National Economics Univeristy

Home Credit Default Risk

Subject: Data Preparation & Visualizarion

Lecturer:
Ms. Giang

Group:
02

Our Team
Members

01 **Thieu Ngoc Mai**

ID: 11213717

02 **Tran Phuong Anh**

ID: 11219258

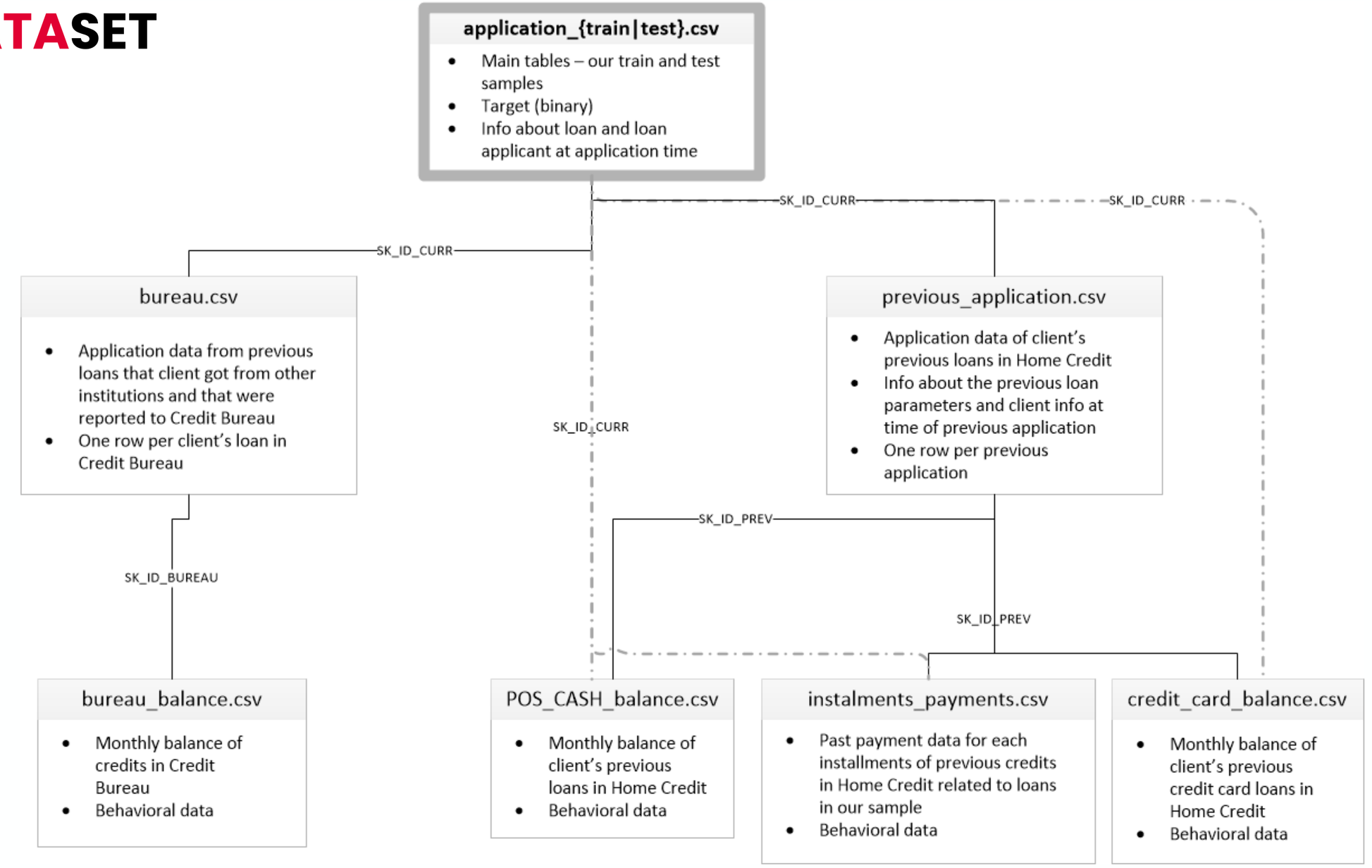
03 **Phan Anh Khoi**

ID: 11212902

04 **Truong Minh Hung**

ID: 11219273

DATASET


application_{train|test}.csv
Main tables – our train and test samples
Target (binary)
Info about loan and loan applicant at application time
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
bureau.csv
Application data from previous loans that client got from other institutions and that were reported to Credit Bureau
One row per client's loan in Credit Bureau
previous_application.csv
Application data of client's previous loans in Home Credit
Info about the previous loan parameters and client info at time of previous application
One row per previous application
SK_ID_PREV
SK_ID_BUREAU
SK_ID_PREV
bureau_balance.csv
Monthly balance of credits in Credit Bureau
Behavioral data
POS_CASH_balance.csv
Monthly balance of client's previous loans in Home Credit
Behavioral data
instalments_payments.csv
Past payment data for each installments of previous credits in Home Credit related to loans in our sample
Behavioral data
credit_card_balance.csv
Monthly balance of client's previous credit card loans in Home Credit
Behavioral data

EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis (EDA) is a crucial step in the data analysis process, providing a comprehensive understanding of the dataset's characteristics, patterns, and potential insights. It serves as a preliminary investigation, allowing data scientists, analysts, and researchers to uncover essential features, relationships, and anomalies within the data.

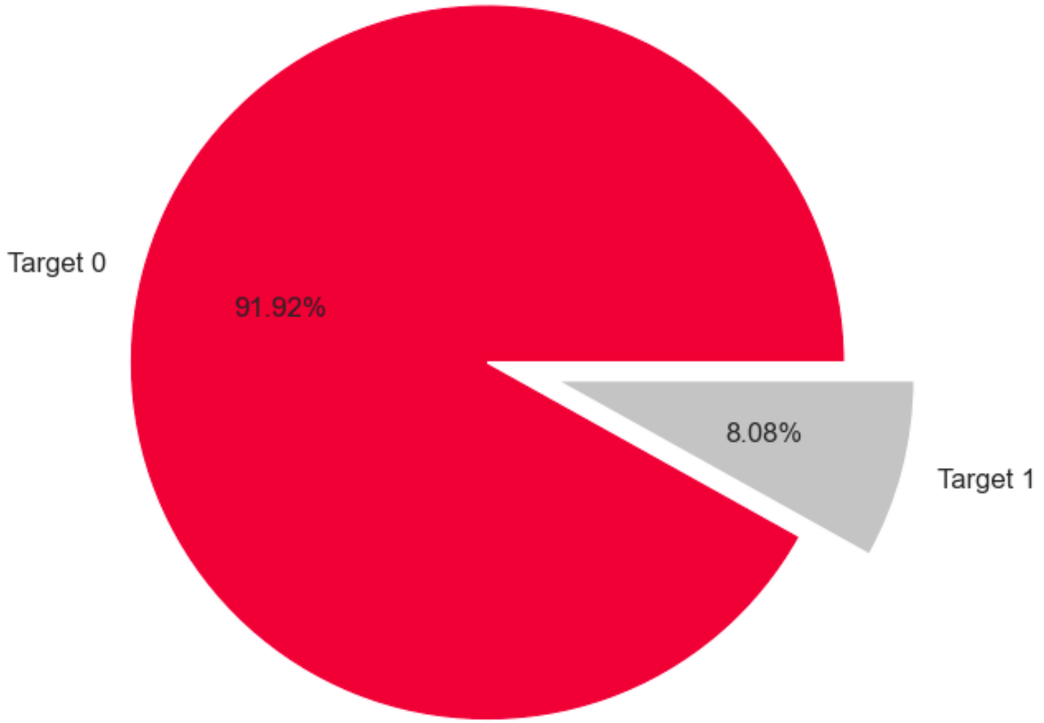
TARGET
variable

Target **0**

226,133

Target **1**

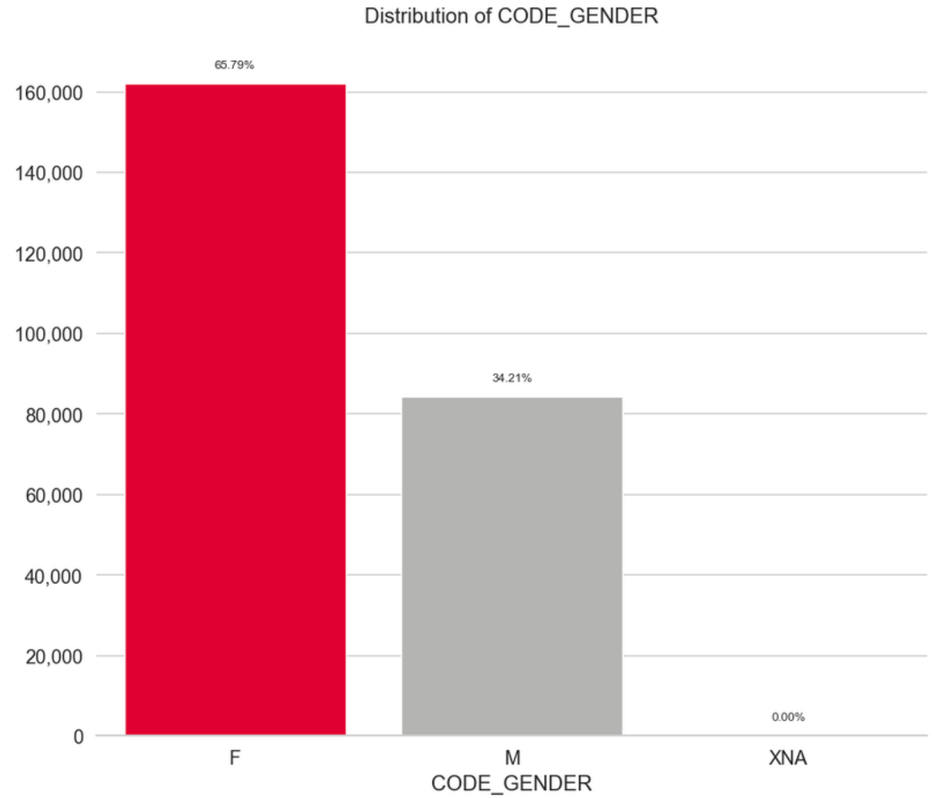
19,876

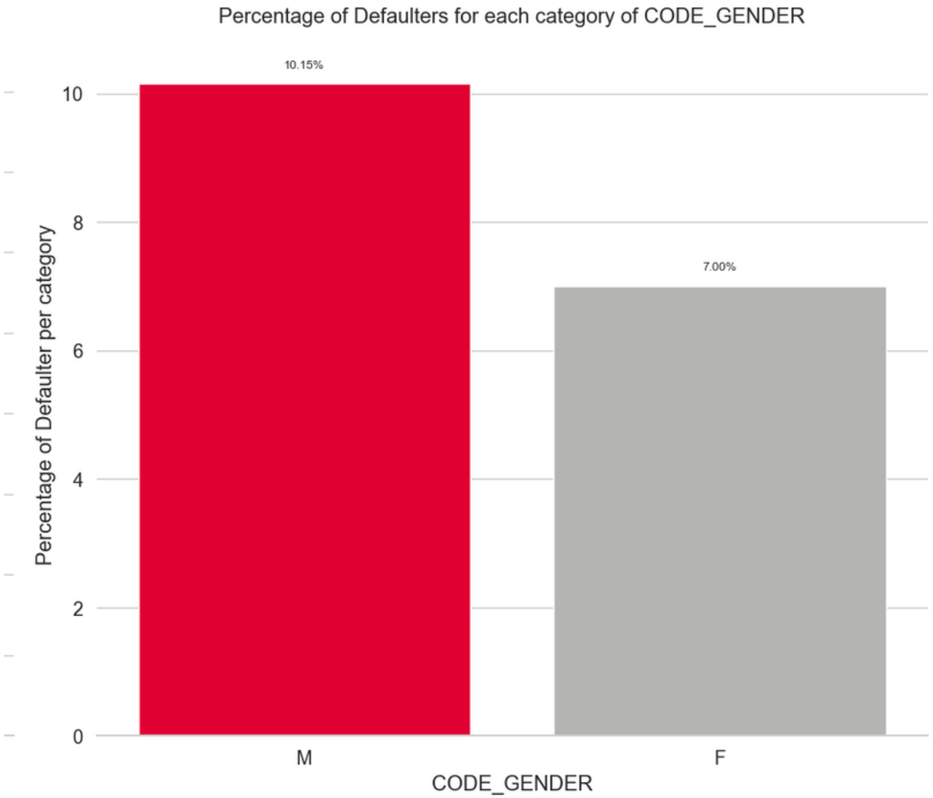


SO WHERE DOES THIS IMBALANCE COME FROM?

APPLICATION TRAIN

categorical variables - **GENDER**

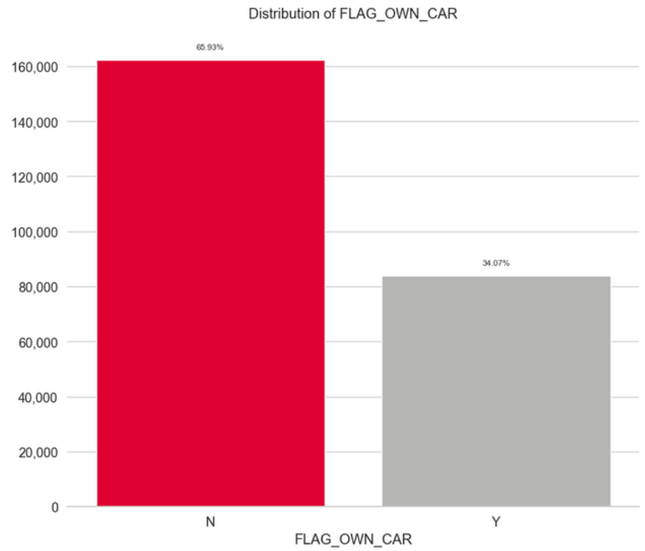


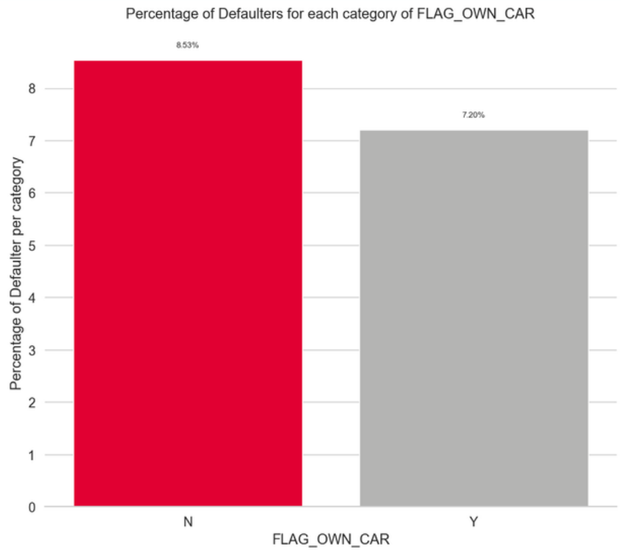


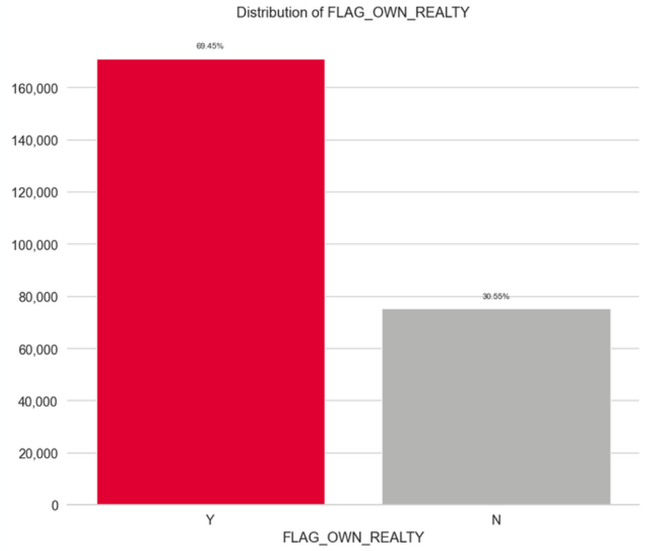
CODE_GENDER

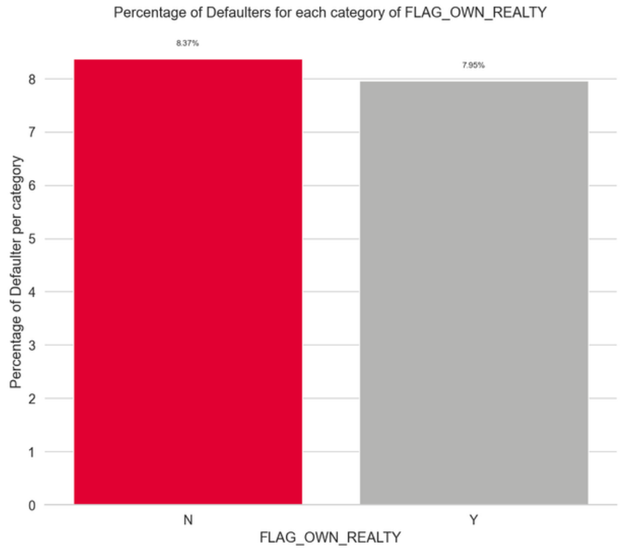
- In loan application data, women applied for the majority of loans, almost twice as often as men.
- However, the proportion of men having problems repaying debt or installments on time is higher (about 10% of the total) than women (about 7%).

categorical variables - **FLAG_OWN _CAR/REALTY**









Question:

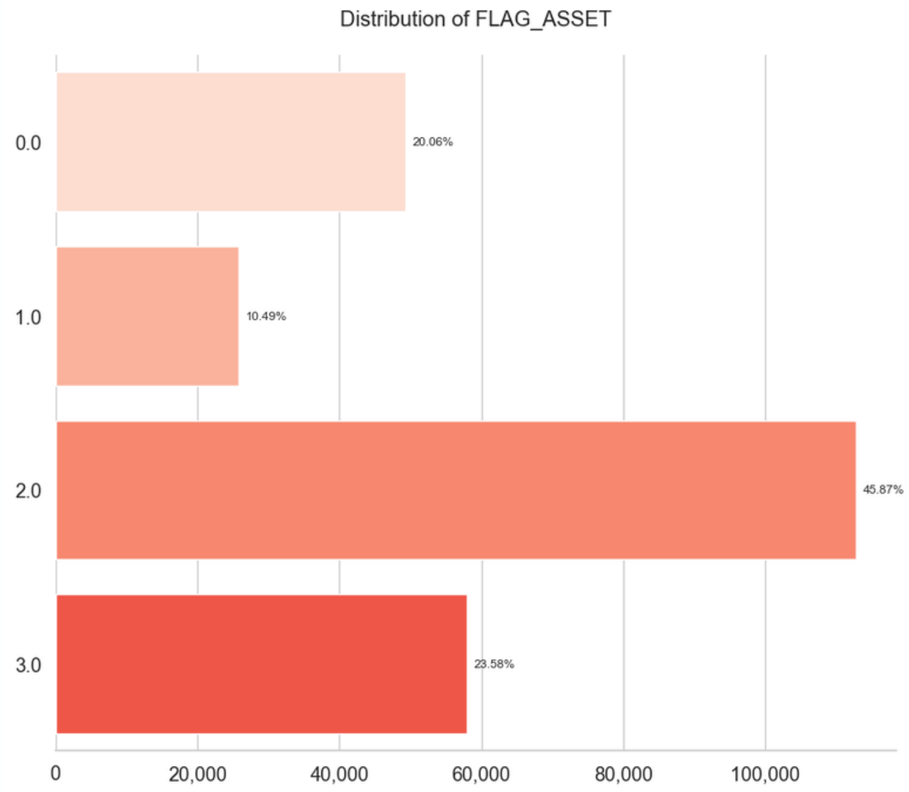
So what is the ability to repay the debt of a customer who **owns a car and real estate**/or **does not own anything**?

