GROUP 3 - DSEB 62

Home Credit Default Risk Report

- Data Preparation and Visualization

Contents of the report

Part I: Overview of the project

Part II: Dataset description

Part III: Data Preparation

Part IV: Feature Construction

Part V: Data Analysis

Part VI: Features affect TARGET

I. OVERVIEW OF THE PROJECT

- Project goal:
  - *Analyze* default risk of customers in a financial institution - Home Credit
  - *Explore insights* from data
  - *Indicate* important factors influencing fraud
- Dataset: *Kaggle:* Home Credit default risk competition

II. DATASET DESCRIPTION

The dataset contains 10 csv files:

- **2 application files** (application_{train|test}.csv)
- **6 files for reference**
    - Reference from **Bureau:** bureau.csv, bureau_balance.csv
    - Reference from **loan products:** POS_CASH_balance.csv, credit_card_balance.csv, installments_payments.csv
    - Application data of **previous loan:** previous_application.csv
- **1 column description file** (HomeCredit_columns_description.csv)
- **1 sample submission file** (sample_submission.csv)

II. DATASET DESCRIPTION

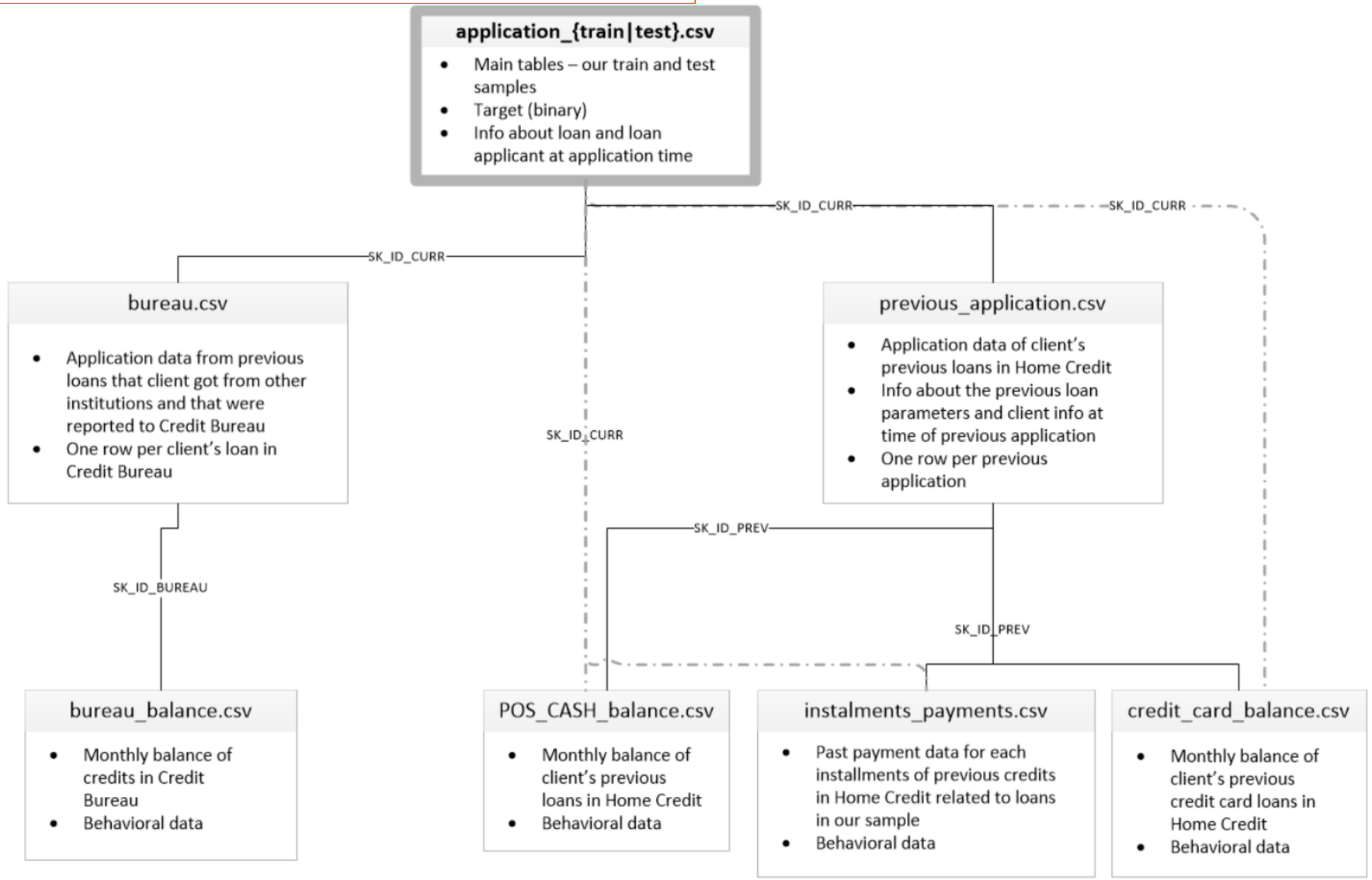


TABLE DESCRIPTION

- Main Tables

application_{train|test}.csv: main tables, broken into two files: **application_train**(with TARGET) and **application_test** (without TARGET). The application train data contains about **308K anonymous clients' with 122 unique features**. **One row represents one loan** in our data sample.

- Bureau
  - *bureau.csv*: All **client's previous credits** provided by other financial institutions that were **reported to Credit Bureau** (for clients who have a loan in our sample).
  - *bureau_balance.csv:* **Monthly balances** of previous credits in Credit Bureau.

TABLE DESCRIPTION

- Loan Products
  - *POS_CASH_balance.csv:* **Monthly balance of previous POS (point of sales) and cash loans** that the applicant had with Home Credit.
  - *credit_card_balance.csv:* **Monthly balance of previous credit cards** that the applicant has with Home Credit.
  - *installments_payments.csv:* **Repayment history** for the previously disbursed credits in Home Credit related to the loans in our sample. One row is equivalent to **one payment of one installment** OR **one installment corresponding to one payment** of one previous Home Credit credit related to loans in our sample.

- Previous Application

- *previous_application.csv:* **All previous applications** for Home Credit loans of clients who have loans in our sample. There is one row for each previous application related to loans in our data sample?

OUR STEPS:

1. Exploring data (EDA)
2. Cleaning columns
3. Combining all tables
4. Preprocessing data
5. Constructing features
6. Analyzing data: Insights
7. Finding features affects TARGET

III. DATA PREPARATION

1. EXPLORING DATA:
a. A FIRST SIGHT TO DATA

- Dataframe.head()
- **Dataframe.info():** a quick overview of the number of records that are filled with values for each column
- **Dataframe.dtypes:** take a look at the data types of the columns
- **Dataframe.unique():** see the distinct values of each attribute

Examine Unique Values of Contract status

Next we can look at the number and values of unique values in categorical column: Contract status

```
1  # CONTRACT STATUS
2  credit_card_balance['NAME_CONTRACT_STATUS'].unique()
```

```
array(['Active', 'Completed', 'Demand', 'Signed', 'Sent proposal',
       'Refused', 'Approved'], dtype=object)
```

1. EXPLORING DATA:
a. A FIRST SIGHT TO DATA

- **Dataframe.shape:** the total number of records and the total number of columns
- **Dataframe.isnull().sum():** the total of missing values in each attribute

```
# Missing values statistics
missing_values = missing_values_table(application_train)
missing_values.head(20)
```

```
Your selected dataframe has 122 columns.
There are 67 columns that have missing values.
```

	Missing Values	% of Total Values
COMMONAREA_MEDI	214865	69.9
COMMONAREA_AVG	214865	69.9
COMMONAREA_MODE	214865	69.9
NONLIVINGAPARTMENTS_MEDI	213514	69.4
NONLIVINGAPARTMENTS_MODE	213514	69.4
NONLIVINGAPARTMENTS_AVG	213514	69.4
FONDKAPREMONT_MODE	210295	68.4

Application_train table

This table is static data for all applications

```
application_train.shape
```

```
(307511, 122)
```

b. SUMMARIZATION

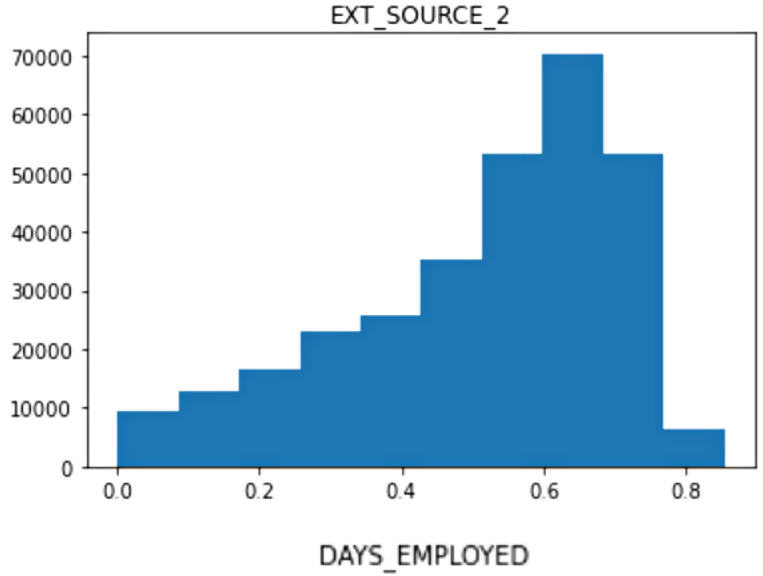
- Dataframe.describe(): summary statistics
- Dataframe.value_counts()

```
1 # Object/categorical columns
2 application_train.describe(include=["object", "category"])
```

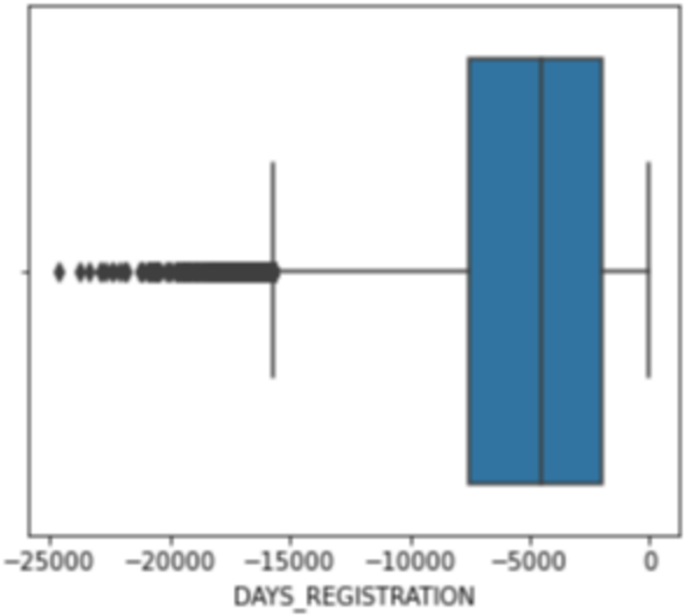
	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	NAME_TYPE_SUITE	NAME_INCOME_TYPE
count	307511	307511	307511	307511	306219	307511
unique	2	3	2	2	7	8
top	Cash loans	F	N	Y	Unaccompanied	Working
freq	278232	202448	202924	213312	248526	158774

VISUALIZATION

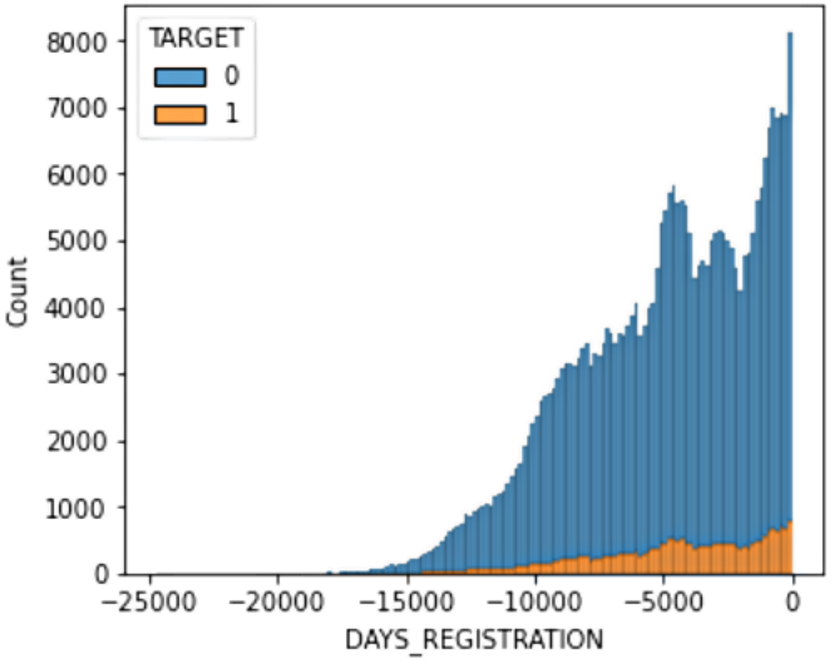
- **Dataframe.hist():** get information about the shape of the distribution and the skewness of the feature



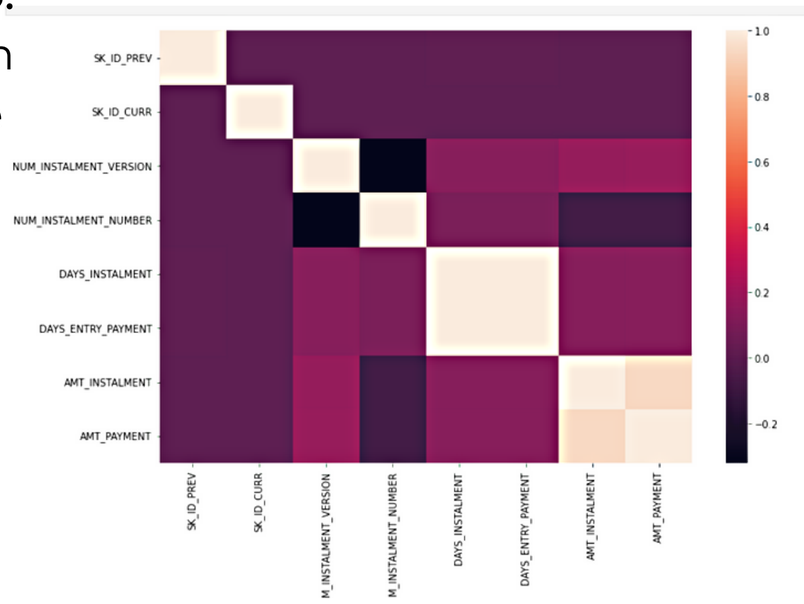
- **sns.boxplot():** check outliers



- **sns.histplot():** plot pairwise relationships between TARGET and other variables



- **sns.heatmap:** correlation table



2. CLEANING COLUMNS

Drop unnecessary/ unsatisfied columns of which:

- Percentage of null > 90%

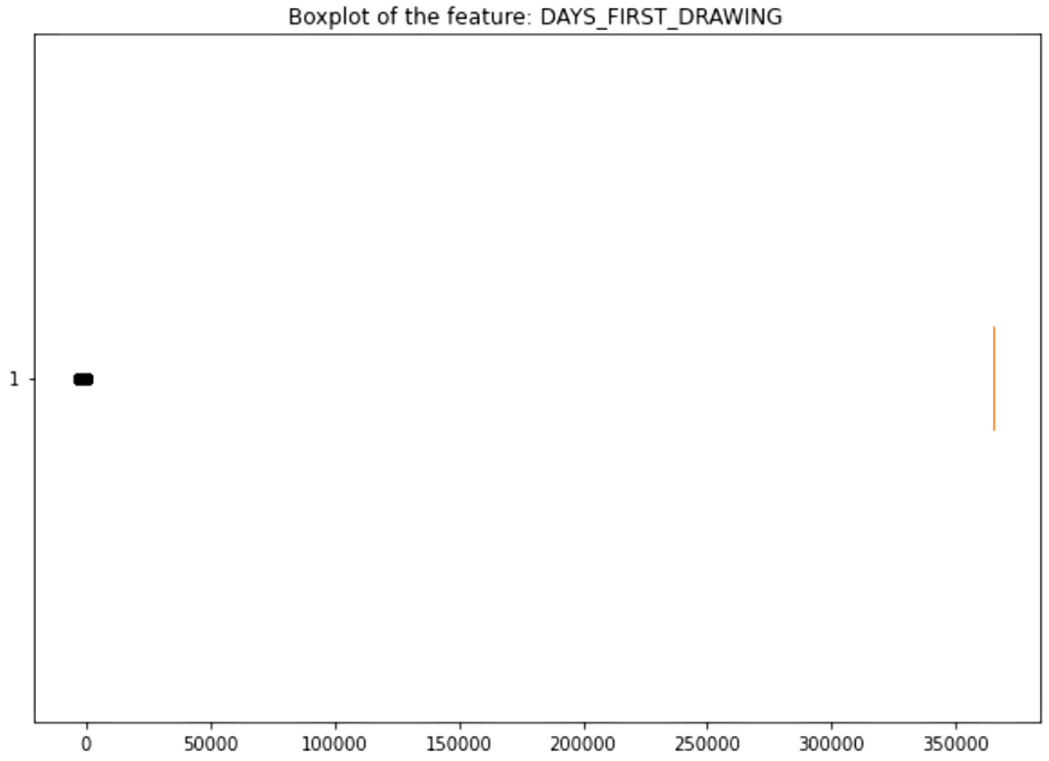
```
null_percent = (previous_application.isnull().sum()/previous_application.shape[0])*100
null_above_90 = pd.DataFrame()
null_above_90['Percentage of null'] = null_percent[null_percent > 90]
null_above_90
```

	Percentage of null
RATE_INTEREST_PRIMARY	99.643698
RATE_INTEREST_PRIVILEGED	99.643698

```
previous_application.drop(null_percent[null_percent > 90].index, inplace=True, axis=1)
```

- Correlation are really high (>0.9)
- Low variance (one value takes up to > 95%)
- Unnecessary for analysis

