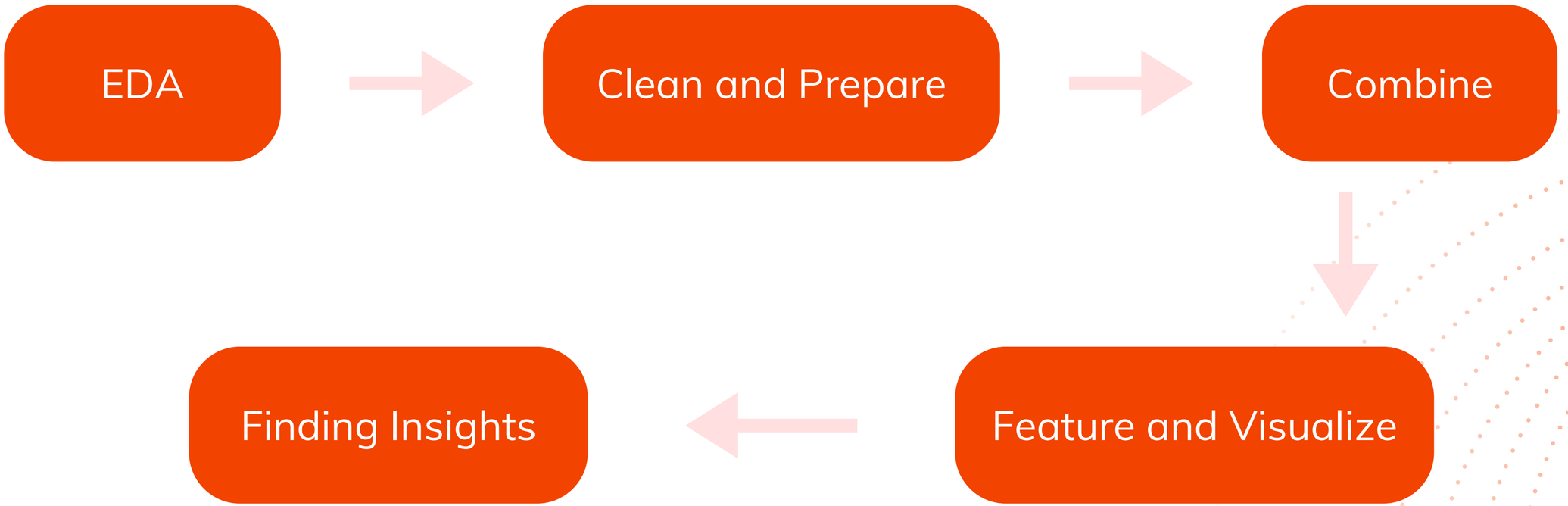
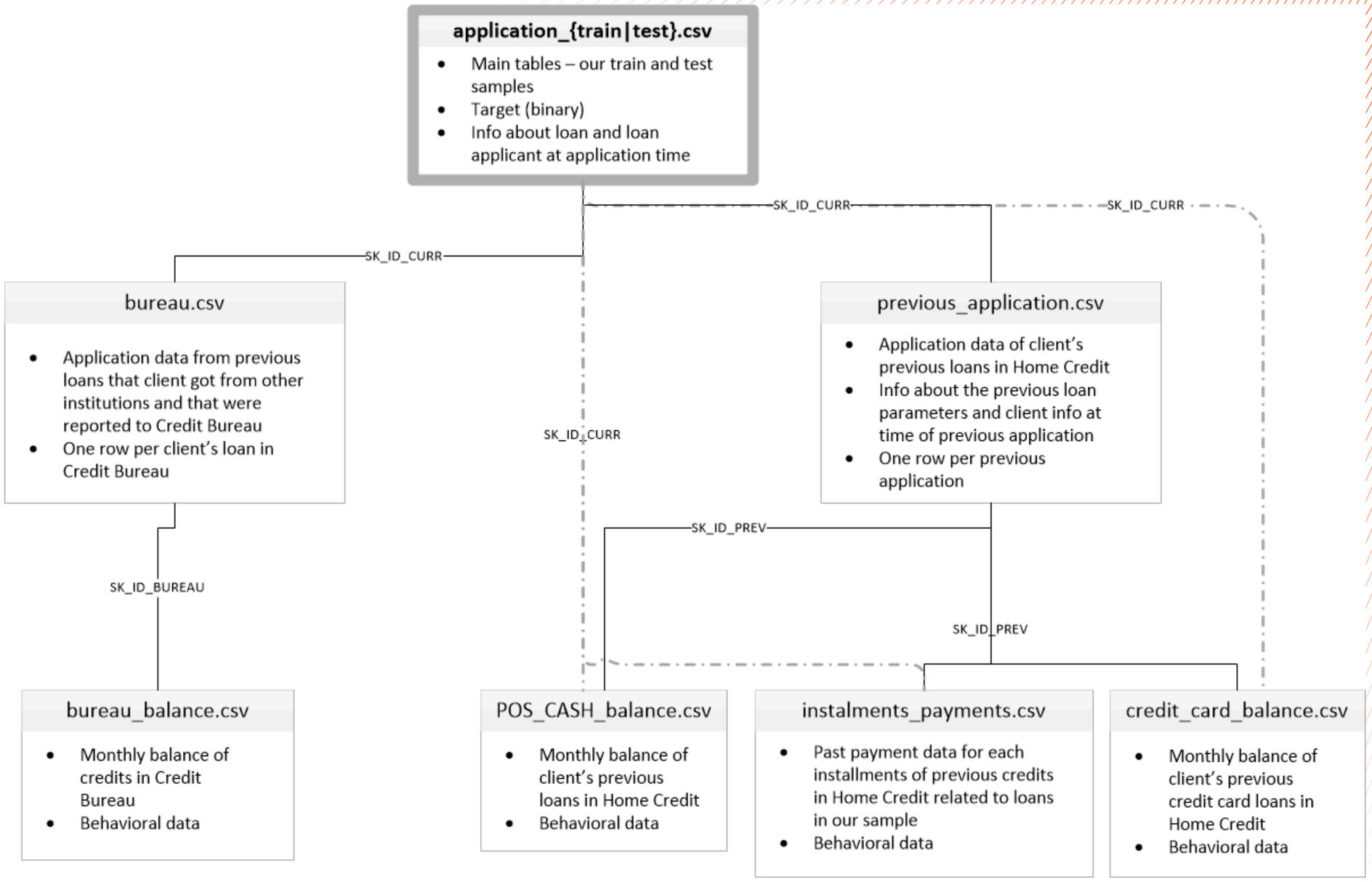
Home Credit Default Risk

Process


EDA
Clean and Prepare
Combine
Feature and Visualize
Finding Insights

Introduction of the Dataset



Step 2

Clean and Preparing

1. Dropping Columns
2. Handling Missing Values
3. Handling Outliers

```
list_drop = []
```

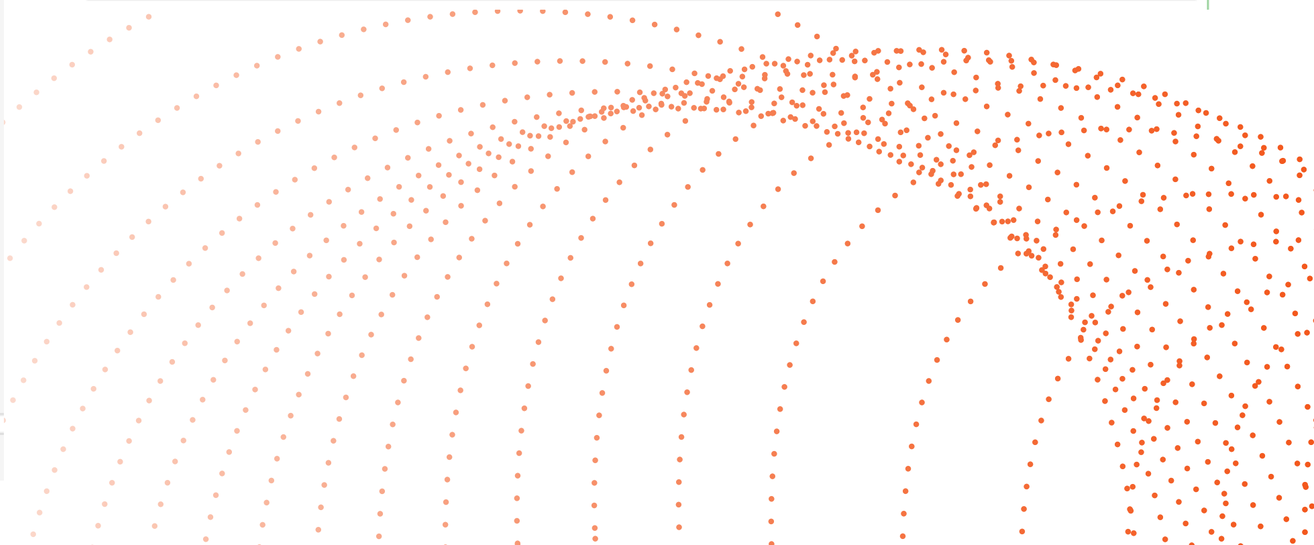
```
nunique_train = train.nunique().reset_index().rename(columns={
    'index': 'Name',
    0: 'NumberOfUnique'
})

nunique_test = test.nunique().reset_index().rename(columns={
    'index': 'Name',
    0: 'NumberOfUnique'
})

nunique_bureau = bureau.nunique().reset_index().rename(columns={
    'index': 'Name',
    0: 'NumberOfUnique'
})

nunique_bureau_balance = b_balance.nunique().reset_index().rename(
    columns={
        'index': 'Name',
        0: 'NumberOfUnique'
    })

nunique_prev = prev.nunique().reset_index().rename(columns={
    'index': 'Name',
    0: 'NumberOfUnique'
})

nunique_credit = credit.nunique().reset_index().rename(columns={
    'index': 'Name',
    0: 'NumberOfUnique'
})

nunique_installment = installment.nunique().reset_index().rename(
    columns={
        'index': 'Name',
        0: 'NumberOfUnique'
    })

nunique_pos = pos.nunique().reset_index().rename(columns={
    'index': 'Name',
    0: 'NumberOfUnique'
})

nunique_all = pd.concat([nunique_train, nunique_test, nunique_bureau,
                         nunique_bureau_balance, nunique_prev, nunique_credit,
                         nunique_installment, nunique_pos])
```

```
list_drop = nunique_all.loc[nunique_all['NumberOfUnique'] == 1, 'Name'].to_list()
```

1. Dropping Columns

Delete Columns That Contain Over 93% On a Single Value

```
#dropping columns with over 93% of a column has one value
l=combined_data.columns

for i in l:
    res=combined_data[i].value_counts(normalize=True)*100
    if res.iloc[0]>=93:
        new_drop_list.append(i)

#delete duplicate in list
new_drop_list = list(dict.fromkeys(new_drop_list))
print(new_drop_list)
```



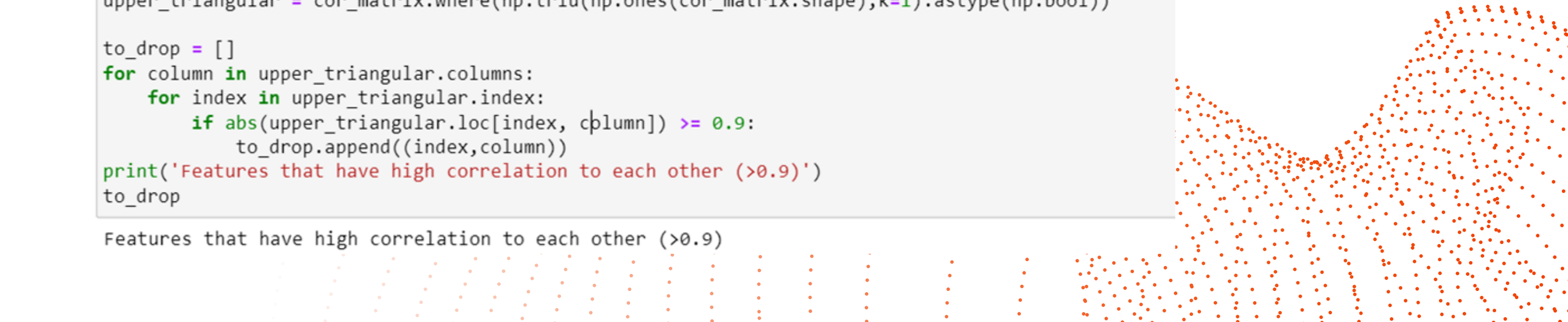
Delete Rows that Contain Duplicate Data

```
train.drop_duplicates(inplace=True)
test.drop_duplicates(inplace=True)
bureau.drop_duplicates(inplace=True)
b_balance.drop_duplicates(inplace=True)
prev.drop_duplicates(inplace=True)
credit.drop_duplicates(inplace=True)
pos.drop_duplicates(inplace=True)
installment.drop_duplicates(inplace=True)
```

Hand picking from column pairs with over 0.9 correlation

```
cor_matrix = train.corr()
upper_triangular = cor_matrix.where(np.triu(np.ones(cor_matrix.shape),k=1).astype(np.bool))

to_drop = []
for column in upper_triangular.columns:
    for index in upper_triangular.index:
        if abs(upper_triangular.loc[index, column]) >= 0.9:
            to_drop.append((index,column))
print('Features that have high correlation to each other (>0.9)')
to_drop
```

```
Features that have high correlation to each other (>0.9)
```



Optimize the data types for each dataset

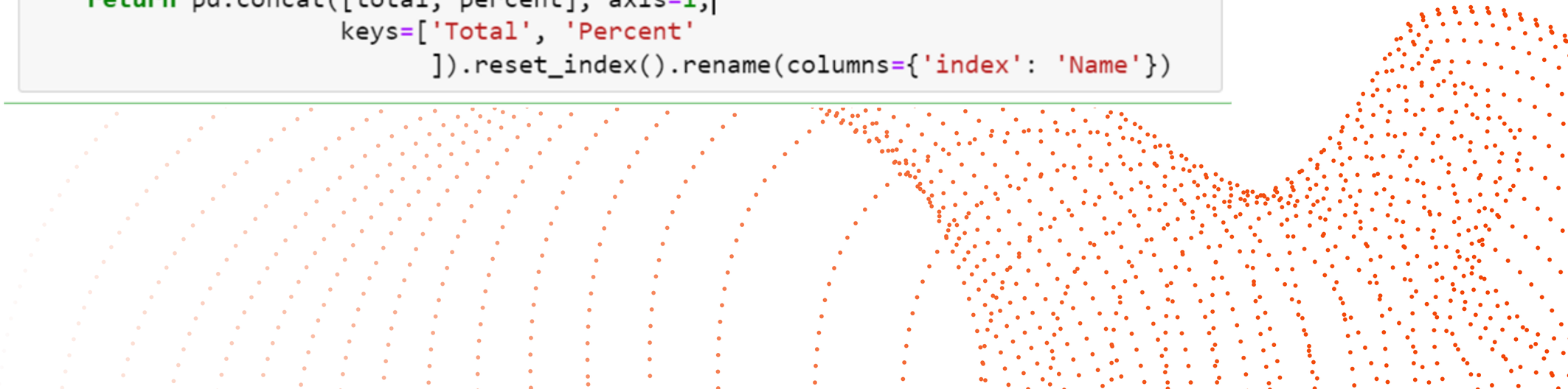
```python
# Function to maximize the data type of dataset
# np.iinfo() -> Machine limits for integer types
# Example: Machine Limit for int16 is min = -32768 | max = 32767
# np.finfo() -> Machine limits for float types
# get max and min value of column then compare with machine limit and decide best data type
def best_dtype(data):
    for col in data.columns:
        col_type = data[col].dtype

        if col_type != object:
            col_min = data[col].min()
            col_max = data[col].max()
            if str(col_type)[:3] == 'int':
                if col_min > np.iinfo(np.int8).min and col_max < np.iinfo(
                        np.int8).max:  #check data type compatibility
                    data[col] = data[col].astype(
                        np.int8)  # change data type per condition
                elif col_min > np.iinfo(np.int16).min and col_max < np.iinfo(
                        np.int16).max:
                    data[col] = data[col].astype(np.int16)
                elif col_min > np.iinfo(np.int32).min and col_max < np.iinfo(
                        np.int32).max:
                    data[col] = data[col].astype(np.int32)
                elif col_min > np.iinfo(np.int64).min and col_max < np.iinfo(
                        np.int64).max:
                    data[col] = data[col].astype(np.int64)
            else:
                if col_min > np.finfo(np.float16).min and col_max < np.finfo(
                        np.float16).max:
                    data[col] = data[col].astype(np.float16)
                elif col_min > np.finfo(np.float32).min and col_max < np.finfo(
                        np.float32).max:
                    data[col] = data[col].astype(np.float32)
                else:
                    data[col] = data[col].astype(np.float64)
        else:
            data[col] = data[col].astype('category')
    return 'Maximize dataset complete'
```

Drop columns with over 80% missing value

```
#let's drop all columns with over 80% missing value
combined_data_missing_val = missing_data(combined_data)
```

```
new_drop_list = combined_data_missing_val.loc[
    combined_data_missing_val['Percent'] >= 80, 'Name'].to_list()
```

```
def missing_data(data):
    total = data.isnull().sum().sort_values(ascending=False)
    percent = (data.isnull().sum() / data.isnull().count() *
               100).sort_values(ascending=False)
    return pd.concat([total, percent], axis=1,
                     keys=['Total', 'Percent'
                           ]).reset_index().rename(columns={'index': 'Name'})
```



2. Handling Missing Values

Numeric Columns

Discrete Data

```
numeric = ['Int8', 'int8', 'int16', 'int32', 'int64',
           'float16', 'float32', 'float64']
discrete = ['int8', 'int16', 'int32']
continuous = ['float16', 'float32', 'float64']
```

```
#Get the discrete data
discrete_df = combined_data.select_dtypes(include = discrete)
discrete_df.shape
```

→ Discrete data variables don't have missing data

Numeric Columns

Continuous Data

Fill missing value:

- Variables with over 25 number of unique will be filled with its mode
- The rest will be filled with mean

Using:

- Imputer estimator

--> replace missing values using mean, median, mode ... (descriptive statistic) values for the corresponding columns

Fill missing value:

- Variables with over 25 number of unique except will be filled with its mode

```
mode_imputer.fit(col_num_unique_less_25)
```

```
SimpleImputer(strategy='most_frequent')
```

```
imputed_mode = mode_imputer.transform(col_num_unique_less_25)
```

```
col_num_unique_less_25[col_num_unique_less_25.isnull()] = imputed_mode
```

```
missing_data(col_num_unique_less_25)
```

```
# Fill in the main dataset
combined_data[list_unique_less25] = col_num_unique_less_25
```

```
missing_data(combined_data[list_unique_less25])
```

Output:

	Name	Total	Percent
0	CNT_FAM_MEMBERS	0	0.0
1	DEF_30_CNT_SOCIAL_CIRCLE	0	0.0
2	DEF_60_CNT_SOCIAL_CIRCLE	0	0.0
3	AMT_REQ_CREDIT_BUREAU_MON	0	0.0
4	AMT_REQ_CREDIT_BUREAU_QRT	0	0.0
5	AMT_REQ_CREDIT_BUREAU_YEAR	0	0.0

Fill missing value:

- The rest will be filled with mean

```
list_unique_over25 = num_unique_df.loc[num_unique_df['NumberOfUnique'] >= 25,
                                       'Name'].to_list()
list_unique_over25.remove('AMT_REQ_CREDIT_BUREAU_YEAR')
col_unique_over25 = continuous_df[list_unique_over25]
```

```
col_unique_over25.head()
```

```
mean_imputer.fit(col_unique_over25)
```

```
SimpleImputer()
```

```
imputed_mean = mean_imputer.transform(col_unique_over25)
```

```
col_unique_over25[col_unique_over25.isnull()] = imputed_mean
```

```
missing_data(col_unique_over25)
```

: Output:

	Name	Total	Percent
0	AMT_INCOME_TOTAL	0	0.0
1	AMT_CREDIT_x	0	0.0
2	AMT_CREDIT_y	0	0.0
3	AMT_DOWN_PAYMENT	0	0.0
4	AMT_GOODS_PRICE_y	0	0.0
...	...	...	...
59	AMT_CREDIT_SUM_DEBT	0	0.0
60	AMT_CREDIT_SUM_LIMIT	0	0.0
61	DAYS_CREDIT_UPDATE	0	0.0
62	AMT_ANNUITY_y	0	0.0
63	AMT_PAYMENT	0	0.0

Categorical Columns

We will simply **fill null** with "**Unknown**"

```
# We will simply fill null with "Unknown"
# Let's add "Unknown" category to these columns
list_cat = cat_mising_df[cat_mising_df['Percent'] > 0].Name.to_list()
for i in list_cat:
    combined_data[i].cat.add_categories('Unknown', inplace = True)
```

EXT_SOURCE_1, EXT_SOURCE_2, EXT_SOURCE_3, are very important features so we will not do anything to it at this moment.

TARGET has missing value because application_test don't have that column

→ Output:

```
missing_data(combined_data)
```

	Name	Total	Percent
0	EXT_SOURCE_1	193910	54.430113
1	EXT_SOURCE_3	69633	19.545831
2	TARGET	48744	13.682334
3	EXT_SOURCE_2	668	0.187506
4	SK_ID_CURR	0	0.000000
...	...	...	...
100	LIVE_CITY_NOT_WORK_CITY	0	0.000000
101	REG_CITY_NOT_WORK_CITY	0	0.000000
102	REG_CITY_NOT_LIVE_CITY	0	0.000000
103	HOUR_APPR_PROCESS_START_x	0	0.000000
104	AMT_PAYMENT	0	0.000000

3. Handling Outliers

- sort the dataset in ascending order
- calculate the 1st and 3rd quartiles (Q1, Q3)
- compute IQR = Q3 - Q1
- compute lower bound = (Q1–1.5*IQR), upper bound = (Q3+1.5*IQR)
- below the lower bound and above the upper bound --> mark them as outliers

```
: def outlier_detector(df, col):
    q1_col = Q1[col]
    iqr_col = IQR[col]
    q3_col = Q3[col]
    return df[((df[col] <
                (q1_col - 1.5 * iqr_col)) | (df[col] >
                                             (q3_col + 1.5 * iqr_col)))]


def lower_outlier(df, col):
    q1_col = Q1[col]
    iqr_col = IQR[col]
    q3_col = Q3[col]
    lower = df[(df[col] < (q1_col - 1.5 * iqr_col))]
    return lower


def upper_outlier(df, col):
    q1_col = Q1[col]
    iqr_col = IQR[col]
    q3_col = Q3[col]
    upper = df[(df[col] > (q3_col + 1.5 * iqr_col))]
    return upper
```

```
Q1 = combined_data.quantile(0.25)
Q3 = combined_data.quantile(0.75)
IQR = Q3 - Q1
```

Replaced the out-of-bounds data with the value of Q3 + (1.5 * IQR) and those before the lower limit position in the chart with the value of Q1 – (1.5 * IQR)

```
def replace_lower(df, col):
    q1_col = Q1[col]
    iqr_col = IQR[col]
    q3_col = Q3[col]
    tmp = 1111111
    lower = q1_col - 1.5 * iqr_col
    df[col] = df[col].where(lambda x: (x > (lower)), tmp)
    df[col] = df[col].replace(tmp, lower)


def replace_upper(df, col):
    q1_col = Q1[col]
    iqr_col = IQR[col]
    q3_col = Q3[col]
    tmp = 9999999
    upper = q3_col + 1.5 * iqr_col
    df[col] = df[col].where(lambda x: (x < (upper)), tmp)
    df[col] = df[col].replace(tmp, upper)
```

Step 3

Combine

a

Handling Train/Test

b

Handling bureau and bureau_balance

c

Handling previous_application credit_card_balance installments_payments POS_CASH_balance

a Handling Train/Test

```
# easily combining due to the similarity in each dataset format with the
# exception of TARGET column
combined_data = train.append(test)
```

```
print(train.shape)
print(test.shape)
```

```
(307511, 77)
(48744, 122)
```

```
#checking the combination
print(train.shape[0] + test.shape[0])
print(combined_data.shape)# equals number of rows => the combination is correct
```

```
356255
(356255, 123)
```

b Handling bureau and bureau_balance

Issue:

1. bureau dataset gives us the application of previous loans that client got from other institution reported to Credit Bureau
2. bureau_balance dataset illustrates the monthly balance of credits in bureau dataset (Behaviral data)
3. bureau matches SK_ID_CURR columns with train/test
4. train/test has one row per customer
5. bureau has multiple row per customer because most customer apply for different loans
6. bureau_balance contains rows for each month of credit reported tobureau_balance
7. bureau_balance matches SK_ID_BUREAU with bureau

b Handling bureau and bureau_balance

Get mean values of columns grouped by SK_ID_BUREAU to minimize bureau_balance

```
# Function to group the ID and get the mean value
def get_mean(data, col_name):
    p = data.groupby(col_name, as_index = False).mean()
    return p
```

```
mean_balance = get_mean(b_balance, 'SK_ID_BUREAU')
mean_balance.head(10)
```

Merge it with bureau

```
mean_bureau = get_mean(
    bureau.merge(mean_balance, on='SK_ID_BUREAU', how='left'), 'SK_ID_CURR')
```

Do the same with bureau but grouped by SK_ID_CURR

```
#Merge the mean_bureau to combine_data (main dataset)
combined_data = combined_data.merge(mean_bureau,
                        on = 'SK_ID_CURR',
                        how = 'left')
pd.concat([combined_data.head(), combined_data.tail()])
```

Merge it with train/test

```
# each row display each client => combine with train/test
# let's merge it by SK_ID_CURR
combined_data = combined_data.merge(prev_loan_count,
                                    on = 'SK_ID_CURR', how = 'left')
```

We have **added 14 new usable columns** from bureau and bureau_balance dataste after the train/test dataset

C Handling previous_application/ credit_card_balance/ installments_payments/ POS_CASH_balance

Issue:

1. previous_application shows data of client's previous loans from Home Credit (one row per previous application)
2. POS_CASH_balance shows the monthly balance of client's previous loans in Home Credit (Behavioral data) (one row for each month)
3. installments_payments shows repayment history for previous credits with Home Credit (Behavioral data) (one row for each payment)
4. credit_card_balance monthly balance of client's previous credit card loans with Home Credit (Behavioral data) (one row for each month)
5. All dataset matches the SK_ID_CURR with combined_data
6. previous_application have SK_ID_PREV as its own column matching with the other 3 datasets.

