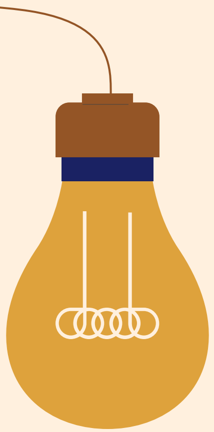
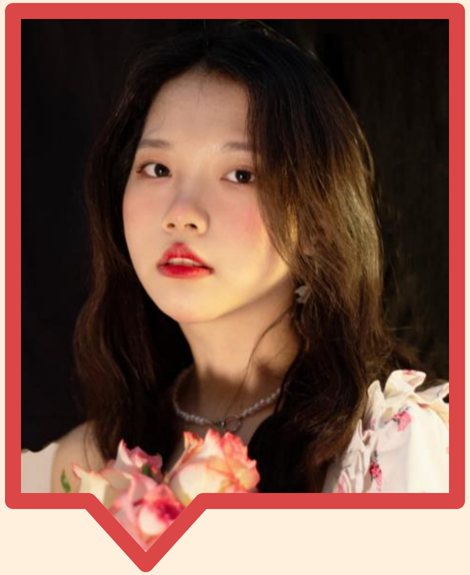


HOME CREDIT DEFAULT RISK

DozenTeam

Our Team

Lê Thị Quỳnh Anh
11219256



Nguyễn Khánh Huyền
11212719

Đào Ngọc Chi
11219263

Table of Contents

01

ALL STEPS OF EDA

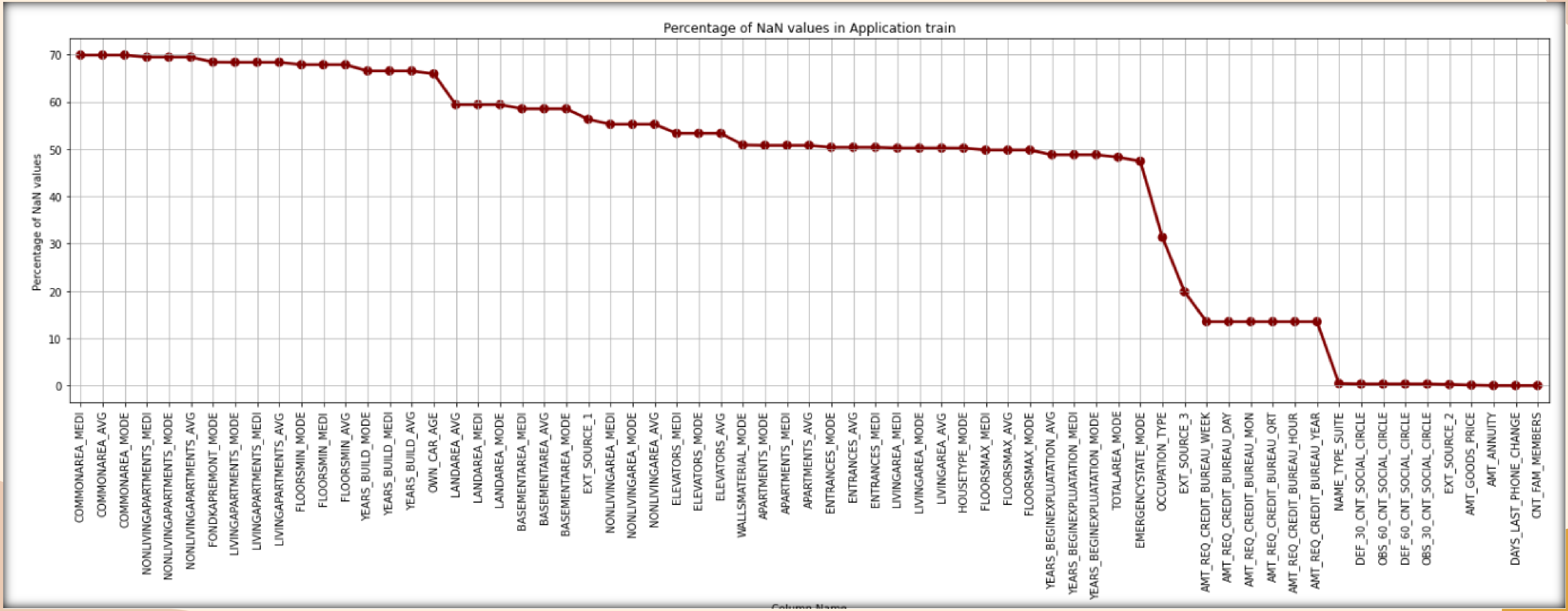
Basic Information of data

df.shape

df.duplicated

df.info()