- High percentage of outliers. For example: *DAYS_FIRST_DRAWING*, day of the first disbursement of the previous application is invalid, because **there are many values above 350000 days** (1000 years)



Columns to drop in each tables:

- bureau.csv + bureau_balance.csv + installments_payments.csv **stay the same**

- {train|test}.csv: **drop 54 columns**
- credit_card_balance: **drop 10 columns**
- POS_CASH_balance.csv: **drop 2 columns**
- previous_application: **12 columns**

Before & After:

for example in *application_train:*

```
print(application_train.shape)
application_train.drop(columns = to_drop_application, inplace = True)
print(application_train.shape)
application_train.head(5)
```

```
(307511, 122)
(307511, 68)
```

3. COMBINING DATA

Combining the entire dataset is done as follows:

1. **First**, combine application_train and application_test, we get a total dataframe named: data

```
1  data = application_train.append(application_test)
2  data.shape
```

```
(356255, 68)
```

2. **Second**, the combined dataset is split into 2 branches: **Left Branch** and **Right Branch**
Regardless of the branch, the general idea is summed up in the following sentence: '**If you want to plug table B into table A with the key K (suggested on the graph), you must generate a key K that exists unique in table B.**'

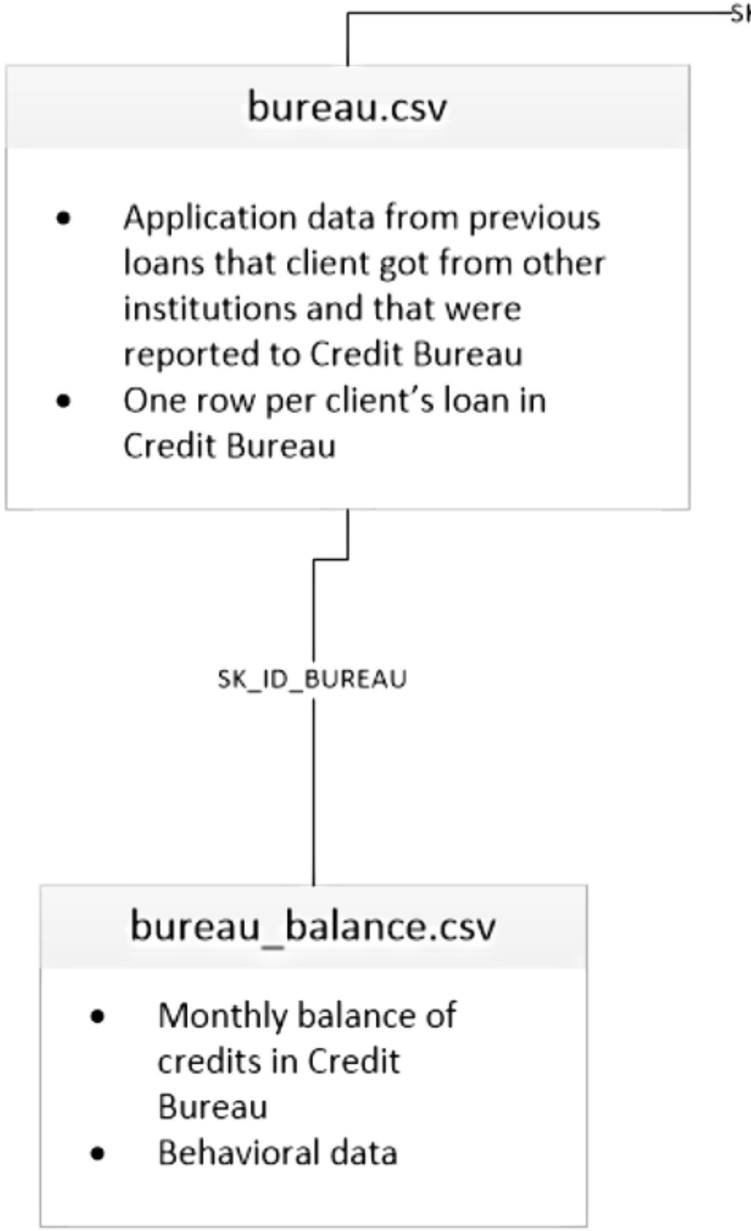
LEFT BRANCH

Left branch:

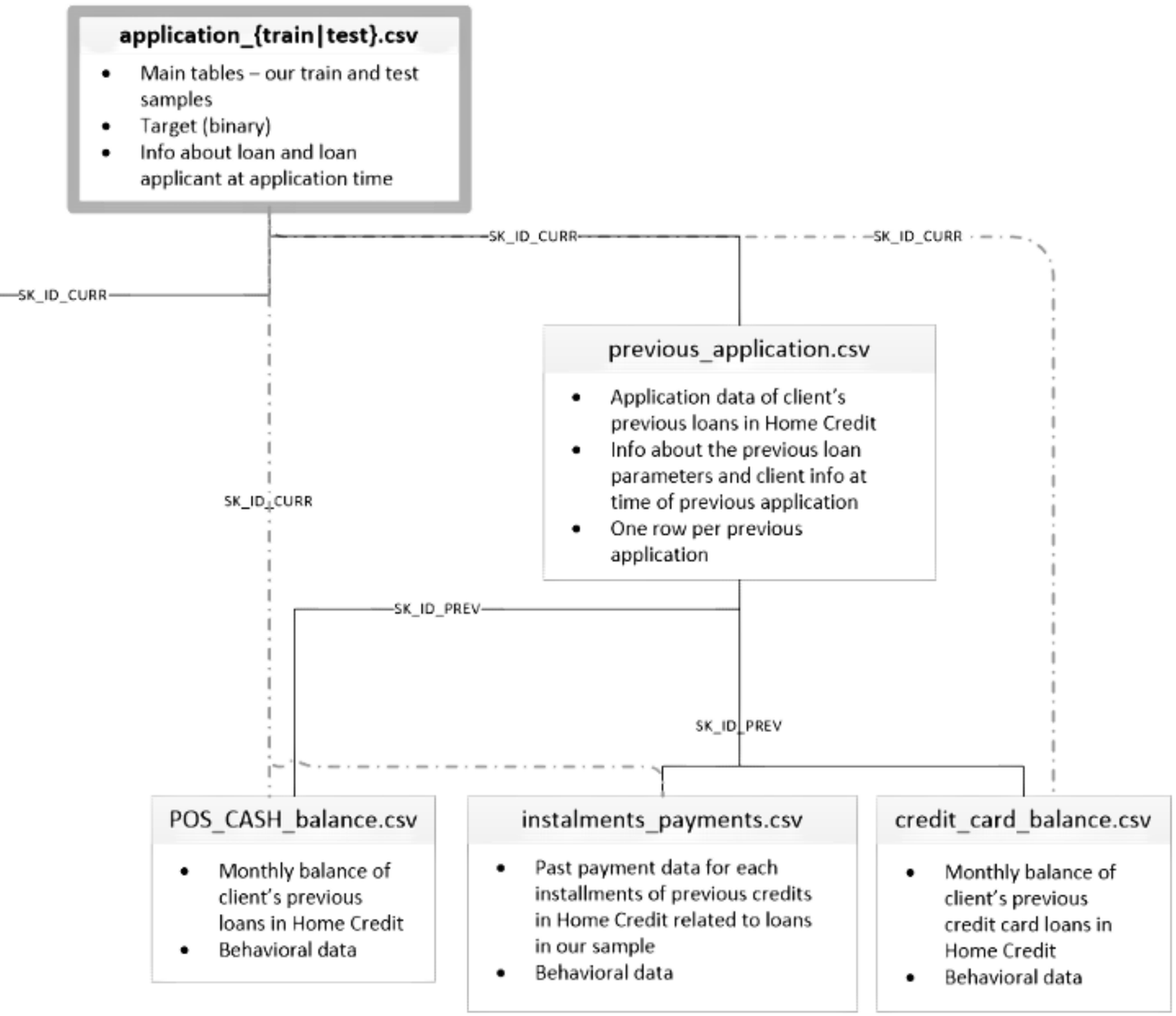
- Groupby table **bureau_balance** according to **SK_ID_BUREAU**, get **mean**, then combine that groupby result **into bureau**

- **Groupby table bureau** by **SK_ID_CURR**, get **mean**, then combine that groupby result **into data**

Add 1 column to count **PREVIOUS_LOANS_COUNT** of each customer **(SK_ID_CURR)** by getting information from groupby from bureau table



RIGHT BRANCH



Right branch:

- Groupby **credit_card_balance** table by **SK_ID_PREV, get mean**, then combine that groupby result **into previous_application**
- Groupby table **instalments_payments** by **SK_ID_PREV**, get **mean**, then combine that groupby result into **previous_application**
- Groupby table **POS_CASH_balance** by **SK_ID_PREV**, get **mean,** then combine that groupby result into **previous_application**
- Groupby **previous_application** table by **SK_ID_CURR**, get **mean**, then combine that groupby result into **data**

AFTER COMBINING

- Left branch:

```
1 data.shape
```

```
(356255, 69)
```

```
1 data = data.merge(bureau_mean_values, on = 'SK_ID_CURR', how = 'left')
```

```
1 data.shape
```

```
(356255, 82)
```

So here we've created 13 new features and added them to our train/test dataset called 'data'

- Right branch:

```
1 print('data shape', data.shape)
2 print('previous applications statistics shape', prev_appl_mean.shape)
```

```
data shape (356255, 83)
previous applications statistics shape (338857, 30)
```

```
1 data = data.merge(prev_appl_mean, on = 'SK_ID_CURR', how = 'left')
```

```
1 print('data shape', data.shape)
```

```
data shape (356255, 112)
```

As we can see, this last sprint over previous applications added 32 new features to our statistics

NOTES:

Note 1:

- This way of combining the dataset will lose the categorical columns in the sub-tables (different from the application_train/test table).

Note 2: Prefixes in columns will help us identify which table the column comes from:

- PREV_BUR_MEAN_: comes from bureau
- PREV_BUR_MEAN_BUR_BAL_MEAN_: comes from bureau_balance
- PREV_APPL_MEAN_: comes from previous_application
- PREV_APPL_MEAN_CARD_MEAN_: comes from credit_card_balance
- PREV_APPL_MEAN_INSTALL_MEAN_: comes from instalments_payments
- PREV_APPL_MEAN_POS_MEAN_: comes from POS_CASH_balance

4. PREPROCESSING

- Split into train and test set

- Filling missing values
  - Fill missing values in 'train' dataset
  - Fill missing values in 'test' dataset

- Outlier Identification and Removal
  - Using boxplot to see outliers of all features that are numerical
  - Drop outliers with Q5%, Q95%

4. PREPROCESSING

Split into train and test set

- Perform split according to IDs in initial train and test datasets

```
train = data[data['SK_ID_CURR'].isin(application_train.SK_ID_CURR)]
test = data[data.SK_ID_CURR.isin(application_test.SK_ID_CURR)]
test.drop('TARGET', axis = 1, inplace = True)
```

```
Initial train set (307511, 68)
Initial test set (48744, 67)
Training Features shape with categorical columns:  (307511, 112)
Testing Features shape with categorical columns:  (48744, 111)
```

- You can see the number of lines are equal, but we will check more to be sure 10001 with 10005 in the test set, and now the train set won't have any null TARGET values

```
test[test["SK_ID_CURR"] == 100001]
```

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FL
307511	100001	Cash loans	F	N	

```
test[test["SK_ID_CURR"] == 100005]
```

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OW
2	100005	Cash loans	M	N	

```
train[train["SK_ID_CURR"] == 100001]
```

SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR

```
train[train["SK_ID_CURR"] == 100005]
```

SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR

4. PREPROCESSING

Filling missing values

Fill missing values in 'train' dataset

- Fill missing values in train data with 2 types 'int64', "float64" by mean of each column
- Fill missing values in train data with "object" type by value 'Unknown'

Fill missing values in 'test' dataset

- Similar as in 'train' set

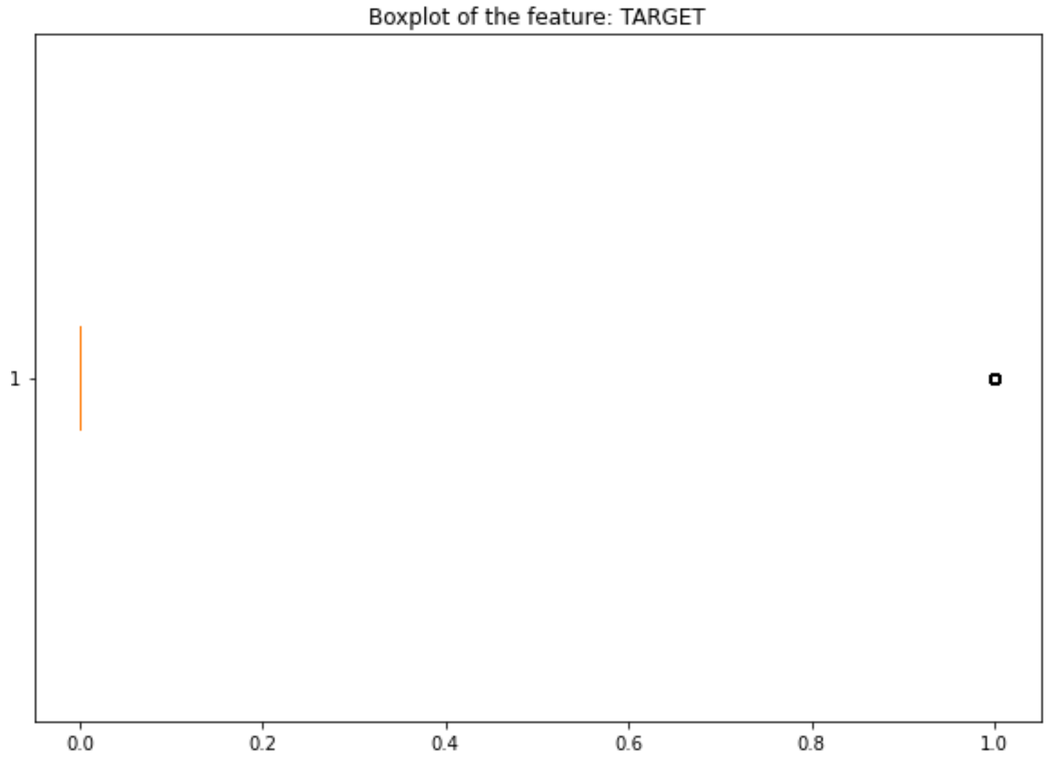
Fill missing values in train dataset

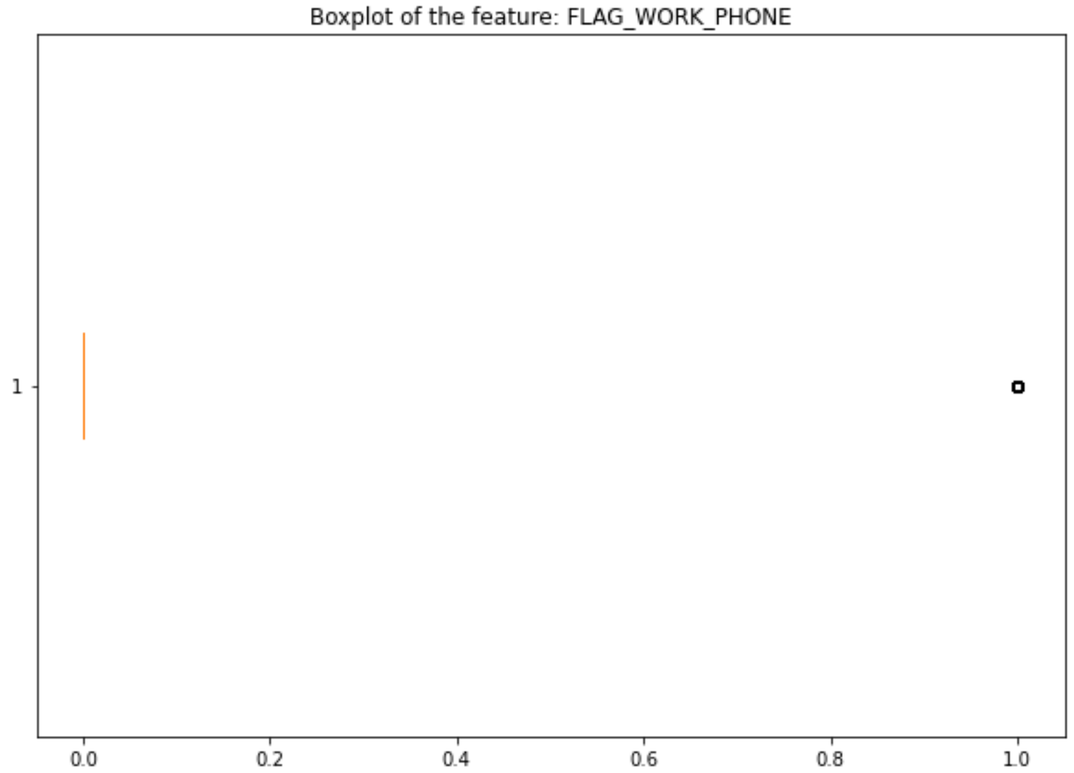
```
# Fill missing values in train data with 2 types 'int64', "float64" by mean of each column
train_type_int_float = train.select_dtypes(include=['int64', "float64"])
train.fillna(train_type_int_float.mean(), inplace=True)
```

```
# Fill missing values in train data with "object" type by value 'Unknown'
list_obj = train.select_dtypes(include=['object']).columns.to_list()
temp = train[list_obj].fillna('Unknown')
train[list_obj] = temp
```