C Handling previous_application/ credit_card_balance/ installments_payments/ POS_CASH_balance

Get mean values of columns grouped by SK_ID_PREV to minimize the three subsidiary datasets (credit, installment, pos)

```
#get the total of each client previous applications reported to Home Credit
prev_app_count = prev.groupby(
    'SK_ID_CURR', as_index=False)['SK_ID_PREV'].count().rename(
        columns={'SK_ID_PREV': 'PREVIOUS_APPLICATION_COUNT'})
pd.concat([prev_app_count.head(), prev_app_count.tail()])
```

```
mean_credit = get_mean(credit, 'SK_ID_PREV')
mean_credit.head(10)
```

```
mean_pos = get_mean(pos, 'SK_ID_PREV')
mean_pos.head(10)
```

```
mean_installment = get_mean(installment, 'SK_ID_PREV')
mean_installment.head(10)
```

Merge the 3 dataset with the main dataset (prev)

```
# Merge the subsidiary dataset with main dataset
prev = prev.merge(mean_pos, on = 'SK_ID_PREV', how = 'left')
prev = prev.merge(mean_credit, on = 'SK_ID_PREV', how = 'left')
prev = prev.merge(mean_installment, on = 'SK_ID_PREV', how = 'left')
pd.concat([prev.head(), prev.tail()])
```

From the merged prev dataset, minimize it by getting mean values

```
# Minimize previous_application dataset
mean_prev = get_mean(prev, 'SK_ID_CURR')
pd.concat([mean_prev.head(), mean_prev.tail()])
```

Merge the minimized prev dataset to the combined dataset

```
# Merge it with combined combined_dataset
combined_data = combined_data.merge(mean_prev, on = 'SK_ID_CURR', how = 'left')
```

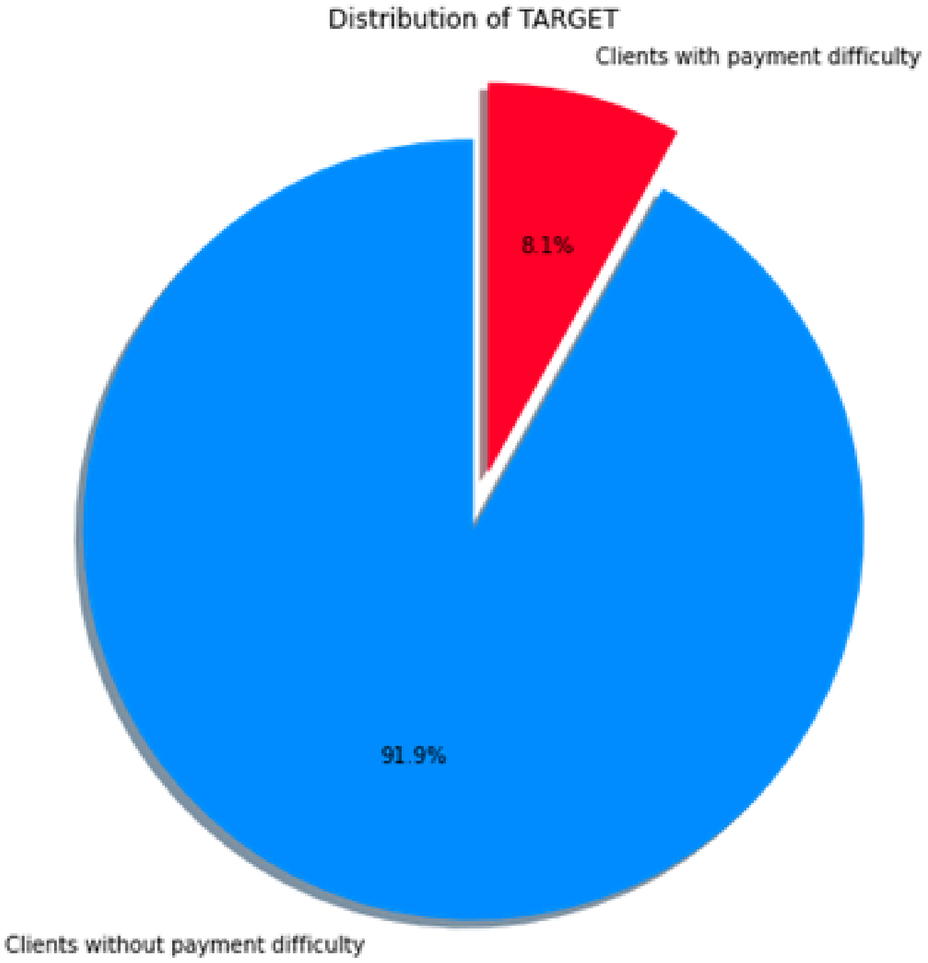
→

We have **added 52 new usable columns** from previous_application, credit_card_balance, POS_CASH_balance, installments_payments dataset for the final combinated dataset of Home Credit Default Risk dataset

Step 4

Feature and Visualize

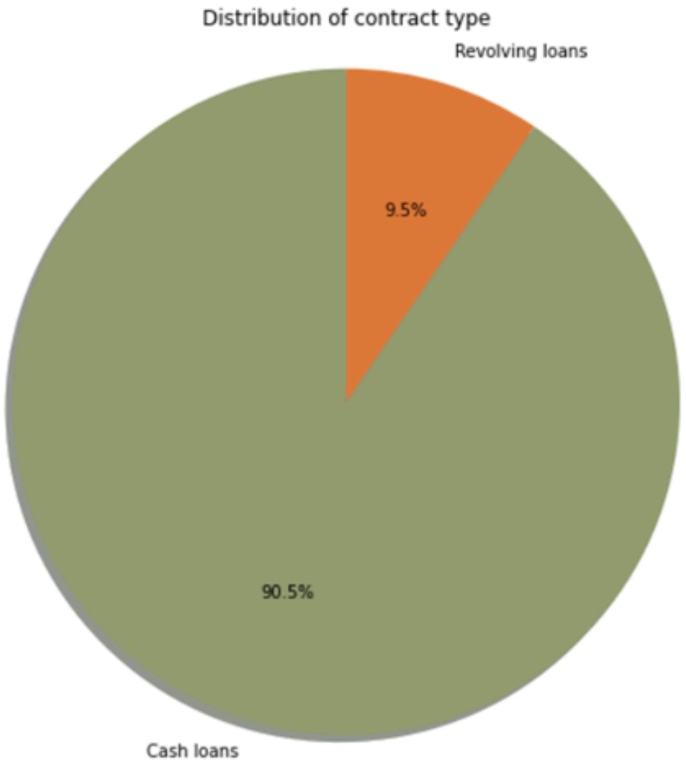
Distribution of TARGET

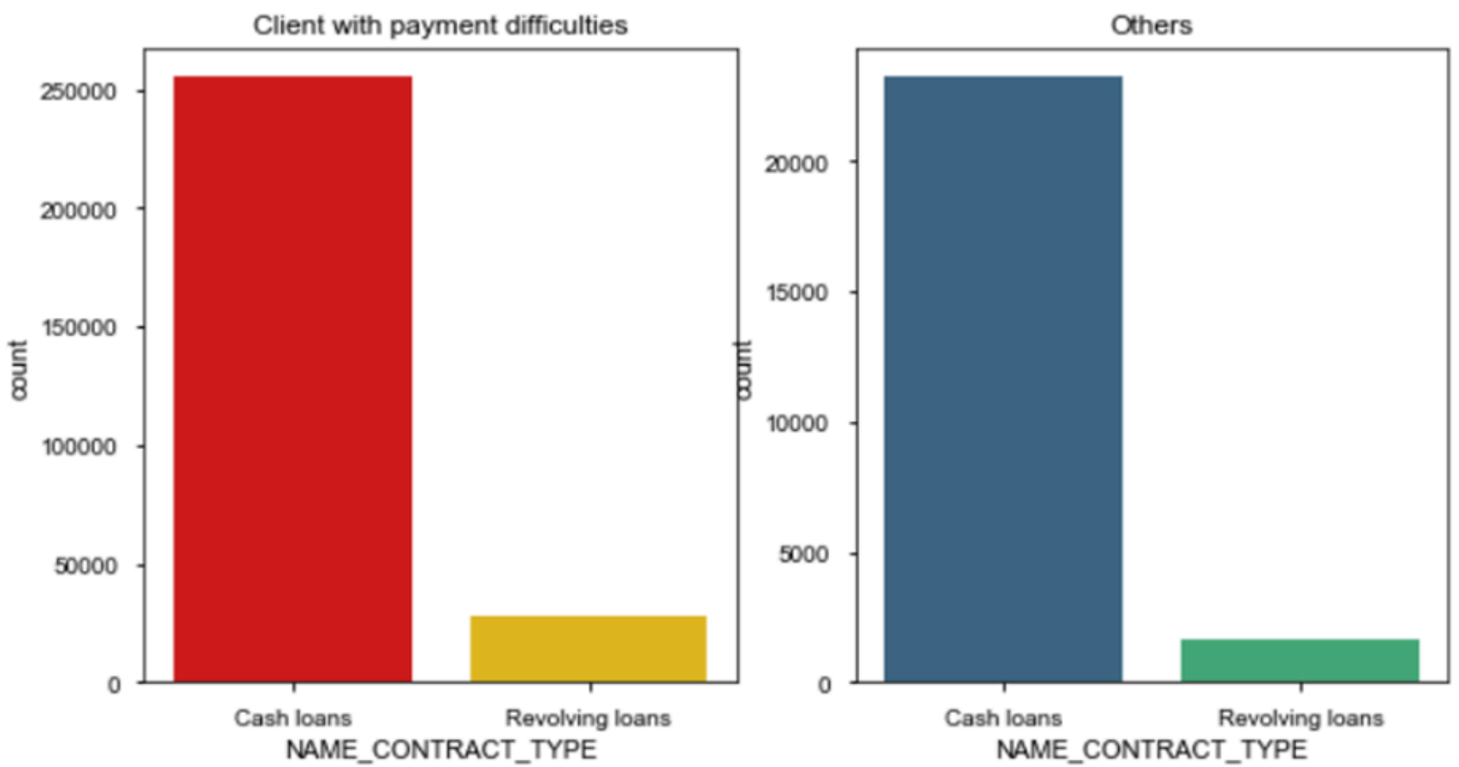


Target feature shows target = 0 is individual who **paid their loan on time** and target = 1 indicating the client had **payment difficulties.**

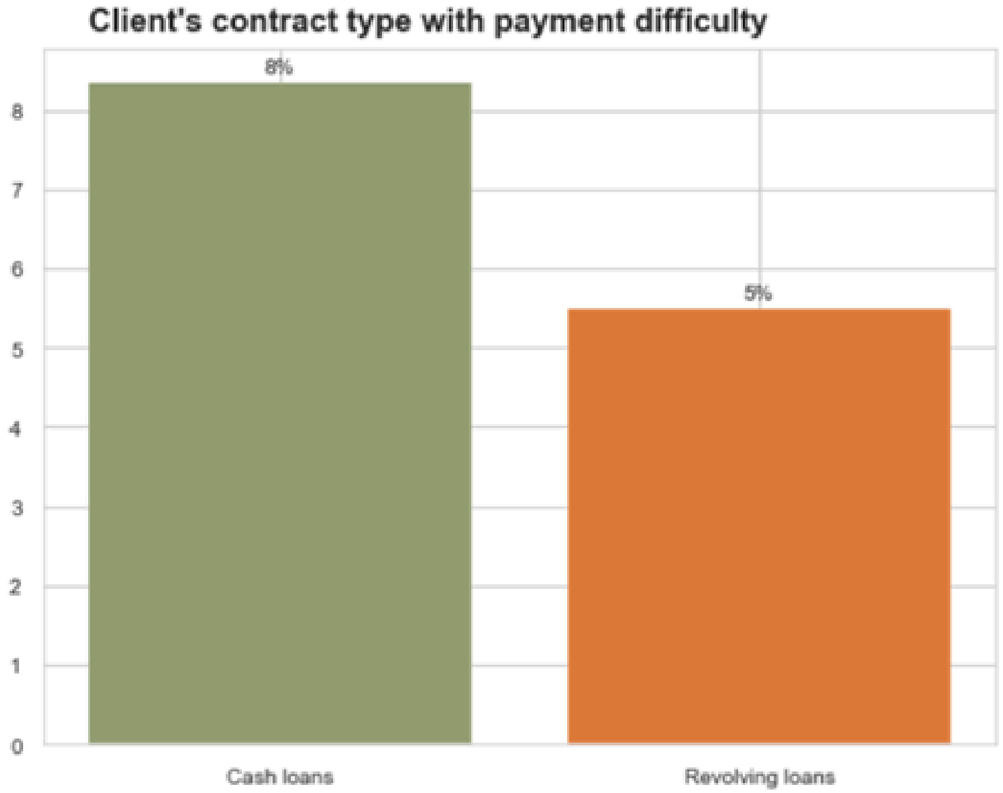
8.07% of the clients have difficulties while the proportion of the clients who are non-default is much higher, approximately 92%. Since the percentage difference is high and not evenly distributed along the target variable --> this is *an imbalance dataset*.

NAME_CONTRACT_TYPE: Identification if the loan is cash or revolving





NAME_CONTRACT_TYPE: Identification if the loan is cash or revolving

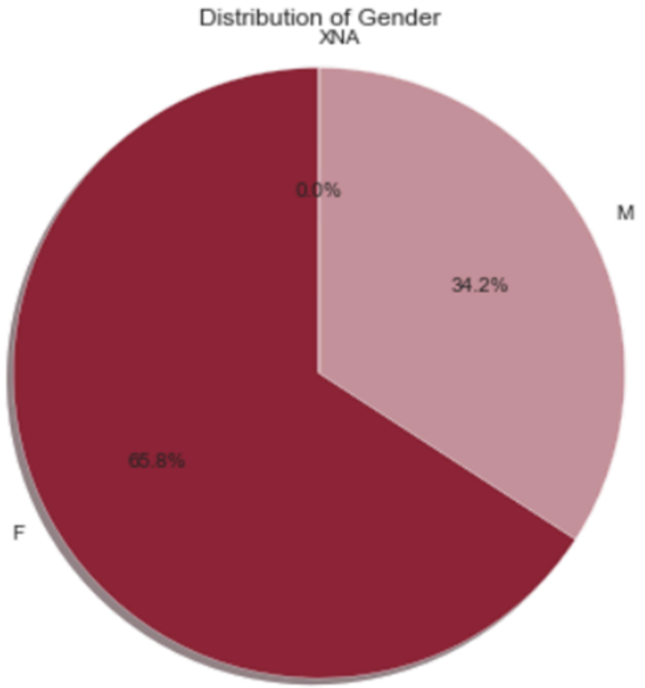


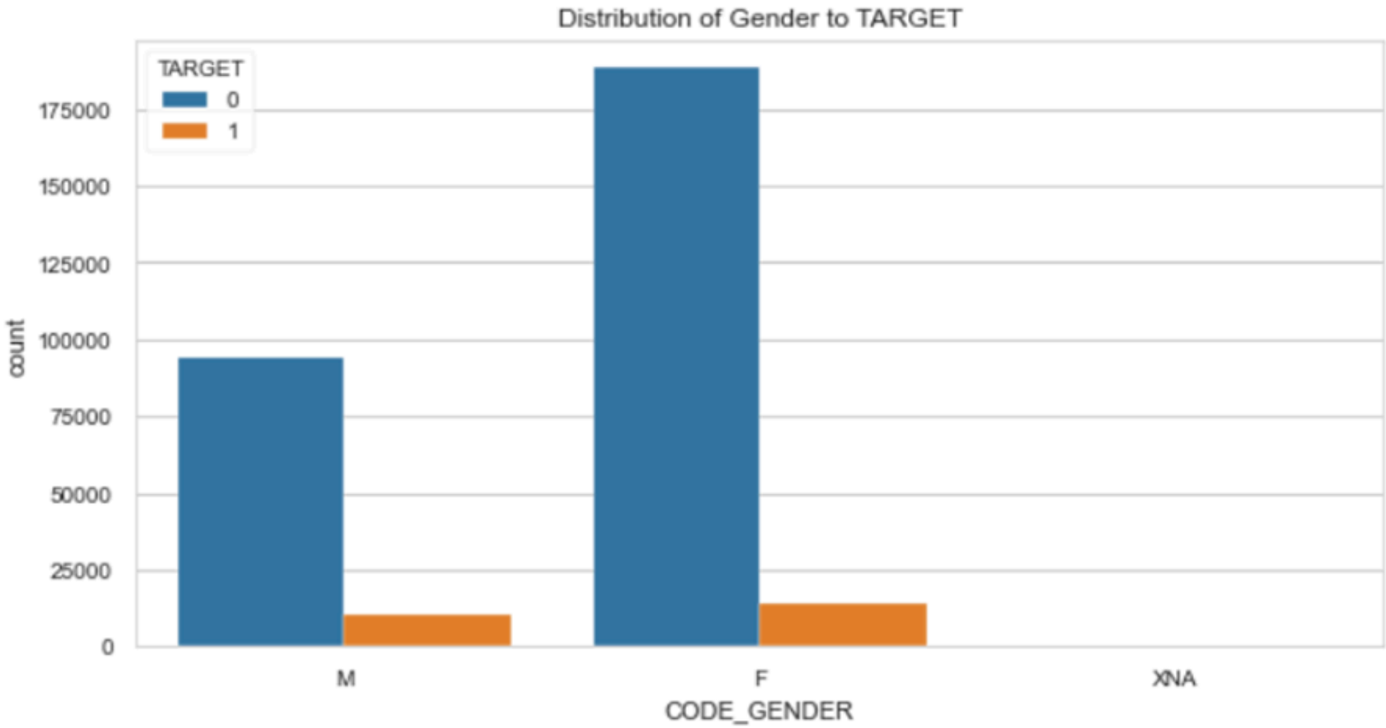
Many people would like to take cash loans instead of revolving loans as **the fraction of** cash loans **is** 10 times **higher than** revolving loans (90.5% compared to 9,5%). This happens both in clients with and without payment difficulties, also in the same ratio 1/10.

However in the number of clients who have difficulties in payment, the difference percentage between cash loans and revolving loans is less, only 3% apart.

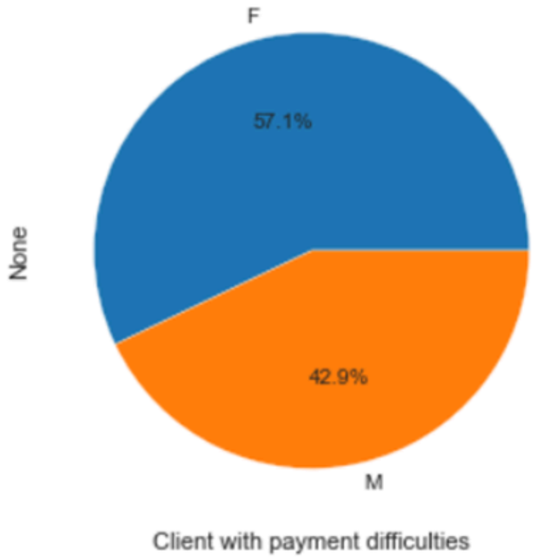
--> Customers who choose cash loans have more difficulty in repaying their loans on time than customers who choose revolving loans.

CODE_GENDER: Gender of the client

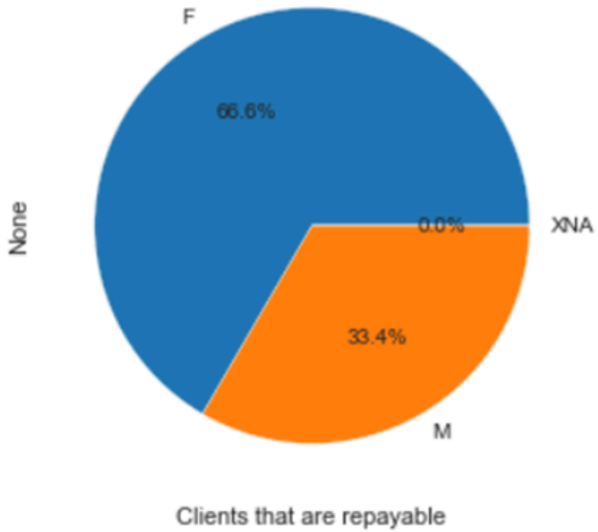




CODE_GENDER: Gender of the client



Client with payment difficulties



Clients that are repayable

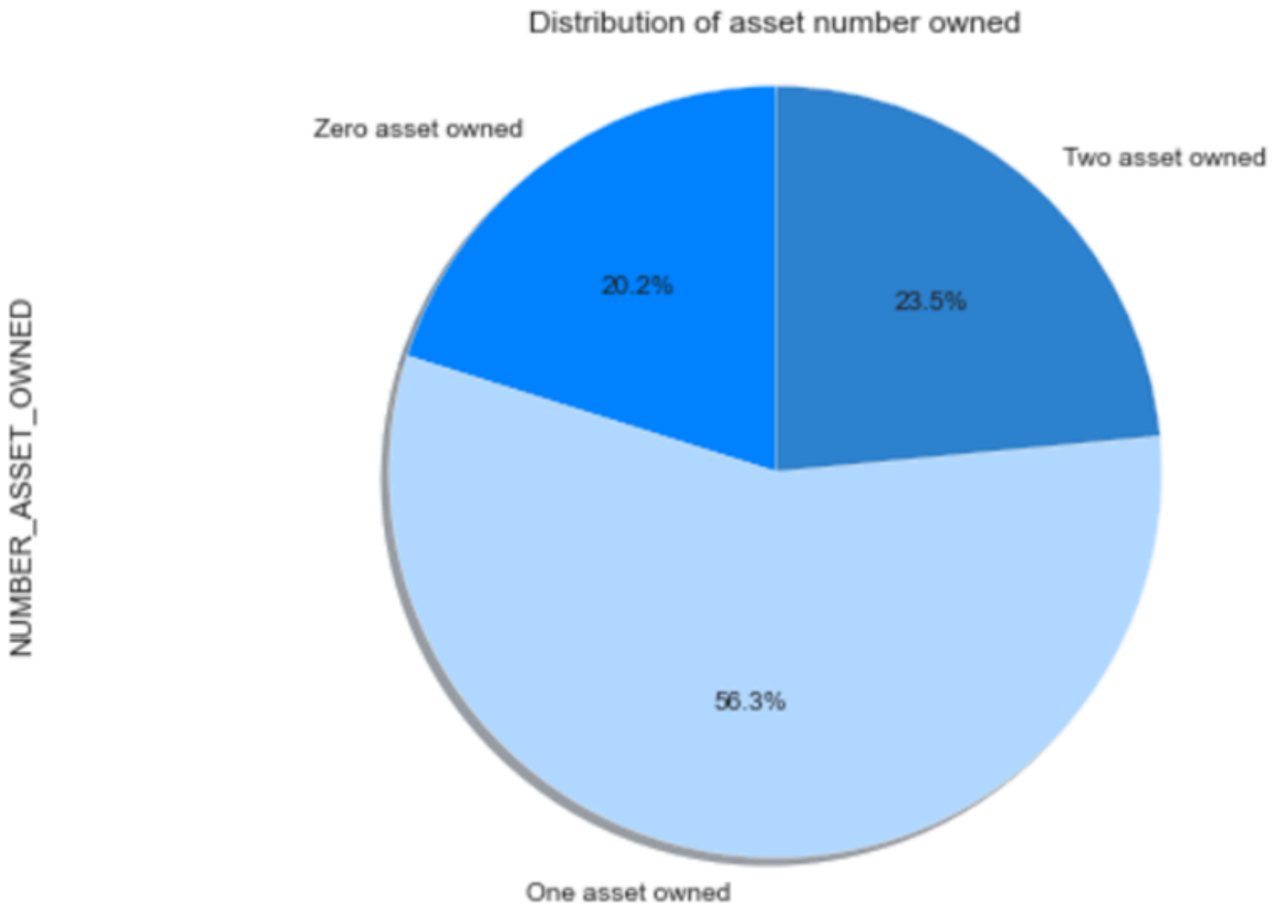
CODE_GENDER: Gender of the client

XNA might be people who ***do not want to be identified*** as Male or Female, which is understandable due to the current state of society where there are more than two genders.

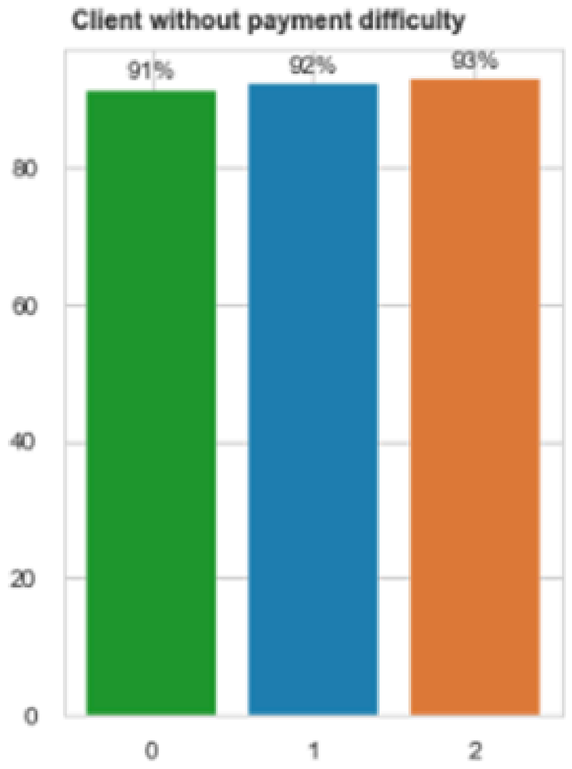
The number of female **borrowed** money is nearly twice as much as male at 202.448 and 105.059 applications filled respectively. Despite the higher amount of female borrower, the proportion of clients who have **difficulties in repayment** of male is 10%, while female is 7%

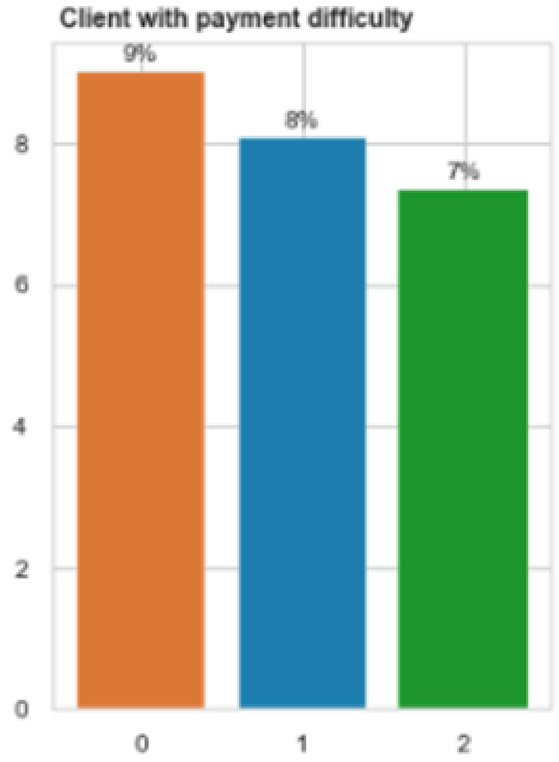
--> Women have a higher ability to pay loans on time than men.

FLAG_OWN_CAR: Flag if the client owns a car
FLAG_OWN_REALTY: Flag if client owns a house or flat



FLAG_OWN_CAR: Flag if the client owns a car
FLAG_OWN_REALTY: Flag if client owns a house or flat





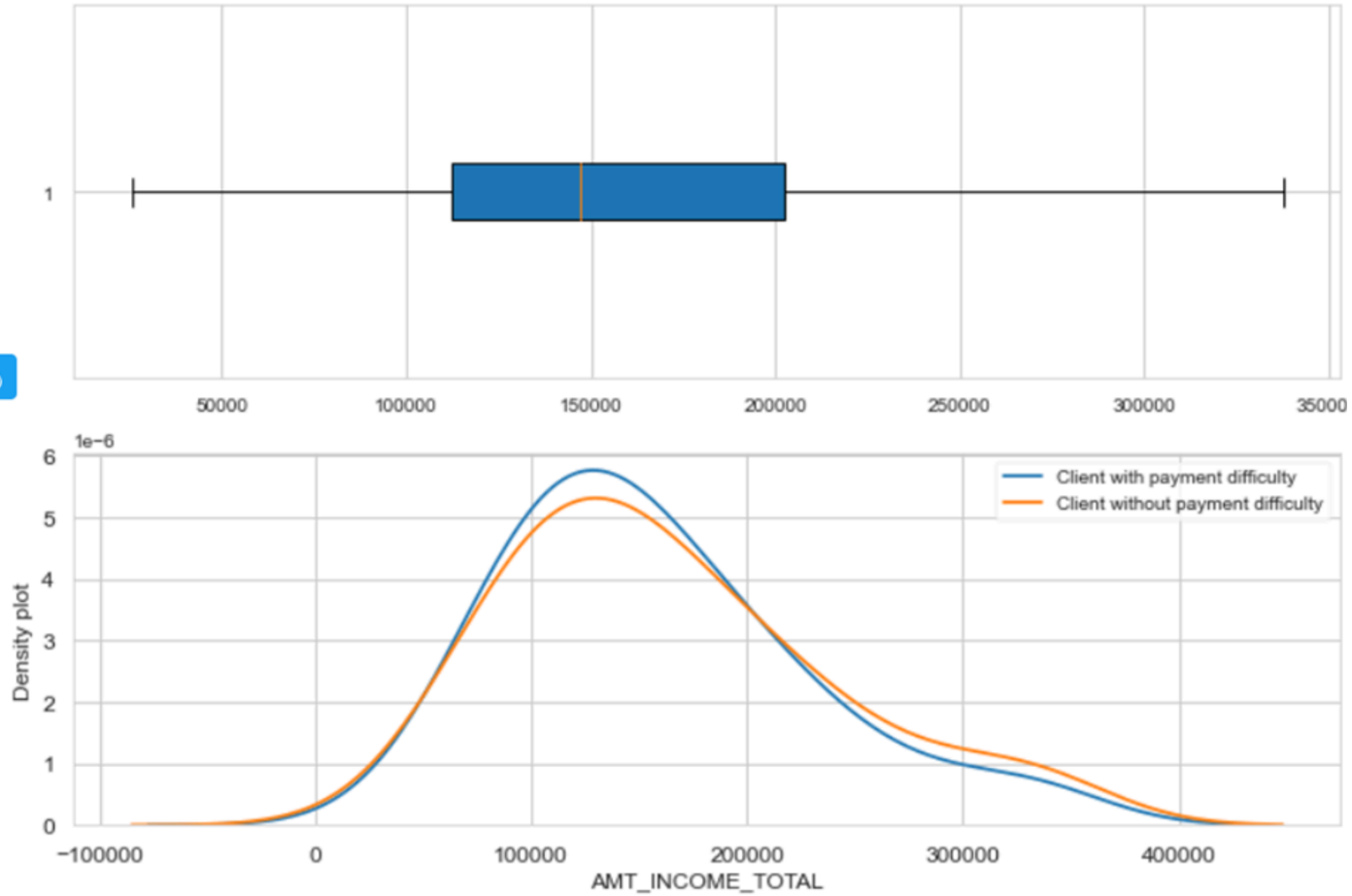
FLAG_OWN_CAR: Flag if the client owns a car
FLAG_OWN_REALTY: Flag if client owns a house or flat

More than 50% of the clients have only one asset: car or house (or flat). 23.5% of the clients have both assets car and house while approximately 20% have zero assets owned.

