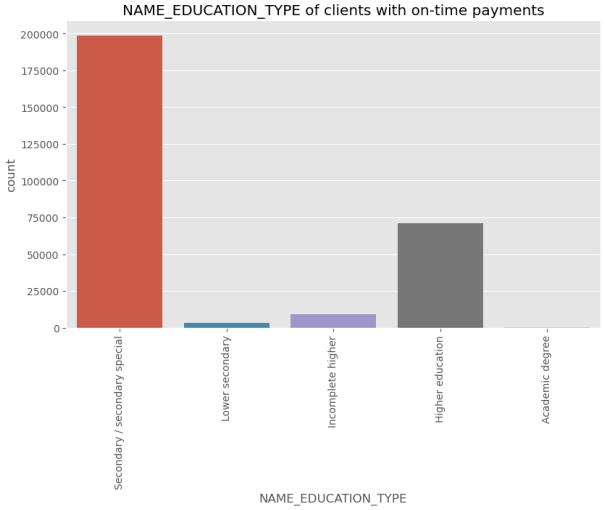

FLAG_OWN_REALTY of clients with payment difficulties
count
FLAG_OWN_REALTY
Y
N

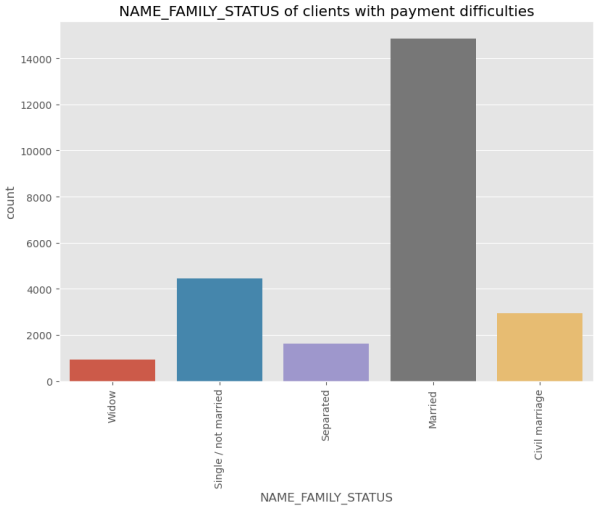

FLAG_OWN_REALTY of clients with on-time payments
count
FLAG_OWN_REALTY
Y
N

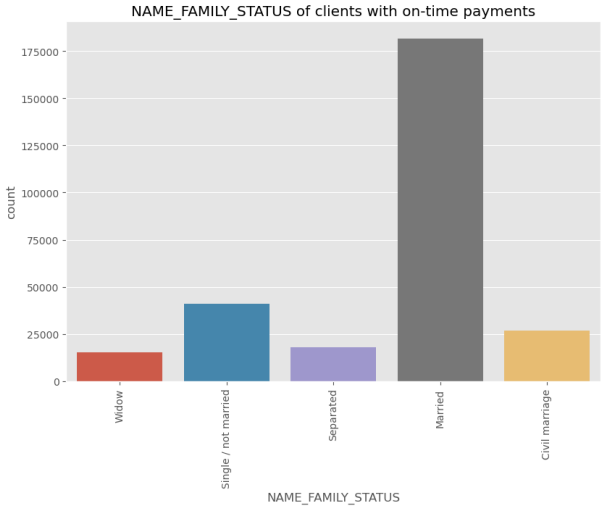

NAME_INCOME_TYPE of clients with payment difficulties
count
NAME_INCOME_TYPE
Working
Unemployed
State servant
Pensioner
Maternity leave
Commercial associate

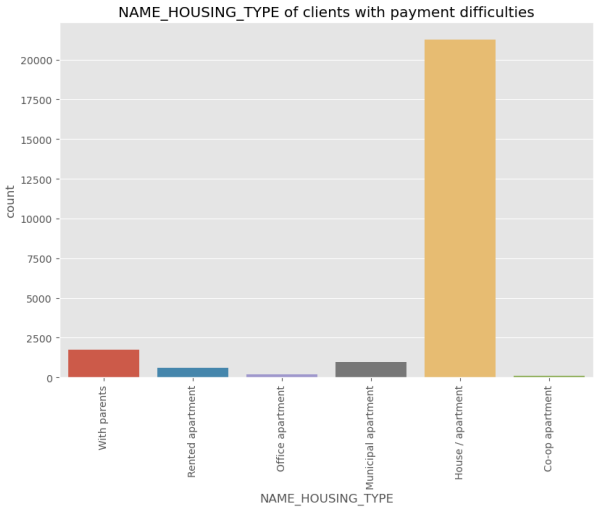

NAME_INCOME_TYPE of clients with on-time payments
count
NAME_INCOME_TYPE
Working
Unemployed
State servant
Pensioner
Maternity leave
Commercial associate

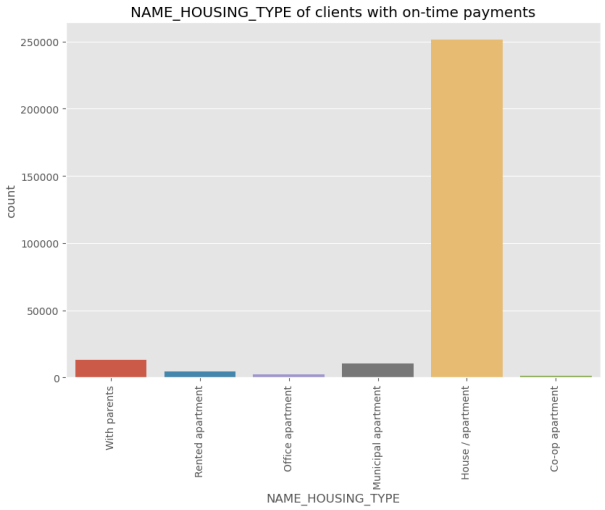

NAME_EDUCATION_TYPE of clients with payment difficulties
count
NAME_EDUCATION_TYPE
Secondary / secondary special
Lower secondary
Incomplete higher
Higher education
Academic degree

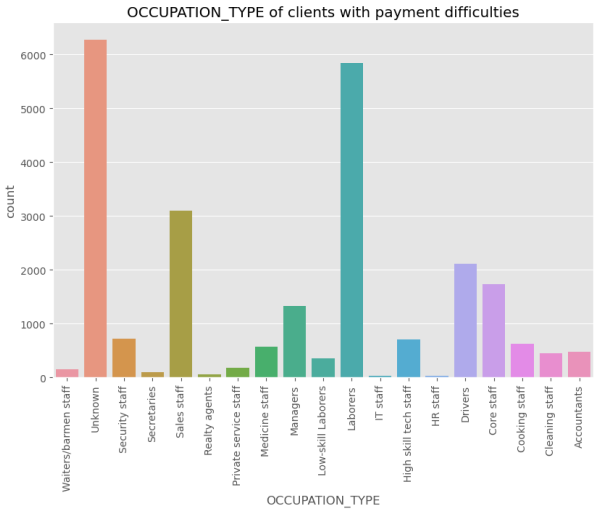

NAME_EDUCATION_TYPE of clients with on-time payments
count
NAME_EDUCATION_TYPE
Secondary / secondary special
Lower secondary
Incomplete higher
Higher education
Academic degree

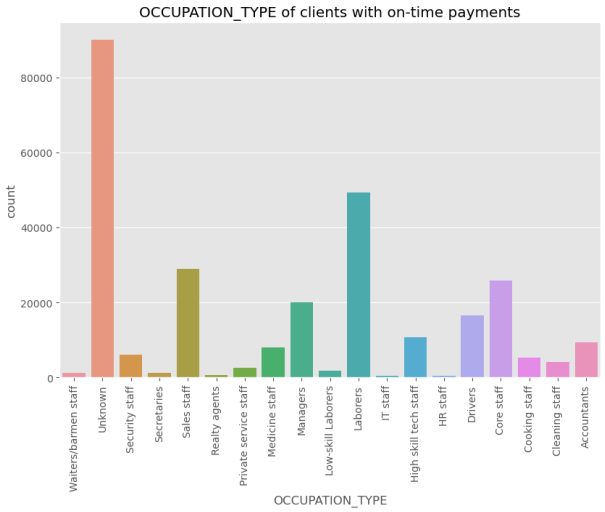

NAME_FAMILY_STATUS of clients with payment difficulties
count
0
2000
4000
6000
8000
10000
12000
14000
Widow
Single / not married
Separated
Married
Civil marriage
NAME_FAMILY_STATUS

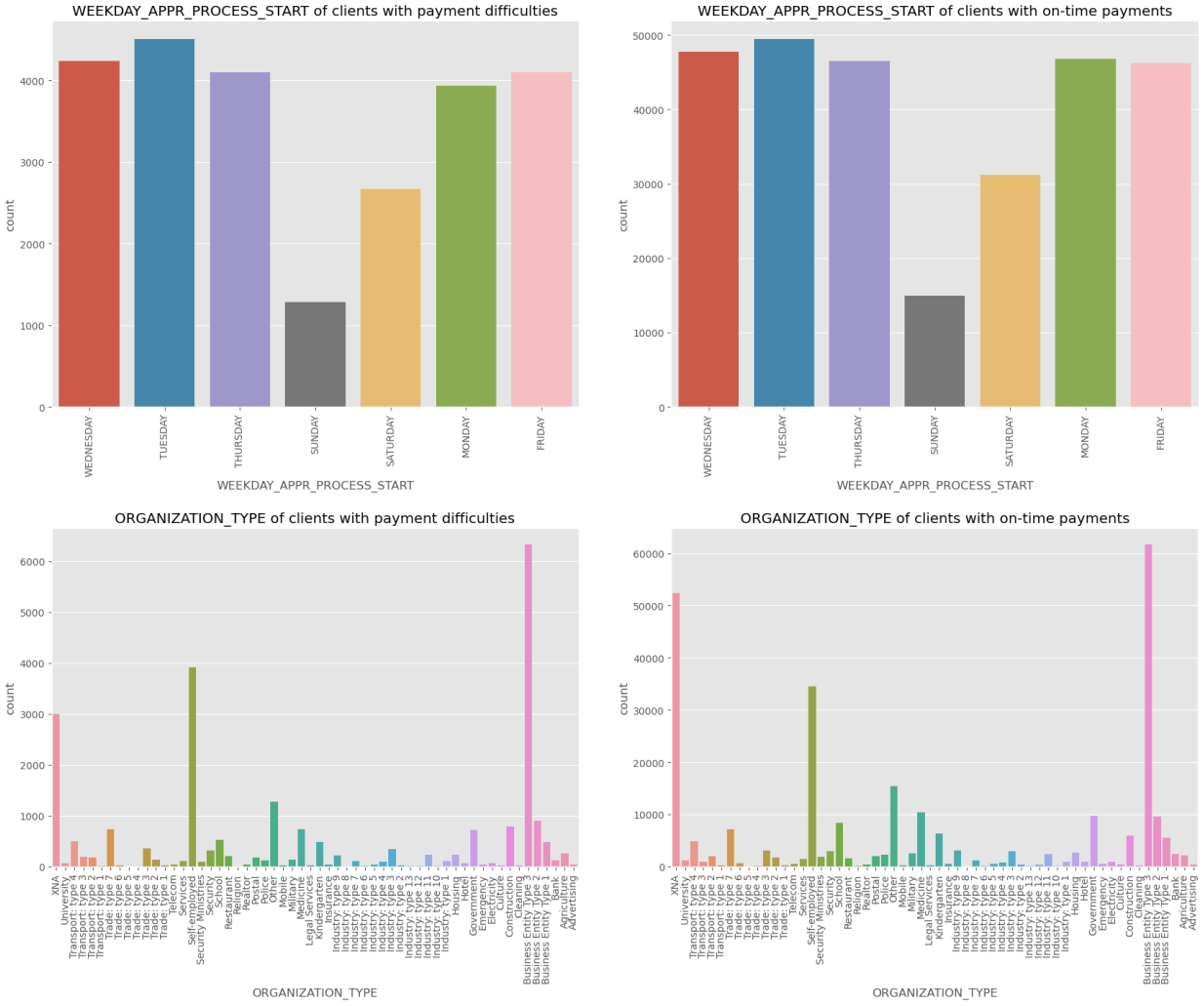

NAME_FAMILY_STATUS of clients with on-time payments
count
0
25000
50000
75000
100000
125000
150000
175000
Widow
Single / not married
Separated
Married
Civil marriage
NAME_FAMILY_STATUS


NAME_HOUSING_TYPE of clients with payment difficulties
count
0
2500
5000
7500
10000
12500
15000
17500
20000
With parents
Rented apartment
Office apartment
Municipal apartment
House / apartment
Co-op apartment
NAME_HOUSING_TYPE


NAME_HOUSING_TYPE of clients with on-time payments
count
0
50000
100000
150000
200000
250000
With parents
Rented apartment
Office apartment
Municipal apartment
House / apartment
Co-op apartment
NAME_HOUSING_TYPE

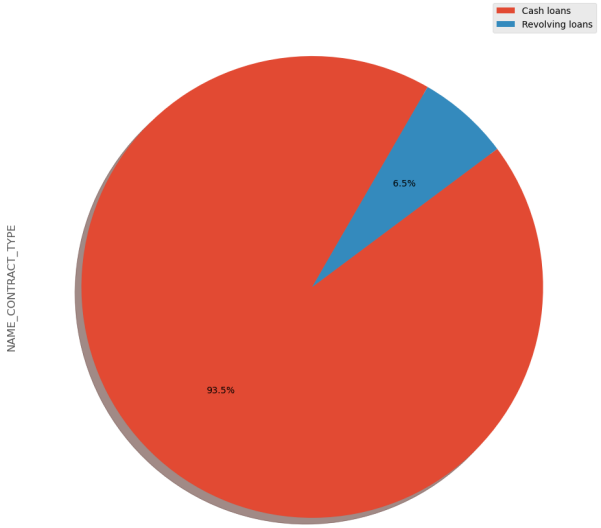

OCCUPATION_TYPE of clients with payment difficulties
count
0
1000
2000
3000
4000
5000
6000
Waiters/barmen staff
Unknown
Security staff
Secretaries
Sales staff
Realty agents
Private service staff
Medicine staff
Managers
Low-skill Laborers
Laborers
IT staff
High skill tech staff
HR staff
Drivers
Core staff
Cooking staff
Cleaning staff
Accountants
OCCUPATION_TYPE