4. PREPROCESSING

Outlier Identification and Removal





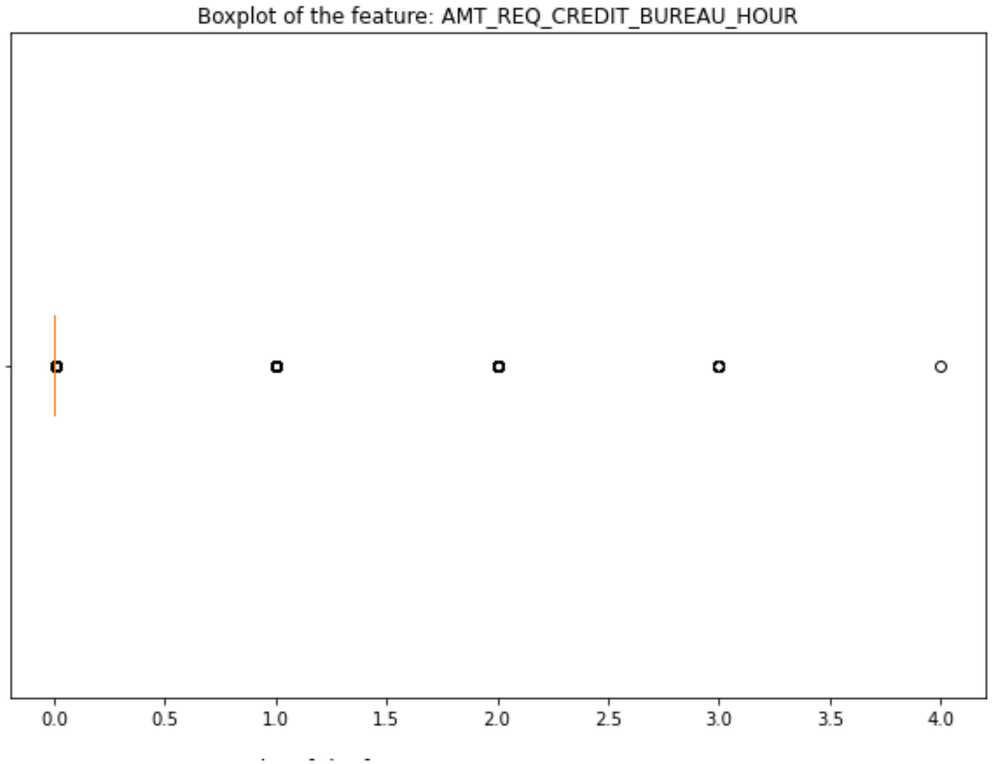
We can see some insights:

- Some columns such as 'TARGET', 'FLAG_WORK_PHONE', etc. just have values '0, 1', so we will remove these columns from the variable 'numeric_col_train', because it is not necessary to drop outliers

=> We do not have to drop outliers of these features

4. PREPROCESSING

Outlier Identification and Removal



We can see some insights:

- 'AMT_REQ_CREDIT_BUREAU_HOUR' have 5 kinds of duplicated values, so we will remove these columns from variable 'numeric_col_train'
- 'SK_ID_CURR' column doesn't need to drop outliers, so we will remove these columns from variable 'numeric_col_train'

=> We do not have to drop outliers of these features

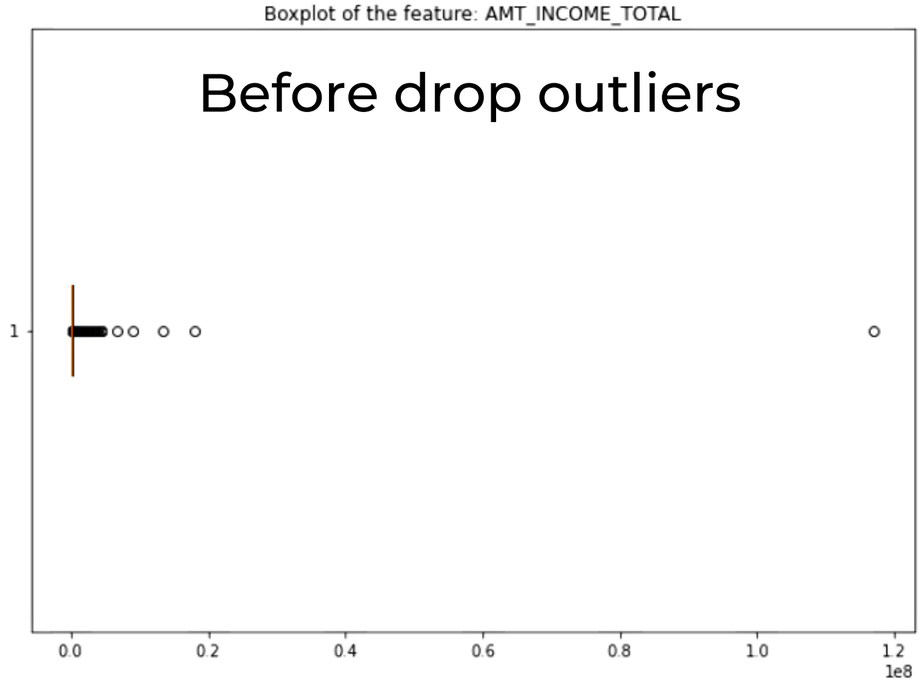
4. PREPROCESSING Outlier Identification and Removal

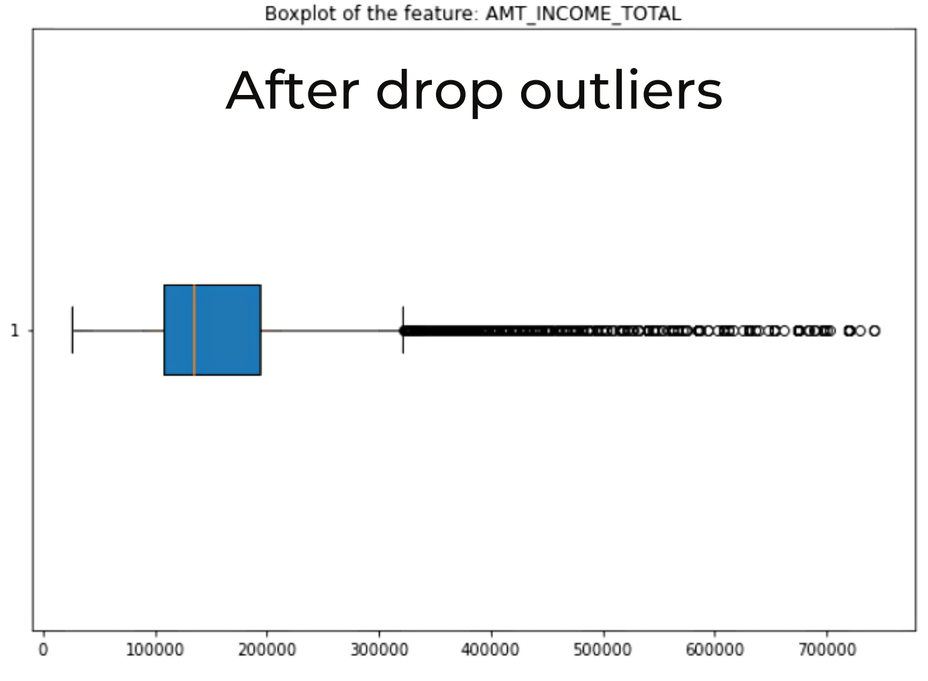
- Remove these columns from variable 'numeric_col_train' , new variable named: ‘final_list’ (these features do not have to drop outliers)

```
# Remove these columns from variable 'numeric_col_train'
list_to_remove = ["SK_ID_CURR", 'TARGET', 'CNT_CHILDREN', 'FLAG_WORK_PHONE', 'REGION_RATING_CLIENT_W_CITY',
                                        'REG_REGION_NOT_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_
                                        'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'FLAG_DOCUMENT_6',
                                        'FLAG_DOCUMENT_8','AMT_REQ_CREDIT_BUREAU_HOUR']

final_list = list(set(numeric_col_train) - set(list_to_remove))
```

- Drop outliers with **Q(0.05), Q(0.95)** of all features from variable ‘final_list’)





4. PREPROCESSING

Output of train and test dataset after preprocessing data

- Train dataset after preprocessing data

Train dataset after preprocessing data

```
1 train.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY
0	100002	1.0	Cash loans	M	N	Y
2	100004	0.0	Revolving loans	M	Y	Y
4	100007	0.0	Cash loans	M	N	Y
5	100008	0.0	Cash loans	M	N	Y
6	100009	0.0	Cash loans	F	Y	Y

- Test dataset after preprocessing data

Test dataset after preprocessing data

```
1 test.head()
```

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY
307511	100001	Cash loans	F	N	Y
307512	100005	Cash loans	M	N	Y
307513	100013	Cash loans	M	Y	Y
307514	100028	Cash loans	F	N	Y
307515	100038	Cash loans	M	Y	N

IV. FEATURE CONSTRUCTION

IV. FEATURE CONSTRUCTION

Create more new features based on methods:

01	**Domain Knowledge Features**
02	**Polynomial features**
03	**Other type features**

1. Domain Knowledge Features

Maybe it's not entirely correct to call this "domain knowledge" because we're not a credit expert, but perhaps we could call this *"attempts at applying limited financial knowledge"*.

- INCOME_CREDIT_PERCENT: the percentage of the income relative to a client's credit amount
- ANNUITY_INCOME_PERCENT: the percentage of the loan annuity relative to a client's income
- CREDIT_TERM: the length of the payment in months (since the annuity is the monthly amount due)
- INCOME_PER_PERSON: Income per person in a family
- INCOME_PER_PERSON: Income per person in a family
- CNT_ADULT_FAM_MEMBER: number of adult members in a family
- RATIO_CHILDREN_TO_ADULTS: ratio of the children - adult
- RATIO_AMT_CREDIT_TO_CNT_FAM_MEMBERS: the credit loan per person in a family
- RATIO_AMT_CREDIT_TO_CNT_ADULT_FAM_MEMBER: the credit loan per adult people in a family
- RATIO_AMT_CREDIT_TO_CNT_ADULT_FAM_MEMBER: the credit loan per adult people in a family
- AMT_INCOME_TOTAL_PER_ADULT_FAM_MEMBER: the income per adult people in a family
- ...

1. Domain Knowledge Features

```python
# INCOME_CREDIT_PERCENT: the percentage of the income relative to a client's credit amount
train_domain['INCOME_CREDIT_PERCENT'] = train_domain['AMT_INCOME_TOTAL']  / train_domain['AMT_CREDIT']

# ANNUITY_INCOME_PERCENT: the percentage of the loan annuity relative to a client's income
train_domain['ANNUITY_INCOME_PERCENT'] = train_domain['AMT_ANNUITY'] / train_domain['AMT_INCOME_TOTAL']

# CREDIT_TERM: the length of the payment in months (since the annuity is the monthly amount due
train_domain['CREDIT_TERM'] = train_domain['AMT_ANNUITY'] / train_domain['AMT_CREDIT']


# INCOME_PER_PERSON: Income per person in a family
train_domain['INCOME_PER_PERSON'] = train_domain['AMT_INCOME_TOTAL'] / train_domain['CNT_FAM_MEMBERS']

# CNT_ADULT_FAM_MEMBER: number of adult members in a family
train_domain['CNT_ADULT_FAM_MEMBER'] = train_domain['CNT_FAM_MEMBERS'] - train_domain['CNT_CHILDREN']
```

```python
# Previous

# PREV_APPL_MEAN_DIFF_AMT_DOWN_PAYMENT_AMT_ANNUITY:
# The average balance after paying down payment and receving annuity of previous application
# If positive: have balance
# If negative: do not have balance
train['PREV_APPL_MEAN_DIFF_AMT_DOWN_PAYMENT_AMT_ANNUITY'] = (train['PREV_APPL_MEAN_AMT_ANNUITY']
                                                             - train['PREV_APPL_MEAN_AMT_DOWN_PAYMENT'])

# Credit_card
# PREV_APPL_MEAN_CARD_MEAN_PCTG_RECEIVABLE_TOTAL_CURRENT:
# The average amount credit that the client hasn't paid during the month in total on the previous credit
train['PREV_APPL_MEAN_CARD_MEAN_PCTG_RECEIVABLE_TOTAL_CURRENT'] = (train['PREV_APPL_MEAN_CARD_MEAN_AMT_TOTAL_RECEIVABLE']
                                                             - train['PREV_APPL_MEAN_CARD_MEAN_AMT_PAYMENT_TOTAL_CURRENT'])

```

2. Polynomial features

We make features that are powers of existing features as well as interaction terms between existing features:

- Based on EXT_SOURCES features: calculate mean, max, sum, min, median:

EXT_SOURCE_mean, EXT_SOURCES_MAX, EXT_SOURCES_SUM, EXT_SOURCES_MIN, EXT_SOURCES_MEDIAN

EXT_SOURCE

```
# Based on EXT_SOURCES features: calculate mean, max, sum, min, median
train['EXT_SOURCE_mean'] = train[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']].mean(axis = 1)
train['EXT_SOURCES_MAX'] = train[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']].max(axis=1)
train['EXT_SOURCES_SUM'] = train[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']].sum(axis=1)
train['EXT_SOURCES_MIN'] = train[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']].min(axis=1)
train['EXT_SOURCES_MEDIAN'] = train[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']].median(axis=1)
```

3. Other type features

Ta sẽ **biến đổi DAYS_LAST_PHONE_CHANGE sang số năm bằng cách chia cho -365 ngày**. Khi đó, ta sẽ có phân bố mới như bên dưới.

```
train['YEARS_LAST_PHONE_CHANGE'] = train['DAYS_LAST_PHONE_CHANGE'] / (-365)
```

How many times clients were late with payments or defaulted his loans. Info about his social circle is also important.
I'll divide values into **3 groups: 0, 1 and more than 1.**

DELIQUENCIES

It is very important to see how many times clients was late with payments or defaulted his loans. I suppose info divide values into 3 groups: 0, 1 and more than 1

```
train.loc[train['OBS_60_CNT_SOCIAL_CIRCLE'] > 1, 'OBS_60_CNT_SOCIAL_CIRCLE'] = '1+'
train.loc[train['DEF_60_CNT_SOCIAL_CIRCLE'] > 1, 'DEF_60_CNT_SOCIAL_CIRCLE'] = '1+'
train.loc[train['DEF_30_CNT_SOCIAL_CIRCLE'] > 1, 'DEF_30_CNT_SOCIAL_CIRCLE'] = '1+'
```

V. DATA ANALYSIS: INSIGHT FINDINGS

- Use these charts to visualize features: **pie chart, bar chart, histogram, etc.**

- We **extract insights** from old features and new features

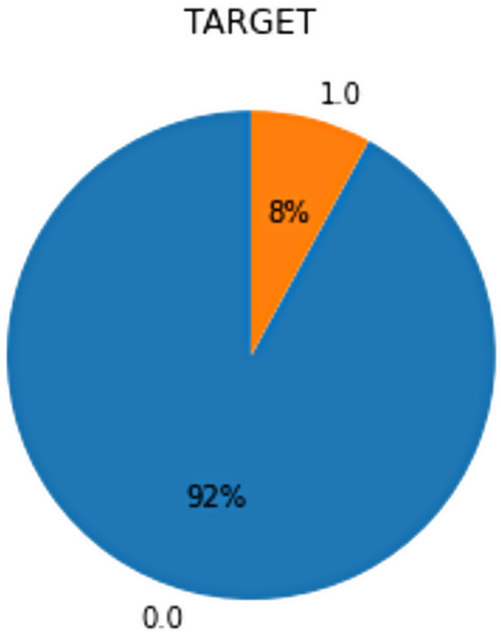
Use these charts to visualize features: **pie chart, bar chart, histogram, etc.**

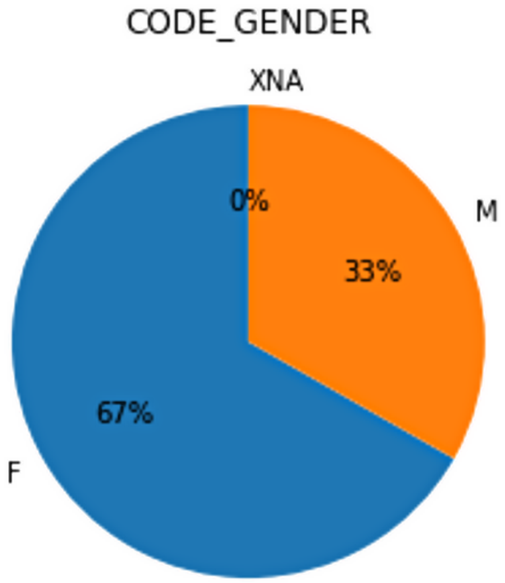
```python
def pie_chart(x):
    temp = train[x].value_counts(normalize= True)
    df = pd.DataFrame({'labels': temp.index,
                       'values': temp.values
                      })
    plt.pie(temp,labels=df['labels'], autopct='%.f%%', startangle=90)
    #plt.legend()
    plt.title("{}".format(x))
    plt.show()
```

```python
def distribution(x):
    plot = sns.distplot(train[x])
    plt.title(x)
    plt.show()
```