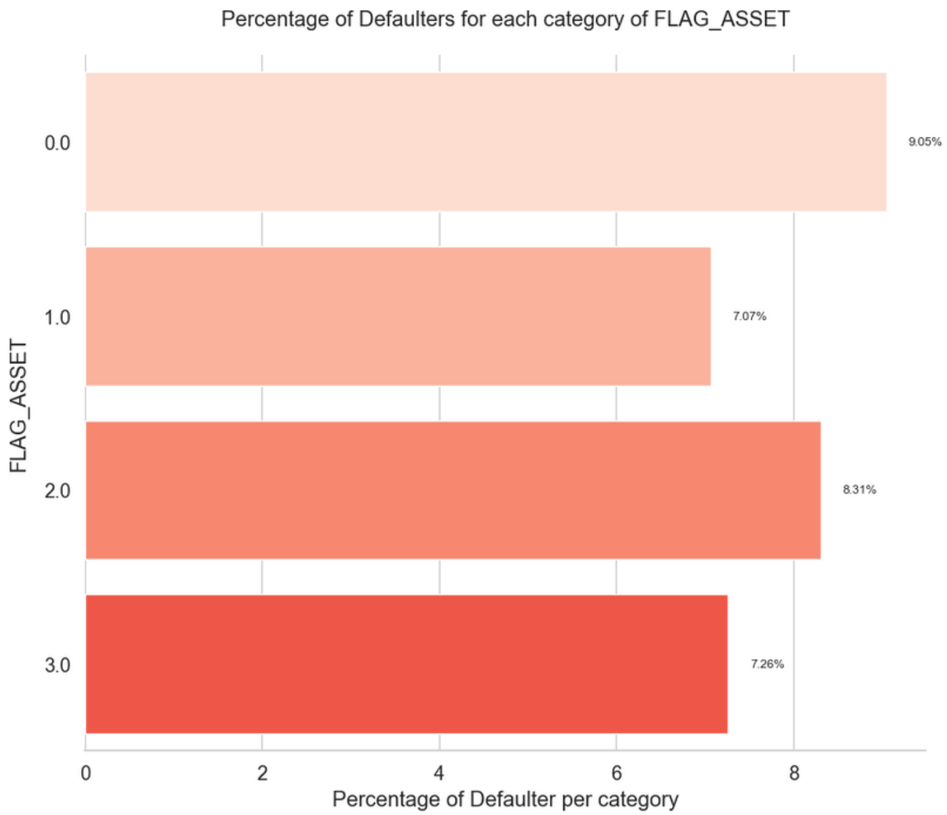
categorical variables - **FLAG_ASSET**

```
def create_flag_asset(df):
    df['FLAG_ASSET'] = np.nan
    filter_0 = (df['FLAG_OWN_CAR'] == 'N') & (df['FLAG_OWN_REALTY'] == 'N')
    filter_1 = (df['FLAG_OWN_CAR'] == 'Y') & (df['FLAG_OWN_REALTY'] == 'N')
    filter_2 = (df['FLAG_OWN_CAR'] == 'N') & (df['FLAG_OWN_REALTY'] == 'Y')
    filter_3 = (df['FLAG_OWN_CAR'] == 'Y') & (df['FLAG_OWN_REALTY'] == 'Y')

    df.loc[filter_0, 'FLAG_ASSET'] = 0
    df.loc[filter_1, 'FLAG_ASSET'] = 1
    df.loc[filter_2, 'FLAG_ASSET'] = 2
    df.loc[filter_3, 'FLAG_ASSET'] = 3
    return df
```

categorical variables - **FLAG_ASSET**

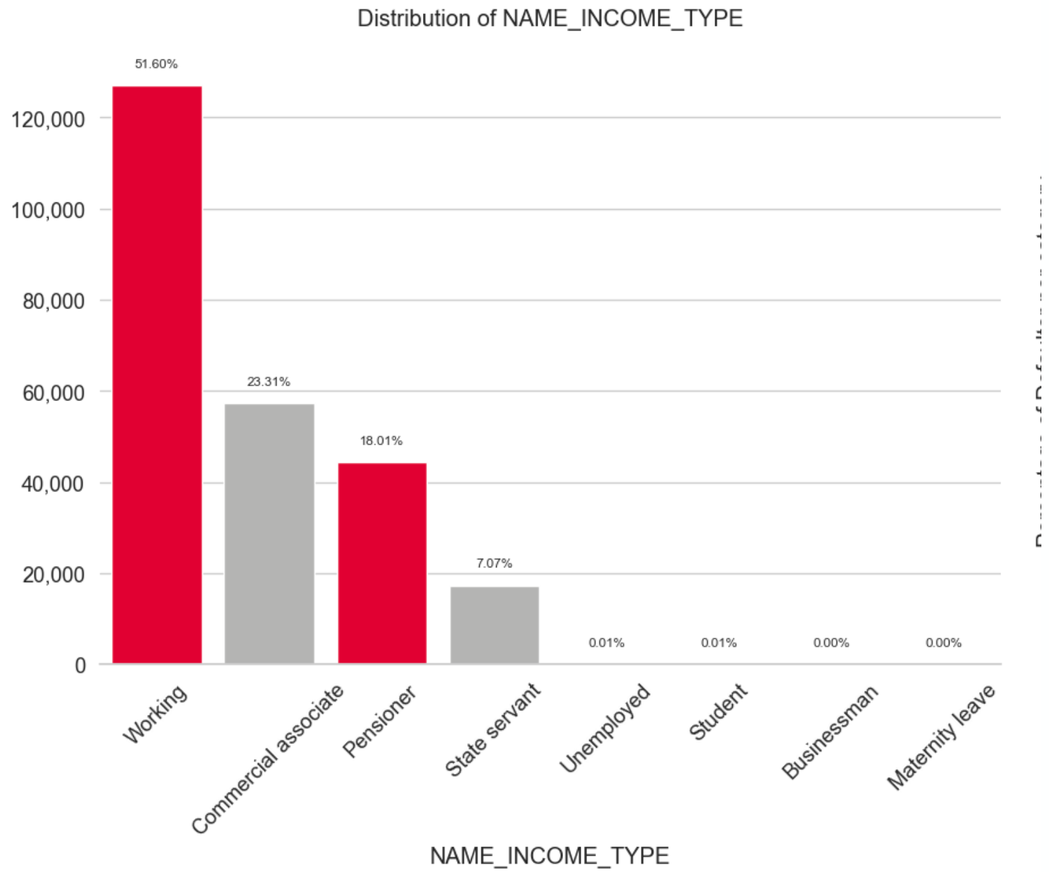


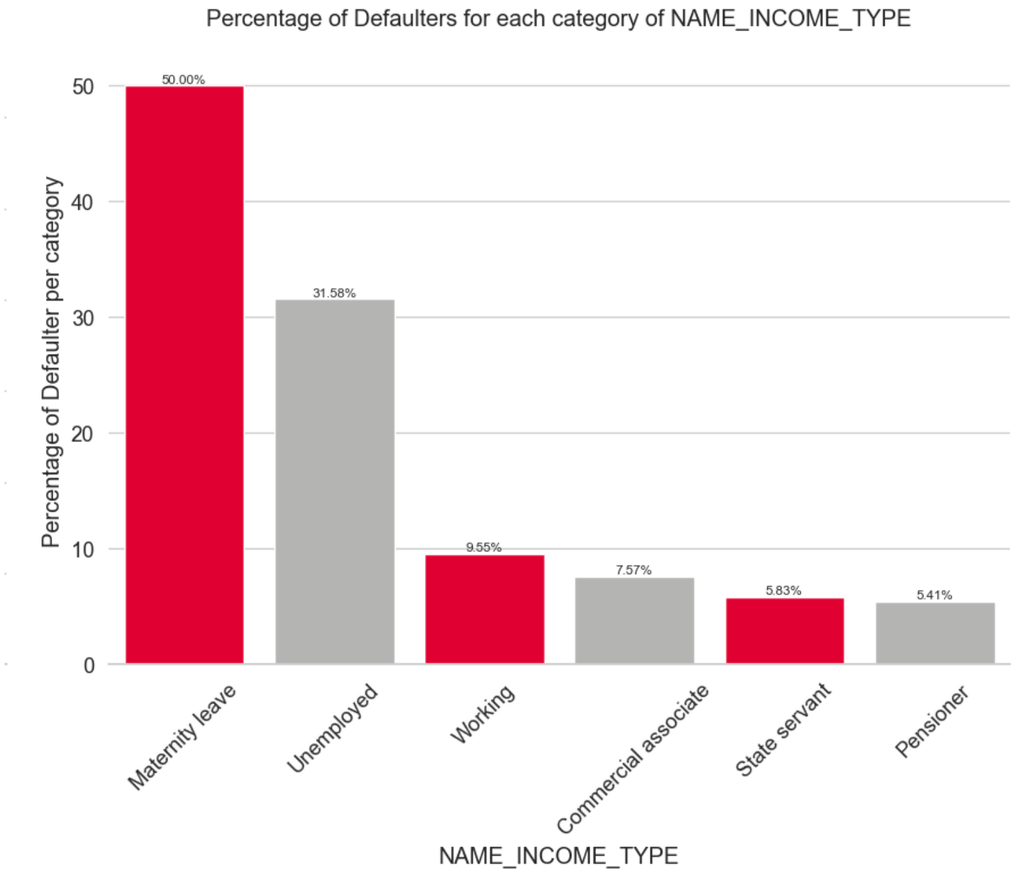


- **Class 0** has the highest rate of not being able to repay the debt on time (this seems understandable) - 9.05%.
- Following that, is **class 2** (does not own a car but owns real estate) - 8.31%.
- Next is **class 3** (owning both) also has a rate of not repaying on time about 7.26%.
- Finally, **class 1** (owns a car but does not own real estate) - 7.07%.

You might consider dropping the variables related to ASSET, keeping the FLAG_ASSET column.

categorical variables - **NAME_INCOME_TYPE**





- Most of the loan applicants' income came from **Working** (51.6%), followed by **Commercial associate** (23.31%), **Pensioner** (18.01%) and **State servant** (7.07%).
- Although applicants with low **Maternity** Leave income category (2 customers), the loan non-repayment rate is 50%. That means of 2 borrowers, 1 of them cannot repay this loan.

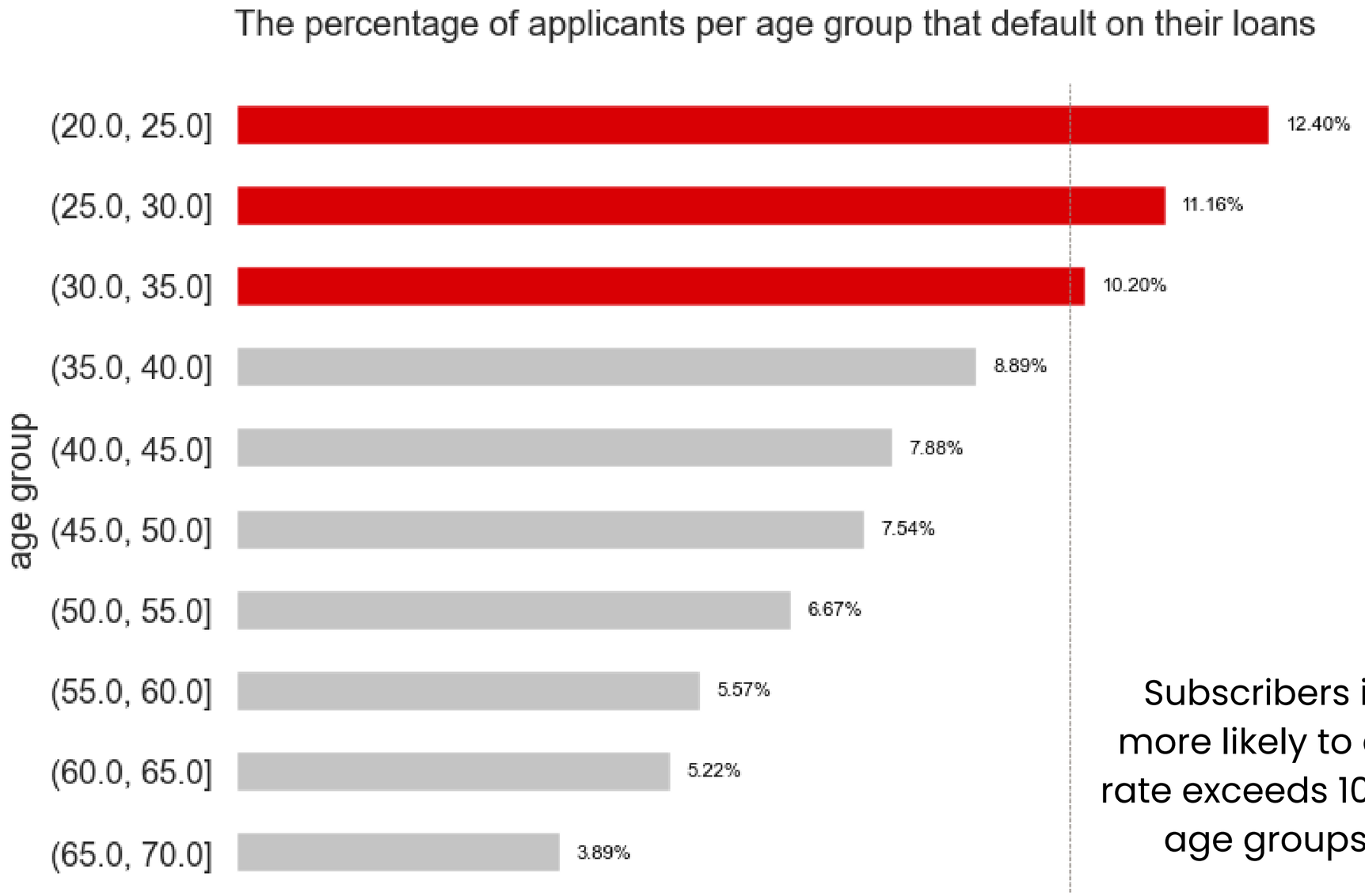
numerical variables - **DAYS_BIRTH**

```
[ ]  age_bin = app_train[['TARGET', 'DAYS_BIRTH']]
     age_bin['YEAR_OLD'] = -app_train['DAYS_BIRTH']/365
```

```
[ ]  age_bin['YEAR_OLD'].describe()
```

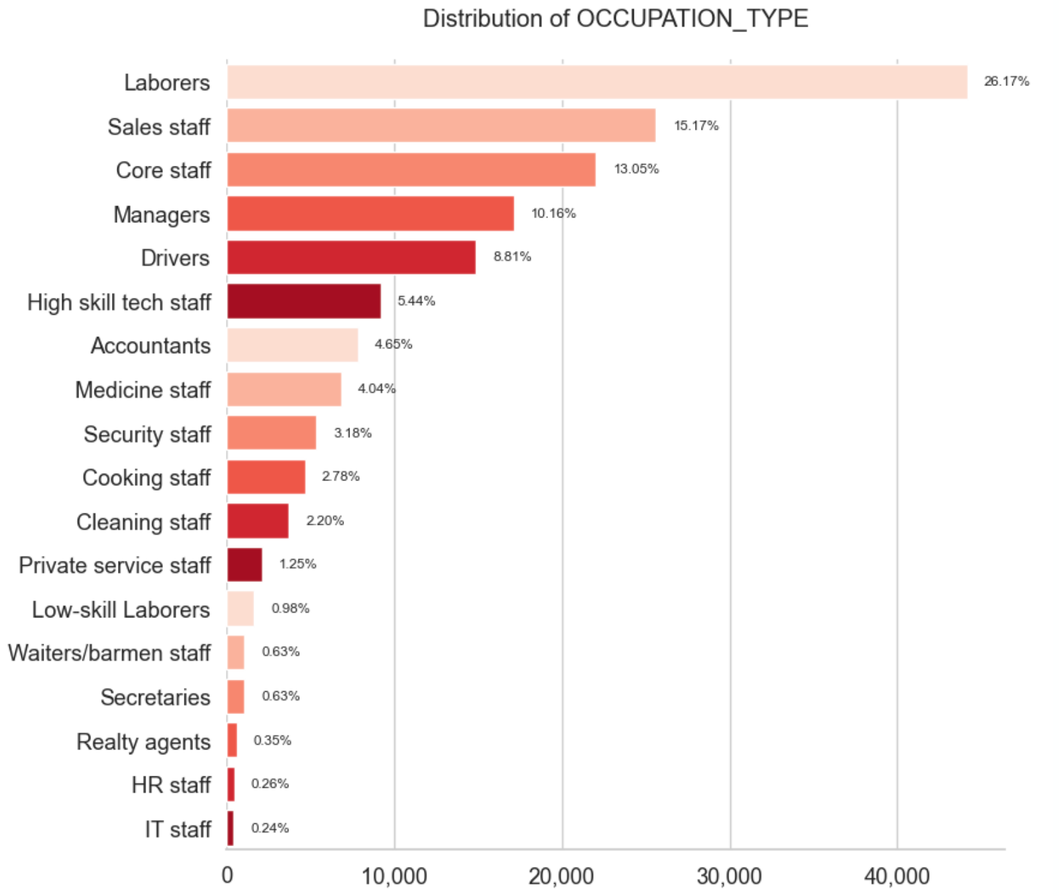
```
count    246009.000000
mean         43.942319
std          11.948673
min          20.517808
25%          34.021918
50%          43.164384
75%          53.923288
max          69.120548
Name: YEAR_OLD, dtype: float64
```

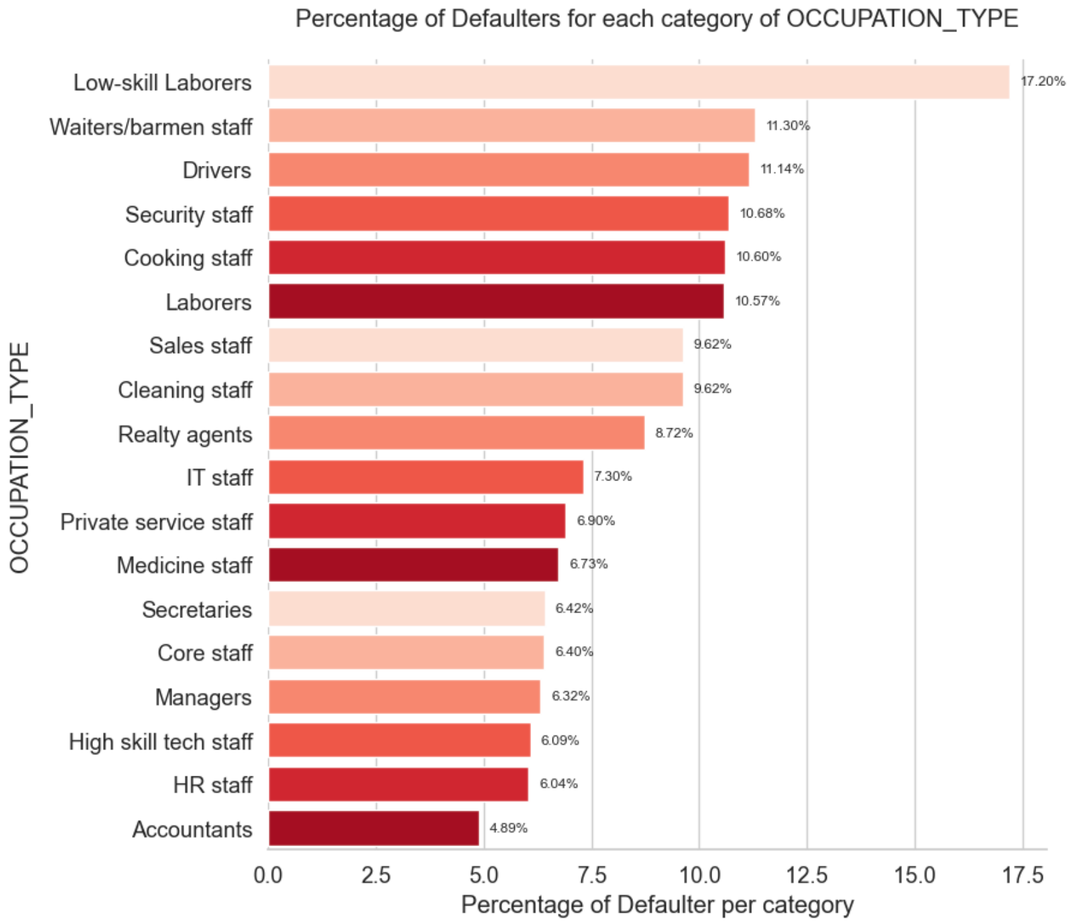
```
[ ]  # Phân chia khoảng tuổi thanh 10 khoảng bằng nhau
     age_bin['DAYS_BIN'] = pd.cut(age_bin['YEAR_OLD'], bins = np.linspace(20, 70, num = 11))
     age_bin.head()
```



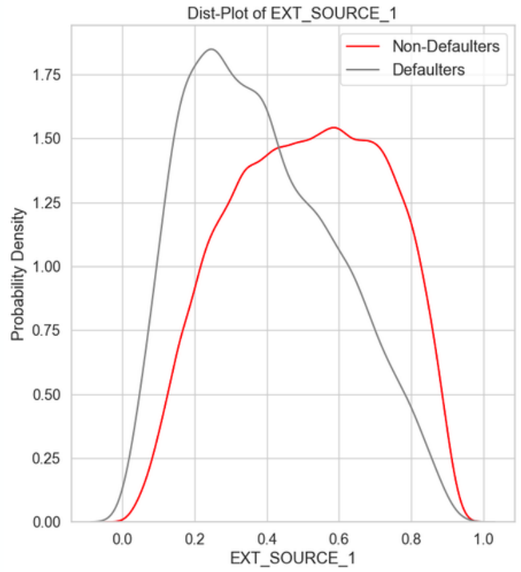
There is a clear trend: Subscribers in the younger group are more likely to default! The debt default rate exceeds 10% for the three youngest age groups and less than 5% for the oldest age group.