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

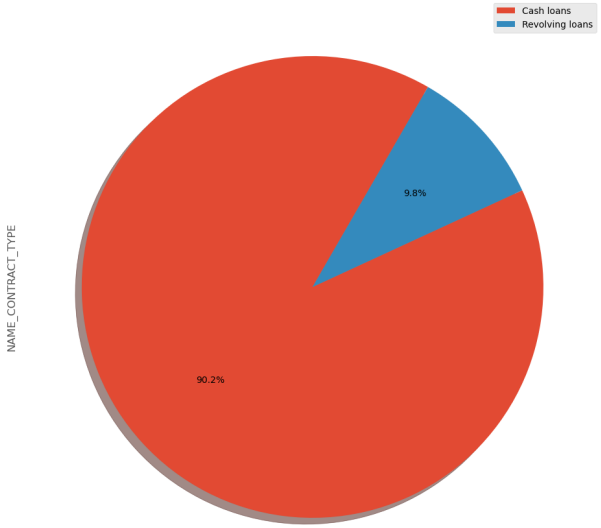


In [ ]:
```
p_plot=['NAME_CONTRACT_TYPE','CODE_GENDER','FLAG_OWN_CAR','FLAG_OWN_REALTY','NAME_FAMI
# Plotting a pie chart on TARGET's object columns
for i in p_plot:
    plt.style.use('ggplot')
    plt.figure(figsize = [20,12])
    # for clients with payment difficulties
    plt.subplot(1,2,1)
    plt.title(f'{i} of clients with payment difficulties')
    df1[i].value_counts().plot.pie(autopct='%1.1f%%',shadow=True, startangle=60, label
    plt.legend()
    # for on-time payment clients
    plt.subplot(1,2,2)
    plt.title(f'{i} of clients with on-time payments')
    df0[i].value_counts().plot.pie(autopct='%1.1f%%',shadow=True, startangle=60, label
    plt.legend()
    plt.tight_layout(pad = 4)
    plt.show()
```

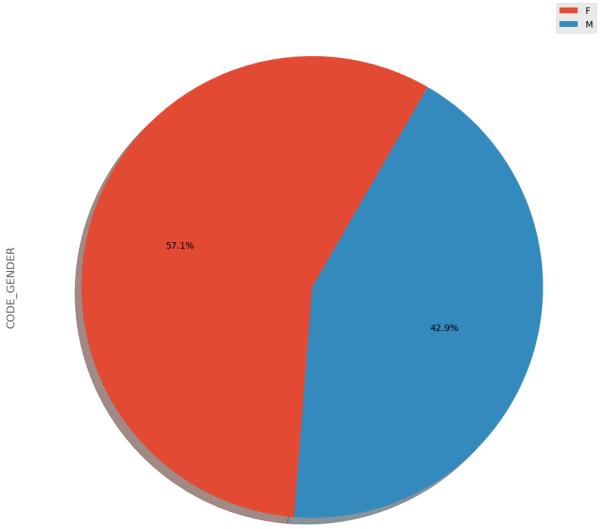
NAME_CONTRACT_TYPE of clients with payment difficulties



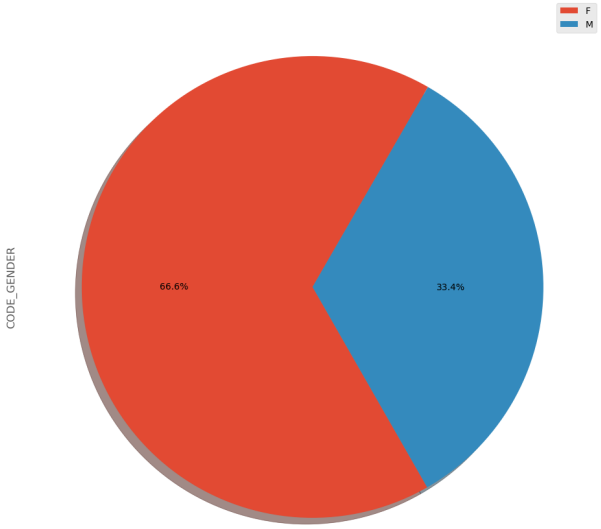
NAME_CONTRACT_TYPE of clients with on-time payments



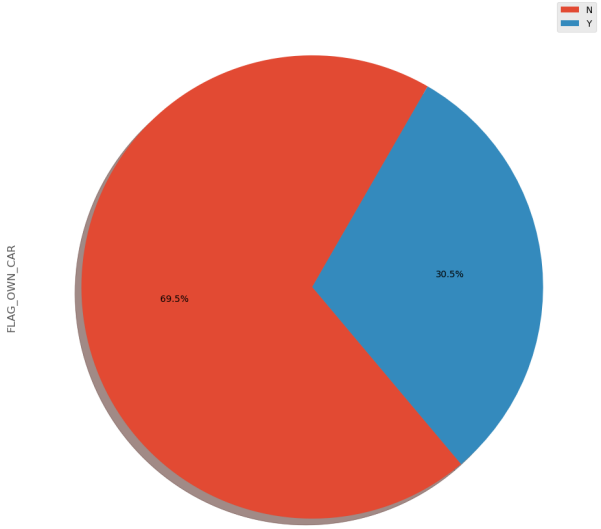
CODE_GENDER of clients with payment difficulties



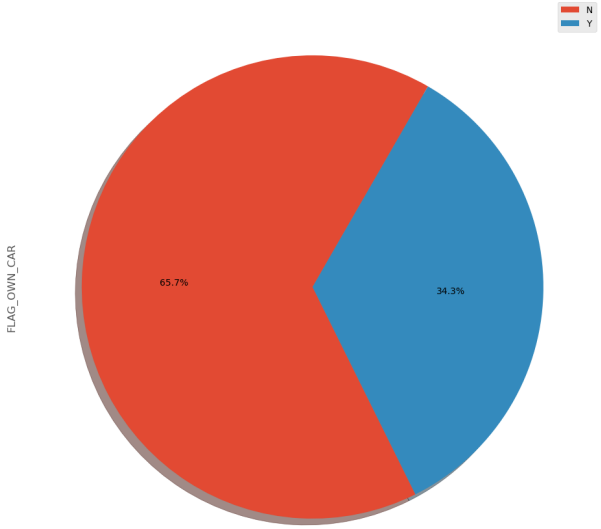
CODE_GENDER of clients with on-time payments

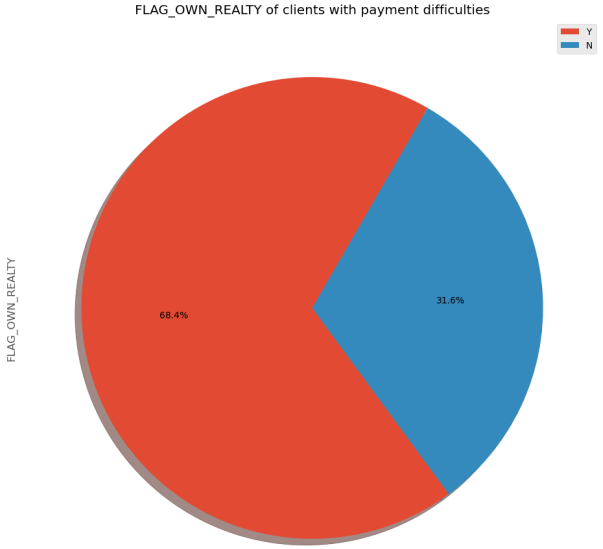


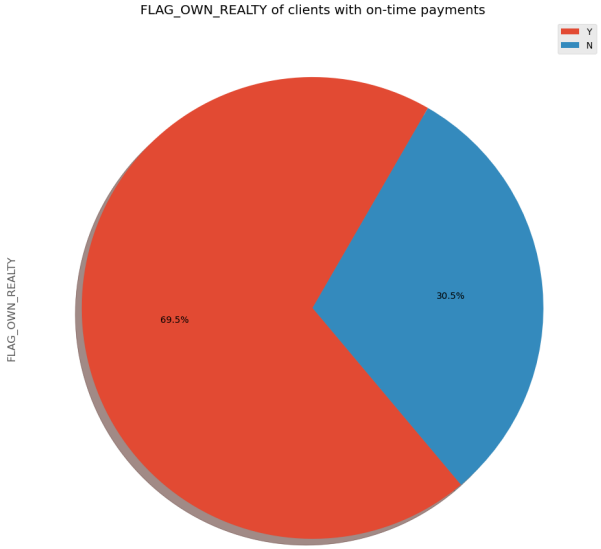
FLAG_OWN_CAR of clients with payment difficulties

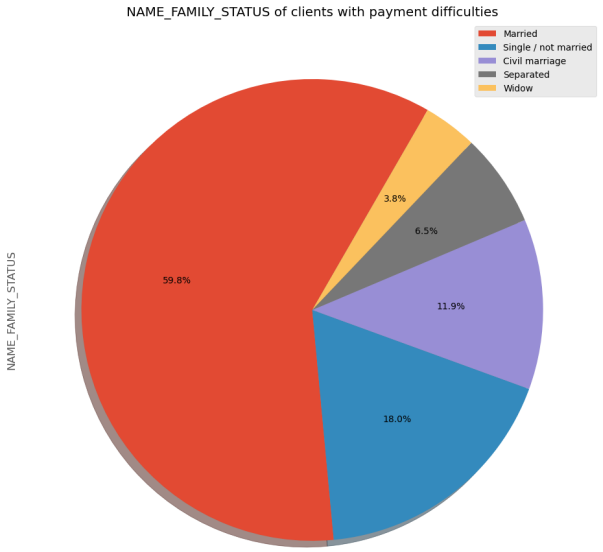


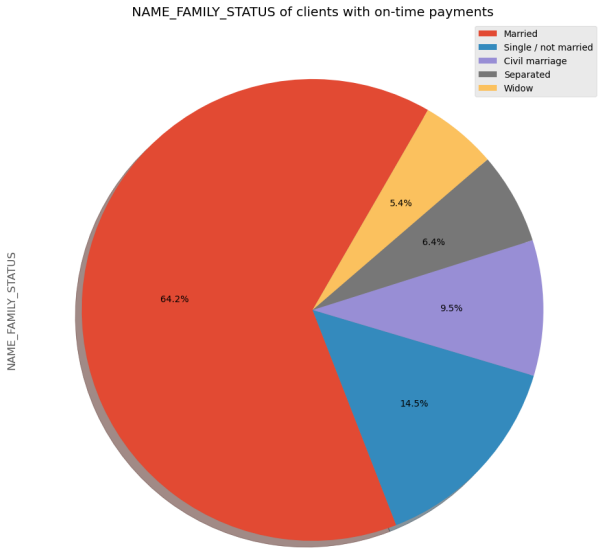
FLAG_OWN_CAR of clients with on-time payments

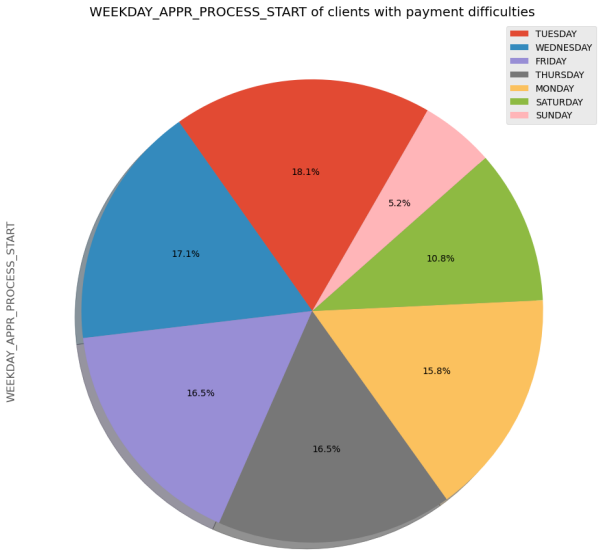


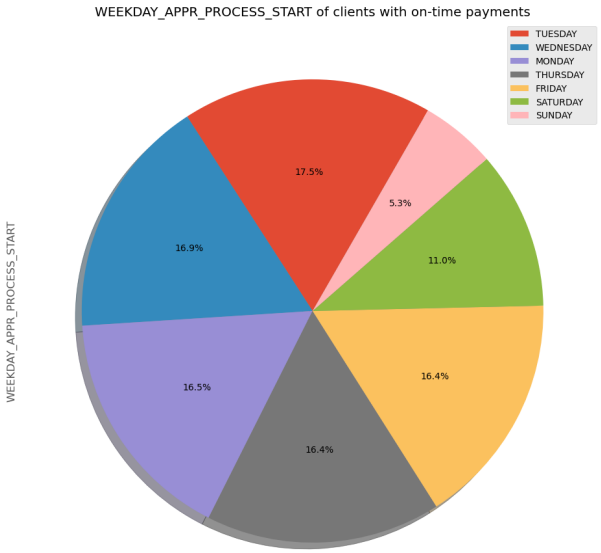












Observations: Cash loans offered are more than revolving loans. Here's the break down for defaulters vs on-time payers For defaulters: 93% Cash loan, 6.5% Revolving loan For on-time payers: 90.2% Cash loan, 9.8% Revolving loan Females have processed more loans in comparison to males. Amoung defaulters: 57.1% are females, 42.8%

are males Amoung on-time payers: 66.6% are females, 33.4% are males There is a 9.4% decrease in "Male" values from CoDE_GENDER b/w clients with payment difficulties to on-time payments. It's a weak correlation that Males have more payment difficulties. Car owners have processed less loans in comparison to those who don't own cars. Amoung defaulters: 69.5% don't own car, 30.5% own car Amoung on-time payers: 65.7% don't own car, 34.3% own car Applicants who own a house have processed more loans in comparison to those who don't. Amoung defaulters: 68.5% own house, 30.5% don't own a house Amoung on-time payers: 69.5% own house, 30.5% don't own a house 'Working' class have processed more loans in comparison to other categories. Pensioners have better on-time payments (Weak correlation as we have less number of pentioners comparatively) Students don't have Payment difficulties (Weak correlation as total students have only 18 observations) Businessmen don't have Payment difficulties (Weak correlation as total Businessmen have only 10 observations) Clients with 'Higher education' have better on-timepayments than payment difficulties and have less payment difficulties. However, this is a weak correlation. Clients who are 'Married' oR 'Widow' do on-time payments better comparatively. Clients who are 'Single/not married' have more difficulties with on-time payments comparatively. However, this is a weak correlation. Amoung defaulters: 59.8% are married, 3.8% are widow, 18.0% single/not married Amoung on-time payers: 64.2%% are married, 5.4% are widow, 14.5% single/not married

Analysis of numeric columns

In [ ]:
```
# checking out total number of numeric columns
app_data.columns[(app_data.dtypes=="int64") | (app_data.dtypes=="float64")]
```