```python
def bar_chart_pct(x):
    sns.set(style="whitegrid")
    ax = sns.histplot(train, x=x, stat="percent", multiple="dodge", shrink=.8)
    plt.xticks(rotation=90)
    plt.show()
```

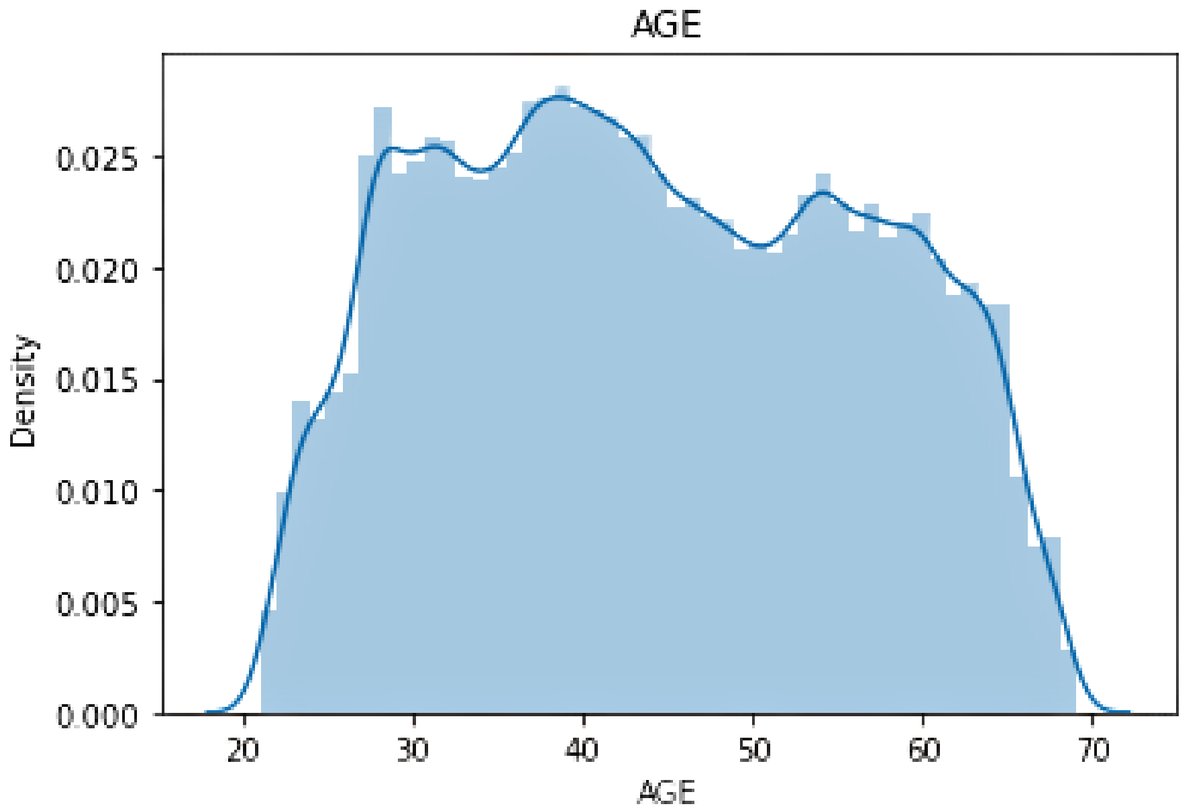
- As we can see data is **highly imbalanced.** Most of the clients are able to repay for the debt. Label 1 for difficulties, label 0 for repayable.





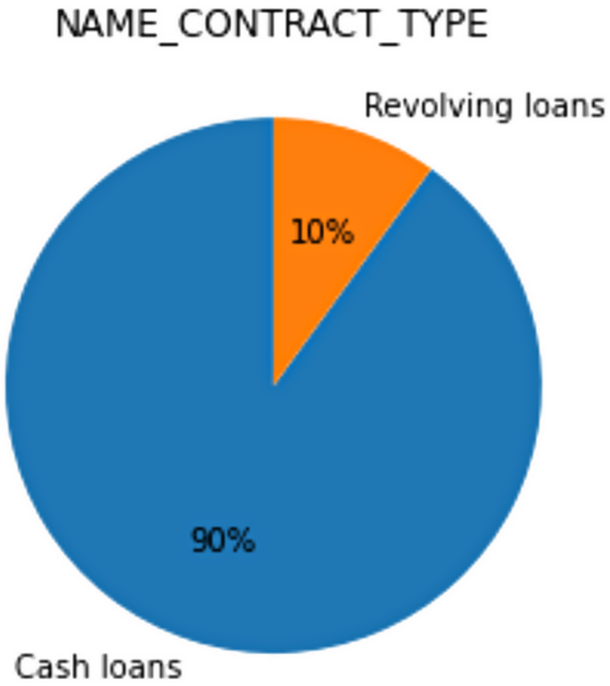
- About **more than 66.8%** of the client is a woman. It implies woman are more likely to make a loan.

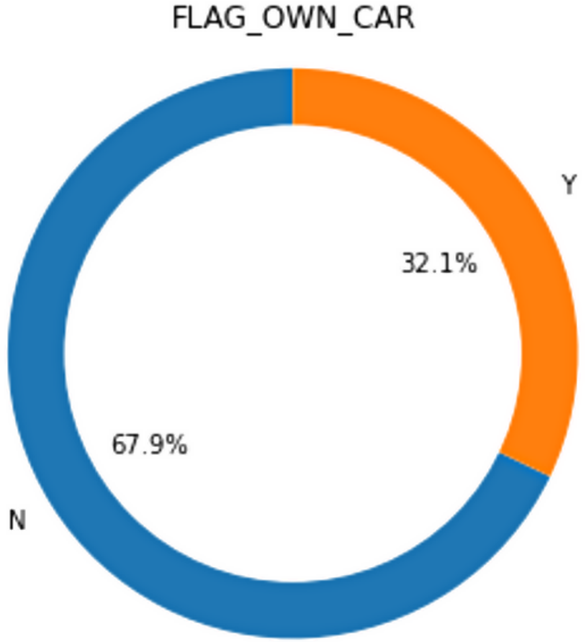
```
In [281]:  1 distribution("AGE")
```



The age of the client is **from 20 to 70**. **People at the age of 35 - 45** are mostly the client for the debt. In general, in the *first phase* of the working age, people **increase the need for application for a loan** from time to time. In the *next period from 40 - 50*, there is a **slight decrease** in need for a loan. But after that, *from over 50 to retirement*, average people **increase** the need for a loan again.

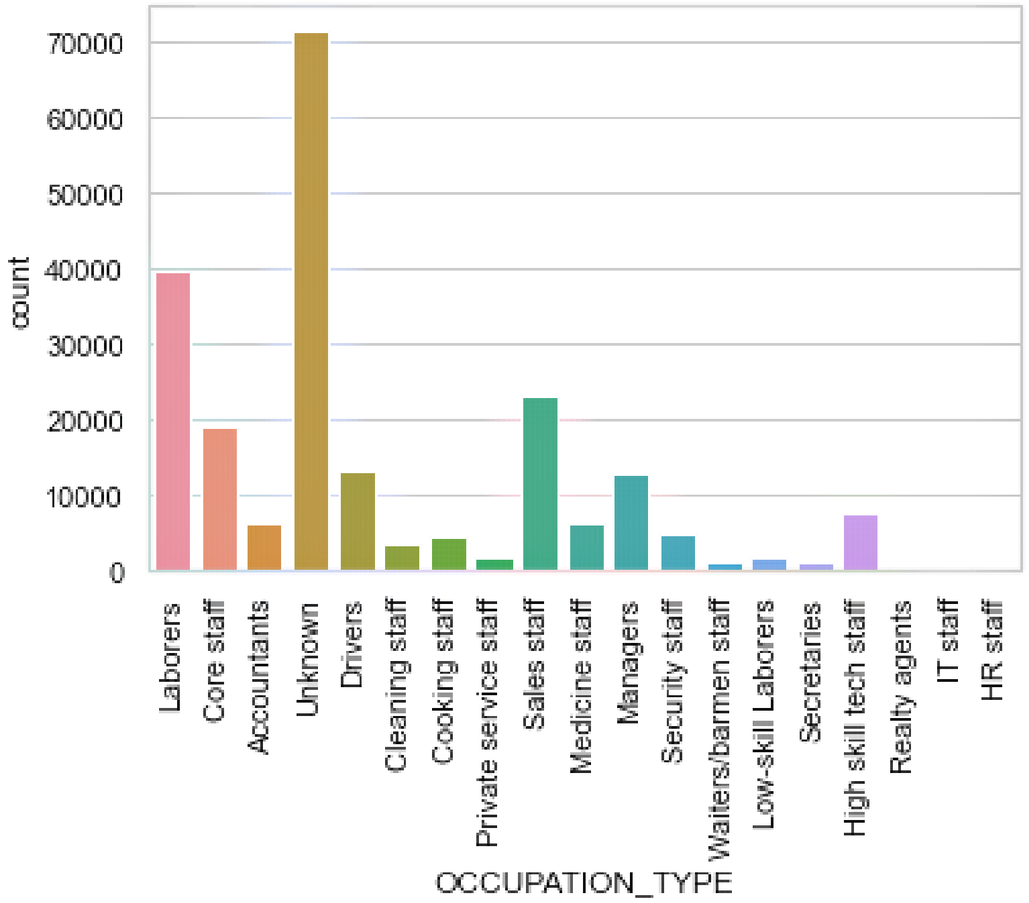
- Most of the loans are **Cash loans** which were taken by applicants (90%)





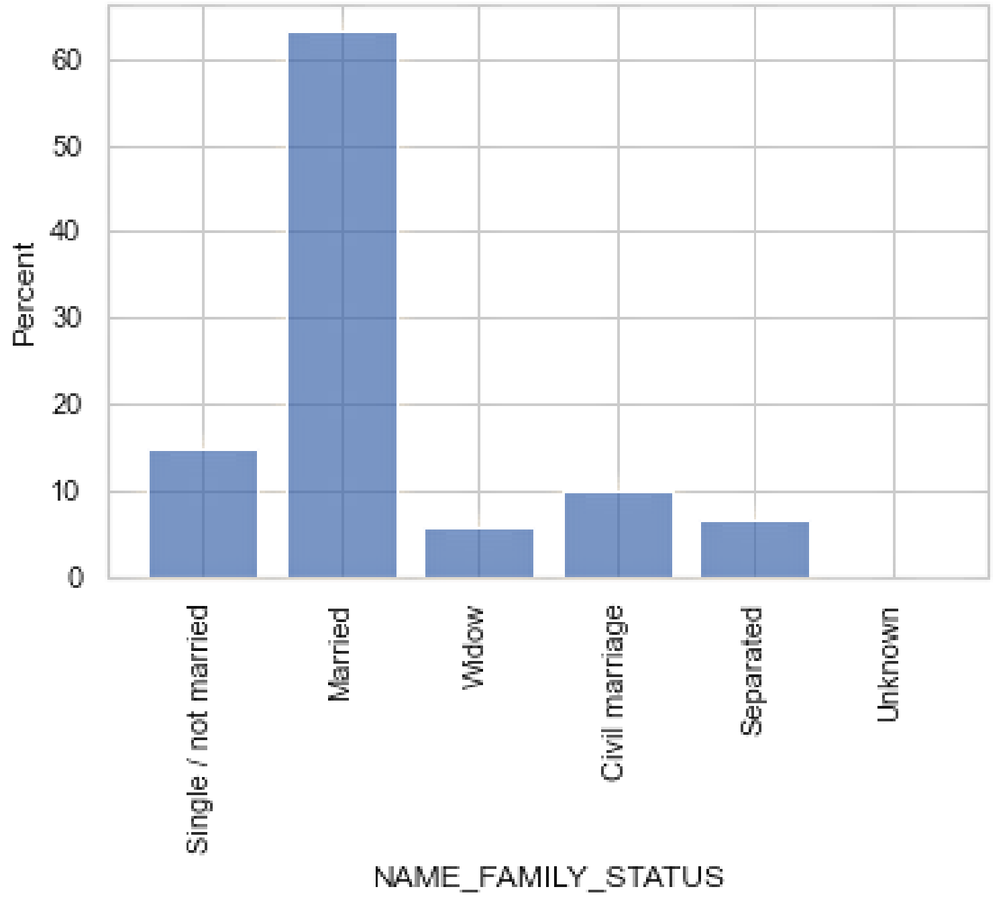
- **Only 32% of clients** have (at least) a car, which can be a good collateral.

```
1 bar_chart("OCCUPATION_TYPE")
```

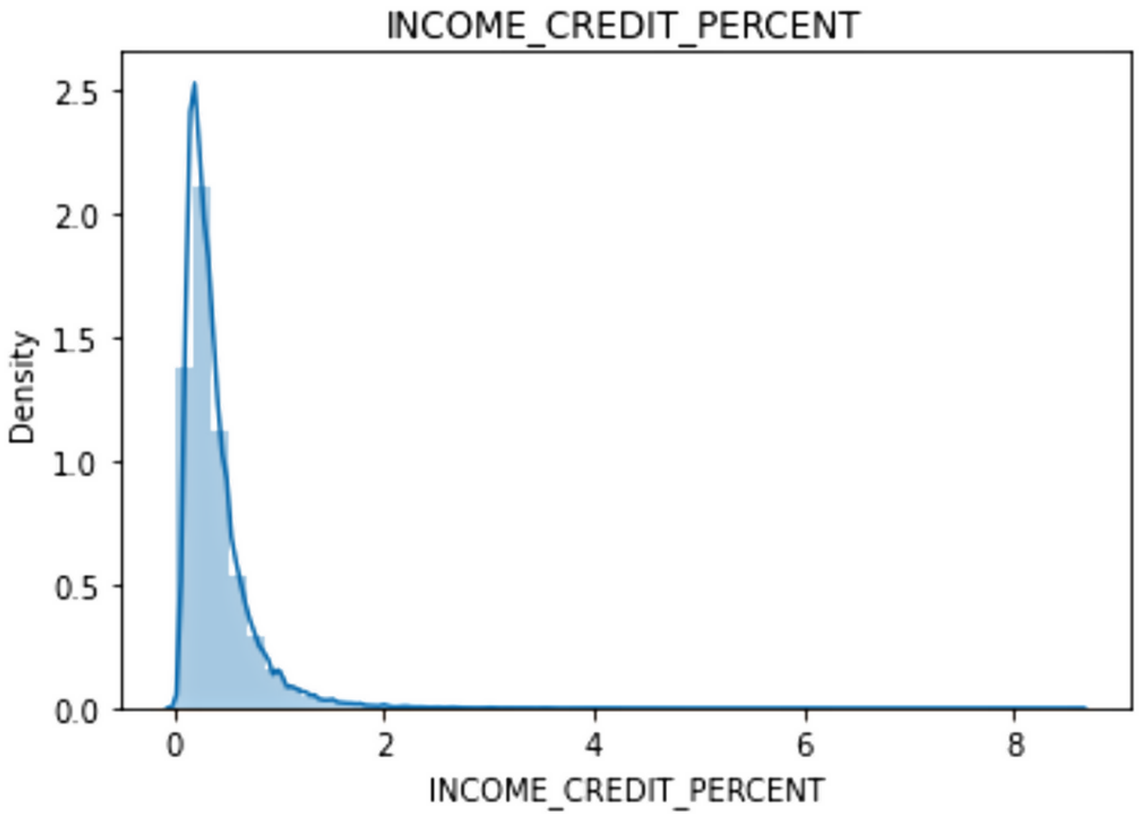


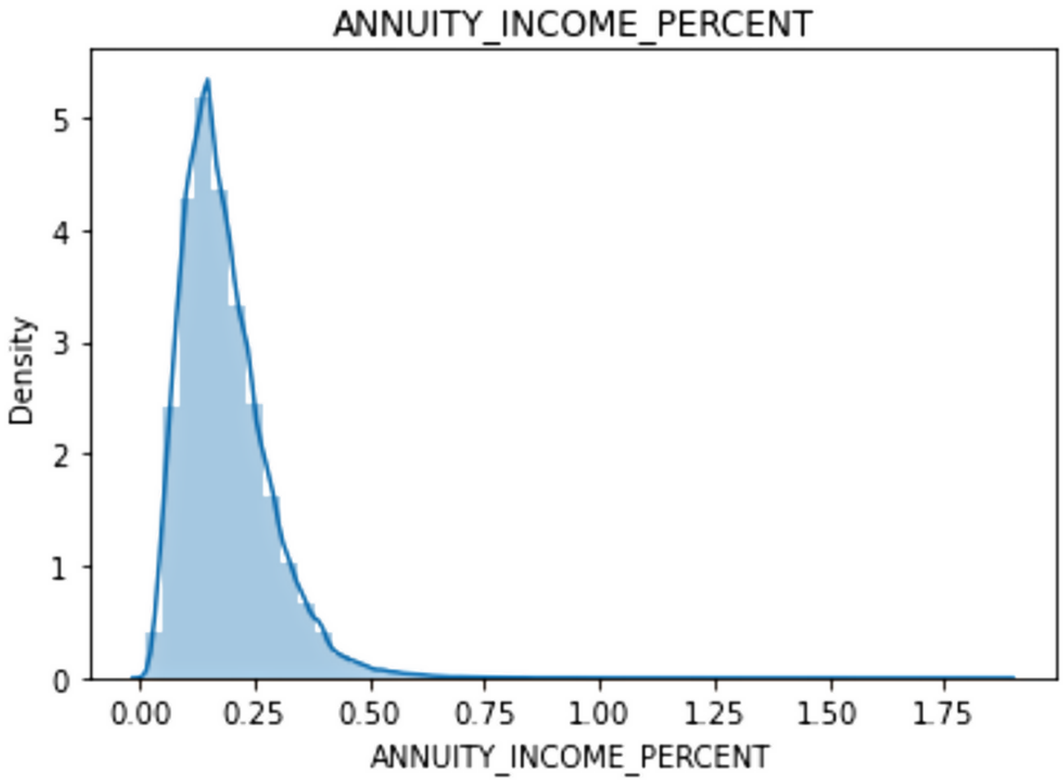
Most of the client is the **laborers, sales staff or core staff,** which are **working** at the time of application. But many of the clients did **not provide** their current job situation.