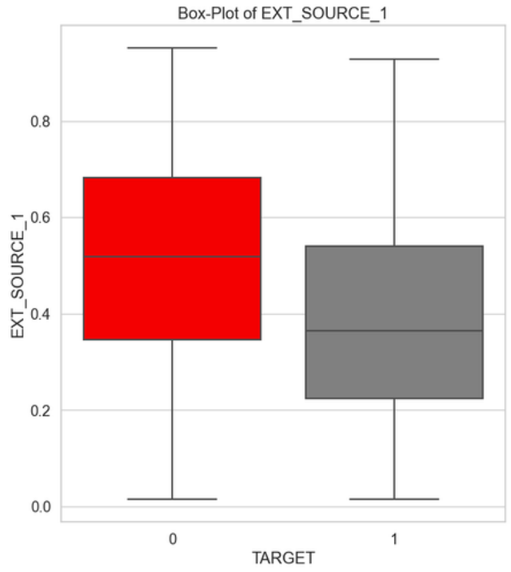
categorical variables - **OCCUPATION_TYPE**

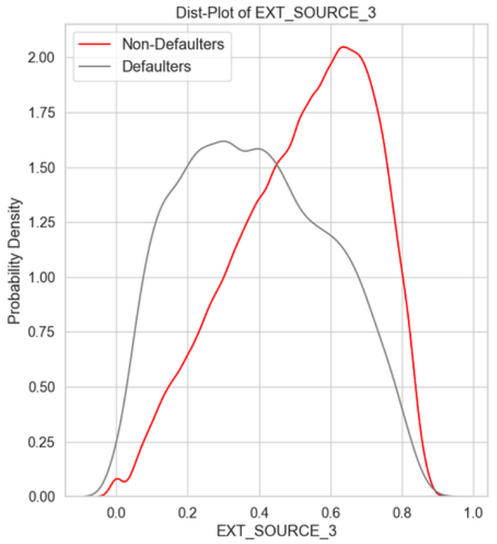


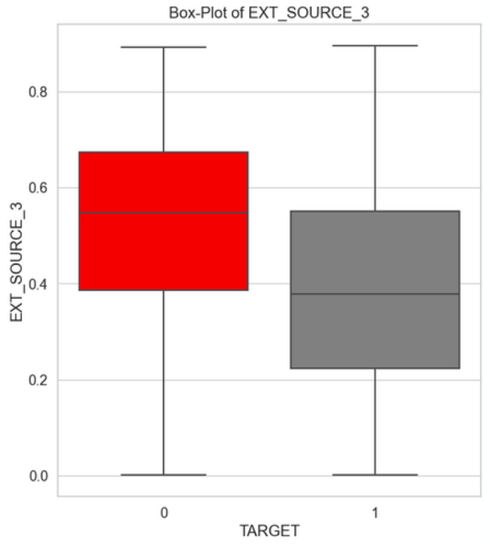


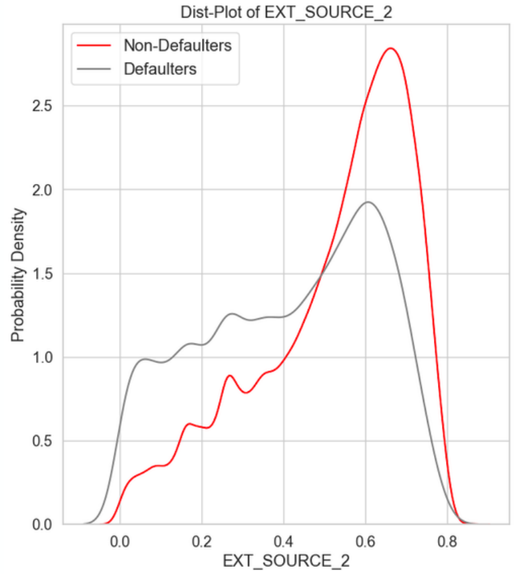
- Most of the loans are taken by **Laborers**, followed by **Sales staff**. **IT staff** take the lowest amount of loans.
- The category with highest percent of not repaid loans are **Low-skill Laborers** (above 17%), followed by **Drivers** and **Waiters/barmen staff**, **Security staff**, **Laborers** and **Cooking staff**.

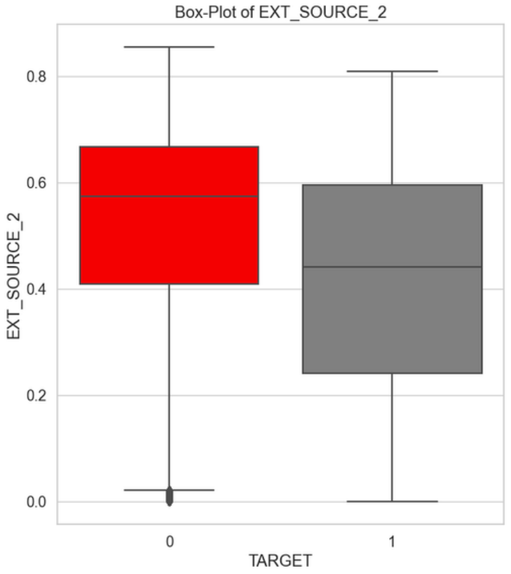

Dist-Plot of EXT_SOURCE_1
Non-Defaulters
Defaulters
Probability Density
EXT_SOURCE_1


Box-Plot of EXT_SOURCE_1
EXT_SOURCE_1
TARGET


Dist-Plot of EXT_SOURCE_3
Non-Defaulters
Defaulters
Probability Density
EXT_SOURCE_3


Box-Plot of EXT_SOURCE_3
EXT_SOURCE_3
TARGET


Dist-Plot of EXT_SOURCE_2
Non-Defaulters
Defaulters
Probability Density
EXT_SOURCE_2

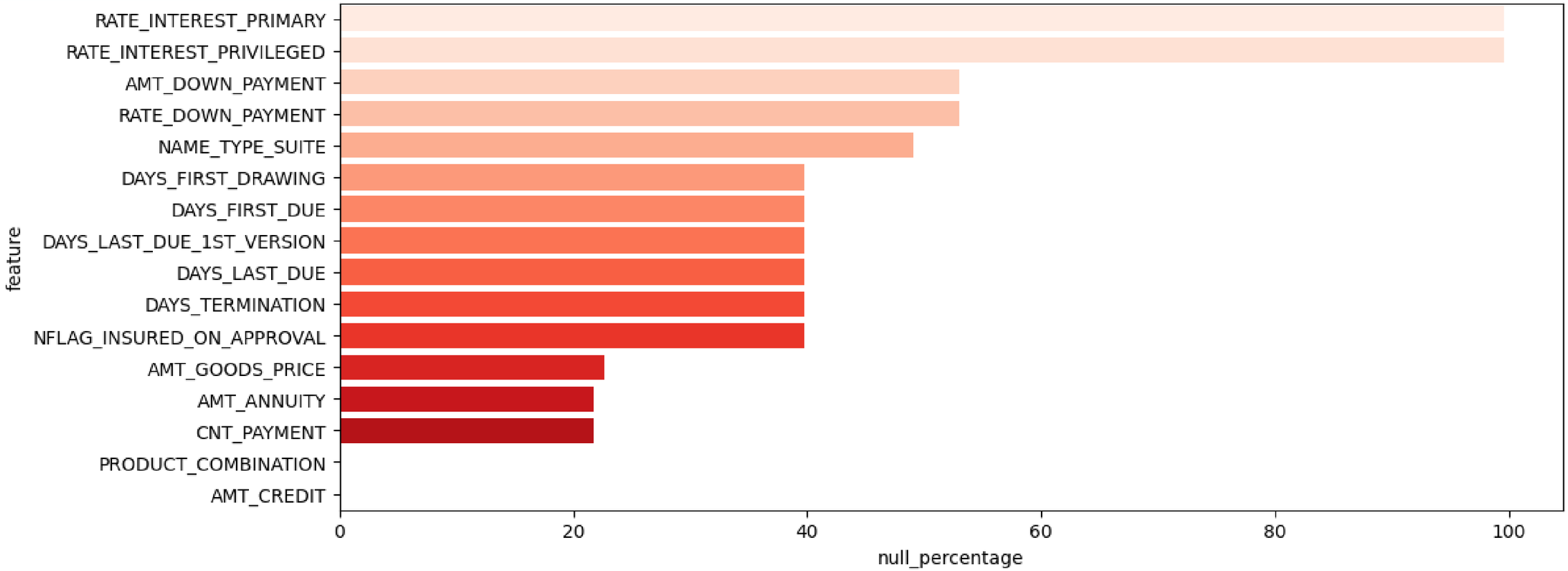

Box-Plot of EXT_SOURCE_2
EXT_SOURCE_2
TARGET

EXT_SOURCE_X

PREVIOUS APPLICATION

- Application of client's previous loans in Home Credit
- Info about previous loan parameters and client info at time of previous application
- One row per previous application

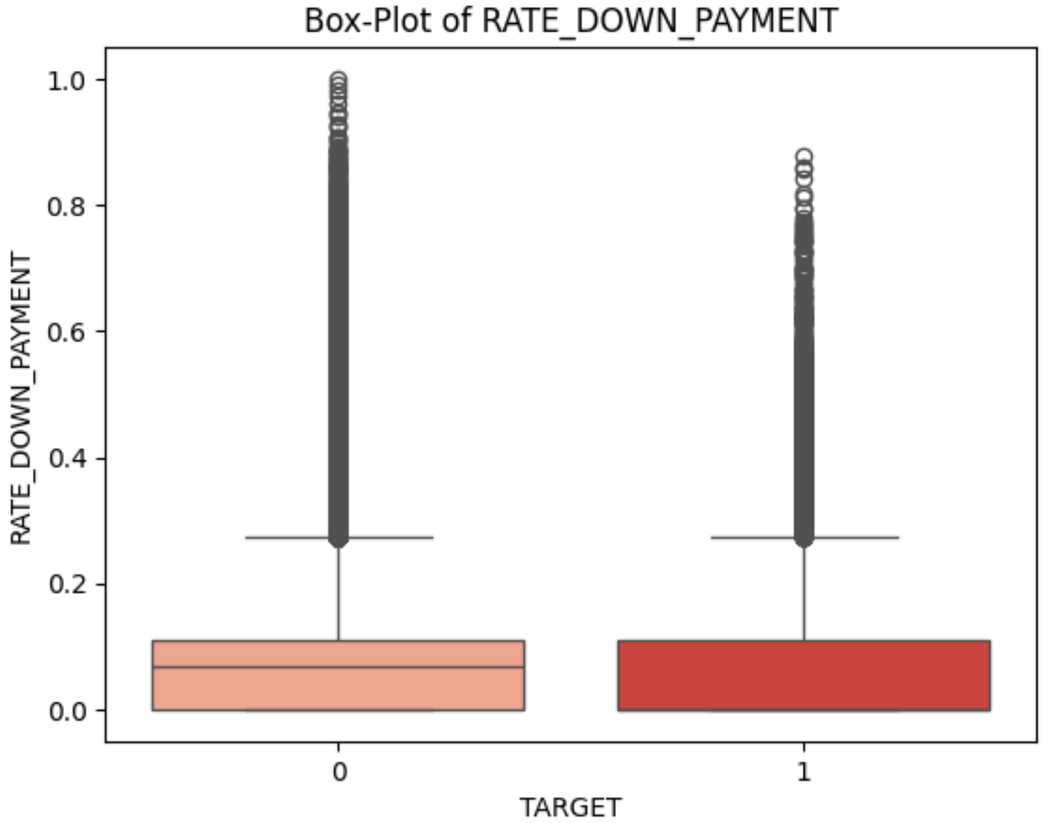
null percentage



- **RATE_INTEREST_PRIMARY** and **RATE_INTEREST_PREVILEDGED** are **99%** null. They would be dropped from the dataframe while proccessing
- The remaining features contain null would be filled with suitable methods

numerical features

Non-defaulter		Defaulter	
count	490355.000000	count	40553.000000
mean	0.081100	mean	0.070795
std	0.108688	std	0.099221
min	-0.000015	min	0.000000
25%	0.000000	25%	0.000000
50%	0.067511	50%	0.000000
75%	0.108914	75%	0.108909
max	1.000000	max	0.878301

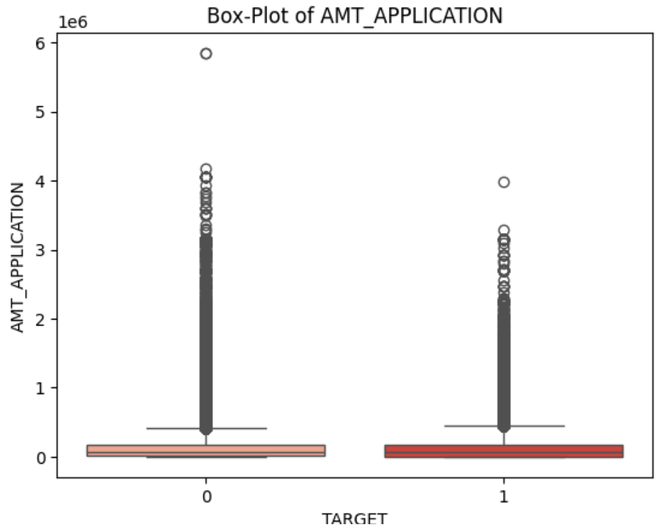


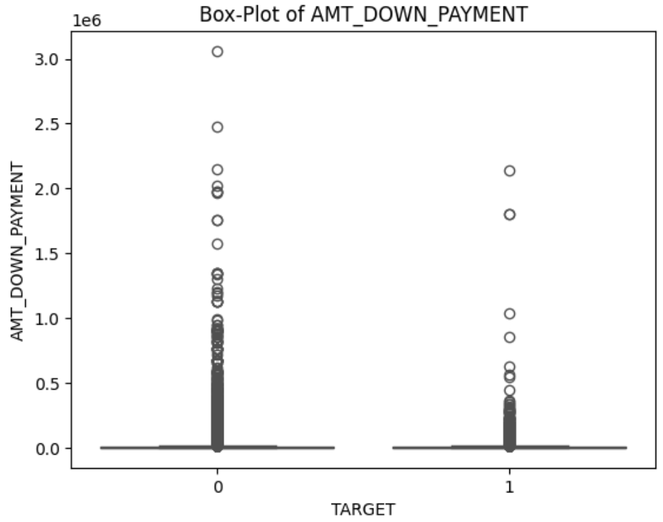
insights:

- Looking at the boxplots and the quantile, it seems that customers with lower **RATE_DOWN_PAYMENT** have more potential to become defaulters.

numerical features

The ratio of customers with AMT_APPLICATION = 0 in the defaulter group is significantly higher than in the non-defaulter group. Thus, customers with **low AMT_APPLICATION** are often **defaulters**. This is also true with AMT_DOWN_PAYMENT





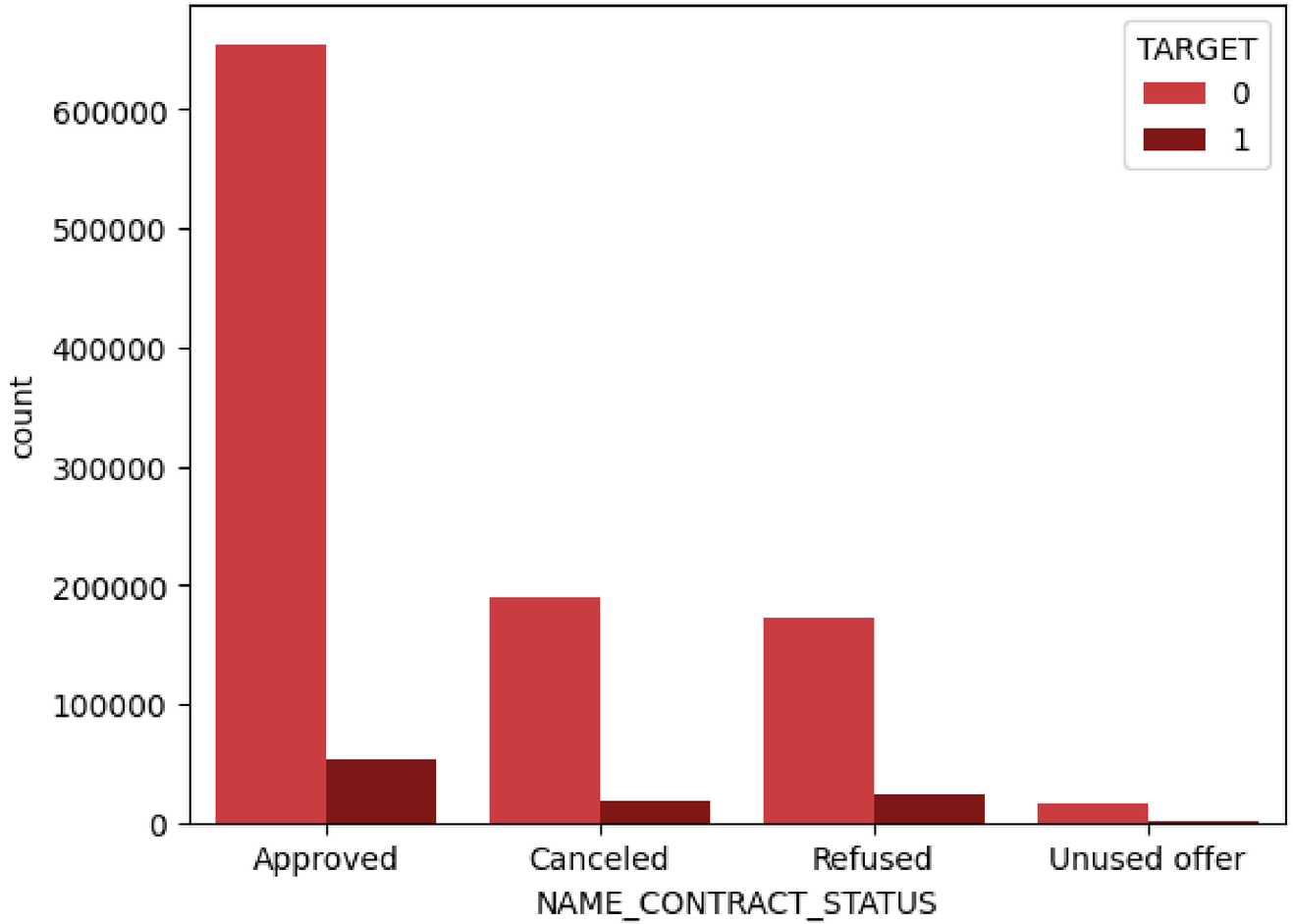
Non-defaulters: count		Defaulters: count	
mean	1.759222e+05	mean	1.691095e+05
std	2.946799e+05	std	2.849821e+05
min	0.000000e+00	min	0.000000e+00
25%	2.020500e+04	25%	0.000000e+00
50%	7.166700e+04	50%	6.606000e+04
75%	1.800000e+05	75%	1.800000e+05
max	5.850000e+06	max	3.982500e+06

Non-defaulters: count		Defaulters: count	
mean	6.784889e+03	mean	5.470812e+03
std	2.125049e+04	std	2.267898e+04
min	-9.000000e-01	min	0.000000e+00
25%	0.000000e+00	25%	0.000000e+00
50%	1.809000e+03	50%	0.000000e+00
75%	7.915500e+03	75%	6.071400e+03
max	3.060045e+06	max	2.135700e+06

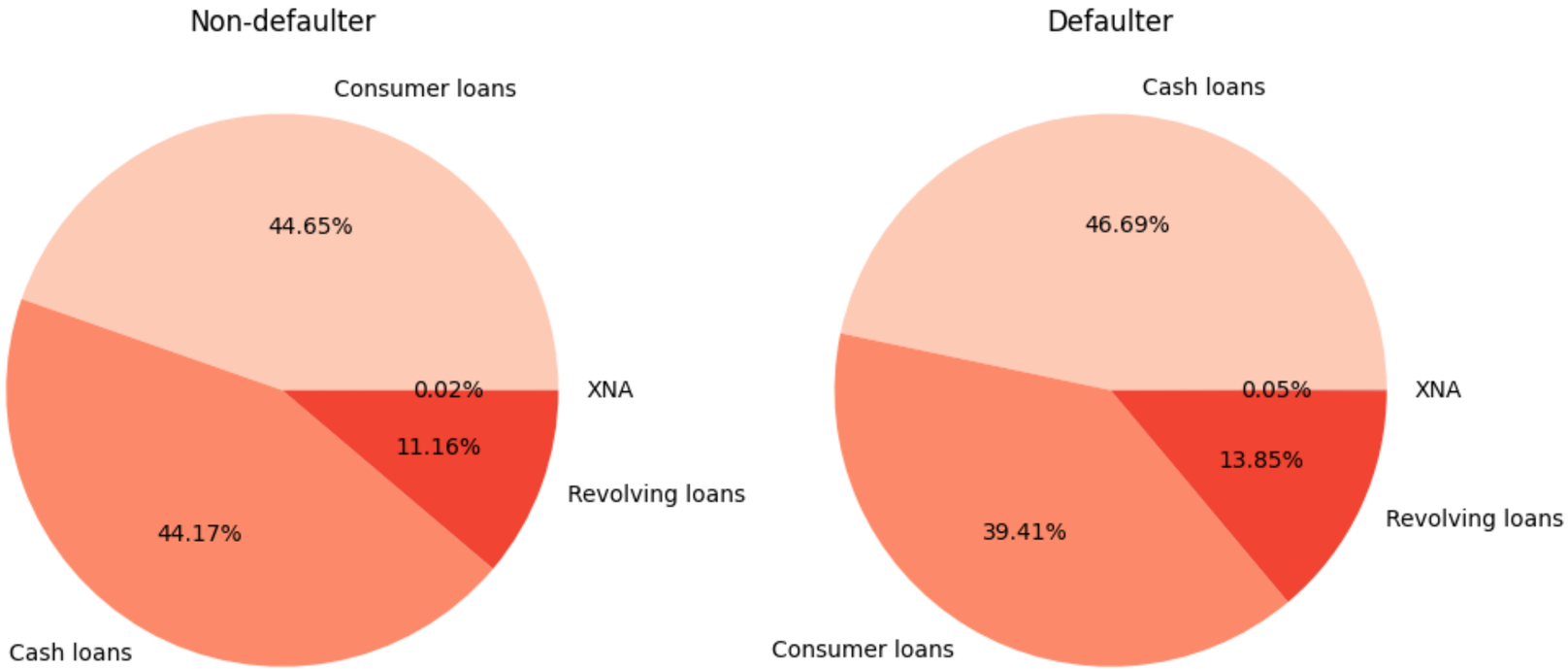
categorical variables

insights:

- Customers who has previous loan approved are more likely to pay their loan on time compared to who refused.
- Default rate of customers who previous application was approved was around 7%.
- Non-default rate of customers who previous application was refused was approximately 89%.