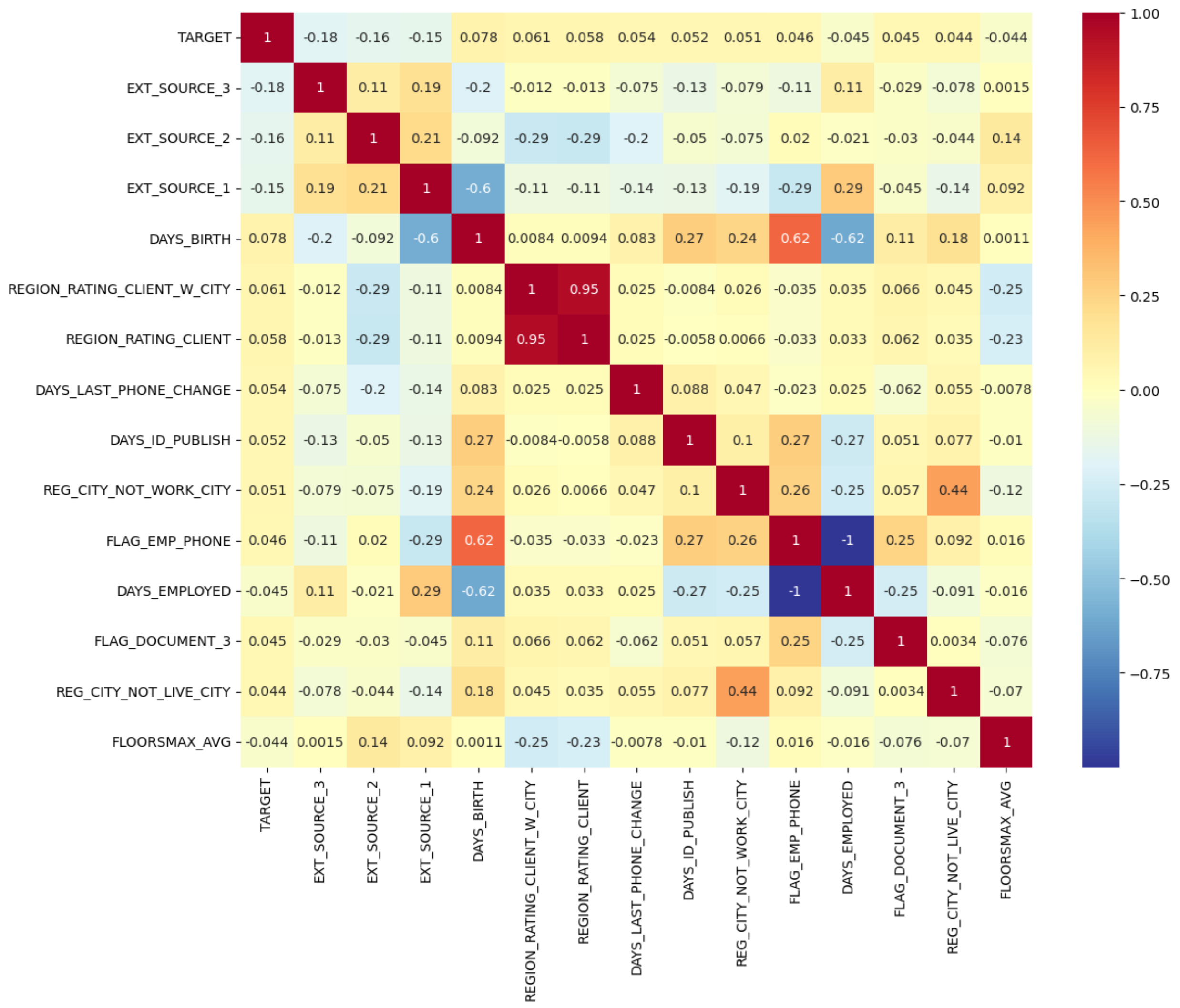
Missing Value

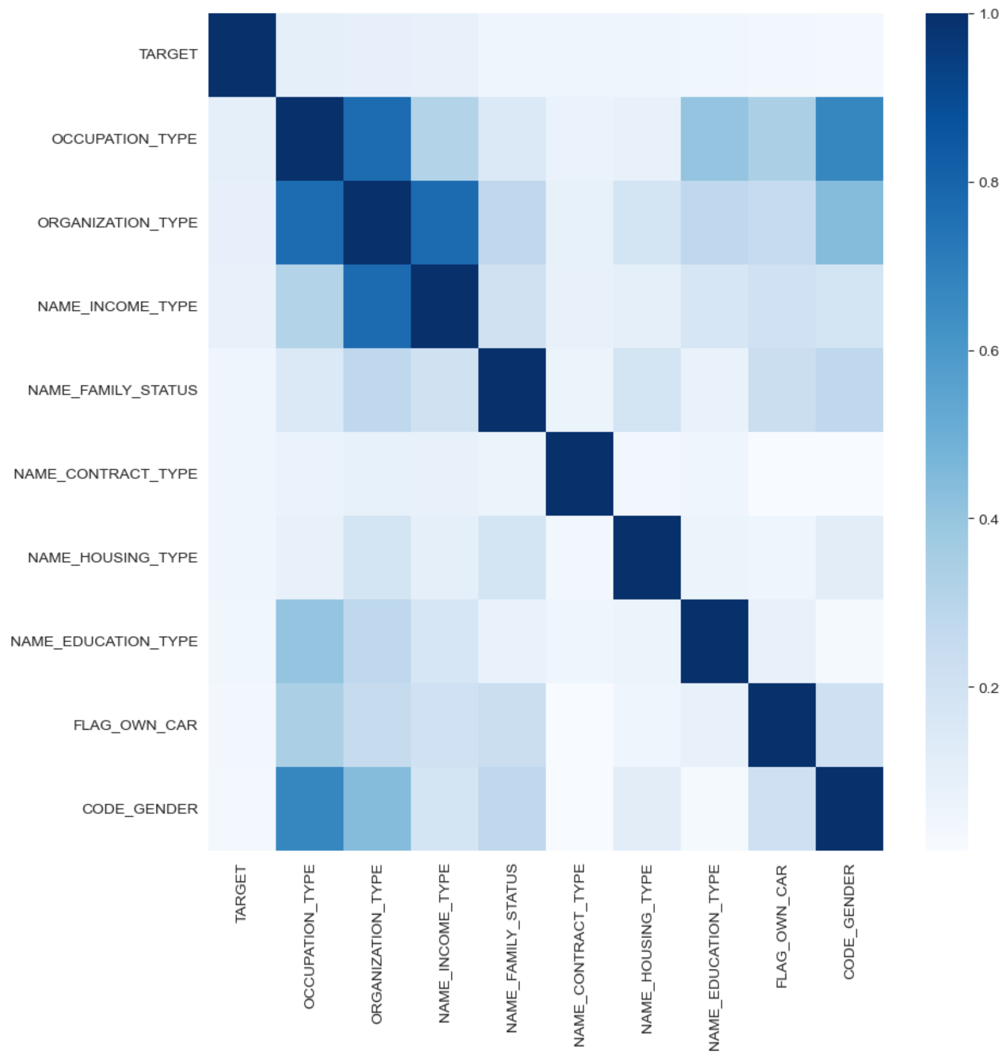


```python
bureau_merged = pd.merge(application_train[["SK_ID_CURR", "TARGET"]], bureau,
                         how='left', on=['SK_ID_CURR'])
bur_bal_merged = pd.merge(application_train[['SK_ID_CURR', 'TARGET']], bureau_balance,
                          on='SK_ID_CURR', how='inner')
prev_merged = pd.merge(application_train[["SK_ID_CURR", "TARGET"]], previous_application,
                       how='left', on=['SK_ID_CURR'])
pos_cash_merged = pd.merge(application_train[["SK_ID_CURR", "TARGET"]], pos_cash,
                           how='left', on=['SK_ID_CURR'])
installments_merged = pd.merge(application_train[["SK_ID_CURR", "TARGET"]], installments_payments,
                               how="left", on="SK_ID_CURR")
credit_card_merged = pd.merge(application_train[["SK_ID_CURR", "TARGET"]], credit_card,
                              how='left', on=['SK_ID_CURR'])
```

Merge TARGET column in application_train to get more insight

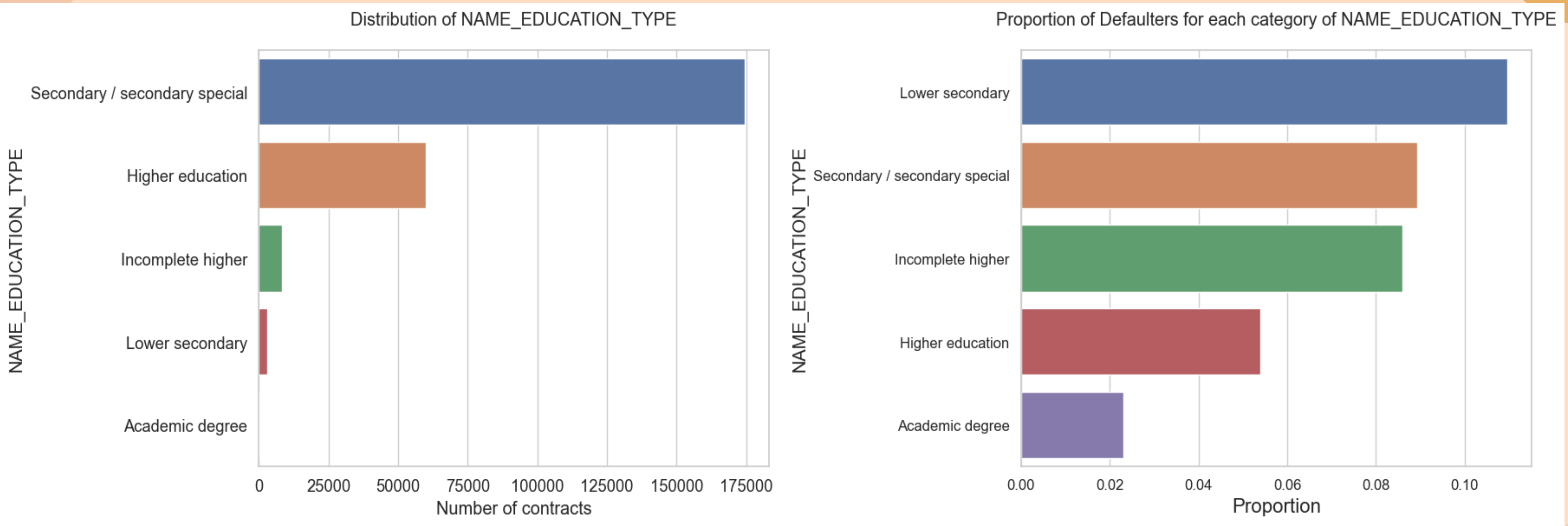
Top correlation with TARGET
by type of variable: Numerical and Categorical





Plot Categorical feature

```python
def plot_categorical_variables(df, column_name, figsize = (18, 6), count_display = True, plot_defaulter = True):
    print(f"Total Number of unique categories of {column_name} = {len(df[column_name].unique())}")
    plt.figure(figsize = figsize, tight_layout = False)
    sns.set(style = 'whitegrid', font_scale = 1.2)

    #plotting overall distribution of category
    data_to_plot = df[column_name].value_counts()
    df_to_plot = pd.DataFrame({column_name: data_to_plot.index, 'Number of contracts': data_to_plot.values})

    # Calculate the percentage of target = 1 per category
    default_percent = df[[column_name, 'TARGET']].groupby([column_name], as_index = False)['TARGET'].mean()
    default_percent.sort_values(by = 'TARGET', ascending = False, inplace = True)

    if count_display:
        plt.subplot(1,2,1)
        s1 = sns.barplot(x = 'Number of contracts', y = column_name, data = df_to_plot)
        #s1.set_yticklabels(s1.get_yticklabels(),rotation = 90)
        plt.title(f'Distribution of {column_name}', pad = 20)

    if plot_defaulter:
        plt.subplot(1,2,2)
        s2 = sns.barplot(x = 'TARGET', y = column_name, data = default_percent)
        #s2.set_yticklabels(s2.get_yticklabels(),rotation=90)
        plt.xlabel('Proportion', fontsize=16)
        plt.title(f'Proportion of Defaulters for each category of {column_name}', pad = 20)


    plt.tick_params(axis='both', which='major', labelsize=12)
    plt.subplots_adjust(wspace = 0.4)
    plt.show();
```