There is not much difference in the percentage of clients who do not have difficulty paying their loans between the three groups of clients' assets. The ratios are only 1% apart with the highest ratio is clients have two assets with 93%. In graph clients with payment difficulty, the ratio is more clearly differentiated between the three groups. Clients who do not have any assets **have more difficulties than others.**

Due to those who do not have a home or a car, which means they have less money and a lower source of income than those in groups 1 and 2. Since their lower income and the need to pay for other things like renting a house while other groups not, they tend to have more difficulties than those who already have assets.

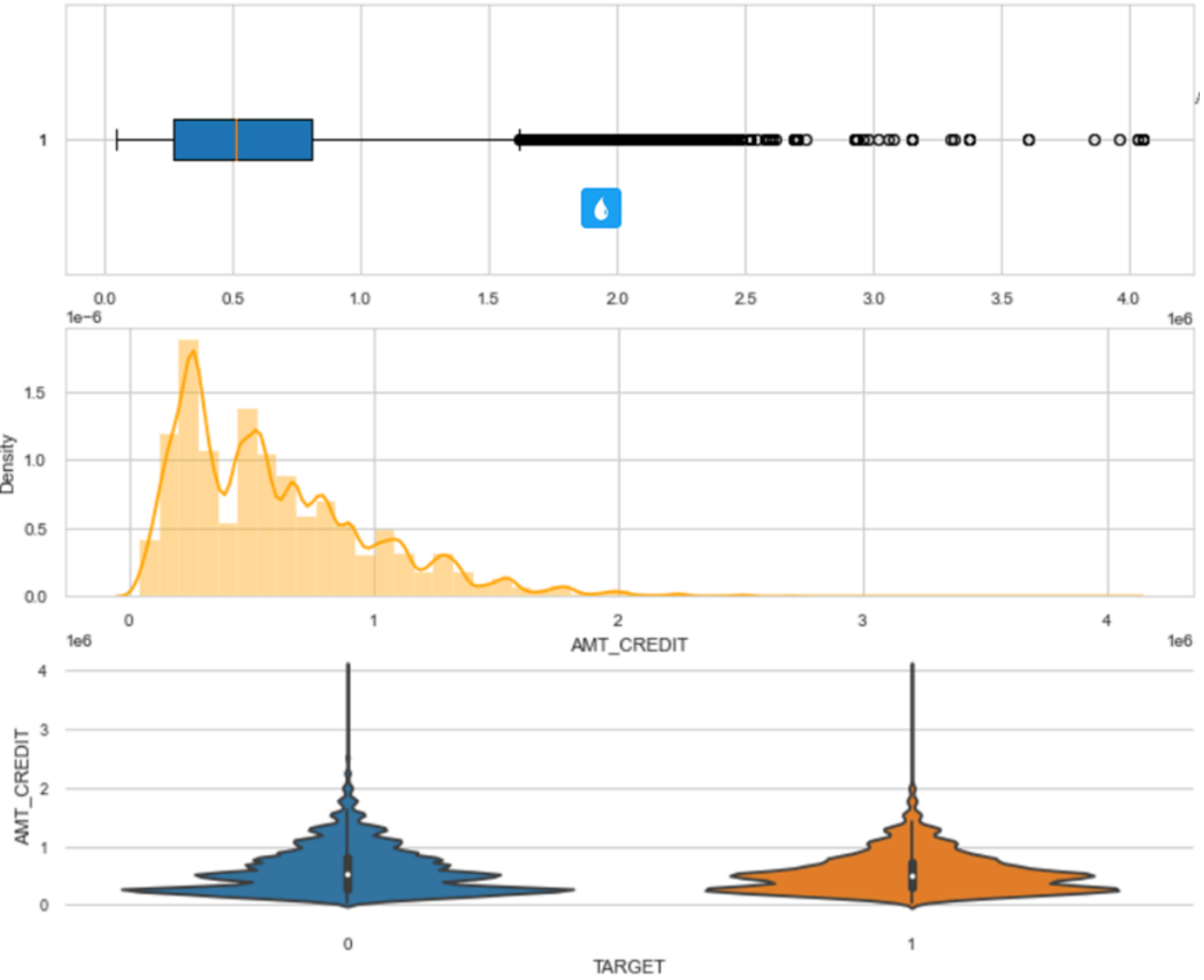
AMT_INCOME_TOTAL: Income of the client



An average income is around 150.000 and most people earn around 180.000 US dollars. In the KDE graph, there is a trend based on the client's income. *The more income total the clients earn, the probability that they have much difficulty in repaying loans is reduced.*

People with low income around 30.000 or less tend to repay more than those with the average income.

AMT_CREDIT: Credit amount of the loan

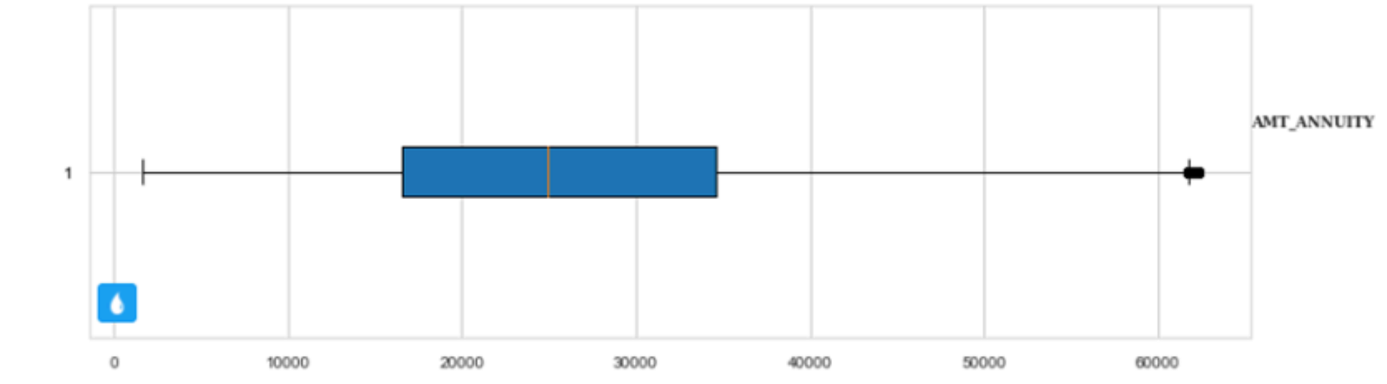


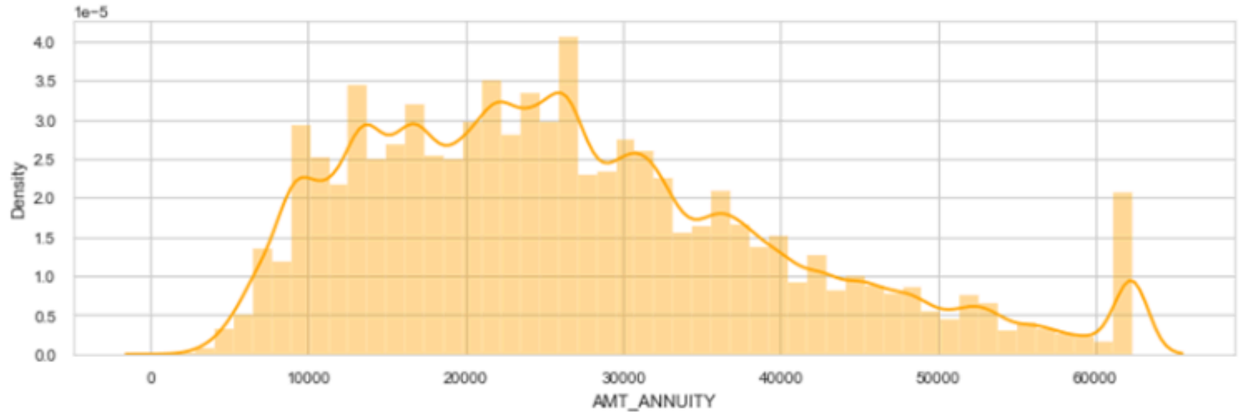
The median value of the credit amount of the customers who are capable of loan repayment is slightly larger than the median value of vustomers who are not capable of repayment.

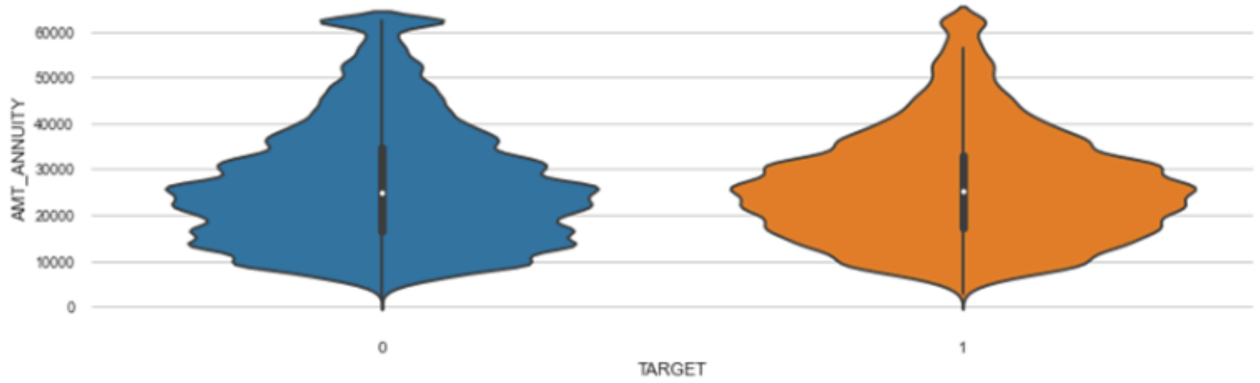
This basically means that the customers with higher credit amount have a slightly higher chances of being capable of loan repayment than customers with a lower credit amount.

A significant difference between repayable loans and difficulty repayable loans at the average credit amount (600,000 credit) where people tend to have difficulty at this amount.

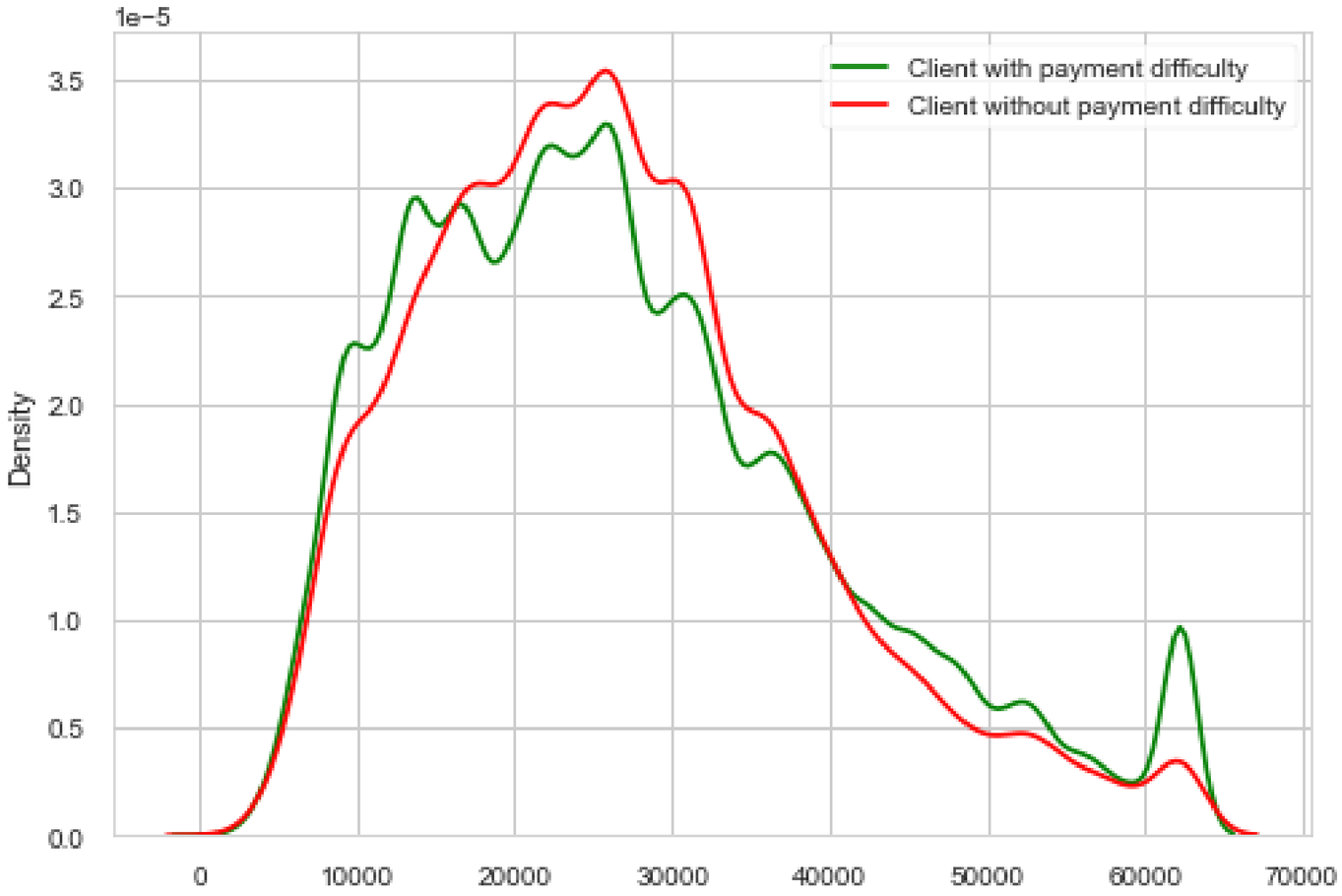
AMT_ANNUITY: Loan annuity





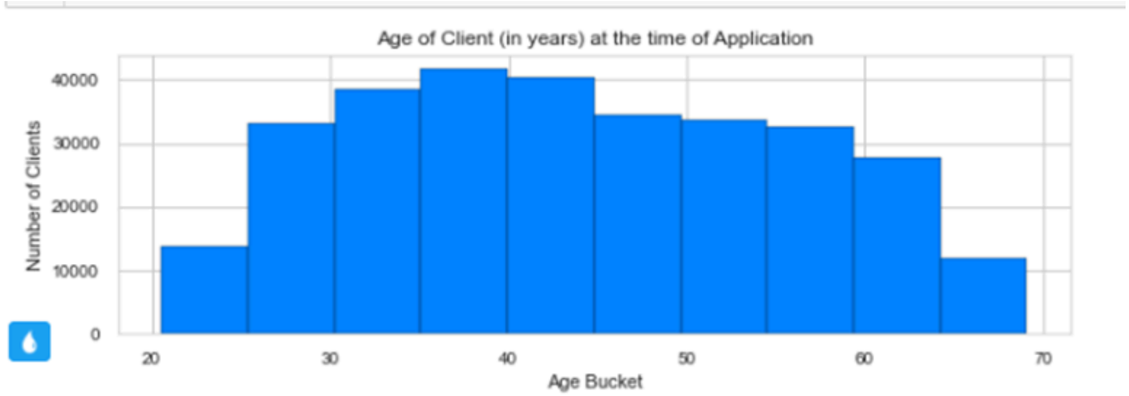


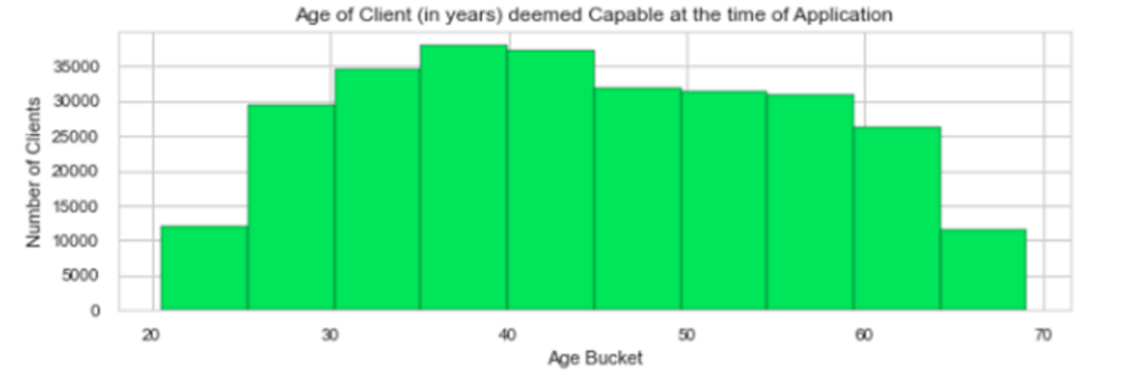
AMT_ANNUITY: Loan annuity

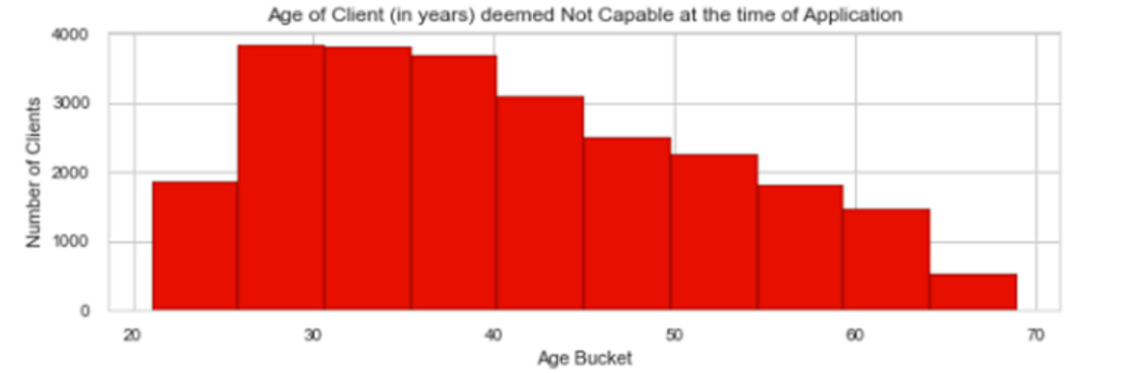


Clients with a medium salary are likely to apply for the loans. Most people pay an annuity below 50000 for the loans. In the KDE graph, the trend in clients with and without payment difficulty have not much differences. Before 40000, the proportion of clients with payment difficulty is higher than those without. Only after 40000, it gradually decreases.

DAYS_BIRTH: Client's age







Most people applying for loans are in the range of (35-40) years whereas this is followed by people in the range of (40-45) years whereas the number of applicants in people aged < 25 or aged > 65 is very low.

People in the same age buckets of (35-40) years and (40-45) years are deemed to be most capable.

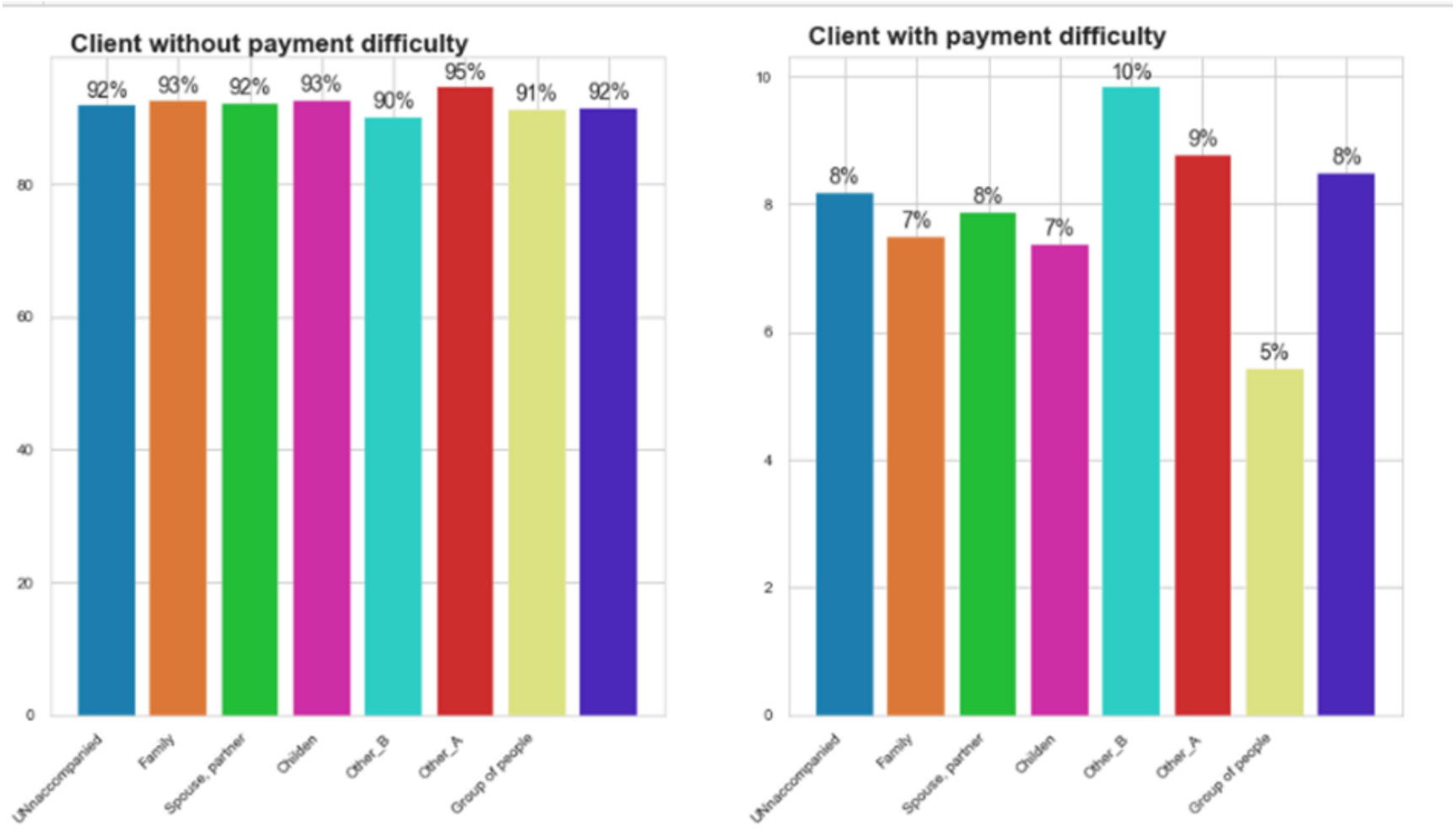
People aged in the bucket (25-30) years and (30-35) years have a large chance of being deemed not capable of loan repayment.

DAYS_BIRTH: Client's age

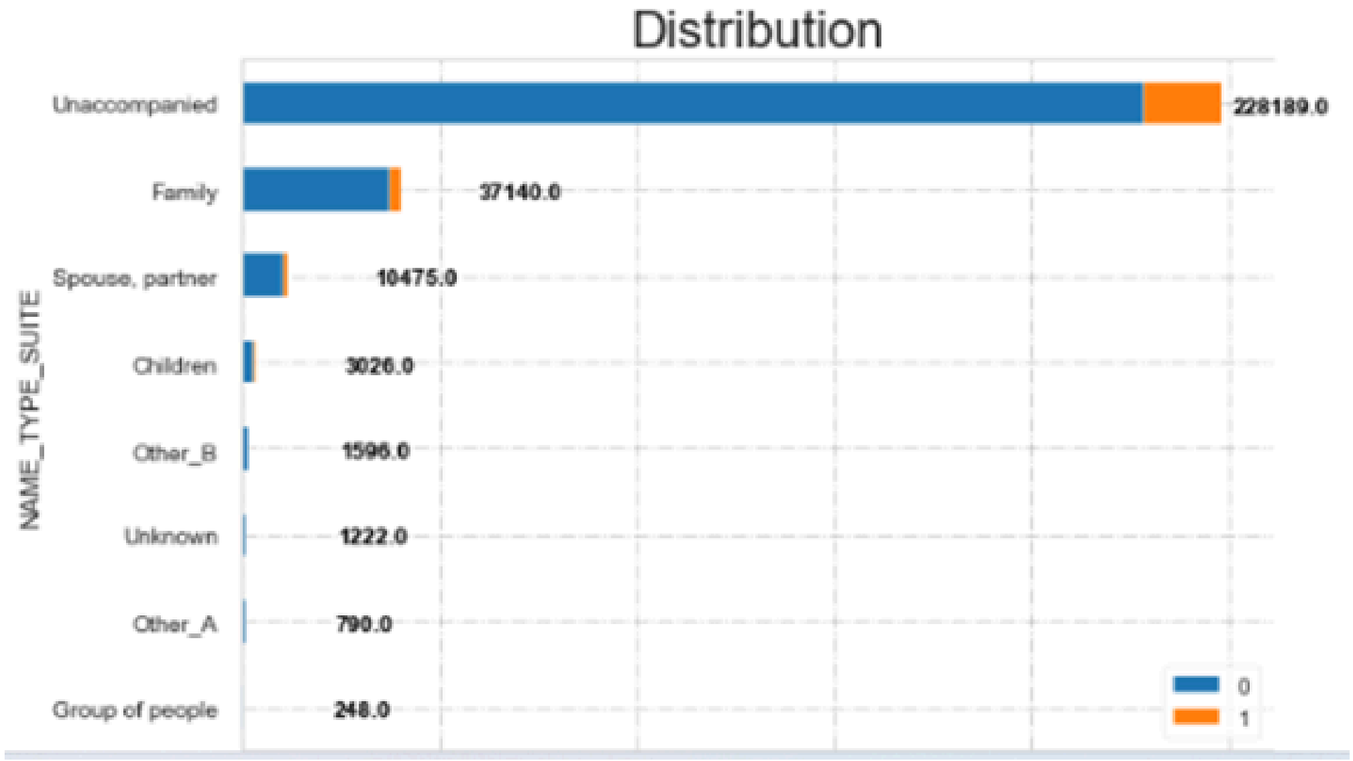
There is a clear trend: ***younger applicants are more likely to not repay the loan on time***. The rate of failure to repay is above 10% for the youngest three age groups and below 5% for the oldest age group.

To avoid this situation, the bank should ask younger clients to ***provide more guidance or financial planning tips***.