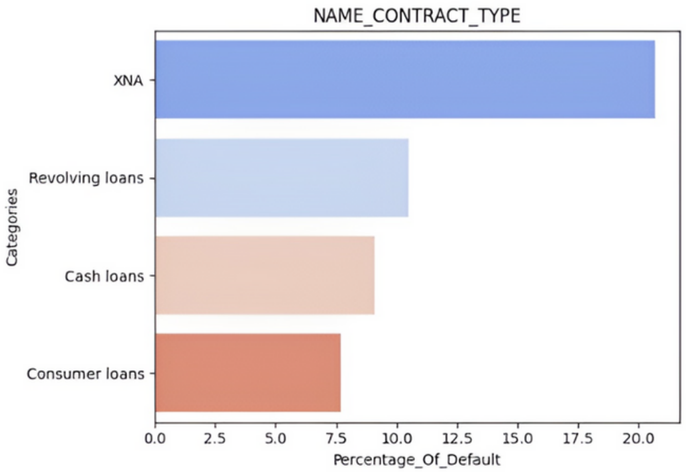
categorical variables

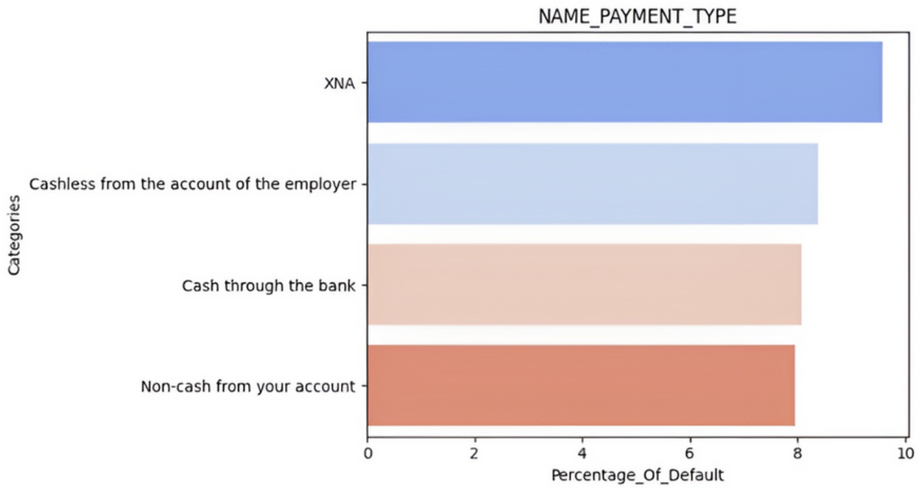


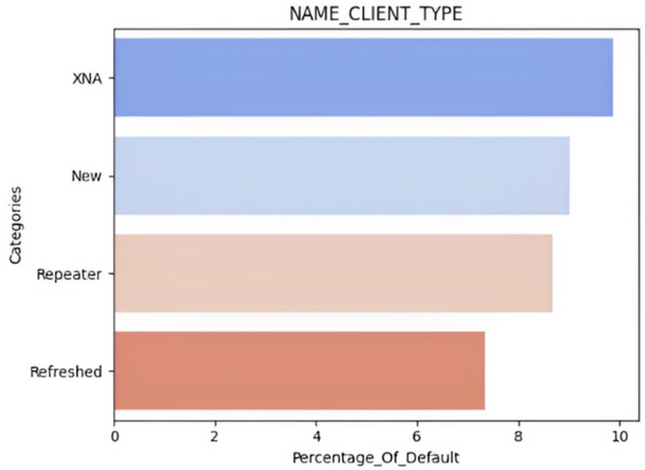
insights:

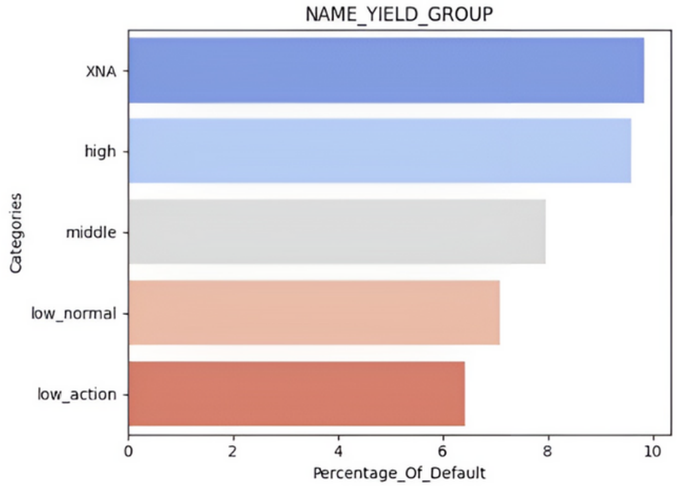
- Defaulters tends to have **NAME_CONTRACT_TYPE** is cash loans.

percentage of **default per category**





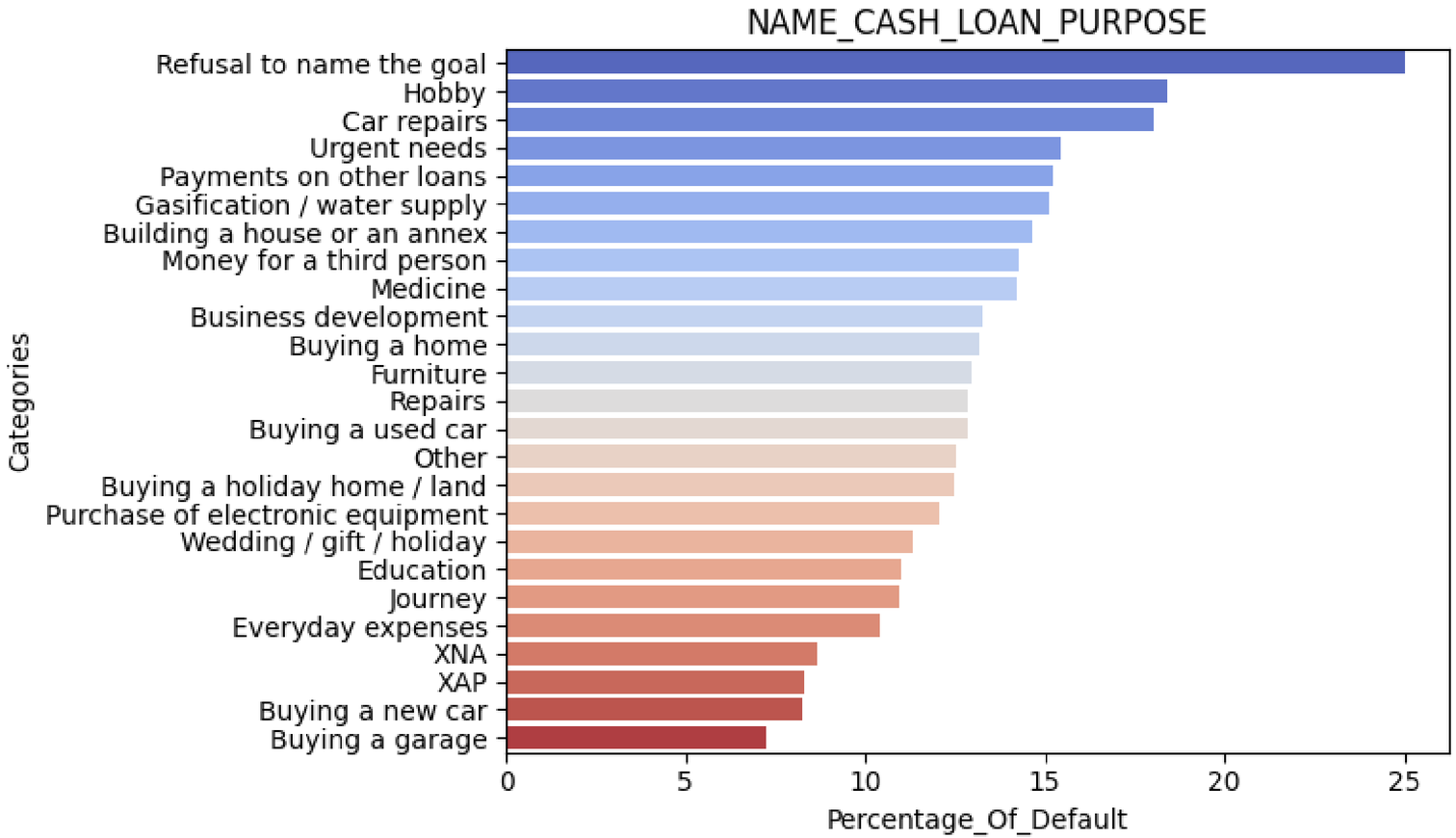




insights:

XNA always account for a large percentage of defaulters. It seems that customer with many XNA values included in the features tend to be defaulters.

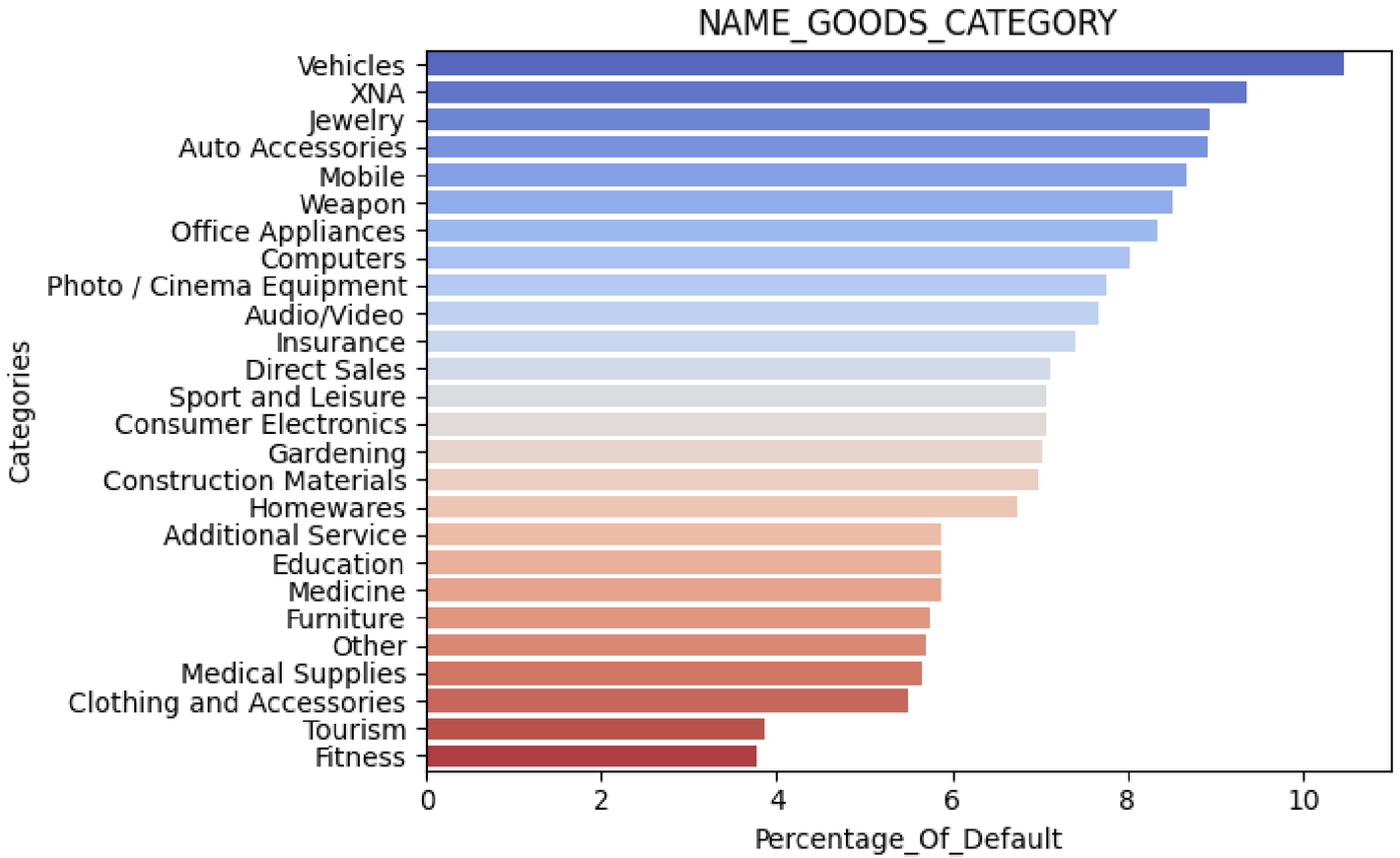
percentage of **default per category**



insights:

Customers refuse to name the goal of loan are the most likely to become the defaulters.

percentage of **default per category**



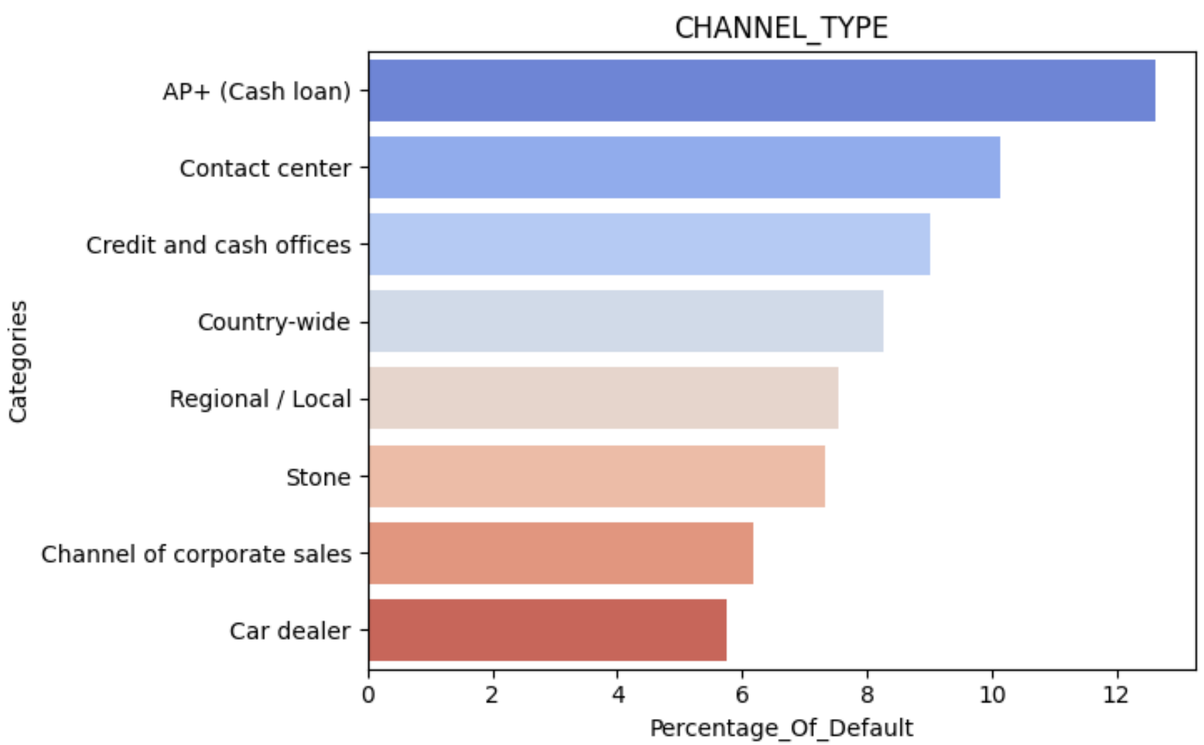
insights:

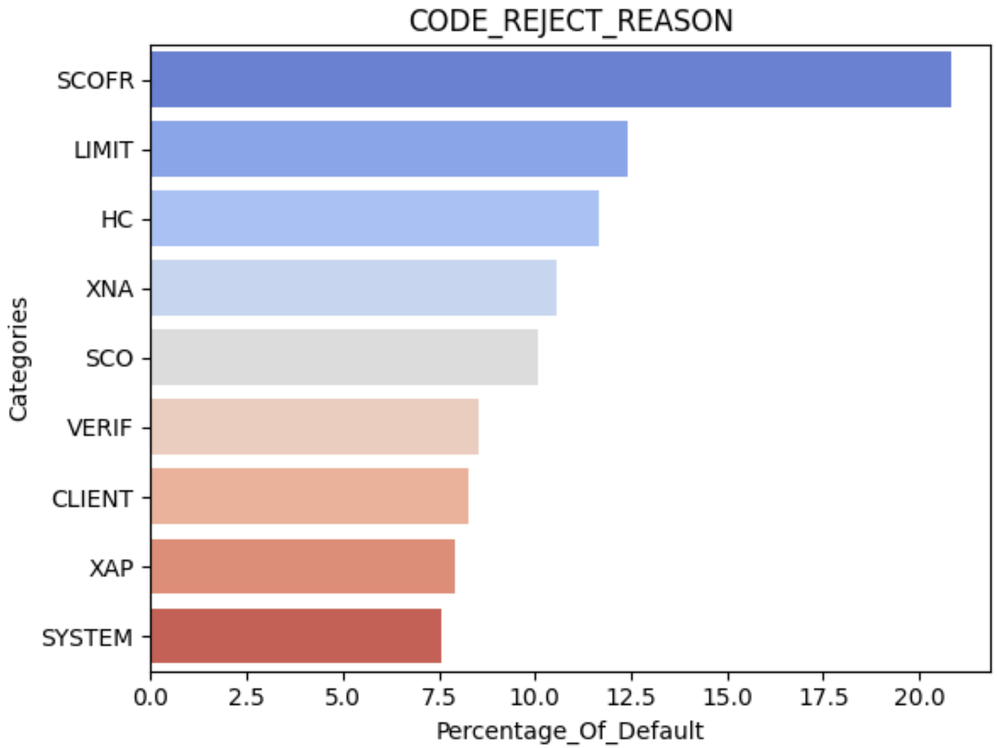
People apply to **buy Vehicles** are the most likely to become defaulters.

percentage of **default per category**

insights:

- Customers who is rejected because of **SCOFR** are the most likely to become defaulters.
- Customers who has channel_tyle are **AP+** are the most likely to become defaulters.