NAME_EDUCATION_TYPE

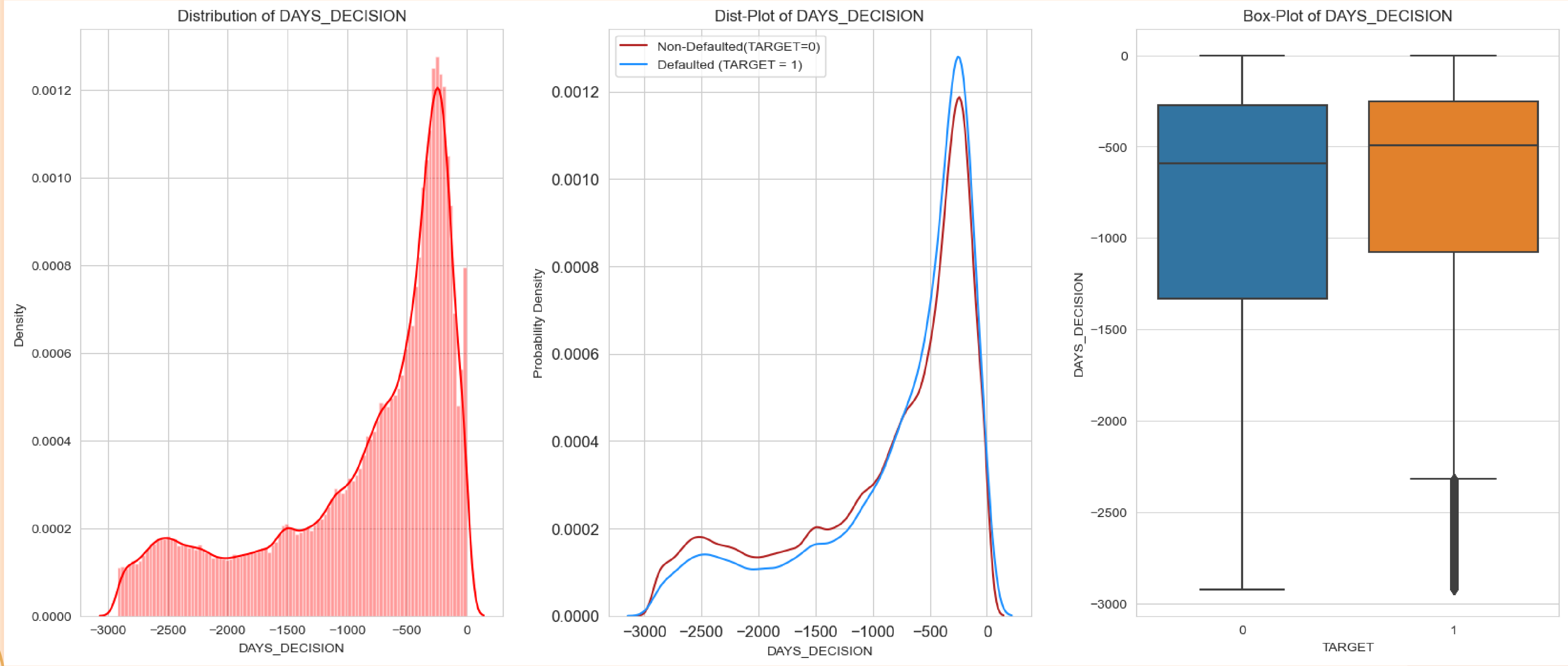
from application_train|test.csv

```python
def plot_numerical_variables(df, column_name, figsize = (20,8), hist_plot = True, box_plot = True,
                             dist_plot = True, number_of_subplots = 3):
    plt.figure(figsize = figsize)
    sns.set_style('whitegrid')
    sns.color_palette("RdBu", 10)
    i = 1
    if hist_plot:
        plt.subplot(1, number_of_subplots, i)
        plt.subplots_adjust(wspace=0.25)
        plt.title("Distribution of %s" %column_name)
        sns.distplot(df[column_name].dropna(),color='red', kde=True,bins=100)
        i+= 1
    if dist_plot:
        plt.subplot(1, number_of_subplots, i)
        plt.subplots_adjust(wspace=0.25)
        sns.distplot(df[column_name][df['TARGET'] == 0].dropna(),\
                     label='Non-Defaulters', hist = False, color = 'firebrick')
        sns.distplot(df[column_name][df['TARGET'] == 1].dropna(),\
                     label='Defaulters', hist = False, color = 'dodgerblue')
        plt.xlabel(column_name)
        plt.ylabel('Probability Density')
        plt.title("Dist-Plot of {}".format(column_name))
        plt.legend(loc="best", labels=['Non-Defaulted(TARGET=0)', 'Defaulted (TARGET = 1)'], fontsize = 'medium')
        plt.tick_params(axis='both', which='major', labelsize=12)
        i+= 1
    if box_plot:
        plt.subplot(1, number_of_subplots, i)
        plt.subplots_adjust(wspace=0.25)

        sns.boxplot(x='TARGET', y=column_name, data=df)
        plt.title("Box-Plot of {}".format(column_name))
    plt.show()
```

Plot Numerical feature

DAYS_DECISION

from previous_application.csv



02

EDA of each table

2.1

Application_train

Imbalanced Data

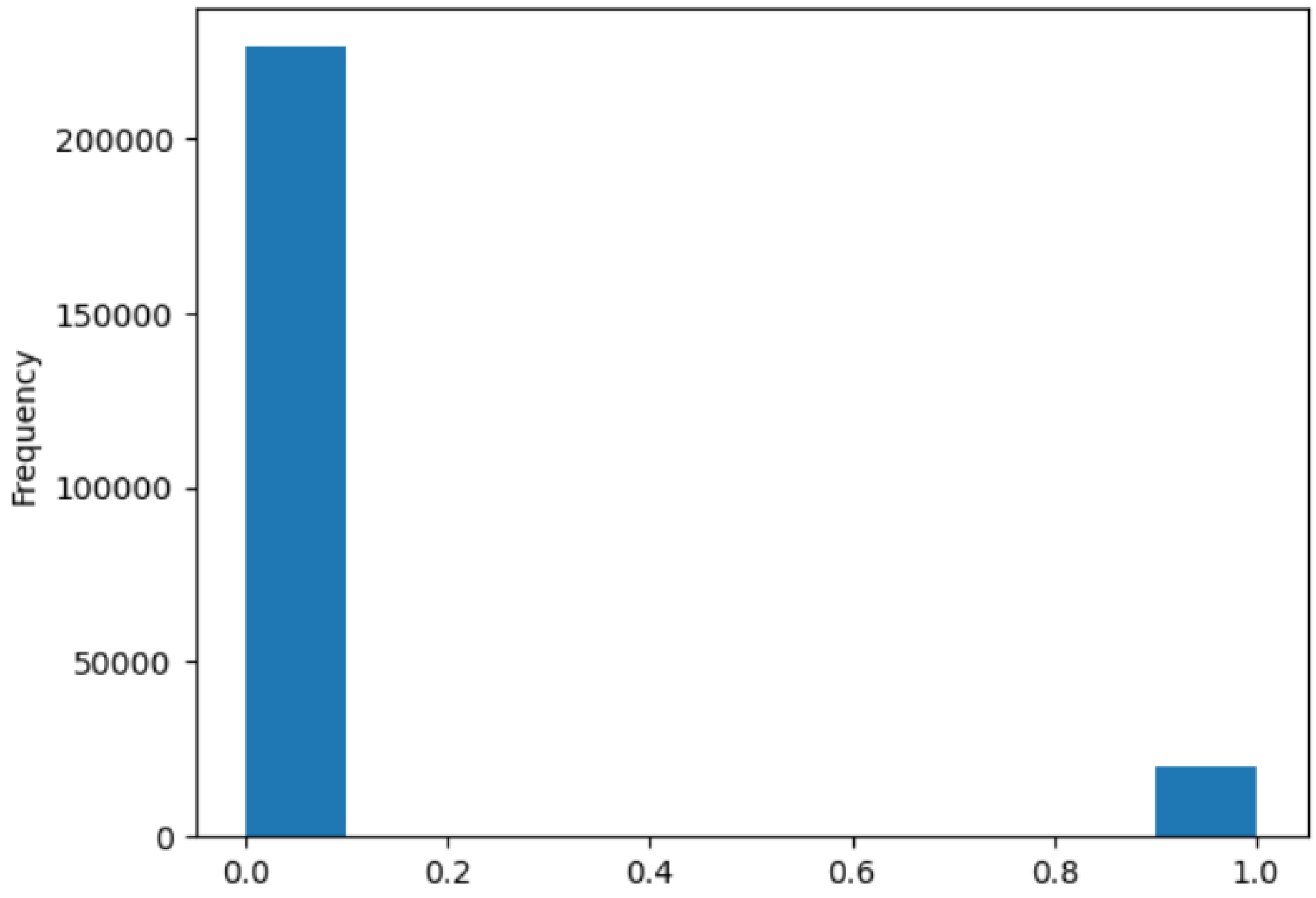
- The number of samples with non-defaulter (TARGET=0) is 10 times greater than the number of samples with defaulter (TARGET=1)

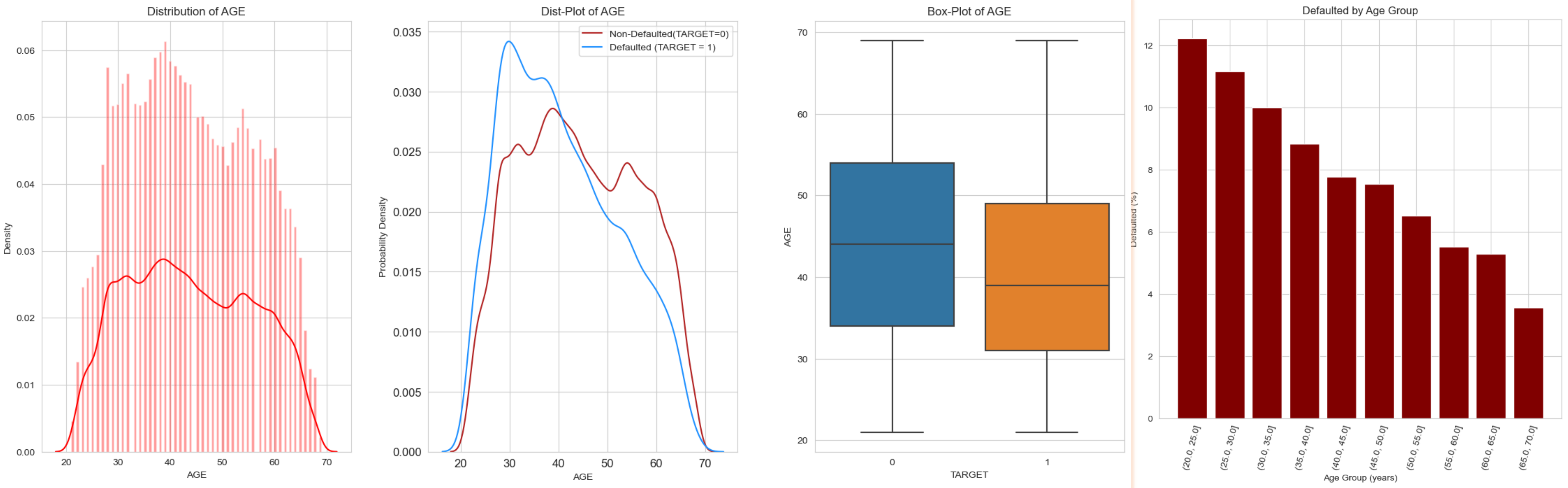
- Using ROC-AUC to for model evaluation

```
application_train['TARGET'].value_counts()
```

```
TARGET
0    226133
1     19876
Name: count, dtype: int64
```

```
application_train['TARGET'].astype(int).plot.hist();
```