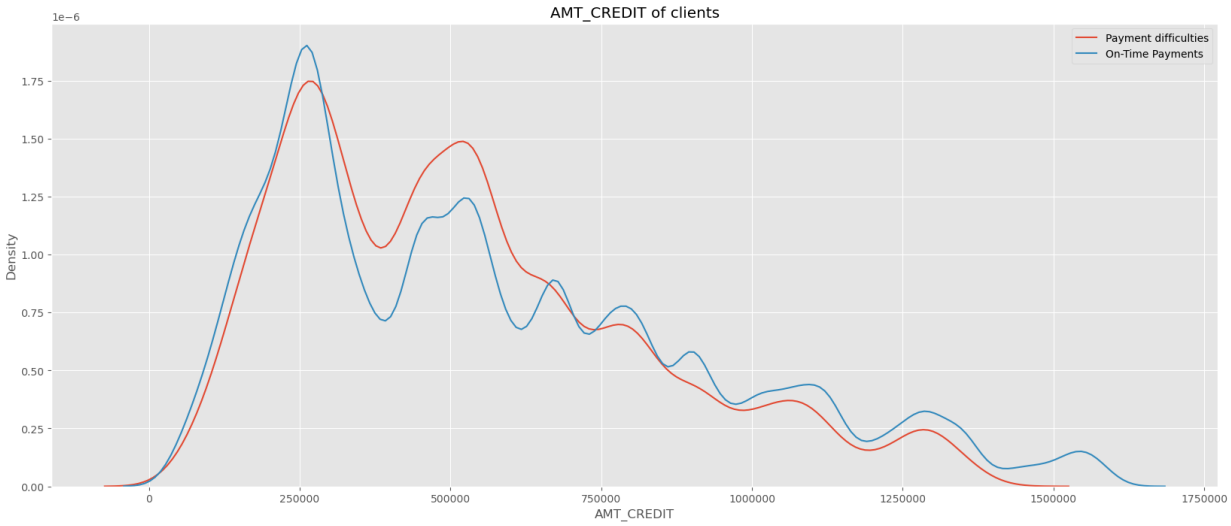
Out[ ]:
```
Index(['SK_ID_CURR', 'TARGET', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
       'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED',
       'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'FLAG_MOBIL', 'FLAG_EMP_PHONE',
       'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE', 'FLAG_EMAIL',
       'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT',
       'REGION_RATING_CLIENT_W_CITY', 'HOUR_APPR_PROCESS_START',
       'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
       'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
       'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'EXT_SOURCE_2',
       'EXT_SOURCE_3', 'YEARS_BEGINEXPLUATATION_AVG', 'FLOORSMAX_AVG',
       'YEARS_BEGINEXPLUATATION_MODE', 'FLOORSMAX_MODE',
       'YEARS_BEGINEXPLUATATION_MEDI', 'FLOORSMAX_MEDI', 'TOTALAREA_MODE',
       'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE',
       'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE',
       'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3',
       'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6',
       'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9',
       'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12',
       'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15',
       'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18',
       'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21',
       'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY',
       'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON',
       'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR',
       'YEARS_BIRTH', 'YEARS_EMPLOYED', 'YEARS_REGISTRATION',
       'YEARS_ID_PUBLISH', 'YEARS_LAST_PHONE_CHANGE'],
      dtype='object')
```

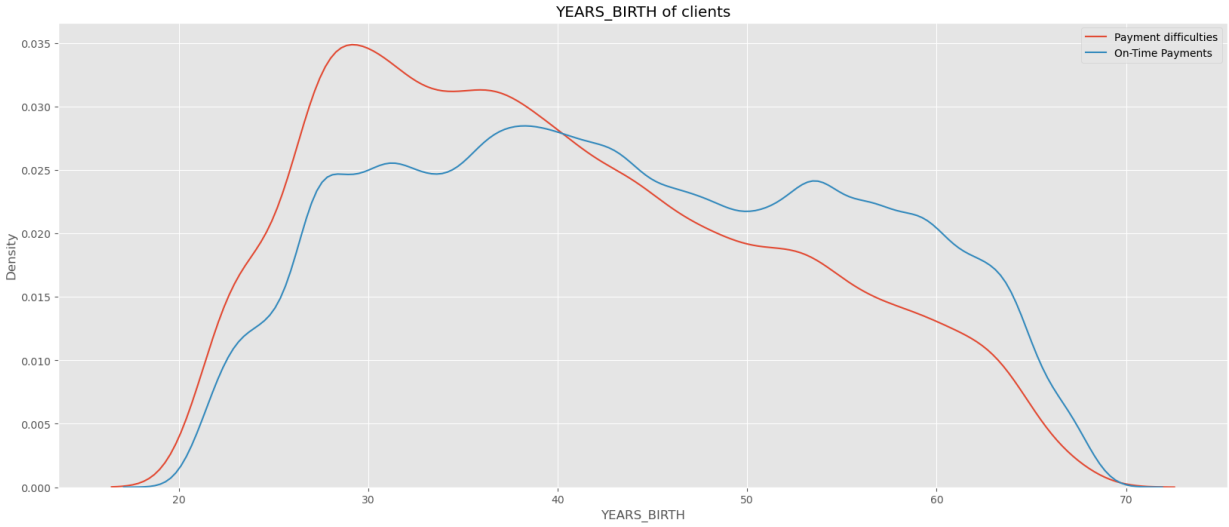
observations:

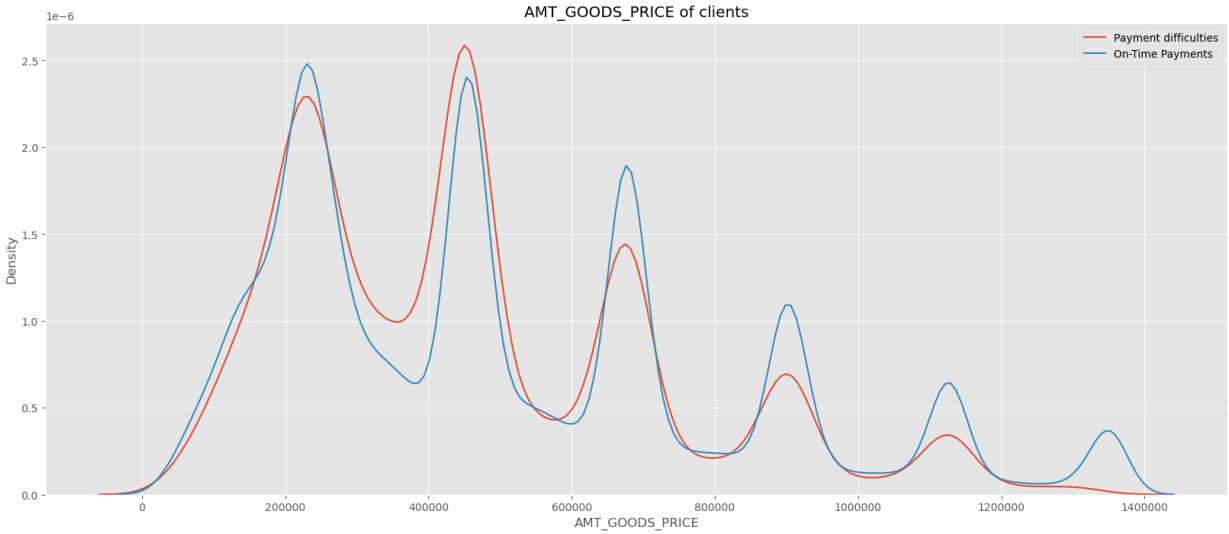
Many columns with int data type are Flag columns. For purpose of calculations we will keep them as int and delete a few unwanted ones. Eg:FLAG_DOCUMENT column group, REG_CITY_NOT_LIVE_CITY etc.

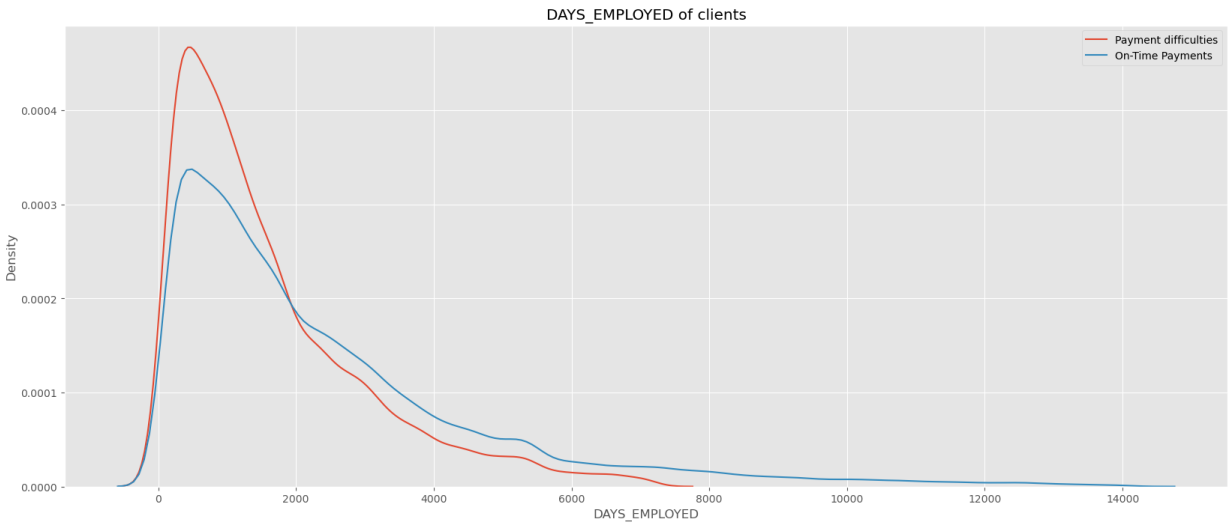
In [ ]:
```
# deleting all the Flag columns
for i in app_data.columns:
    if i.startswith("FLAG"):
        app_data.drop(columns=i, inplace=True)
```

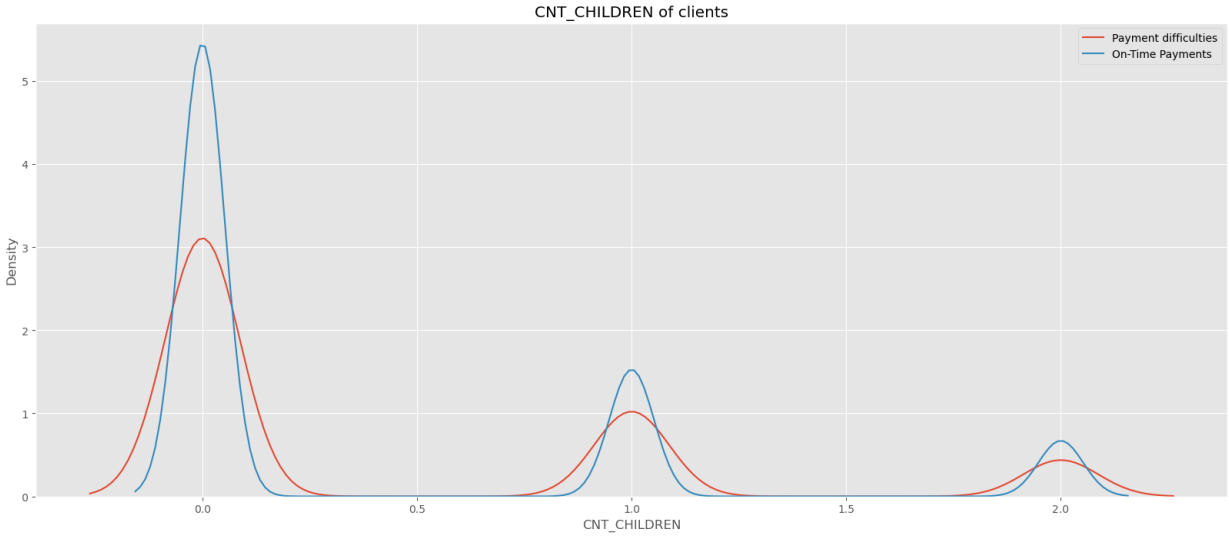
In [ ]:
```
# plotting a dist plot without outliers
d_plot=['AMT_CREDIT','YEARS_BIRTH','AMT_GOODS_PRICE','DAYS_EMPLOYED','CNT_CHILDREN','A
for i in d_plot:
    # calculationg IQR for clients with payment difficulties
    df1_Q1 = df1[i].quantile(0.25)
    df1_Q3 = df1[i].quantile(0.75)
    df1_IQR = df1_Q3 - df1_Q1
    Min_value1 = (df1_Q1 - 1.5 * df1_IQR)
    Max_value1 = (df1_Q3 + 1.5 * df1_IQR)
    # calculationg IQR for clients with on-time payments
    df0_Q1 = df0[i].quantile(0.25)
    df0_Q3 = df0[i].quantile(0.75)
    df0_IQR = df0_Q3 - df1_Q1
    Min_value0 = (df0_Q1 - 1.5 * df0_IQR)
    Max_value0 = (df0_Q3 + 1.5 * df0_IQR)
    #Removing outliers and plotting distplot
    plt.figure(figsize = [20,8])
    sns.distplot(df1[df1[i] <= Max_value1][i],label = 'Payment difficulties', hist=Fal
    sns.distplot(df0[df0[i] <= Max_value0][i],label = 'On-Time Payments', hist=False)
    plt.title(f'{i} of clients')
    plt.ticklabel_format(style='plain', axis='x')
    plt.legend()
    plt.show()
```

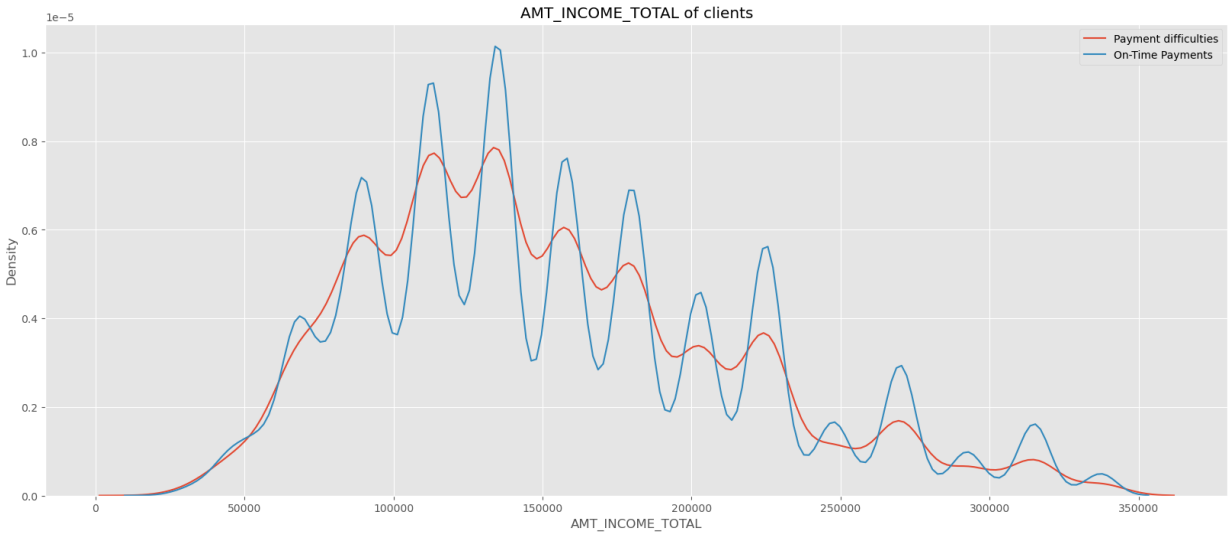



YEARS_BIRTH of clients
Payment difficulties
On-Time Payments
Density
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
20
30
40
50
60
70
YEARS_BIRTH


AMT_GOODS_PRICE of clients
1e−6
Payment difficulties
On-Time Payments
Density
2.5
2.0
1.5
1.0
0.5
0.0
0
200000
400000
600000
800000
1000000
1200000
1400000
AMT_GOODS_PRICE


DAYS_EMPLOYED of clients
Payment difficulties
On-Time Payments
Density
0.0004
0.0003
0.0002
0.0001
0.0000
0
2000
4000
6000
8000
10000
12000
14000
DAYS_EMPLOYED