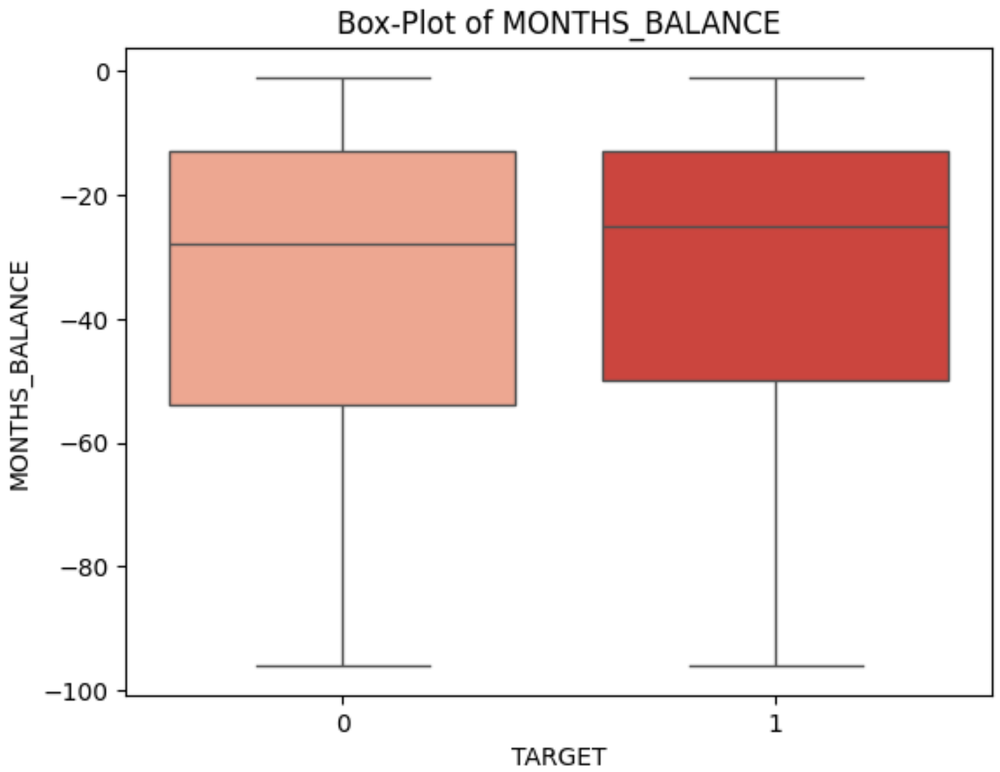


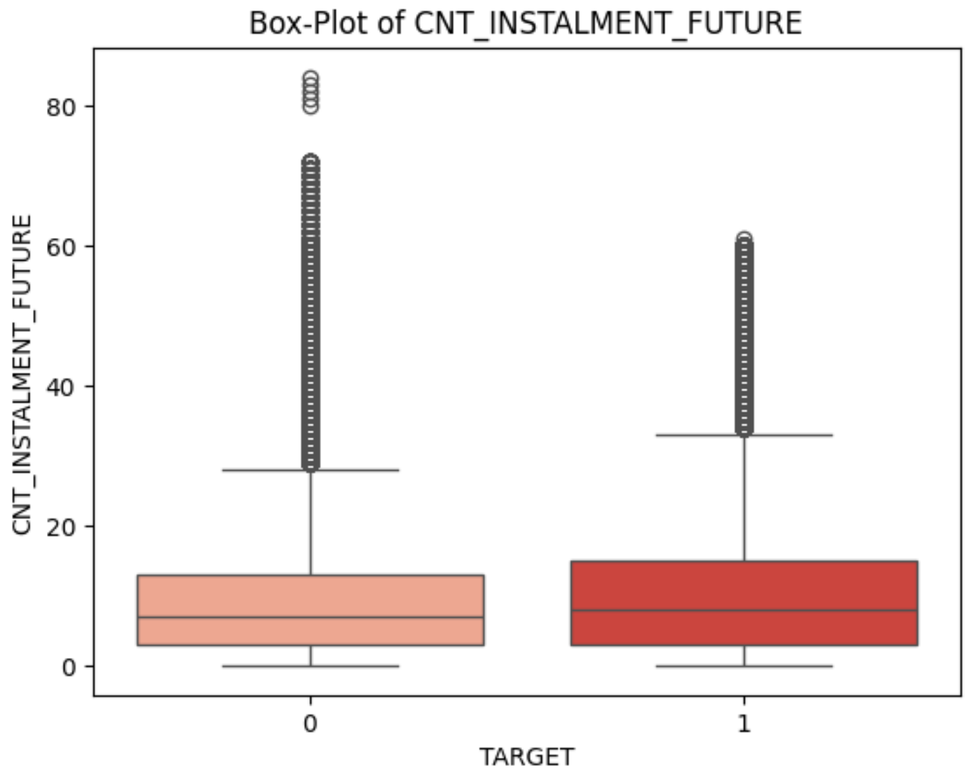


POS_CASH BALANCE

- Monthly balance of client's previous loans in Home Credit
- Behavioral data

numerical variables





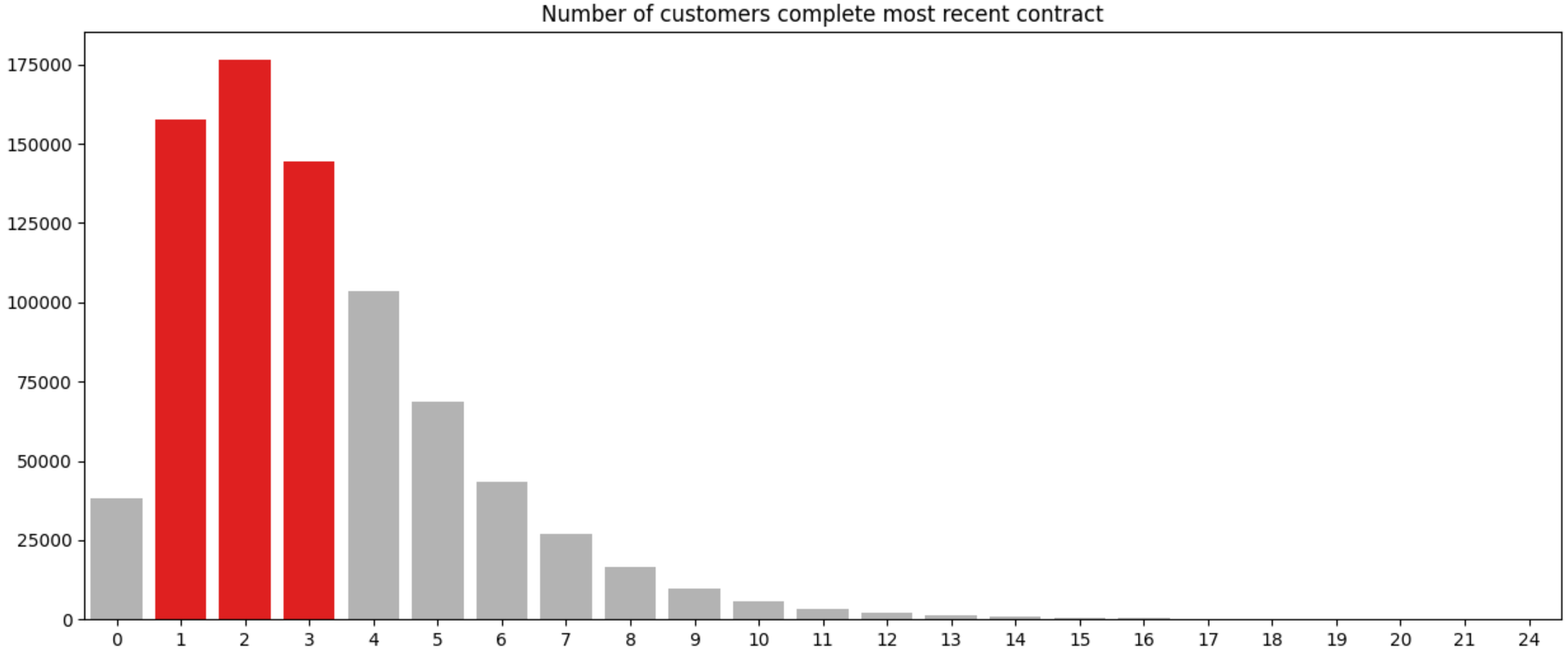
insights:

- According to the boxplots, Defaulters tend to have shorter MONTH_BALANCES compare to non-defaulter. Which means they seems to make loan more frequently.
- Defaulters also have higher CNT_INSTALMENT_FUTURE (installments left to pay on the previous credit) than non-defaulters.

number of customers

complete most recent contract

- Most of the customers complete 2 payments, follows by 1 and 3 payments.
- The higher the number of the payments, the lower the number of customers can complete it.



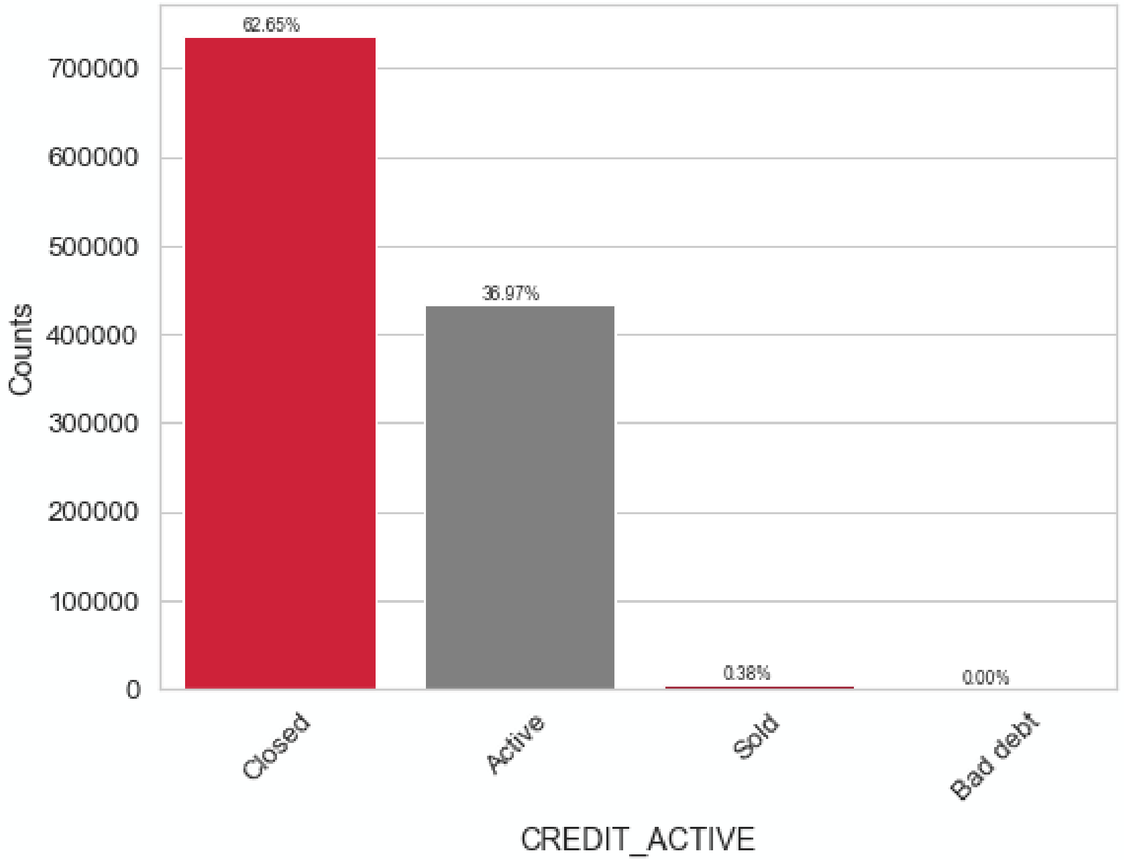
GROUP 2

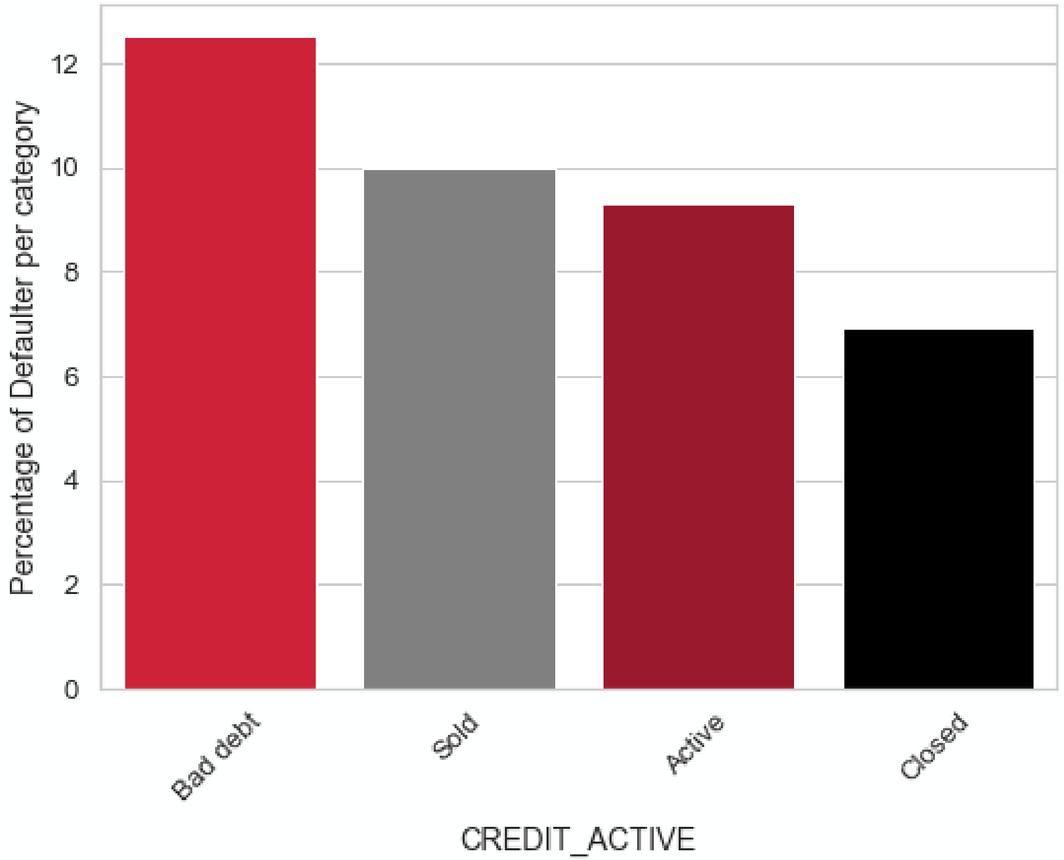
BUREAU & BUREAU BALANCE

More Info

CREDIT_ACTIVE variable

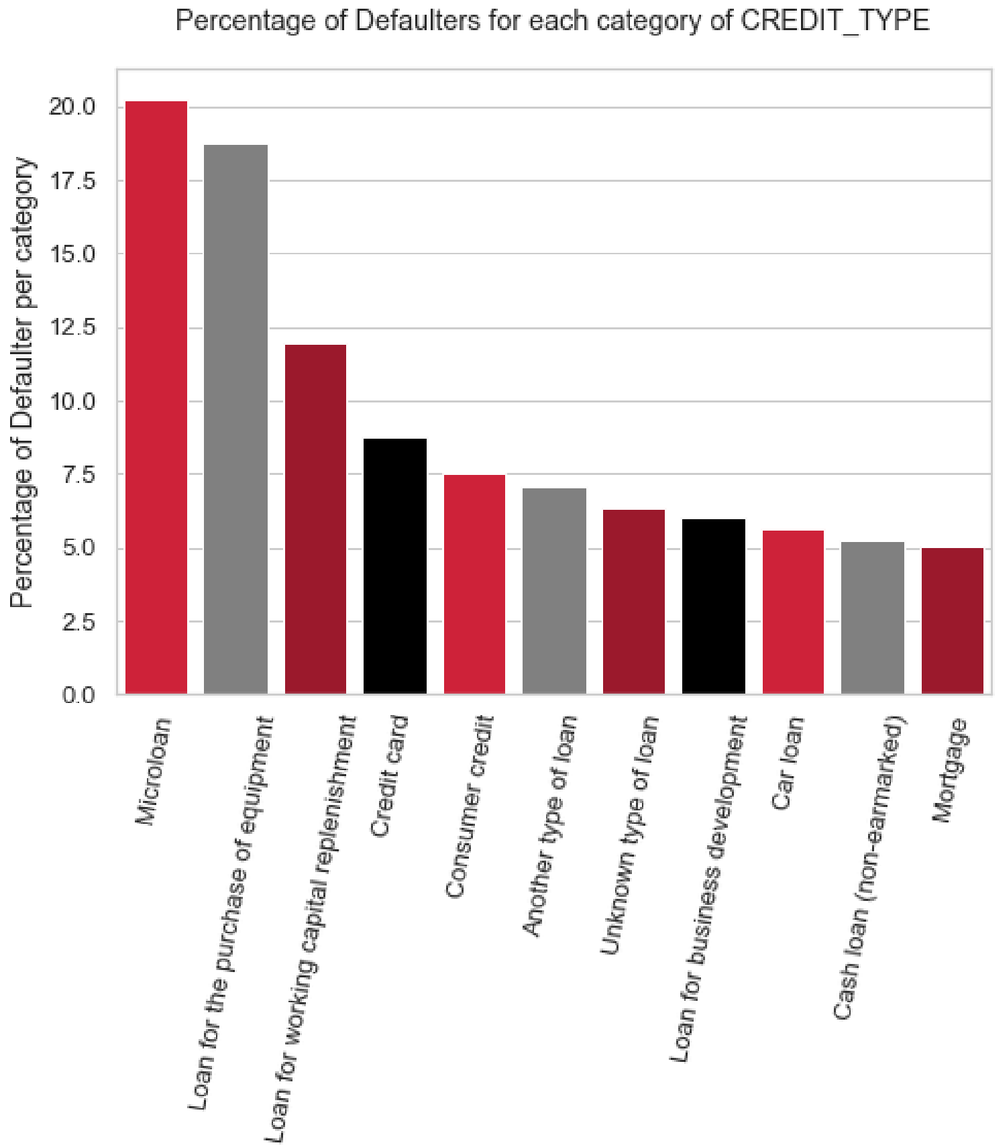
The higher the '**Bad debt**', '**Sold**' and '**Active**' loans a customer has, the higher his or her chances of becoming a defaulter.





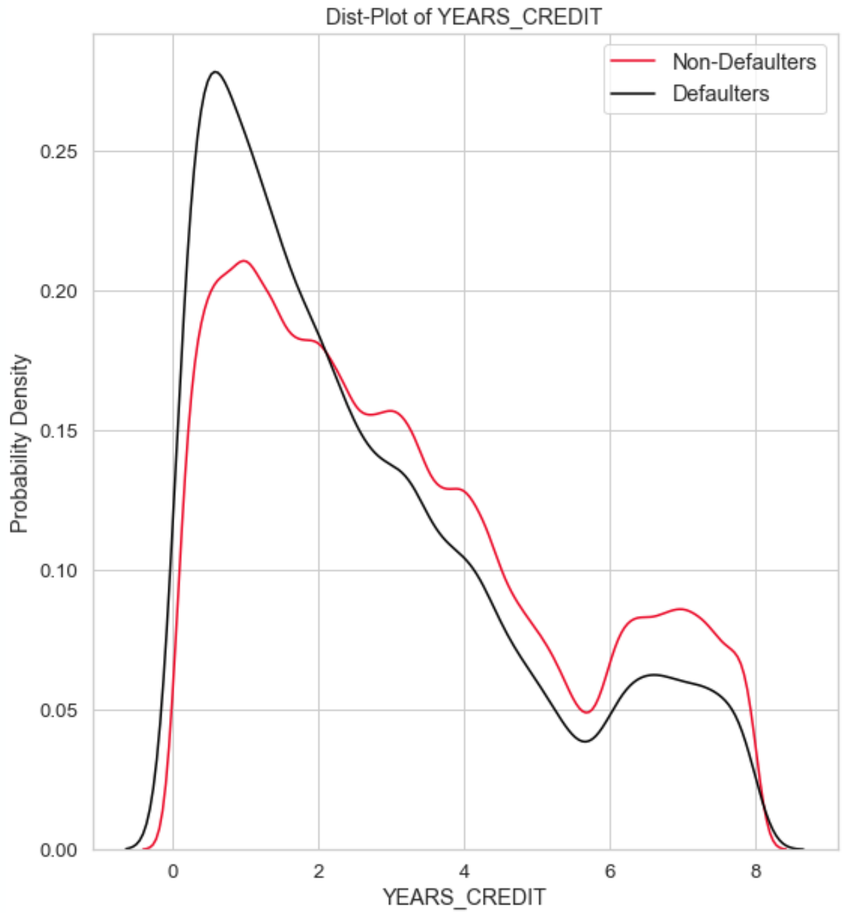
CREDIT_TYPEE variable

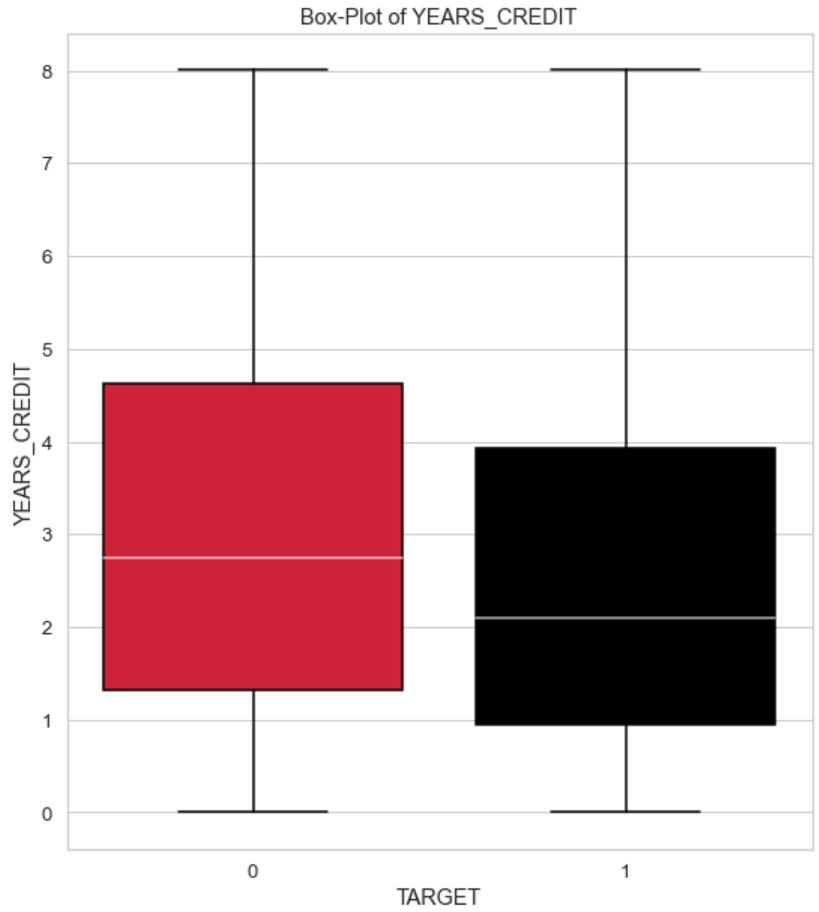
- Microloans and Loan for the purchase of equipment has very high percentage of defaulters.
- Credit card loan has higher percentage of defaulter than consumer credit.
- Car loans and mortgages appear to be relatively low risk.