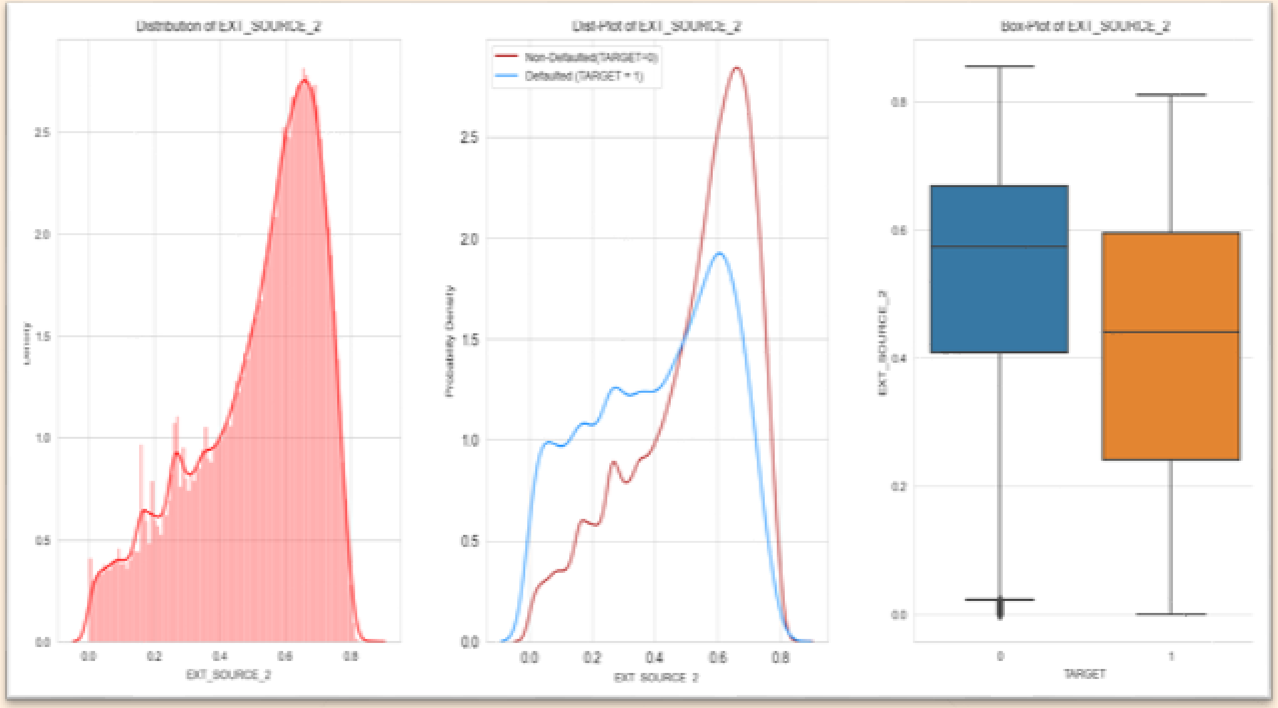


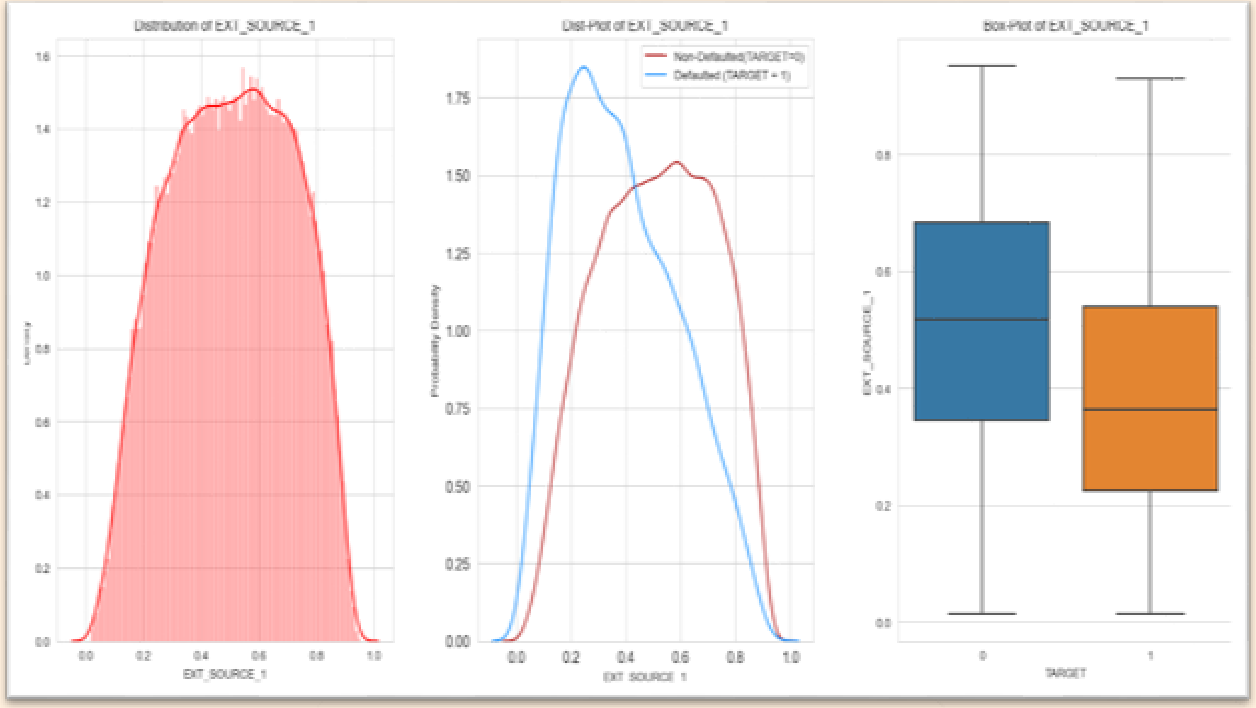
AGE

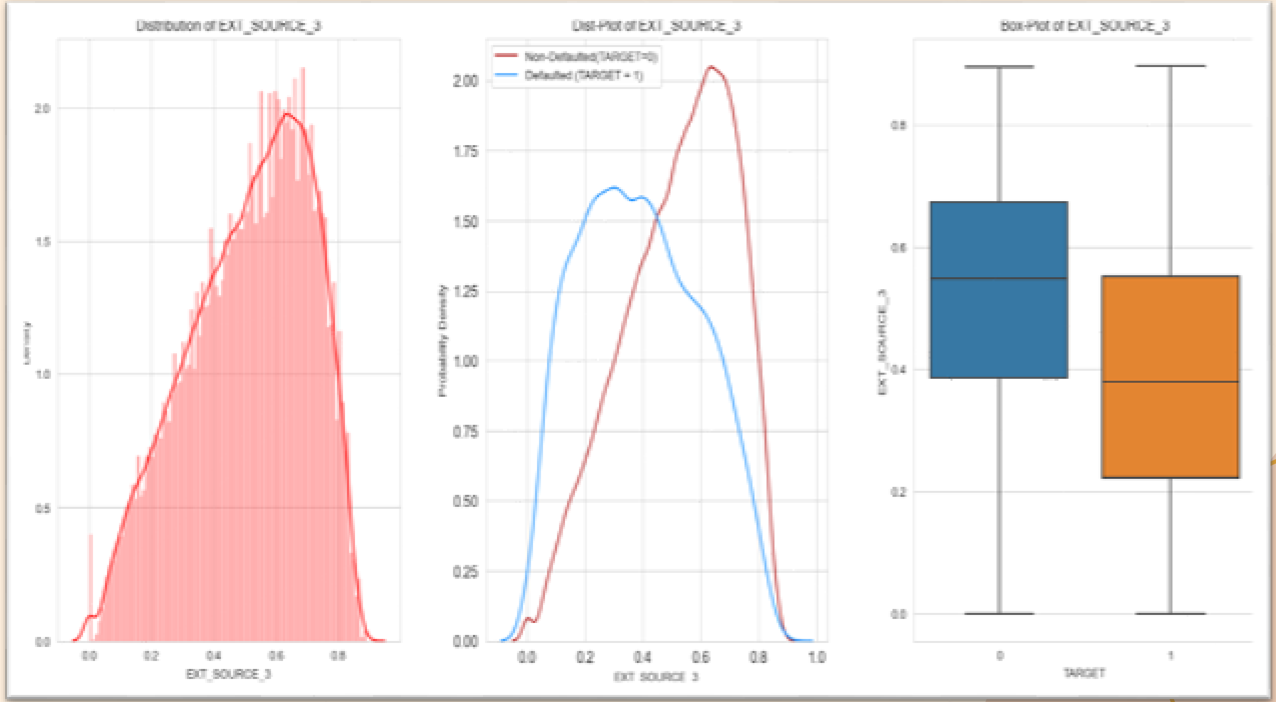
1. Convert DAYS_BIRTH data to year format
2. The age with the most defaulter is 30 years old. Young people tend to have more bad debt than older people
3. Splitting AGE into bins is also an important feature for the 'TARGET' variable. New features can be created from dividing into bins