NAME_TYPE_SUITE: Who was accompanying client when applying for the loan



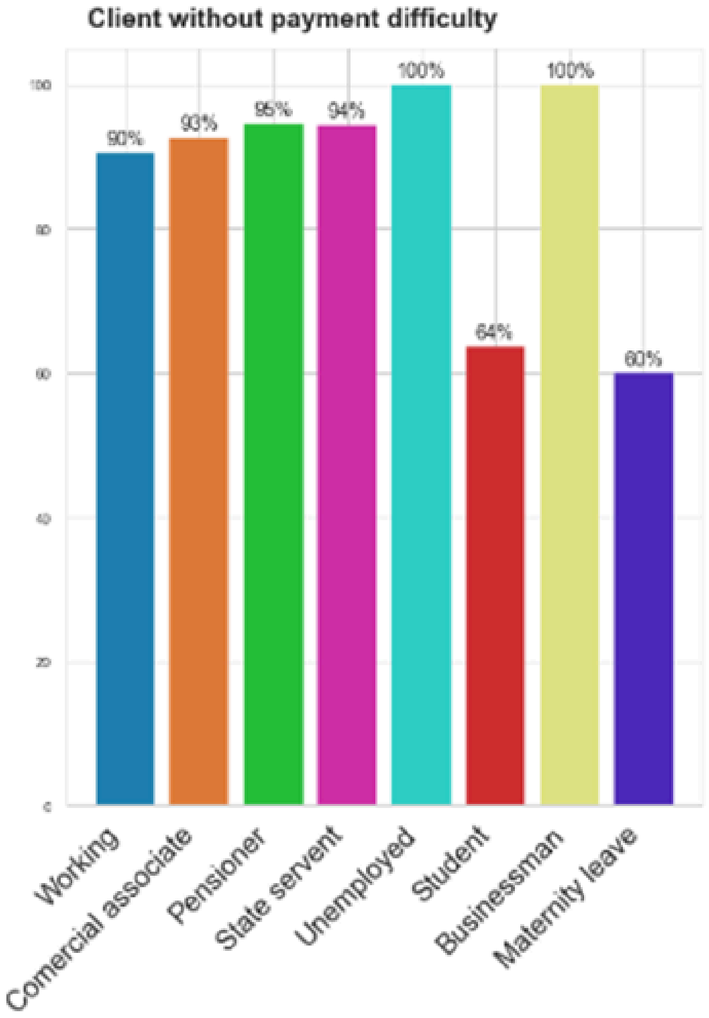
NAME_TYPE_SUITE: Who was accompanying client when applying for the loan

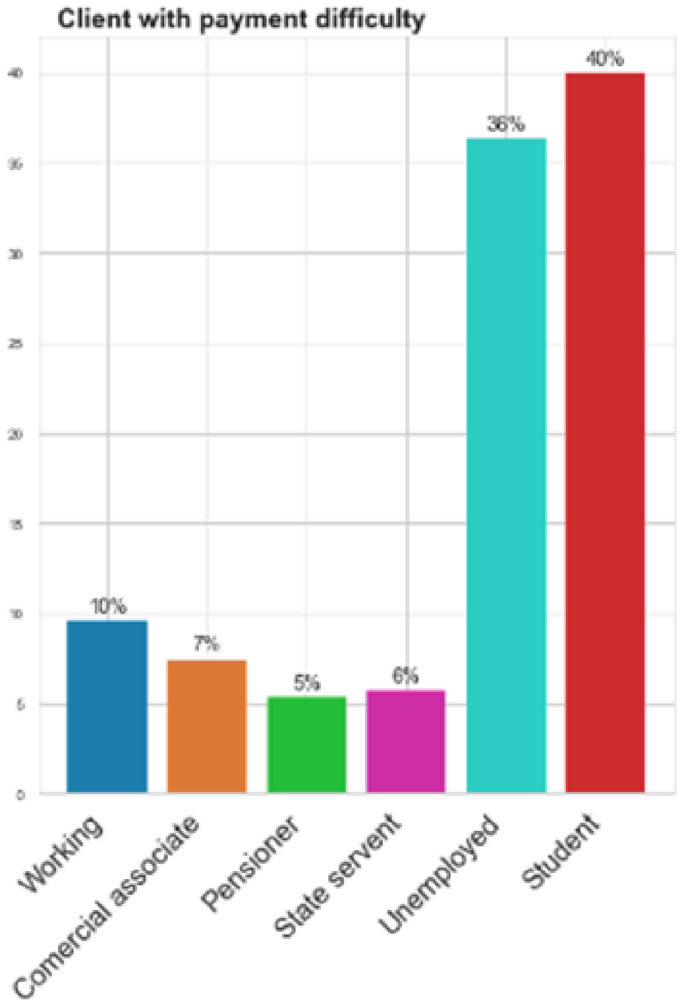


The client from unaccompanied to the bank in the most number of cases, out of which approx. 92% of the time, the bank finds the client to be capable of loan repayment whereas the *remaining 8%* of the time, the client is *not capable* of the same.

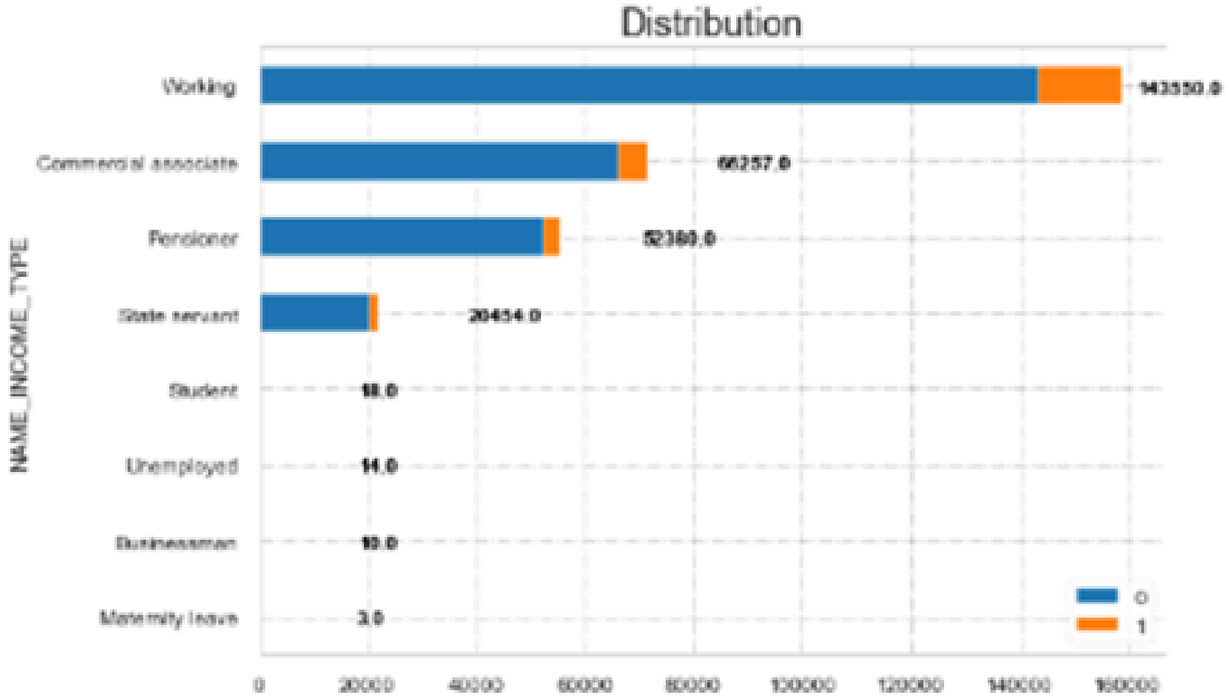
Clients without repayment difficulty don't have much difference from each feature, ranging from 90% to 96%. While in graph clients with repayment difficulty, we can see that clients from other_B have the highest rate, but other_B is not defined yet. Group of people feature have the lowest rate of having difficulty in repaying loans.

NAME_INCOME_TYPE: Clients income type (businessman, working, maternity leave,...)



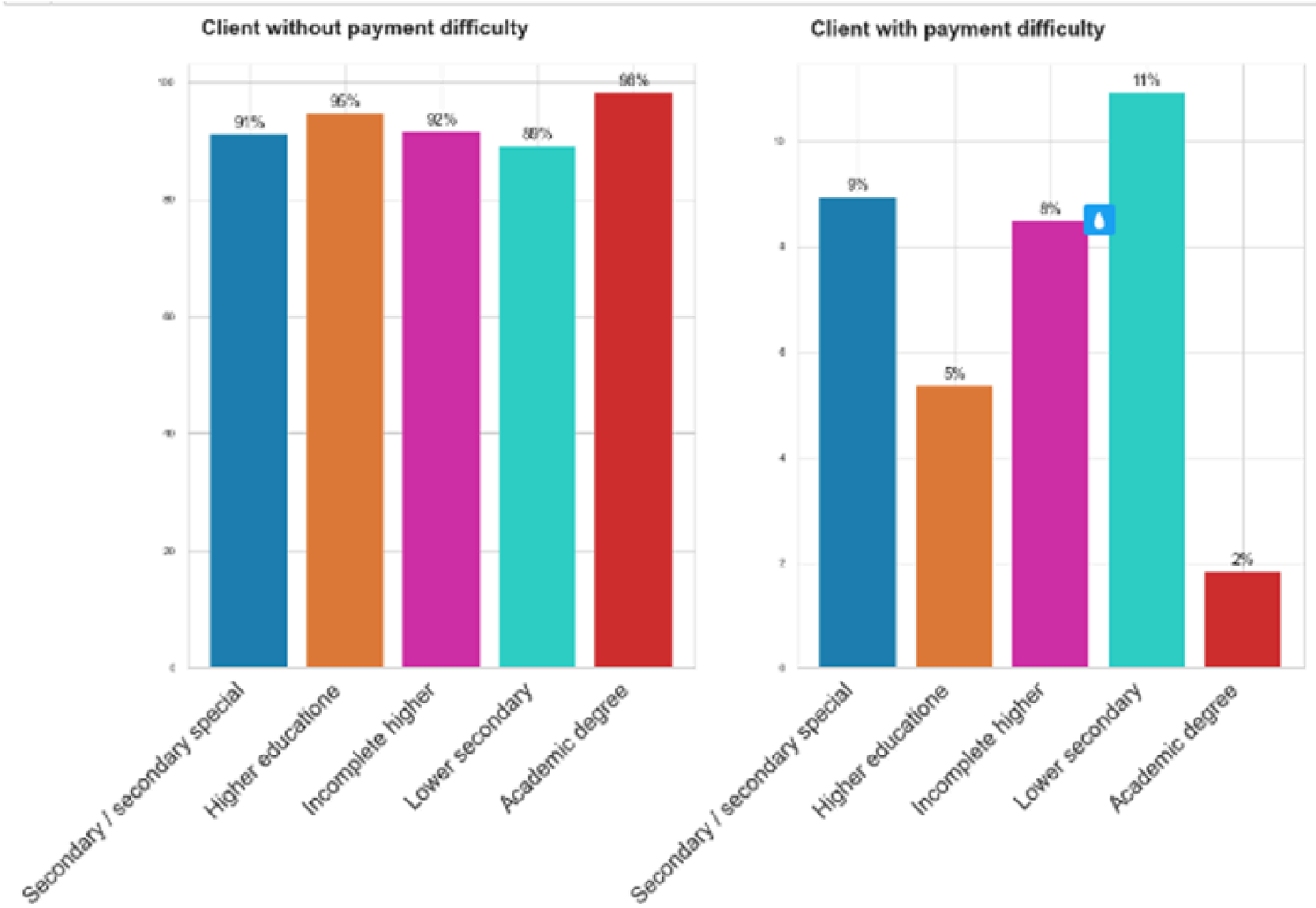


NAME_INCOME_TYPE: Clients income type (businessman, working, maternity leave,...)

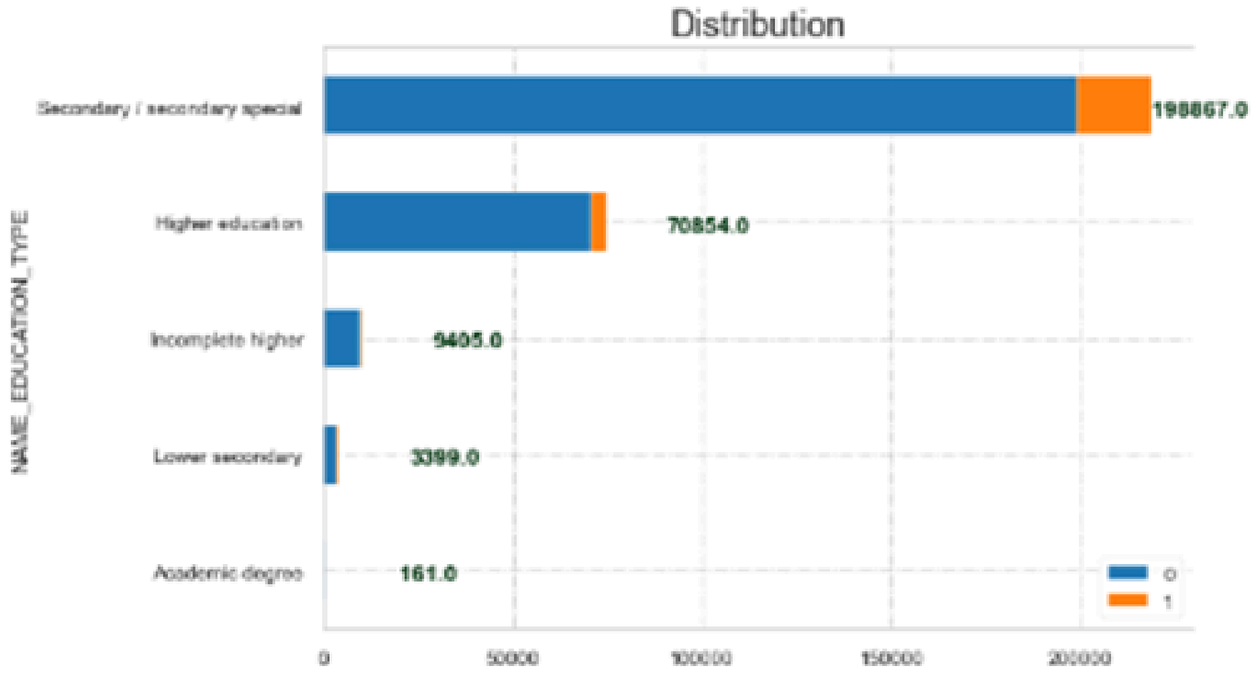


- The people who are working take the most number of loans whereas Commercial Associates, Pensioners and State Servants take considerably lesser number of loans
- We have very little data points related to Unemployed people, Students, Businessmen and women on Maternity leave. Again, there's a lot of variability in this scenario.
- One interesting observation over here is the fact that whatever loans the students and businessmen have applied to, they have been deemed capable of repayment of the same

NAME_EDUCATION_TYPE: Level of highest education the client achieved



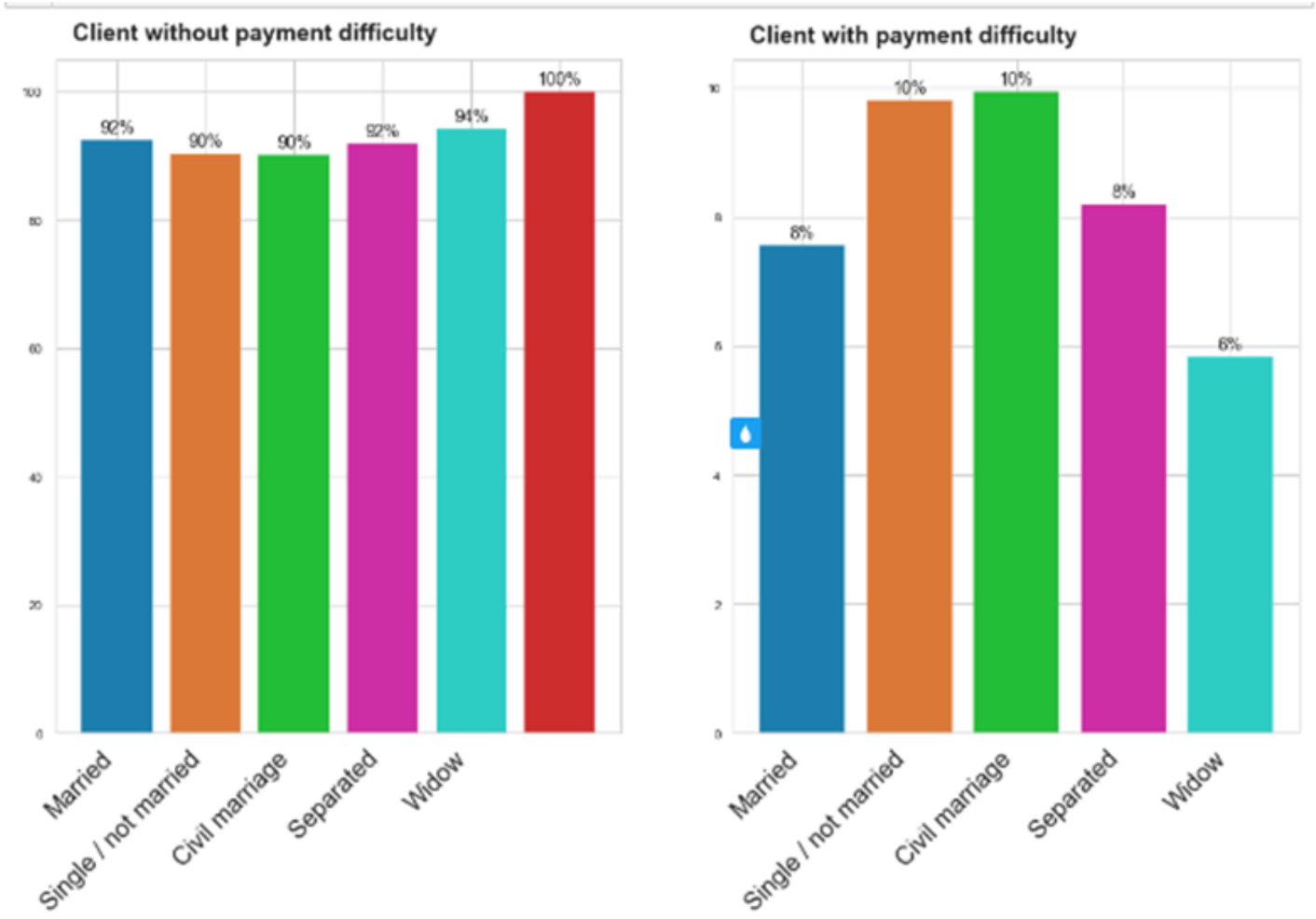
NAME_EDUCATION_TYPE: Level of highest education the client achieved



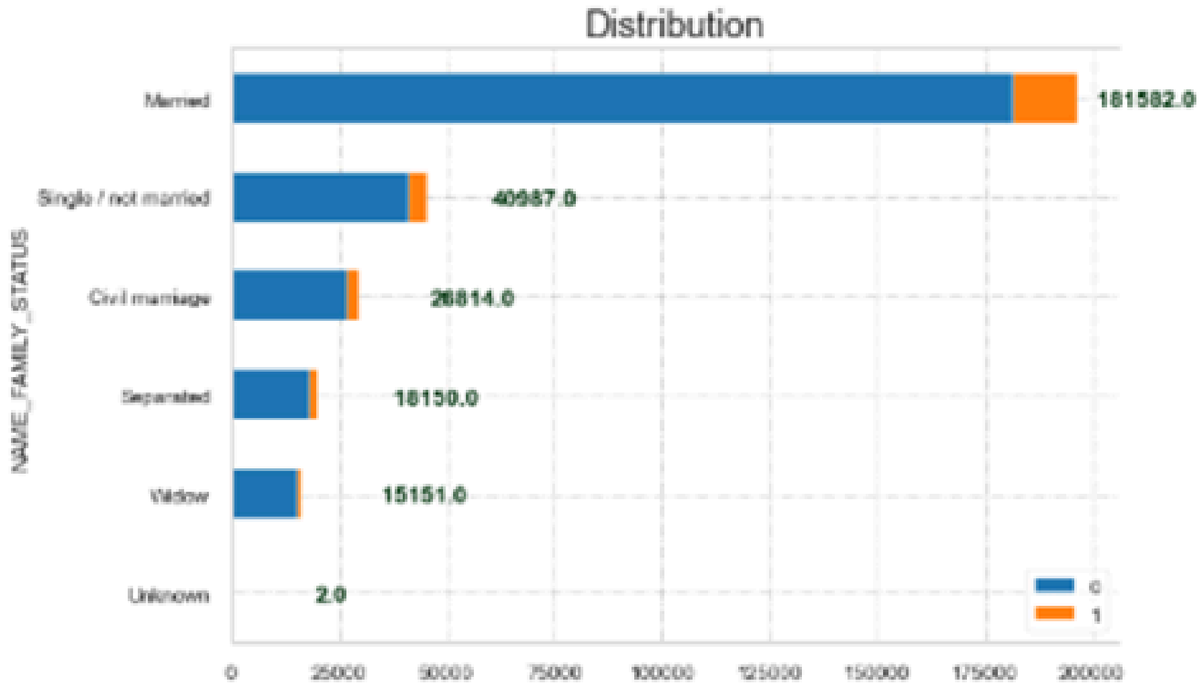
- Again, there's a lot of variability in this scenario among education types of the applicants.

- People with Secondary/Secondary Special as the highest level of education apply for the most number of loans and they are also the highest defaulters. However, the default percentage is not very different across various education levels.

NAME_FAMILY_STATUS: Family status of the client

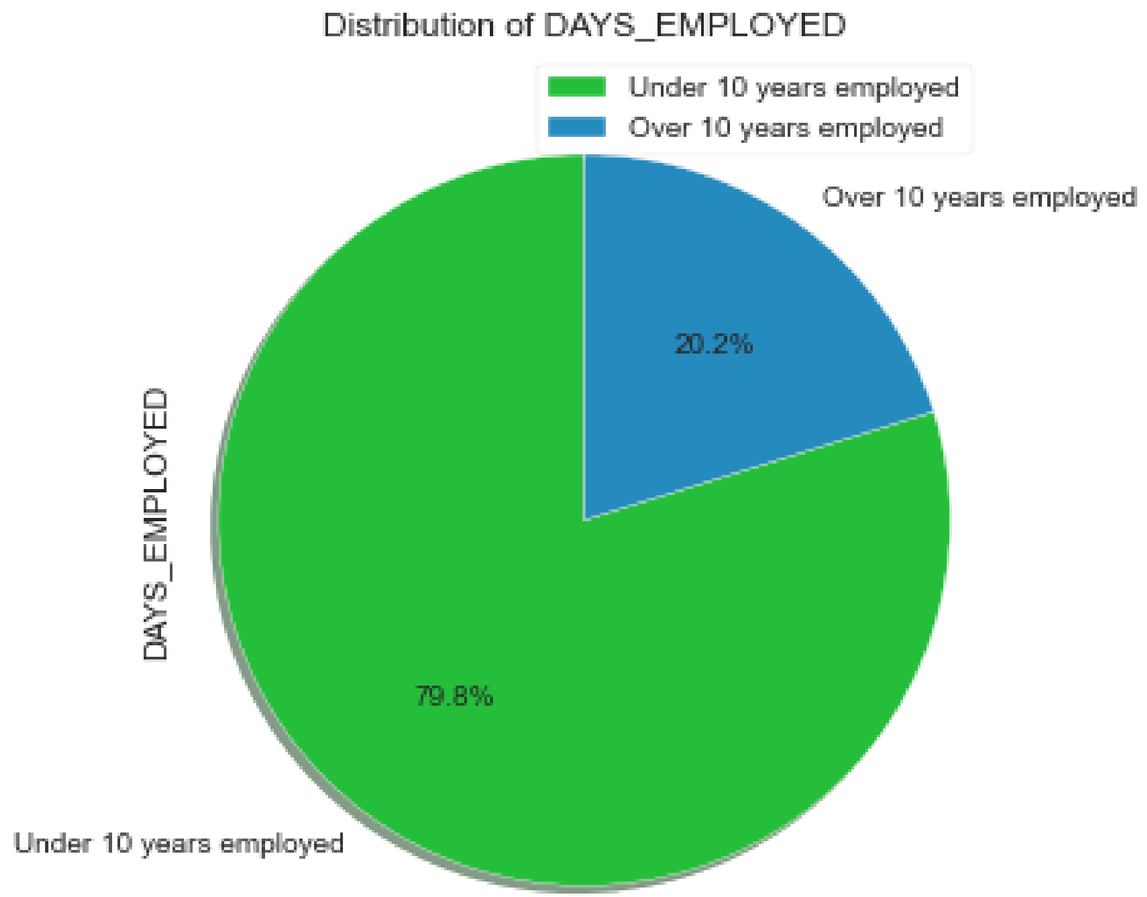


NAME_FAMILY_STATUS: Family status of the client



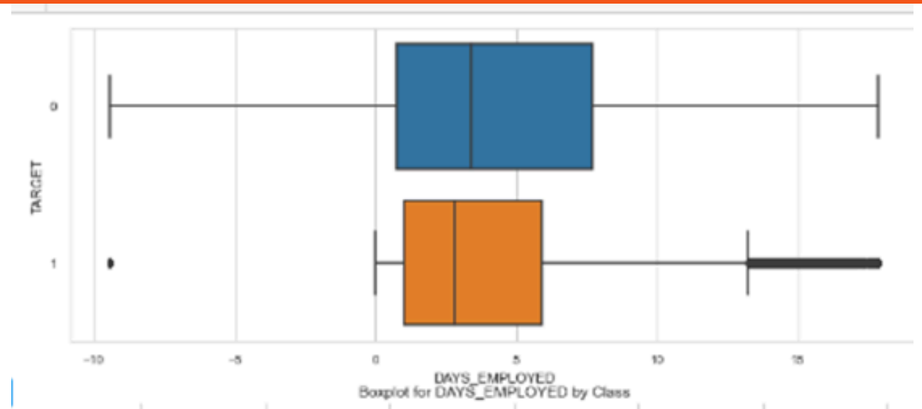
- There is variability among the Family Status of the applicants but there is not much variability if the majority class (Married) is ignored.
- Married people apply for the most number of loans and the number of people deemed incapable of repayment is also the highest.