```
1 bar_chart_pct("NAME_FAMILY_STATUS")
```



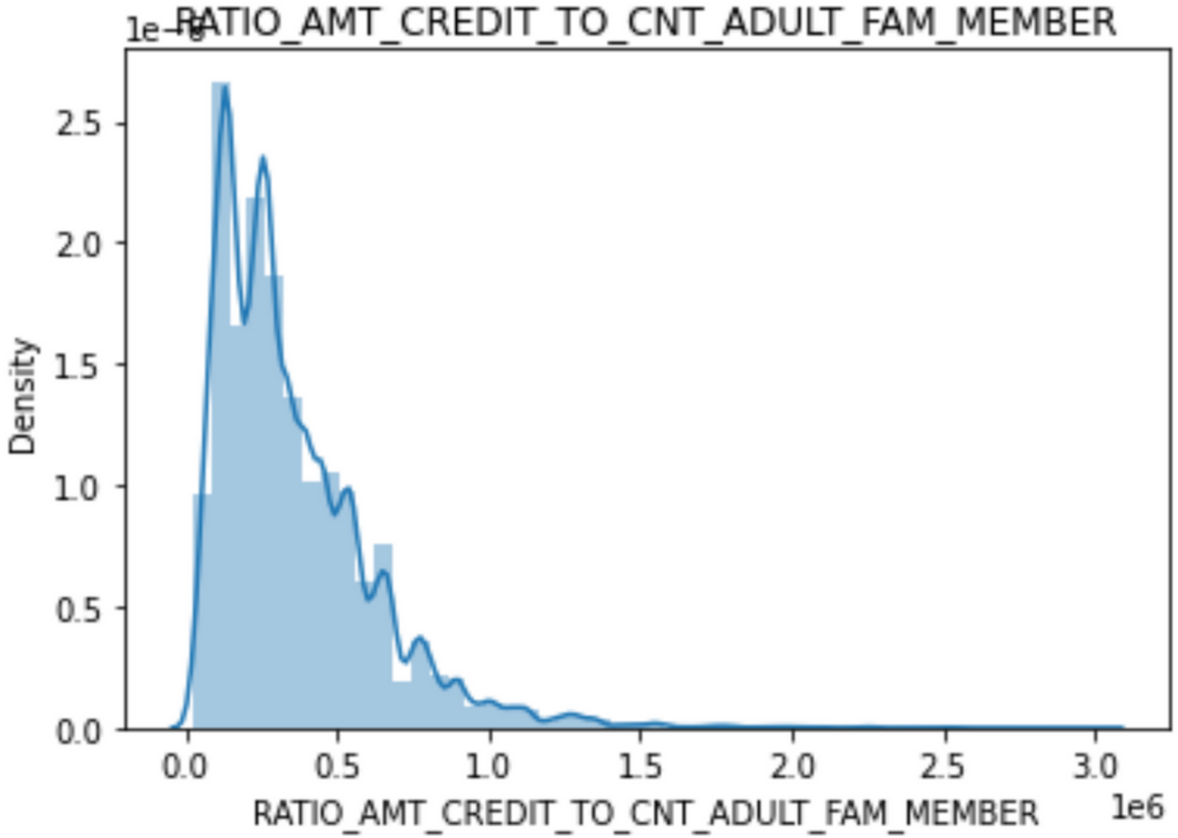
Most of the clients are **in a marriage**, but when applying for the loan, they are **unaccompanied.**



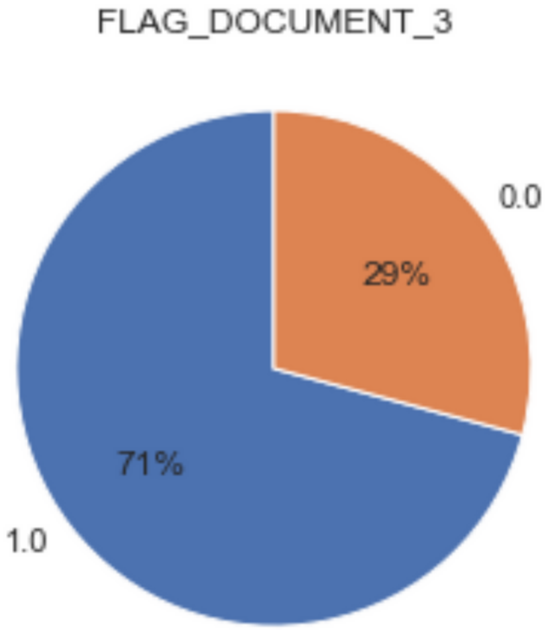


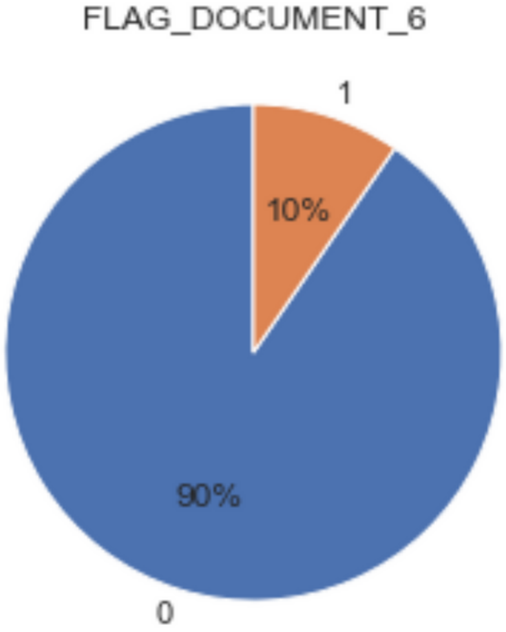
Most people have **enough income to pay for their credit**. Some people even have 8 times more than the amount of the loan

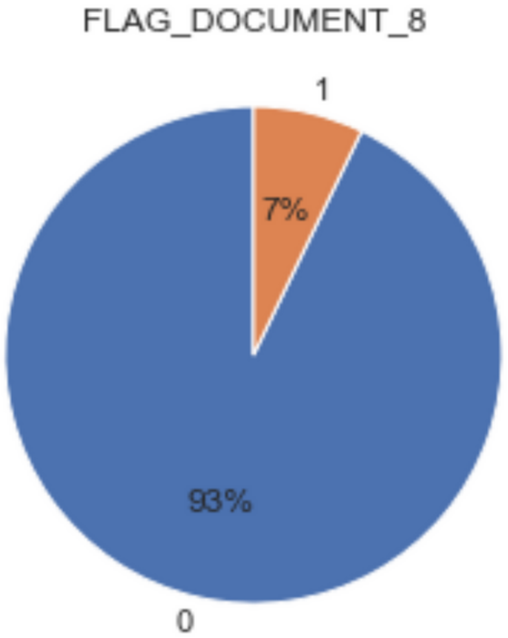
The most frequent percentage of annuity per income is below 25%. **That is, most clients have to spend at about 1/4 of their income to pay for the debt.**

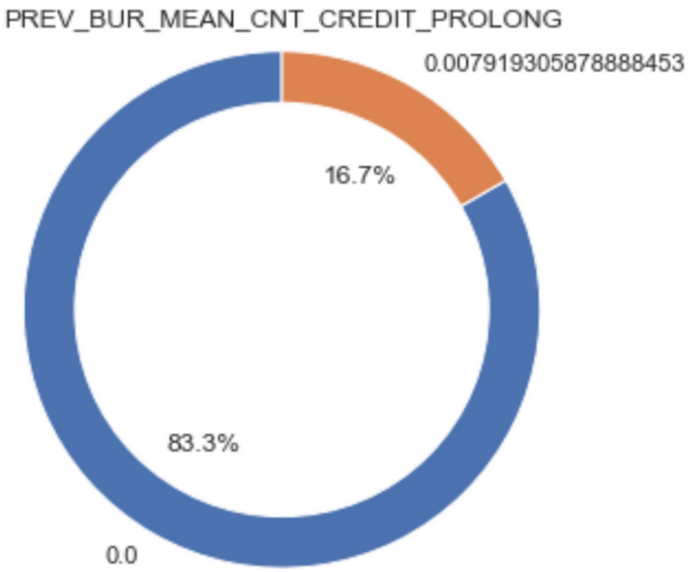


Only the adults in the family pay for the debt. We can try to divide the amount of credit per person to see how much they have to pay. Credit amount per person in the family **mostly falls to 250000.**





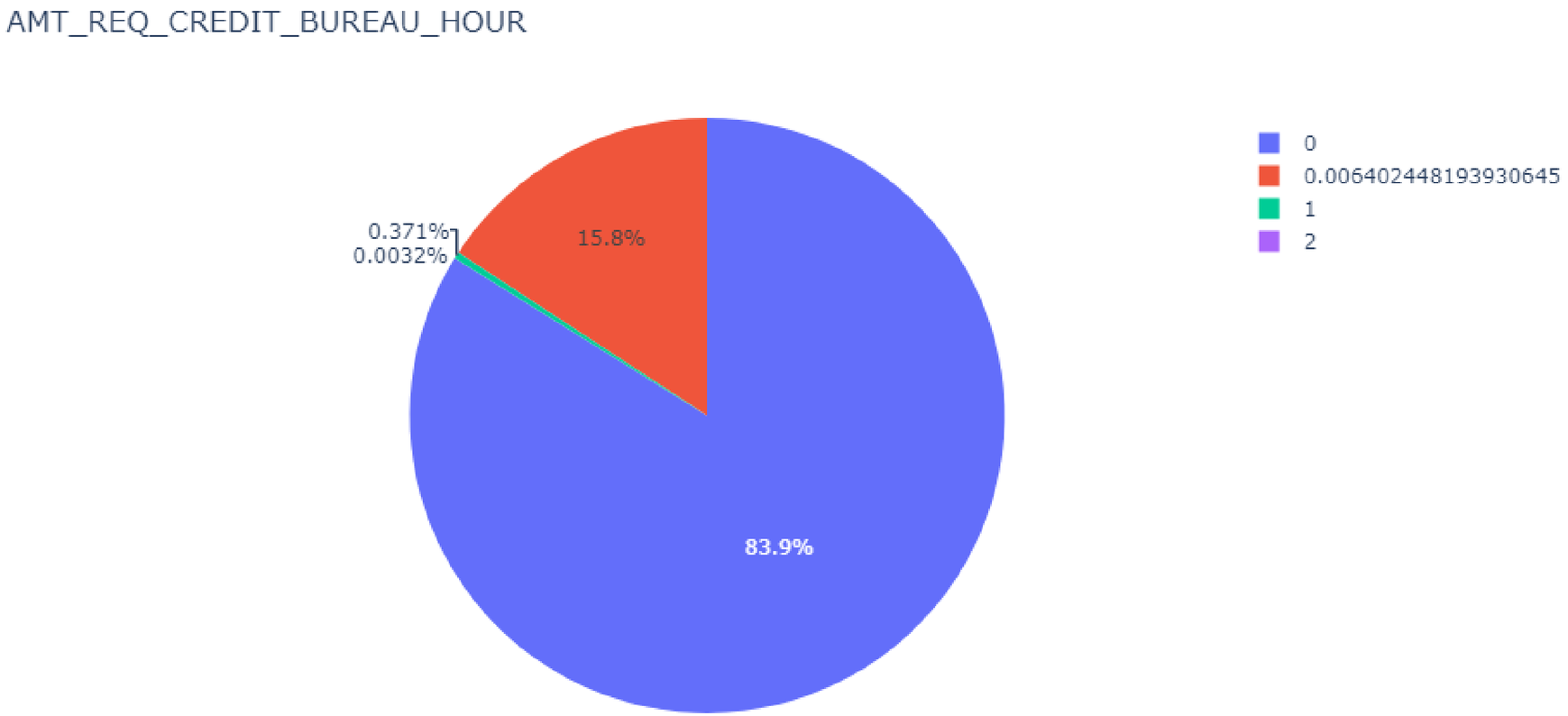


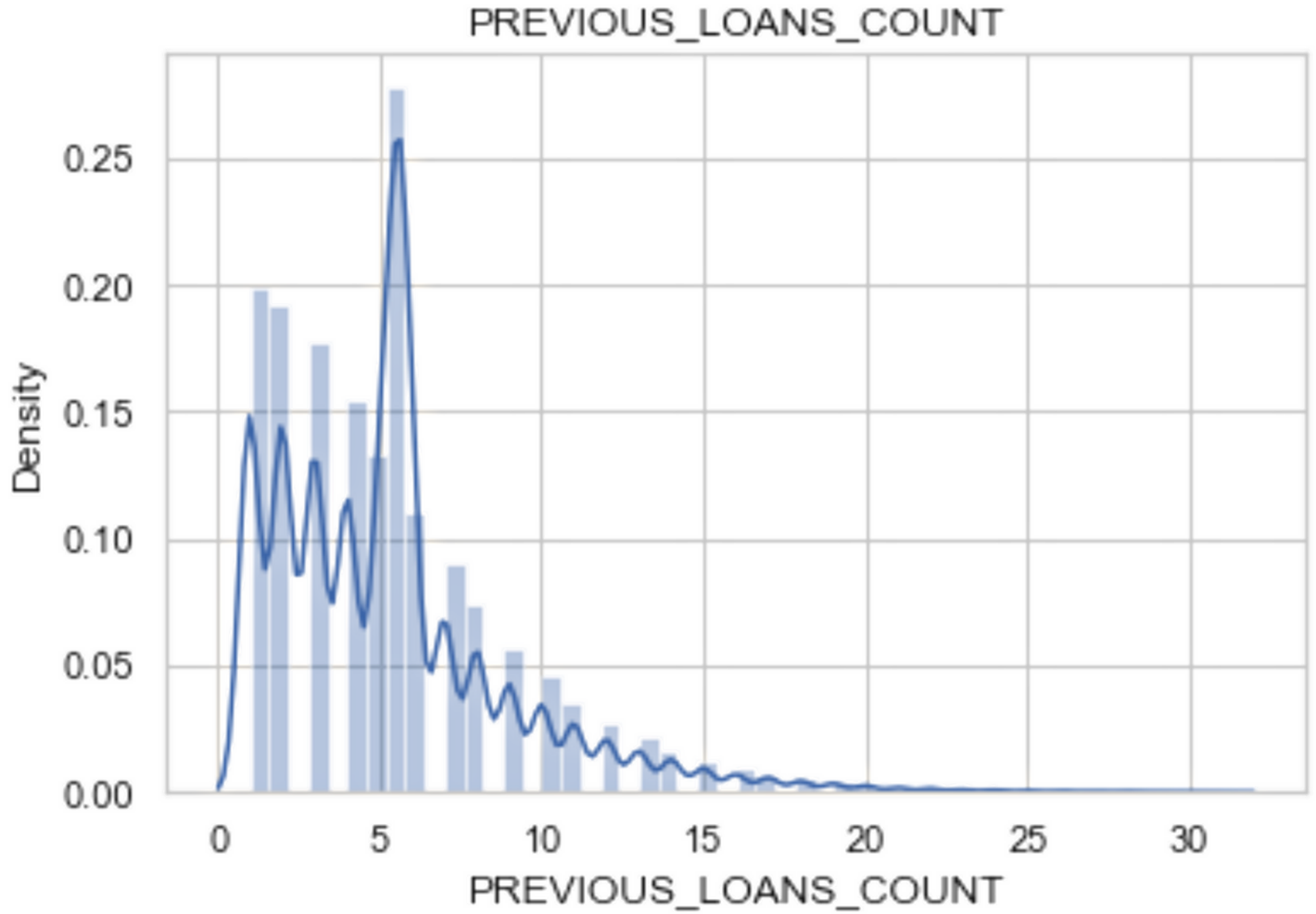


- There are **only 29% provided** document 3, while more than 90% didn't provide document 6 and 8.

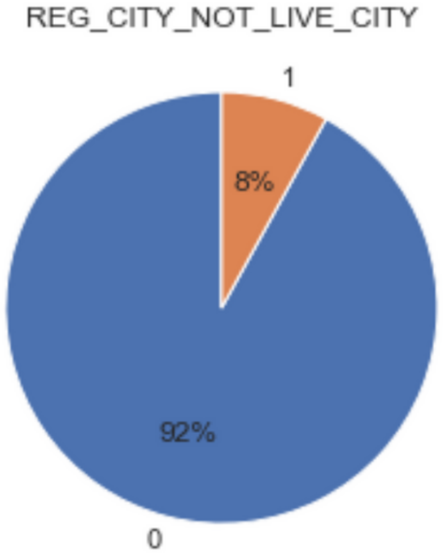
- **17% did prolong** their credit account in the past.

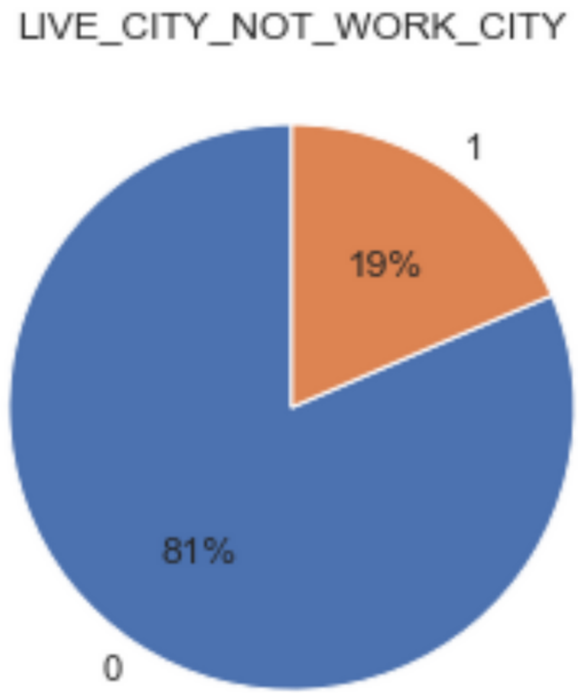
- Usually **more than 80%** have no inquiries before an hour, a day, a week before application. From a month, a year, there are more enquiries.



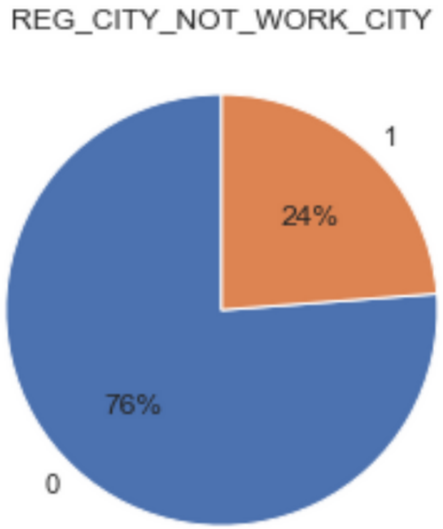


- Most people have **at least one loan before** (mostly 5 loans).



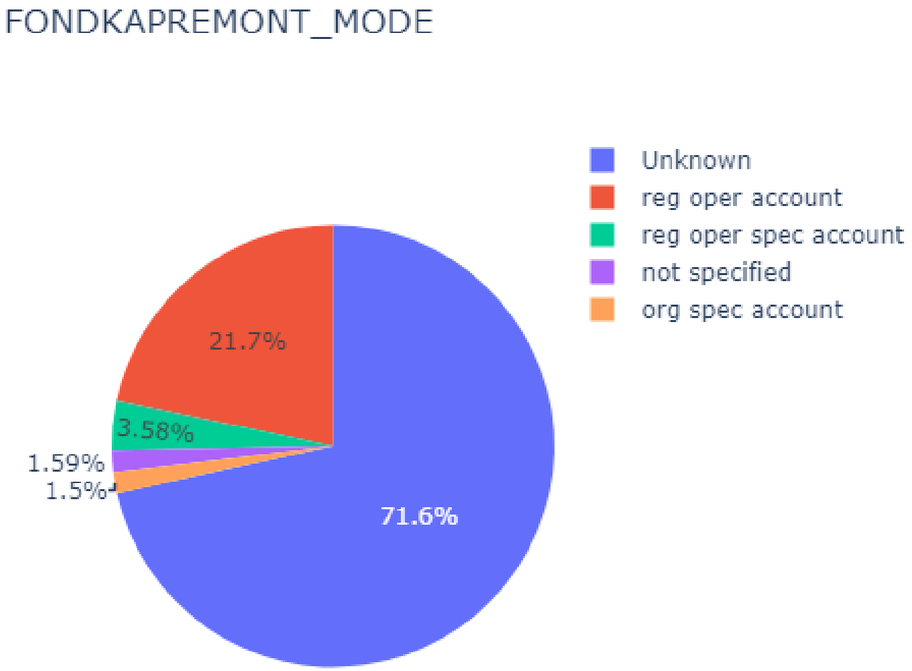


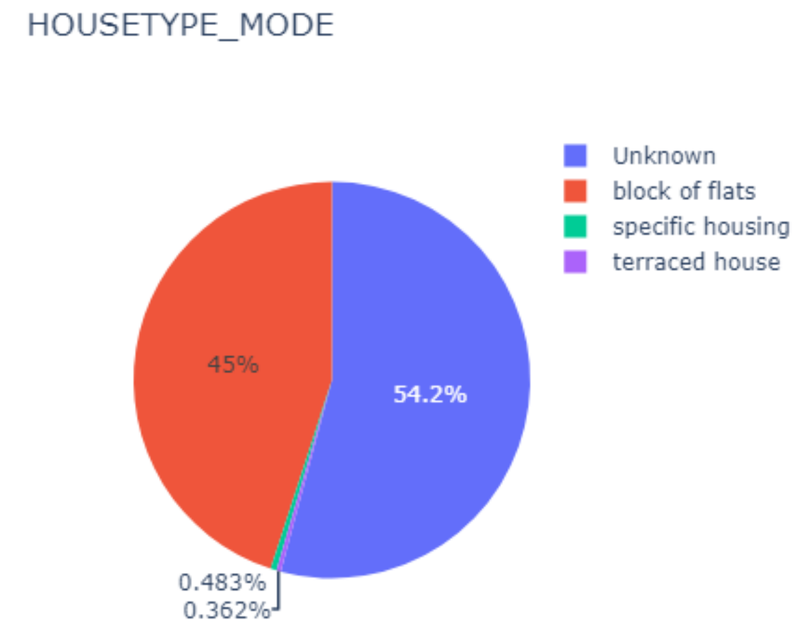