EXT_SOURCE

- The distribution of people with defaulter and people without defaulter is clearly different
- The EXT_SOURCE features show relatively good linear separation between defaulters and non-defaulters. Among them, EXT_SOURCE_1 and EXT_SOURCE_3 tend to show better separation ability than EXT_SOURCE_2
- Create some more features from EXT_SOURCE variables

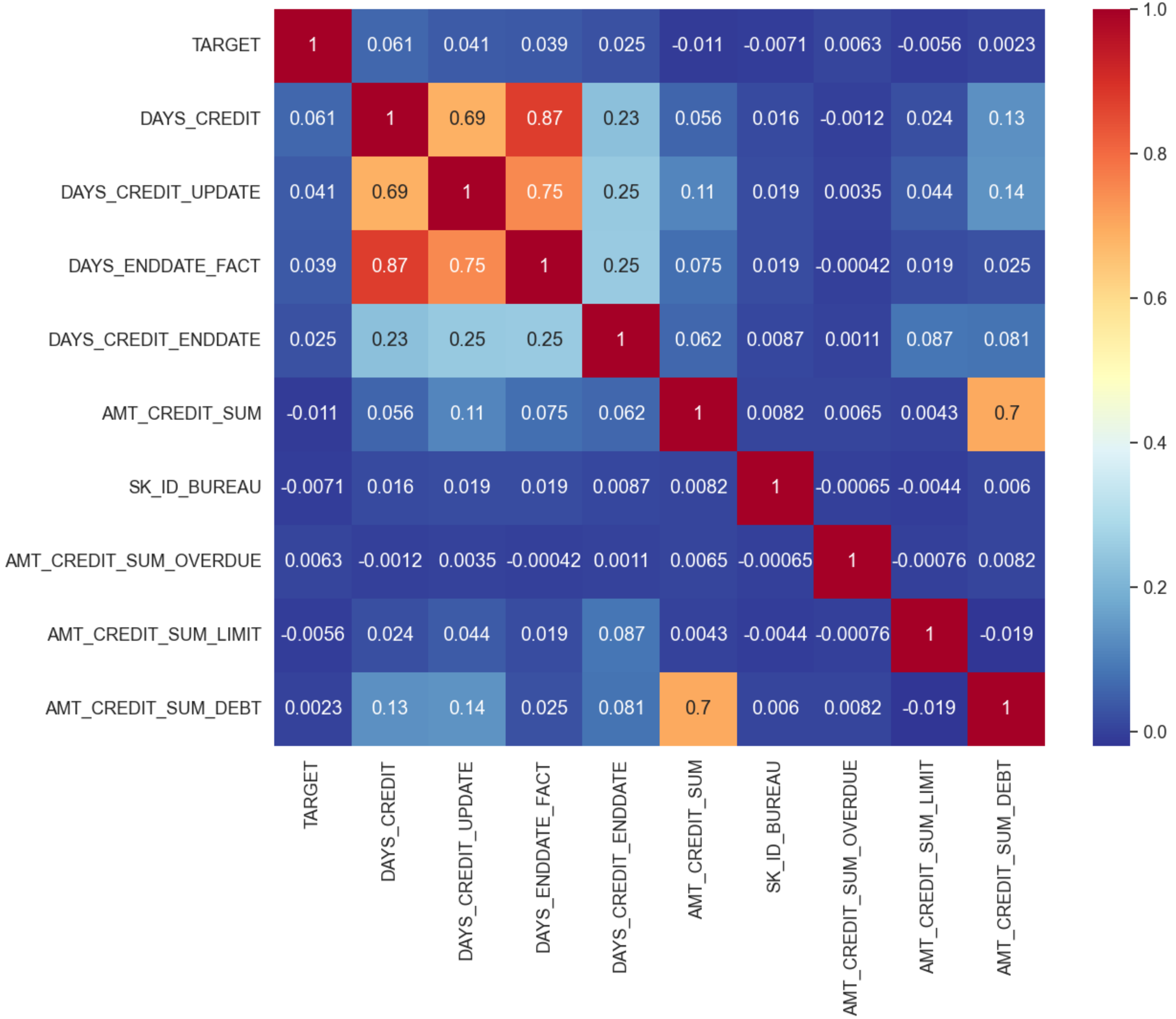






2.2

Bureau



Top Correlation of Bureau

```
bureau_merged['DAYS_CREDIT_ENDDATE'].describe()
```

```
count    1.102115e+06
mean     4.909742e+02
std      4.955661e+03
min     -4.206000e+04
25%     -1.142000e+03
50%     -3.330000e+02
75%      4.730000e+02
max      3.119800e+04
Name: DAYS_CREDIT_ENDDATE, dtype: float64
```

DAYS_CREDIT_ENDDATE

```
bureau_merged['DAYS_ENDDATE_FACT'].describe()
```

```
count    737359.000000
mean      -1022.087408
std         718.422310
min      -42023.000000
25%       -1502.000000
50%        -898.000000
75%        -427.000000
max           0.000000
Name: DAYS_ENDDATE_FACT, dtype: float64
```

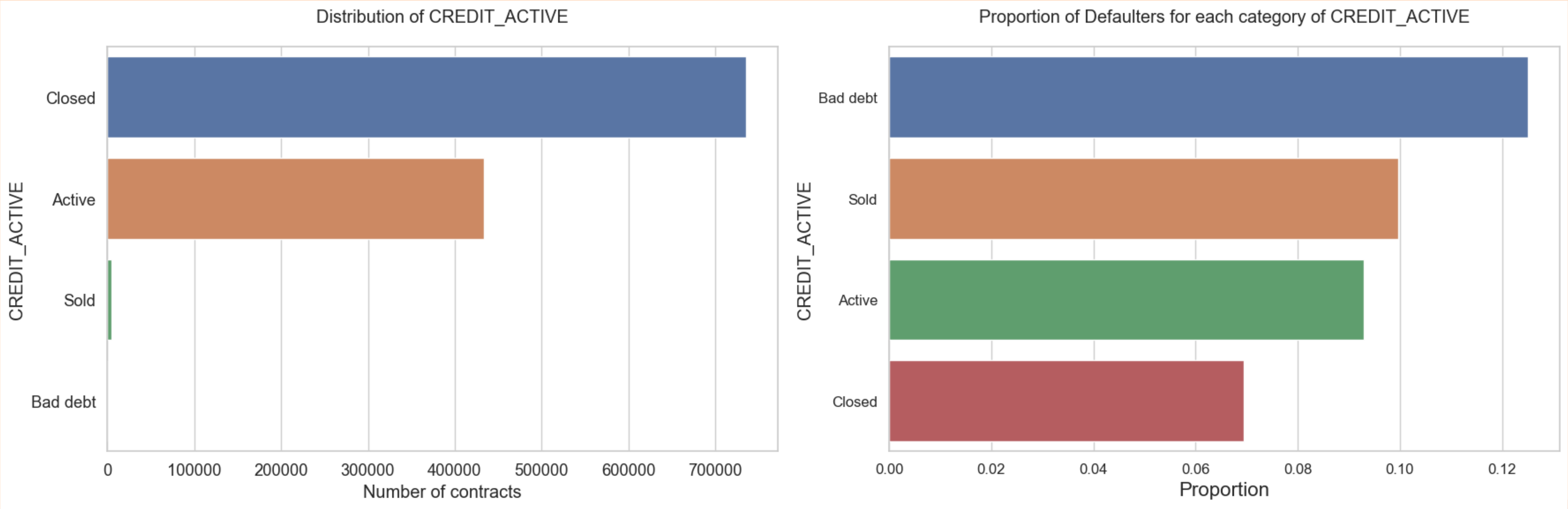
DAYS_ENDDATE_FACT

```
bureau_merged['DAYS_CREDIT_UPDATE'].describe()
```

```
count    1.173378e+06
mean    -5.993146e+02
std      7.317144e+02
min     -4.194700e+04
25%     -9.030000e+02
50%     -4.060000e+02
75%     -3.300000e+01
max      3.720000e+02
Name: DAYS_CREDIT_UPDATE, dtype: float64
```