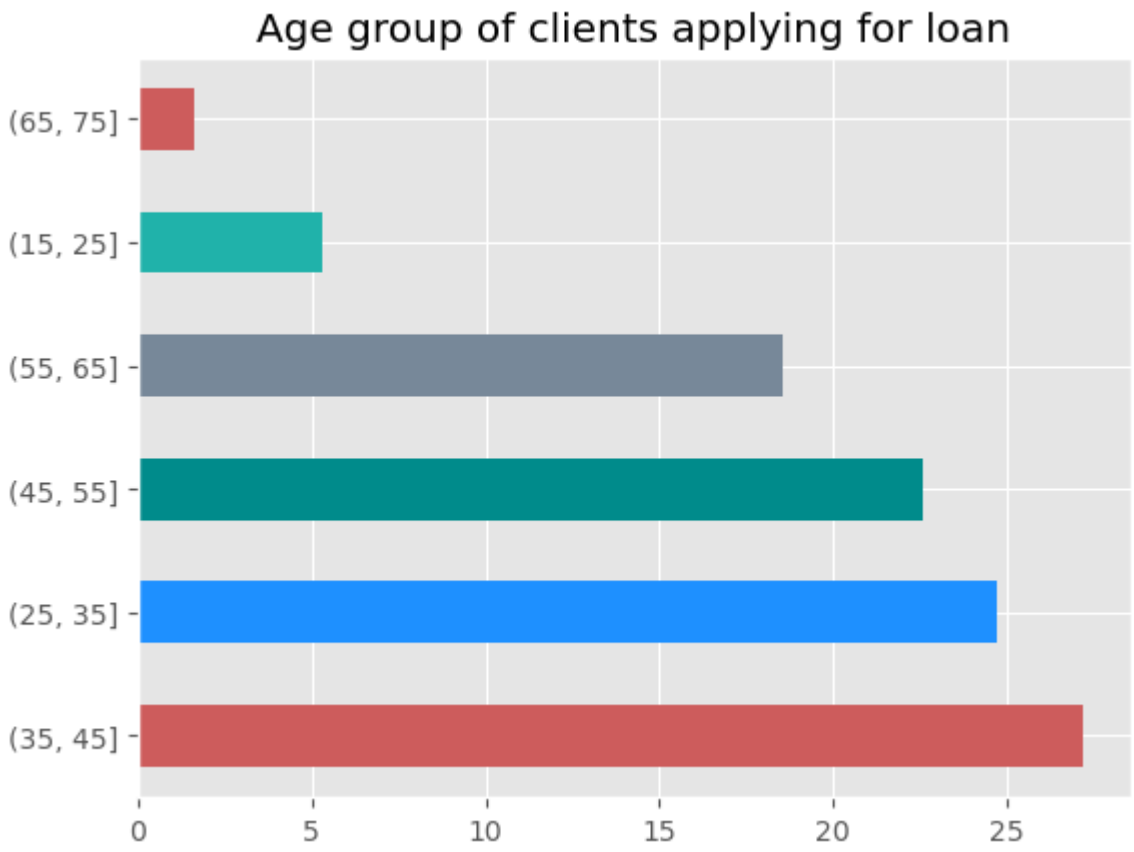




Observations: For AMT_CREDIT between 250000 and approximately 650000, there are more clients with Payment difficulties For AMT_CREDIT > 750000 , there are more clients with on-Time Payments For YEARS_BIRTH between 20 and 40, there are more clients with Payment difficulties For YEARS_BIRTH > 40 , there are more clients with on-Time Payments For AMT_GOODS_PRICE between ~250000 and ~550000, there are more clients with Payment difficulties For DAYS_EMPLOYED less than 2000, there are more clients with Payment difficulties For DAYS_EMPLOYED > 2000 , there are more clients with on-Time Payments, impluing that those who are employed longer have better chances of repaying the loan For CNT_CHILDREN=0 (those with no children), there are lots of clients with on-Time Payments For CNT_CHILDREN with 1 oR 2 (those with 1 or 2 children), there are few more clients with on-Time Payments For clients with Payment difficulties, the AMT_INCOME_TOTAL distribution resembles a normal distribution approximately

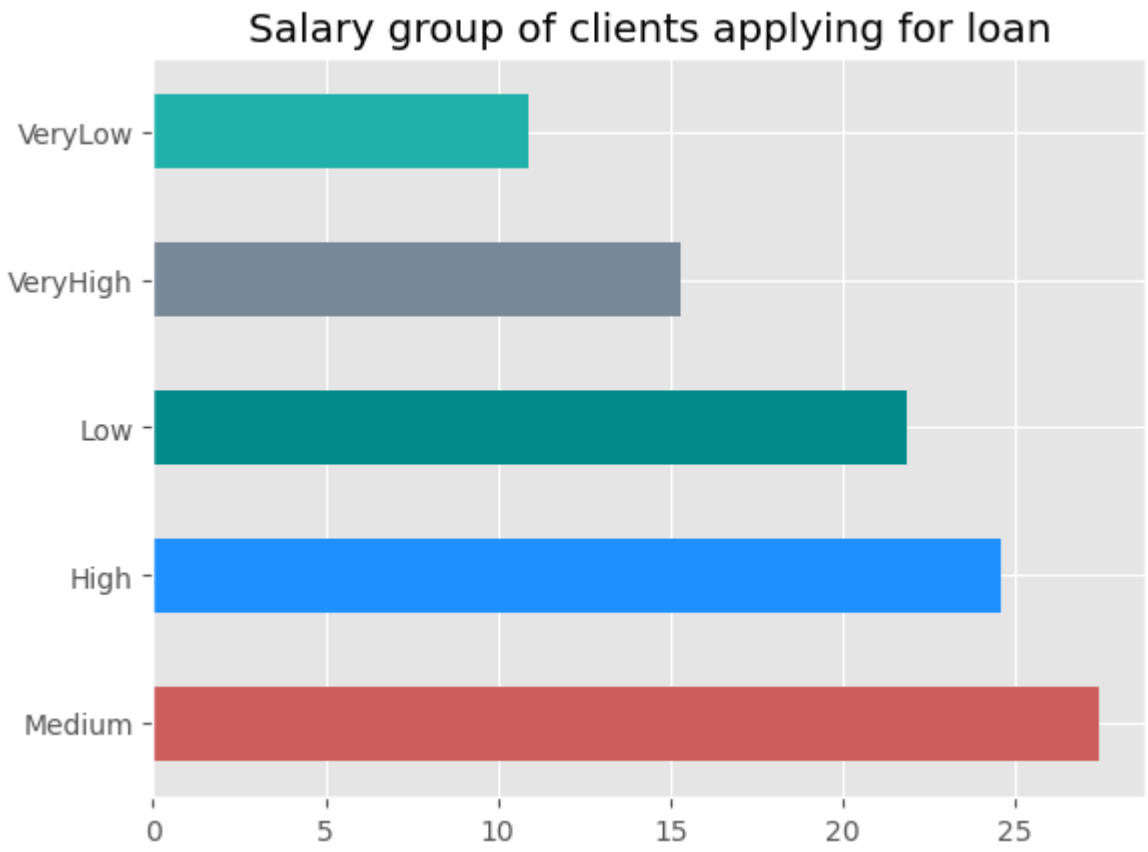
Analysis of numeric columns by Binning

In [ ]:
```
# binning YEARS_BIRTH and creating new column "AGE_GROUP"
app_data['AGE_GROUP']= pd.cut(app_data['YEARS_BIRTH'],bins=[15,25,35,45,55,65,75])
(app_data['AGE_GROUP'].value_counts(normalize=True)*100).plot.barh(title ="Age group c
plt.xticks(rotation=0)
plt.show()
```



Observations: 35-45 Age group is the largest Group of Age applying for loans.

In [ ]:
```
# binning 'AMT_INCOME_TOTAL' and Creating new column "INCOME_GROUP"
app_data['INCOME_GROUP']= pd.qcut(app_data['AMT_INCOME_TOTAL'], q=[0,0.1,0.3,0.6,0.8,1
(app_data['INCOME_GROUP'].value_counts(normalize=True)*100).plot.barh(title ="Salary g
plt.xticks(rotation=0)
plt.show()
```



Observations: 'Medium' Income group is the largest group applying for loans, followed by 'High' income group. 'VeryLow' income group is the smallest group applying for loan.

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

Bivariate/Multivariate analysis

In [ ]:
```
l1=['AMT_GOODS_PRICE','AMT_ANNUITY','DAYS_EMPLOYED','DAYS_BIRTH']
for a in range(len(l1)):
     print(l1[a])
```

```
AMT_GOODS_PRICE
AMT_ANNUITY
DAYS_EMPLOYED
DAYS_BIRTH
```

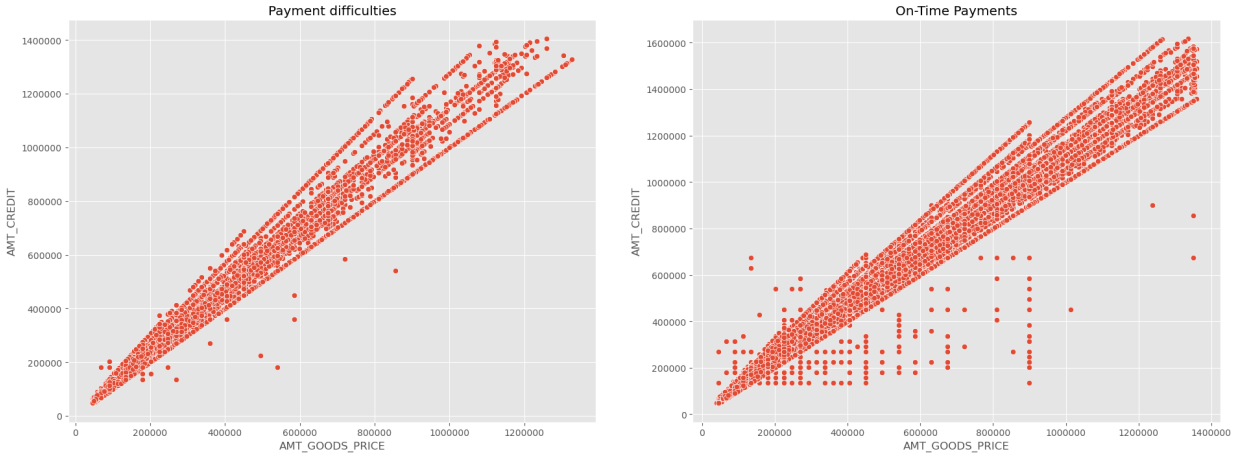
Analysis of Continuous V/S Continuous variables

In [ ]:
```
# function to calculate min max value for IQR
def outlier_range(dataset,column):
    Q1 = dataset[column].quantile(0.25)
```

```
    Q3 = dataset[column].quantile(0.75)
    IQR = Q3 - Q1
    Min_value = (Q1 - 1.5 * IQR)
    Max_value = (Q3 + 1.5 * IQR)
    return Max_value
```

In [ ]:
```
# outlier analysis of AMT_GOODS_PRICE V/S AMT_CREDIT
max_value1_AMT_GOODS_PRICE = outlier_range(df1,'AMT_GOODS_PRICE')
max_value1_AMT_CREDIT = outlier_range(df1,'AMT_CREDIT')
max_value0_AMT_GOODS_PRICE = outlier_range(df0,'AMT_GOODS_PRICE')
max_value0_AMT_CREDIT = outlier_range(df0,'AMT_CREDIT')
```

In [ ]:
```
# plotting a scatter plot to see the relation
plt.figure(figsize = [20,8])
plt.subplot(1,2,1)
plt.title('Payment difficulties')
sns.scatterplot(x = df1[df1['AMT_GOODS_PRICE'] < max_value1_AMT_GOODS_PRICE].AMT_GOODS
plt.ticklabel_format(style='plain', axis='x')
plt.ticklabel_format(style='plain', axis='y')
plt.subplot(1,2,2)
plt.title('On-Time Payments')
sns.scatterplot(x = df0[df0['AMT_GOODS_PRICE'] < max_value0_AMT_GOODS_PRICE].AMT_GOODS
plt.ticklabel_format(style='plain', axis='x')
plt.ticklabel_format(style='plain', axis='y')
plt.tight_layout(pad = 4)
plt.show()
```

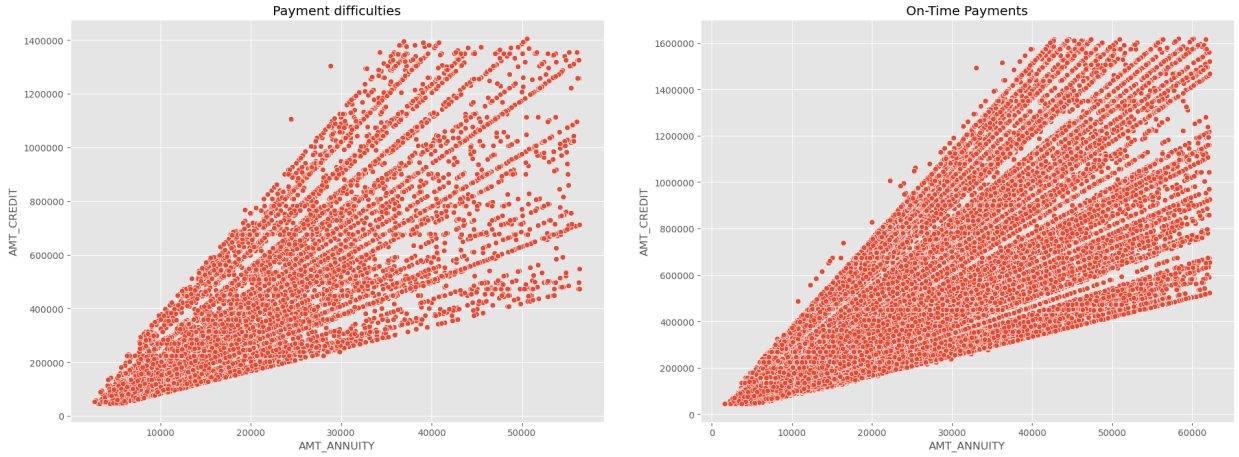


Observations: AMT_GOODS_PRICE and AMT_CREDIT have strong positive correlation. This means that as Goods price increases, so does Credit Amount

In [ ]:
```
# outlier analysis of AMT_ANNUITY V/S AMT_CREDIT
max_value1_AMT_ANNUITY = outlier_range(df1,'AMT_ANNUITY')
max_value1_AMT_CREDIT = outlier_range(df1,'AMT_CREDIT')
max_value0_AMT_ANNUITY = outlier_range(df0,'AMT_ANNUITY')
max_value0_AMT_CREDIT = outlier_range(df0,'AMT_CREDIT')
```

In [ ]:
```
# plotting a scatter plot to see the relation
plt.figure(figsize = [20,8])
plt.subplot(1,2,1)
plt.title('Payment difficulties')
sns.scatterplot(x = df1[df1['AMT_ANNUITY'] < max_value1_AMT_ANNUITY].AMT_ANNUITY, y =
plt.ticklabel_format(style='plain', axis='x')
plt.ticklabel_format(style='plain', axis='y')
plt.subplot(1,2,2)
```