```
1 pie_chart('REG_CITY_NOT_WORK_CITY')
```

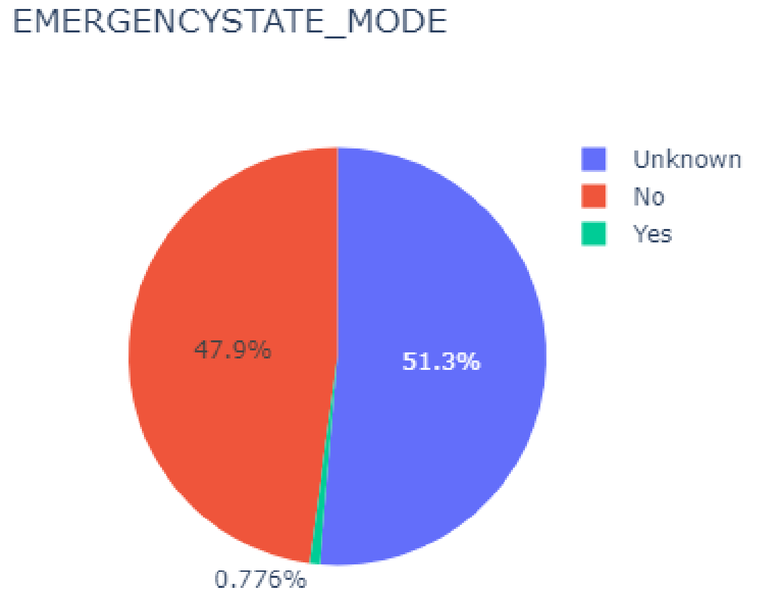


Most of the contacts and living places are the same. About 20% of customers have a work address that does not match their contact address or place of residence

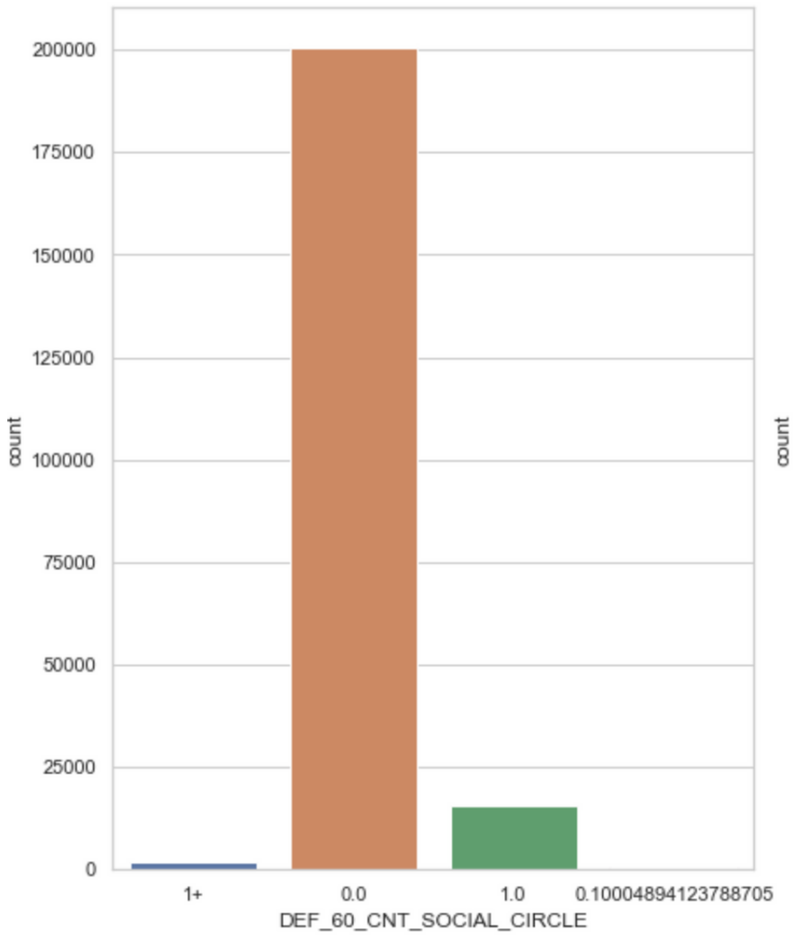
=> **Roughly a quarter of our clients WORK IN OTHER CITY WHERE TO LIVE**

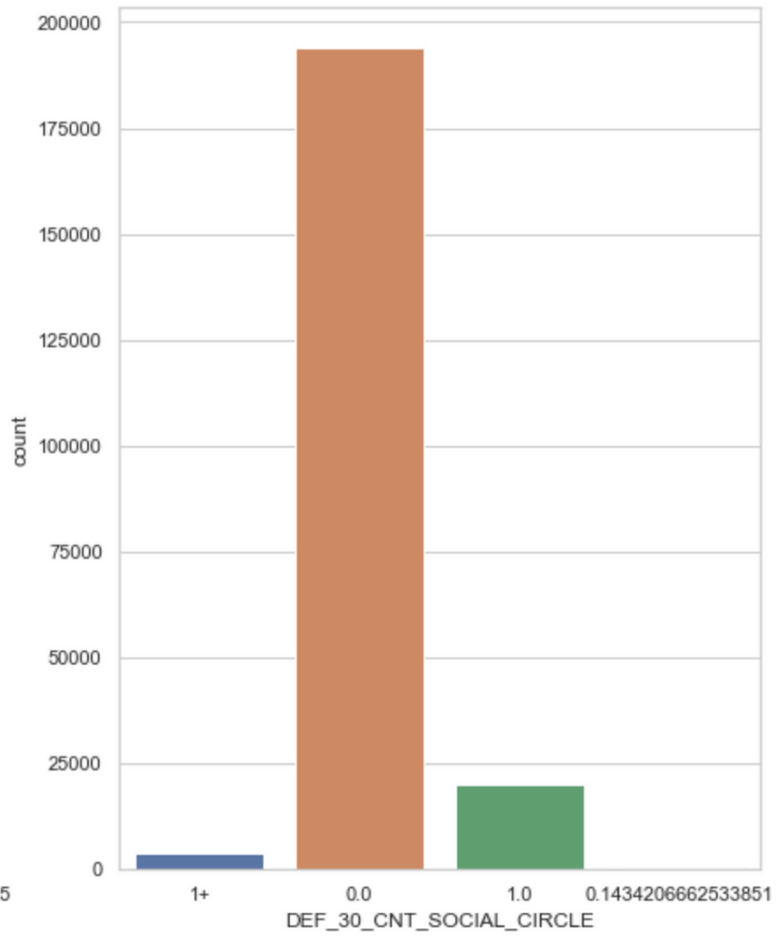


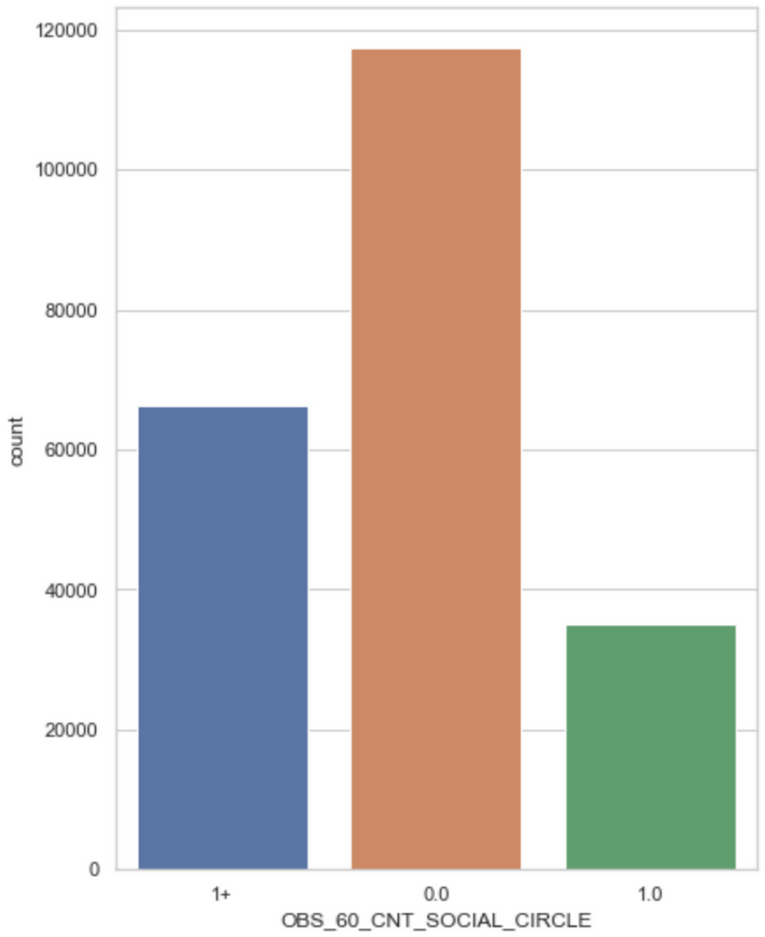




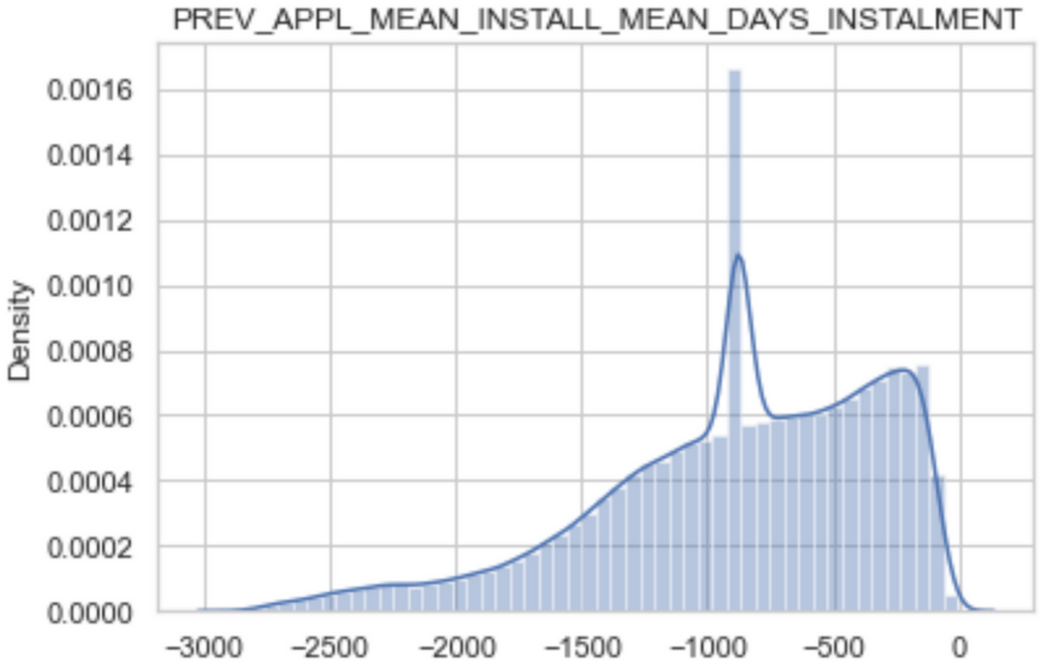
Lack of data about state of customer (EMERGENCYSTATE_MODE, HOUSETYPE_MODE,FONDKAPREMONT_MODE) as percentage of unknown value accounts more than a half

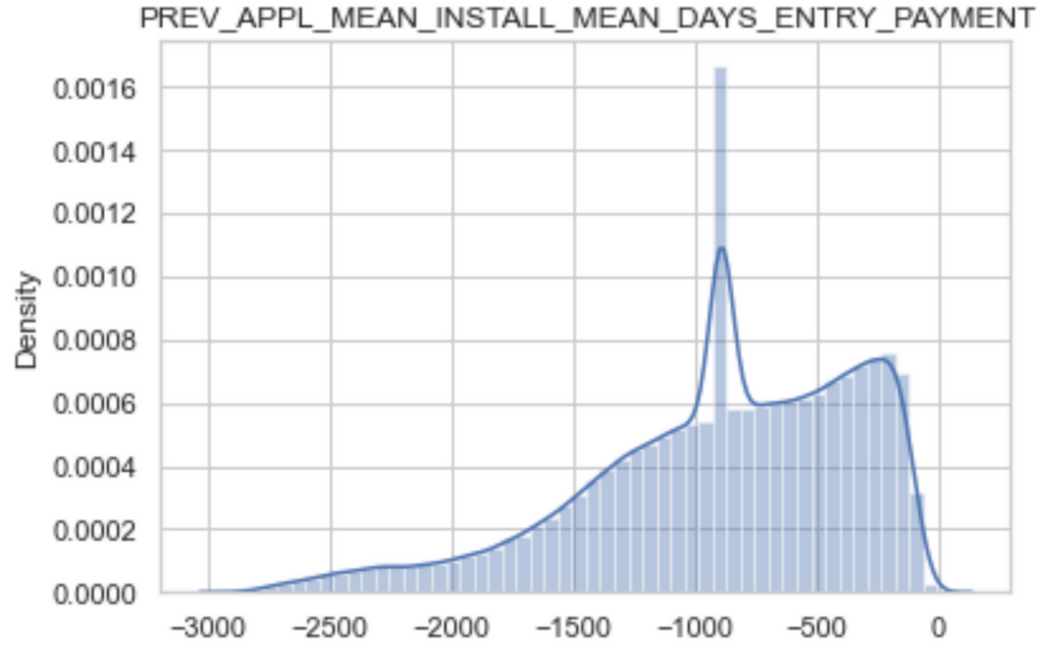






The number of people with late payment (>1 day) seems to be much higher in observations of 60 days interval installment.

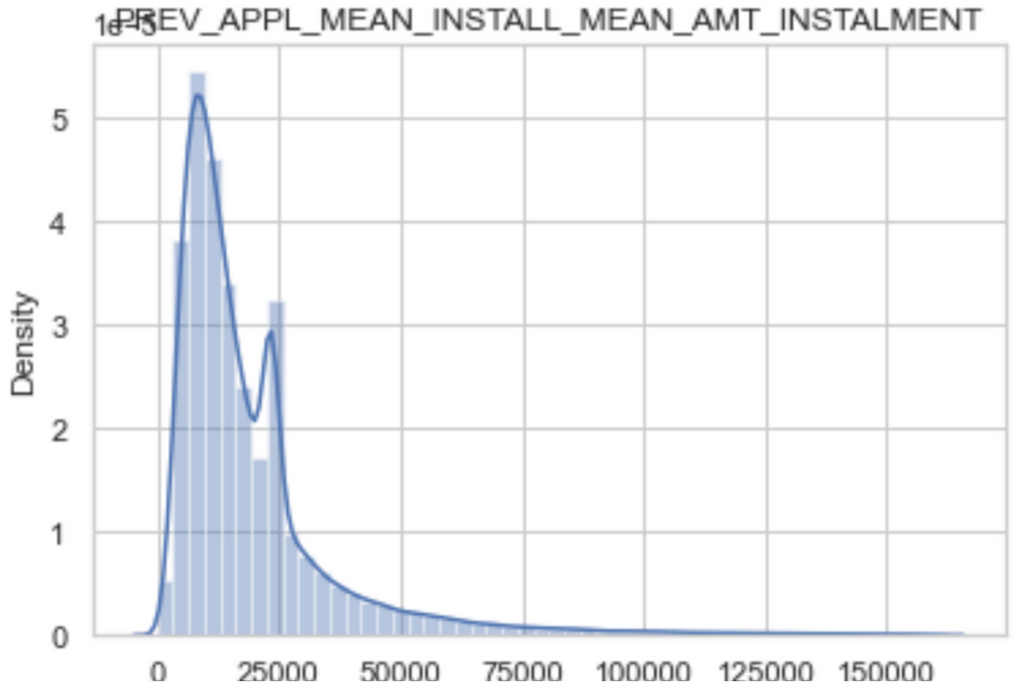


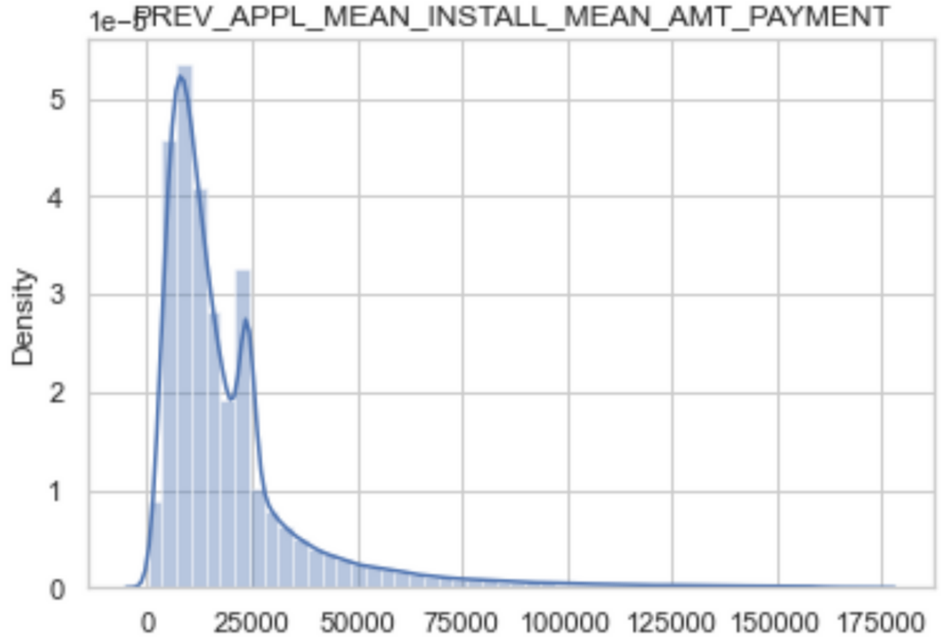


- PREV_APPL_MEAN_INSTALL_MEAN_DAYS_INSTALMENT: The day that the installment of previous credit was supposed to be paid is **highest around 850 days ago** and it takes **a lot from 100 to 2000 days ago.**

- PREV_APPL_MEAN_INSTALL_MEAN_DAYS_ENTRY_PAYMENT: The day that the installments of previous credit paid actually is the same with PREV_APPL_MEAN_INSTALL_MEAN_DAYS_INSTALMENT

=> Both these features have the **same distribution** and it is **skewed to the right**

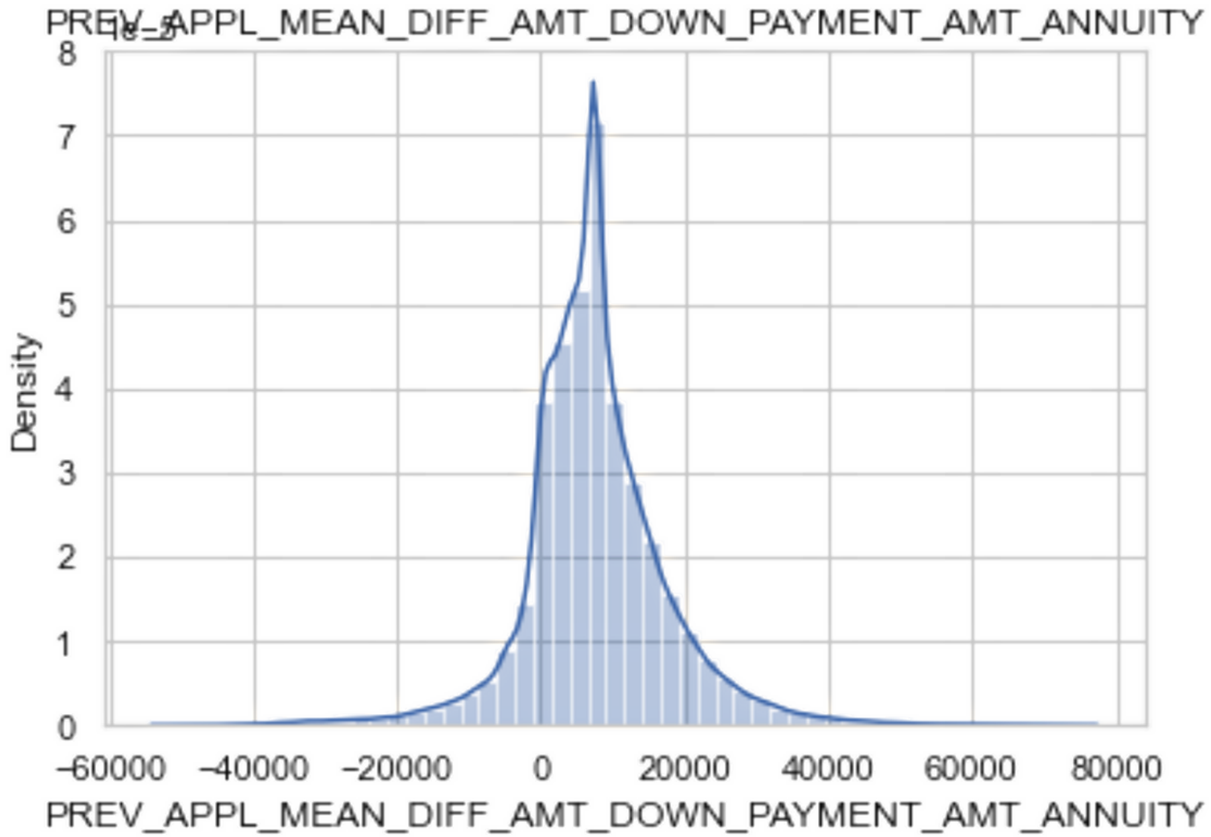




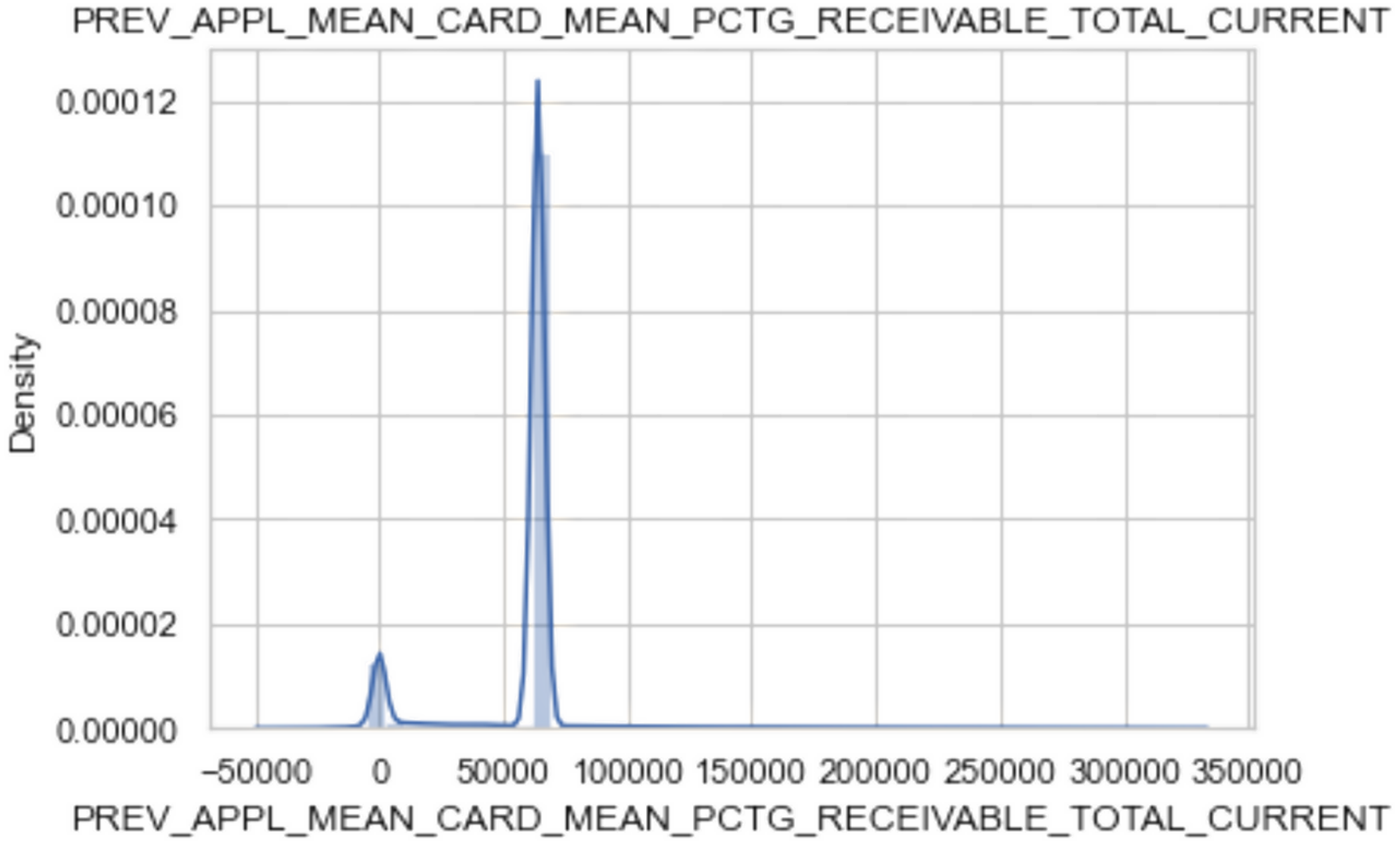
- PREV_APPL_MEAN_INSTALL_MEAN_AMT_INSTALMENT: The average prescribed installment amount of previous credit on this installment is mostly **from 1000 to 25000. The highest amount is around 22000.**

- PREV_APPL_MEAN_INSTALL_MEAN_AMT_PAYMENT: The average amount that the client actually paid on previous credit on this installment is the same with PREV_APPL_MEAN_INSTALL_MEAN_AMT_INSTALMENT.

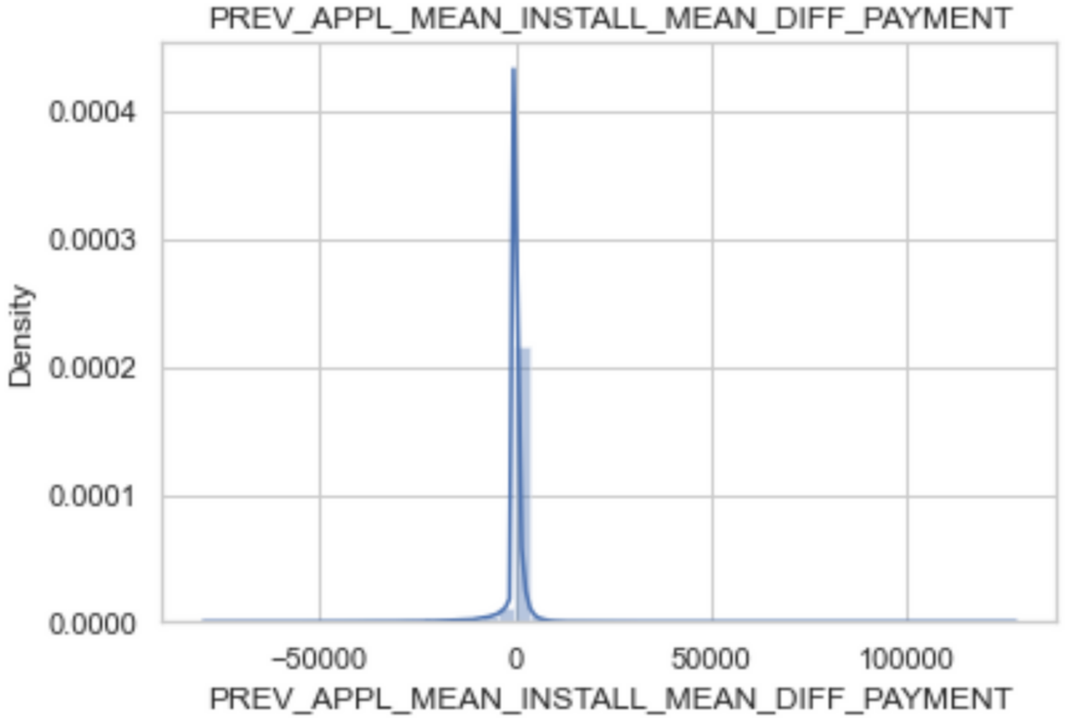