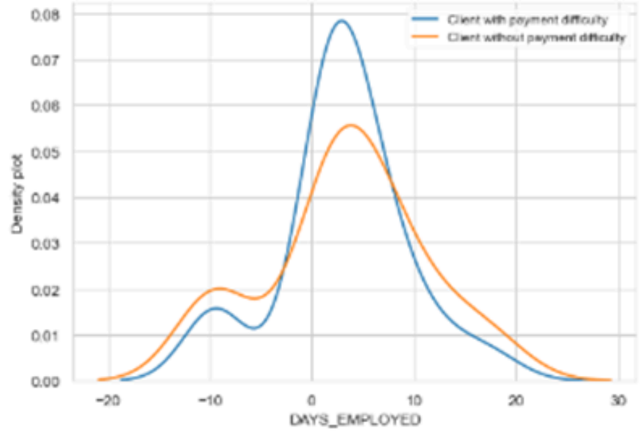
DAYS_EMPLOYED: How many days before the application the person started current employment



We could clearly see that client that worked under 10 years apply for the loans is forth times as much as the one worked over 10 year

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

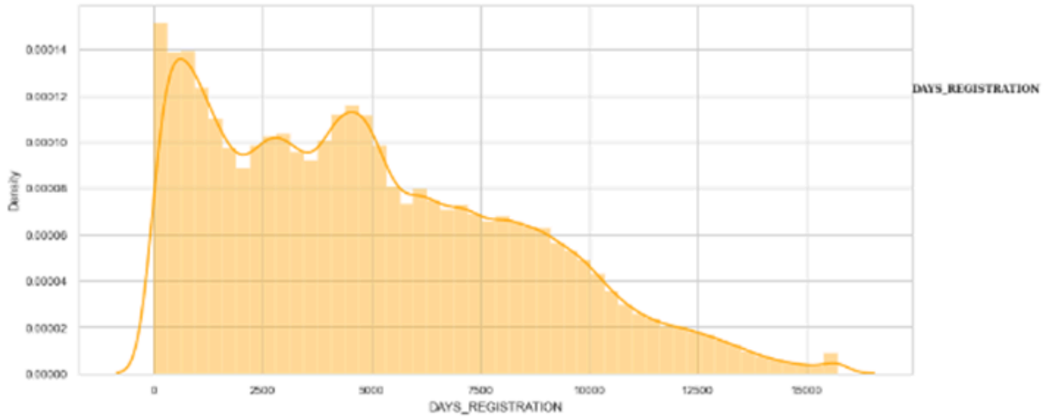


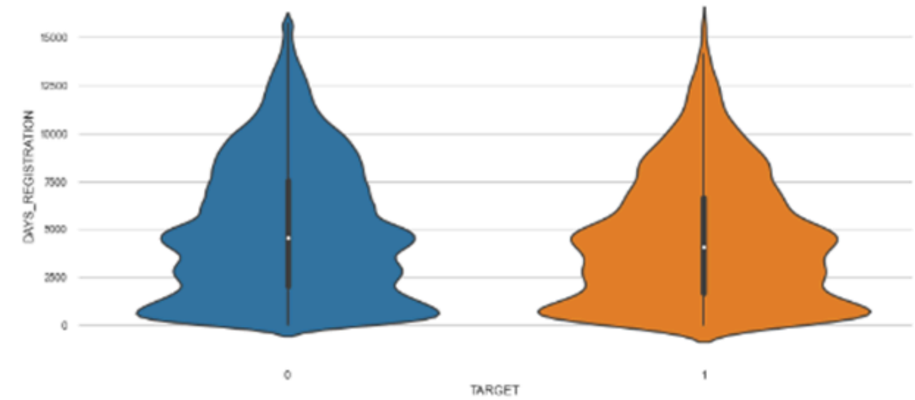


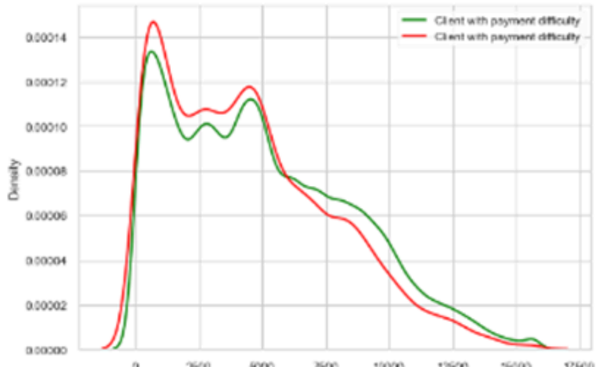
- Here also, we have already converted the days into years for easy analysis, and we can see from the histogram that most of the clients that have worked under 10 years (79.8%)

- Easy to understand that the longer you work the less difficult you find yourself repaying loans

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

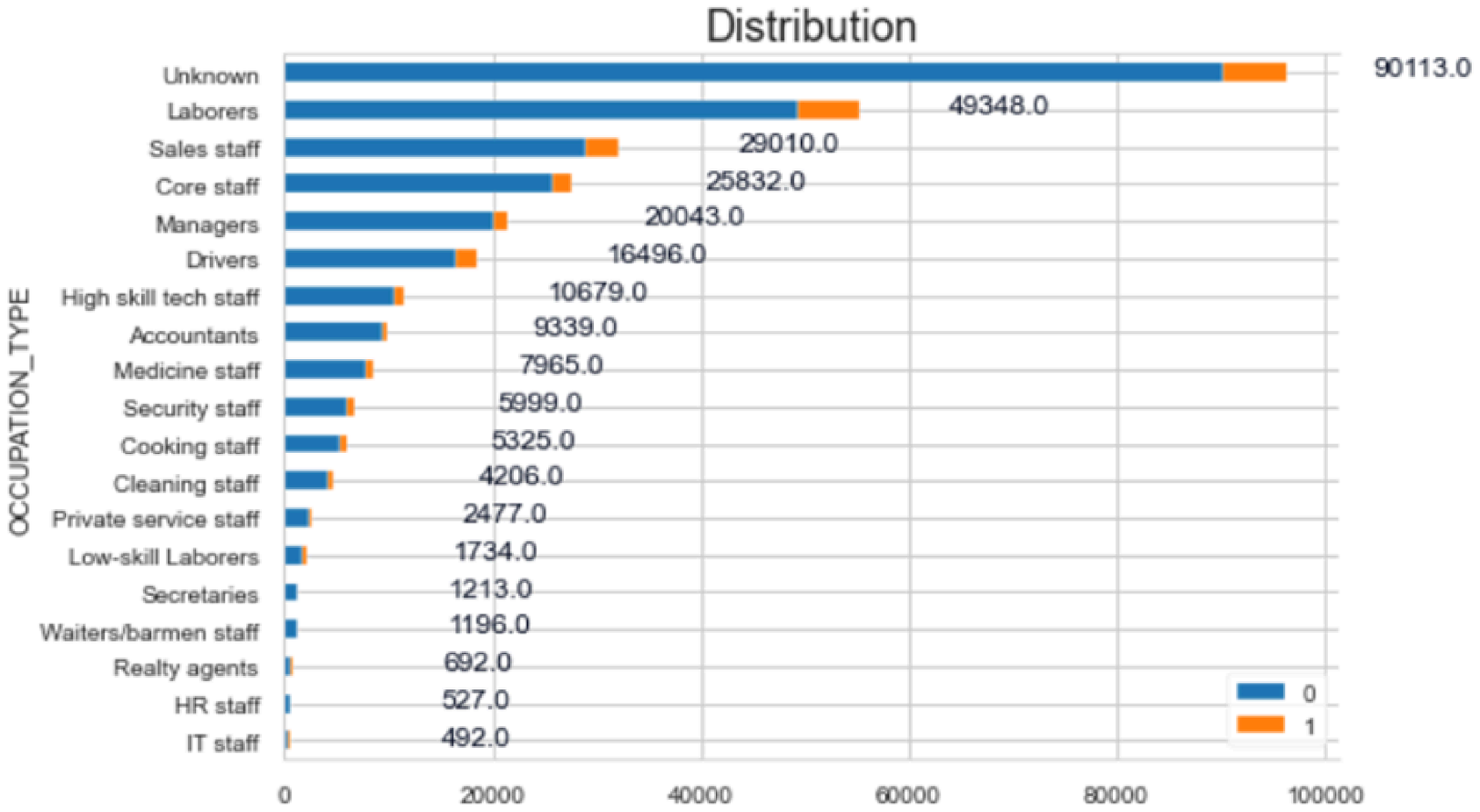




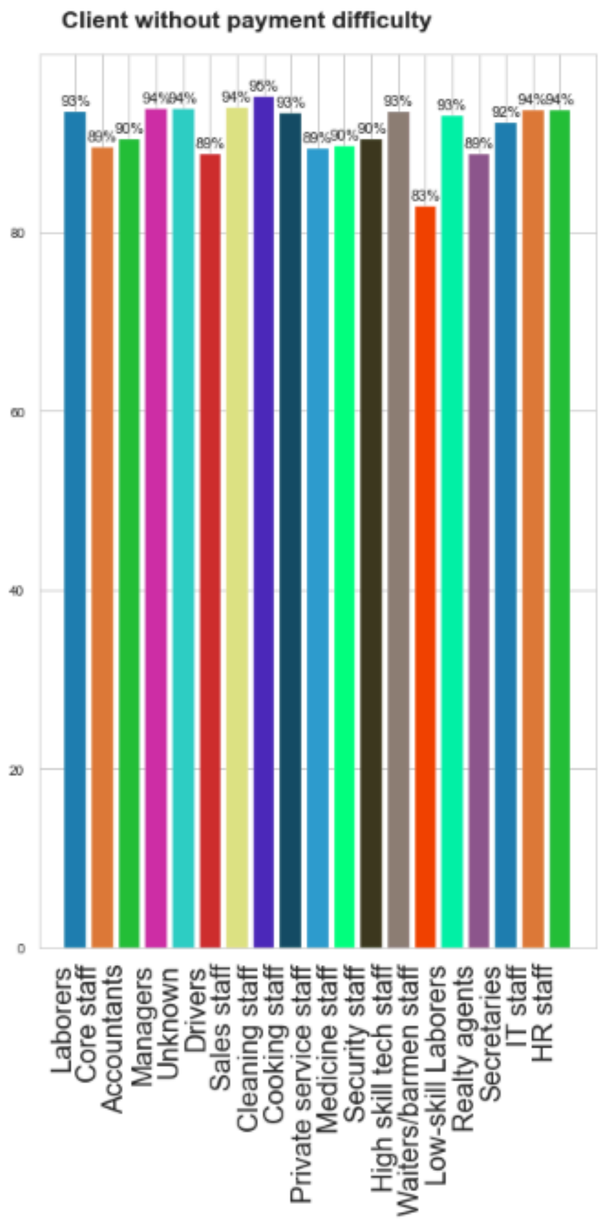


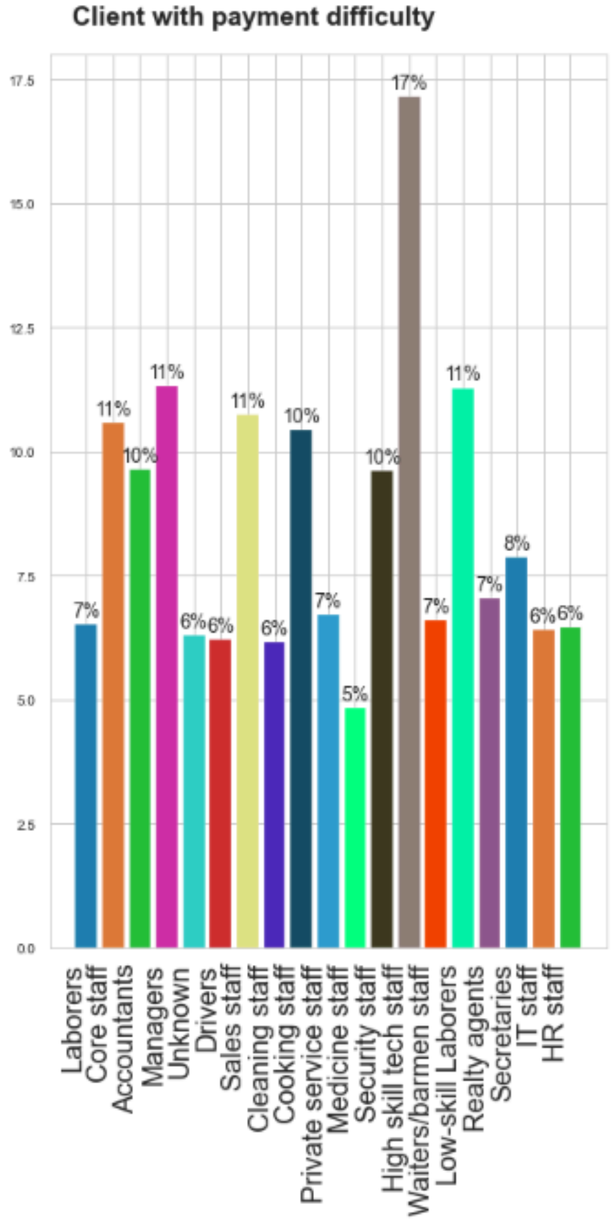
Most of the clients have changed their registration less than 15000 days (41 years) before the loan application, whereas in most cases it is less than 5000 days (13 years).

OCCUPATION_TYPE: What kind of occupation does the client have



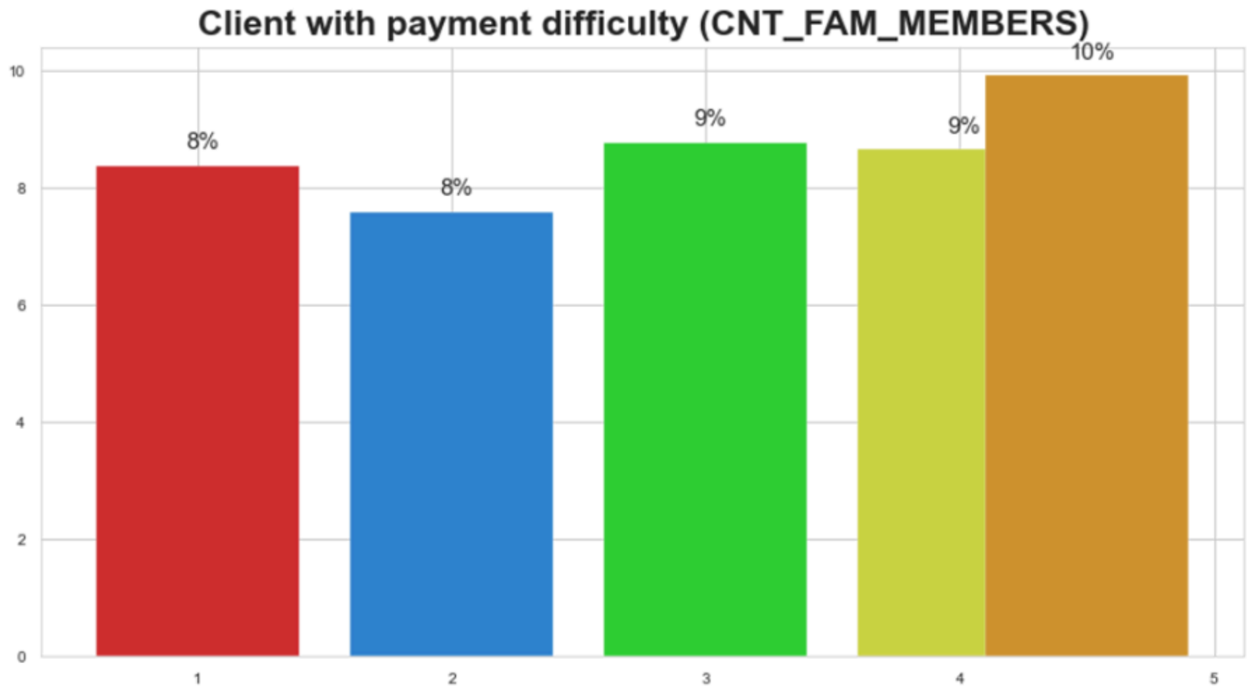
OCCUPATION_TYPE: What kind of occupation does the client have

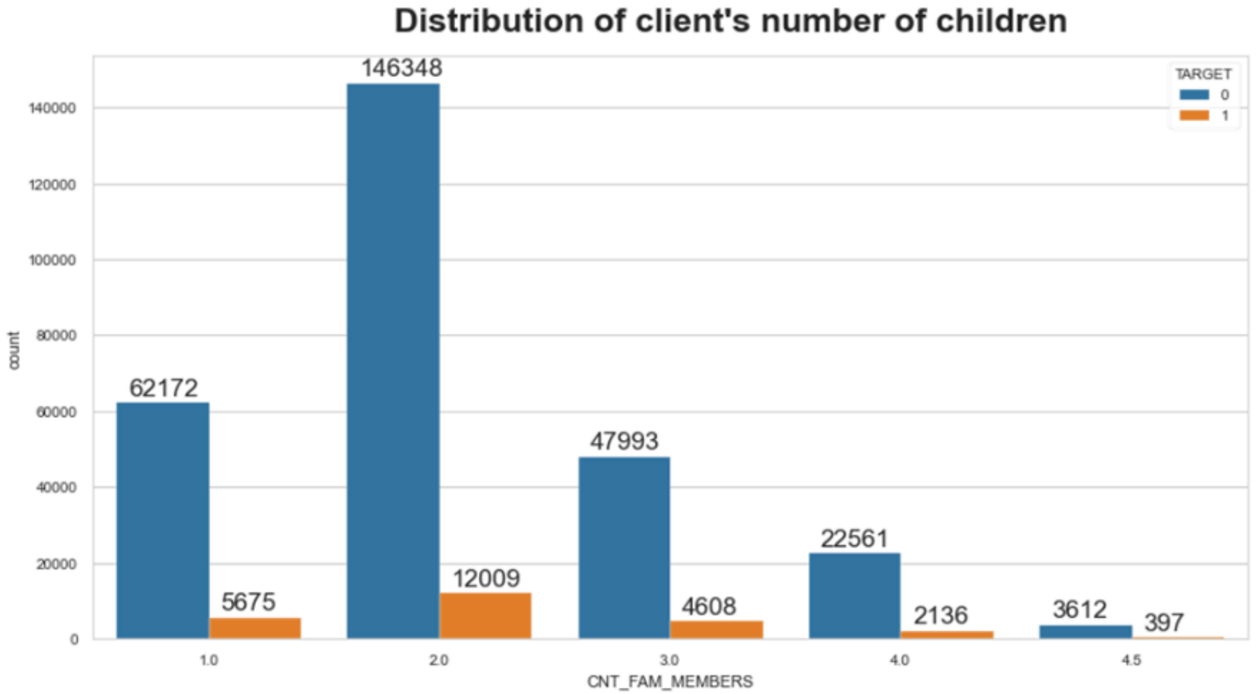




- Out of all the possible Occupation Types, the majority of applicants have not provided their occupation Type in the application (approx. 31.39%) which is followed by Laborers (approx. 18%).
- Out of all the occupations, Waiters/barmen staff are considered to be the least capable of repayment followed by Laborers Though laborers have considerably higher applications as compared to Waiters/barmen staff.

CNT_FAM_MEMBERS: How many family members does client have

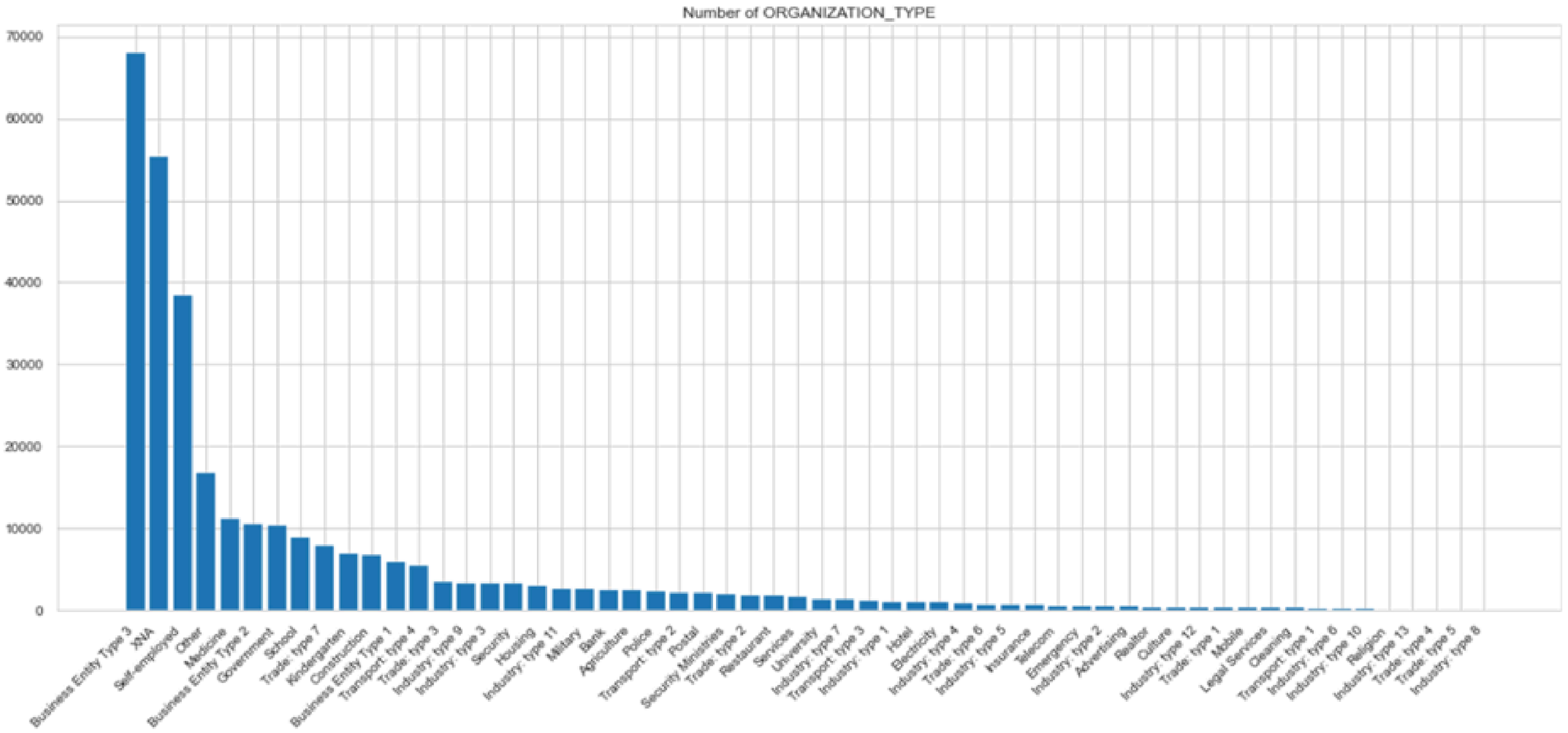




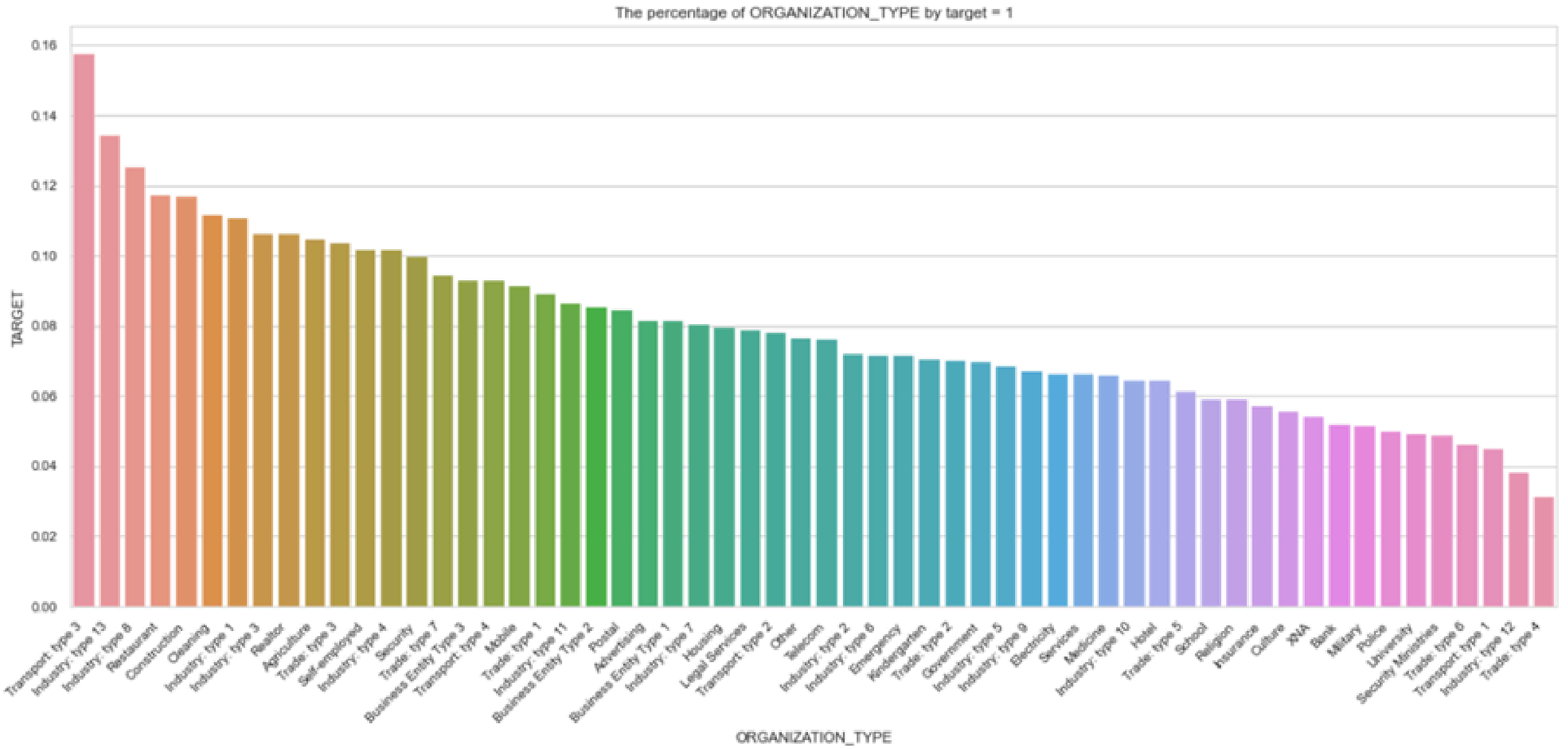
- Each family has about 1 to 5 members and most customers have 2 family members.
- Client' family that have 5 members tend to have more difficult with payment than the others
- Also the highest number of people who can pay their loan is from clients who have 2 family members.
- The more kids they have the smaller percentage that they apply for the loans

→ There is not much information and difference in the loan repayment status for the customer → This feature is not very useful

ORGANIZATION_TYPE: Type of organization where client works



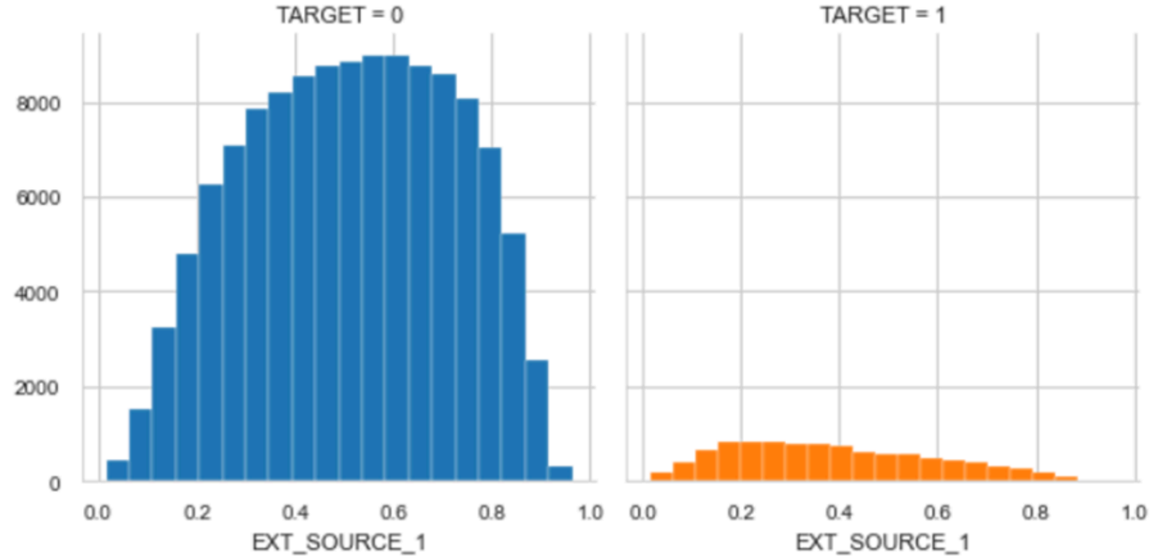
ORGANIZATION_TYPE: Type of organization where client works

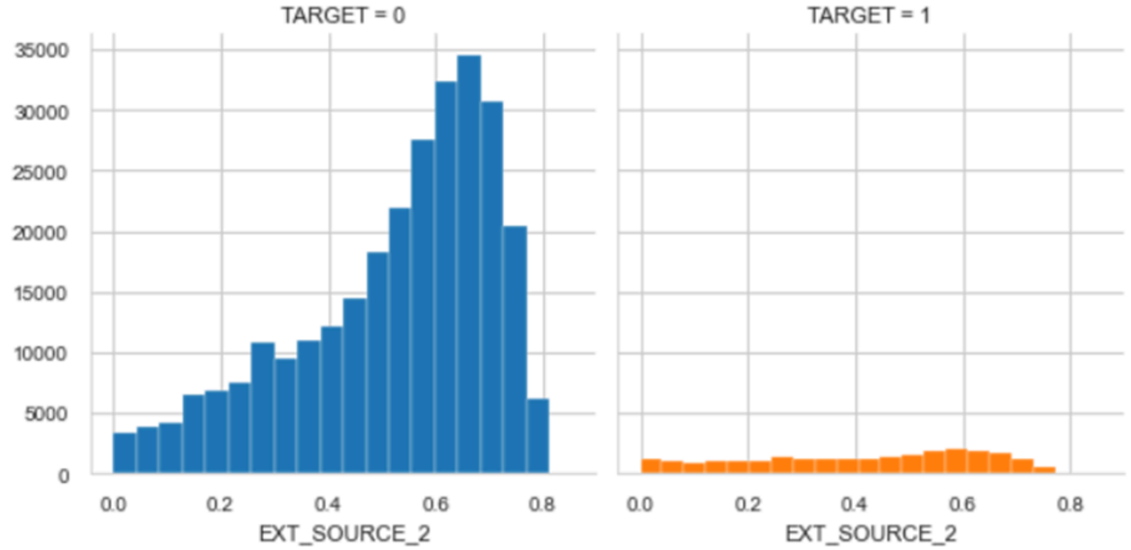


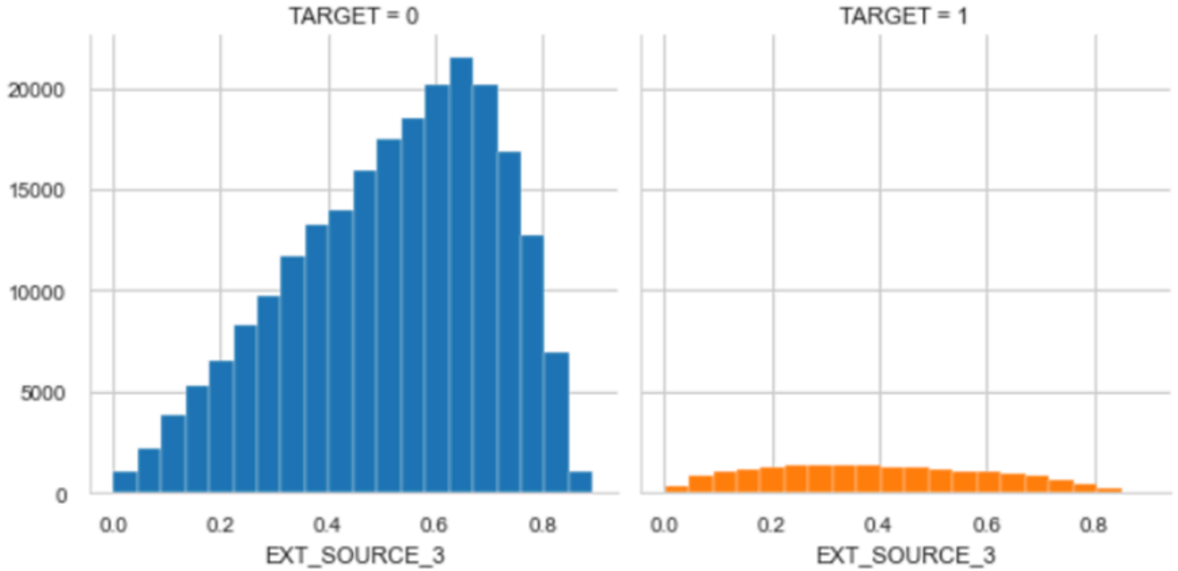
- Type of organization that have the highest non-repayment rate is: Transport: type 3 (16%), Industry: type 13 (14%), Industry: Type 8 (12%)