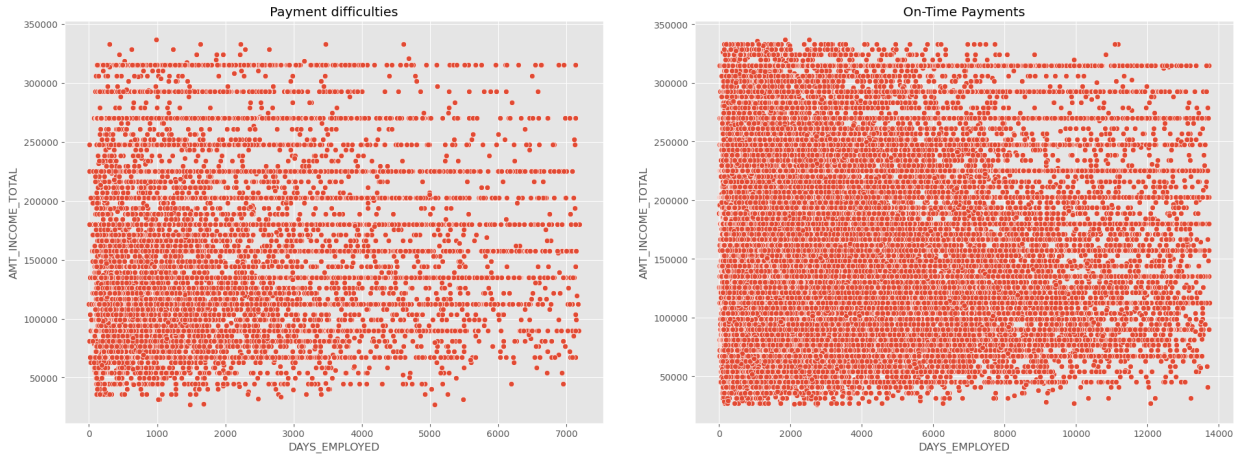
```
plt.title('On-Time Payments')
sns.scatterplot(x = df0[df0['AMT_ANNUITY'] < max_value0_AMT_ANNUITY].AMT_ANNUITY, y =
plt.ticklabel_format(style='plain', axis='x')
plt.ticklabel_format(style='plain', axis='y')
plt.tight_layout(pad = 4)
plt.show()
```



Observations: AMT_ANNUITY and AMT_CREDIT have strong positive correlation. This means that as Annuity Amount increases, so does Credit Amount

In [ ]:
```
# outlier analysis of DAYS_EMPLOYED V/S AMT_INCOME_TOTAL
max_value1_DAYS_EMPLOYED = outlier_range(df1,'DAYS_EMPLOYED')
max_value1_AMT_INCOME_TOTAL = outlier_range(df1,'AMT_INCOME_TOTAL')
max_value0_DAYS_EMPLOYED = outlier_range(df0,'DAYS_EMPLOYED')
max_value0_AMT_INCOME_TOTAL = outlier_range(df0,'AMT_INCOME_TOTAL')
```

In [ ]:
```
# plotting a scatter plot to see the relation
plt.figure(figsize = [20,8])
plt.subplot(1,2,1)
plt.title('Payment difficulties')
sns.scatterplot(x = df1[df1['DAYS_EMPLOYED'] < max_value1_DAYS_EMPLOYED].DAYS_EMPLOYED
plt.ticklabel_format(style='plain', axis='x')
plt.ticklabel_format(style='plain', axis='y')
plt.subplot(1,2,2)
plt.title('On-Time Payments')
sns.scatterplot(x = df0[df0['DAYS_EMPLOYED'] < max_value0_DAYS_EMPLOYED].DAYS_EMPLOYED
plt.ticklabel_format(style='plain', axis='x')
plt.ticklabel_format(style='plain', axis='y')
plt.tight_layout(pad = 4)
plt.show()
```



Observations: Clients who are employed for a long time (>7000) days are making their payments on-time but these category of clients do not exist in Payments difficulties group Even looking at Payment difficulties group, clients with more than 4000 days of employment are sparse

In [ ]:
```
# outlier analysis of AMT_ANNUITY V/S AMT_GOODS_PRICE
max_value1_AMT_CREDIT = outlier_range(df1,'AMT_ANNUITY')
max_value1_DAYS_BIRTH = outlier_range(df1,'AMT_GOODS_PRICE')
max_value0_AMT_CREDIT = outlier_range(df0,'AMT_ANNUITY')
max_value0_DAYS_BIRTH = outlier_range(df0,'AMT_GOODS_PRICE')
```

In [ ]:
```
# plotting a scatter plot to see the relation
plt.figure(figsize = [20,8])
plt.subplot(1,2,1)
plt.title('Payment difficulties')
sns.scatterplot(x = df1[df1['AMT_ANNUITY'] < max_value1_AMT_ANNUITY].AMT_ANNUITY, y =
plt.ticklabel_format(style='plain', axis='x')
plt.ticklabel_format(style='plain', axis='y')
plt.subplot(1,2,2)
plt.title('On-Time Payments')
sns.scatterplot(x = df0[df0['AMT_ANNUITY'] < max_value0_AMT_ANNUITY].AMT_ANNUITY, y =
plt.ticklabel_format(style='plain', axis='x')
plt.ticklabel_format(style='plain', axis='y')
plt.tight_layout(pad = 4)
plt.show()
```

Observations: AMT_ANNUITY and AMT_GOODS_PRICE have strong positive correlation. This means that as Annuity increases, so does Goods Price

Analysis of Continuous V/S Categorical variables

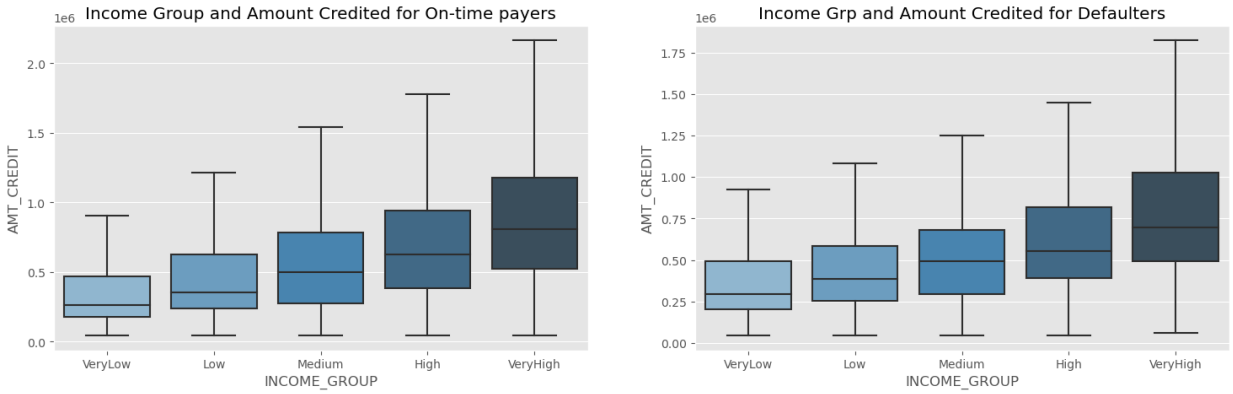
In [ ]:
```
# outlier analysis of NAME_EDUCATION_TYPE V/S AMT_CREDIT V/S CODE_GENDER
max_value1_AMT_CREDIT = outlier_range(df1,'AMT_CREDIT')
max_value0_AMT_CREDIT = outlier_range(df0,'AMT_CREDIT')
```

In [ ]:
```
df1.groupby(by = ['NAME_EDUCATION_TYPE','CODE_GENDER']).AMT_CREDIT.describe().head()
```

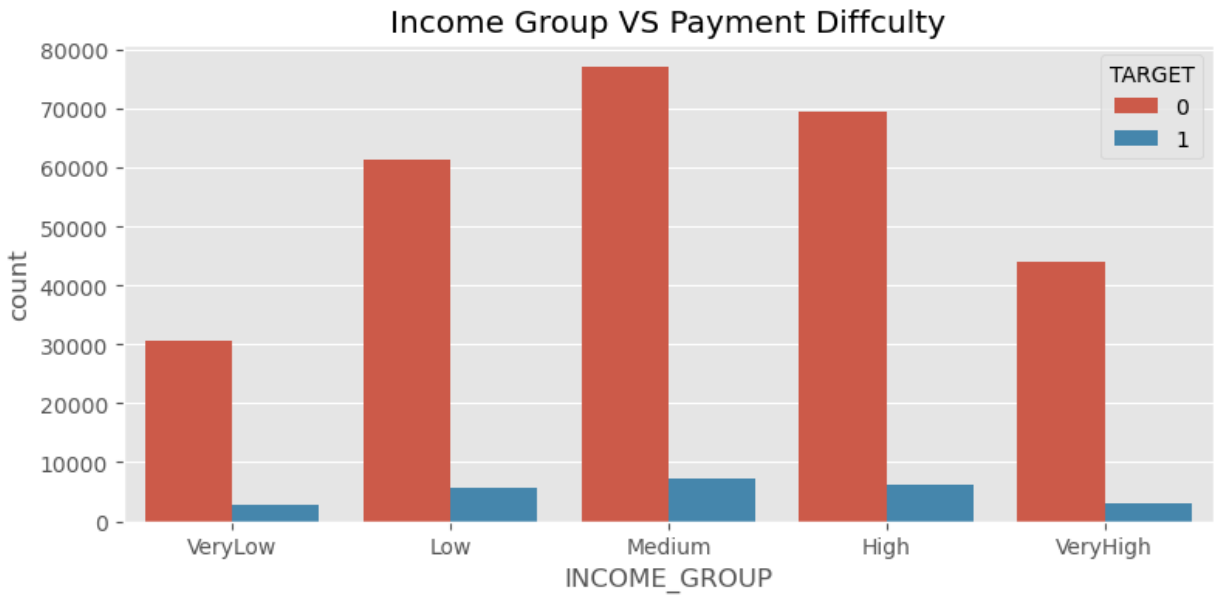
Out[ ]:

NAME_EDUCATION_TYPE	CODE_GENDER	count	mean	std	min	25%	
Academic degree	F	3.0	950245.500000	504711.375348	544491.0	667660.5	79
Higher education	F	2438.0	648500.581624	410415.703440	47970.0	323595.0	54
	M	1571.0	642070.944940	409695.934841	45000.0	329202.0	54
Incomplete higher	F	504.0	520150.348214	352227.493127	91692.0	269550.0	45
	M	368.0	554723.816576	347843.706372	74628.0	284400.0	49

In [ ]:
```
# function to perform categorical analysis
df0=app_data[app_data['TARGET']==0]
df1=app_data[app_data['TARGET']==1]
plt.figure(figsize = (18,5))
plt.subplot(1,2,1)
plt.title("Income Group and Amount Credited for On-time payers")
sns.boxplot(x = 'INCOME_GROUP', y = 'AMT_CREDIT', data =df0, showfliers=False, palette
plt.subplot(1,2,2)
plt.title("Income Grp and Amount Credited for Defaulters")
sns.boxplot(x = 'INCOME_GROUP', y = 'AMT_CREDIT', data = df1, showfliers=False, palett
plt.show()
```

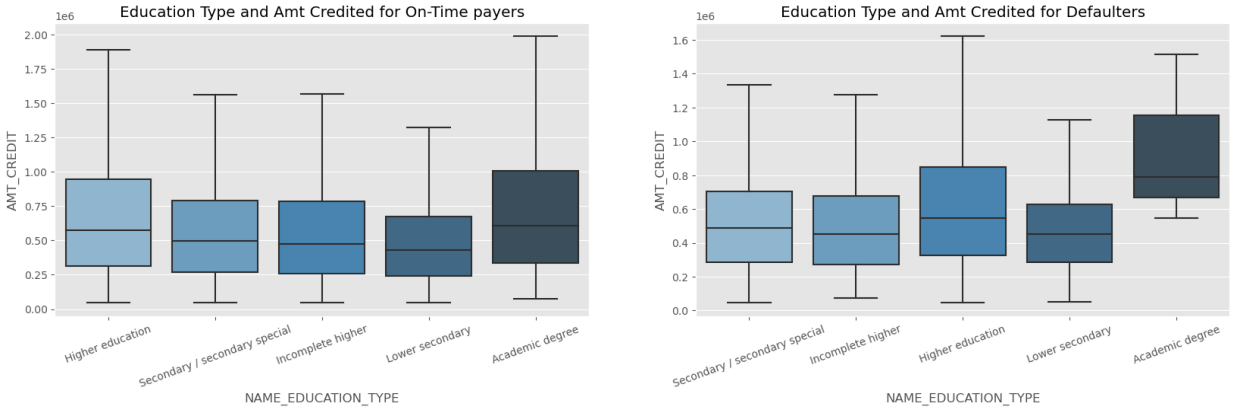


In [ ]:
```
#Checking the same columns to see the affect of Very High Income group
plt.figure(figsize = (20,4))
plt.subplot(1,2,1)
plt.title("Income Group VS Payment Diffculty")
sns.countplot('INCOME_GROUP', hue = 'TARGET', data =app_data)
plt.show()
```



Observations: We can infer that though the maximum no of loans is given to Medium income group. Default value per loan is highest in High income group as the AMT_CREDIT is higher too. The loan book of the financial institution can get affected due to higher amount not being paid back. The company must devise a different set of rules and policies while approving higher income group loans.

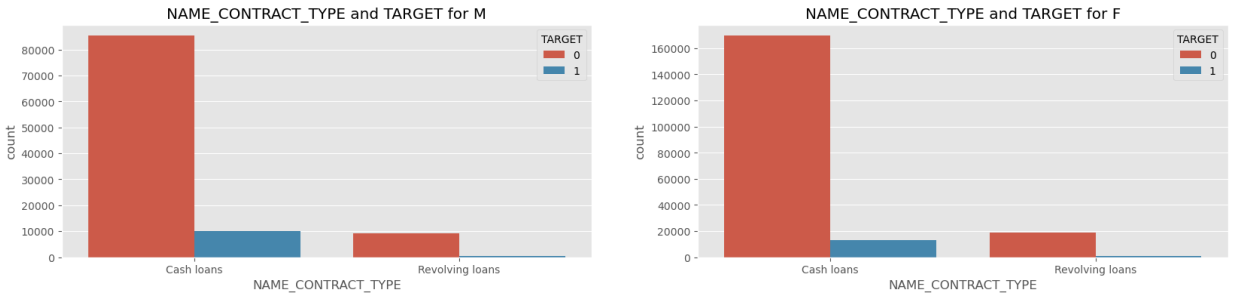
In [ ]:
```
# plot for Education Type and Amt Credited
plt.figure(figsize = (20,5))
plt.subplot(1,2,1)
plt.title("Education Type and Amt Credited for On-Time payers")
sns.boxplot(x = 'NAME_EDUCATION_TYPE', y = 'AMT_CREDIT', data =df0,showfliers=False, p
plt.xticks(rotation=20, fontsize= 10,fontweight= 2)
plt.subplot(1,2,2)
plt.title("Education Type and Amt Credited for Defaulters")
sns.boxplot(x = 'NAME_EDUCATION_TYPE', y = 'AMT_CREDIT', data = df1,showfliers=False,
plt.xticks(rotation=20, fontsize= 10,fontweight= 2)
plt.show()
```



Observations: Median of Loan values defaulting for Applicants with Academic degree is higher. But as we saw in a plot above, no of applicants with academic degree is miniscule No inference can be drawn from this analysis.