=> Both of these features have the **same distribution and it is skewed to the left.**



The average balance after paying down payment and receiving annuity of previous application of each person is almost from 0 to 20000. **We can see that many of them have a balance after these activities (almost around 6000 to 7000).**

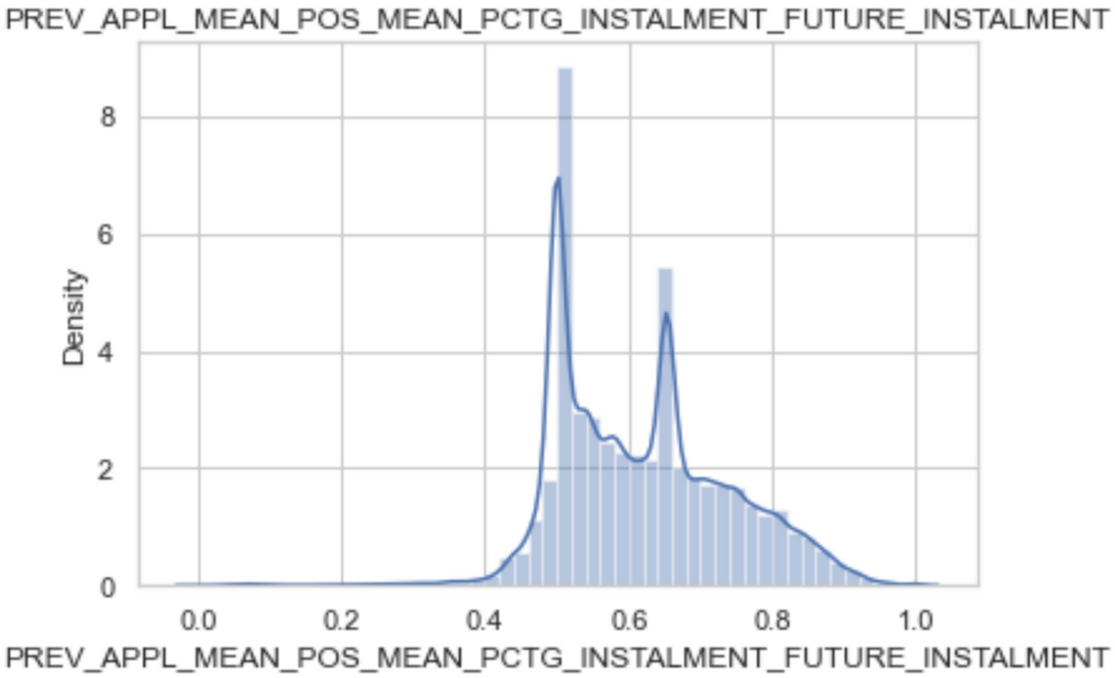


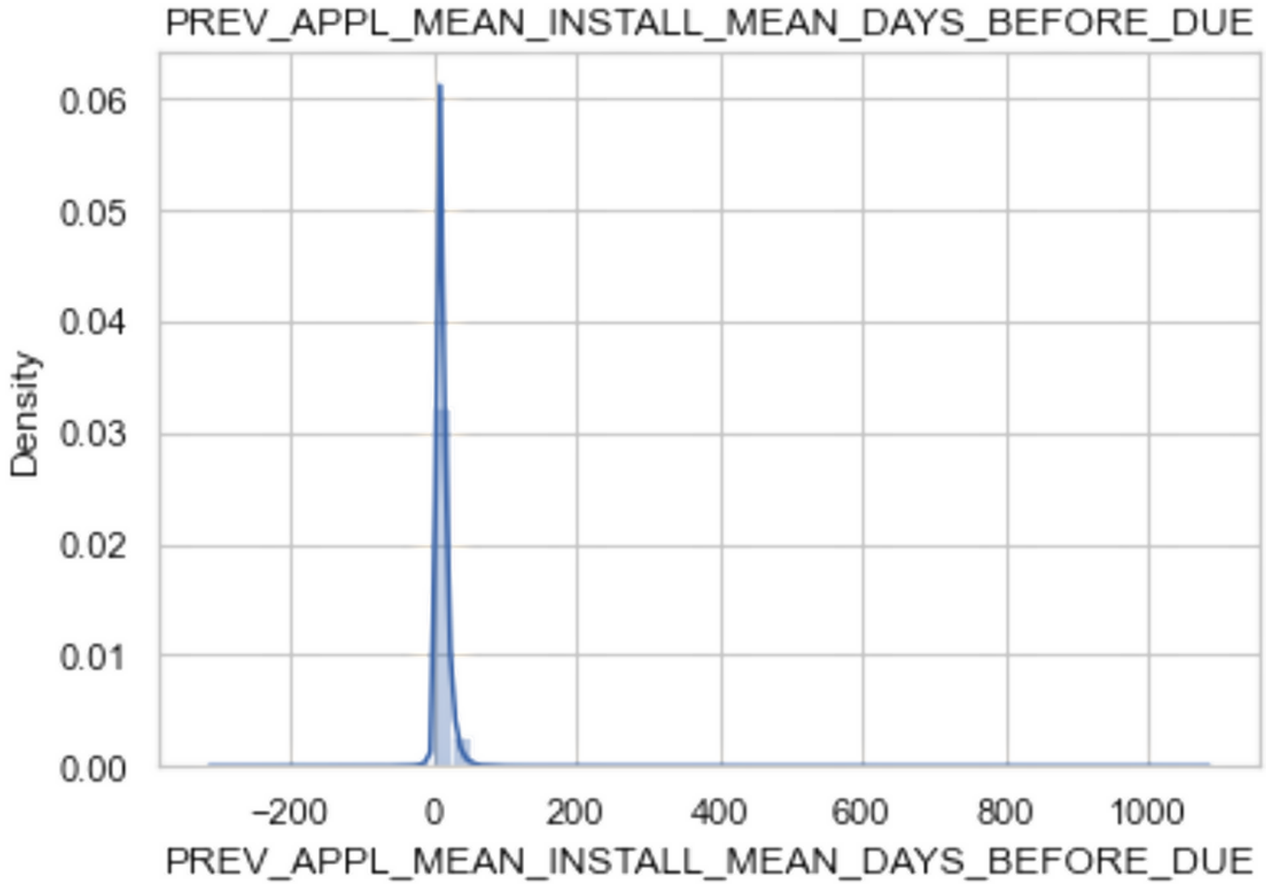
The average amount credit that the client hasn't paid during the month in total on the previous credit is **almost above 50000 (around 65000). The second is that some clients can pay in full (around 0).**



The amount of Installment that client hasn't paid **is almost around 0, we can see that the difference between the average of required payment value and the amount that was actually paid is not too much different and it's quite balanced.**

The percentage of the installments value that wasn't paid is **mostly from 0.5 to 0.7 (50-70%). Most clients can pay only nearly a half.**





The average number of days that was paid early is mostly positive, it proves that the number of clients **pay before due** or **on time is a lot.**

In contrast, a small amount is negative, it proves that there is a small number of clients paying lately -> These clients may have problems with financial conditions or there are other factors causing them to delay.

VI. IMPORTANT FEATURES THAT AFFECT TARGET

CRITERIA

This section indicates our final results on finding important features that influence customers' ability of repayment.

We have discussed in more detail in the notebook file.

01

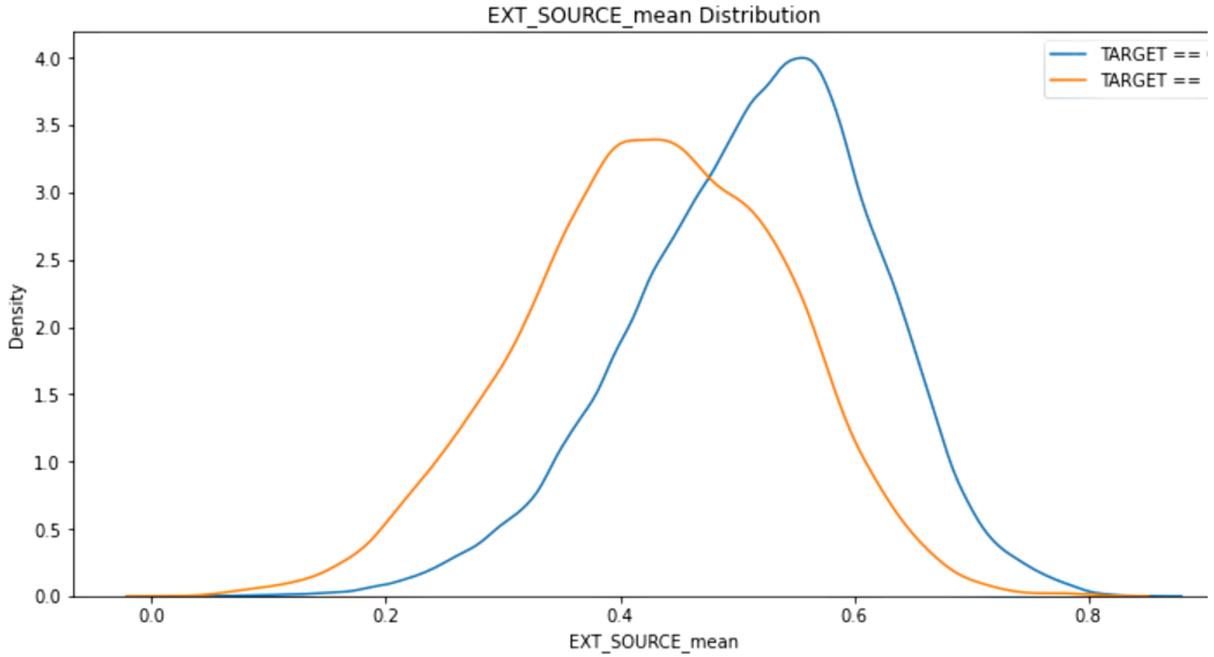
High correlation with TARGET

We choose feature that have the highest correlation score with TARGET

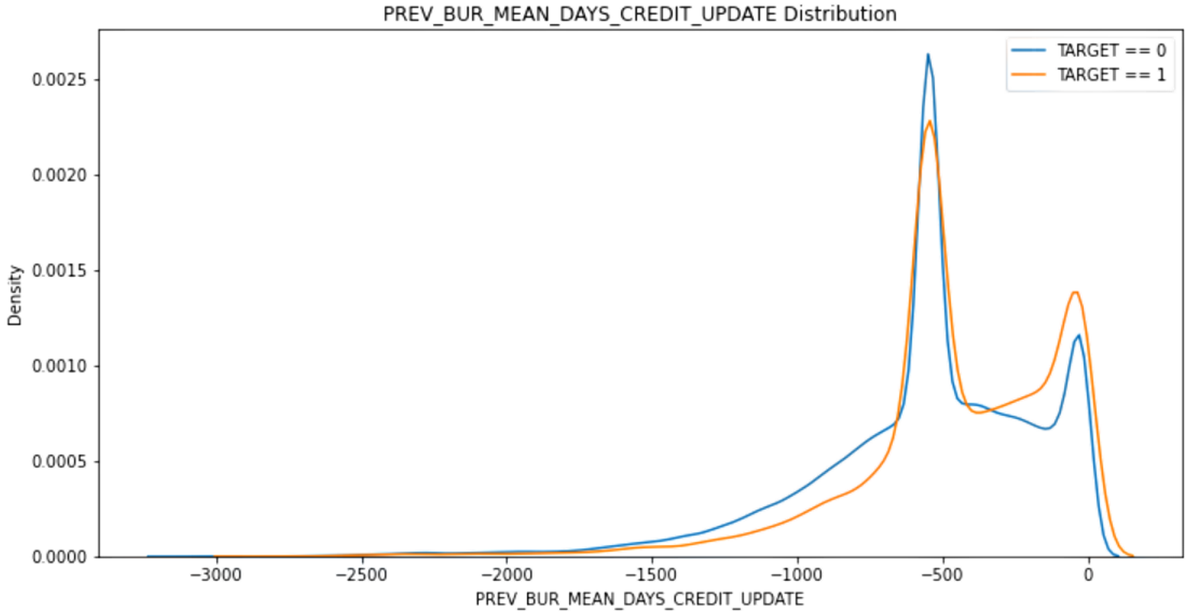
02

Have more weight of influencing TARGET

By looking at Kernel density estimation (KDE)



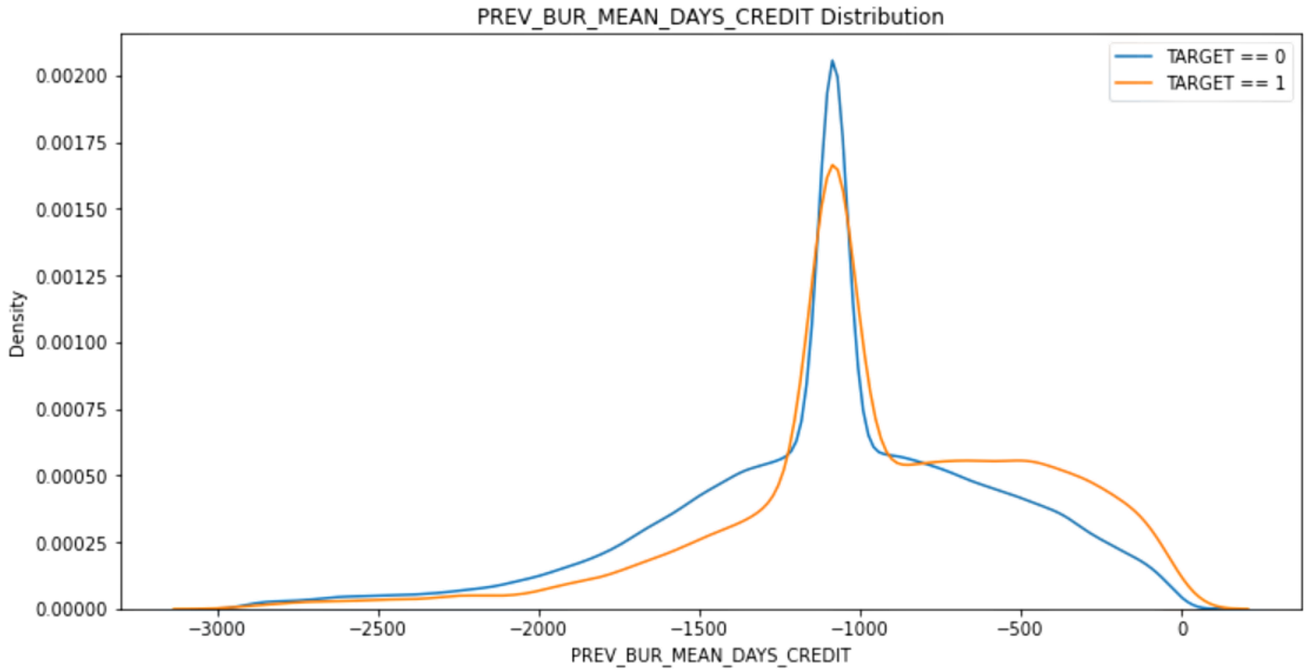