DAYS_CREDIT analysis

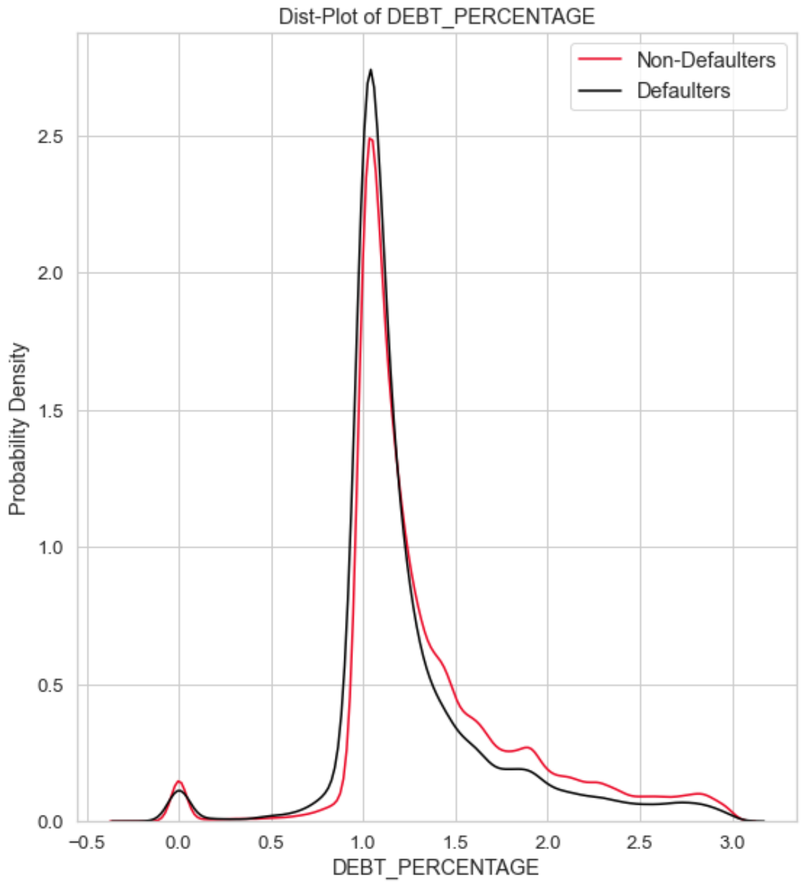
- From this plot, we observe that the Non-Defaulters usually have longer DAYS_CREDIT as compared to Defaulters. The Defaulters have a higher Peak in PDF in lower YEARS_CREDIT range of values.

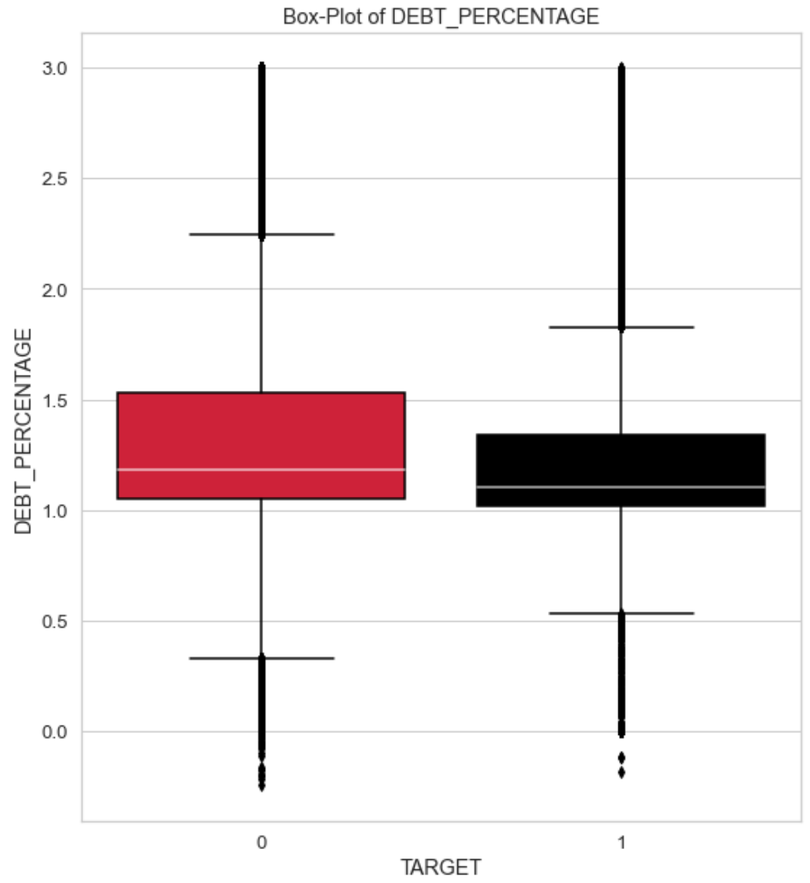




$$\textbf{DEBT\_PERCENTAGE} = \frac{\text{AMT\_CREDIT\_SUM}}{\text{AMT\_CREDIT\_SUM\_DEBT}}$$

- Non-Defaulters usually have higher AMT_CREDIT_SUM/AMT_CREDIT_SUM_DEBT ratio than Defaulters.

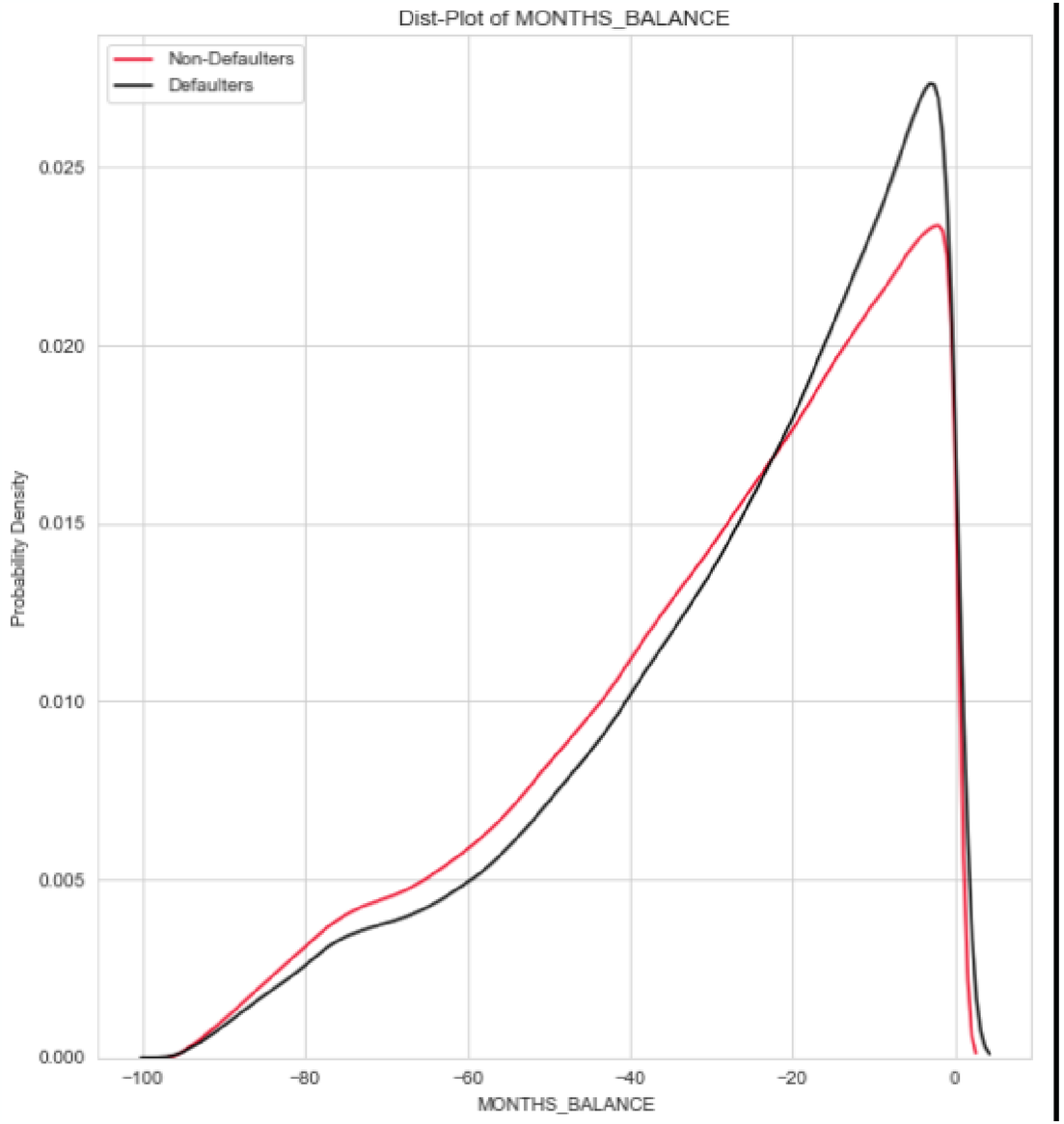




MONTHS_BALANCE variable

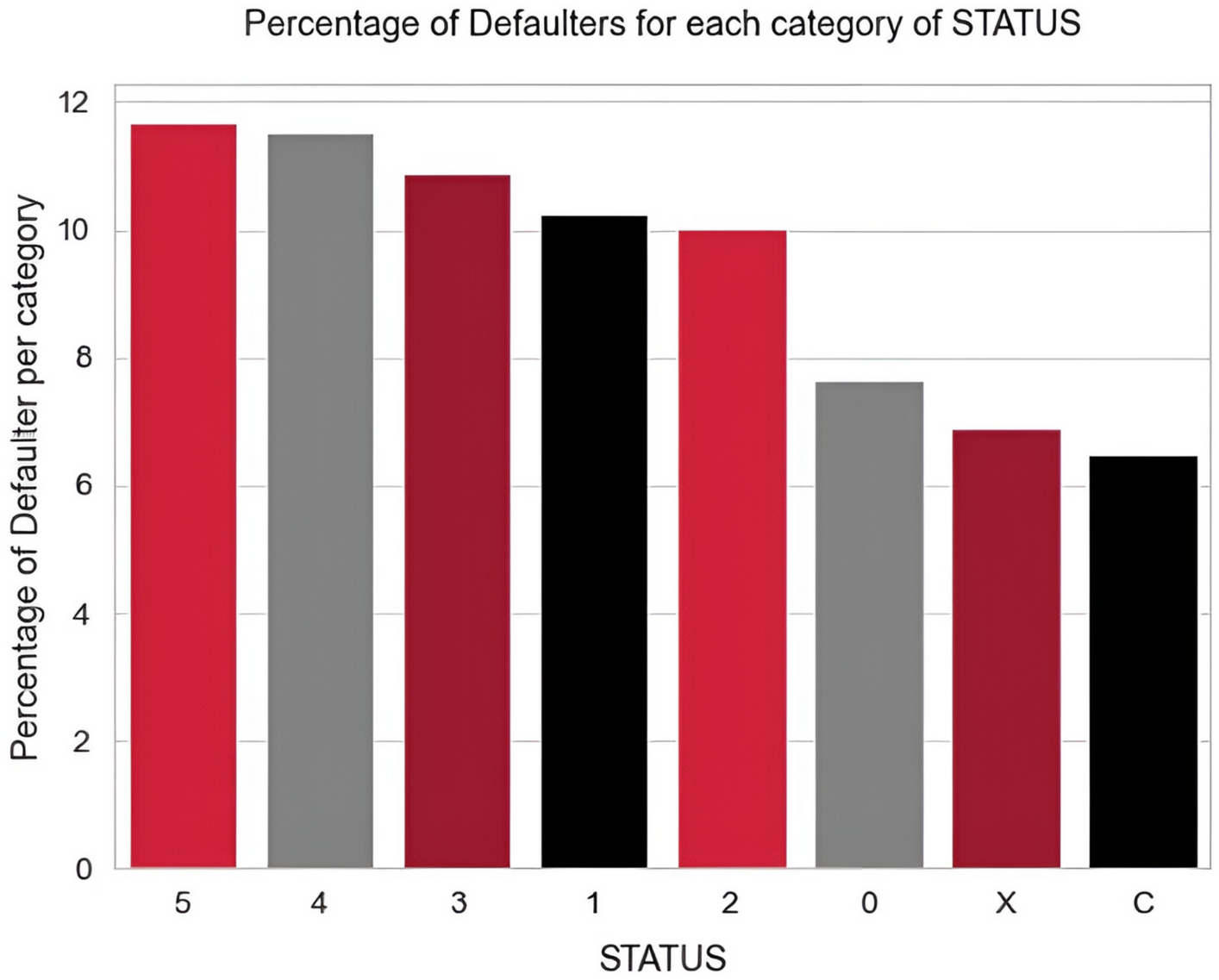
Both defaulters and non-defaulters have MONTHS_BALANCE distributed mainly over a period of < 20 days. With MONTHS_BALANCE <-20, non-defaulters have greater density than defaulters.

Overall, non-defaulters are more likely to have long credit history than defaulters.



STATUS variable

- **Defaulters** are more likely to repay their loan overdue than **Non-Defaulter**

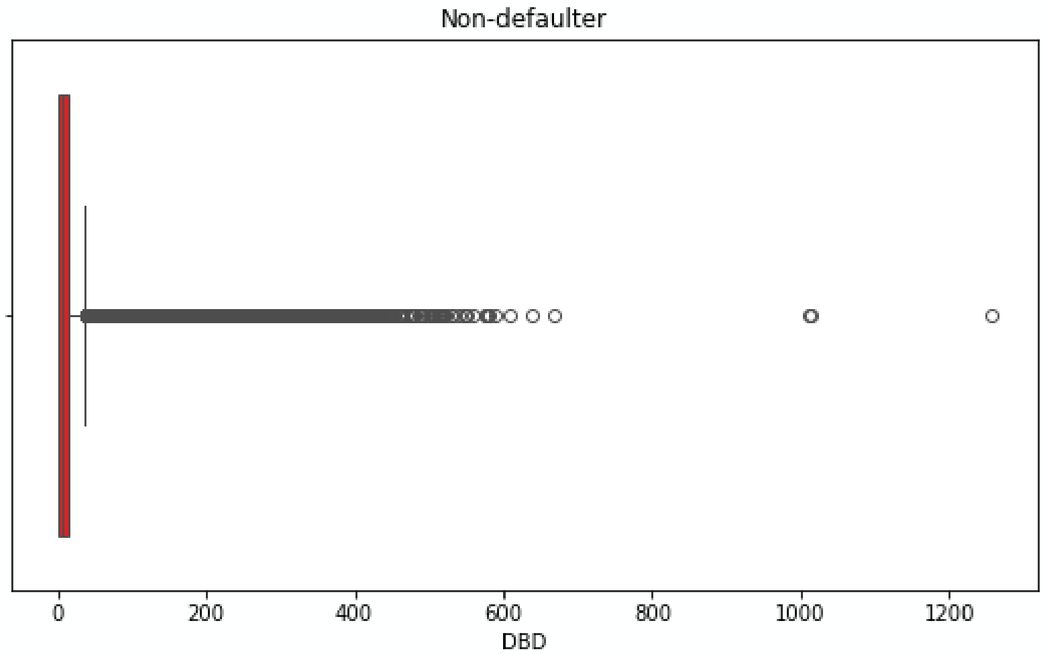


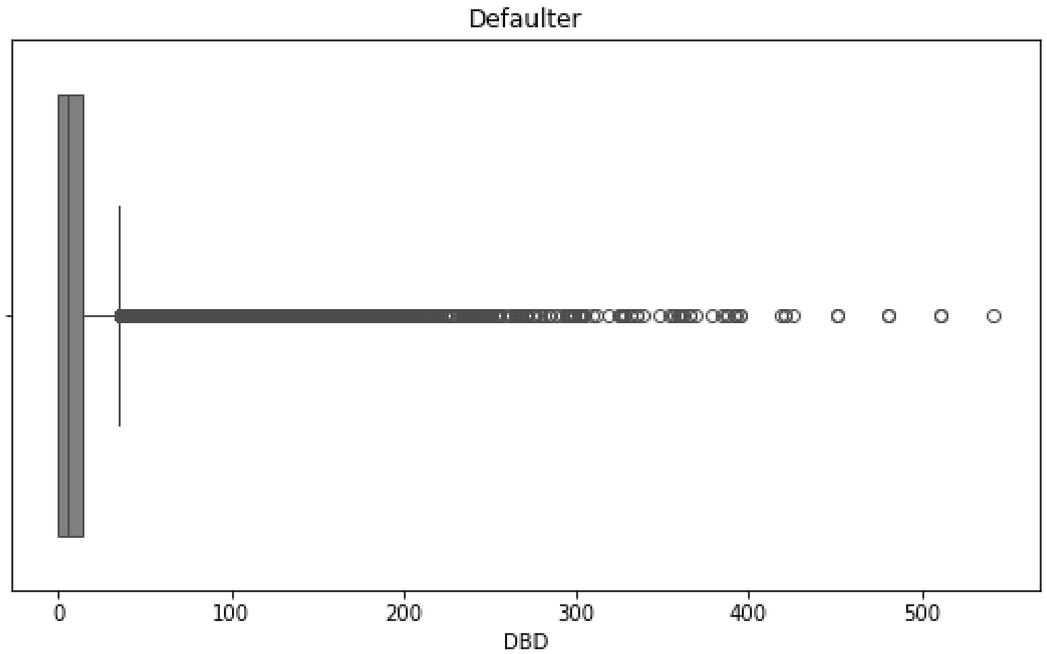
Installment Payment Agreement	
This installment payment agreement is made on this day _____ (give date) between _____ _____ (name of borrower and his address) and _____ _____ (name of lender and his address).	
The borrower owes the lender a sum of _____ bearing a simple interest of _____ per annum.	
The borrower shall make a payment of _____ to the lender within the first 10 days of every month from a period of _____ to _____, resulting in _____ installments. Failure to do so will lead to a late charge of _____ in addition to the installment amount and interest amount.	
The borrower shall have the right to prepay the amount at any time, the sum of which shall include the principal amount and the interest	
If the entire loan amount is not paid in full within the time period mentioned above, legal action will be taken against the borrower.	
Signature of borrower	Signature of lender
Date	

INSTALMENT PAYMENT TABLE

- **History record of loan payment (Time and Amount)**
- **Plan vs Actual**

on time or not?