In [ ]:
```
# plot to check male Vs female default rate
plt.figure(figsize = (20,4))
plt.subplot(1,2,1)
plt.title("NAME_CONTRACT_TYPE and TARGET for M")
sns.countplot('NAME_CONTRACT_TYPE', hue = "TARGET", data=app_data[(app_data['CODE_GEND
plt.subplot(1,2,2)
plt.title("NAME_CONTRACT_TYPE and TARGET for F")
```

```
sns.countplot('NAME_CONTRACT_TYPE', hue = "TARGET", data=app_data[(app_data['CODE_GEND
plt.show()
```

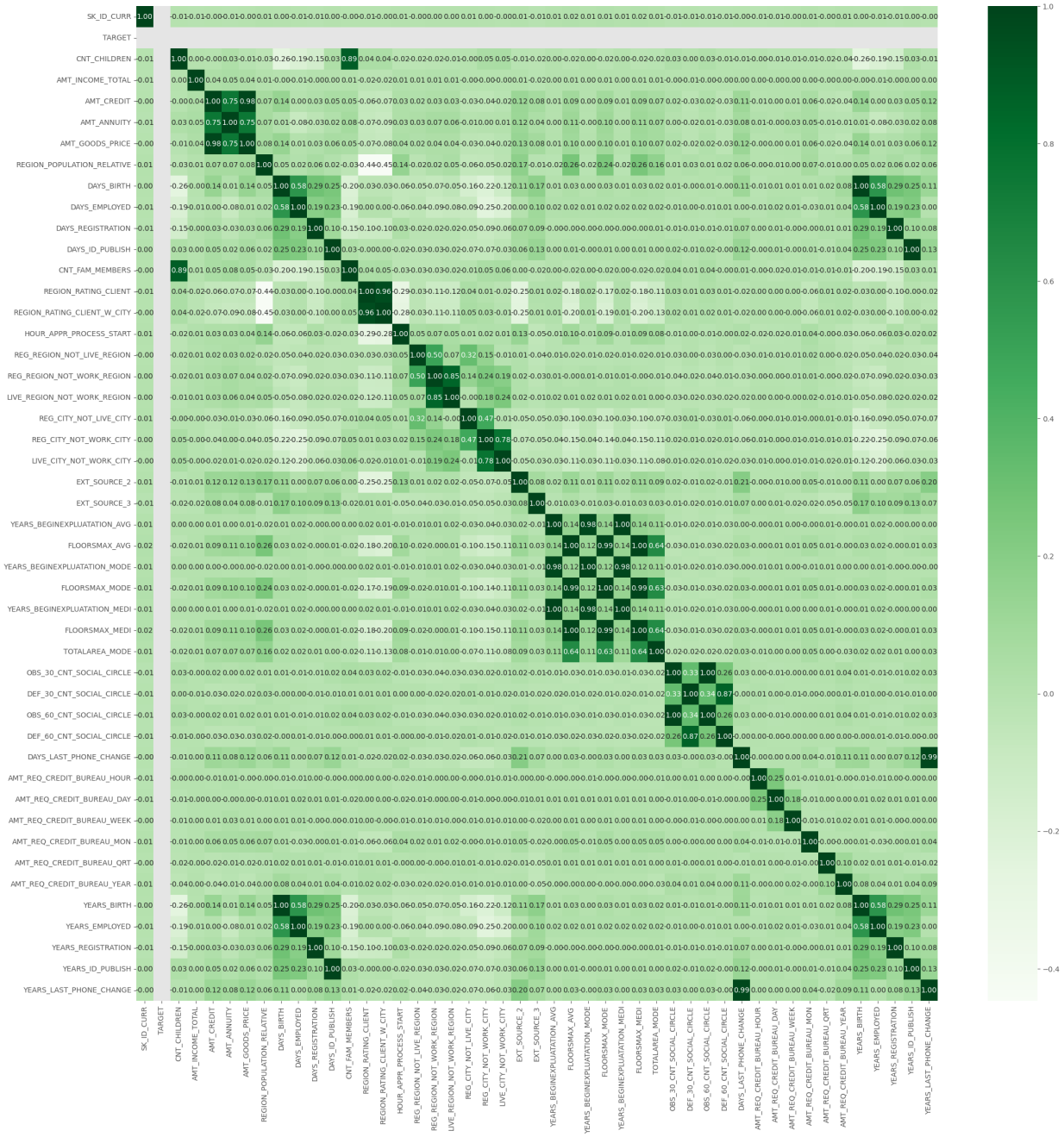


Observations: Male applicants are defaulting more that female applicants

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.

Top Correlation

In [ ]:
```
# creating a HeatMap to view the correlations above 80% and 99.99%
for i in app_data.columns:
    if i.startswith("FLAG"):
        app_data.drop(columns=i, inplace=True)
corr_df1 = df1.select_dtypes(include=["int64","float64"]).corr()

plt.figure(figsize = (25,25))
sns.heatmap(data = corr_df1, annot = True, cmap = "Greens", cbar = True, fmt='.2f')
plt.show()
```



In [ ]:
```
# getting top 10 correlations for Payment Difficulties
corr_df1[corr_df1 <= 0.99].unstack().sort_values(ascending = False).head(20)
```

Out[ ]:

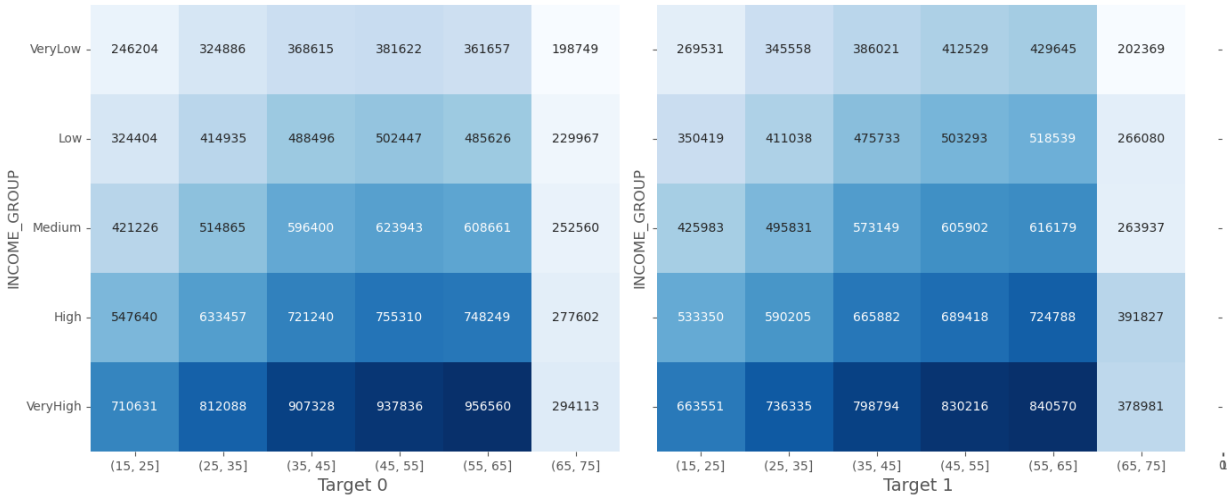
FLOORSMAX_MODE	FLOORSMAX_MEDI	0.989195
FLOORSMAX_MEDI	FLOORSMAX_MODE	0.989195
YEARS_LAST_PHONE_CHANGE	DAYS_LAST_PHONE_CHANGE	0.988086
DAYS_LAST_PHONE_CHANGE	YEARS_LAST_PHONE_CHANGE	0.988086
FLOORSMAX_AVG	FLOORSMAX_MODE	0.986594
FLOORSMAX_MODE	FLOORSMAX_AVG	0.986594
AMT_GOODS_PRICE	AMT_CREDIT	0.983103
AMT_CREDIT	AMT_GOODS_PRICE	0.983103
YEARS_BEGINEXPLUATATION_MODE	YEARS_BEGINEXPLUATATION_AVG	0.980466
YEARS_BEGINEXPLUATATION_AVG	YEARS_BEGINEXPLUATATION_MODE	0.980466
YEARS_BEGINEXPLUATATION_MODE	YEARS_BEGINEXPLUATATION_MEDI	0.978073
YEARS_BEGINEXPLUATATION_MEDI	YEARS_BEGINEXPLUATATION_MODE	0.978073
REGION_RATING_CLIENT_W_CITY	REGION_RATING_CLIENT	0.956637
REGION_RATING_CLIENT	REGION_RATING_CLIENT_W_CITY	0.956637
CNT_FAM_MEMBERS	CNT_CHILDREN	0.885484
CNT_CHILDREN	CNT_FAM_MEMBERS	0.885484
DEF_30_CNT_SOCIAL_CIRCLE	DEF_60_CNT_SOCIAL_CIRCLE	0.868994
DEF_60_CNT_SOCIAL_CIRCLE	DEF_30_CNT_SOCIAL_CIRCLE	0.868994
REG_REGION_NOT_WORK_REGION	LIVE_REGION_NOT_WORK_REGION	0.847885
LIVE_REGION_NOT_WORK_REGION	REG_REGION_NOT_WORK_REGION	0.847885

dtype: float64

Observations: As we have duplicate combinations, looking at the above and removing dups, we get top 10 correlations as: AMT_GOODS_PRICE AMT_CREDIT - 0.98 REGION_RATING_CLIENT REGION_RATING_CLIENT_W_CITY - 0.96 CNT_FAM_MEMBERS CNT_CHILDREN - 0.89 DEF_60_CNT_SOCIAL_CIRCLE DEF_30_CNT_SOCIAL_CIRCLE - 0.87 REG_REGION_NOT_WORK_REGION LIVE_REGION_NOT_WORK_REGION - 0.85 LIVE_CITY_NOT_WORK_CITY REG_CITY_NOT_WORK_CITY - 0.78 AMT_ANNUITY AMT_GOODS_PRICE - 0.75 AMT_ANNUITY AMT_CREDIT - 0.75 DAYS_EMPLOYED FLAG_DOCUMENT_6 - 0.62 DAYS_BIRTH DAYS_EMPLOYED - 0.58

In [ ]:
```
#Analysing relationship of AMT_CREDIT with AGE GROUP and income group
plt.figure(figsize=[12,10])
f,(ax1,ax2,ax3) = plt.subplots(1,3,sharey=True,gridspec_kw={'width_ratios':[1,1, 0.001
res=pd.pivot_table(data=df0, index="INCOME_GROUP",columns="AGE_GROUP",values='AMT_CRED
c=sns.heatmap(res, annot=True,cmap='Blues', fmt="g", ax=ax1, cbar=False)
c.set_xlabel('Target 0 \n', fontsize=14,fontweight= 7)
c.set_xticklabels(c.get_xticklabels(), rotation = 0, fontsize = 10)
res1=pd.pivot_table(data=df1, index="INCOME_GROUP",columns="AGE_GROUP",values='AMT_CRE
c1=sns.heatmap(res1, annot=True,cmap='Blues', fmt="g", ax=ax2, cbar=False)
c1.set_xlabel('Target 1 \n', fontsize=14,fontweight= 7,)
c1.set_xticklabels(c1.get_xticklabels(), rotation = 0, fontsize = 10)
c.set_yticklabels(c.get_yticklabels(), rotation = 0, fontsize = 10)
plt.tight_layout()
plt.show()
```

<Figure size 1200x1000 with 0 Axes>



Observations: Age Group 55-65 in Very High income group has high amount credit. As explained above, this could result as loss in loan book

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.

6. Conclusion: Client categories to be targeted for providing loan

Clients in the age range 30-40 and 40-50 Clients who are employed for more than 19 years Female clients who are working Clients who are Married Male clients with Academic degree Students and Businessman Repeater clients

Pre-application analysis

In [ ]:
```
# checking the top 5 rows
pre_data.head()
```

Out[ ]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDI
0	2030495	271877	Consumer loans	1730.430	17145.0	17145
1	2802425	108129	Cash loans	25188.615	607500.0	679671
2	2523466	122040	Cash loans	15060.735	112500.0	136444
3	2819243	176158	Cash loans	47041.335	450000.0	470790
4	1784265	202054	Cash loans	31924.395	337500.0	404055

5 rows × 37 columns

Observations: The header row looks fine

In [ ]:
```
# checking the bottom 5 rows
pre_data.tail()
```

Out[ ]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT
1670209	2300464	352015	Consumer loans	14704.290	267295.5	
1670210	2357031	334635	Consumer loans	6622.020	87750.0	
1670211	2659632	249544	Consumer loans	11520.855	105237.0	
1670212	2785582	400317	Cash loans	18821.520	180000.0	
1670213	2418762	261212	Cash loans	16431.300	360000.0	

5 rows × 37 columns

Observations:

The bottom rows looks fine. There are no junk values like page number, NaN values in bottom most row

In [ ]:
```
# checking total no of rows and columns
pre_data.shape
```

Out[ ]:
```
(1670214, 37)
```

Observations: Dataframe has 1670214 rows and 37 columns

Dealing with incorrect data types - Previous Application

In [ ]:
```
# checking the info of the dataframe
pre_data.info(null_counts=True, verbose=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 
 25  CHANNEL_TYPE                 1670214 non-null  object 
 26  SELLERPLACE_AREA             1670214 non-null  int64  
 27  NAME_SELLER_INDUSTRY         1670214 non-null  object 
 28  CNT_PAYMENT                  1297984 non-null  float64
 29  NAME_YIELD_GROUP             1670214 non-null  object 
 30  PRODUCT_COMBINATION          1669868 non-null  object 
 31  DAYS_FIRST_DRAWING           997149 non-null   float64
 32  DAYS_FIRST_DUE               997149 non-null   float64
 33  DAYS_LAST_DUE_1ST_VERSION    997149 non-null   float64
 34  DAYS_LAST_DUE                997149 non-null   float64
 35  DAYS_TERMINATION             997149 non-null   float64
 36  NFLAG_INSURED_ON_APPROVAL    997149 non-null   float64
dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
```

Observations: There are a lot of columns with null values but datatypes of these columns looks fine

In [ ]:
```
# checking the statistics summary of the dataframe
pre_data.describe()
```

Out[ ]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PA
count	1.670214e+06	1.670214e+06	1.297979e+06	1.670214e+06	1.670213e+06	7.743
mean	1.923089e+06	2.783572e+05	1.595512e+04	1.752339e+05	1.961140e+05	6.697
std	5.325980e+05	1.028148e+05	1.478214e+04	2.927798e+05	3.185746e+05	2.092
min	1.000001e+06	1.000010e+05	0.000000e+00	0.000000e+00	0.000000e+00	-9.000
25%	1.461857e+06	1.893290e+05	6.321780e+03	1.872000e+04	2.416050e+04	0.000
50%	1.923110e+06	2.787145e+05	1.125000e+04	7.104600e+04	8.054100e+04	1.638
75%	2.384280e+06	3.675140e+05	2.065842e+04	1.803600e+05	2.164185e+05	7.740
max	2.845382e+06	4.562550e+05	4.180581e+05	6.905160e+06	6.905160e+06	3.060

8 rows × 21 columns

In [ ]:
```
pre_data.isna().sum()/len(pre_data)*100
```

Out[ ]:
```
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
DAYS_LAST_DUE                40.298129
DAYS_TERMINATION             40.298129
NFLAG_INSURED_ON_APPROVAL    40.298129
dtype: float64
```