```
The correlation between EXT_SOURCE_mean and the TARGET is -0.2190
Median value for loan that was not repaid = 0.4307
Median value for loan that was repaid =     0.5240
```




```
The correlation between PREV_BUR_MEAN_DAYS_CREDIT_UPDATE and the TARGET is 0.0652
Median value for loan that was not repaid = -498.6190
Median value for loan that was repaid =     -546.6325
```

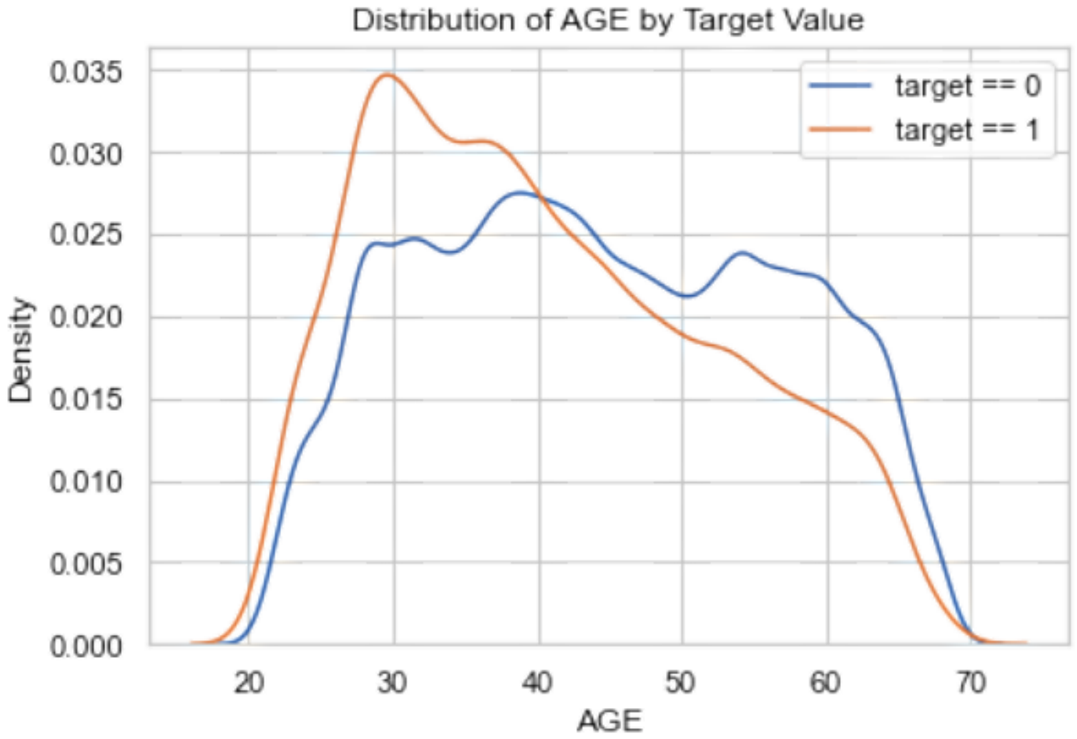
1. **External resource (EXT_SOURCE) :** Normalized score from external data source

2. PREV_BUR_MEAN_DAYS_CREDIT_UPDATE: How many days before loan application did last information about the Credit Bureau credit come

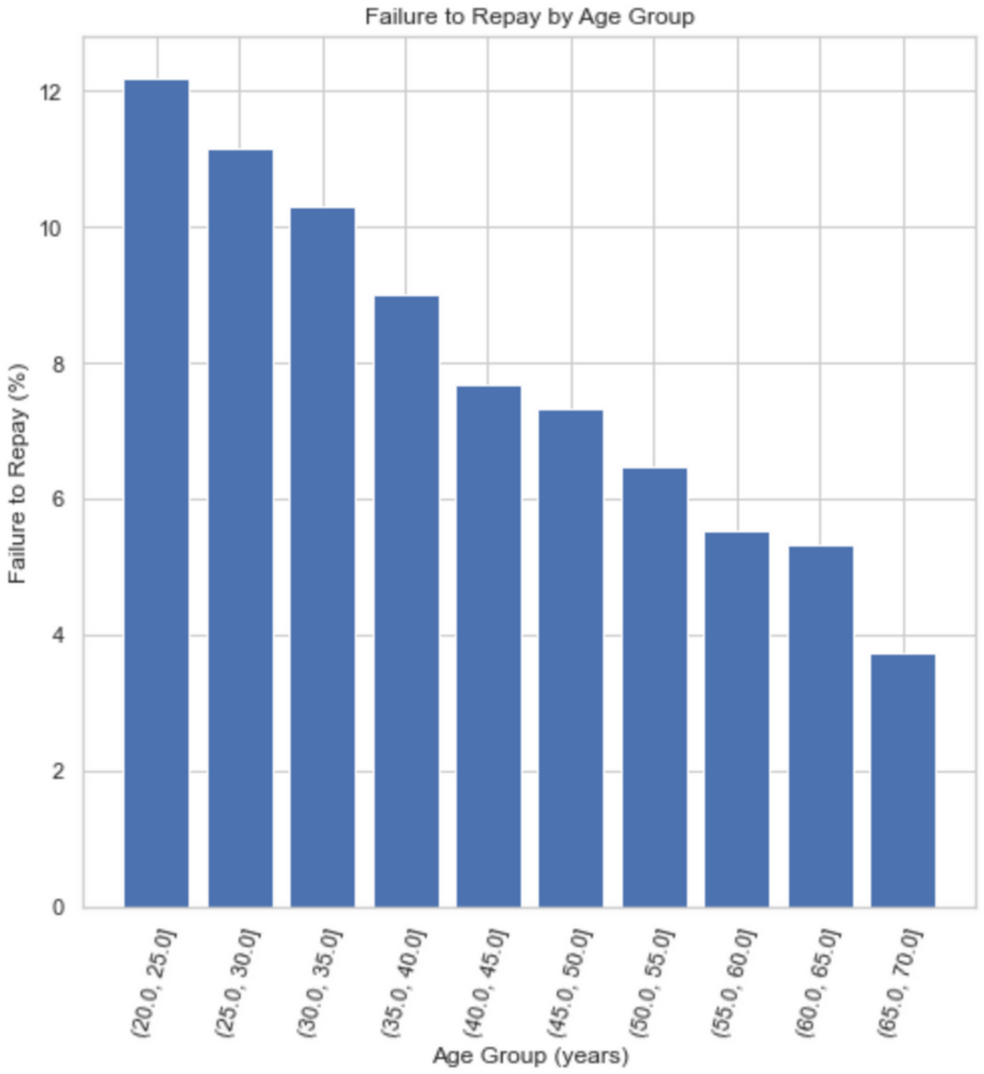



```
The correlation between PREV_BUR_MEAN_DAYS_CREDIT and the TARGET is 0.0792
Median value for loan that was not repaid = -1031.5192
Median value for loan that was repaid =     -1083.0471
```

3. PREV_BUR_MEAN_DAYS_CREDIT: How many days before current application did client apply for Credit Bureau credit



4. AGE: originally from DAYS_BIRTH



4. AGE:

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

This is information that could be directly used by the bank: because younger clients are less likely to repay the loan, maybe **they should be provided with more guidance or financial planning tips**. This does not mean the bank should discriminate against younger clients, but it **would be smart to take precautionary measures to help younger clients** pay on time.

CONCLUSION

- Reference from **external resources** seems to be reliable in terms of predicting the repayment ability

- The next features can be **considered** are: the number of days before loan application last information about the Credit Bureau credit came, the number of days before current application client applied for Credit Bureau credit and lastly clients' age