→ There is not much information and difference in the loan repayment status for the customer → This feature is not very useful

Distribution of EXT_SOURCE

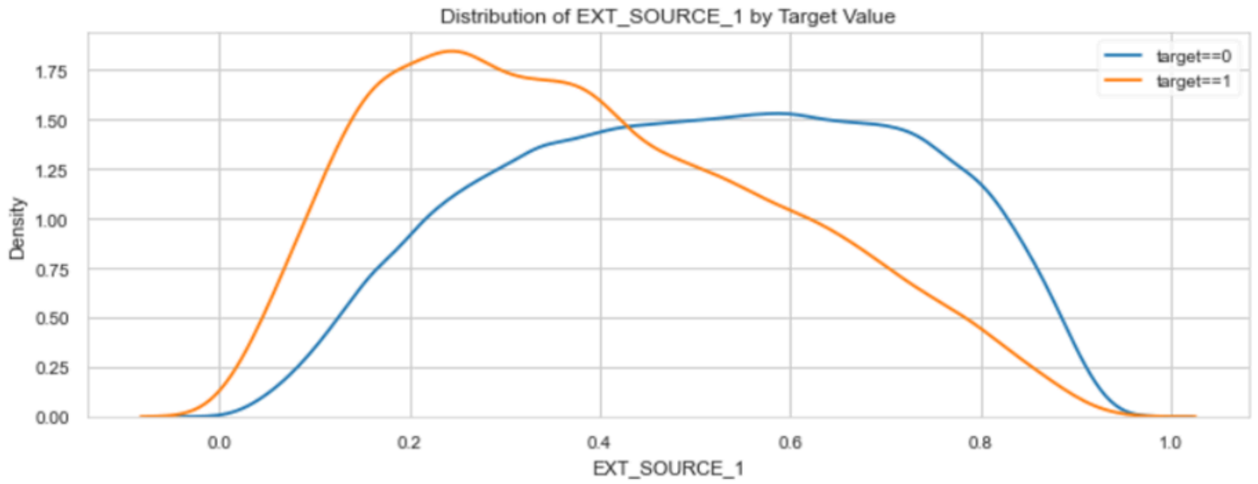


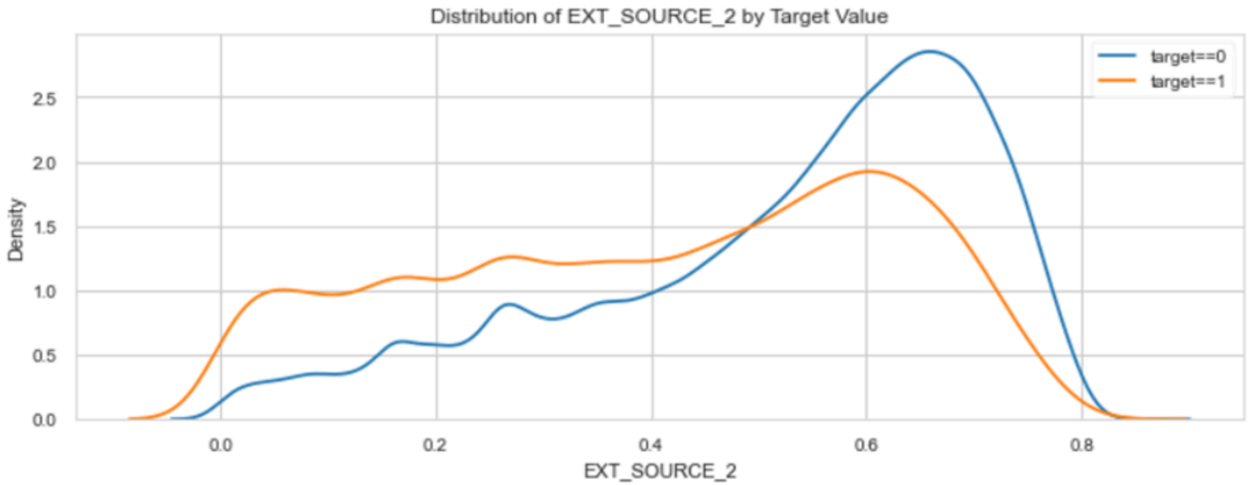


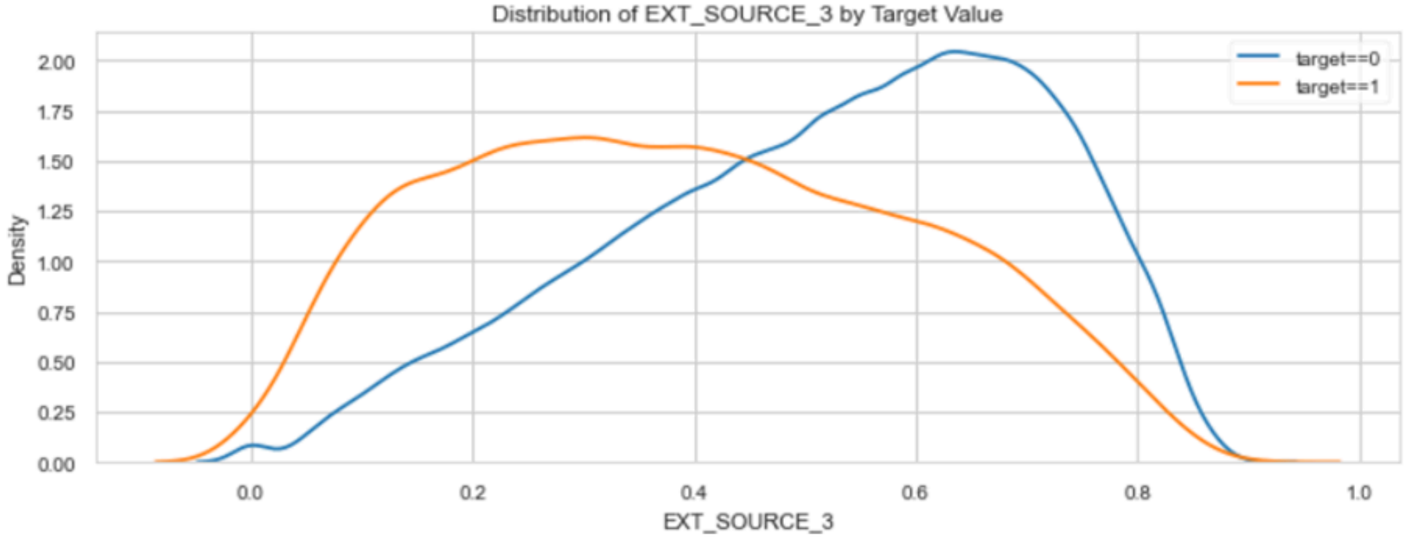


- Since these 3 features are the highest negative correlation among all, it shows a strong negative correlation with the target variable. The higher the score, the lower the probability that the person will default on their loan.
- From these visualizations, we glean that borrowers with higher scores from all three external sources were more likely to repay their loans.

Distribution of EXT_SOURCE

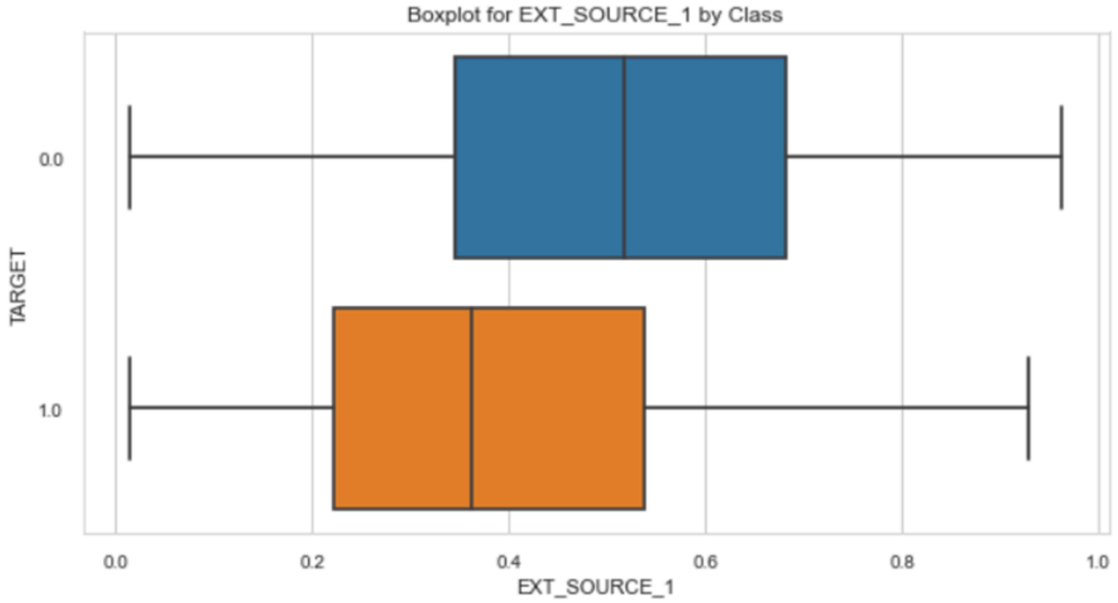


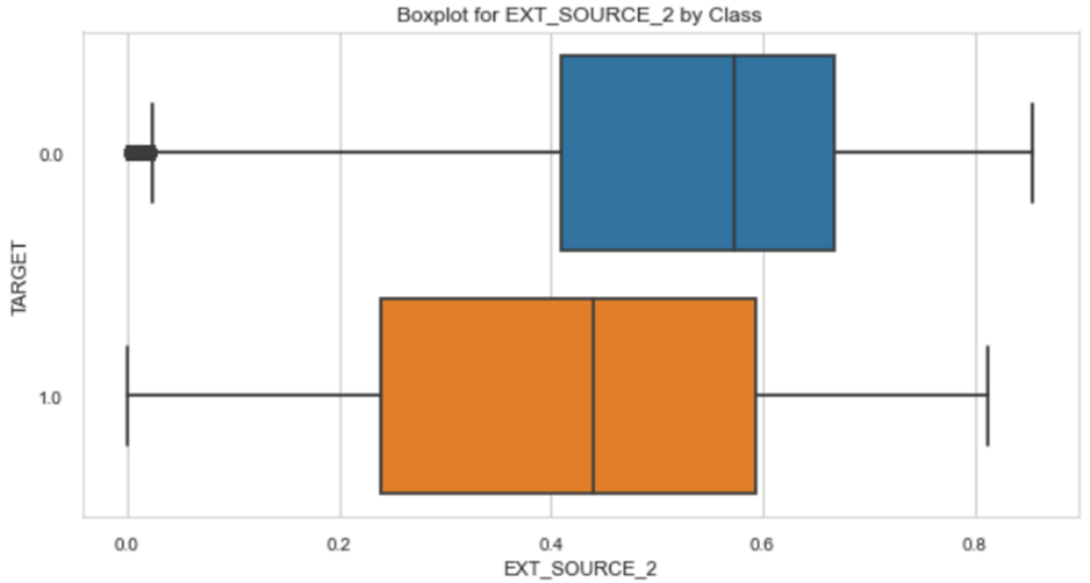


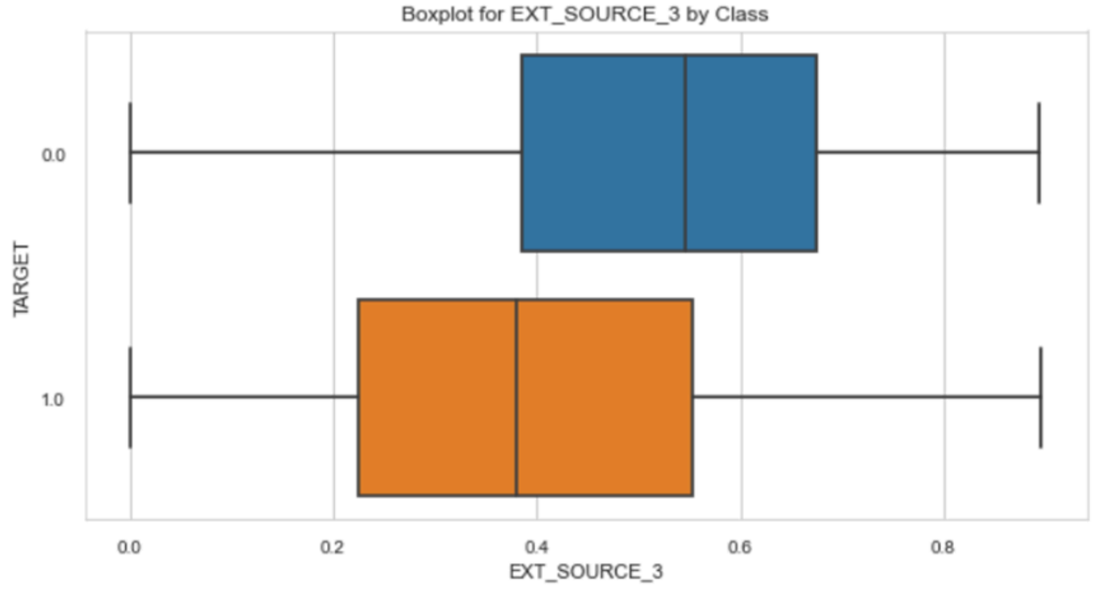


Three features that we are seeing where there are some considerable differences among the 2 classes, as we can see from the density plot. Therefore, they are going to be an important feature.

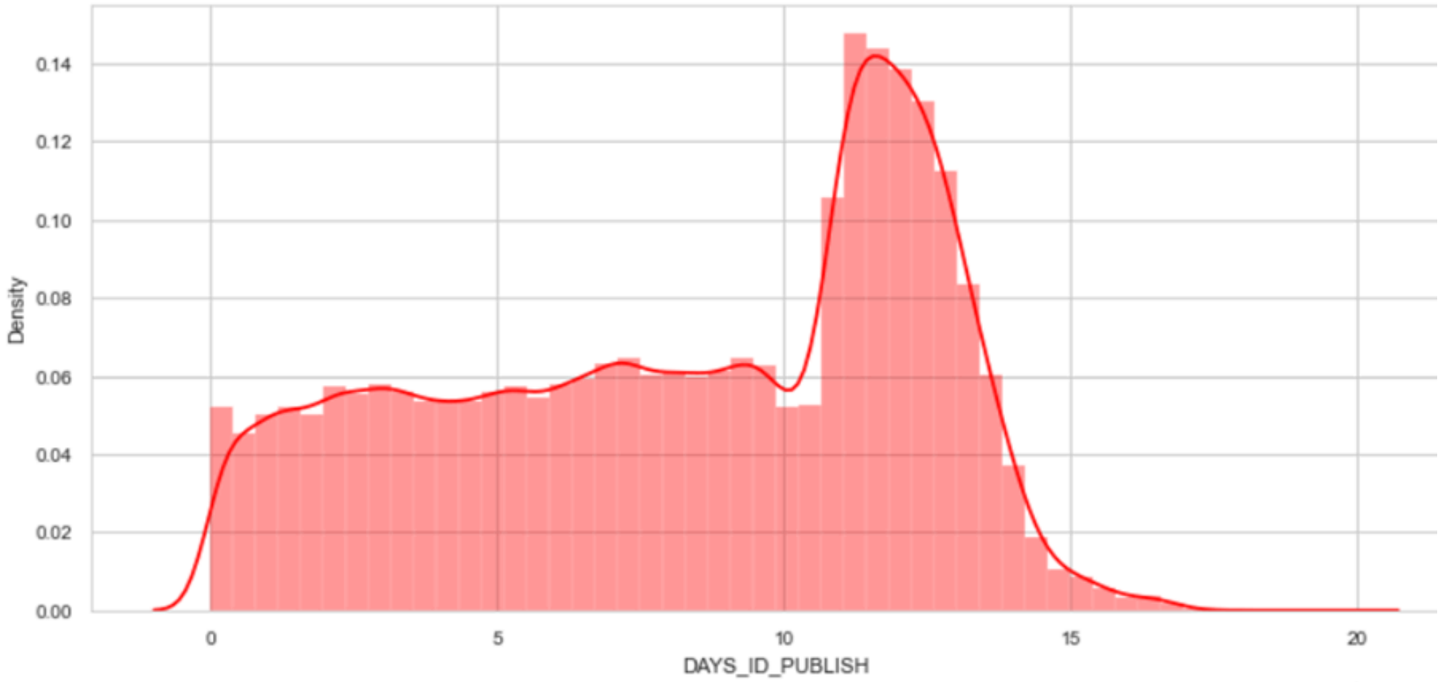
Distribution of EXT_SOURCE



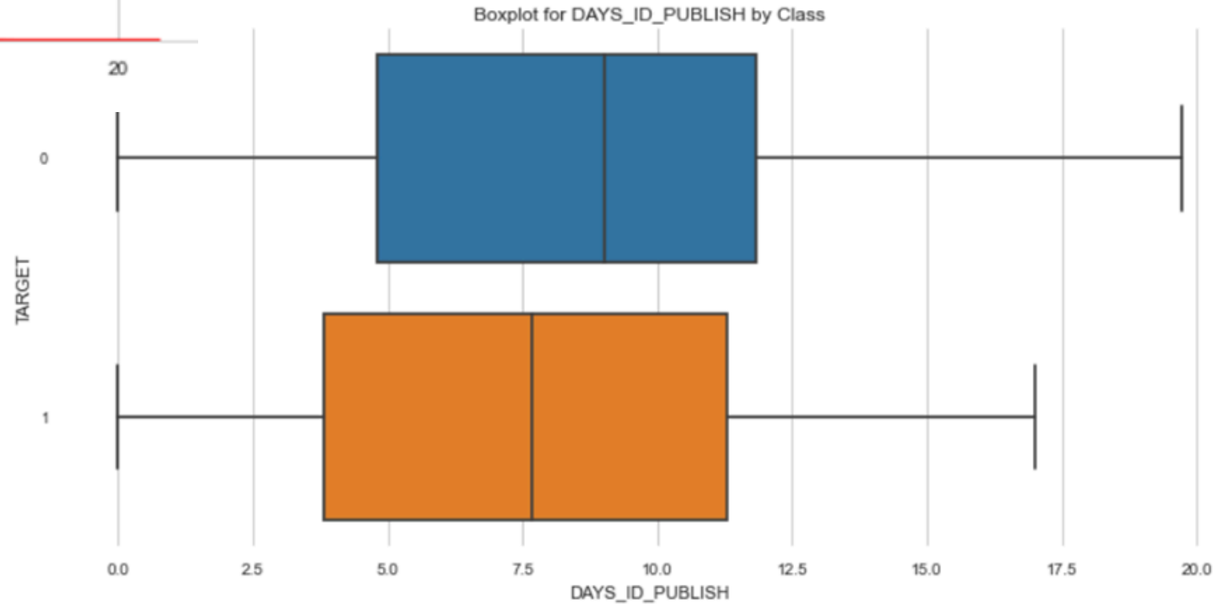




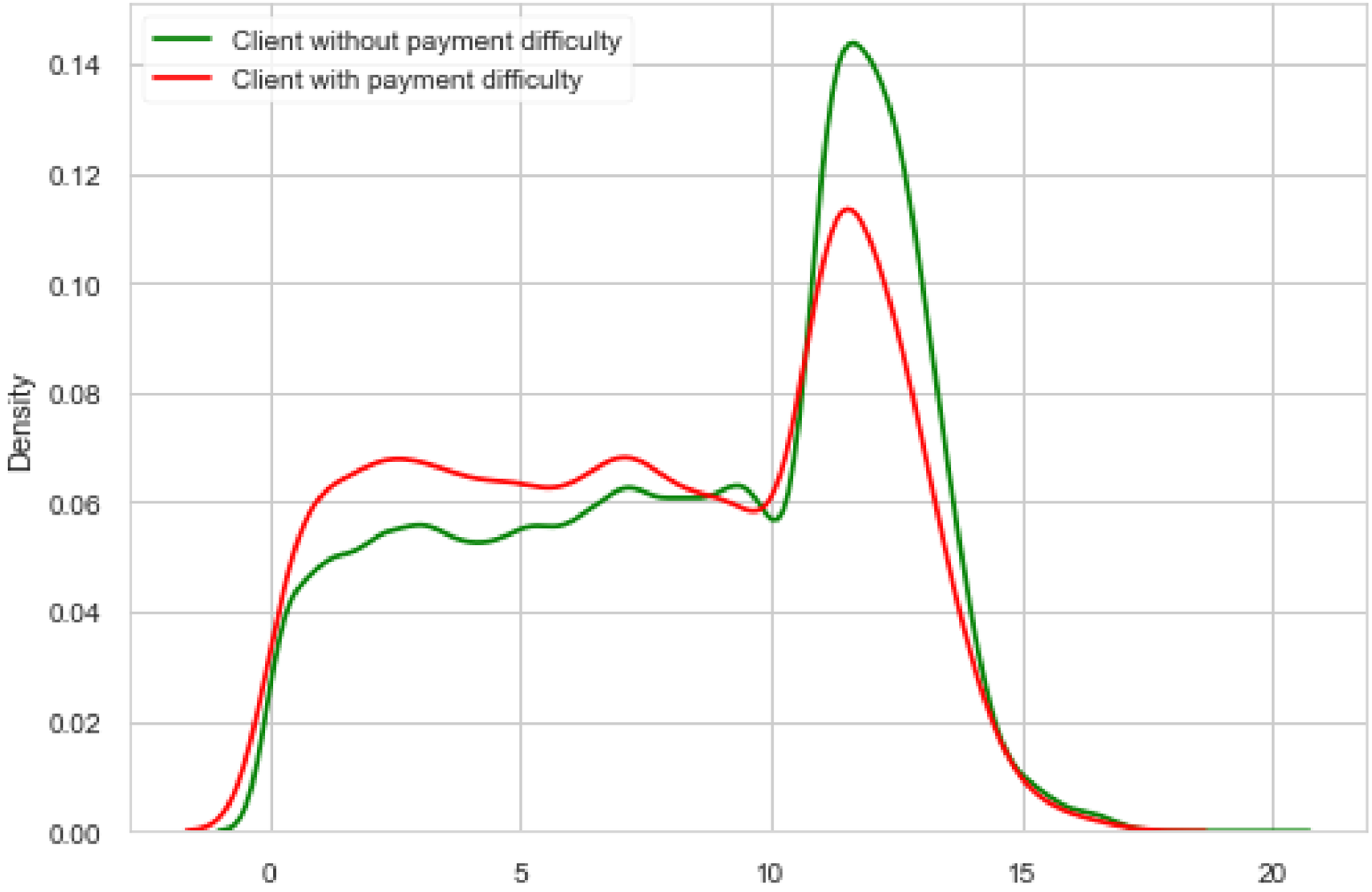
DAYS_ID_PUBLISH: How many days before the application did client change the identity document



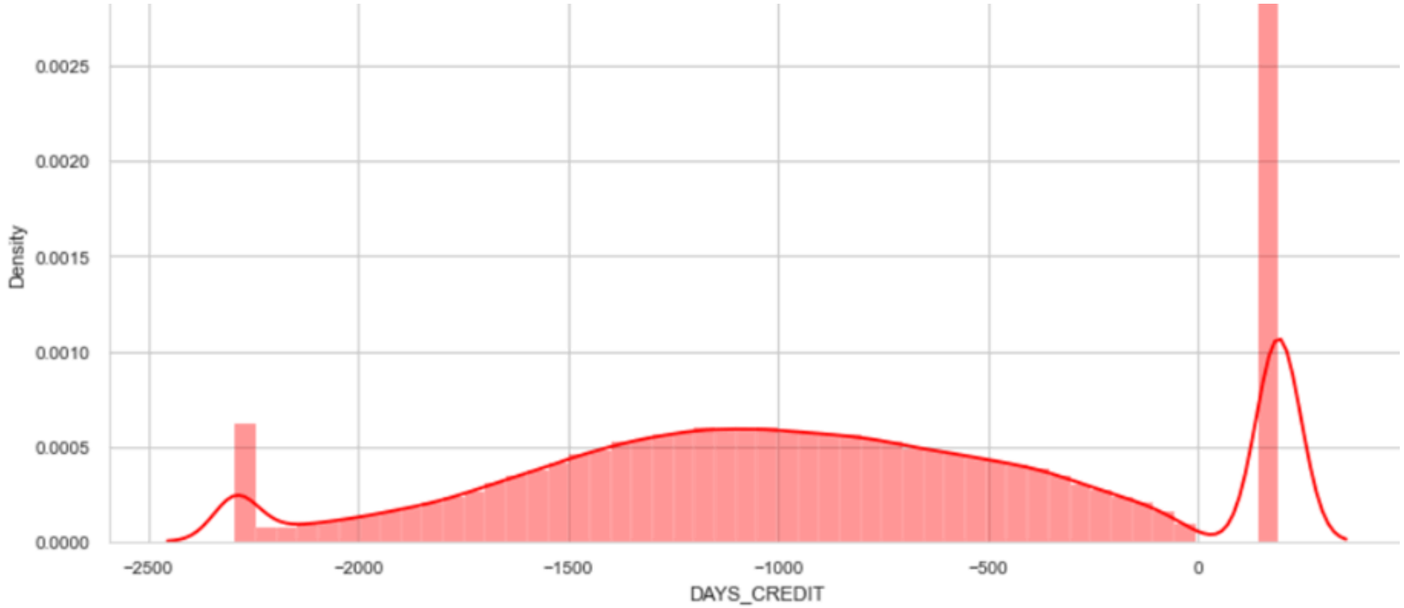
Most of the clients have changed their identity documents around 4000 days (10.95 years) before the application date.