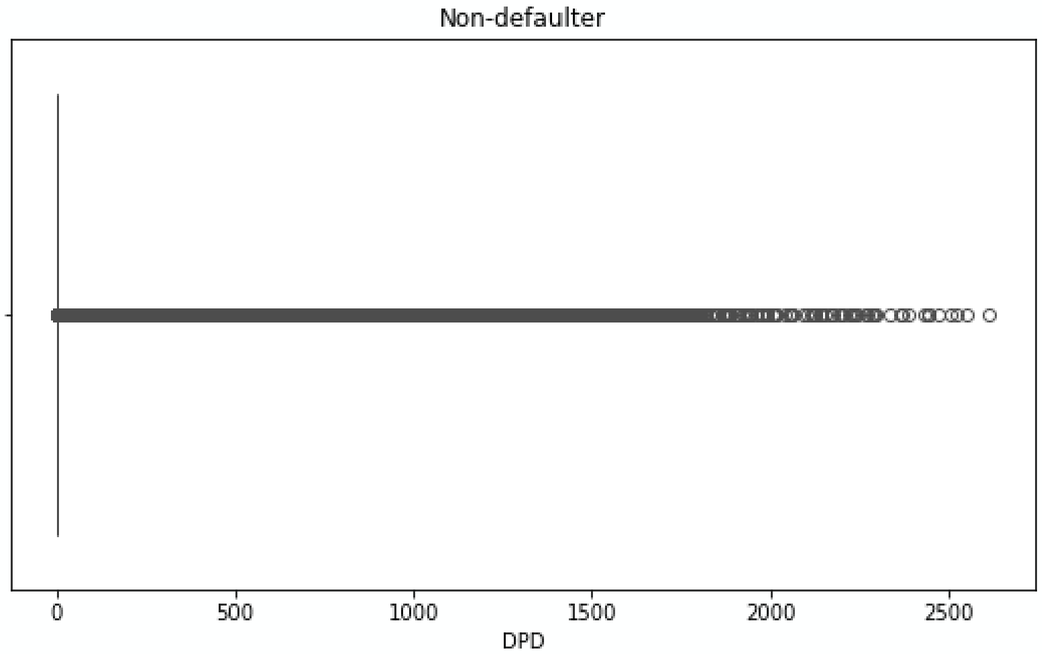


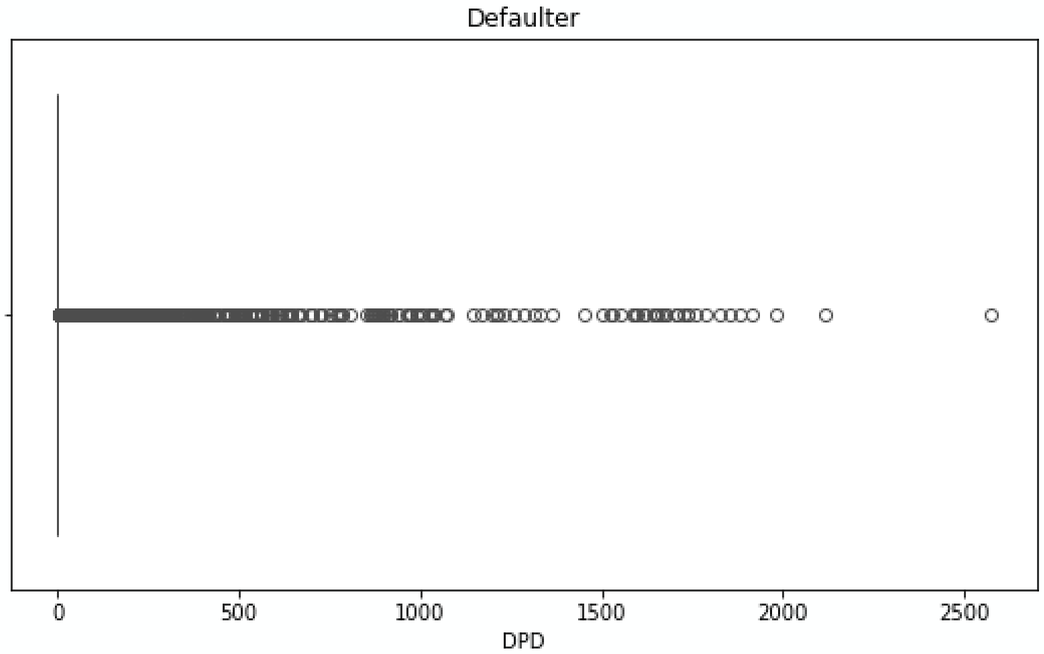
DAYS BEFORE DUE

- Using Box-plot Chart, the Non-defaulters tend to have longer days before due (1~410 days) than the defaulter (1~300 days).
- The interquartile of Non-Defaulter smaller than Defaulter (less than 10 days).

Non-defaulters usually pay the loans earlier than the defaulters instead of waiting until the days near maturity day.

on time or not?





DAYS PAST DUE

- Using Box-plot Chart, recognizing the non-defaulter days past due (1, 1600 days) is longer than the defaulters (0, 750 days).

Non-defaulters somehow tend to delay paying the loans more than the defaulters. Dividing the time periods in different segments.

on time or not? – DAYS PAST DUE SEGMENTS

We separate the days past due in 3 different time periods:

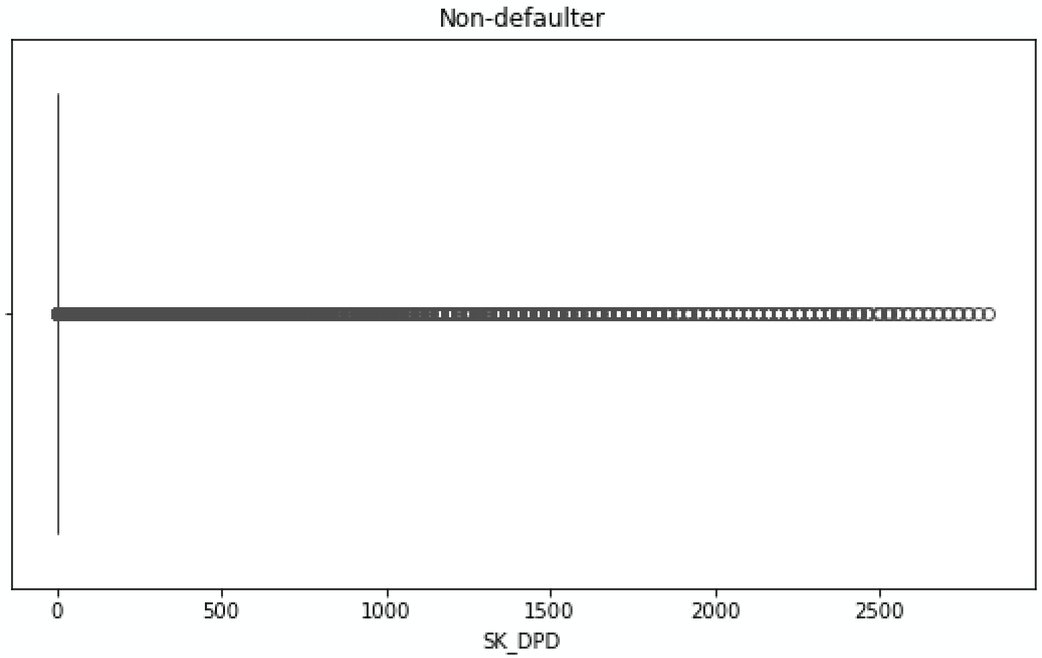
1. **Short-term**: Under 1 year
   - 7 days
   - 15 days
   - 120 days
2. **Mid-term**: From 1 year to 5 years (36-60 months).
3. **Long-term**: Over 5 years.

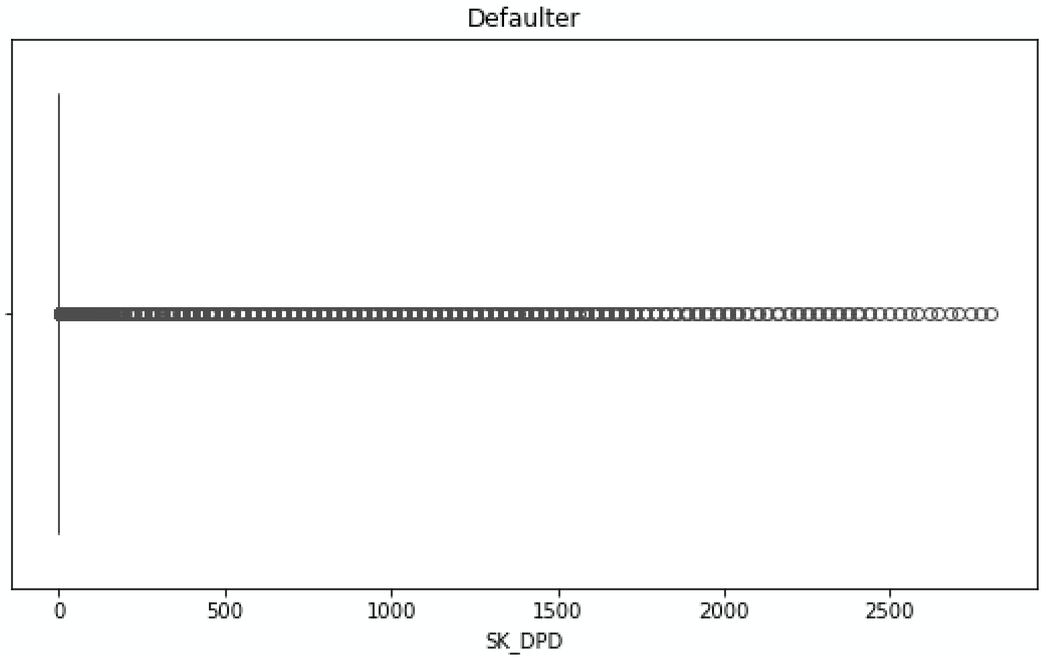
```
pay['DPD_7'] = pay['DPD'].apply(lambda x: 1 if x >= 7 else 0)
pay['DPD_15'] = pay['DPD'].apply(lambda x: 1 if x >= 15 else 0)
pay['INS_IS_DPD_UNDER_120'] = pay['DPD'].apply(lambda x: 1 if (x > 0) & (x < 120) else 0)
pay['INS_IS_DPD_OVER_120'] = pay['DPD'].apply(lambda x: 1 if (x >= 120) else 0)
    for months in [36, 60]:
        recent_prev_id = pay[pay['DAYS_INSTALMENT'] >= -30*months]['SK_ID_PREV'].unique()
        pay_recent = pay[pay['SK_ID_PREV'].isin(recent_prev_id)]
        prefix = 'INS_{}M_'.format(months)
        pay_agg = group_and_merge(pay_recent, pay_agg, prefix, INSTALLMENTS_TIME_AGG)
```

CREDIT CARD BALANCE

- History record of credit card transactions
- AMT vs CNT

default risk segment



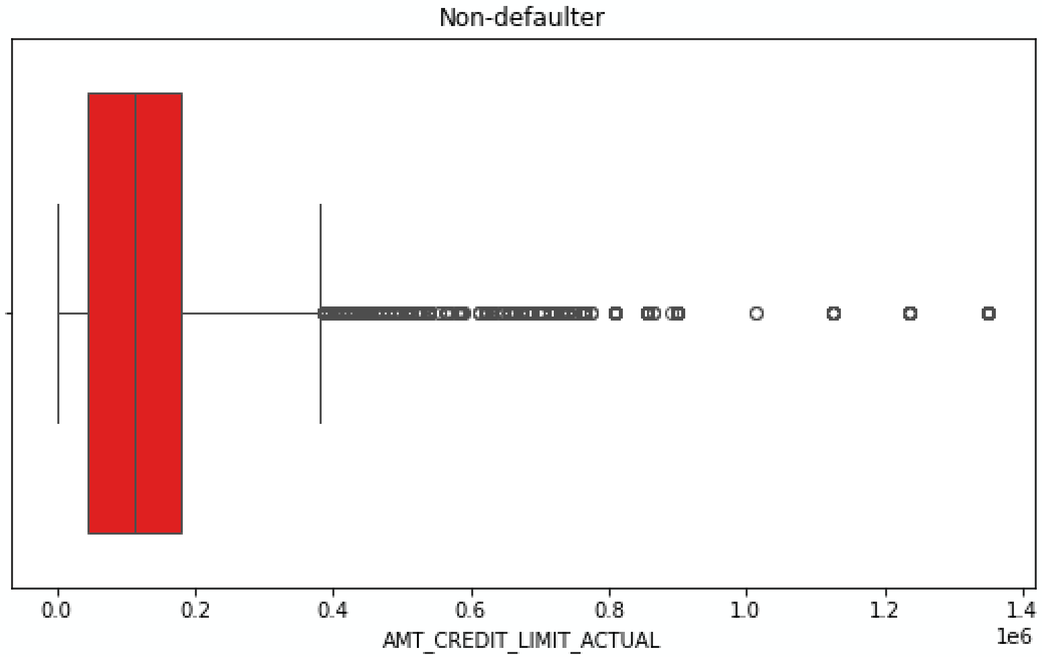


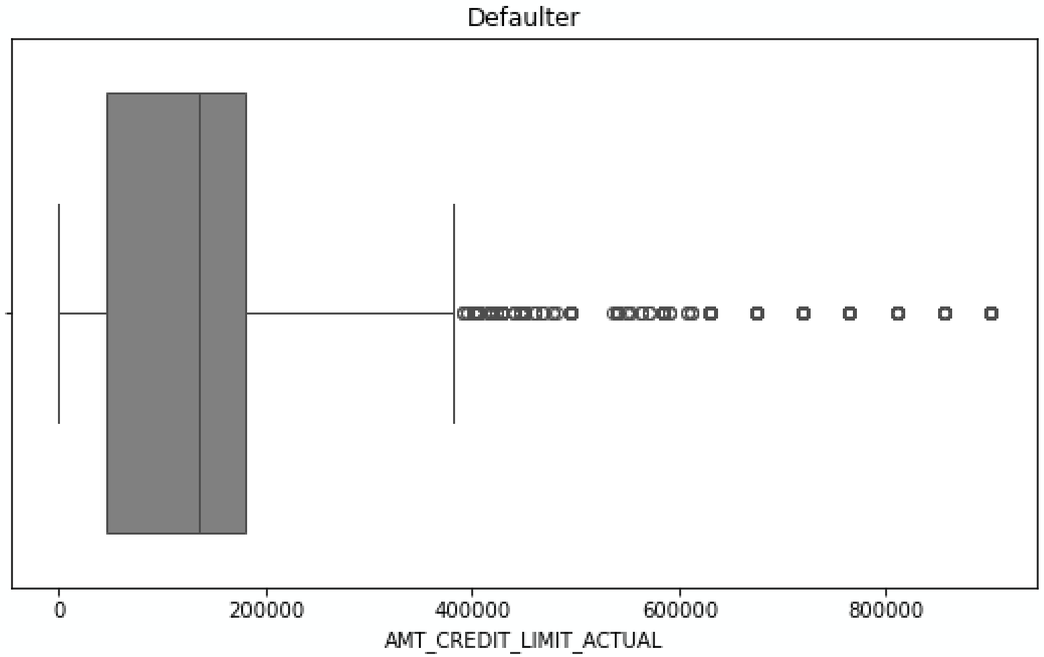
DAYS PAST DUE

- The concentration range of the days past due of non-defaulters is wider than the defaulter.

Non-defaulters tend to delay the payments longer than the defaulters however ...

default risk segment



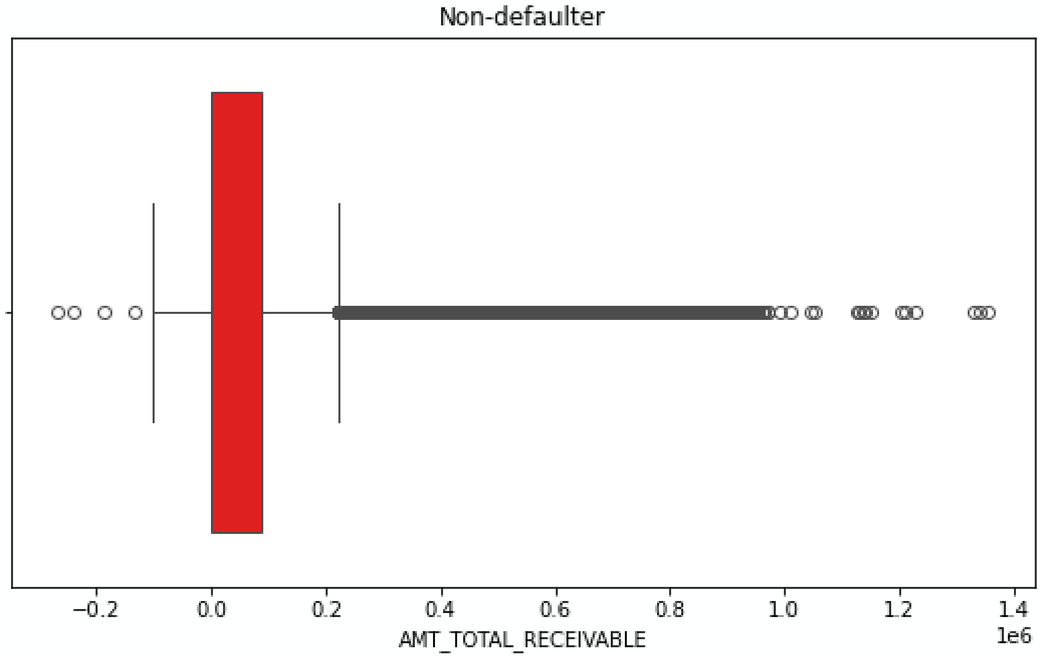


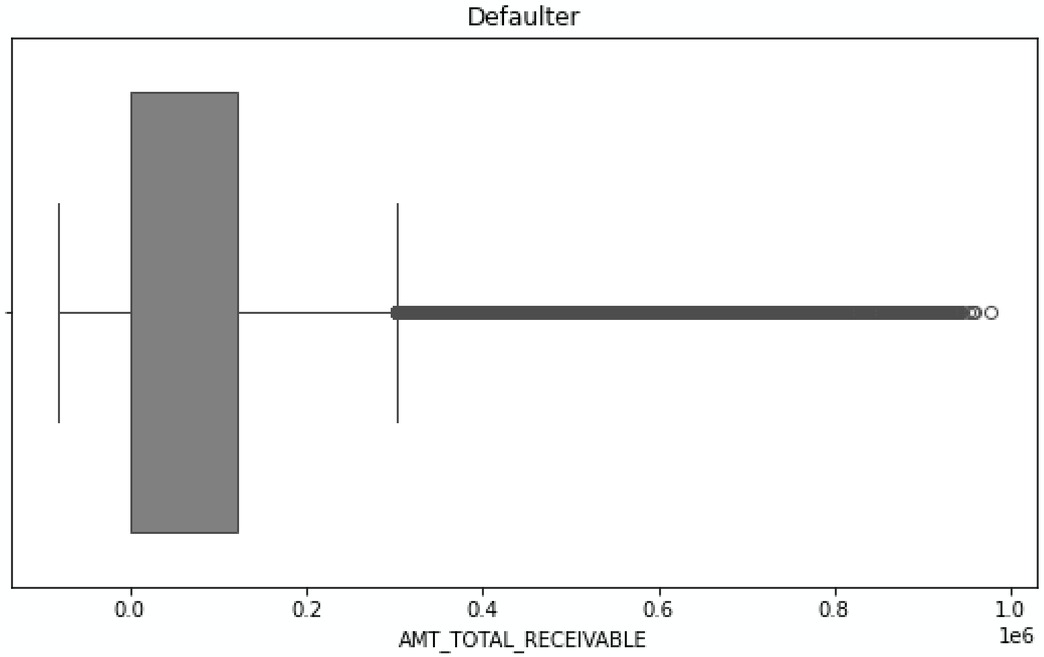
AMT_CREDIT_LIMIT_ACTUAL

- The quartile range of the credit limit defaulter is wider than the non-defaulter, however the number of outliers (>400,000 USD) of non-defaulter is more than the defaulters.