DAYS_CREDIT_UPDATE

CREDIT_ACTIVE

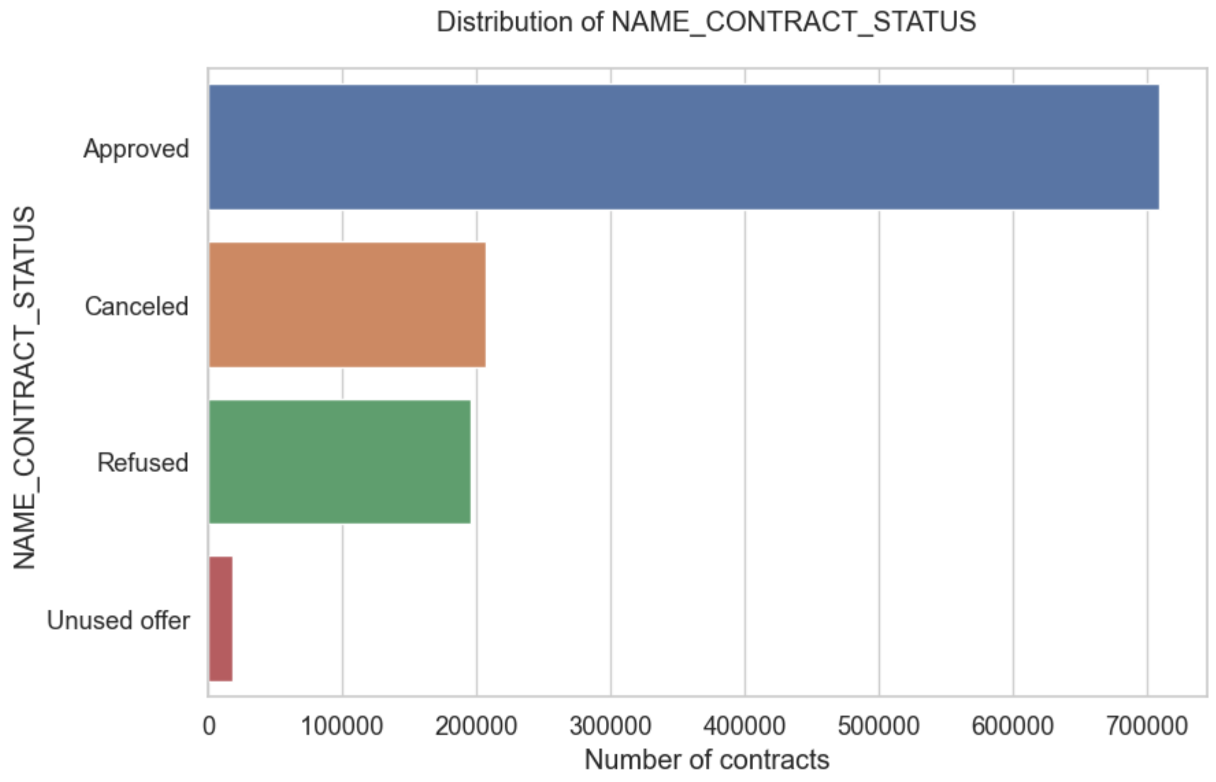


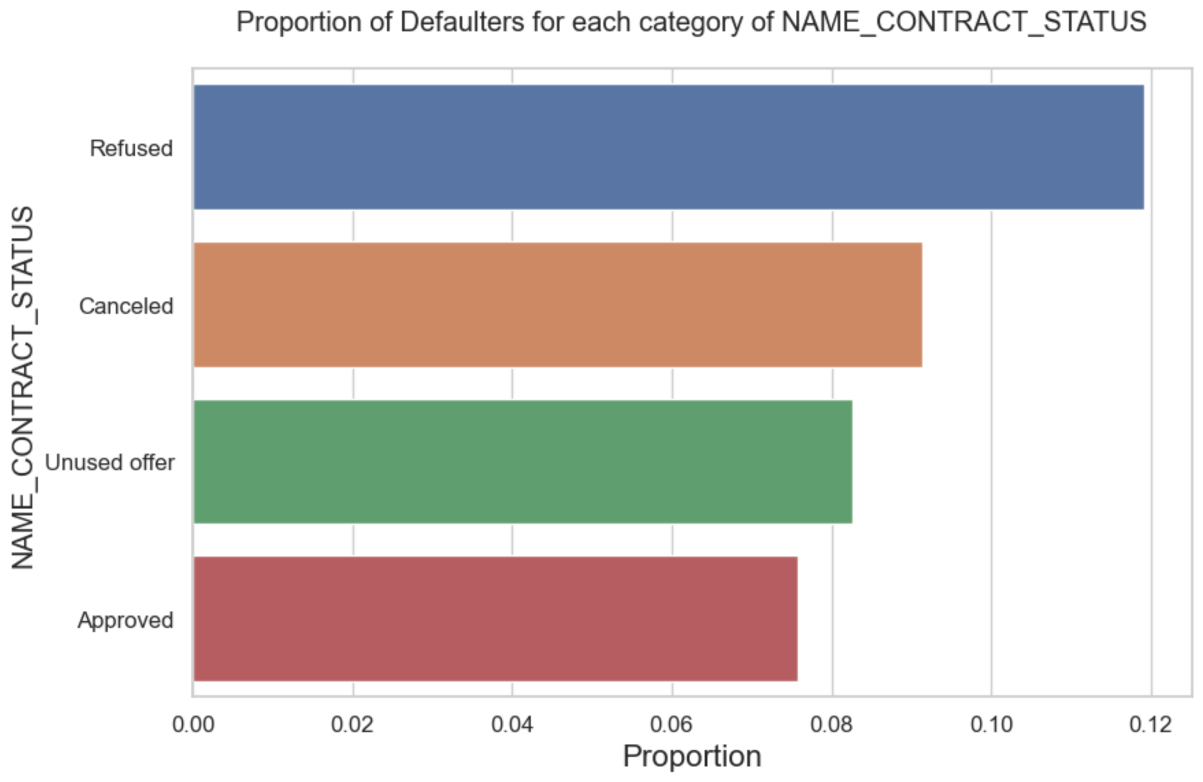
The majority of loans are Closed, followed by Active loans. Feature Engineering we can only focus on two states, Closed and Active

2.3

Previous_application

NAME_CONTRACT_STATUS

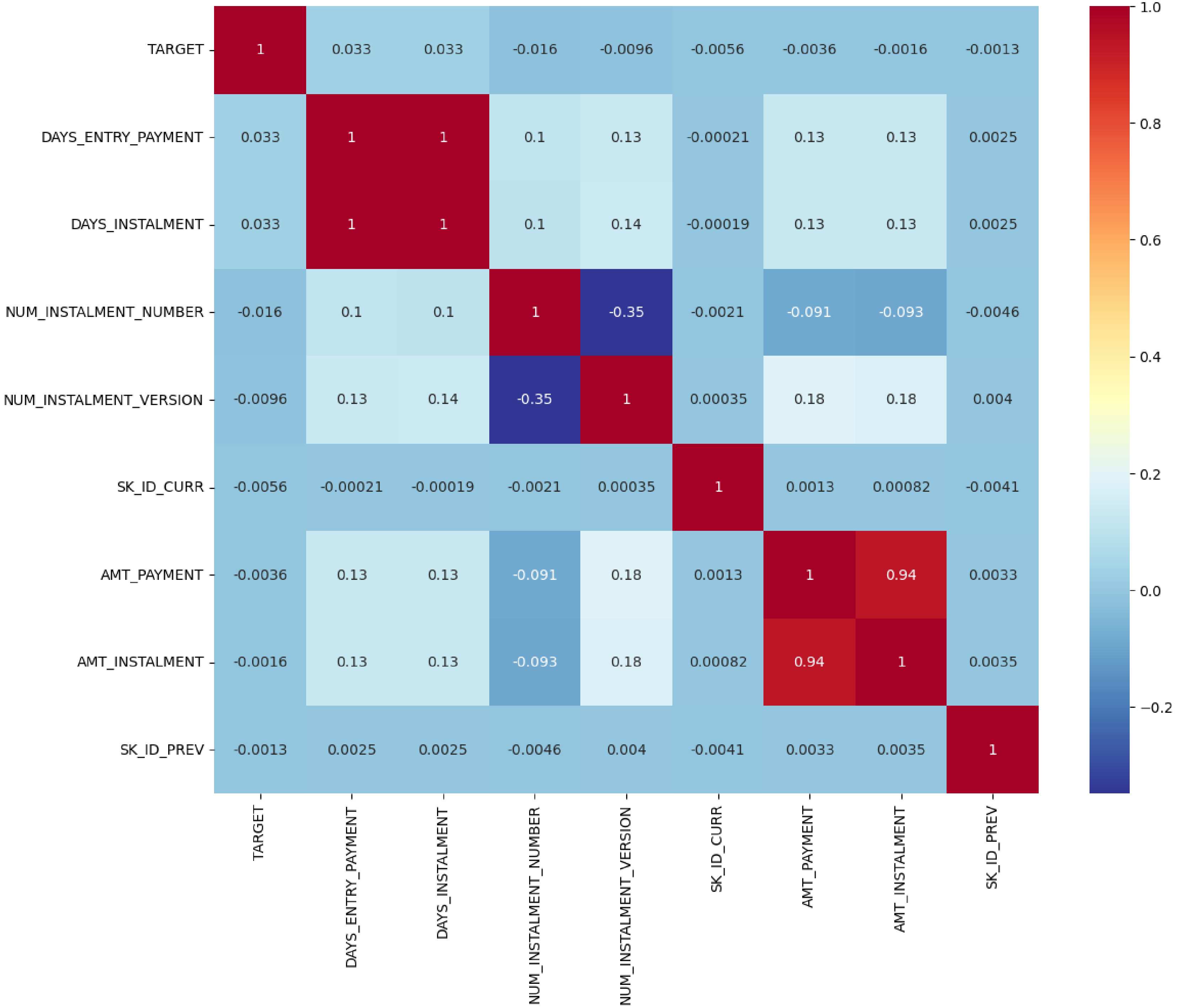




The status of the most previous loans is 'Approved', accounting for more than 65% of the total number of loans, while the highest rate of defaulter is 'Refused'. This makes sense as people with previous unsuccessful loans tend to have defaulter