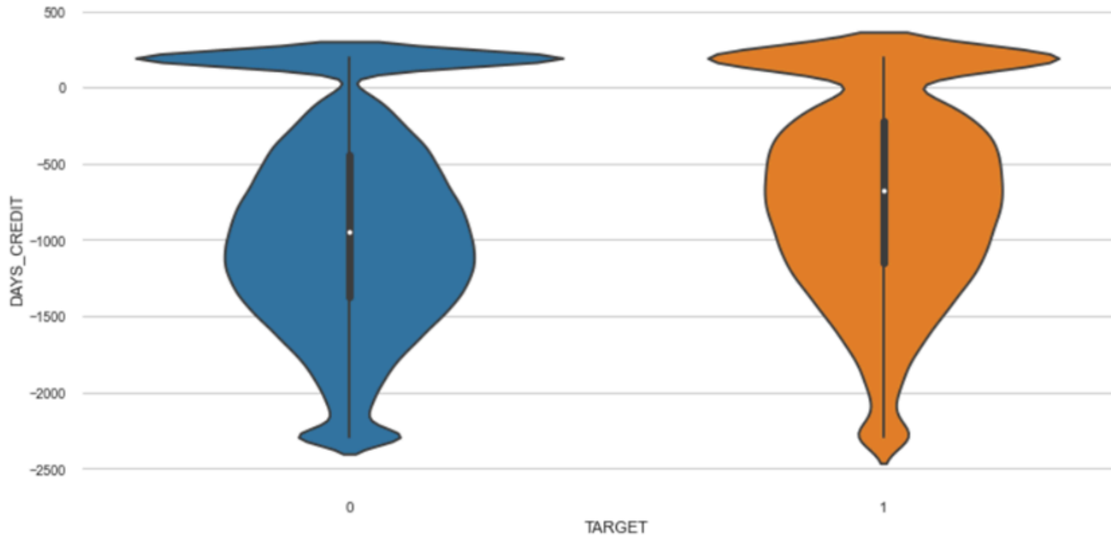


DAYS_ID_PUBLISH: How many days before the application did client change the identity document



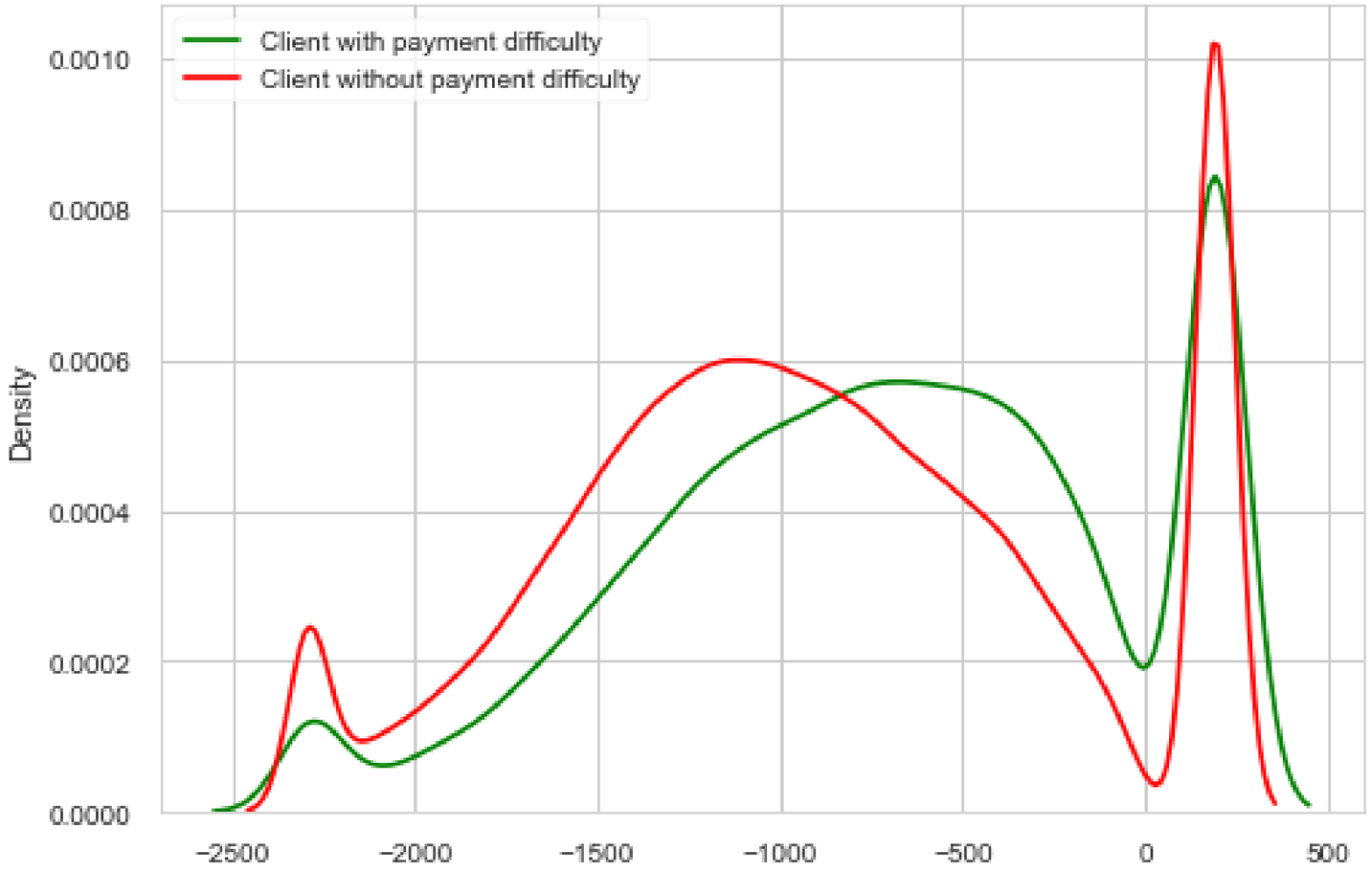
DAYS_CREDIT: How many days before current application did client apply for Credit Bureau credit





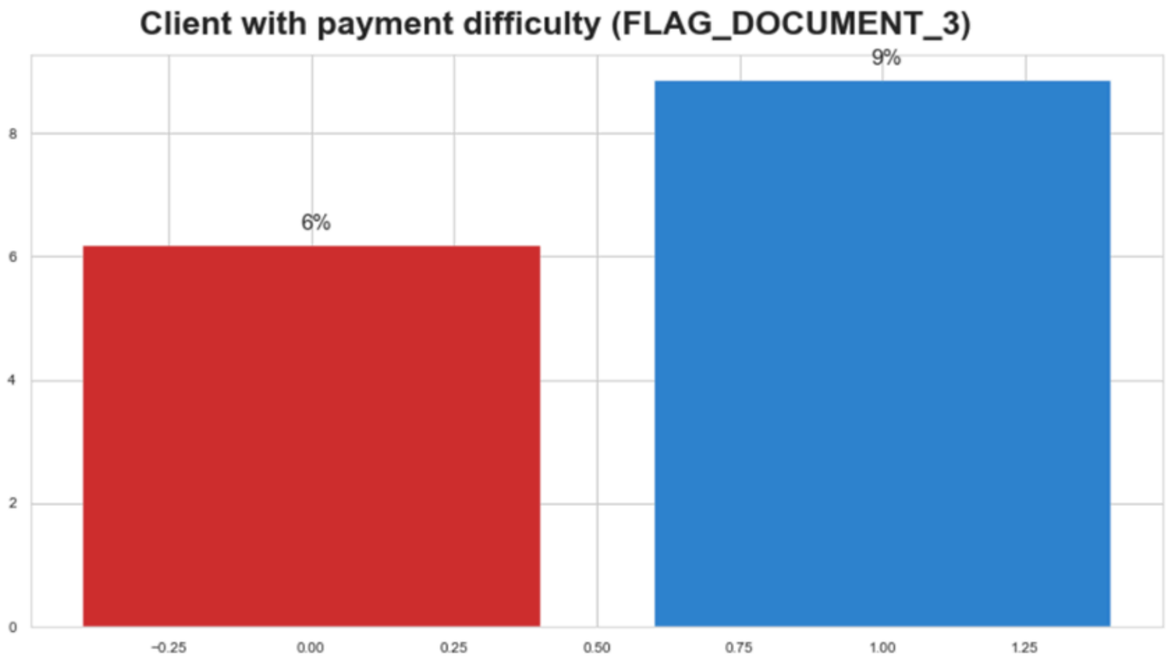
The time to apply for a client's credit before the official registration date usually falls in the range of 1000-1500 days. In general, customers have to wait a long time for credit to be granted, and as long as 2500 days before it is officially granted.

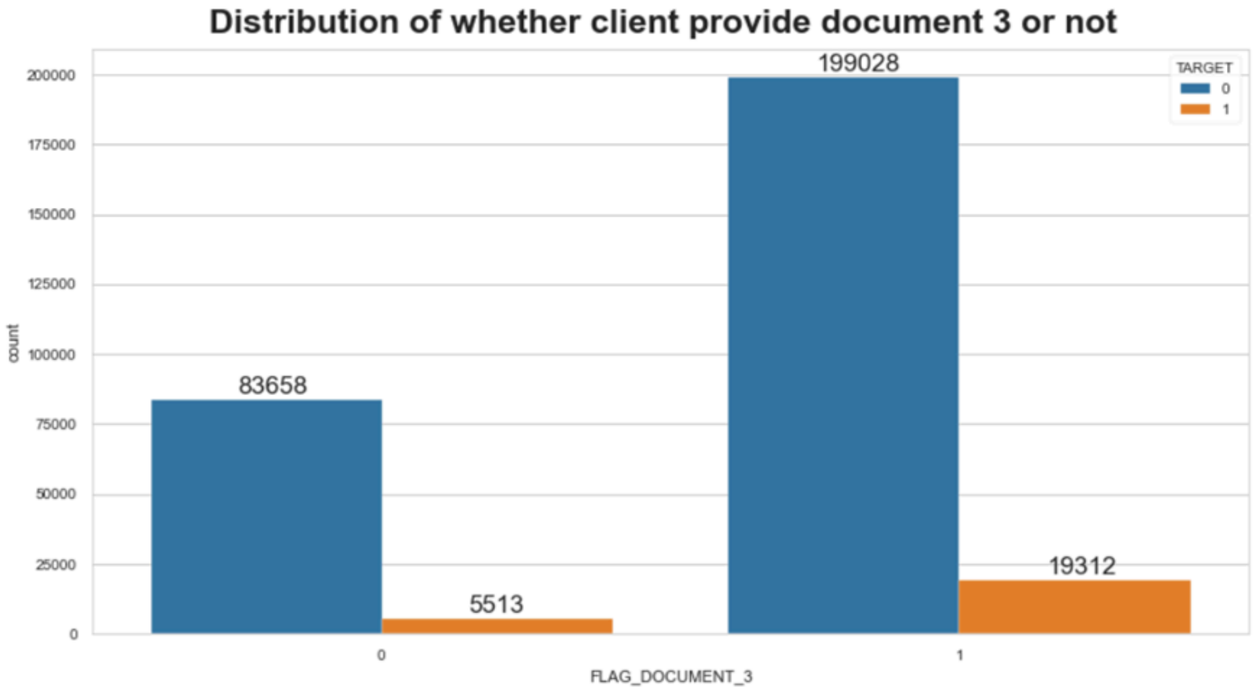
DAYS_CREDIT: How many days before current application did client apply for Credit Bureau credit



However, the highest density of customers applying for credit after formal application is 250 days after the credit is granted.

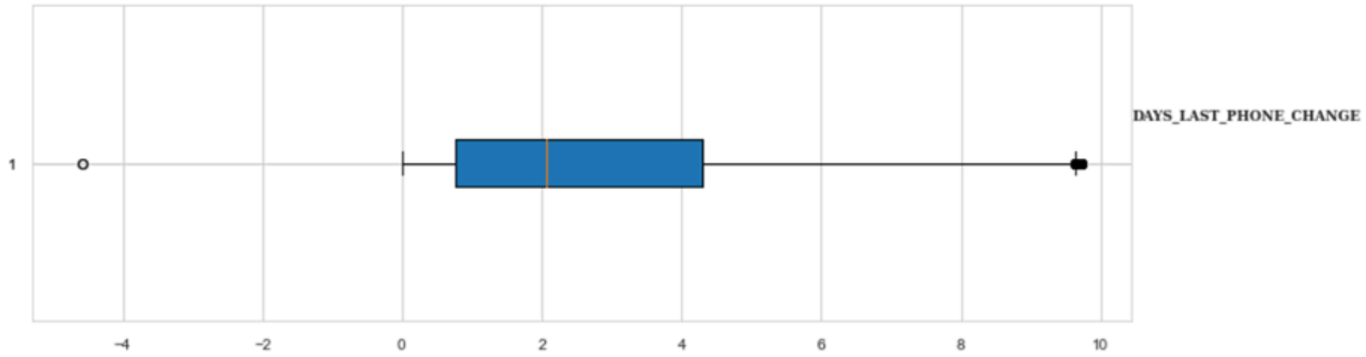
FLAG_DOCUMENT_3: Did client provide document 3





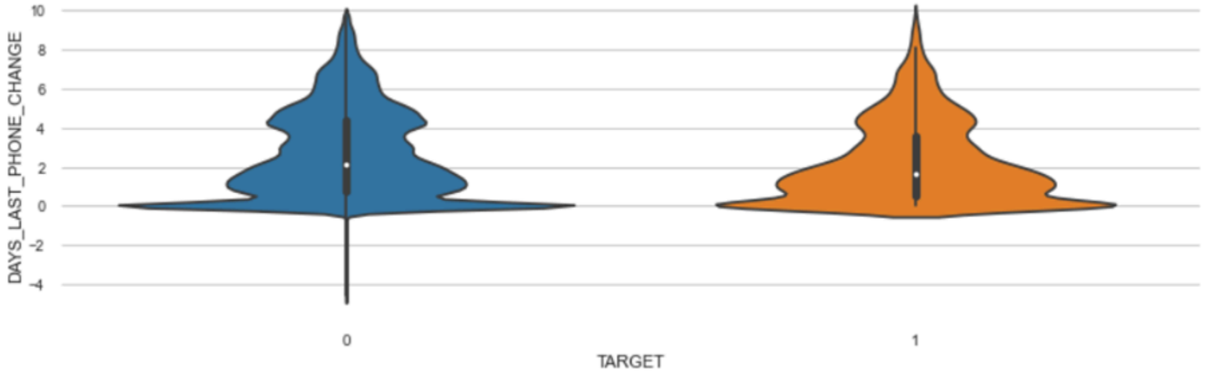
Most clients provided document number 3, about 2/3 of all customers

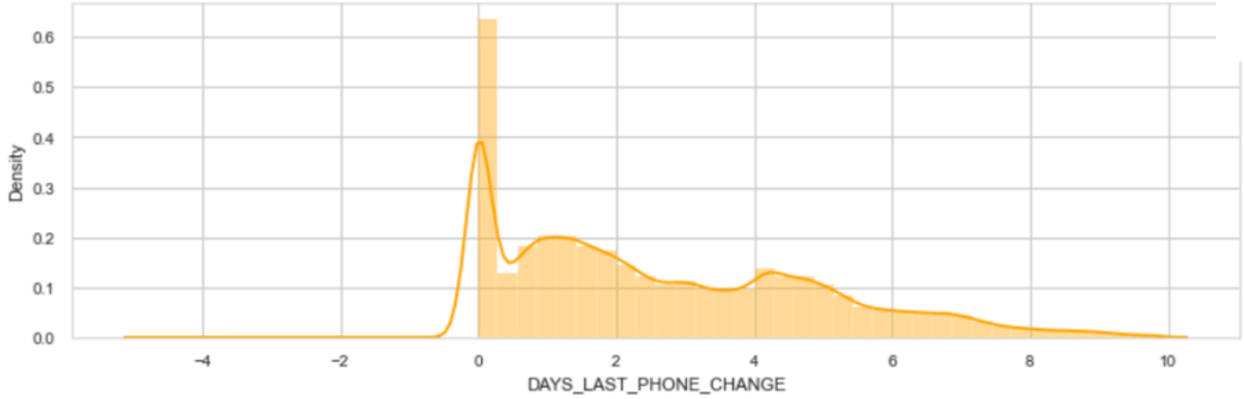
DAYS_LAST_PHONE_CHANGE: How many days before application did client change phone



The trend that the majority of customers will change phones before credit is around half a year (look at the graph and divide it from 0-2).

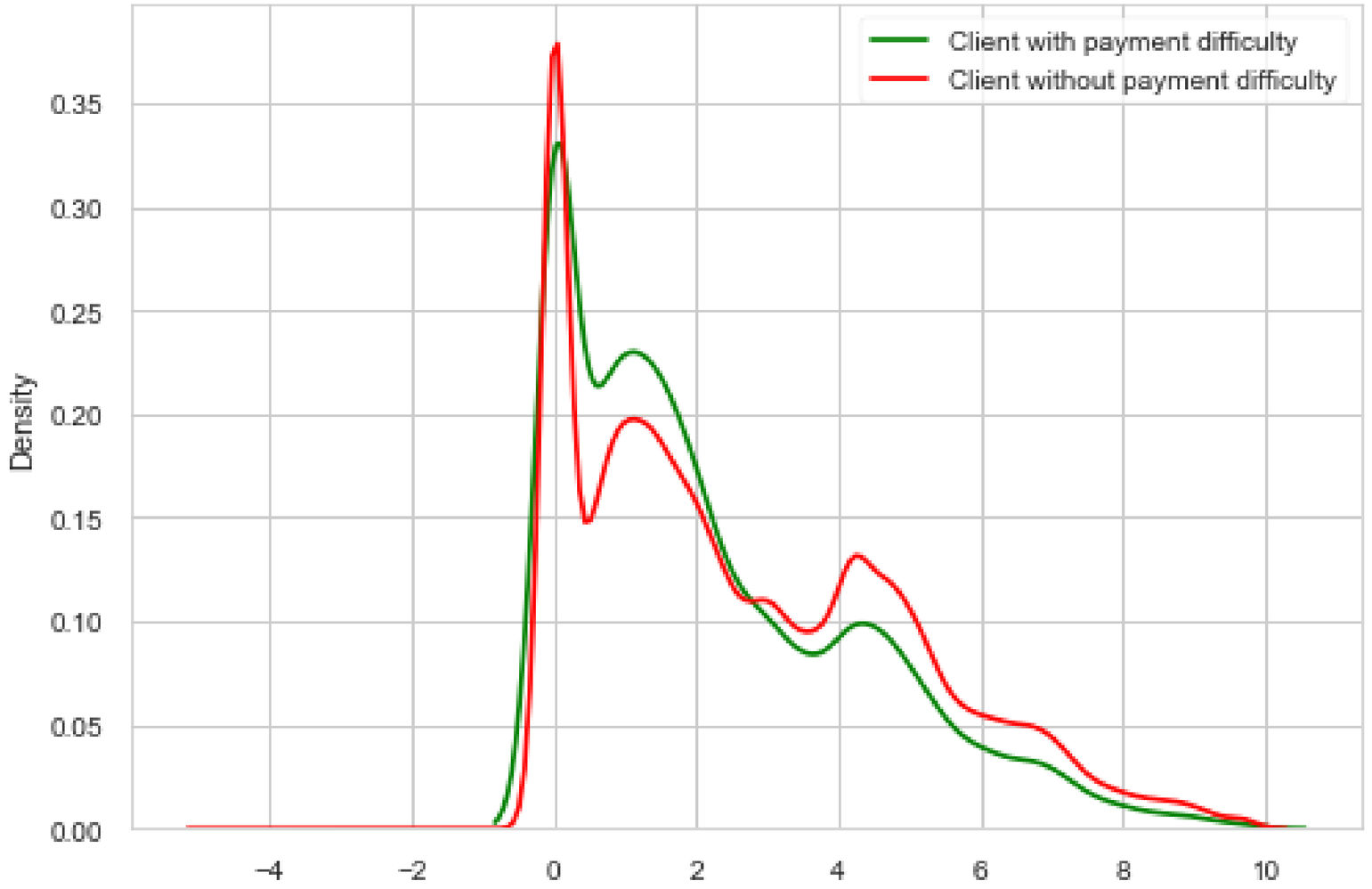
Most customers change phones before signing up and getting credit. This can be understood that before being granted credit, customers have intend that they will repay amount for phones when they have money.





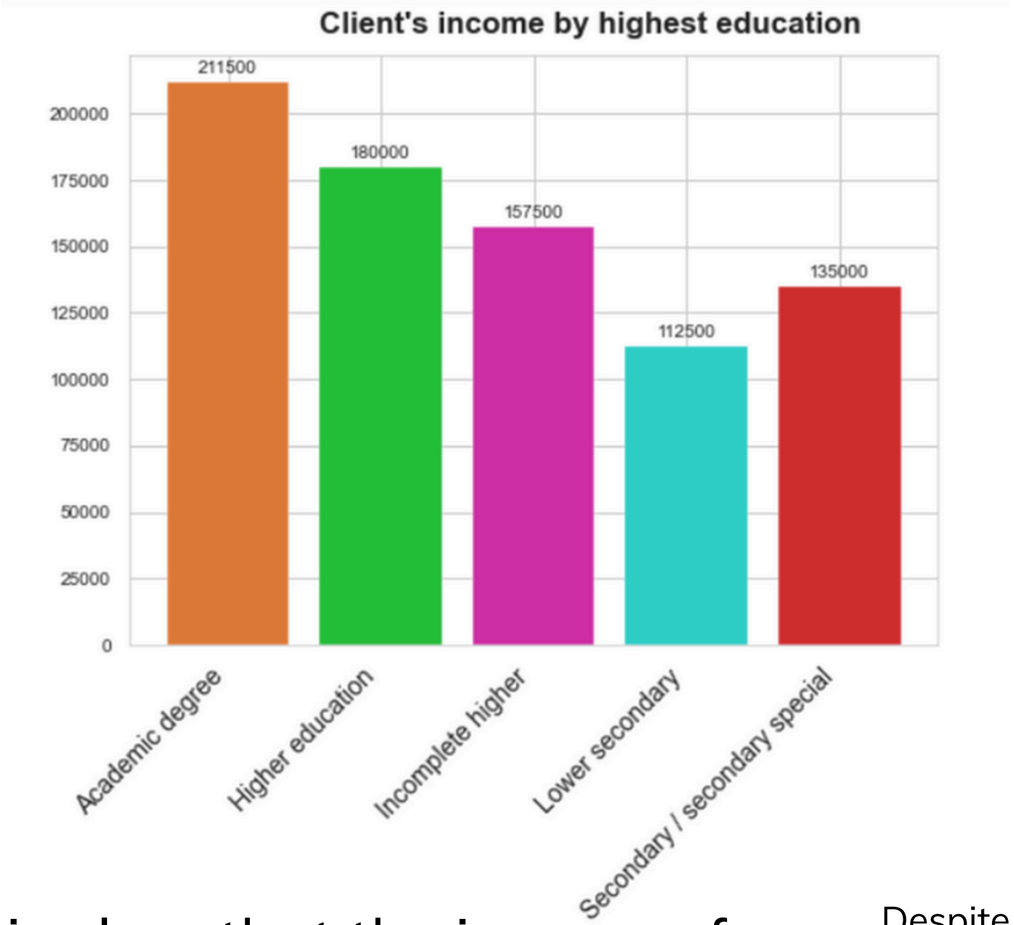
Only a few people change phones after they get credit.

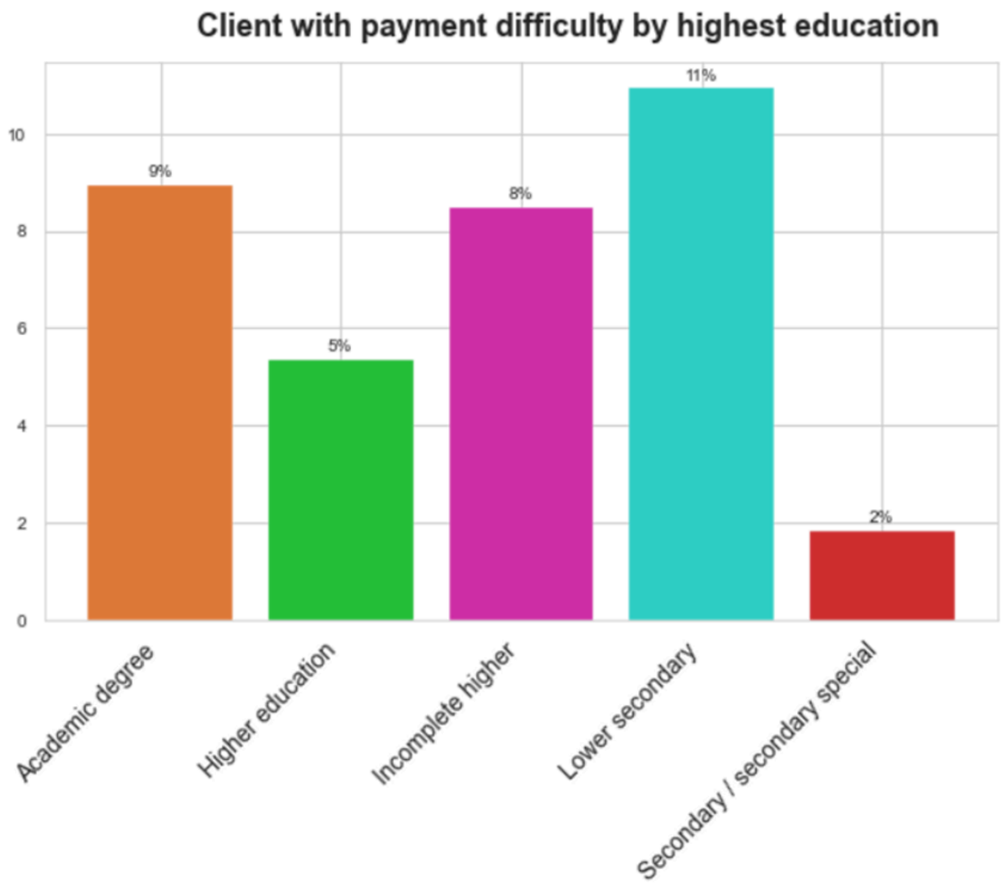
DAYS_LAST_PHONE_CHANGE: How many days before application did client change phone



Let's consider some important features to engineer and get new features for better result

MEDIAN_INCOME_EDU_TYPE: Median income by highest education type

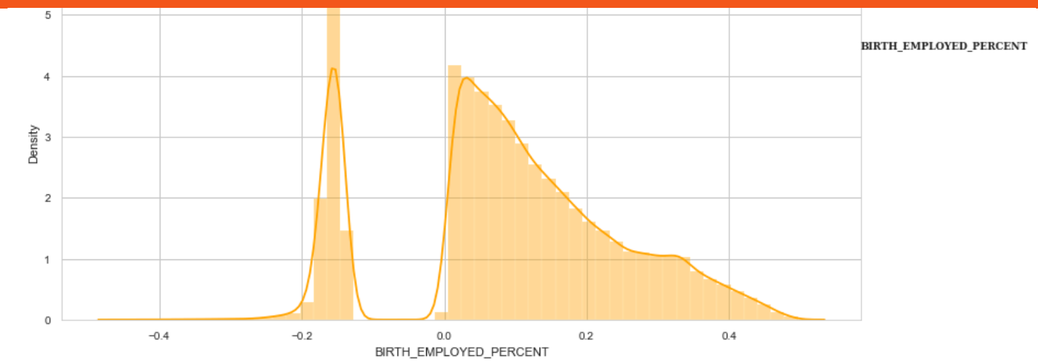


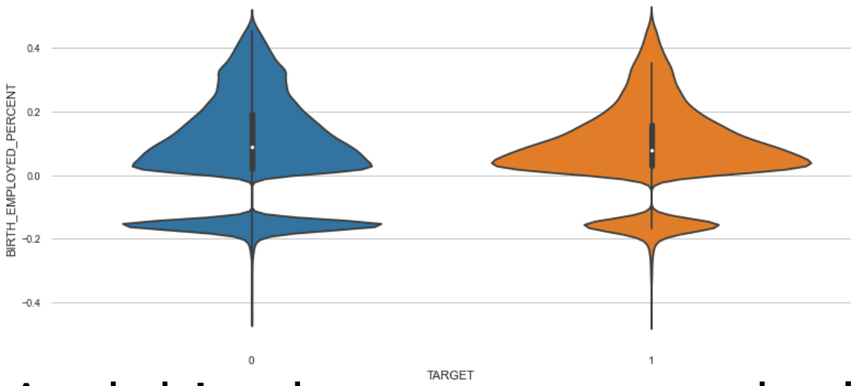


It is clear that the income of customers increases (or higher) when they have higher education level.

Despite having a high level of income based on qualifications, Academic degree's Clients and Higher education's Clients have 9% and 8%, respectively, of which people having difficulty paying, followed by Lower secondary has the most difficult group of people to pay. The reason may be that the job is increasingly demanding high qualifications, so those with an Lower secondary degree will only receive a low income. And so it is difficult for them to repay the debt.