Non-defaulter have larger credit limit <--> longer period to repay loan

interest earning segment



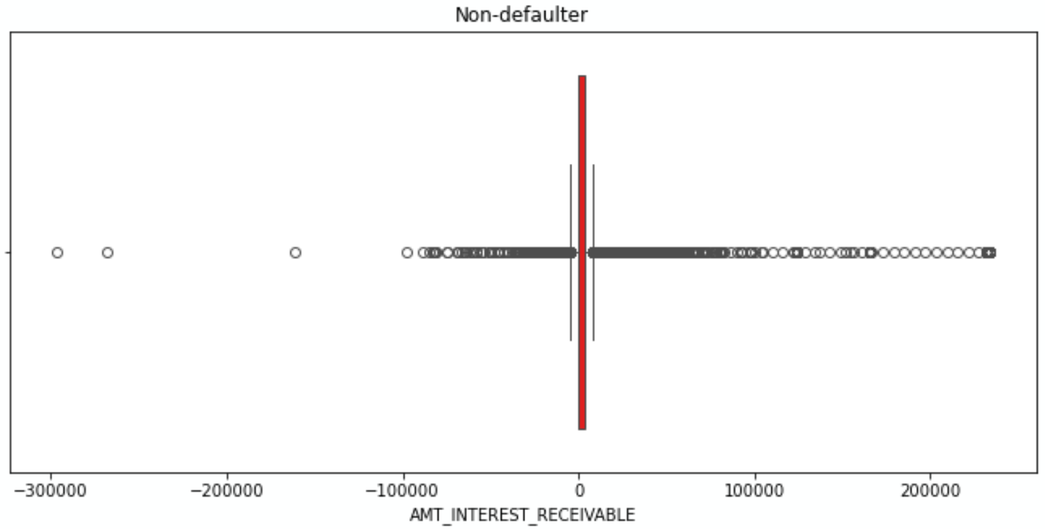


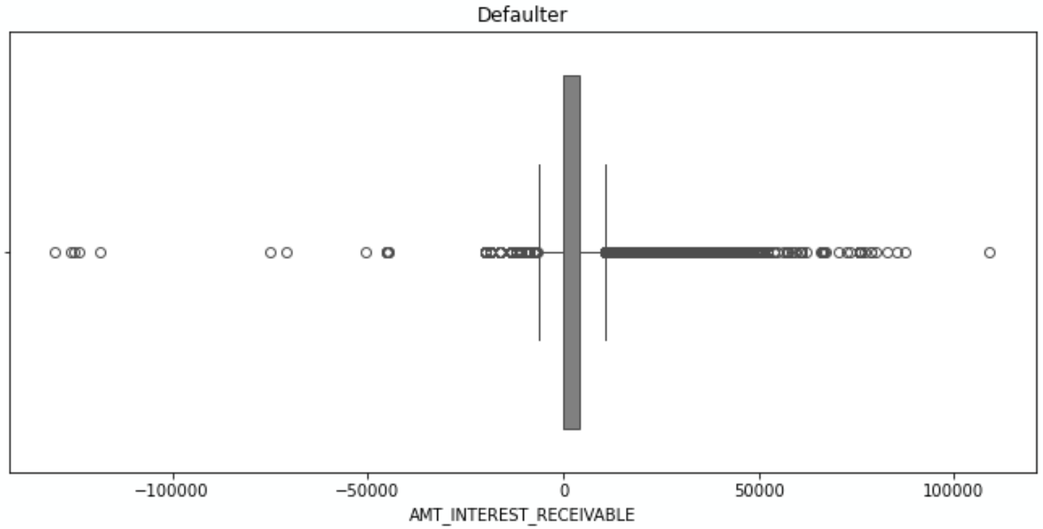
AMT_TOTAL_RECEIVABLE

- The Non-defaulters tend to have smaller interquartile than the defaulters.
- Non-defaulters have more outliners than Defaulters.

The total receivable amount of non-defaulters is mildly similar, however ....

interest earning segment





AMT_INTEREST_RECEIVABLE

- The range of interest receivable amount of non-defaulters is wider than the defaulters.
- The income of bank mainly come from interest charge on loan.

The non-defaulter likely have to pay more interest payment than the defaulter

PREPROCESSING & FEATURE ENGINEERING

In machine learning, preprocessing and feature engineering are pivotal steps. Preprocessing involves data refinement, handling missing values, and encoding, while feature engineering enhances model performance by creating informative features. Together, they optimize input data, mitigating noise and improving a model's predictive accuracy.

APPLICATION TRAIN | TEST

More Info

target **encoder**

class **category_encoders.target_encoder.TargetEncoder**(*verbose=0, cols=None, drop_invariant=False, return_df=True, handle_missing='value', handle_unknown='value', min_samples_leaf=20, smoothing=10, hierarchy=None*) [source]

Target encoding for categorical features.

Supported targets: binomial and continuous. For polynomial target support, see PolynomialWrapper.

For the case of categorical target: features are replaced with a blend of posterior probability of the target given particular categorical value and the prior probability of the target over all the training data.

For the case of continuous target: features are replaced with a blend of the expected value of the target given particular categorical value and the expected value of the target over all the training data.

```
# Create a list of feature from the training set exclude TARGET and SK_ID_CURR
feats = [f for f in train.columns if f not in ['TARGET','SK_ID_CURR']]
# Encode all the categorical variables with TargetEncoder
target = train['TARGET']
enc = TargetEncoder(return_df = True)
train_encode = enc.fit_transform(train[feats],target)
train_encode['SK_ID_CURR'] = train['SK_ID_CURR']
train_encode['TARGET'] = target
test_encode = enc.transform(test[feats])
test_encode['SK_ID_CURR'] = test['SK_ID_CURR']
train = train_encode
test = test_encode
# df = train.append(test)
df = pd.concat([train, test]).reset_index(drop=True)
del train, test; gc.collect()
```

data cleaning & **feature engineering**

- Replace all invalid days of day features by np.nan.
- Create feature to count the total of FLAG_DOCUMENT_x customer provided.
- Create new features which are:
  - Income band based on AMT_INCOME_TOTAL.
  - Age groups based on DAYS_BIRTH.
  - The information about the owning asset based on FLAG_OWN_CAR and FLAG_OWN_REALTY.
  - Information about the children.
  - Some ratio features: Time ratios, credit ratios, income ratios.
- New features created base on polynomial, ratios, differences and aggregations (by mean, max, min, median, var) of EXT_SOURCE are spotted to improved the model a lot.
- Drop useless features related to housing information and documents.

PREVIOUS APPLICATION

More Info

data cleaning & **feature engineering**

- Replace all invalid days of day features by np.nan.

- One-hot encoding categorical features.

- Create new features which are:
  - Ratios and difference of day features.
  - Features about Interest rate.

- Drop useless features that are spotted during EDA.

---

data cleaning & **feature engineering**

```
# Aggregations for approved and refused loans
agg_prev = group_and_merge(approved, agg_prev, 'APPROVED_', PREVIOUS_APPROVED_AGG)
refused = prev[prev['NAME_CONTRACT_STATUS_Refused'] == 1]
agg_prev = group_and_merge(refused, agg_prev, 'REFUSED_', PREVIOUS_REFUSED_AGG)

# Aggregations for Consumer loans and Cash loans
for loan_type in ['Consumer loans', 'Cash loans']:
    type_df = prev[prev['NAME_CONTRACT_TYPE_{}'.format(loan_type)] == 1]
    prefix = 'PREV_' + loan_type.split(" ")[0] + '_'
    agg_prev = group_and_merge(type_df, agg_prev, prefix, PREVIOUS_LOAN_TYPE_AGG)

# Get the SK_ID_PREV for loans with late payments (days past due)
pay['LATE_PAYMENT'] = pay['DAYS_ENTRY_PAYMENT'] - pay['DAYS_INSTALMENT']
pay['LATE_PAYMENT'] = pay['LATE_PAYMENT'].apply(lambda x: 1 if x > 0 else 0)
dpd_id = pay[pay['LATE_PAYMENT'] > 0]['SK_ID_PREV'].unique()

# Aggregations for loans with late payments
agg_dpd = group_and_merge(prev[prev['SK_ID_PREV'].isin(dpd_id)], agg_prev,
                          'PREV_LATE_', PREVIOUS_LATE_PAYMENTS_AGG)
del agg_dpd, dpd_id; gc.collect()
```

POS_CASH BALANCE

More Info

data cleaning & **feature engineering**

- One-hot encoding categorical features.

- Feature about percentage of previous loans completed and completed before initial term.

- Feature about number of remaining installments (future installments) and percentage from total.

- Features about percentage of late payments for the 3 most recent applications.

INSTALLMENT PAYMENTS

More Info

data cleaning & **feature engineering**

```
# Flag k threshold late payments
pay['DPD_7'] = pay['DPD'].apply(lambda x: 1 if x >= 7 else 0)
pay['DPD_15'] = pay['DPD'].apply(lambda x: 1 if x >= 15 else 0)

#----- BỔ SUNG---------------------------------------------------------
pay['INS_IS_DPD_UNDER_120'] = pay['DPD'].apply(lambda x: 1 if (x > 0) & (x < 120) else 0)
pay['INS_IS_DPD_OVER_120'] = pay['DPD'].apply(lambda x: 1 if (x >= 120) else 0)


# Installments in the last x months
for months in [36, 60]:
    recent_prev_id = pay[pay['DAYS_INSTALMENT'] >= -30*months]['SK_ID_PREV'].unique()
    pay_recent = pay[pay['SK_ID_PREV'].isin(recent_prev_id)]
    prefix = 'INS_{}M_'.format(months)
    pay_agg = group_and_merge(pay_recent, pay_agg, prefix, INSTALLMENTS_TIME_AGG)
```

CREDIT CARD BALANCE

More Info

exponential moving average

WHAT IS **THE PURPOSE?**

$$\sigma_n^2(ewma) = \lambda\sigma_n^2 + (1-\lambda)u_{n-1}^2$$

where:

λ = the degree of weighting decrease

σ^2 = value at time period n

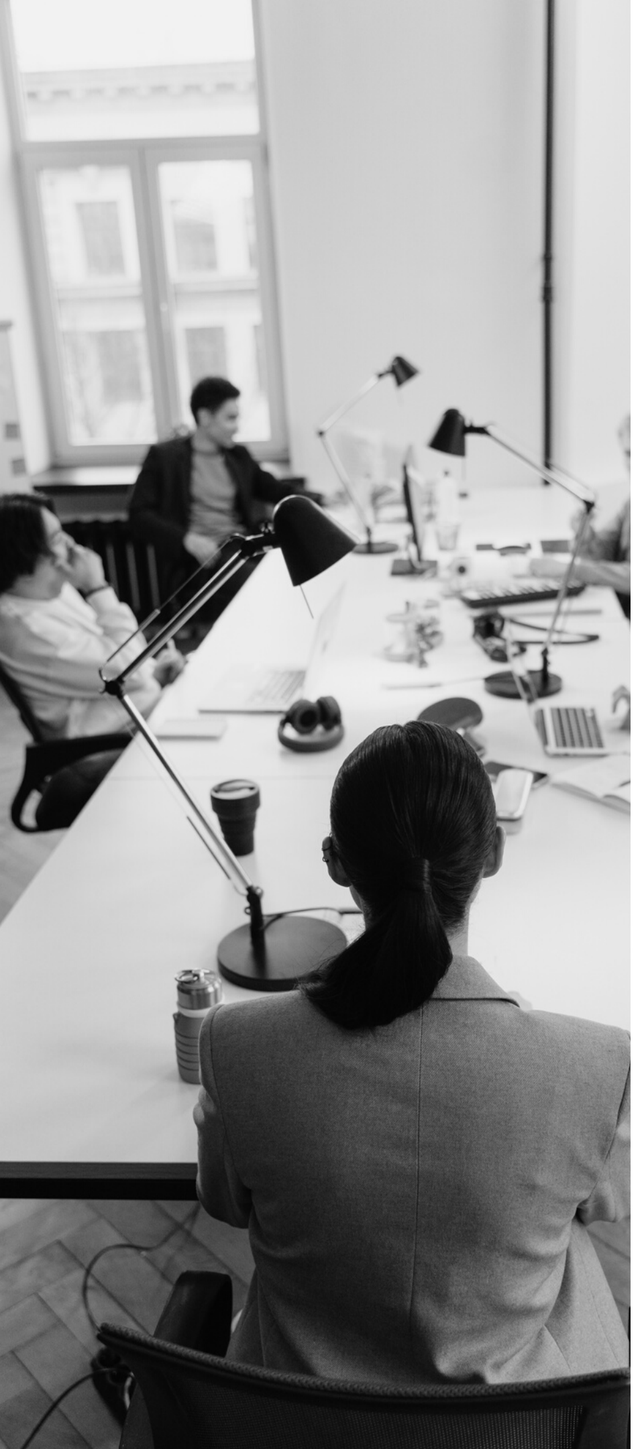
u^2 = value of EWMA at time period n

Some features having time based data. Instead of using Moving Average which treats everything equally, we chose Exponential Moving Average with the goal is to provide more weight to recent observations while still considering historical data, making it particularly useful for capturing trends and reacting quickly to changes.

data cleaning & **feature engineering**

```
#calculating the rolling Exponential Weighted Moving Average over months for certain features
rolling_columns = [
    'AMT_BALANCE',
    'AMT_CREDIT_LIMIT_ACTUAL',
    'AMT_RECEIVABLE_PRINCIPAL',
    'AMT_RECEIVABLE',
    'AMT_TOTAL_RECEIVABLE',
    'AMT_DRAWING_SUM',
    'BALANCE_LIMIT_RATIO',
    'CNT_DRAWING_SUM',
    'MIN_PAYMENT_RATIO',
    'PAYMENT_MIN_DIFF',
    'MIN_PAYMENT_TOTAL_RATIO',
    'AMT_INTEREST_RECEIVABLE',
    'SK_DPD_RATIO' ]
exp_weighted_columns = ['EXP_' + ele for ele in rolling_columns]
cc[exp_weighted_columns] = cc.groupby(['SK_ID_CURR','SK_ID_PREV'])[rolling_columns].transform(lambda x: x.ewm(alpha = 0.7).mean())
"
```

The aggregation functions performed on these features were mostly '**LAST**' and '**MEAN**'.



data cleaning & **feature engineering**

- One-hot encoding categorical features.

- Aggregate data by active and close loan status.

- Aggregate data by main loan types.

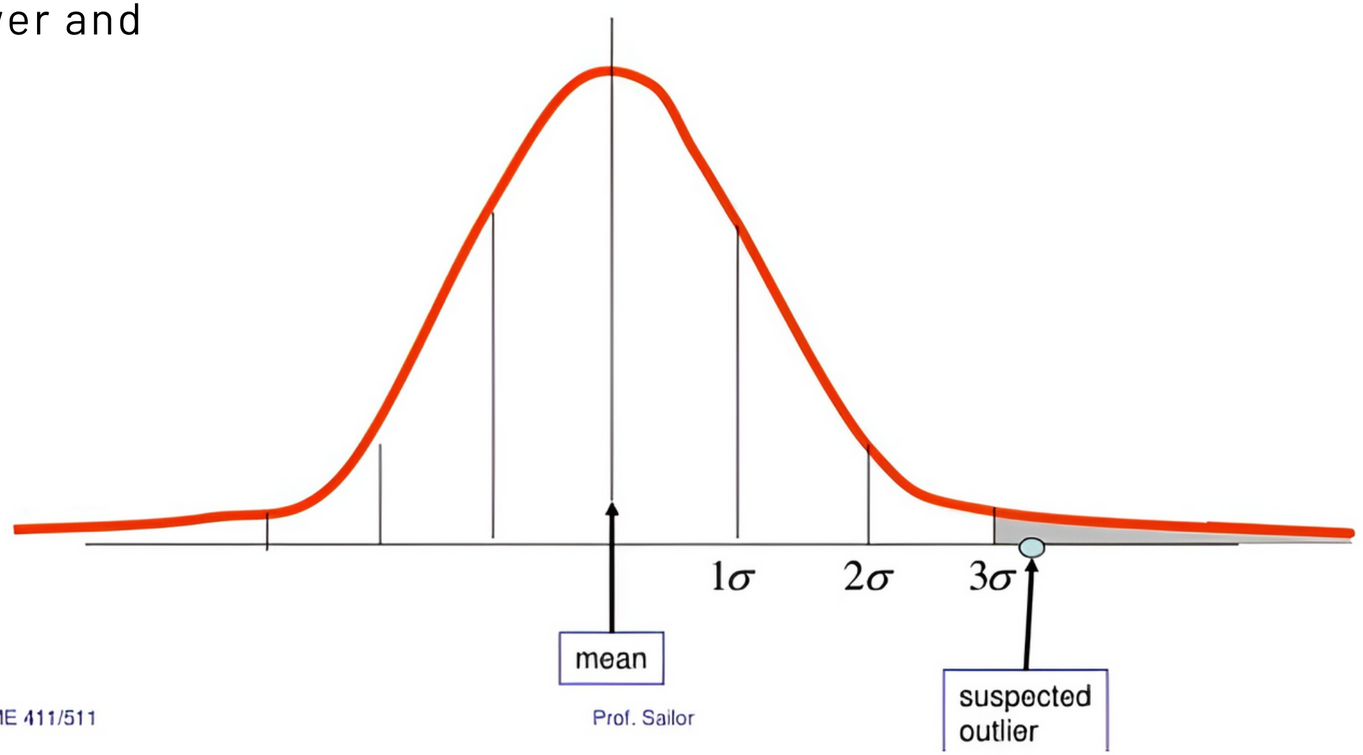
- Aggregate data by time during 6 and 12 months.

---

handling **outliers**

Using 3 sigma rules to handling outliers. We basically replace extreme values with lower and upper bound.

```
# Define low and high thresholds
low =  mu - 3*sigma
high = mu + 3*sigma
```



feature selection

```python
LIGHTGBM_PARAMS = {
    'boosting_type': 'goss',
    'n_estimators': 10000,
    'learning_rate': 0.005134,
    'num_leaves': 54,
    'max_depth': 10,
    'subsample_for_bin': 240000,
    'reg_alpha': 0.436193,
    'reg_lambda': 0.479169,
    'colsample_bytree': 0.508716,
    'min_split_gain': 0.024766,
    'subsample': 1,
    'is_unbalance': False,
    'silent':-1,
    'verbose':-1
}
```

```python
def feature_importance(X_train, y_train):
    feature_importances = np.zeros(X_train.shape[1])
    model = LGBMClassifier(**{**LIGHTGBM_PARAMS})

    # Training the lgbm for 2 rounds to avoid overfitting
    for i in range(2):
        train_features, valid_features, train_y, valid_y = train_test_split(X_train, y_train, test_size=0.25, random_st
        model.fit(train_features, train_y, eval_set=[(valid_features, valid_y)], eval_metric='auc',
                  callbacks=[early_stopping(stopping_rounds=100)])

        feature_importances += model.feature_importances_

    # The final feature importance scores is the average of scores of 2 training rounds
    feature_importances /= 2
    feature_importances = pd.DataFrame({'feature': list(X_train.columns), 'importance': feature_importances}).sort_valu

    # Return a list of features with zero importance, which would be dropped from the final data frame.
    zero_features = list(feature_importances[feature_importances['importance'] == 0.0]['feature'])
    return zero_features
```


missing values **& scaling data**

```
with timer('Fill nan'):
    imputer = SimpleImputer(strategy='median')
    imputer = imputer.fit(X_train)
    X_train = pd.DataFrame(imputer.transform(X_train), columns=X_train.columns)
    X_test = pd.DataFrame(imputer.transform(X_test), columns=X_test.columns)
```

```
with timer('Scale by StandardScaler'):
    std = StandardScaler()
    X_train = pd.DataFrame(std.fit_transform(X_train), columns=X_train.columns)
    std2 = StandardScaler()
    X_test = pd.DataFrame(std2.fit_transform(X_test), columns=X_test.columns)
```


hyparameters

tunning

```
with timer('Start tune with GridSearchCV'):
    model = LogisticRegression()
    param_grid = {
        'penalty' : [
                    'l2'
                   #  'l1',
                      'elasticnet'
                    ],
        'C' :[
            # 0.001,
            0.01,
              # 0.1, 1,
              # 10
            # 15, 20, 25
            ],
        'solver' : [
            'liblinear',
            # 'saga'
            # 'lbfgs',
            # 'newton-cg',
              'sag'
        ],
        # 'max_iter' : [700]
    }

    grid_search = GridSearchCV(estimator=model, param_grid=param_grid,verbose=10,scoring='roc_auc_ovr')
    grid_search.fit(X_train, y_train)
    print(f'Best param: {grid_search.best_params_}')
    y_prob_train = grid_search.predict_proba(X_train)[:,1]
    y_prob_test = grid_search.predict_proba(X_test)[:,1]
    print(f'AUC: {roc_auc_score(y_train, y_prob_train)}')
```

	SK_ID_CURR	fs11	fs12	fs13	fs14	fs16	fs18
0	83659	0.129271	0.131168	0.114922	0.113863	0.121514	0.127414
1	174814	0.060277	0.060166	0.061297	0.061239	0.059153	0.058527
2	179486	0.197447	0.197868	0.195962	0.195536	0.194121	0.194596
3	57038	0.022182	0.022172	0.022042	0.022010	0.022795	0.020562
4	25672	0.059562	0.060142	0.061025	0.060270	0.057913	0.058304
...	...	...	...	...	...	...	...
61497	150442	0.022891	0.023160	0.025377	0.025071	0.022944	0.023793
61498	5217	0.024458	0.024668	0.023202	0.023031	0.023275	0.023949
61499	260741	0.049051	0.048107	0.044961	0.044452	0.051552	0.049807
61500	284794	0.012973	0.013005	0.013093	0.013054	0.012791	0.012993
61501	258643	0.039480	0.039555	0.044306	0.044176	0.042341	0.039814

	fs11	fs12	fs13	fs14	fs16	fs18
fs11	1.000000	0.999794	0.997798	0.998017	0.998454	0.995640
fs12	0.999794	1.000000	0.997992	0.997792	0.998253	0.995430
fs13	0.997798	0.997992	1.000000	0.999751	0.997328	0.993703
fs14	0.998017	0.997792	0.999751	1.000000	0.997533	0.993896
fs16	0.998454	0.998253	0.997328	0.997533	1.000000	0.993112
fs18	0.995640	0.995430	0.993703	0.993896	0.993112	1.000000
test12	0.996538	0.996328	0.994680	0.994875	0.994414	0.999932

blending technique

fs18 is the best result based on the LB.

The correlation between **fs18** & **fs16** is lowest.

Mix two of that with ratio **0.9:0.1**

for fs18 & fs16 respectively.

THANK YOU SO MUCH!