2.4

Installments Payments



Top Correlation of Installments Payments

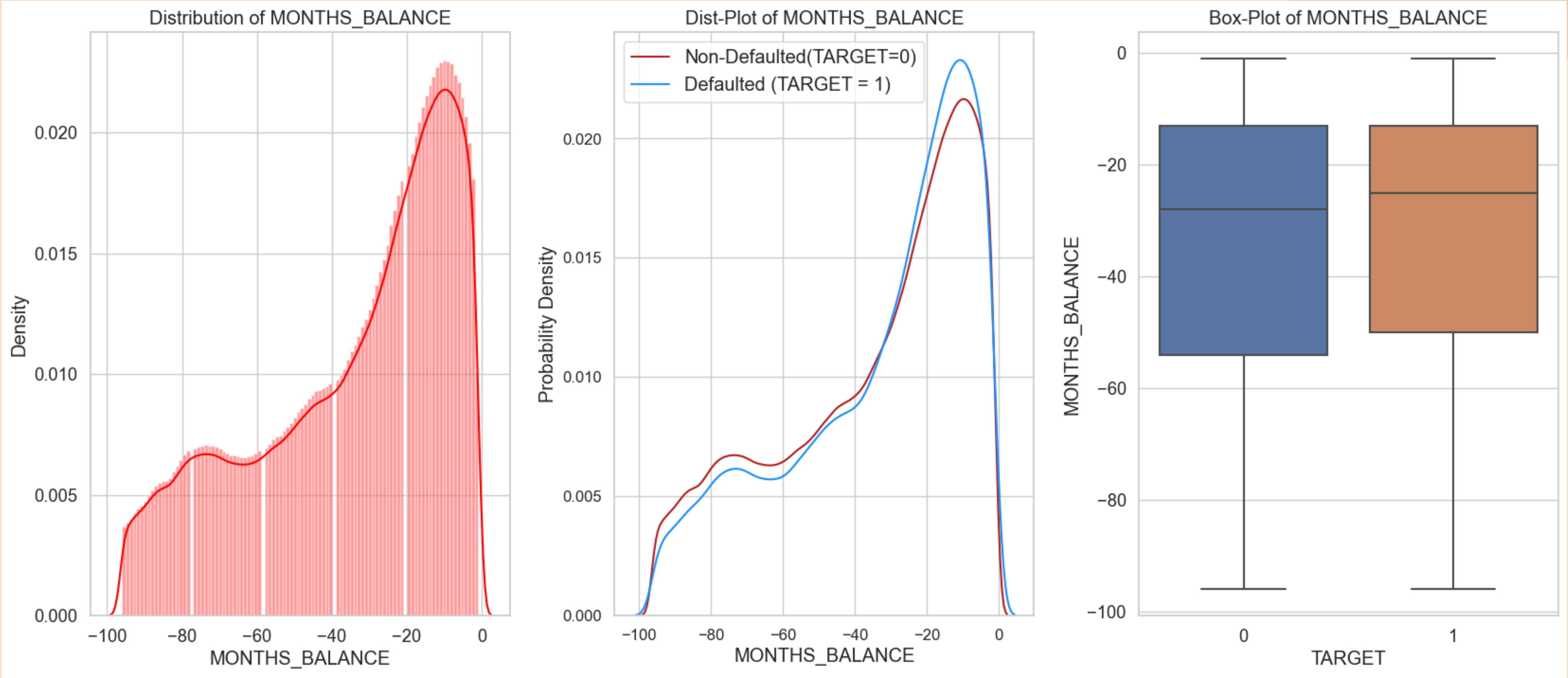
A few pairs of features with high correlation

- DAYS_INSTALMENT and DAYS_ENTRY_PAYMENT
- AMT_INSTALMENT and AMT_PAYMENT

It is understandable that these two pairs of characteristics are highly correlated, because these are expected and actual information variables.

2.5

Bureau Balance, POS CAS Balance, Credit Card Balance



MONTHS_BALANCE

- MONTHS_BALANCE is a time-series variable
- When FE we can sort value according to this variable and perform some FE techniques to create some features such as using Exponential Weighted Moving Average(EMA) to weight the records
- The closer the recording is to the present time, the more important it is and should have higher weight.

03

FEATURE ENGINEERING AND SELECTION

3.1

CLEANING DATA

Handle missing value, outlier, annomolies

```python
# Missing value
def remove_missing_col(df, threshold = 0.6):
    # Calculate the percentage of missing values for each column
    missing_percentage = df.isnull().mean()

    # Identify columns where the missing percentage is greater than the threshold
    columns_to_drop = missing_percentage[missing_percentage > threshold].index

    # Drop the identified columns from the DataFrame
    df = df.drop(columns=columns_to_drop)

    return df

def fill_nan(df):
    numeric_columns = df.select_dtypes(include=['number']).columns.tolist()
    df[numeric_columns] = df[numeric_columns].fillna(df[numeric_columns].mean())

    categorical_columns = df.dtypes[df.dtypes == 'object'].index.tolist()
    df[categorical_columns] = df[categorical_columns].fillna(df[categorical_columns].mode())
    return df
```

```python
## Xử lý Outliers
def get_thresh(col, df):
    xs = df[col]
    mu = xs.mean()
    sigma = xs.std()
    low =  mu - 3*sigma
    high = mu + 3*sigma
    return low, high

def change_value(x, low, high):
    if x < low:
        return low
    elif x > high:
        return high
    else:
        return x

def replace_outlier(df):
    num_columns = df.select_dtypes(include=['int64', 'float64'])
    for col in num_columns.columns:
        if col == 'TARGET':
            pass
        else:
            low, high = get_thresh(col, df)
            df[col] = df[col].apply(lambda x: change_value(x, low, high))
    return df
```

```python
def check_imbalance(series):
    value_counts = series.value_counts(normalize=True)
    return value_counts.max() > 0.98

def drop_column_unique_value(df):
    filtered_columns = [col for col in df.columns
                        if df[col].nunique() > 2 or not check_imbalance(df[col])]
    # Create a new DataFrame with selected columns
    df_filtered = df[filtered_columns]
    return df_filtered
```

Encoding categorical feature

Using both LabelEncoder and OneHotEncoder

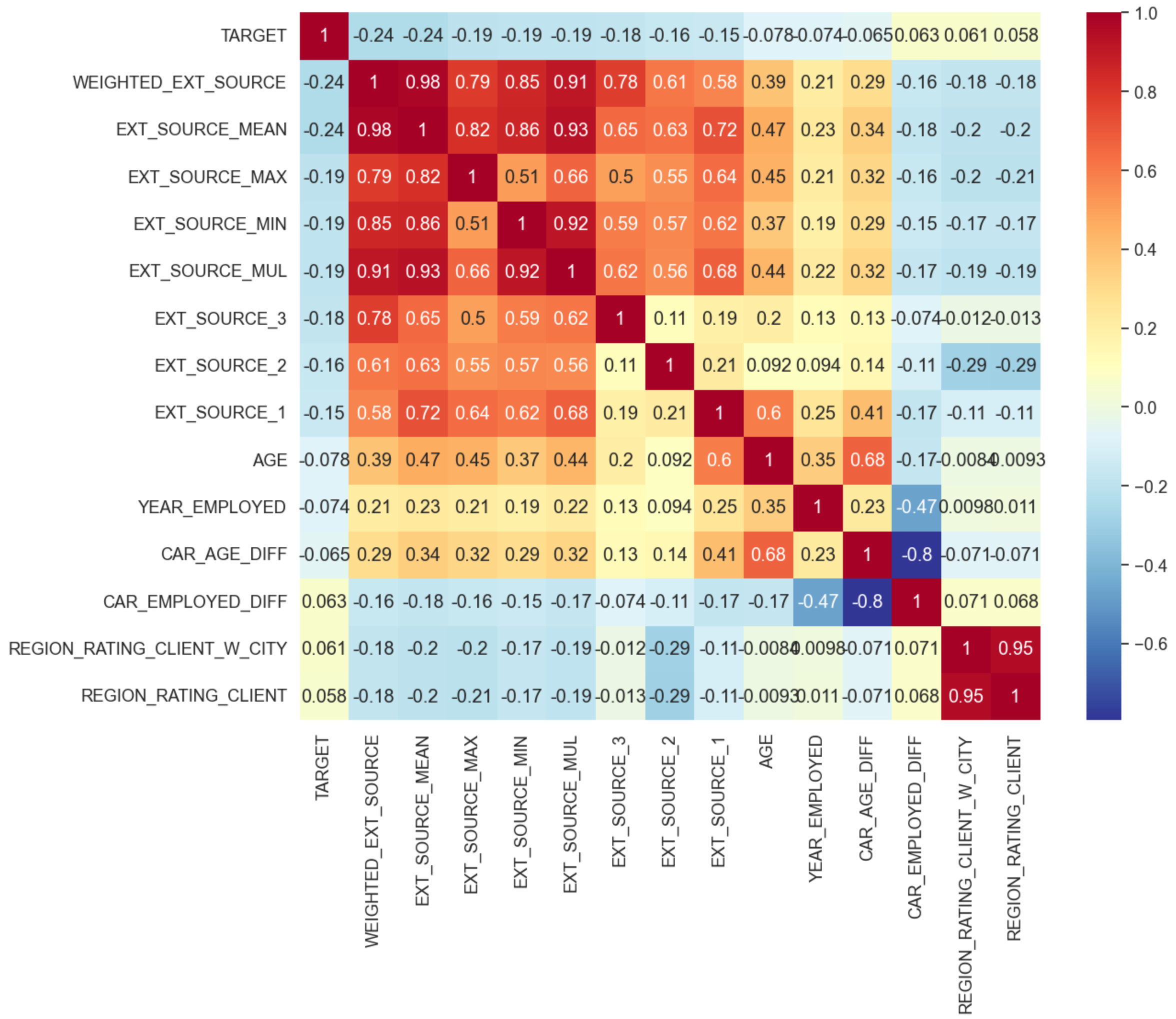
```python
from sklearn.preprocessing import LabelEncoder

def encode(df):
    label = LabelEncoder()
    categorical_cols = df.select_dtypes(include=['object']).columns

    for col in categorical_cols:
        nunique_cols = df[col].nunique()
        if nunique_cols == 2:
            df[col] = label.fit_transform(df[col])

    df = pd.get_dummies(df)
    return df
```