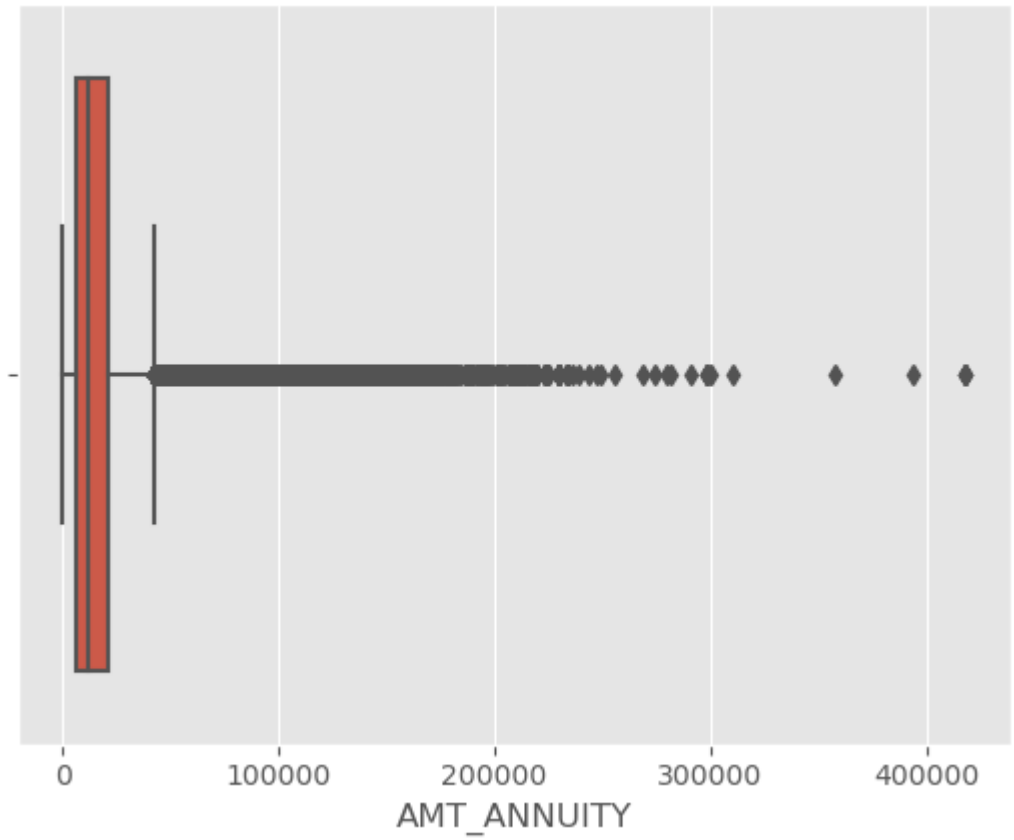
Removing columns with more than 35% null values

In [ ]:
```
nullvalue = pre_data.isnull().sum()/len(pre_data)*100
nullvalue = nullvalue[nullvalue.values>35]
nullvalue = list(nullvalue.index)
pre_data.drop(labels=nullvalue, axis=1, inplace=True)
pre_data.isnull().sum()/len(pre_data)*100
```

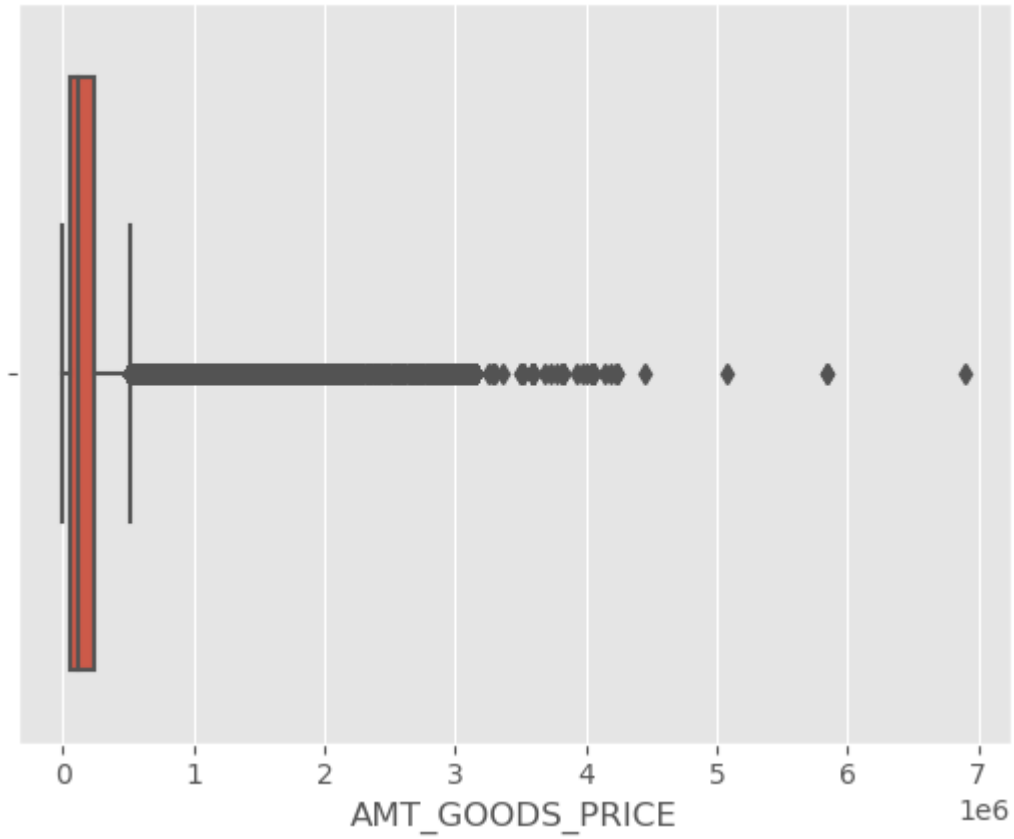
Out[ ]:
```
SK_ID_PREV                    0.000000
SK_ID_CURR                    0.000000
NAME_CONTRACT_TYPE            0.000000
AMT_ANNUITY                  22.286665
AMT_APPLICATION               0.000000
AMT_CREDIT                    0.000060
AMT_GOODS_PRICE              23.081773
WEEKDAY_APPR_PROCESS_START    0.000000
HOUR_APPR_PROCESS_START       0.000000
FLAG_LAST_APPL_PER_CONTRACT   0.000000
NFLAG_LAST_APPL_IN_DAY        0.000000
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                  22.286366
NAME_YIELD_GROUP              0.000000
PRODUCT_COMBINATION           0.020716
dtype: float64
```

Observations: Null values have to be imputed for the columns AMT_ANNUITY :imputing missing values with median as there are outliers in AMT_ANNUITY AMT_GOODS_PRICE :imputing missing values with median as there are outliers in AMT_GOODS_PRICE CNT_PAYMENT :imputing missing values with median as there are outliers in CNT_PAYMENT PRODUCT_COMBINATION: imputing missing values with mode as it is categorical data

In [ ]:
```
sns.boxplot(pre_data.AMT_ANNUITY)
pre_data.AMT_ANNUITY.fillna(pre_data.AMT_ANNUITY.median(), inplace=True)
```

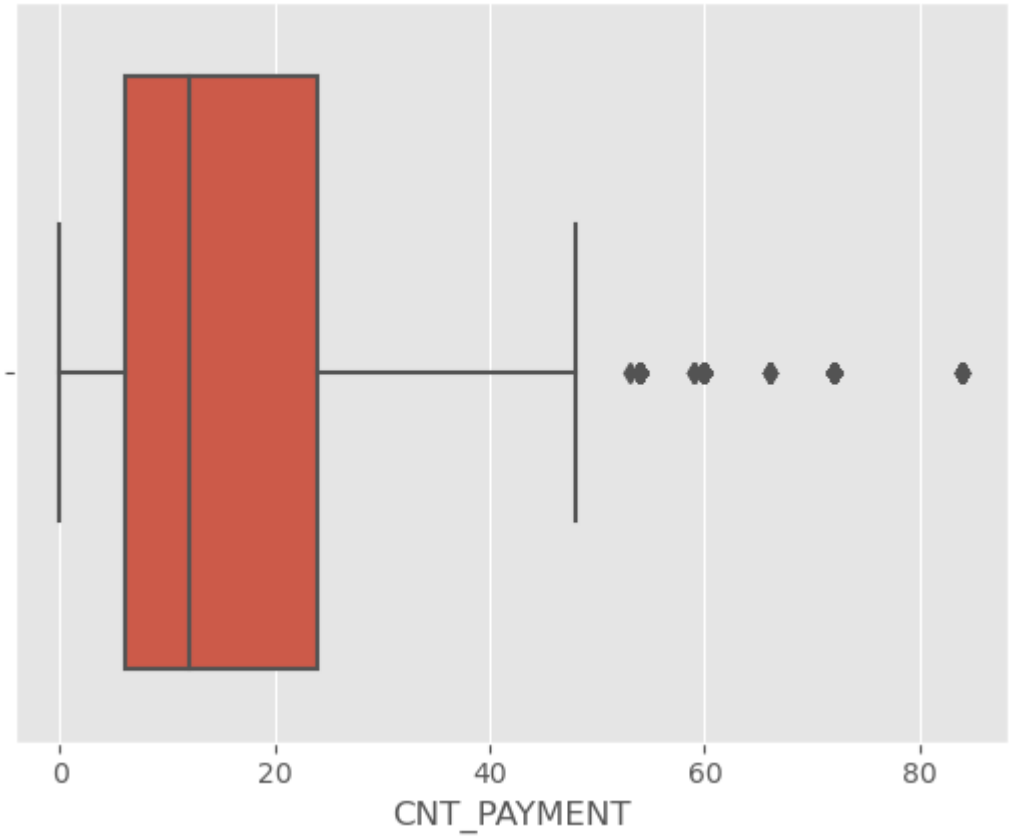


In [ ]:
```
sns.boxplot(pre_data.AMT_GOODS_PRICE)
pre_data.AMT_GOODS_PRICE.fillna(pre_data.AMT_GOODS_PRICE.median(), inplace=True)
```



In [ ]:
```
sns.boxplot(pre_data.CNT_PAYMENT)
print(pre_data.CNT_PAYMENT.value_counts().head())
pre_data.CNT_PAYMENT.fillna(pre_data.CNT_PAYMENT.median(), inplace=True)
```

```
12.0    323049
6.0     190461
0.0     144985
10.0    141851
24.0    137764
Name: CNT_PAYMENT, dtype: int64
```

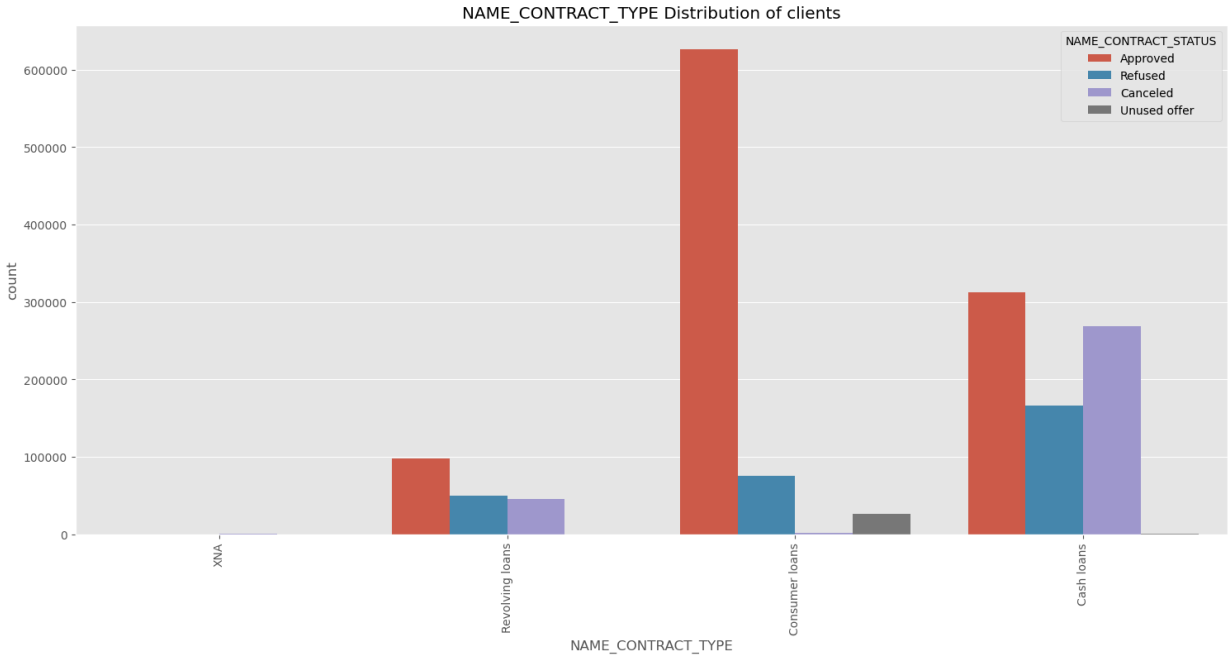


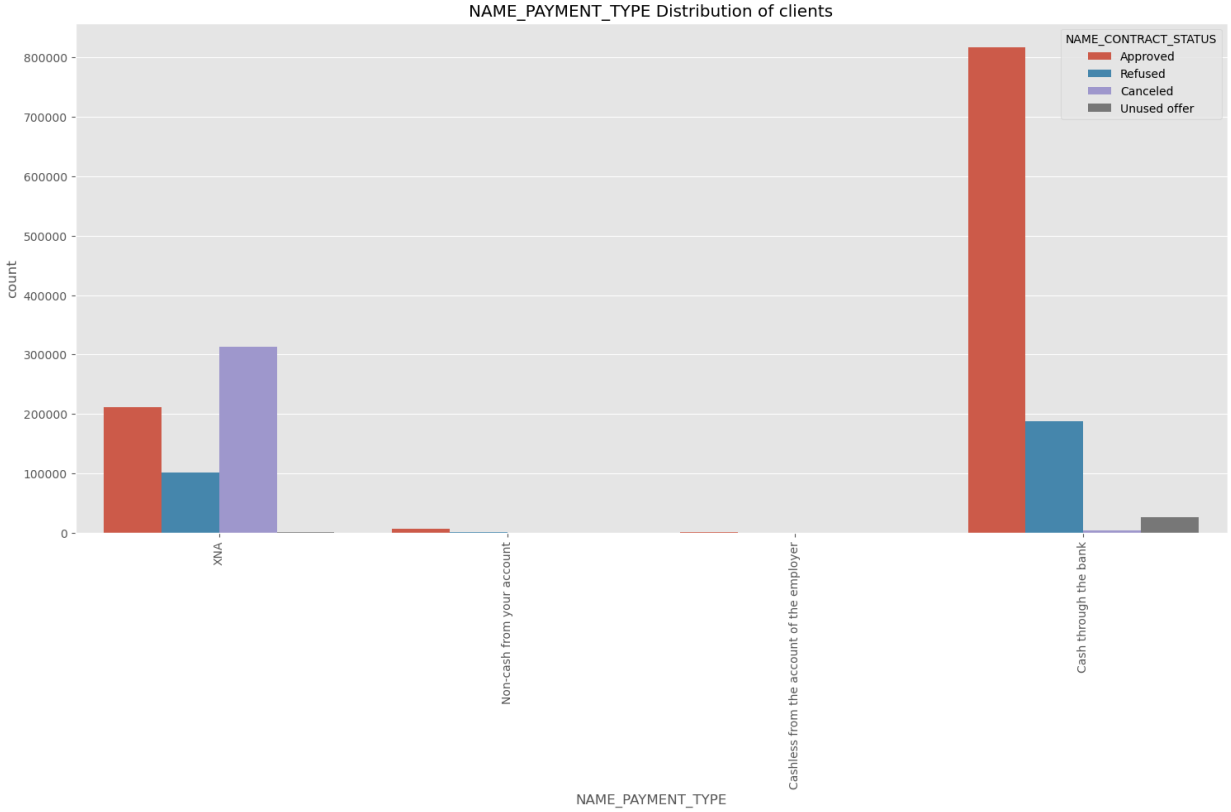
In [ ]:
```
pre_data.PRODUCT_COMBINATION.value_counts()
```