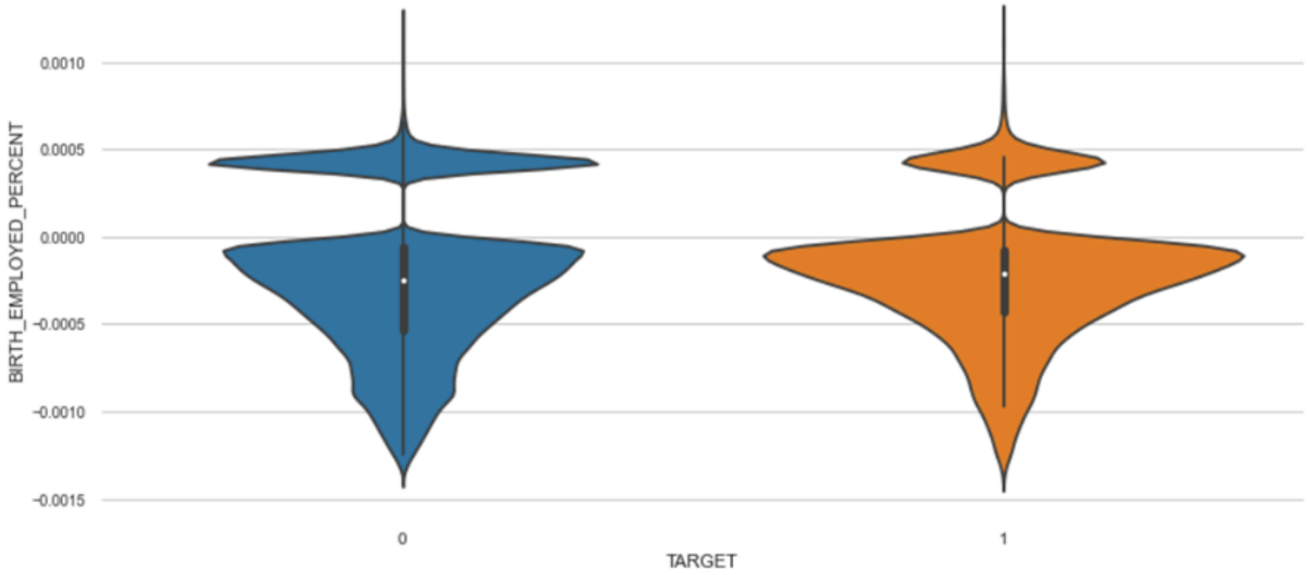
BIRTH_EMPLOYED_PERCENT: Percentage of client's being employed over client's age



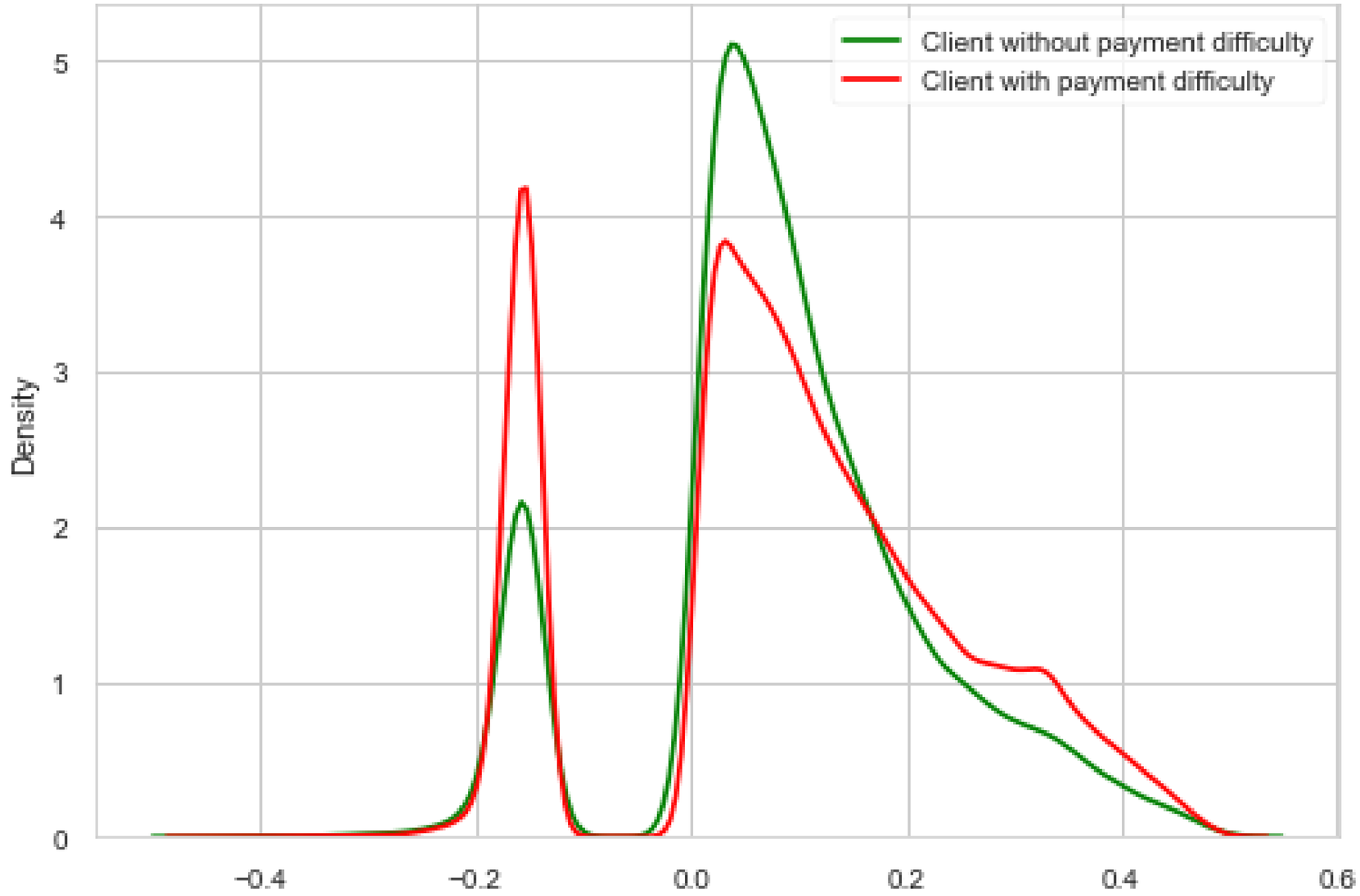


And this phenomenon gradually decreases to nearly 0.5%, ie 0.5% of their life, the job is stable and there is little need to borrow. So few customers have payment problems

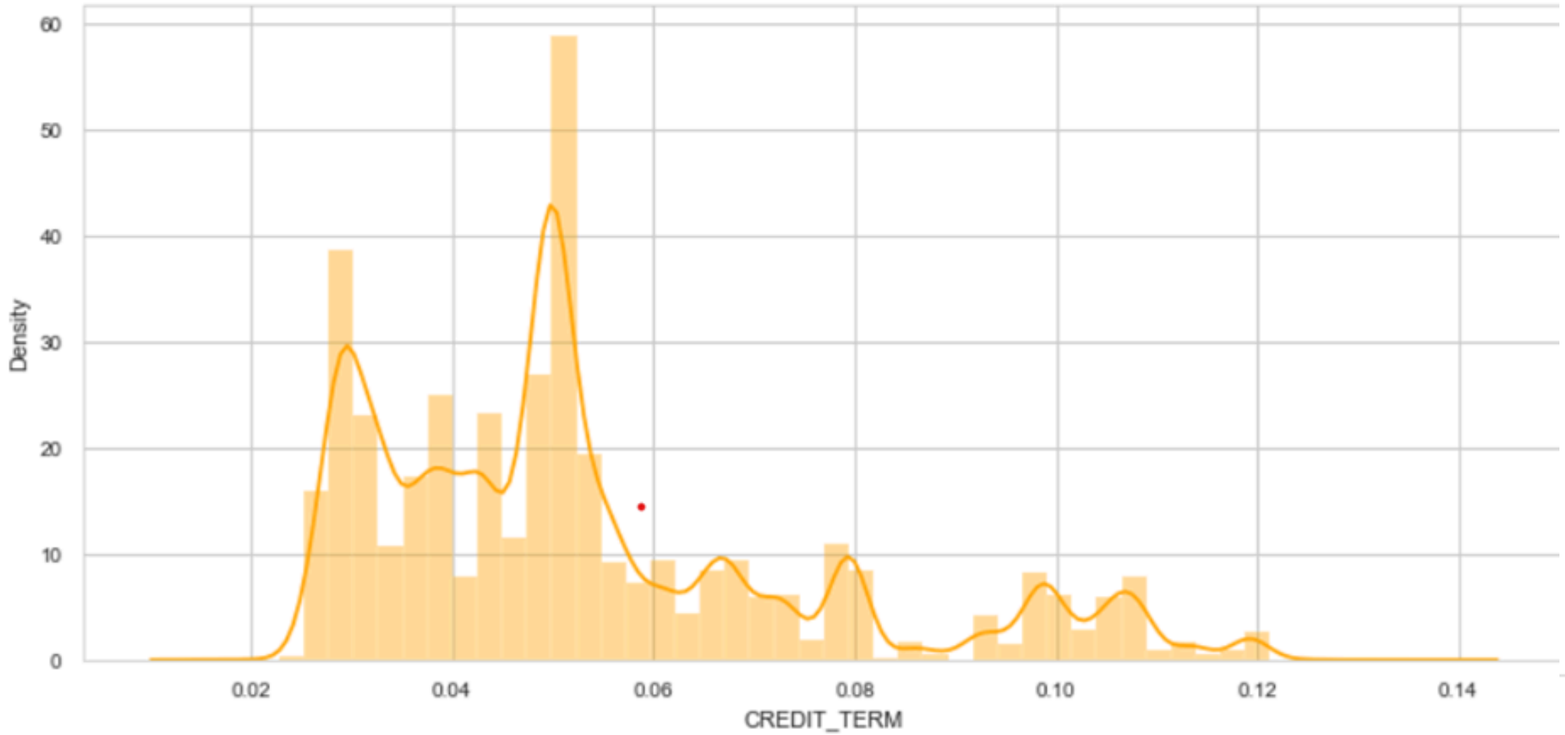
The majority of those who have difficulty paying falls into the 0.1% range at most. It is understandable that these people have just joined the labor force, have low income, but they often borrow, so this phenomenon occurs



BIRTH_EMPLOYED_PERCENT: Percentage of client's being employed over client's age

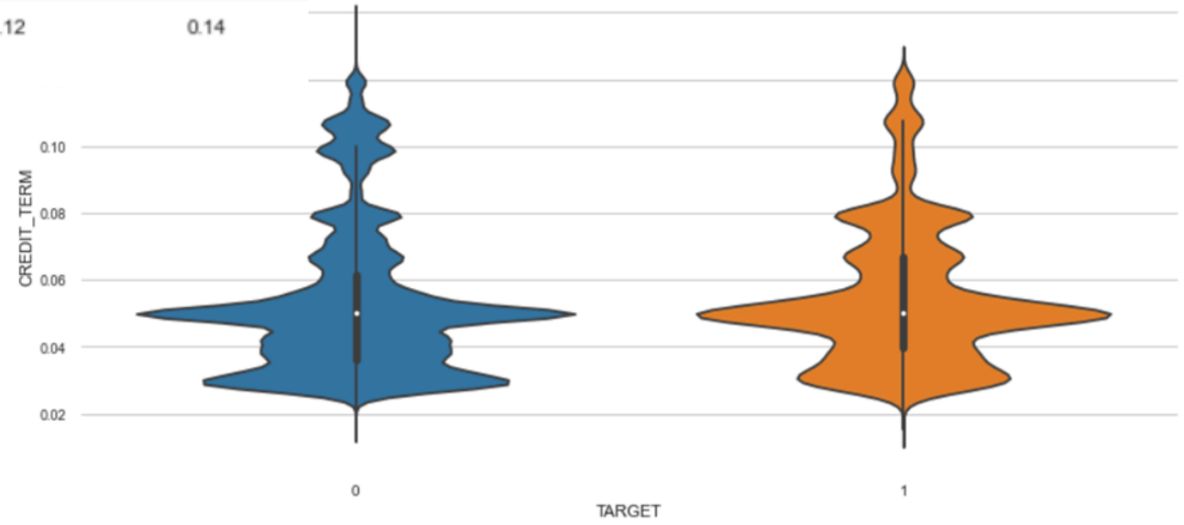


CREDIT_TERM: The length of the payment in months (since the annuity is the monthly amount due

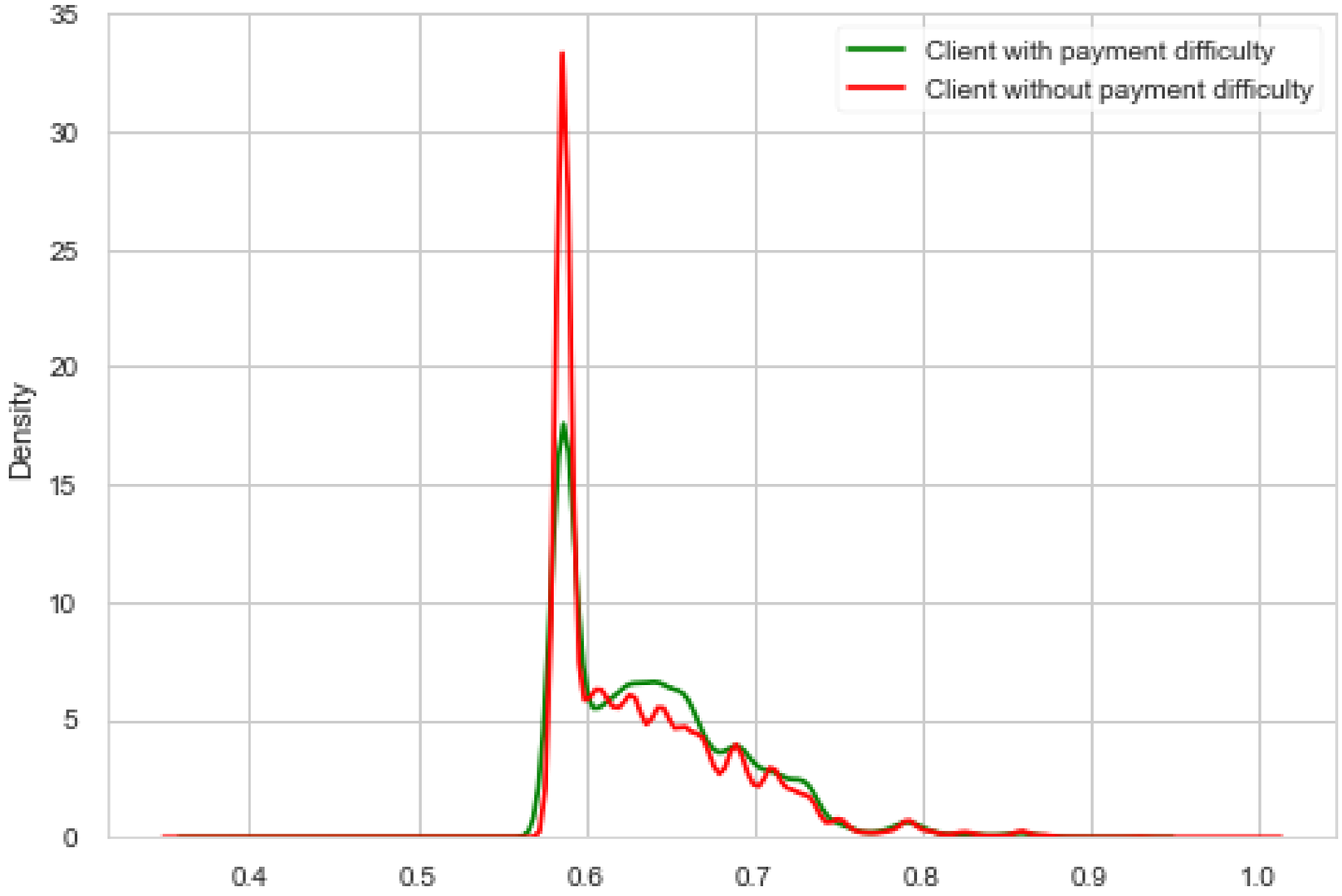


Most clients will buy an annuity that is 5% or 3% higher than their loan. And often customers at level 3% and 5% will have more difficulty paying their debts

And the more the annuity is different from the loan, the less clients have trouble paying. It also reflects that their income is high (i.e. they buy more annuity than the loan) then they will have less problems paying.



CREDIT_TERM: The length of the payment in months (since the annuity is the monthly amount due



CREDIT_GOODS_DIFF: Difference between the credit amount versus the price of the goods the loan paid for

At the 0.6% difference between the credit and the goods price, many people will have problems

However, the higher the disparity, the fewer people have difficulty paying. That is to say, shifting commodity prices (i.e. small spreads) causes them to spend more, and it narrows the value of their loans. The loan will mostly have to pay for the goods. And perhaps when they borrow, they intend to use the loan for other purposes so that they can make a profit and eventually repay the loan. However, high commodity prices have hindered that.

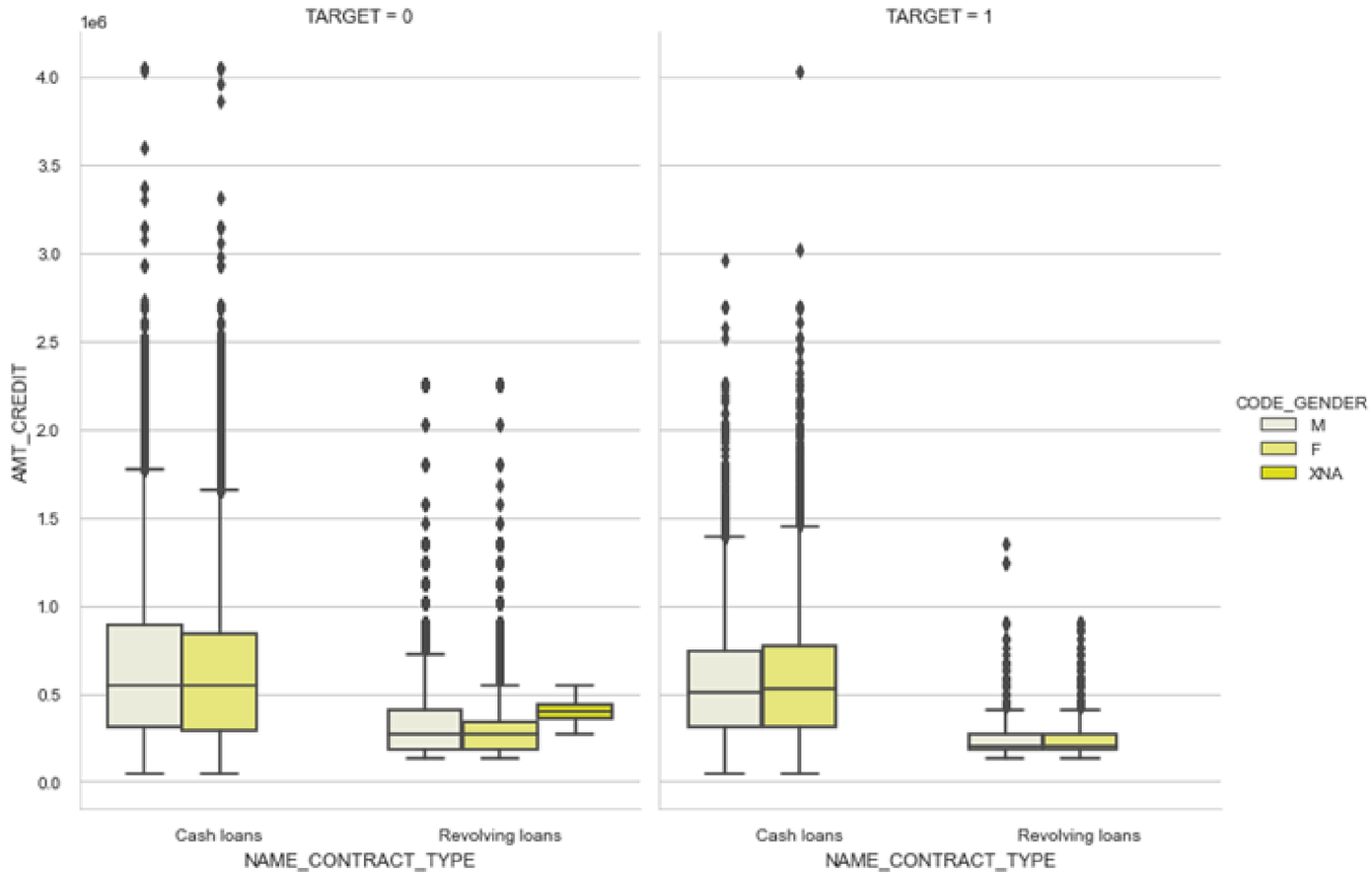
Bivariate Analysis

NAME_CONTRACT_TYPE vs AMT_CREDIT

AMT_CREDIT vs NAME_INCOME_TYPE

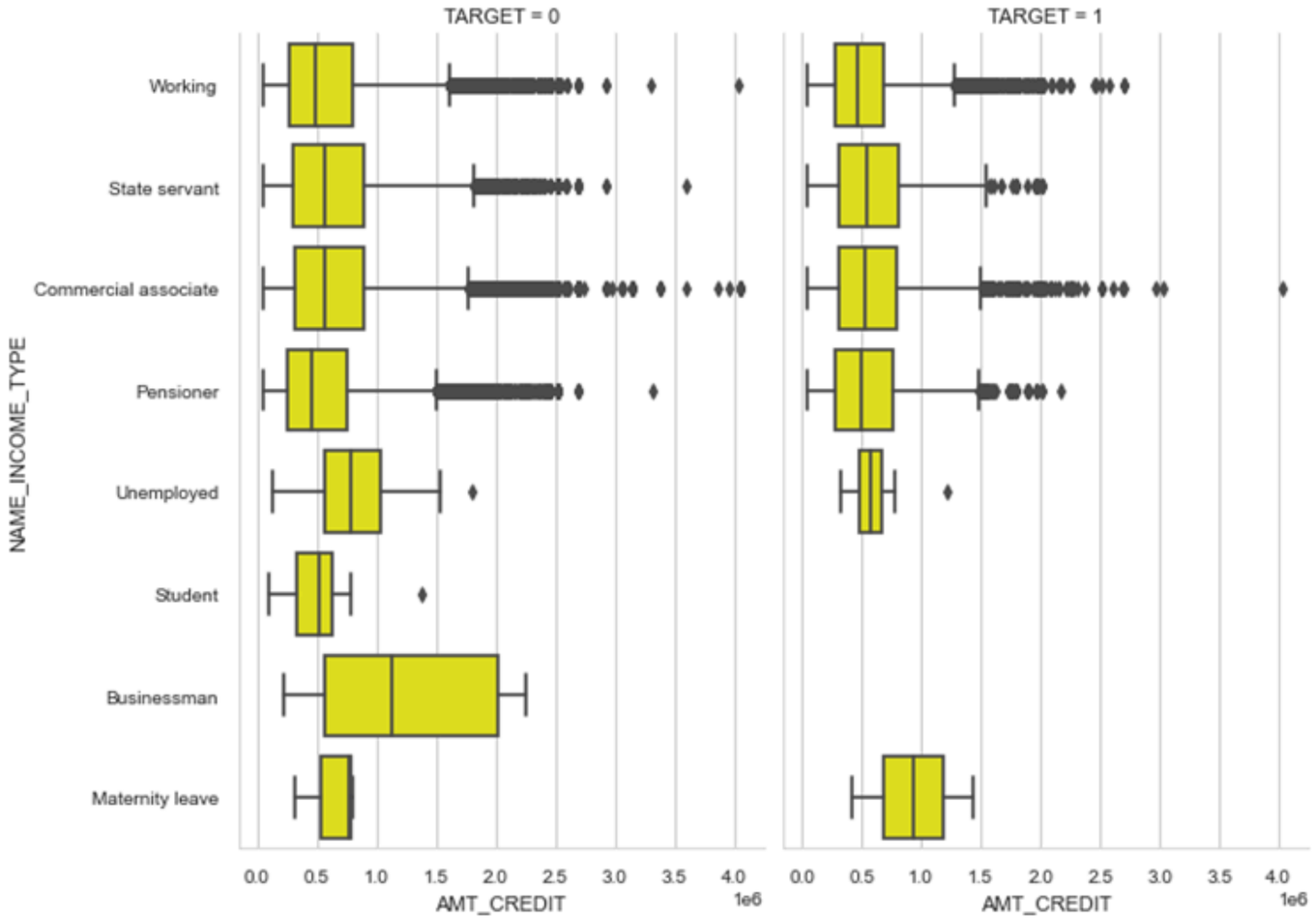
AMT_CREDIT vs AMT_ANNUITY

NAME_CONTRACT_TYPE vs AMT_CREDIT



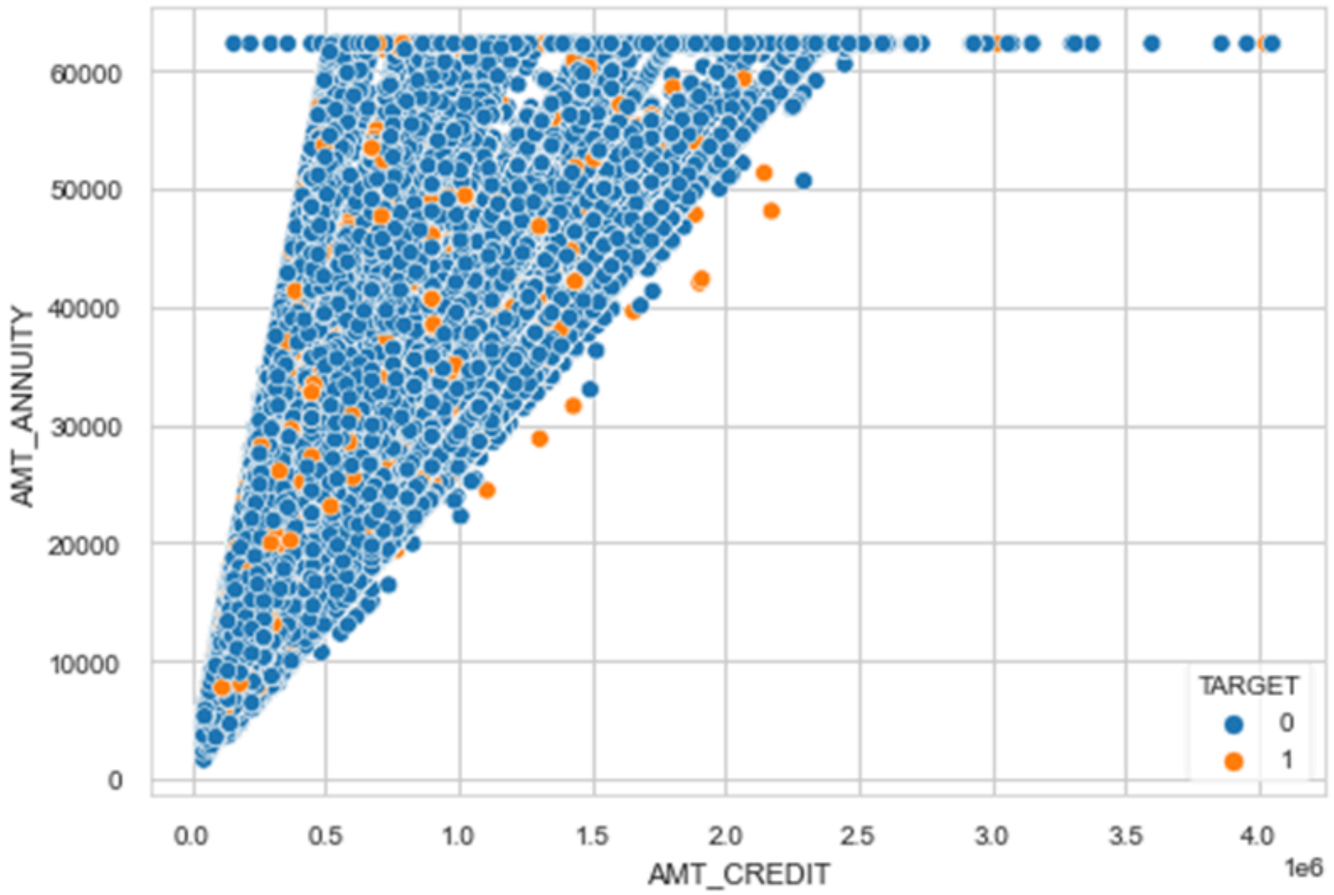
This shows that Men & Women with Cash Loans have higher chances of being deemed capable of loan repayment based on their Credit Amount.

AMT_CREDIT vs NAME_INCOME_TYPE



This shows that applicants with a Higher Value of Credit Amount across various income types have a Higher Likelihood of deemed capable of Loan Repayment, especially in the case of 'Unemployed','Student' and 'Businessmen'.

AMT_CREDIT vs AMT_ANNUITY



This shows that both the Amount Credit and Amount Annuity are directly proportional to each other. If the Credit Amount is high, the Annuity Amount for the same will also be high.

We are able to see an almost Linear graph. However, based on these 2 numeric features, we are not able to carry out Binary Classification with the help of Simple Logistic Regression. This means we need to carry out Feature Engineering on the same.

Thank you for reviewing our project