3.2

CREATE FEATURE

Perform mathematical operations

```python
#ext_sources
data['EXT_SOURCE_MEAN'] = (data['EXT_SOURCE_1'] + data['EXT_SOURCE_2'] + data['EXT_SOURCE_3'] ) / 3

data['EXT_SOURCE_MUL'] = data['EXT_SOURCE_1'] * data['EXT_SOURCE_2'] * data['EXT_SOURCE_3']

data['EXT_SOURCE_MAX'] = [max(ele1,ele2,ele3) for ele1, ele2, ele3
                          in zip(data['EXT_SOURCE_1'], data['EXT_SOURCE_2'], data['EXT_SOURCE_3'])]

data['EXT_SOURCE_MIN'] = [min(ele1,ele2,ele3) for ele1, ele2, ele3
                          in zip(data['EXT_SOURCE_1'], data['EXT_SOURCE_2'], data['EXT_SOURCE_3'])]

data['EXT_SOURCE_VAR'] = [np.var([ele1,ele2,ele3]) for ele1, ele2, ele3
                          in zip(data['EXT_SOURCE_1'], data['EXT_SOURCE_2'], data['EXT_SOURCE_3'])]

data['WEIGHTED_EXT_SOURCE'] =  data['EXT_SOURCE_1'] * 2 + data['EXT_SOURCE_2'] * 3 + data['EXT_SOURCE_3'] * 4
```



CREATE NEW FEATURE

INCOME & CREDIT FEATURES	
1. CREDIT_INCOME_RATIO	Số tiền vay chiếm bao nhiêu phần trăm tổng thu nhập của khách hàng.
2. CREDIT_ANNUITY_RATIO	Số tiền vay gấp bao nhiêu lần số tiền trả góp hàng tháng.
3. ANNUITY_INCOME_RATIO	Số tiền trả góp hàng tháng chiếm bao nhiêu phần trăm tổng thu nhập của khách hàng.
4. INCOME_ANNUITY_DIFF	Số tiền còn lại sau khi khách hàng trả góp hàng tháng.
5. CREDIT_GOODS_RATIO	Số tiền vay gấp bao nhiêu lần giá trị hàng hóa.
6. CREDIT_GOODS_DIFF	Số tiền vay còn lại sau khi mua hàng.
7. GOODS_INCOME_RATIO	Giá trị hàng hóa chiếm bao nhiêu phần trăm tổng thu nhập của khách hàng.
8. PAYMENT_RATE	Tỷ lệ phần trăm số tiền vay được khách hàng trả hàng tháng.
9. INCOME_CREDIT_PERC	Tổng thu nhập gấp bao nhiêu lần số tiền vay.
10. INCOME_TO_EMPLOYED_RATIO	Mức thu nhập trung bình hàng năm của khách hàng.
AGE RATIOS & DIFFS	
11. AGE_EMPLOYED_DIFF	Chênh lệch giữa tuổi thực của khách hàng và tuổi bắt đầu đi làm.
12. EMPLOYED_TO_AGE_RATIO	Tỷ lệ thời gian khách hàng đã đi làm so với tuổi của họ.
13. INCOME_TO_BIRTH_RATIO	Mức thu nhập trung bình hàng năm của khách hàng.
14. ID_TO_BIRTH_RATIO	Giá trị này cho biết độ vững chắc về tài chính và tín dụng của khách hàng.

```python
#income and credit features
data['CREDIT_INCOME_RATIO'] = data['AMT_CREDIT'] / (data['AMT_INCOME_TOTAL'] + 0.00001)
data['CREDIT_ANNUITY_RATIO'] = data['AMT_CREDIT'] / (data['AMT_ANNUITY'] + 0.00001)
data['ANNUITY_INCOME_RATIO'] = data['AMT_ANNUITY'] / (data['AMT_INCOME_TOTAL'] + 0.00001)
data['INCOME_ANNUITY_DIFF'] = data['AMT_INCOME_TOTAL'] - data['AMT_ANNUITY']
data['CREDIT_GOODS_RATIO'] = data['AMT_CREDIT'] / (data['AMT_GOODS_PRICE'] + 0.00001)
data['CREDIT_GOODS_DIFF'] = data['AMT_CREDIT'] - data['AMT_GOODS_PRICE'] + 0.00001
data['GOODS_INCOME_RATIO'] = data['AMT_GOODS_PRICE'] / (data['AMT_INCOME_TOTAL'] + 0.00001)
data['PAYMENT_RATE'] = data['AMT_ANNUITY'] / (data['AMT_CREDIT'] + 0.00001)
data['INCOME_CREDIT_PERC'] =  data['AMT_INCOME_TOTAL'] / (data['AMT_CREDIT'] + 0.00001)
data['INCOME_TO_EMPLOYED_RATIO'] = data['AMT_INCOME_TOTAL'] / (data['YEARS_EMPLOYED'] + 0.00001)


#age ratios and diffs
data['AGE_EMPLOYED_DIFF'] = data['AGE'] - data['YEARS_EMPLOYED']
data['EMPLOYED_TO_AGE_RATIO'] = data['YEARS_EMPLOYED'] / (data['AGE'] + 0.00001)
data['INCOME_TO_BIRTH_RATIO'] = data['AMT_INCOME_TOTAL'] / (data['AGE'] + 0.00001)
data['ID_TO_BIRTH_RATIO'] = data['YEARS_ID_PUBLISH'] / (data['AGE'] + 0.00001)
```

Exponential Weighted Moving Average (EMA)

```
bureau_balance['EXP_WEIGHTED_STATUS'] =
    bureau_balance.groupby('SK_ID_BUREAU')['WEIGHTED_STATUS'].transform(lambda x:x.ewm(alpha = 0.8).mean())
bureau_balance['EXP_ENCODED_STATUS'] =
    bureau_balance.groupby('SK_ID_BUREAU')['STATUS'].transform(lambda x: x.ewm(alpha = 0.8).mean())
```

WOE - Binning Process

```
# binning process
target_var = application_train['TARGET']
application_train.drop('TARGET', axis = 1, inplace = True)
cate_cols = application_train.select_dtypes(include='object').columns.tolist()

columns = application_train.columns.tolist()
binning_process = BinningProcess(columns, categorical_variables=cate_cols, max_n_prebins=30)

binning_process.fit(application_train, target_var)
```

GROUPBY, AGGREGATE

```python
aggregations_basic = {
    'MONTHS_BALANCE' : ['mean','max'],
    'STATUS' : ['mean','max','first'],
    'WEIGHTED_STATUS' : ['mean','sum','first'],
    'EXP_ENCODED_STATUS' : ['last'],
    'EXP_WEIGHTED_STATUS' : ['last']}

#aggregating over whole dataset first
aggregated_bureau_balance = bureau_balance.groupby(['SK_ID_BUREAU']).agg(aggregations_basic)
aggregated_bureau_balance.columns = ['_'.join(ele).upper() for ele in aggregated_bureau_balance.columns]
```

```python
aggregations_for_year = {
    'STATUS' : ['mean','max','last','first'],
    'WEIGHTED_STATUS' : ['mean','max', 'first','last'],
    'EXP_WEIGHTED_STATUS' : ['last'],
    'EXP_ENCODED_STATUS' : ['last']}

#aggregating some of the features separately for latest 2 years
aggregated_bureau_years = pd.DataFrame()
for year in range(2):
    year_group = bureau_balance[bureau_balance['MONTHS_BALANCE'] == year].groupby('SK_ID_BUREAU').agg(aggregations_for_year)
    year_group.columns = ['_'.join(ele).upper() + '_YEAR_' + str(year) for ele in year_group.columns]

    if year == 0:
        aggregated_bureau_years = year_group
    else:
        aggregated_bureau_years = aggregated_bureau_years.merge(year_group, on = 'SK_ID_BUREAU', how = 'outer')
```

3.3

FEATURE SELECTION

StandardScaler()

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(C = 0.1)
train, test = train_test_split(application_train,test_size=.25,random_state = 123)

#separating dependent and independent variables
train_x1 = train[[i for i in train.columns if i not in ['SK_ID_CURR'] + [ 'TARGET']]]
train_y1 = train[["TARGET"]]

test_x1 = test[[i for i in test.columns if i not in ['SK_ID_CURR'] + [ 'TARGET']]]
test_y1 = test[["TARGET"]]

scaler = StandardScaler()
train_x1 = scaler.fit_transform(train_x1)
test_x1 = scaler.fit_transform(test_x1)
```

KBestSelection

```
#Using SelectKBest để chọn ra bộ feature tốt nhất
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

k = int(0.855 * train_X.shape[1])
k_best = SelectKBest(score_func=f_classif, k=k)
```

Thanks for Listening!

Do you have any question?