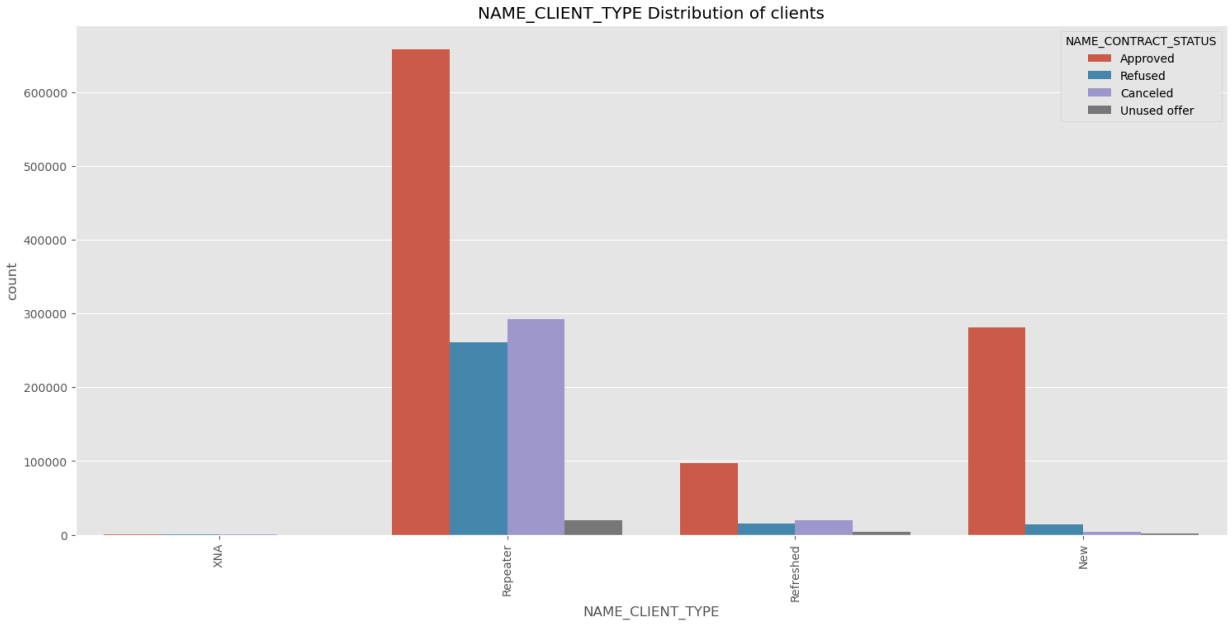
Out[ ]:
```
Cash                                285990
POS household with interest         263622
POS mobile with interest            220670
Cash X-Sell: middle                 143883
Cash X-Sell: low                    130248
Card Street                         112582
POS industry with interest    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

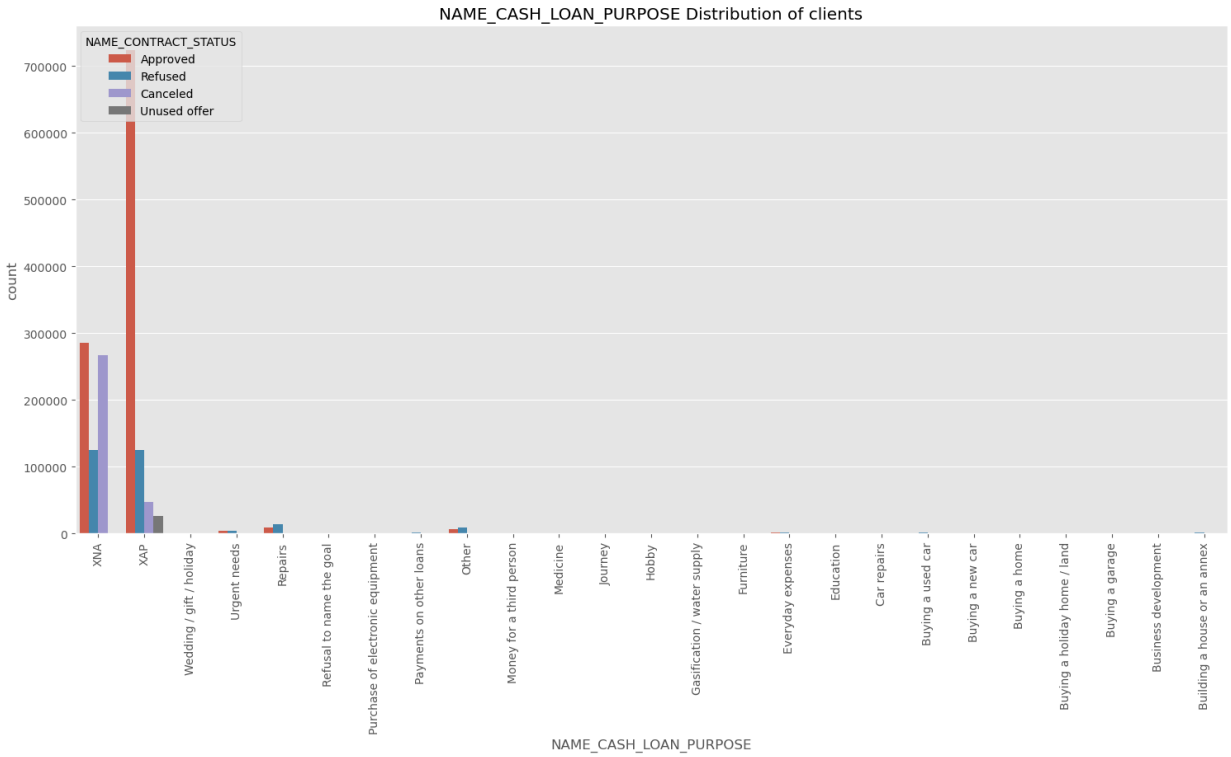
In [ ]:
```
pre_data.PRODUCT_COMBINATION.fillna(pre_data.PRODUCT_COMBINATION.mode()[0], inplace=Tr
```

In [ ]:
```
# Plotting a count plot on TARGET's object columns
c_plot=['NAME_CONTRACT_TYPE','NAME_PAYMENT_TYPE','NAME_CLIENT_TYPE','NAME_CASH_LOAN_PU
for i in c_plot:
    plt.style.use('ggplot')
    plt.figure(figsize = [18,8])
    plt.title(f'{i} Distribution of clients')
    sns.countplot(data=pre_data, x =i, hue='NAME_CONTRACT_STATUS', order = sorted(pre_
    plt.xticks(rotation = 90)
```



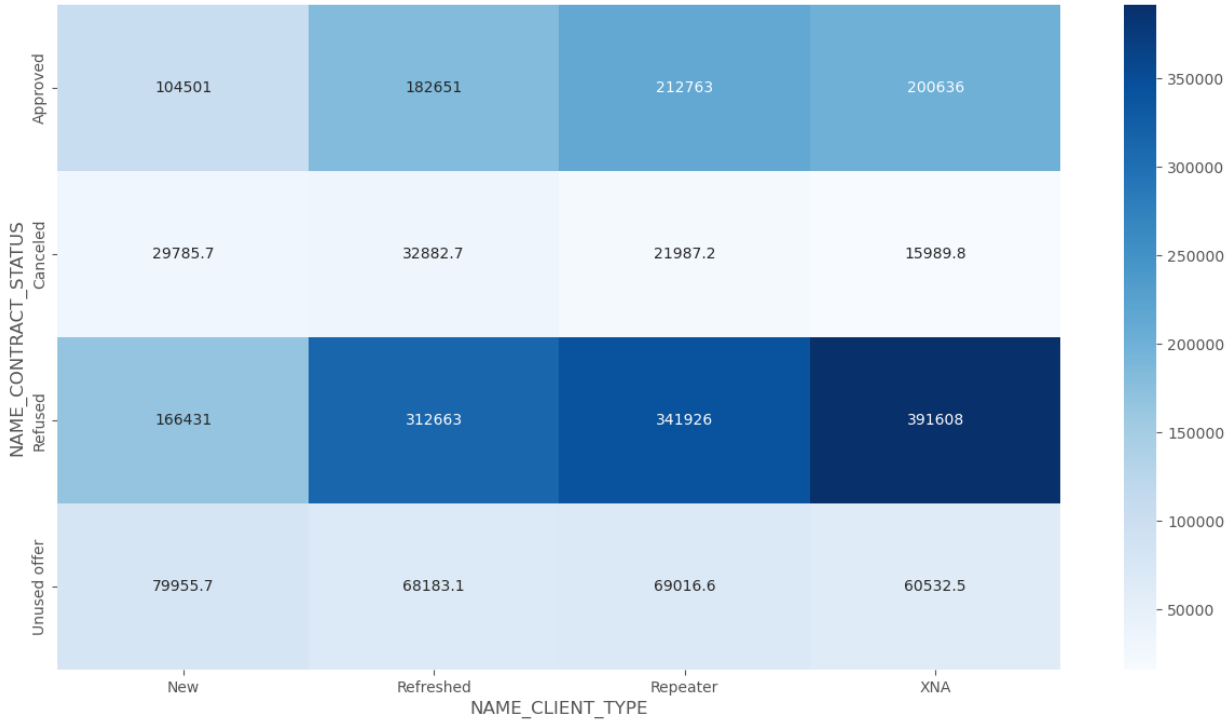






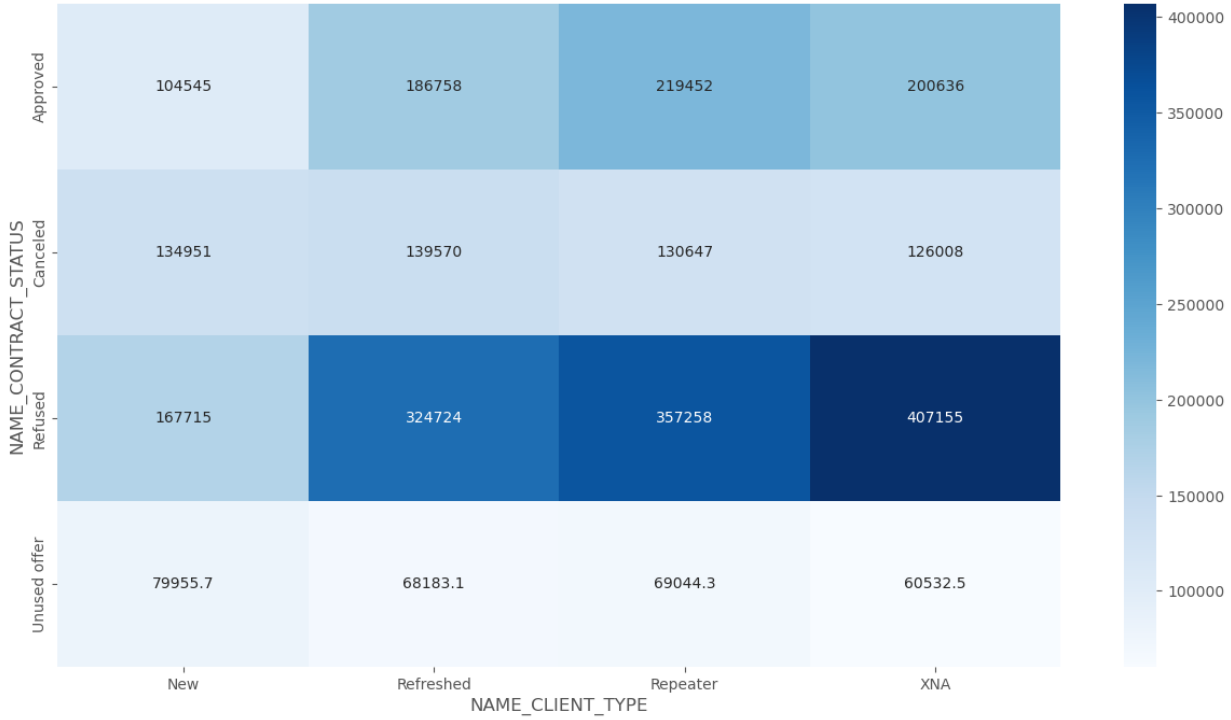
Observations: In approved category, consumer loan has largest no of applicants. There seem to be no cancelled loans in cash loan category than consumer loan. More cash loans have been refused than consumer loans. The bank has more repeaters in all approved, refused, unused, cancelled categories POS transactions seem to be consumer loans and as stated before more cash loans have been refused than POS.

In [ ]:
```
# Checking contract status vs name client type aggregating over application amount
res=pd.pivot_table(data=pre_data, index="NAME_CONTRACT_STATUS",columns='NAME_CLIENT_TY
plt.figure(figsize=(15,8))
sns.heatmap(res, annot=True,cmap='Blues', fmt="g")
plt.show()
```



Observations: Unused offer CREDIT AMOUNT is low. This may be the reason for customer not using Unable to understand why for cancelled and refused there should be any credit amount?

In [ ]:
```
# Checking contract status vs name client type aggregating over AMOUNT GOOD PRICE
res=pd.pivot_table(data=pre_data, index="NAME_CONTRACT_STATUS",columns='NAME_CLIENT_TY
plt.figure(figsize=(15,8))
sns.heatmap(res, annot=True,cmap='Blues', fmt="g")
plt.show()
```



Observations: All cancelled and refused cases have higher value of goods than other categories

Case Study Summary:

Chances of client havind payment difficulty All the below variables were established in analysis of Application dataframe as leading to default. Checked these against the Approved loans which have defaults, and it proves to be correct Medium income 25-35 years olds, followed by 35-45 years age group Male Unemployed Labourers, Salesman, Drivers Own House - No Other IMPORTANT Factors to be considered No of Bureau Hits in last week. Month etc – zero hits is good Amount income not correspondingly equivalent to Good Bought – Income 'Low' and 'High' is a concern Previous applications with Refused, Cancelled, Unused loans also have default which is a matter of concern. This indicates that the financial company had Refused/Cancelled previous application but has approved the current and is facing default on these. Credible Applications refused Unused applications have lower loan amount. Is this the reason for no usage? Female applicants should be given extra weightage as defaults are lesser. Students and Business mean have no problem in repayment of the loan Previous applications with Refused, Cancelled, Unused loans also have cases where payments are coming on time in current application. This indicates that possibly wrong decisions were done in those cases.

DATASET LINK AND PYTHON JUYPETER NOTBOOK LINK

application_data.csv https://drive.google.com/file/d/1BXczEWnzo8EMOKGPg3D7E0gWEEdZHUo1/view?usp=share_link columns_description.csv https://drive.google.com/file/d/1XnCBYXiGyYwK3G6Y_XDd0QVpZTu2znHR/view?usp=share_link previous_application.csv https://drive.google.com/file/d/1eZ4R-WzGyyimBXDTCrz2EGcnnlteFLyu/view?usp=share_link Py juypeter file :- BANKLOANCASESTUDY.ipynb https://drive.google.com/file/d/1YcLnEPK3JVr1Eu4t0jYQBHEh-bOkiS79/view?usp=share_link

In [ ]: